教程一: 什么是逆向工程

一、什么是逆向工程?

逆向工程是通过编译的二进制文件,尝试重建(或简单理解) 程序原始的工作方法。程序员最初在写程序时,一般使用像 C++、 VB、God forbid、Delphi 等高级语言。因为计算机本身不能够 理解这些语言,所以程序员所写的代码需要被组装成特定的更机 器化的格式,也就是计算机所能理解的格式。这个足够原始的代 码被叫做机器语言。对人类而言这些代码不太友好,经常需要耗 费大量的脑力才能准确的明白程序员的思想。

二、逆向工程是干什么的?

逆向工程能够被用于计算机科学的很多领域,不过这里有几个通用分类:

- 它使得与历史遗留代码(就是已经没有了源代码)进行交 互成为可能
- 打破拷贝保护(即打动你的朋友和省钱)
- ●研究病毒和恶意软件
- 评估软件质量和稳健性
- 向软件中添加功能

第一个分类就是当源代码不可用时,通过逆向工程编码与已 存在的二进制程序进行交互。关于这个我不会讨论太多,因为它 太枯燥了。 分类二(也是最大的)是打破拷贝保护。就是禁用限时试用 限制,干掉注册,以及免费获得商业软件的其他所有功能。这方 面我们会进行大量的讨论。

分类三是学习病毒和恶意软件代码。之所以需要逆向工程, 是因为没有几个病毒编写者会向外说出他是如何编写的代码,应 该具有什么功能,以及怎样完成这些功能(除非他们真的很愚 蠢)。这真是一个让人兴奋的领域,不过这也需要大量的知识。 现在我们不会讨论太多,具体的都在后面章节。

分类四是评估软件安全和漏洞。当创建大型应用(想想 Windows 操作系统),逆向工程被用来确保系统不会包含任何主 要的漏洞、安全缺陷。坦率的说,是让破解者破解软件时尽可能 的困难。

最后一个分类是向现有软件中添加功能。就我个人来说,我 认为这是最有趣的地方之一。不喜欢你的网站设计软件中的图 片?换掉它们。想在你最喜欢的字处理软件中添加一个加密文档 的菜单项?那就加上。想要在 windows 计算器中添加一个损人的 消息框去无止尽的作弄你的同事?那就干他一票。在后面的系列 中我们将进入这个世界。

三、需要什么知识?

与你猜测的一样,成为一名合格的逆向工程师需要大量的知识。幸运的是,在开始逆向工程时大量的知识都不是必须的。这 正是我想要开始的地方。也就是说,享受逆向的乐趣以及从本教 程中收获一些东西。然后你应该至少对一个程序是如何工作的有 一个基本的理解(比如,你应该知道一个基本的 if..... then 语句是什么样,数组是什么样,至少理解一个基本的 hello world 程序)。第二,强烈建议熟悉汇编语言。即使没有汇编基础你也 可以通过本教程,不过在有些地方你就会希望自己是 ASM 的大 牛,以便真正理解你正在做什么。另外,你需要大量的时间来学 习怎样使用工具。这些工具对于逆向工程来说是无价的,也需要 学习这些工具的快捷键、缺陷和特性。最后,逆向工程需要大量 的实践。与不同的壳/保护/加密设计玩耍。学习编写程序的原始 语言(甚至 Delphi)、破解反逆向工程的技巧等等。本教程的最 后,我加上了"进一步阅读"部分,有一些建议。如果你真的想 要学好逆向,强烈建议你进一步阅读其他内容。

四、使用哪种工具?

在逆向领域中有很多种不同的工具可用。在逆向二进制时有 许多特别的保护需要被解决。有一些可以让逆向者的生活更轻 松。有一些我认为是"订书针"项目,就是经常用到的那些。对 于大部分工具来说,是符合几种分类的:

1、反汇编器

反汇编器尝试将二进制形式的机器语言代码以一种友好的 形式显示出来。它们也进行数据推断比如函数调用、传递变量和 文本字符串。这就让可执行文件看起来更像人类可读的代码,而 不是一串数字串起来的样子。反汇编器非常多,它们中的一些专 门做一些特定的工作(比如 Delphi 中的写二进制)。主要是你找 一 个 你 觉 得 最 舒 服 的 。 我 喜 欢 用 IDA (http://www.hex-rays.com/上有免费版本可用),以及一些不 太知名的在特定情况下比较有用的工具。

2、调试器

调试器是逆向工程师的面包和黄油。它们首先分析二进制文件,这一点特别像反汇编器。然后调试器允许逆向者单步执行代码,一次运行一行并且查看结果。这对于发现一个程序是如何工作来说是无价的。最后,一些调试器允许对改变的代码做说明,然后带着那些变化再次允许。示例调试器是 Windbg 和 011ydbg。 我几乎只用 011ydbg (http://www.o11ydbg.de/),除非调试内 核模式的二进制文件,不过我们不久就会接触到的。

3、十六进制编辑器

十六进制编辑器可以让你查看二进制文件的指定字节,并且 可以更改它们。也提供了搜索指定字节,保存部分二进制数据到 磁盘等等。有许多免费的十六进制编辑器,并且大部分都挺好用。 本教程的大部分都不会用到它们,不过有时候它们是无价的。

4、PE 和资源 查看器/编辑器

每一个被设计在 windows 上运行的二进制文件 (linux 也是 一样),在它的起始地方都有一个数据区用于告知操作系统如何 设置和初始化程序。它告诉 OS 它需要多少内存、它需要借用哪 些 DLL 的代码、对话框的相关信息等等。它叫做可移植的执行体 (Portable Executable),所有被设计用来在 windows 上运行的 程序都需要有一个。

在逆向工程的世界里,这个结构的字节就变得非常重要,因 为它给逆向者需要的关于二进制文件的信息。最终,你想要(或 需要)改变这个信息,要么让程序做一些和它初衷不一样的事情, 要么让程序回到它以前的样子(像保护器让代码变得很难理解之 前的样子)。有非常多的 PE 查看器和编辑器工具。我用 CFF Explorer (http://www.ntcore.com/exsuite.php) 和 LordPE (http://www.woodmann.com/collaborative/tools/index.php/ LordPE),不过你可以随意使用你觉得舒服的工具。

大多数的文件也有资源区。包括图像、对话框、菜单、图标 和文本字符串。有时候你只看(和修改)资源区就觉得很有意思。 此教程的最后我会给你展示一个例子。

5、系统监视工具

在逆向程序时,有时候观察应用程序对系统做出的改变也很 重要(在研究病毒和恶意软件时尤其重要),是不是有注册表项 被创建或查询?是不是有.ini 文件被创建?是不是有进程被创 建,或许用来阻挠软件被逆向? 系统监视工具 有 <u>procmon</u>、 <u>regshot</u> 和 process hacker。后面我们会讨 论这些。

6、其他工具和信息

在学习过程中有其他工具我们将用到,比如脚步、脱壳工具、

壳识别器等。一些Windows API 参考工具也归于此类。这个 API 是巨大的,有时也很复杂。在逆向工程中知道被调用的 API 在干 什么是非常有用的。

7、啤酒

五、那么,我们开始吧!

即使我们在拥有很少知识的情况下开始,在第一课中我想让你尝尝逆向的滋味。此教程中包含有一个资源查看/编辑器,叫做 <u>XN Resource Editor</u>。

它是免费的。基本上,这个程序可以让你查看 exe 文件的资源区,当然也可以修改这些资源。我已经感觉到你对"这些"的 巨大兴趣,你可以修改程序中的菜单、图标、图片、对话框,你 可以给它命名。下面我们来改一个试试:

首先,运行 XN。点击顶部的载入图标,找到 Windows\System32\并且载入 calc.exe(你的 windows 默认的位 置可能不同)。你会看到一堆可用的文件夹:



可以看到有 Bitmaps 文件夹(程序显示的任何图片), Menu (顶级菜单项), Dialog(对话框,相关文本和按钮), String Table,, IconGroup 等。你可以对它们为所欲为了。确保另存为 一个不同的文件(你肯定不喜欢因为一个 XX 计算器就重装 windows)。细节如下:

点击那个靠近 Menu 的加号。你会看到以数字命名的文件夹。 它是程序中资源的 ID, 以便 windows 用来访问相关资源。同样 打开该文件夹。你可以看到一个"English (United States)" 图标或者类似的东西。如果你点击它,你会看到一个有关菜单外 观的图表 (你能点击旁边一个类似真实菜单的菜单)。

Calc.exe - XN Resource Editor			
Elle Edit View Resource Menu He Company - In Har Har Menu Elle Company - Info Menu Elle Company - Info Menu Elle Edit View Resource Menu Her Hereitari Elle Edit View Resource Menu Hereitari Elle E	Caption Shortcut	&Nerd (None)	Edit View Help
	Enabled Checked	True False	Scientific Digit grouping

现在,点击菜单项"Scientific"。那个 Caption 字段应该 改成"&Scientific"。那个符号是告诉你"热键"是什么,这 里是大写的"S"。假如我想用"e"代替作为热键,应该是这样 的"Sci&entific"。好了,和内建的计算器热键不一样吧? 仅 仅修改了它们!!不过,我们做一些其他的吧。在 Captial 字段, 将 &Scientific 替换成 "&Nerd"。这将会把菜单项修改成 "Nerd"并且使用热键"N"(我们看看菜单中其他选项以确保 没有其他的菜单项使用"N"作为热键)。你应该对所有的菜单项 做这个工作。现在,到上面的 File 菜单(在 XN Resource 中) 并且选择"Save As..."。用不同的名字保存你的新版计算器 (最好保存到不同的路径),然后运行它。



当然,你不需要止步于此。为了开动我同事的榆木脑袋,我 修改了他的计算器的所有数字。

Calc	ulator						
Edit Vie	Edit View Help						
0.							
\Box	Backs	Dace	CE		С		
MC	3	8	4	1	sqrt		
MR	1	9	5	-	2		
MS	2	6	0	<u> </u>	1/x		
M+	7	+/-	•	+	=		

如你所见,限制你的只有天空。

延伸阅读(译者注:这里都是英文书,可能部分有中文翻译, 未经查证):

1、汇编语言。<u>Assembly Language For Intel Based</u> <u>Computers</u>中是一本关于汇编的书。你也可以查看一些网站,提 供了大量的<u>下载</u>,<u>说明</u>,<u>示例代码</u>和<u>帮助</u>。另一个好的资 源是"The Art of Assembly"。我将会在今后的某个章节中包 含进来,不过你也可以从<u>这里</u>下载。

2、PE 文件结构。最好的资源是微软自己的"<u>An in-depth</u> <u>look into the Win32 Portable Executable File Format</u>"。 另一个好的文档(有很多漂亮的图片)是"<u>PE File Structure</u>"。 它是一个可下载的 PDF 文件。

3、Windows 操作系统内核。有 Mark Russinovich 的书

"<u>Microsoft Windows Internals</u>"。它想女人的棒球(baseball) 一样让人兴奋,不过它是 THE 资源。

4、破解教程。www.Tuts4You.com 就是这样的一个地方。

教程二、介绍 011y Debug

一、什么是 011y Debugger?

援引作者 Oleh Yuschuk 的话 "OllyDbg 是一个用于微软 Windows 的 32 位汇编级分析调试器"。在没有源代码的情况下, 二进制代码分析非常有用。Olly 也是一个动态调试器,意味着 它允许用户在程序运行时修改一些东西。这在实际分析二进制文 件尝试找出程序工作原理时非常的重要。Olly 有许多许多很棒 的特性,这就是为什么它是逆向工程领域的天字第一号调试器 (至少在 Ring3 级是,我们马上就接触到了)。

二、概览

下面是011y的主界面图片,上面有一些说明性的标签。



打开 011y 时有一个默认的子窗口是 CPU 窗口。这是那个"大 图片"中大部分数据所在的地方,如果你什么时候把它关掉了, 只需要点击工具栏中那个"C"图标就行了。窗口被分成了四个 部分:反汇编区 (Disassembly),寄存器区 (Registers),堆栈 区 (Stack) 以及内存数据区 (Dump)。下面是对每个区的说明:

1、反汇编区

该部分主要包含了二进制文件的反汇编代码。这是011y显示二进制信息的地方,包括操作码(opcode)和翻译的汇编代码。

第一列是指令的地址(内存中地址)。第二列按汇编语言叫操作码,每个指令至少对应一条代码(有很多对应多条)。这才是 CPU 真正需要并且是唯一能读懂的代码。这些操作码组成了"机器语 言",也就是计算机的语言。如果你看过二进制的原始数据(用 十六进制编辑器),你除了看到这些操作码的字符串以外,就没 有其他的了。011y的一个主要工作是将这些"机器语言""反汇 编"成人类可读的汇编语言。第三列是汇编语言。不过退一步讲, 对于不太懂汇编的人来说,汇编不比操作码好多少。不多随着学 的越来越多,汇编提供了远多于代码所做的更多信息。

最后一列是 011y 对于该行代码的注释。有时候会包含所调 用 API 的名字,比如 CreateWindow 和 GetDlgItemX。011y 也 会尝试通过将非 API 调用命名来帮助我们理解代码,上图中的 "ImageRed.00510C84" 和"ImageRed.00510BF4"就是此类情 况。退一步讲,这些东西不是那么有用,011y 也允许我们将它 们修改成一个有意义的名字。你也可以在该列写自己的注释。只 要双击该列中的某行,就会弹出一个对话框让你输入注释。这些 注释会自动保存到下一次。

2、寄存器区

每个 CPU 都有一组寄存器。用来临时存放数值,和高级语言中的变量很像。下面是寄存器窗口的特写(有标记):



顶部实际上是 CPU 的寄存器。如果值有变化,寄存器会从黑 色变为红色(对于观察数值的变化真的非常有用)。你也可以双 击任何一个寄存器来改变它的内容。这些寄存器能做很多事情, 后面会讨论更多。

中间那块是标志寄存器,是 CPU 用来标记代码中一些事情的 发生(两个数相等、一个数比另外一个大等等)。双击其中一个 标志寄存器就可以修改它。这些玩意儿在我们的学习过程中扮演 着重要的角色。

底下的部分是 FPU,或者叫浮点运算器。只要 CPU 执行任何 涉及小数点的运算就会用到它们。逆向者很少用到它们,主要是 在我们接触加密的时候用。

3、堆栈区

0012FFC4 0012FFC8 0012FFD0 0012FFD0 0012FFD4 0012FFD8 0012FFD6 0012FFE4 0012FFE4 0012FFE8 0012FFE8 0012FFE8 0012FFF8 0012FFF8 0012FFF8	70317077 70910228 FFFFFFF 7FFFFFF 895486ED 0012FFC8 891DC820 FFFFFFF 70839AD8 70817080 00000000 00000000 00000000 00000000	RETURN to Kernel32.70817077 ntdll.70910228 End of SEN chain SE handler kernel32.70817080 ImageRed.(ModuleEntryPoint)

堆栈是内存中的一段区域,用于存储二进制数据的临时列 表。这些数据包括指向内存中地址的指针,字符串,制造者 (makers)及大部分重要的数据,还包括函数调用后的返回地址。 当程序中的一个方法调用另一个方法时,控制权需要转移到新方 法以便于它能够返回。CPU必须知道一个新方法执行完后它是从 哪被调用的,CPU能够返回到它被调用的地方,继续执行该调用 之后的代码。堆栈就是 CPU 保存返回地址的地方。

关于栈你需要知道一点,他是"先进后出"的数据结构。打 个常用的比方,就像是自助餐厅里下面带有弹簧的一摞盘子一 样。当你向顶部"压 (PUSH)"进一个盘子,下面的所有盘子都 会被往下压。当你移除 ("POP")顶部的一个盘子,下面的所有 盘子都会被往上提升一级。下个教程我们会实际看看,所以这里 别担心看不太懂。

图片中,第一列是每一个数据成员的地址,第二列是十六进

制的 32 位数据,如果 011y 能够分析出来的话,那么最后一列是 011y 关于数据项的注释。如果你注意看第一行的话,会看到 "RETURN to kernel…"的注释。这里是 CPU 放在栈上的一个地址,以便于在当前的函数执行完后, CPU 知道返回到哪。

在 011y 中,你可以右键点击堆栈区,并且选择"修改 (modify)"来更改内容。

4、内存数据区

0050C000 00	Address	Address	Hex dump	ASCII
0050C020 32 13 8B C0 02 00 8B C0 00 30 40 00 00 00 00 00 00 00 00 00 00 00 00	00500000	00500000	00 00 00 00 00 00 00 00 02 20 40 00 00 00 00 00	
00500020 32 13 58 00	00500010	00500010		5001610111611161
005300300 40 CC 24 40 00 78 26 40 00 54 28 40 00 00 CB CC C8 \$4.480.T#0.TH 00530050 C9 D7 CF C8 CD CE DB D8 DA D9 CA DC DD DE DF E0 005500600 E1 E3 00 E4 E5 SD 40 00 45 72 72 6F 72 00 SB C0 0050070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20 00500090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 00500090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 00500800 00 00 00 00 00 00 00 00 00 00 00 0	00506020	00506020	32 13 56 10 02 00 56 10 00 50 40 00 00 50 40 00	201-0.11010.
0050C060 E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0 BT.2010.TT 0050C060 E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0 BT.2010.TT 0050C070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20 20 Runtime error 0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0050C090 00 00 00 00 00 00 00 00 00 00 00 00	00500040	00500030	CC 24 40 00 79 26 40 00 54 20 40 00 00 00 00 00 00 00 00	It's a ven THR THE
0050C0600 E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0 βπ.Σσ10 Error T 0050C070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20 Runtime error 0050C090 20 20 61 74 20 30 30 30 30 30 30 30 30 30 30 00 8E C0 at 00000000 T 0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDE 0050C090 00 00 00 00 00 00 00 00 00 00 00 00	00500050	00500050	C9 DZ CE C8 CD CE DB D8 D9 D9 C9 DC DD DE DE E0	
0050C070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20 8 Runtime error 0050C050 20 20 61 74 20 30 30 30 30 30 30 30 30 30 30 00 8E C0 0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0050C090 00 00 00 00 00 00 00 00 00 00 00 00	00500060	00500060	E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0	BT. Zgi@. Error. 14
0050C050 20 20 61 74 20 30 30 30 30 30 30 30 30 30 00 8E C0 at 00000000.T 0050C090 30 31 32 38 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDE 0050C080 00 00 00 00 00 00 00 00 00 00 00 00	0050C070	0050C070	52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20	Runtime error
00500090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDE 00500A0 00 00 00 00 00 00 00 00 00 00 00 00	00500080	00500080	20 20 61 74 20 30 30 30 30 30 30 30 30 30 00 8B C0	at 0000000.14
0050C0A0 00 00 00 00 00 00 00 00 00 00 00 00	00500090	00500090	30 31 32 38 34 35 36 37 38 39 41 42 43 44 45 46	0123456789ABCDEF
0050C020 00 00 00 00 00 00 00 00 00 00 00 00	0050C0A0	0050C0A0	00 00 00 00 00 00 00 00 00 00 00 00 00	
00500000 00 00 00 00 00 00 00 00 00 00 0	0020C0B0	RAPACABA		**************
0050C0E0 1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00 1F 00 XX.X.X.X.X.X.X.	00500000	00500000	00 00 00 00 00 00 00 00 00 00 00 00 00	
005000F0 IE 00 IF 00 IE 00 IF 00 IF 00 ID 00 IF 00 IE 00 A.T.A.T.T.H.T.A	GGEGCGEG	GASACODA		* * * * * * * * * *
AAAAAAIA AB AA SI AA SE AA SI AA SI AA SE AA SI AA SE AAI-1.1-1.1-1.1-1.1-1.1-1.1-1.1-1.1-1.1-1.	0050C0E0	RASACRER	IE AA IF AA IF AA IF AA IF AA ID AA IF AA IF AA	A. V. A. Y. V. H. Y. A.
0050C100 1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00 T.A.T.A.T.A.T	0050C100	0050C100	1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00	T.A. T. T.A. T.A. T.
00500110 00 00 00 00 00 00 00 00 00 00 00 0	0050C110	0050C110	00 00 00 00 00 00 00 00 00 00 00 00 00	
00500120 00 00 00 00 00 00 00 00 00 00 00 00 0	0050C120	0050C120	00 00 00 00 00 00 00 00 00 00 00 00 00	
00500130 78 8F 40 00 00 00 00 00 00 00 00 00 E0 9B 40 00 xA0,	00500130	00500130	78 8F 40 00 00 00 00 00 00 00 00 00 00 E0 9B 40 00	×Α@α¢@.
0050C140 EC 98 40 00 00 00 00 00 00 00 40 00 00 00 00	0050C140	0050C140	<u>EC 98 40 00 00 00 00 80 00 00 00 40 00 00 00 C0</u>	w¢@Ç@
	00500150	00500150	40 40 40 40 40 40 40 40 41 40 40 40 42 40 40 40	

在教程的开始,当我们讨论 CPU 从二进制文件中读取的原生 "操作码"时,我提到过你能在十六进制查看器中看到原始数据。 不过,在 011y 中你不需要这么做。因为内存数据区就是一个内 置的十六进制查看器,以便于你查看原始的二进制数据,只查看 内存中的而不是磁盘上的。通常对于同样的数据有两种查看方 式,十六进制的和 ASCII 的。图片中右边的两列就是(第一列是 数据驻留内存中的地址)。011y 允许修改这些数据的显示方式, 后面的教程就会看到。

三、工具栏

不幸的是,011y的工具栏给大家留下了一点念想(尤其是 当英语并不是作者的母语)。我将左边的工具栏图标进行了注释:



这些都是控制代码运行的主要工具。记住这些,尤其是你开始使用 011y 的时候,这些按钮的所有功能都可以从"调试 (Debug)"菜单的下拉菜单中访问到。如果你不知道某些东西是 什么,你可以从菜单中看到。

关于一些图标我要多说几句。"Re-load"是用来重新启动应 用并暂停在入口点处。所有的补丁(后面会看到)都会被删除, 一些断点会失效,应用程序也不会运行任何代码。好吧,大部分 情况下是这样的。"Run"和 "Pause"做的就是你看到的那样。 "Step In"意思是运行一行代码然后暂停,如果有的话它会跟 进函数的内部。"Step Over"做同样的事情,不过它会跳过对 另一个函数的调用。"Animate"有点像 Step In 和 Step Over, 不过它特别慢好让你观察。这个你用的不多,不过有时候看代码 运行也挺有意思的,尤其是遇到多态二进制的时候能够观察到代 码的变化。讲的有点超前了.....

下面是各窗口的按钮图标 (更加有点神秘):

MEI	B MORY PATO	REAKPO	DINTS SETTI	NGS
LOG	WINDOWS		TRACE	HELP
	THREADS CPU		SOURCE	
EXECUTA MODUL	BLE HANDLES	REFE	RENCES	OLORS

点击其中的任何一个按钮都会弹出一个窗口,有些你会经常 用到,而有的却很少用。看这些字母并不是很直观,这点你可以 像我学习,把它们都点一遍直到你找到你需要的那个。每一个都 可以通过"View"菜单来访问,所以在第一次征程时你可以获得 些许帮助。下面我会介绍最常用的窗口:

1、(M)emory——内存映射窗口

M Memory map							
Address	Size	Owner	Section	Contains	Туре	Rocess	Initial acc
00010000 00020000 00120000 00120000 00120000 00140000 00150000 00250000 00250000 00250000 00250000 00250000 002500000	00001000 00001000 00003000 00003000 00001000 00001000 00006000 00006000 00006000 00006000 00006000 000016000 00016000			stack of main thread	Priv 00021004 Priv 00021004 Priv 00021104 Priv 00021104 Priv 00021040 Priv 00021040 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002 Map 00041002	RW RW Guarded RW Guarded R RWE RW RW RW R R R R R R R R R R R R	RW RW RW RW RW RW RW RW RW RW RW RW RW R
00220000 00330000 00350000 00350000 00370000 00370000 00350000 00350000 00350000 00350000 00350000 00350000 00350000 00350000 00400000 00400000 00405000 00427000		showstri showstri showstri showstri showstri	.text .bss .data .idata	PE header code code code, data code, imports	hap 00041002 Map 00041002 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002 Priv 00021004 Map 00041002 Map 00041002 Priv 00021040 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002	rrærræren Errærrærre H	L C C C C C C C C C C C C C
00428000 00478000 00476000 00500000 50091000 50091000 50102000 50125000 50125000 73090000 73090000 73091000 73091000 73091000 73091000 73085000 76380000 76380000 76381000 76381000 76381000 76381000 76381000 76381000 76381000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77305000 77462000 77462000		Showstri COMCTL32 COMCTL32 COMCTL32 COMCTL32 COMCTL32 COMCTL32 CRTDLL CRTDLL CRTDLL CRTDLL CRTDLL CRTDLL CRTDLL CRTDLL CRTDLL INM32 INM32 INM32 COMDL632 COMDC632 COM	.rsrc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc	Code, resources PE header code, imports, exports code, data code, resources code, relocations PE header code, imports, exports code, resources code, resources code, resources code, nections PE header code, imports, exports code, imports, exports code, relocations PE header code, imports, exports code, resources code, resources code, resources code, resources code, resources code, relocations PE header code, imports, exports code, resources code, resources code, resources code, imports, exports code,	Imag 01001002 Map 00041020 Map 00041020 Map 00041020 Priv 00021104 Imag 01001002 Imag 00000 Imag 0000002 Imag 0000002	RREA Guarded Guarded Guarded Guarded Guarded Guarded Guarded Guarded Guarded Guarded	Read and the second sec
774CD0000 774E0000 774E1000 77601000 7760E000 7760E000 7760E000 77610000 77C10000	00001000 0001000 00120000 00006000 00002000 00002000 00002000 00001000 00001000	comet[1 ole32 ole32 ole32 ole32 ole32 ole32 ole32 msvort msvort	.reloc .text .orpc .data .rsrc .reloc .text	code, relocations PE header code, imports, exports code code, data code, resources code, relocations PE header code, imports, exports	Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002	YRRRRRR.	RWE RWE RWE RWE RWE RWE RWE RWE RWE RWE

内存窗口显示程序已经分配的所有的内存块。它包括正在运行的程序的主段(本例中,是Owner列中的"Showstr")。在下面你能看到很多其他的段,这些都是程序载入进内存的DLL的, 准备将来用的。如果你双击其中的任何一行,都会打开一个显示 该段的反汇编代码(或十六进制数据)的窗口。这个窗口也显示 了块的类型和访问权限、大小以及该段载入内存的地址。

2. (P)atches——补丁窗口

该窗口显示的是你做的任何"补丁",即对原始代码的任何 修改。注意那个状态(State 列)是激活的(Active)。如果你 重新载入应用程序(通过点击 re-load 图标),这些补丁就会失 效。为了简便的使它们重新生效(或失效),点击期望的补丁以 及敲击空格键。这可以打开或关闭补丁。注意那个"Old"和"New" 列,显示的是原始的指令和修改后的指令。

3. (B) reakpoints——断点窗口

B Brea	kpoints			
Address	Module	Active	Disassembly	Comment
00401240 00401273 00401290	showstri showstri showstri	Always Always Always	HUD ESP,4 PUSH DWORD PTR DS:[426020] PUSH EBP	
				e

该窗口显示了当前所有断点设置的位置。这个窗口将会是你 的好朋友

4. (K)all Stack——调用栈窗口

(哎呀,我知道为什么初学者记这些图标比较难了.....)

-			and the second state of th		
K Call s	stack of n	nain thread			\times
Address	Stack	Procedure / arguments	Called from	Frame	~
0012FEBC 0012FEC0 0012FEE4 0012FEE8 0012FEE8	7E4191BE 7E42776B 00401395 0012FF30 00000000	Includes ntdll.KiFastSystemCallRet USER32.7E419182 <jmp.&user32.getmessagea> pMsg = 0012FF30 hWnd = NULL Weet lteemMin = 0</jmp.&user32.getmessagea>	USER32.7E4191BC USER32.7E427766 showstri.00401390	0012FEE0 0012FEE0 0012FEE0	
0012FEF0 0012FEF4 0012FF50 0012FF54 0012FF58	00000000 00000000 00404EF5 00400000 00000000 00000000	MsgFilterNax = 0 showstri,0040129C Arg1 = 00400000 Arg2 = 00000000 0rg2 = 00151555	showstri.00404EF0	0012FF4C	G
0012FF60 0012FF70 0012FF70 0012FFC4	000000001 00401284 7C817077	Arg4 = 00000001 showstri.00404E94 Maybe showstri.0040126C	showstri.0040127F kernel32.7C817074	0012FF6C 0012FFC0	
					~

这个窗口与前面看到的"堆栈区"不一样,它显示了更多信息,有关于代码中的调用、发送给这些函数的值以及其他的东西。 不久我们会了解到更多。

下一教程,我会包含我的经过"升级"的011y版本,有些 是你一看就明白的按钮。这里有张图片



四、上下文菜单

本教程的最后,我会快速介绍011y的右键菜单。它是许多操作产生的地方,所以你最少应该熟悉一下它。右键反汇编区的任何地方都会调出该菜单:



我只会介绍最常用的几项。随着经验的增多,你最终会遇到 那些较少用到的选项。"Binary"菜单项允许你按字节编辑二进 制数据。在这里你可以将埋在一堆二进制数据中的"未注册"几 个字改成"已注册"。"Breakpoint"菜单可以设置断点。断点分 好几种,下一章我会讲到。"Search for"有一个相当大的子菜 单。这里你可以搜索类似字符串、函数调用等二进制数据。 "Analysis"菜单会强制 011y 重新分析当前正在查看的代码段。 有时候 011y 会对你正在查看的是代码还是数据感到困惑(记住, 它们俩都只是一些数字),这个可以强制 011y 将你正在看的内容 当做是代码,并且尝试猜测该部分看起来应该是什么样子的。

注意,我的菜单看起来和你的可能不太一样,因为我装了一些插件,这些插件在菜单中添加了一些功能。不过别担心,后面的教程中我会介绍这些菜单的。

教程三: 011yDbg 的使用(上)

本章中我将会介绍 011yDbg 的使用。011y 有许多的功能, 唯一学好它们的方式是实践和练习。也就是说,本教程也只是给 你一个简单的概述。此教程不会涉及额外的内容,后面会进行重 点讨论。到最后,你应该会比较好的掌握 011y。

本章包含了一些文件。你能够下载那些文件,以及可以在这 里下载到次教程的 PDF 版本。

它们包括一个我们将在 011y 中用到的二进制文件、一个 011y 备忘单、我使用的外观上有些不同的 011y 以及一个新的 ini 文件。你可以用这个 ini 文件替换掉 011y 默认的 ini, 可以给 新人提供一些帮助(感谢伟大的 Lena151 做的这些)。你可以从 这 里 直接下载或者从教程页面下载。如果你更愿意用原版的 011y, 你可以从这里下载。

一、载入应用

第一步是将目的二进制文件载入 011y。你可以将二进制文件拖放到 011y 的反汇编窗口,或者点击顶部工具栏中的载入图标选择目的文件。我们这里载入"FirstProgram.exe",可以从本网站下载。011y 会进行分析(011y 的底部状态栏会显示分析进程) 然后停在程序的入口点(EP):



需要注意的第一件事是 EP 的地址是 401000,就是图片中的 第一列。这是可执行文件的一个相当标准的起点(该可执行文件 至少没有加过壳或混淆过)。如果你的看起来不太一样,并且011y 没有停在 401000,你可以尝试点击 Appearance 菜单,然后选择 debugging options,点击 "Events" 标签,并且确保 WinMain (if location is known)" 被勾选上。然后重启应用。

让我们给"FirstProgram.exe"的内存空间占用情况来张快照。点击"Me"图标(如果你使用的是不同版本的011y的话应该是"M"):

Address Size Dwn 00010000 00001000 00001000 0012000 00001000 00003000 0012000 00001000 00023000 0012000 00005000 00005000 0014000 00005000 00024000 0012000 00005000 00025000 00240000 00005000 00240000 00250000 00011600 00220000 00220000 00041000 00220000 00220000 00041000 0022000 00320000 00001000 Fir 00400000 00001000 Fir 00400000 00001000 Fir 00402000 00001000 Fir	ner Section	Contains	Type Priv 00021004 Priv 00021004 Priv 00021104 Priv 00021104 Priv 00021104 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002	Access RW RW Guarded RW Guarded R RW RW RW R R R R R R R R R R R R R R	Initial access A RW RW RW RW RW RW RW RW RW RW RW R R R R R R R
00010000 0001000 00020000 00001000 00120000 00003000 00120000 00003000 00140000 00005000 00240000 00005000 00250000 00015000 00250000 00015000 00250000 00015000 00250000 00015000 00250000 00015000 00250000 00001600 00320000 00002000 00320000 00001000 Fir 00400000 00001000 Fir 00402000 00001000 Fir 00402000 00001000 Fir 00402000 00001000 Fir	rstPro	stack of main thread	Priv 00021004 Priv 00021004 Priv 00021104 Priv 00021104 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041002	RW RW RW Guarded RW Guarded R RW RW RW R R R R R R R R R R R R R R	RW RR RW RW RW RW RW RW RW RW RW RW RW R
002200000 00041000 00320000 00004000 00330000 00002000 00400000 00002000 00400000 00001000 Fir 00401000 00001000 Fir 00402000 00001000 Fir 00402000 00001000 Fir	rstPro		Map 00041002	15	R
00410000 001000 00520000 001000 00530000 00076000 00330000 00076000 00330000 00001000 00340000 00004000 00350000 00004000	rstPro .text rstPro .rdata rstPro .data rstPro .rsrc	PE header SFX,code data,imports resources	Hap 00041020 Map 00041022 Imag 01001002 Map 00041002 Priv 00021004 Priv 00021004 Priv 00021004 Hap 00021004	rrrr Rrrrrrrr Rrrrrrrrr	R R R R R R R R R R R R R R R R R R R
00560000 00001000 00500000 00001000 5D090000 00001000 5D091000 00001000 5D091000 0001000 5D102000 00020000 5D122000 00020000 5D122000 00020000 76390000 0001000 76390000 00001000 76390000 00001000 76390000 00001000 76390000 00001000 76390000 00001000 76390000 00001000 76390000 00001000 7700000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 77246000 00001000 7741000 00001000 7741000 00001000 7741000 00001000 7741000 00001000	moti32 moti32 moti32 moti32 moti32 reico ma2 reico ma2 reico ma2 reico vapi32 vapi32 vapi32 vapi32 vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico vapi32 reico	PE header SFX.code, imports, exports resources PE header SFX.code, imports, exports data resources PE header SFX.code, imports, exports resources PE header SFX.code, imports, exports resources PE header SFX.code, imports, exports resources PE header SFX.code, imports, exports resources PE header SFX.code, imports, exports	Priv 00021040 Map 00041002 Priv 00021104 Imag 01001002 Imag 00	Rue Ru Ru Guarded RR RR RR RR RR RR RR RR RR RR RR RR RR	RWE R R R R R WE R R WE R WE R WE R WE

如果你看地址那一列,你会看到 401000 那行包含有大小 1000、名称 "FirstPro" (FirstProgram 的简写形式)、区块名 ".text"、包含里是 "SFX,code"。随着学习进度的展开,我们 就会知道 exe 文件中有不同的区块,包含不同的数据类型。该区 块中是程序的 "代码"。它有 1000 字节大,从内存的 401000 开 始。 在这个的下面你会看到 FirstProgram 的其他区块。其中.rdata 区包含着数据,其导入地址是 402000,地址 403000的.data 区中什么都没有。最后的那个.rsrc 区中存有资源(比如对话框、图片、文本等)。要注意的是这些区可以叫任何名字, 这个完全依赖于程序员。

你可能会问为什么.data 区是空的。好吧,它事实上就是那样。它一般包含全局变量和随机数据。011y只是选择了不显示,因为它确实不知道那里存储了哪种数据。

区段的顶部是一个叫做"Pe Header"的区块。这是一个非常重要的区,一个我们会在将来文章中深入探讨的区。不过对于目前来说,我们只需要知道它对于Windows就像一本指令手册一样,用来按步将文件载入内存,程序运行需要多少空间,还有其他某些事情等。它在大约所有 exe 的头部 (DLL 也是一样)。

如果你继续往下看,你可以看到不只是 FirstProgram 程序, 还有其他的文件。我们看到有 comct132, imm32,gdi32,kernel32 等。这些 DLL 是我们程序运行所需要的。DLL 是函数的集合,我 们的程序能够调用那些 Windows 已经提供的(或者其他程序员提 供的)函数。比如打开对话框、比较字符串、创建窗口以及类似 的功能。统称为 Windows API。程序使用这些函数的原因是,假 如我们写每一个用到的函数,仅仅显示一个消息框就需要数千行 的代码。然而, Windows 已经提供了像 CreateWindow 这样的函数来为我们做这些工作。对于程序员来说这使得编程要简单的多。

你或许会问这些 DLL 是如何进入我们的地址空间的, windows 是怎么知道哪一个是我们需要的。好吧,这些信息是存 储在上列出的 PE Header 中的。当 Windows 将我们的 exe 载入内 存时,它会检查头并找出 DLL 的名字,以及每个 DLL 中我们程序 需要的函数,然后将这些函数载入我们的程序内存空间,以便于 我们的程序调用它们。每个程序被载入内存时,它所需要的 DLL 也会被载入它的内存空间。可以想象得到,当有好几个程序当前 都需要被载入内存并且都需要某个特定的 DLL 时,那么有些 DLL 就有可能被载入内存好几次。如果你需要准确的知道我们的程序 调用了哪些函数,你可以右键点击 011y 的反汇编窗口,选择 "Search for" ——> ""All Intermodular Calls"。会显示如 下图:

R Found intermodular calls					
Address	Disassembly	Destination 🔨			
00401002 00401002 00401027 00401027 00401027 00401029 00401095 00401095 00401095 00401080 00401080 00401080 00401080 00401080 004011051 004011180 004011180 004011180 004011180 004011180 004011180 004011180 004011180 004011180 004011180 004011282 00401282 00401282 00401282	PUSH 0 CALL CALL <td>Initial CPU selection: kernel32.6etModuleHandleA kernel32.ExtCommandLineA kernel32.ExtCommandLineA kernel32.ExtProcess wser32.LoadIconA wser32.RegisterClassExA camct132.InitCommonControls wser32.CreateDialogParamA wser32.CreateDialogParamA wser32.SendDlgItemMessageA comct132.InageList_Create wser32.LoadImageA comct132.InageList_Add gdi32.DeleteObject wser32.SendDlgItemMessageA wser32.SendDlgItemMessageA wser32.SendDlgItemMessageA wser32.SetDlgItem wser32.SetDlgItem wser32.GetDlgItem wser32.IDialogMessageA wser32.IDialogMessageA wser32.DispatchMessageA wser32.DispatchMessageA wser32.BetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.DefWindow wser32.DefWindow wser32.DefWindow</td>	Initial CPU selection: kernel32.6etModuleHandleA kernel32.ExtCommandLineA kernel32.ExtCommandLineA kernel32.ExtProcess wser32.LoadIconA wser32.RegisterClassExA camct132.InitCommonControls wser32.CreateDialogParamA wser32.CreateDialogParamA wser32.SendDlgItemMessageA comct132.InageList_Create wser32.LoadImageA comct132.InageList_Add gdi32.DeleteObject wser32.SendDlgItemMessageA wser32.SendDlgItemMessageA wser32.SendDlgItemMessageA wser32.SetDlgItem wser32.SetDlgItem wser32.GetDlgItem wser32.IDialogMessageA wser32.IDialogMessageA wser32.DispatchMessageA wser32.DispatchMessageA wser32.BetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.SetDlgItemTextA wser32.DefWindow wser32.DefWindow wser32.DefWindow			
<		×			

这有点惊奇,不过这个列表非常的小。通常,对于一个商业 产品来说,需要数百或数千函数。不过因为我们的程序太简单了, 它需要的不是很多。你想想我们的程序干了什么,看起来好像是 那么多的函数只完成了如此简单的功能! 欢迎来到 Windows。该 窗口首先显示了 DLL 的名字,紧跟着的是函数的名字。比如, User32. Load I conA 是在 DLL User32 中,函数名字是 Load I conA。 该函数通常用来载入窗口左上角的图标。

下一步,我们搜索下程序中的所有字符串。右键点击反汇编窗口,选择"SearchFor"-> "All Referenced Text Strings":

R Text	strings referenced in FirstPro:.text	
Address 00401032 00401062 00401069 004010A5 004011F5	Disassembly MOV ELOCAL.121.30 MOV ELOCAL.31,FirstPro.00403009 MOV ELOCAL.21,FirstPro.00403000 PUSH FirstPro.00403010 PUSH FirstPro.00403019	Text string (Initial CPU selection) ASCII "MyMenu" ASCII "DLGCLASS" ASCII "DLGCLASS" ASCII "Dialog" ASCII "Dialog As Main"
E (11)		×

该窗口显示了我们程序中所有能找到的字符串。因为程序非 常简单,所以这里只有一点。大多数的程序如果没有加壳或混淆 的话,都有多得多的字符串(有时能达到十万)。这种情况下, 你有可能一个也看不到!加壳工具这样做的原因是逆向工程师 (至少新人是这样)严重依赖字符串来查找重要的函数。而删除 了字符串后就会难的多。想象一下,如果你搜索字符串然后看到 了"Congratulations! You entered the correct serial(恭 喜!你输入了正确的序列号)"会怎么样?嗯,这对于逆向来说 是巨大的帮助(我们会一次又一次的看到这个)。另外,双击其 中的字符串,你会来到反汇编窗口中使用该字符串的指令那。这 是一个很好的特性,你能够正确的跳转到使用字符串的代码。

二、运行程序

如果你看 011y 的左上角的话,会看到一个黄色背景的小区 块,里面写着"暂停 (Pause)"。意思是程序已经暂停了(本例 中是在开始的时候),等着你进行其他操作。所以,咱们开始干 一票吧!按一下 F9 (或者从"Debug"菜单中选择"Run")。一 会儿后,我们的程序会弹出一个对话框(它有可能显示在011y的后面,所以最小化011y以确保能看见窗口)。



刚才显示"Pause"的地方现在应该显示的是"Runing"。意 思是程序正在运行,不过是在 011y 中运行的。你可能会与我们 的程序进行一些交互,看看它是如何工作的以及它干了些什么。 如果你不小心关了它的话,返回到 011y 并且按下 Ctr1+F2 (或 选择 Debug->Restart)以重新载入程序,然后你可以点击 F9 让 程序再一次运行起来。

现在照着做:程序运行的时候,点击回到 011y 中,然后点击暂停图标(或点击 F12,也可以点击 Debug->Pause 菜单)。即使我们的程序正在运行,该操作会让程序暂停在内存中的任何地方。如果这时候你想看看程序,你会发现挺有意思的(程序一点

也不会显示出来)。这是因为当程序暂停的时候,Windows不会 更新视图。现在再一次点击 F9,你会发现你又可以和程序进行 交互了。如果有什么问题的话,只需要点击那个双左尖括号图标 或 Debug-restart (或者 ctr1-F2),程序就会重新载入并暂停在 入口处。如果你需要的话,你可以再一次运行它。

三、单步运行程序

运行一个程序确实挺爽,不过你却得不到有关于程序运行的 太多信息。让我们试试单步运行。重新载入应用程序(重新载入 按钮、Ctr1+F2或 Debug->restart),然后我们会暂停在程序的 开始处。按一下F8,你就会发现当前的行选择器下移了一行。 011y运行了一行指令,然后又暂停了下来。如果你够激灵的话, 就会发现堆栈区下滚了一行,并且在顶部有了一个新的入口。

CoModule = NH1 RETURN to kernel32.7C817077 ntdll.7C910228 7C817077 7C910228 FFFFFFF 7FFDA000 00000640 0012FFC8

这是因为我们执行了一条指令, "PUSH 0" 往堆栈里"压" 了一个 0。在堆栈中的显示是"pModule=NULL"。NULL 是 0 的另 一个名字。你有可能也注意到了那个寄存器区, ESP 和 EIP 寄存 器变红了。



当一个寄存器变红的时候, 这就意味着最后执行的指令修改 了该寄存器。本例中, ESP 寄存器(用来存放指向栈顶的地址) 增加了1, 因为我们向栈中压了一个新值。EIP 寄存器增加了2, 其中存放了将要运行的指令的地址。因为我们已经不在地址 401000了, 而是在401002。因为上一个运行的指令是两个字节 长。我们现在暂停在下一个指令处。这个指令是在401002, 这 正是当前 EIP 的值。

011y 现在暂停的指令是一个 CALL。CALL 指令意味着我们要临时暂停在我们当前所在的函数中, 然后去运行另一个函数。这 类似于高级语言中的方法调用, 举个例子:

int main()

{

```
int x = 1;
call doSomething();
x = x + 1;
```

}

这段代码中,我们首先让 x 等于 1,然后呢我们要在逻辑上 暂停这行代码,转而去调用 doSomething()。当 doSomething() 执行完毕后,我们会返回我们原来的逻辑,然后将 x 加 1。 当然,在汇编语言里也是一样。我们首先往栈中压了一个 0,现在呢我们又想调用一个函数,例子中调用了 Kernel32 dl1 中的 GetModuleHandleA():

C *G.P.U* - main thread, module FirstPro					
00401000	٢\$.	6A 00	PUSH 0	pModule = NULL	
00401002		E8 CF020000	CALL	GetModuleHandleH	
00401007		A3 28304000	MOU DWORD PTR DS: [403028], EAX		
0040100C		E8 BF020000	CHLL (JMP.&kernel32.GetCommandLineA)	GetCommandLineR	

好,再按一次F8。当前的行指示器会下移一行,而 EIP 仍 然会保持红色并且加了 5 因为刚刚运行的指令是 5 字节大小), 堆栈也回到了它原来的地方。刚刚发生的这些是从我们按下 F8 开始的,F8 的意思是"Step-Over (单步步过)",CALL 中的代码 被调用,然后 011y 暂停在了 CALL 的下一行。也就是 CALL 中的 程序执行了也做了某些事,但是我们跳过去了。

好了,现在我们看看其他的选项。重启程序(Ctr1+F2),按 下F8步过第一条指令,不过在CALL指令上我们这次按F7。你 会注意到整个窗体都变得不一样了:

C *G.P.U* - main thread, module FirstPro							
00401206 00401222 00401223 00401223 00401225 004012F5 004012F5 004012F5 004012F7 004012F7 004012F9 004012F8 004012F8 004012F8 004012F8 004012F5 00401300 00401303	26250505 FFFFFF8000000000000000000000000000	24204000 00204000 0C204000 08204000 04204000	JMP Du JMP Du JMP Du JMP Du DB 000 DB 00000 DB 0000000000	IORD PTR IORD PTR IORD PTR IORD PTR	DS:[<&kerne DS:[<&comct DS:[<&comct DS:[<&comct DS:[<&comct DS:[<&comct	132.GetModule 132.ImageList 132.ImageList 132.ImageList 132.InitCommo	kernel32.GetModuleHandleA comot132.ImageList_Add comot132.ImageList_Create comot132.ImageList_Destroy comot132.InitCommonControls

这是因为 F7 "Step-In(单步步入)" 那个 CALL, 意思是 011y

做了这个调用并暂停在了新函数的第一行。这种情况下,CALL 跳转到了一个新的内存区域(EIP=4012d6)。理论上,如果我们 按行通过这个新函数的话,我们最终还是会回到将我们带进来的 那个 CALL 后面的语句。当然,有快捷键可以完成同样的功能, 不过目前来说,咱们还是重启程序从头来吧。因为我怕教的太多 容易忘。

现在我们暂停在了程序的开始,按下F8(单步步过)4次, 我们会停在如下图的语句处:



你会看到在一块的四个 PUSH 语句。这回当你四次按下 F8 的 时候,注意观察堆栈区,会看到栈的增长(确实是向下增长,还 记不记得那个盘子的例子?)。我觉得我们开始理解什么是压栈 了.....

你可能会问我们为什么要将这些乱七八糟的数字往栈里压。 本例中这四个数字是作为参数传递给函数的(那个函数是在地址 401021 处)。我们将前面的那个高级语言程序做一点点修改就会 比较清楚了:

int main()
```
{
    int x = 1;
    int y = 0;
    call doSomething( x, y );
    x = x + 1;
}
```

这里我们声明了两个变量 x 和 y,并且将它们传递给了 doSomething()函数。doSomething 函数将会(可能)对这些变 量做些什么,然后将控制权还给调用该函数的程序。通过堆栈是 将变量传递给函数的主要方法之一:每个变量被压进堆栈,然后 调用函数。然后在函数中,这些变量被访问到。通常 PUSH 指令 的逆操作是 POP。

堆栈并不是做这件事的唯一方法,它只是最常用的。这些变量也可以被放到寄存器中,然后在被调用的函数内部访问寄存器。不过本例中,我们程序的编译器选择将变量放到堆栈中。在你学了汇编语言后,这些东西都会变得清晰(你正在学习汇编语言,不是吗?)。后面我们还会复习几次的。

现在,如果我们再按一次 F8,你会注意到 011y 的工具栏中 会显示 "Runing",我们程序的对话框就会显示。这是因为我们 单步步过了那个 CALL,说明那个 CALL 中存在程序的大部分。这 个调用的代码是等待用户进行一些操作的循环,所以我们永远也 不会将控制权交给 CALL 的下一行。那么,让我们修复它...... 点击回到我们的程序,点那个关闭按钮来结束应用。011y 会立即暂停在那个 CALL 的下一行:

 00401018
 . FF35 28304000
 PUSH DWORD PTR DS:[403028]
 Arg1 = 00400000

 00401026
 . E8 06000000
 PUSH Enx
 FirstPro.0040102C

 00401026
 . 50
 PUSH EAX
 PUSH EAX

 0040102C
 . 55
 PUSH EBP
 CALL (JMP.&kernel32.ExitProcess)

 0040102C
 . 55
 PUSH EBP
 CALL (JMP.&kernel32.ExitProcess)

你会注意到我们的程序也消失了。那是因为,在那个 CALL 的某个地方,对话框窗口被关闭了。如果你看下一行,你会发现 我们正准备调用 kernel32.dl1 -> ExitProcess。这是一个停止 应用程序的 Windows API。所以,基本上 011y 在窗口被关闭了 之后就暂停了,不过是在程序确实被终止之前。如果你这时按 F9,程序就会终止,011y 的活动栏就会显示"Terminated (已 终止)",我们就再也不能调试任何东西了。

四、断点

我们试试别的东西,重新载入应用(Ctr1+F12),然后在地址 401011 处的第二列上双击(也就是双击那个"6A 0A"操作码)。然后地址 401011 就会变红:



你刚才做的就是在地址 401011 处设置断点。当 011y 到达该 处时,断点就会强制 011y 暂停。有好几种不同的断点会因为不 同的事件而阻止程序运行。

1、软件断点(Software Breakpoints)

软件断点就是将断点所在地址处的字节用 0xCC 操作码替换 掉,也就是 int 3 指令。这是一个特殊的中断,用以告知操作系 统调试器希望在这里暂停,并且在执行该指令之前将控制权交给 调试器。你不会看到指令被修改成 0xCC,因为 011y 在背后做了 这个。当 011y 遇到异常时它会设一个陷阱,让用户做他们希望 做的事。如果你选择让程序继续运行(通过运行它或单步运行), 0xCC 操作码就会被原来的操作码替换回来。为了设置一个操作 码,你可以双击操作码那一列,也可以先选中你想设置断点的那 一行,然后右键点击它,选择 Breakpoints->Troggle(或按下 F2)。要删除断点你可以双击同一行,或右键点击选择 Breakpoints->Remove Software Breakpoint (或再次按下F2)。

现在我们在 401011 处设置了一个 BP (Breakpoints), 让程 序暂停在第一行指令处。按下 F9, 程序将运行并在我们设置的 断点处暂停。

这里我告诉大家一些有用的东西。点击工具栏上的"Br"图标或选择菜单中的View->Breakpoints。你会看到一个断点窗口, 里面显示了我们设置的断点。

B Brea	kpoints			
Address 00401011	Module Tutorial	Active Always	Disassembly PUSH ØA	
				~

通过它你可以快速的浏览所有你设置的断点。你可以双击任何一个断点,然后反汇编窗口就会跳转到那个断点处(如果你没有改变程序控制流的话,EIP仍然停在原来的地方。双击 EIP 寄存器会回到当前的行,并准备执行下一行)。

如果你选中一个断点,然后敲击空格键,断点就会在可用和 禁用之间来回切换。你可以选中一个断点,然后敲一下"De1" 键就会删除断点。

最后,重启程序,打开断点窗口,选中401011处的断点。 敲一下空格键,然后"Active"列将会变成"Disable(禁用)"。 现在运行程序(F9),你会发现011y不会停在我们设置的断点处, 因为它被禁用了。

2、硬件断点(Hardware Breakpoints)

硬件断点使用的是 CPU 的调试寄存器。CPU 内建的有 8 个寄存器,是 R0-R7。即使芯片中内建了 8 个,但是我们只能使用四个。它们可以被用来断在内存区的读、写和执行。硬件断点和软件断点的不同之处在于,硬件断点不会修改进程的内存,所以它

更可靠,尤其是在加壳或被保护的软件中。通过右键点击相关行可以设置硬件断点,选择 Breakpoints,然后选择 Hardware, on Execution。



唯一查看你已经设置的内存断点(译者注:这里应该是硬件断点)是打开"Debug"菜单,选择"Hardware Breakpoints"。 有个插件可以提供方便,不过我们后面再讨论。



3、内存断点 (Memery Breakpoints)

有时候你想查找程序内存中的字符串或在常量,但是你又不 知道程序在内存的什么地方。你可以用内存断点来告诉 011y 只 要任何一条指令读或写一个内存地址(或许多内存地址),然后 暂停就行,在任何地方都无所谓。有三种方法设置内存断点。

■ 对于一条指令,右键点击该行,然后选择 Breakpoint->Memory,

On Access or Memory, On Write.

- 要在内存数据区设置断点,在数据窗口中选中一个或多个字 节,然后右键选择和上面一样的操作。
- 你也可以对整个内存区域设置断点。打开内存映射窗口("Me"

图标或 View->Memery), 右键相关内存区域, 在弹出菜单中选择"Set Break On Access for either Access or Write"。

Memo	ory map						
Address	Size	Owner	Section	Contains	Access		
003F0000 00400000	00002000	FirstPro		PE header	_	Map 00041020 Imag 01001002	RE
00402000 00403000 00404000 00410000 00520000 00530000 00530000	00001000 00001000 00001000 0001000 0001000 00001000 00001000	FirstPro FirstPro FirstPro FirstPro	.text .rdata .data .rsrc	data, imports resources	Actualize View in Disa Dump in CP Dump	assembler 'U	Enter
00840000 00850000 00860000 00900000	00004000 00003000 00001000 00001000				Search		Ctrl+B
50090000 50091000 50102000	00001000 00071000 00003000	comet 132 comet 132 comet 132	.text	PE header SFX,code,impor	Set break-c	F2	
5D105000	00020000	comet 132	rsrc	resources	Secmemory	/ oreaxpoint on acces	S D
76390000 76391000 76386000	00001000 00015000 00001000	imm32 imm32 imm32	32 .reloc PE header .text SFX,code,impo .data data	PE header SFX,code,impor data	Set memory Set access	+	
763A7000 763AC000	00005000 00001000	imm32 imm32	.rsrc .reloc	resources	Allocate Me	morv	

五、内存数据面板的使用

你可以用数据面板检查被调试进程内存空间中的内容。如果 反汇编窗口的指令、寄存器或堆栈中的任何一项包含了对内存位 置的引用,你可以在该引用上右键然后选择"Follow in Dump", 随即数据面板就会向你显示该地址引用的内容。你也可以在数据 面板的任何地方右键单击选择"GoTo",然后输入要查看的地址。 咱们现在试试。

确保 FirstProgram 已经载入并且停在了入口处。现在,按下 F8 八次,来到了 401021 地址指令处,该处指令是 CALL FirstPro. 40102c。如果你注意看这行的话,会注意到这个 CALL 会向下跳转到 40102c 处,在当前行下面的第三行的地方。按下 F7 我们单步步入那个跳转,然后我们就来到了 40102c。记住这

是一个 CALL 指令,所以我们最后还是会回到 401021 的(至少是 该条指令后面的那条)。

00401019 0040101B 00401021	. 6A 00 . FF35 28304000 F8 0600000	PUSH 0 PUSH DWORD PTR DS: [403028]	Arg2 = 00000000 Arg1 = 00400000 FirstPro 00401020
00401026	. 50	PUSH EAX	ExitCode = 1423A8
00401027	L. ES 9E020000	CALL (JMP.&kernel32.ExitProcess)	ExitProcess
00401020	- BBEC	MOU EBP. ESP	
0040102F	. 83C4 AC	ADD ESP54	
00401032	. C745 D0 30000000	MOV ELOCAL, 123, 30	
00401039	. C745 D4 0300000	MOU FLOCAL, 101, FirstPro, 00401106	
00401047	. C745 DC 0000000	MOV ELOCAL.93,0	
0040104E	. C745 E0 1E000000	MOU ELOCAL.83,1E	

现在,单步执行代码(F8)直到401062。你也可以在这行 设置断点,然后按F9运行。还记得怎么设置断点吗?双击你想 设置断点的那行的操作码列。你也可以选中该行,然后按F2去 设置或取消断点。现在我们断在了401062:



现在,我们看看断下的那行。相关指令是 MOV [LOCAL.3], FirstPro.00403009。我确定你知道(因为你已经学了汇编语 言:P)这条指令是将地址 00403009 中的内容移动到堆栈中(这 里 011y 是用 LOCAL.3 表示的)。你可以在注释列看到 011y 已经 发现了该地址处的内容是字符串 "MyMenu"。好,下面让我们看 看。在指令上右键,选择 "Follow in Dump(数据窗口跟随)"。 注意这里有好几选项:

1	Run trace		r.	
	Go to			
CL	Thread		•	rsorA
	Follow in Pump		۶	Selection
SI	View call hee	Ctrl+K		Immediate constant
	Search for		۲.	Memory address
	Find references to			= NULL
-	View		٠	2
	Copy to executable			

这里我们选择"Immediate constant"。这回载入指令中的 任何地址。如果你选择了"Selection",数据窗口会显示高亮行 所在的地址,这里是 401062(也就是我们暂停的地方)。基本上 我们在数据窗口中看到的就是我们在反汇编窗口中看到的。最 后,如果我们选择了 Memory address"数据窗口会显示 LOCAL.3 的内存区域。这会显示我们正在使用的变量(在堆栈中)的内存。 下面是选择了 Immediate constant 之后的数据窗口的样子:

Address	Hex o	tump	and the second	-		-	-	-	-		L.C.	-		-		ASCII
00403009	4D 79	9 4D	65	6E	75	00	4D	79	44	69	61	60	6F	67	00	MyMenu.MyDialog.
00403019	44 69	9 61	6C	6F	67	20	41	73	20	40	61	69	6E	00	00	Dialog As Main
00403029	00 40	00 6	00	00	00	00	00	00	00	00	00	00	00	00	00	.0
00403039	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403049	00 00	00 6	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403059	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403069	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403079	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403089	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403099	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030A9	00 00	00 6	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030B9	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403009	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403009	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030E9	00 00	00	00	00	ØØ	00	00	00	ØØ	00	00	00	00	00	00	
004030F9	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403109	00 00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00400340	00 00	00	00	00	õõ.	00	ôô.	00	ôô.	00	00	00	ôô.	00	õõ.	

就像你看到的,数据窗口显示的内存是从403009开始的。 正是011y按指令从中载入字符串的地址。在右边你可以看到字 符串"MyMenu"。左边你可以看到每个字符的十六进制数据。你 可能注意到了在"MyMenu"后面有些其他的字符串。这些字符串 会在程序的其他部分被用到。

六、最后,来点有意思的!

此次教程的最后,让我们做些有意思的事情。让我们修改二进制数据来显示我们自己的信息!我们将字符串"Dialog As Main"改成我们自己的,然后看看效果。

首先,在数据窗口的ASCII列,点那个"Dialog As Main" 中的"D":



注意, 左边的第一个十六进制数据也高亮了。这个数字对应 字母"D"。如果你查查 ASCII 码表的话, 就会发现字母"D"的 十六进制数正是 0x44。现在, 选中整个"Dialog As Main"字 符串:

Address	Hex dump	ASCII
00403009 00403019 00403029 00403039 00403049	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	MyMenu.MyDialog. Dialog As Main .@.

在选中的内容上右键,选择"Binary" -> "Edit"。我们 就可以修改我们程序在内存的内容:

Backup Copy		LFirstPro.0040102C LEXitCode = 142388 LEXitProcess				
Binary	•	Edit Ctrl+E				
Label : Breakpoint	•	Fill with 00's				
Search for Find references Ctu View executable file	rl+R	Binary copy Binary paste				
Copy to executable file Go to		ASCII "DLGCLASS" RercName = IDI_APF hInst = MULL LoadIconA				
✓ Hex Text Short		RercHame = IDC_AF hInst = NULL LoadCursorA				
Page 1		- 1 10 - 6 - 6811				

然后就会弹出一个如下的窗口:

Edit data	at 00403019
ASCII UNICODE	Dialog As Main
HEX +00	E14 69 61 6C 6F 67 20 41 73 20 4D 61 69 6E
└─ Keep	size OK Cancel

第一个文本框以 ASCII 码的形式显示字符串。第二个文本框 是以 UNICODE 形式(我们的程序用不着,所以空着),最后那个 文本框是相关字符串的原始数据。好,咱们改一下。点一下字符 串的第一个字母("D"),然后输入任何你想要将"Dialog As Main" 覆盖掉的内容。要注意的是你输入的长度,别多了。否则你就会 覆盖掉程序需要的其他字符串,或者更糟糕是覆盖掉了程序需要 的代码!!! 这里呢,我输入的是 "Program R4ndom":

5E 64

完了之后呢点 OK 按钮,并允许程序(点 011y 的运行按钮或 按下 F9)。切换到我们的程序,然后随便输入什么都行,然后选 择菜单 "Option" -> "Get Text"。现在看看我们的对话框!

Dialog As Main		
Options		
Blah blah blah		
	Program R4ndom 🔀	
	Blah blah blah	
	OK	

注意到对话框的标题有什么不同没有。

(这一章真TM长啊,翻的我累死了!!!)

教程四:011yDbg的使用(下)

一、简介

此次教程我们继续学习 011y 的使用。我们将继续使用上一章的程序(我也会将它包含在下载里)。

你可以在 tutorials 中下载文件和 PDF 版的教程。

二、DLLS

就像我前面说的,当你启动程序时,DLL 被系统载入器载入。这回 我会细致的讲解。DLL (Dynamic Link Libraries) 是函数的集合,通 常由 Windows 提供(当任何人都可以提供),其中含有很多 Windows 程 序要用的函数。这些函数可以让程序员更容易的完成一些乏味的重复 性的任务。

例如,将字符串全部转换成大写是许多程序要实现的功能。如果 你的程序要多次使用该功能的话,你有三个选择:一是在你的程序中 自己编码实现;问题是,你不知道你的下一个程序是不是也会用到该 功能很多次。你可能需要在你使用到的程序里复制粘贴很多次相同的 代码。二是创建一个自己的库,这样任何程序都可以调用。这种情况 下,你可以创建一个 DLL,然后包含在程序中。该 DLL 可能有像 convertToUpper 这样的通用函数以便于程序调用,因此你只需要写一 次代码就行了。这样做的另一个好处是,你可以说你为字符串转大写 想到了一个很好的优化方案。第一个例子中,你需要将代码拷贝到所 有要用到该代码的程序中,但是在那个通用 DLL 例子中,你只需要修 改 DLL 的代码, 然后所有使用该 DLL 的程序都可以以最快的速度获益。 爽吧! 这就是 DLL 产生的真正原因。

最后一个选择是,使用 Windows 提供的一堆 DLL 中包含的数千个 函数中的一个。这样做有很多好处。第一个是, Microsoft 的程序员已 经花了多年时间来优化他们的函数,他们在很大程度上要比你牛逼。 第二,你不需要将你的 DLL 包含在应用中,因为 Windows 操作系统已 经内建了这些 DLL。最后,如果 Windows 决定修改他们的操作系统,你 自己的 DLL 有可能和新系统不兼容。同时,如果你使用 Windows 的 DLL, 它们肯定是兼容的。

三、如何使用 DLL

现在你已经知道了什么是 DLL,那就谈谈如何使用它们。DLL 基本 上就是一个你的程序可以调用的函数库。在你第一次载入应用程序时, Windows 载入器就会检查 PE 头(还记不记得 PE 头?)的特定区段,看 看你的程序调用了哪些函数,以及这些函数都在哪些 DLL 中。在将你 的程序载入内存后,载入器就迭代这些 DLL,将它们载入到你的应用程 序的内存空间。然后它再仔细检查你的程序的代码,并将你的程序调 用的 DLL 函数注入到正确的地址。例如,如果你的程序调用 Kernel 32 DLL 中(只是一个例子啊)的 StrToUpper 函数来将一个缓冲区里的字 母转换成大写,载入器要找到 Kernel 32 DLL,找到 StrToUpper 函数的 地址,并将地址注入到你的程序中调用该函数的那行代码处。你的程 序就会通过调用进入到 Kernel 32 DLL 的内存空间,执行 StrToUpper 函数,最后再返回到程序中。

让我们实际看看这个过程。011y 载入本教程包含的

FirstProgram.exe。011y 断在了第一行代码(从现在起我们就叫它入口点(Entry Point)——这很重要,因为这是我们详细讨论 PE 头的时候 PE 头中的叫法)。



如果你看第二行代码的话,你会看到一个对函数 kernel32.GetModuleHandleA的调用。第一步,我们看看这个函数是干 嘛的。我已经将WIN32.HLP文件以及一个教你怎样将它安装到你的 011y中的文本文档包含在了本课的下载里,就是为了防止你上一课没 有拿到它。安装该文件后,你在你不熟悉的Windows API上右键,会 显示一个该 API 是干什么的菜单。在你拷贝过去后,你需要重启下 011y。现在,在GetModuleHandleA 上右键,选择"Help on Symbolic

Name"。然后 011y 会显示一个该函数的备忘单:

GetMode	- HOLL Latter Allen	A Reg
FirstP	Backup	► X
Arg4 =	Сору	•
Arg2 =	Binary	► •
FirstPi	Assemble	Space 🚪
ExitPre	Label	: P
	Comment	; 0
	Breakpoint	► 10 m
	Hit trace	► 1516
	Run trace	► 51.55.4
FirstPi kernel:	Follow	Enter -
ASCII :	New origin here	Ctrl+Gray * 🕴
RsrcNar	Go to	
LoadIco	Thread	► ti
FirstPi FirstPi	Follow in Dump	► 100
-RsrcNar hInst :	View call tree	Ctrl+K
FirstP	Search for	•
[pWndCla Registe	Find references to	
CInitCor Param	View	
pDlgPro	Copy to executable	
pTempla	Analysis	
Createl	Help on symbolic name	Ctrl+F1
FUSCEI	Detach Process	
	Process Patcher	

🗞 Win 32 Programmer's Reference
File Edit Bookmark Options Help
Contents Index Back ≤< ≥>
GetModuleHandle Quick Info Overview Group
The GetModuleHandle function returns a module handle for the specified module if the file has been mapped into the address space of the calling process.
HMODULE GetModuleHandle(LPCTSTR <i>lpModuleName</i> // address of module name to return handle for);
Parameters IpModuleName Points to a null-terminated string that names a Win32 module (either a .DLL or .EXE file). If the filename extension is omitted, the default library extension .DLL is appended. The filename string can include a trailing point character (.) to indicate that the module name has no extension. The string does not have to specify a path. The name is compared (case independently) to the names of modules currently mapped into the address space of the calling process. If this parameter is NULL, GetModuleHandle returns a handle of the file used to create the calling process.
Return Values
If the function succeeds, the return value is a handle to the specified module.
If the function fails, the return value is NULL. To get extended error information, call GetLastError.
Remarks The returned handle is not global, inheritable, or duplicative, and it cannot be used by another process.
The handles returned by GetModuleHandle and <u>LoadLibrary</u> can be used in the same functions — for example, <u>GetProcAddress</u> , <u>FreeLibrary</u> , or <u>LoadResource</u> . The difference between the two functions involves the reference count. LoadLibrary maps the module into the address space of the calling process, if necessary, and increments the module's reference count, if it is already mapped. GetModuleHandle , however, returns the handle of a mapped module without incrementing its reference count.
Note that the reference count is used in French the determine whether the structure the function from the order of the second structure

Note that the reference count is used in **FreeLibrary** to determine whether to unmap the function from the address spac of the process. For this reason, use care when using a handle returned by **GetModuleHandle** in a call to **FreeLibrary**

那么,基本上这个函数就是为了获取我们程序内存空间的句柄。 在Windows中,如果你想对一个窗口(或者是相当一部分的其他对象) 做任何事情,你必须取得它的句柄。这基本上是Windows知道你正在 操作的对象的唯一标识符。GetModuleHandle其实比这个稍微复杂点, 不过当我们经历了更多知道了更多知识以后再回过头来讨论这个。

关闭帮助窗口,我们看看这个 CALL 去了哪里。011y 已经试着帮助 我们,它用函数名替换掉了 GetModuleHandleA 的真实地址。让我们看 看它驻留的地址是什么。点一下调用 GetModuleHandleA 的那行代码, 再按一下空格键,就会打开一个汇编窗口:

Assemble at 0040	1002		
CALL 004012D6	G		•
Fill with NOP's		Assemble	Cancel

该窗口有两个目的:第一它向你显示了正在被计算的(以防 011y 帮助性的替换了地址)真实的汇编语言指令,第二它允许我们编辑汇 编语言指令。在下一课前我们不会做任何编辑,这次我们只是看看地 址:4012D6。有两种方法可以跳转到该地址看看那儿有什么(而不用 真的运行程序)。选中"Call GetModuleHandleA"然后按下"Enter", 你也可以按 Ctrl+G 手动输入地址。我们试试第一种方法,选中401002 那行(第三列有相关指令)然后按回车键,你就会来到该 CALL 要调用 的地方:



现在这里比较有趣:它看起来确实不像执行 GetModuleHandleA 的 代码。更像是一些跳转。对此有一个很好的原因说明,不过不幸的是, 需要解释一下。

四、地址跳转表

有件事你需要知道,DLL 并不是总是一次性全部载入到内存。 Windows 载入器负责载入你的程序和所有需要的DLL,它可以修改被载 入的DLL 在内存中的位置(坦白的说,它甚至能够修改你的程序被载 入的位置,这些我们后面再说)。原因是这样的,现在有一个 Windows DLL 属于最先载入的那种,被映射到地址 80000000。好吧,恰好你自 己的程序也带有一个 DLL 且需要载入到地址 80000000。两个 DLL 当然 不能被载入到同一个地址,载入器必须将其中一个移到另一个地址。 这种情况时常发生,还被叫做重定位。

这里有个问题: 在你首次编写一个程序并写了一个调用 GetModuleHandleA的指令,编译器会准确的知道正确的DLL在哪, 然 后它会放一个地址在指令里, 有些类似于"Call 80000000"。现在, 当你的程序被载入内存时,它仍然会让这个 CALL 调用 80000000(我这 里说的有点过于简单了)。不过,如果载入器将这个 DLL 移到 80000E300 会怎么样?你的 CALL 会调用错误的函数!

PE 文件和此后的 Windows 文件围绕这个问题提出的解决方法是建 立一个跳转表。意思是你的代码在首次编译时,每一个对 GetModuleHandleA 的调用都指向你的程序的一个地点,然后这个地点 就会立即跳转到一个随意的地址(这是最后的正确的地址)。事实上, 所有对 DLL 函数的调用都采用了同样的技术。它们每一个调用特定的 地址,然后立即跳转到一个随意的地址。当载入器载入所有的 DLL 时, 它会遍历"跳转表",然后在内存中用真实的函数地址替换掉所有的随 意地址。下面是所有真实地址被填充后的跳转表的样子:



这个有点复杂,下面我举个例子。我会写一个短程序,使用完全随意的信息(只是为了证明我们的观点)来调用一个 Kernel32 DLL 中的函数 ShowMarioBrosPicture。下面是我的程序(没有特指哪种语言):

```
main()
{
    call ShowMarioBrosPicture();
    call ShowDoYouLikeDialog()
    exit();
}
ShowDoYouLikeDialog()
{
    If ( user clicks yes )
    {
        call ShowMarioBrosPicture();
        call ShowMarioBrosPicture();
        Call ShowMessage( "Yes, it's our favorite too!")
    }
    else
    {
        call showMessage( "You obviously never played Super Mario
Bros.");
    }
    这些代码被编译之后,对函数的调用将会被真实的地址替换,就
像下面这样 (再次声明,这里没有特指某种语言):
```

```
401000
         call 402000
                       // Call ChowMarioBrosPicture
401002
         call 401006
                       // Call showDoYouLikeDialog
         call ExitProcess
401004
401006
         Code for "Do You like It" dialog
        if (user clicks yes)
40109A
       call 402000 // call showMarioBrosPicture
40109C
         call 4010FE // call show message
40109E
       call ExitProcess
4010a1
      if (user clicks no)
4010a3
      call 4010FE // call show message
4010a5
4010a7
       call ExitProcess
4010FE code for show message
. . .
40110A retn
这些代码的后面就有可能是我们的跳转表(本例中, 跳转表中只
```

有 ShowMarioBrosPicture)。

402000 JMP XXXXXXXX

我们的程序(译者注:这里作者应该是将 our program 写成了 out program,所以我给翻译成我们的,小伙伴们可以自己查阅)并不知道 ShowMarioBrosPicture 在哪(或者说不知道 Kerner32 DLL 在哪),我 们程序的编译器只是用实际的调用地址填充 X(并不是正真的地址,你知道那么意思就行)。

当 Windows 载入器载入我们的程序时,它首先将二进制文件载入 内存,完成跳转表的构建,不过跳转表里没有任何真实的地址。然后 开始载入 DLL 到我们的内存空间,最后开始找出所有函数驻留的地方。 一旦它找到了 showMarioBrosPicture 的地址,它就准备进入跳转表并 用函数的真实地址替换掉 X。假定 showMarioBrosPicture 的地址是 77CE550A。我们的跳转表代码就会被替换成如下:

402000 JMP 77CE550A

因为 011y 能够发现该地址指向的是 showMarioBrosPicture,所以 它会帮助性的进入跳转表并将跳转表显示如下:

402000 JMP DWORD PTR DS: [<&kernel32.showMarioBrosPicture>]

现在,让我们回到FirstProgram 看看跳转表:

004012	iC \$− FF25 14204000	JMP DWORD PTR DS:[<&gdi32.DeleteObject>]	adi32.DeleteObject 😪
004012	52 \$- FF25 74204000	JMP DWORD PTR DS: [<&user32.CreateDialogParamA>]	user32.CreateDialogParamA
004012	8 \$- FF25 70204000	JMP DWORD PTR DS:[<&user32.DefWindowProcA>]	user32.DefWindowProcA
004012	E \$- FF25 6C204000	JMP DWORD PTR DS:[<&user32.DestroyWindow>]	user32.Destroublindow
0040120	4 \$- FF25 68204000	JMP DWORD PTR DS: [<&user32.DispatchMessageA>]	user32.DispatchMessageA
0040120	6 \$- FF25 60204000	JMP DWORD PTR DS:[<&user32.GetDlgItem>]	user32.GetDlgItem
004012	0 \$- FF25 64204000	JMP DWORD PTR DS:[<&user32.GetDlgItemTextA>]	user32.GetDlgItemTextA
004012	76 \$- FF25 5C204000	JMP DWORD PTR DS: [<&user32.GetMessageA>]	user32.GetMessageA
004012	2C \$- FF25 58204000	JMP DWORD PTR DS: [<&user32. IsDialogMessageA>]	user32.IsDialogMessageA
004012	32 \$- FF25 40204000	JMP DWORD PTR DS: [<&user32.LoadCursorA>]	user32.LoadCursorA
004012	38 \$- FF25 2C204000	JMP DWORD PTR DS:[<&user32.LoadIconA>]	user32.LoadIconA
004012	BE \$- FF25 30204000	JMP DWORD PTR DS: [<&user32.LoadImageA>]	user32.LoadImageA
004012	94 \$- FF25 34204000	JMP DWORD PTR DS: [<&user32.MessageBoxA>]	user32.MessageBoxA
004012	9A \$- FF25 38204000	<pre>JMP DWORD PTR DS:[<&user32.PostQuitMessage>]</pre>	user32.PostQuitMessage
004012	10 \$- FF25 3C204000	JMP DWORD PTR DS: [<&user32.RegisterClassExA>]	user32.RegisterClassExA
004012	A6 \$- FF25 78204000	JMP DWORD PTR DS:[<&user32.SendDlgItemMessageA>]	user32.SendDlgItemMessageA
004012	AC \$- FF25 44204000	JMP DWORD PTR DS:[<&user32.SetDlqItemTextA>]	user32.SetDlqĪtemTextA
004012	32 \$- FF25 48204000	JMP DWORD PTR DS: [<&user32.SetFocus>]	user32.SetFocus
004012	38 \$- FF25 4C204000	JMP DWORD PTR DS:[<&user32.ShowWindow>]	user32.ShowWindow
004012	3E \$- FF25 50204000	JMP DWORD PTR DS:[<&user32.TranslateMessage>]	user32.TranslateMessage
0040120	24 \$- FF25 54204000	<pre>JMP DWORD PTR DS:[<&user32.UpdateWindow>]</pre>	user32.UpdateWindow
0040120	CA FF25 20204000	JMP DWORD PTR DS:[<&kernel32.ExitProcess>]	kernel32.ExitProcess
0040120	0 \$- FF25 1C204000	JMP DWORD PTR DS:[<&kernel32.GetCommandLineA>]	kernel32.GetCommandLineA
004012)6 \$- FF25 24204000 -	<pre>JMP DWORD PTR DS:[<&kernel32.GetModuleHandleA>]</pre>	kernel32.GetModuleHandleA
0040120	C \$- FF25 00204000	JMP DWORD PTR DS: [<&comct132.ImageList_Add>]	comctl32.ImageList_Add
004012	E2 \$− FF25 0C204000	<pre>JMP DWORD PTR DS:[<&comctl32.ImageList_Create>]</pre>	comctl32.ImageList Create
004012	8 \$- FF25 08204000	JMP DWORD PTR DS: [<&comct[32.ImageList_Destroy>]	comctl32.ImageList_Destroy
004012	EE \$- FF25 04204000	JMP DWORD PTR DS: [<&comet[32.InitCommonControls]	comet132.InitCommonControls
004012	- 00	DB 00	
	-+ 00		
004012	5 00	DB 00	
004012 004012	-4 00 -5 00 -6 00	DB 00 DB 00	

在首次编写这个程序时,各种 DLL 中的函数被调用,但是编译器 不知道我们的程序在运行的时候这些函数是在内存的什么地方,所以 它要创建一些像下面这样的东西(不是很准确这里):

40124C	JMP XXXXX	// gdi32.DeleteObject
401252	JMP XXXXX	// user32.CreateDialogParamA
401258	JMP XXXXX	// user32.DefWindowProcA
40125E	JMP XXXXX	// user32.DestroyWindow
•••		

在载入器将我们的程序载入之后,再载入所有的 DLL 并查找所有的函数的地址。然后它会遍历每一个函数,用这些函数当前驻留的真实地址替换,就行前面图片中的那样。如果仔细想想的话,这确实是

相当巧妙的处理方法。如果不这样做的话,那么载入器就得遍历整个 程序,并对每个 DLL 中的每个函数的调用都用真实的地址进行替换。 那个工作量就大了。使用这种方法,载入器对于每个函数的调用只需 要替换一个地方,就是跳转表那样的。

还是看看我们自己的程序吧。重载应用并按下F7。点击选中401002 那行指令(和前面做的一样),再按下空格(和前面做的一样):



再一次提醒你注意那个地址,4012D6。现在按F7步入那个CALL, 注意我们来到了4012D6。如果你向上翻,你会注意到我们来到了跳转 表的中间:



现在,再点一下F7,我们就会来到GetModuleHandleA的真正的地址7780B741。有两种方法可以知道我们现在正在模块kernel32中,两者在不同的场合你都可能用到。第一个是Olly的CPU窗口标题:

*								
File	View	Debug	Plugins	Options	Window	Help	Tools	Brea
Pau	sed	P K					-	
С	G.P.	U* - ma	in thre	ad, mod	ule kern	nel32		
708 708 708 708 708	08741 08743 08744 08746 08746	38FF 55 88EC 837D ~ 74 1	08 00		10V EDI E PUSH EBP 10V EBP,E CMP DWORD	DI SP PTR S	S: LEBF	+81,0

你能看到它显示的是"module kernel32"。第二种方法是到内存

映射窗口查看地址:

*	6				
File View Debug Plugins P Paused	Options Window Help Tools	BreakPoint->	2 Ma Th Wi Ha Ca Pa Si H	n Re II Sr 👩	112
C *G P U* - main thread	Memory map				- 🗆 🛛
TC888F14 38FF 7C608748 55 7C608748 88EC 7C608748 88EC 7C608746 870.08.00 7C808746 FF75.08 7C808746 850.0 7C808746 850.0 7C808746 850.0 7C808746 850.0 7C808756 74.08 7C808756 74.08 7C808756 574.08 7C808766 50 7C808764 64:A1 1800000 7C808764 8840.98 7C808776 8840.98 7C808776 90 7C808777 38FF 7C808778 90 7C808775 90 7C808776 90 7C808775 </td <td>Hddress Size Duner 77F57000 0002000 gdi32 77F51000 0002000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77570000 00034000 kerne132 7030100 00034000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70900000 00001000 ntd11 7091000 00005000 ntd11 7091000 00002000 ntd11 7093000 00022000 user32 7E471000 00002000 user32 7E471000 00002000 user32 7E472000 00002000 user32 7E472000 00002000 user32 7E472000 0</td> <td>Section .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc</td> <td>Contains PE header SFX,code, imports,exports resources PE header SFX,code, imports,exports resources PE header SFX,code,exports resources PE header SFX,code, imports,exports resources data block of main thread</td> <td>Type Tmag 01001002 Tmag 00001002 Tmag 00001002 Tmag 000000 Tmag 0000002 Tmag 0000000 Tmag 0000000 Tmag 00000000 Tmag 0000000000 Tmag 000000000000000000000000000000000000</td> <td>Hooest RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR</td>	Hddress Size Duner 77F57000 0002000 gdi32 77F51000 0002000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77F57000 00021000 secur32 77570000 00034000 kerne132 7030100 00034000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70380000 00005000 kerne132 70900000 00001000 ntd11 7091000 00005000 ntd11 7091000 00002000 ntd11 7093000 00022000 user32 7E471000 00002000 user32 7E471000 00002000 user32 7E472000 00002000 user32 7E472000 00002000 user32 7E472000 0	Section .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc	Contains PE header SFX,code, imports,exports resources PE header SFX,code, imports,exports resources PE header SFX,code,exports resources PE header SFX,code, imports,exports resources data block of main thread	Type Tmag 01001002 Tmag 00001002 Tmag 00001002 Tmag 000000 Tmag 0000002 Tmag 0000000 Tmag 0000000 Tmag 00000000 Tmag 0000000000 Tmag 000000000000000000000000000000000000	Hooest RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR

你会发现我们所在的地址(7780B741)是在 kernel32 的代码段地 址空间中。

现在我们回头看看其他的函数调用。重启应用,按F8 直到 40100C 处。那行代码是对 GetCommandLineA 的调用。点击选中指令再按下空 格键,你就能够看见它指向的地址,是 4012D0:

Assemble at 0040100C		
CALL 0040120		-
Fill with NOP's	Assemble	Cancel

(不好意思, 鼠标把地址挡住了, 它是 4012D0) 现在我们来试试

手动定位该地址,你会经常用到这种方法的。按 Ctrl+G 或点那个**转到** 图标[▶],输入我们想要转到的地址:



你的"GOTO (转到)"窗口可能不太一样,这个问题待会解决。现

在点击 OK, 我们就会跳到跳转表中 GetCommandLineA 的位置:

004012D0 \$- FF25 1C204000 JMP DWORD PTR DS: [{&kernel32.GetCommandLineA}] kernel32.GetCommandLineA 004012D6 \$- FF25 24204000 JMP DWORD PTR DS: [{&kernel32.GetToduleHandleA}] kernel32.GetCommandLineA 004012D0 \$- FF25 00204000 JMP DWORD PTR DS: [{&kernel32.GetToduleHandleA}] comct132.ImageList_GetAd 004012E2 \$- FF25 02204000 JMP DWORD PTR DS: [{&comct132.ImageList_Create}] comct132.ImageList_Create 004012E8 \$- FF25 08204000 JMP DWORD PTR DS: [{&comct132.ImageList_Destroy}] comct132.ImageList_Destroy 004012E8 \$- FF25 04204000 JMP DWORD PTR DS: [{&comct132.ImitCommonControls} comct132.ImageList_Destroy 004012F4 00 DB 00 DB 00 comct132.ImitCommonControls comct132.ImitCommonControls 004012F5 00 DB 00 DB 00 DB DB </th <th>C *G.P.U* - main thread, mo</th> <th>dule FirstPro 🔉</th> <th></th>	C *G.P.U* - main thread, mo	dule FirstPro 🔉	
	09401203 \$- FF25 1C204000 00401206 \$- FF25 24204000 0040120C \$- FF25 00204000 004012E2 \$- FF25 00204000 004012E8 \$- FF25 08204000 004012E8 \$- FF25 04204000 004012E8 \$- FF25 04204000 004012F5 00 004012F5 00 004012F5 00 004012F5 00 004012F5 00 044012F5 00	JMP DWORD PTR DS:[{&kernel32.GetCommandLineA>] JMP DWORD PTR DS:[{&kernel32.GetModuleHandleA>] JMP DWORD PTR DS:[{&comctl32.ImageList_Add>] JMP DWORD PTR DS:[{&comctl32.ImageList_Create>] JMP DWORD PTR DS:[{&comctl32.ImageList_Destroy>] JMP DWORD PTR DS:[{&comctl32.InitCommonControls} DB 00 DB 00 DB 00 DB 00 DB 00	kernel32.GetCommandLineA kernel32.GetModuleHandleA comotl32.ImageList_Add comotl32.ImageList_Create comotl32.ImageList_Destroy comotl32.InitCommonControls

按下F7 我们就来到了 kernel32 中的 GetCommandLineA 的开始处。 这个函数从 7C812FBD 开始:

7C812FBD	A1 <u>F455887C</u>	MOV EAX, DWORD PTR DS: [7C8855F4]	
7C812FC2	C3	RETN	
7C812FC3	90	NOP	
70812F04	90	NOP	
70012506	90	NOP	
7C812FC7	90	NOP	
7C812FC8	FFFF	222	Unknown command
7C812FCA	FFFF	222	Unknown command
7C812FCC	0000	ADD BYTE PTR DS:[EAX],AL	
7C812FCE	0000	ADD BYTE PTR DS:[EAX],AL	
7C812FD0	<u>B3_21</u>	MOV BL,21	
7C812FD2	<u>807C</u> 90 90 90	CMP BYTE PTR DS:[EAX+EDX*4-70],90	
7C812FD7	90	NOP	
70812FD8	90		
70812FD9	SBFF	DUCH EDD	ntall./C910228
7C812FDD	SS	MOULERP ESP	
7C812FDE	56	PUSH EST	
7C812FDF	64:A1 18000000	MOV EAX. DWORD PTR FS: [18]	
DOOLOFFE	2020 02 54	040 DU000 0T0 00 1500 01 00	

五、跳入及跳出 DLL

当我们围着一个程序转的时候,你不知道什么时候就在 DLL 中结 束了。如果你正在尝试攻克一些保护方案时,通常你是不愿意在 DLL 中转的,因为 Windows DLL 中真的没什么东西。关于这方面的一个告 诫,如果你正试着逆向的程序本身就带有 DLL 并且你就是想将它们也 进行逆向工程(或者是保护机制确实在 DLL 中)。这里有几种从 DLL 回 到我们的程序的方法。一个方法是单步通过所有的 DLL 函数代码直到 最后你返回到程序,当然这可能得一会时间(有些情况下像 VB 程序, 就是永远)。第二个选择是,点开"Debug"菜单并选择"Execute till user code (执行直到用户代码)"或者按 Alt+F9。意思是执行 DLL 中 的代码直到我们返回到我们自己的程序代码。要注意的是,有时候这 不一定好使,因为如果 DLL 访问了一个在我们的程序空间中的 buffer 或者变量的话,011y 就会停在那儿,所以你最终可能会按 Alt+F9 好几 次才能回来。

我们来试试这个方法。我们当前应该暂停在 7C812FBD,也就是 GetCommandLineA 的开始处。好,按下 Alt+F9。我们会回到程序中对 kernel32 调用的指令的后面那条指令(往上一行就是那个 CALL)。

现在我们来试试另外一个回到我们的代码的方法。重启程序,单步步过(F8)直到对GetCommandLineA调用的那个CALL(40100C)。单步步入(F7)那个CALL,并且单步步入那个jmp进入跳转表。现在,我们回到了GetCommandLineA的开始处:



现在打开内存映射窗口,滚动到我们的程序的代码段那块(起始 地址是 400000,写着 PE Header):

M Memo	ory map					. 🗆 🔛
Address 00240000 00250000 00260000	Size 00006000 00003000 00016000	Owner	Section	Contains	Type Priv 00021004 Map 00041004 Map 00041002	RW RW R
00280000 00200000 00320000 00350000 00350000 00401000 00401000 00401000 00403000 00403000 00403000 00404000 00404000 00404000	00041000 00041000 00006000 00005000 00005000 00001000 00001000 00001000 00001000 00001000 00001000 00001000	FirstPro FirstPro FirstPro FirstPro FirstPro FirstPro	.text .rdata .data .rsrc	PE header SFX.code data.imports resources	Map 00041002 Map 00041002 Map 00041020 Map 00041020 Map 00041020 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Map 00041002 Map 00041002	22222222222222222222222222222222222222
00520000 00830000 00830000 00850000 00960000 00900000 5D090000 5D091000 5D102000 5D102000 5D125000	0001000 00002000 00001000 00004000 00003000 00001000 00001000 00001000 00001000 00001000 00003000 00020000 00020000	comet 132 comet 132 comet 132 comet 132 comet 132	.text .data .rsrc .reloc	PE header SFX,code,imports,exports resources	Priv 00021004 Map 00041020 Priv 00021004 Map 00041002 Priv 00021040 Map 00041002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002	

现在,点击选中401000 那行,我们的.text 区段在那行。按下 F2

设一个内存访问断点(或右键选择 Breakpoint on access):

Address Size Owner Section Contains Type Access 00240000 00006000 Priv 00021004 RW 00260000 00016000 Priv 00021004 RW 00280000 00016000 Priv 00041002 R 00280000 00041000 Priv 00041002 R 00320000 00006000 Priv Priv 00041002 R 00330000 00005000 Priv Priv	Memory map					. 🗆 🛛
00240000 00006000 Priv 00021004 RW 00250000 00003000 Map 00041004 RW 00260000 00016000 Map 00041002 R 00220000 00041000 Map 00041002 R 00320000 00065000 Map 00041002 R 00330000 00005000 Map 00041002 R 00350000 00005000 Map 00041002 R 00350000 00005000 FirstPro PE header Map 00041002 R 00402000 00001000 FirstPro .text SFX.code Imag 01001002 R 00402000 00001000 FirstPro .rdata data, imports Imag 01001002 R 00404000 00001000 FirstPro .rsrc resources Imag 01001002 R 004000 00001000 FirstPro .rsrc resources Imag 01001002 R	Address Size	Owner	Section	Contains	Туре	Access A
00860000 00001000 RWE 00900000 00002000 comctl32 resources Priv 00021040 RWE 5D090000 00001000 comctl32 .text SFX.code, imports, exports Imag 01001002 R 5D102000 000020000 comctl32 .text .sFX, code, imports, exports Imag 01001002 R 5D105000 000020000 comctl32 .text .sFX, code, imports, exports Imag 01001002 R 5D125000 000020000 comctl32 .rsrc resources Imag 01001002 R 5D125000 000020000 comctl32 .reloc resources Imag 01001002 R	00240000 00006000 00250000 00003000 00260000 00016000 00280000 00016000 00280000 000416000 00280000 000416000 00220000 00041000 00350000 00005000 00350000 00005000 00350000 00005000 00350000 00001000 0040000 00001000 00402000 00001000 00402000 00001000 00402000 00001000 00402000 00001000 00403000 00001000 0041000 00103000 00440400 00001000 00520000 00001000 00530000 00002000 00530000 00002000 00530000 00002000 00530000 00002000 00530000 00002000 00530000 00002000 0050000 00002000 0050000 00002000 00900000 00002000 <td>FirstPro FirstPro FirstPro FirstPro FirstPro comet 132 comet 132 comet 132 comet 132</td> <td>.text .rdata .data .rsrc .text .data .rsrc .reloc</td> <td>PE header SFX.code data, imports resources PE header SFX.code, imports, exports resources</td> <td>Priv 00021004 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041020 Map 00041020 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Map 00041002 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002 Priv 00021040 Map 00041002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002</td> <td>лахахахададададахахалахахахада Пт тт так К</td>	FirstPro FirstPro FirstPro FirstPro FirstPro comet 132 comet 132 comet 132 comet 132	.text .rdata .data .rsrc .text .data .rsrc .reloc	PE header SFX.code data, imports resources PE header SFX.code, imports, exports resources	Priv 00021004 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041002 Map 00041020 Map 00041020 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002 Map 00041002 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Priv 00021004 Map 00041002 Priv 00021040 Map 00041002 Imag 01001002 Imag 01001002 Imag 01001002 Imag 01001002	лахахахададададахахалахахахада Пт тт так К

现在,运行程序。011y 会断在和上面相同的那行,就是 401011 处,也就是我们对 DLL 调用 CALL 之后的那行!!! 好,现在删除内存断 点,否则你会纳闷,为什么每次你运行程序的时候它都会断在下一行 六、再议堆栈 堆栈是逆向工程中的非常重要的一部分,如果对它理解的不够深入的话,你永远也不会成为一个伟大的逆向工程师。下面我们针对它做几个实验:

首先,看看寄存器窗口(在重启应用之后),看那个 ESP 寄存器。 该寄存器中的地址指向栈顶。本例中,ESP 的值是 12FFC4。现在看看 下面的堆栈窗口,列表中的顶部地址和 ESP 中的地址是一样的。

0012FFC4 0012FFC0 0012FFC0 0012FFD4 0012FFD4 0012FFD4 0012FFD5 0012FFD0 0012FFE0 0012FFE0 0012FFE4 0012FFF8 0012FFF6 0012FFF6 0012FFF6	7C817077 7C910228 FFFFFFF 7FFD6000 8054B6ED 0012FFC8 870CB4D0 FFFFFFFF 7C837080 00000000 00000000 00000000 00401000 00000000	RETURN to kernel32.7C817077 ntdll.7C910228 End of SEH chain SE handler kernel32.7C817080 FirstPro. <moduleentrypoint></moduleentrypoint>	
--	---	---	--

现在按F8(或者F7)一次,将0压入堆栈,再看看堆栈窗口:

0012FFC0	00000000	🖣 pModule = NULL 🔍
0012FFC4	7C817077	RETURN to kernel32.7C817077
0012FFC8	7C910228	ntdl,7C910228
0012FFCC	FFFFFFF	
0012FFD0	7FFD6000	
0012FFD4	8054B6ED	
0012FFD8	0012FFC8	
0012FFDC	870CB4D0	
0012FFE0	FFFFFFF	End of SEH chain
0012FFE4	70839808	SE handler
0012FFE8	70817080	kernel32.7C817080
0012FFEC	00000000	
0012FFF0	000000000	
0012FFF4	000000000	Einst Dus (Madula EstavuDalista)
0012FFF8	00401000	FirstFro. (Hodu leEntryFoint/
0012FFFC	00000000	

就像我们上次课提到的那样,该操作将0(null)压入堆栈。现在 看看 ESP 寄存器:

Registers (F	PU)
EAX 00000000	
50X 0012FFB0	ptdll KiEsstSustemCallRet
7FFD6000	noutrakti astoystendattnet
ESP 0012FFC0	
EBP 0012FFF0	
EDI 7C910228	ntdll.7C910228
EIP 00401002	FirstPro.00401002
C 0 FS 0023	32bit 0(FEFEFEE)

已经变成了 12FFCO。因为,在向堆栈中压入一个字节后,该字节 就变成了新的栈顶。按 F8 一次,单步步过对 GetModuleHandleA 的调

用,再看看堆栈窗口:

0012FFC4 0012FFC8 0012FFC0 0012FFD0 0012FFD0 0012FFD0 0012FFD0 0012FFD0 0012FFE0 0012FFE0 0012FFE4 0012FFE8 0012FFF0 0012FFF0 0012FFF6 0012FFF6	7C817077 7C910228 FFFFFFF7 7FFD6000 805486ED 0012FC8 870CB4D0 FFFFFF8 7C837080 00000000 00000000 00000000 00000000	RETURN to kernel32.7C817077 ntdll.7C910228 End of SEH chain SE handler kernel32.7C817080 FirstPro. <moduleentrypoint></moduleentrypoint>	<
0012FFFC	00000000		

注意我们的堆栈已经向下回退了一位(ESP 寄存器也回到了原来的 值)。这是因为 GetModuleHandleA 函数使用了这个被压入堆栈的 0,并 把它作为参数。然后把它"POP (弹)"出了堆栈,因为这个 0 已经没用 了。就行上一课提到的,这是向函数传递参数的一种方法:将参数压 栈,被调用的函数将它们弹出栈,使用它们,然后返回,通常我们需 要的信息都在寄存器里 (后面会看到)。

接着继续...。如果你按 F8 两次单步步过对 GetCommandLineA 的 调用,会发现堆栈并没有改变。因为,我们没有向堆栈中压入任何信 息以供函数使用。接下来,是一个 PUSH OA 的指令。这是准备传递给 下一个被调用函数的第一个参数。单步步过,然后你会发现 OA 出现在 了栈顶,ESP 寄存器下移了 4 (当你向堆栈压入一个值时,ESP 寄存器 会向下移,因为堆栈在内存中是向下"增长"的。译者注:堆栈是从高 址向低址增长。)现在再按一次 F8,ESP 寄存器会再次下移 4。因为我 们向堆栈中压入了一个 4 字节的值。如果你看堆栈的顶部,就会发现 我们向堆栈中压入了 00000000。为什么呢?

我们看看做这个压入操作的那行代码,在401013处:

PUSH DWORD PTR DS:[40302c]

这行代码的意思(我保证你知道什么意思,因为你已经学了汇编

语言:p) 是取地址 40302C 开始的 4 字节内容, 然后将它们压入堆栈。 那么在 40302C 的是什么呢? 好吧, 当然是 00000000! (开个玩笑) 我 们来自己看看。右键 401013 处的指令,选择"Follow in Dump (数据 窗口跟随) "->"Memory Address (内存地址)"。然后会在内存数据窗 口中显示以 40302C 开始的内存中的内容:

Address	He	k di	amp														ASCII	
0040302C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040303C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040304C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040305C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040306C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040307C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040308C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040309C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030AC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030BC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030CC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030DC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	<u>.</u>	
004030EC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030FC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040310C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040311C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040312C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040313C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0040314C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

显然,哪里可没有那么多内容!不过你至少知道 0 是从哪儿来的。 如果你想知道更多的细节比如这块内存是干什么的,这块内存空间被 用来存储变量,并且最终会被这些变量填充。不过对于目前来说,所 有的变量都被初始化为 0。

现在按一下 F8, 我们遇到了另一个 PUSH 指令,不过这次是从 403028 开始。如果你在数据窗口中向上翻,会看到该地址处也是 0(在 我们上一次课修改的字符串的后面)。这一块正在做的是将内存指针压 栈,当前被设置为 0,我们的代码将会以变量的形式使用。单步步过上 一个 PUSH 然后单步步入对地址 40101C 的调用。你应该注意的第一件 事是有什么东西被压入堆栈里了:我们的 CALL 的返回地址,401026。

任何代码在使用 CALL 指令时,在我们还没有执行这个调用前,下 一条将要被执行的指令(译者注:非 CALL 内部的指令)的地址会被自 动的压入堆栈。原因是,我们调用的函数执行完后,它需要知道返回 到什么地方。被自动压入堆栈的地址就是返回地址。看那个堆栈窗口的顶部:



可以看到 011y 已经指出了它是一个返回地址,并且它指回到我们的程序(FirstPro),需要被返回的地址是 40102C(CALL 的下一条指令)。

现在,在函数的结尾,一个 RETN 指令将会被执行(你肯定知道它 是"return"的意思,因为它出现在你的汇编语言书的开头处)。这个返 回指令真正的意思是"弹出栈顶的地址,将正在运行的代码指向这个地 址"(它主要是用弹出的值替换 EIP 寄存器——存储当前正在运行的行 的地址)。那么现在,被调用的函数在执行完后准确的知道了要返回到 哪!事实上,如果你向下滚动一点,就会发现 4011A3 处的 RETN 语句 会从堆栈中弹出这个地址,然后从该地址开始运行:



(RETN 语句后面的那个 10, 意思是给我返回地址, 然后再从堆栈 中删除 10h 字节的空间, 因为我再也不需要它们了。看看你汇编语言 书籍的下一页吧)

这里我们花点时间来启动一句,我保证在逆向工程社区会火的口头禅。我喜欢叫它"Random's Essential Truths About Reversing Data (Random 关于逆

向数据的必备真言——译者注:大体这个意思吧,就这么翻吧,反正咱们也不会 喊)",或者 R. E. T. A. R. D (首字母缩写的听起来还不错)。我正式开启下面这个 即将成为传奇的戒律:

#1. You MUST learn assembly language(#1、你必须学习汇编语言).

如果你还没有的话,在逆向工程领域你不会取得成功。就是那么简单。

本次教程我准备最后谈论的是,011y 怎么处理参数和本地变量的显示。如果你双击 EIP 寄存器,我们就能跳回到代码的当前行(在40101C 处),往下可以看到好几行蓝色标记的行,显示的有 LOCAL 字样(其中一个显示 ARG):



如果你没有任何编程经验,你可能不太知道本地变量和参数之间 有什么不同。对于参数,就像我们早些时候讨论的,是传递给函数的 变量,通常通过堆栈传递。本地变量是被调用函数"创建"的用来临 时性存储数据的一种变量。下面是一个例子程序,其中有两个不同的 概念:

```
main()
{
    sayHello("R4ndom");
}
sayHello(String name)
{
```

```
int numTimes = 3;
String hello = "Hello, ";
for( int x = 0; x < numTimes; x++)
        print( hello + name );
}
```

程序中,字符串"R4ndom"是传递给 sayHello 函数的参数。在汇 编语言中,这个字符串(至少是这个字符串的地址)会被压入堆栈, 以便于 sayHello 函数引用。一旦控制权转给了 sayHello 函数, sayHello 需要设置一对本地变量(LOCAL VARIABLES),这对变量函数 会使用,不过一旦函数执行完毕就不再需要它们了。例子中的本地变 量是整形数据 numTimes、字符串 hello、整形 x。不幸的是,为了防止 堆栈不够负责,参数和本地变量都存储在堆栈中。堆栈通过 ESP 寄存 器来实现这个,不过寄存器可没有超能力。它通常指向栈顶,不过它 是可以被修改的。所以,可以说我们进入了 sayHello 函数,并且堆栈 中有下面的数据:

1、字符串"R4ndom"的地址

2、让我们进入函数的那个 CALL 的返回地址。

如果我们想要创建一个本地变量,我所需要做的的是从 ESP 寄存 器中减去一定的值,这样就会在堆栈中创建一定的空间! 假如我们将 ESP 减去4(会有4个字节大小,或者一个32位的数)。堆栈会像下面 这样:

1、空的32位数

2、字符串"R4ndom"的地址

3、让我们进入函数的那个 CALL 的返回地址。

现在,我们可以在这个地址里放任何数据,比如,我们可以让它

存储 sayHello 函数中的变量 numTimes。因为我们的函数使用了三个变量(所有的都是 32 位长),需要从 ESP 减去 12 字节(或十六进制的 0xC),然后我们就有了三个可以使用的变量。堆栈就会像下面这样:

1、指向字符串"hello"的空的 32 位地址。

2、变量"x"的空的 32 位数

3、变量 "numTimes" 的空的 32 位数

4、字符串"R4ndom"的地址

5、让我们进入函数的那个 CALL 的返回地址。

现在, sayHello 可以填充、修改以及重用这些地址以用于我们的 变量,在第一个位置处有传递给函数的参数(就是字符串"R4ndom")。 当 sayHello 执行完毕后,它有两种方法来删除这些变量和参数(因为 函数执行完毕后不在需要它们),然后将堆栈还原:1)它可以将 ESP 寄存器修改回它被修改之前;2)使用后面带数字的 RETN 指令。第一 种方法,为了让程序能够记住 ESP 的原始数据,它使用了另一个寄存 器——EBP,目的是当我们第一次进入 sayHello 函数时能够追踪到堆 栈指向的原始位置。当函数准备返回时,它从 EBP 中拷贝 ESP 的原始 值(开始的时候存储在 EBP 中)到 ESP 和 BAM 中。返回地址现在在堆 栈的顶部,当 RETN 指令运行时,它用通过这个返回到我们的主程序中。

第二种方法,你可以告诉 CPU 堆栈中有多少字节你不再需要了, 然后它就会从栈顶删除这些字节。在我们的例子里,我们用 RETN 16 (十六进制就是 0xF),这样就会从栈顶除去 16 字节(或4个32 位数), 将返回我主程序的地址留在新的栈顶。具体的返回机制依赖于编译器, 不过你两个都会看到。

现在,我们回到我们的FirstProgram.exe:



可以看到 011y 已经注释出了一个参数和 12 个本地变量。在我们的程序中这些本地变量是用来追踪类似于图标、我们输入的文本的缓存地址、输入的文本长度等。完成后,就会弹出这些值、将 ESP 寄存器值改回 EBP 或 RETN 一个数字(本例中,三个都有!!!)

我知道堆栈是非常复杂的设计,但是我保证在混乱一段时间以后你会掌握它的窍门。汇编语言的书也会帮很大忙的。

(最近忙着装修进度较慢,而且第三章和第四章真的好长,这一 章近万字,翻译不易呀)

教程五: 第一次破解(算是)

一、简介

此次教程通过预览一个 crackme, 我们会结束 011y 使用方面剩下的内容。好吧, 算是一个 crackme。其实就是我们前面使用的程序,不过被修改成需要序列号注册了,如果输入正确序列号会显示一个好消息,否则显示一个坏消息。我选择这么做而不是用一个完全不同的 crackme, 是因为我想要你能够专注于序列号校验程序部分,而不是陷入其他的代码中。下一课我们会研究一个真正的 crackme (我保证)。

此次教程你所需要的就是一个 011yDBG (我的版本或者原始 版本都可以),以及一个我改进了的 crackme。顺便说一下,我 把改进后的 crackme 叫做 "First Assembly Kracking Engine", 或者是 F. A. K. E. 它包含在此次教程的文件下载中。(是的,Gdogg, 我知道 Kracking 不是以字母 "K"开头的。译者注:如果取 crack 首字母,最后缩写就是 face 了,还有那个 Gdogg 我不造啥意思) 我们开始吧。


如果你在 011y 中载入了 FAKE. exe 的话, 就会发现第一页代码和我们上一次学习用的程序一样。



INASM	

点击注册弹出下面的对话框。

	Enter Registration Code	
	J.	Enter Serial Exit
输入	序列号。	
	Enter Registration Code	
	12121212121212121212	Enter Serial Exit
在点	了 Enter Serial 后,出现了	下面这个坏消息
	Enter Registration Code	
	That serial is incorrect	Enter Serial Exit

真见鬼! 我那么努力的尝试!!! 😀

现在我向你介绍每个新手查找注册校验代码的第一个方法。

二、搜索所有的文本字符串

先说一下,有许多"老练"的逆向者(或破解者)觉得这个 方法已经很少用了。因为这个方法太过于明显了,所以凡是想保 护自己的软件不被逆向的人都会让这招失效。这些软件被压缩、 保护、加密或修改,只要作者不是一个完完全全的傻子,就会加 密字符串以让"Search for strings(搜索字符串)"方法失效。 话虽如此,不过我还是发现有许多傻子,这个消息可别告诉任何 老鸟,所以我做的第一件事就是检查这个(ps.这其实也是老鸟 做的第一件事)。

基本上,该方法都会涉及到让011y 搜索你的程序的内存空间,搜索任何看起来像是 ASCII 或 Unicode 文本字符串。通常,可以立即发现该方法好不好用,会有大量的文本字符串,许多看起来很诱人(比如"Thank you for registering!!!(谢谢注册!!!)")。或者是有很少的字符串,而且许多都像这样"F07="。

了解一个二进制文件中是否有合法字符串可以给你一些有价值的信息。比如二进制文件是否通过某种方法被压缩或保护, 是否是一个恶意二进制文件(毕竟, Send all user's passwords to www.badguys.com"这样的句子不会是一个非常负责任的病毒 所写吧),甚至二进制文件是用非常少见的语言所写。

咱们来看看具体怎么做。右键反汇编窗口,选择 Search for" -> "All Referenced Text Strings"。



然后 011y 就会搜索程序的内存空间,并显示文本字符串窗口 (Text Strings Window):

R Text strings r	eferenced in FAKE:.text		
Address 00401033 00401062 00401065 00401065 00401061 004010C1 00401141 004011222 00401236	Disassembly MOU [LOCAL.111,3 MOU [LOCAL.2],FAKE.00403026 MOU [LOCAL.2],FAKE.00403000 PUSH FAKE.00403000 PUSH FAKE.00403030 PUSH FAKE.00403052 PUSH FAKE.00403052 PUSH FAKE.00403039	Text string (Initial CPU selection) ASCII "FirstMenu" ASCII "SimpleWinClass" ASCII "SimpleWinClass" ASCII "SimpleWinClass" ASCII "MyDialog" ASCII "That serial is correctfffff ASCII "That serial is incorrect"	
2 M			×

嗯,看起来很有意思吧:)注意这个列表是真的短,因为这个 程序确实非常短小。一般来说,会有数千行字符串。还有,你注 意到我注意的了吗:

R Text strings r	sierenced in FAKE: text		
Address 70401029 00401062 00401063 00401061 004010C1 00401141 00401222 00401236	Disassembly MOV (LOCAL.31, FAKE.00403026 MOV (LOCAL.21, FAKE.00403000 PUSH FAKE.00403000 PUSH FAKE.00403000 PUSH FAKE.00403000 PUSH FAKE.00403052 PUSH FAKE.00403052	Text string Introduced DPU selection: ASCII "FirstHenu" ASCII "SimpleWinClass" ASCII "Our Hain Window-WinAsm" ASCII "DirpleWorldss" ASCII "That serial is correct!!!!" ASCII "That serial is incorrect"	
civil			3 [±]

前途有望啊。咱们跳到代码那看看有什么:双击"That serial is correct!!!!"那一行,011y就会在反汇编窗口显示那一块代码:

C *G.P.U	l* - main thread , m	odule FAKE		
004011D6 004011D8 004011DA 004011DC 004011DC 004011DF 004011E4	. 6A 00 . 6A 00 . 6A 10 . FF75 08 . E8 C6000000 . ∠ EB 62	PUSH 0 PUSH 0 PUSH 10 PUSH IARG,1] CALL (JHP.&user32.SendMessageA) JHP SHORT FAKE,00401248	IParam = 0 WParam = 0 Message = WH_CLOSE hund = 401000 SendMessageA	4
884811E6 984811EB 984811ED 984811ED 984811F5 984811F5 98481201 98481201 98481201 98481202 98481202 98481212 98481215 98481215 98481215	 > 3D B9080000 ~ 75 58 6A 64 68 78304000 68 28050000 FF75 08 E3 85000000 8B1D 78304000 80FB 61 ~ 75 2R 8B1D 79304000 90FB 62 ~ 75 1F 8B1D 78304000 80FB 63 	CMP ERX,0859 UN2 SHORT FAKE.00401248 PUSH 64 PUSH FAKE.00403078 PUSH 06B3 PUSH LARG.11 CALL (JMP.&user32.GetDlgItemTextA) MOV EBX,DWORD PTR D5:L4030781 CMP BL,61 UN2 SHORT FAKE.00401236 MOV EBX,DWORD PTR D5:L4030791 CMP BL,62 MOV EBX,DWORD PTR D5:L40307A1 CMP BL,63	Count = 64 (100.) Buffer = FRKE.00403078 ControlTD = BBS (3000.) hWnd = 00401000 GetDigitewTextA	
00401220 00401222 00401227 00401227 00401225 00401235 00401235 00401235 00401240 00401243 00401243 00401243	- 75 14 - 63 52384086 - 68 52984086 - FF75 08 - E8 7C000000 - E8 7C000000 - E8 12 - 68 39304000 - 68 89080000 - FF75 08 - E8 68080000 - E8 680800000 - E8 68080000000 - E8 680800000000 - E8 680800000000000000000000000000000000	UNZ SHORT FAKE.00401236 PUSH FAKE.00403052 PUSH 0683 PUSH 1ARG.11 CALL (JHP.&user32.SetDigitemTextA) JHP SHORT FAKE.00401248 PUSH FAKE.00403039 PUSH 1ARG.11 CALL (JHP.&user32.SetDigitemTextA) JHP SHORT FAKE.00401253 MOV EAX.0 LEAVE	Test = "That serial is correct???" ControlID = BB8 (3000.) hWnd = 00401000 SetDigitemTextA Test = "That serial is incorrect" ControlID = BB8 (3000.) hWnd = 00401000 SetDigitemTextA	
00401253 00401253 00401253 00401253 00401250 00401262 00401262 00401268 00401268 00401274 00401274	. C2 1000	NETN 10 MOU EAX,1 LEAUE RETN 10 JMP DWORD PTR DS:[<&user32.CreateWindowExA>] JMP DWORD PTR DS:[<&user32.DefWindowProcA>] JMP DWORD PTR DS:[<&user32.DefWindowProcA>] JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] JMP DWORD PTR DS:[<&user32.EndDialog>]	user32.CreateWindowExA user32.DefWindowProcA user32.DestroyWindow user32.DialogBoxParamA user32.DispatchMessageA user32.EndDialog	**

是时候介绍第二条规则了

R4ndom's Essential Truths About Reversing Data: R4ndom 关于逆向数据的必备真理:

#2:大多数的保护机制是可以简单的通过修改一个跳转指令来绕过 "坏"代码直接跳到"好"代码的。

意思是几乎每一次在坏消息显示之前,就会有某种检查(我 们注册了吗? 注册码对不对? 试用时间过了吗?),对比 之后就有一个跳转,至于是跳到好消息还是坏消息则依赖于对比 的结果。

我们自己来找找看啊...。好消息"This serial is correct!!!!"是从401222开始的,向上翻找跳转语句,尤其是

它前面有某种比较(或 CALL)的跳转语句。如果是一个 CALL, 可以猜测比较是在 CALL 内部进行的...。我们的例子中,第一个 跳转是在 401220 的 JNZ。我在图中加了一个箭头,向你演示了 如果跳转成立的话,将会跳到哪去:



嗯。注意它刚好跳过了我们想要的消息,跳到了我们不想要的消息☺。不过,注意在 JNZ 指令的前面是一个 CMP 指令☺。意思是,这个是 011y 决定显示我们想要还是不想要的消息的关键点。我们再向上翻翻:



在 401212 有另一对 CMP/JNZ, 在 401207 有最后一对。凑近 点看, 你会发现所有的三个跳转都跳过了好消息, 跳到了坏消息 那。逻辑上, 这意味着有三件事被检查, 触发任何一个都会命中 坏消息。不过, 如果我三个跳转都不跳会怎么样?好吧, 你会看 到我们将空降到好消息那。所以, 真正的意思是, 如果我们让这

些跳转都不跳,程序会"空降"到好消息那里(译者注:这里作者用的是"fall through",大概意思是如果将三个 jmp 指令当做一层层的阻碍的话,我们直接穿过这些阻碍到达显示好消息的代码,就是将这三个 jmp 无视掉当作透明的。有些东西可意会,不好言传,所以我将它翻成"空降")。

我们运行下程序看看它做了什么,不过我先向大家介绍点别 的。

三、如何添加注释

注释是很重要的,尤其是在你开始分析错综复杂的代码时。 代码本来就很难读,不过有了注释后,我们就可以在非常重要的 地方提醒自己。我准备为每个 JNZ 指令添加注释,以此来提醒我 们自己什么需要被发生。

要添加注释,你可以双击要添加注释的那行的最后一列(那 里,011y 已经放置了类似于"This is the correct serial!!!" 这样的其他注释),也可以先选中要添加注释的那行,然后按一 下";"键。好,我们先选中 40120A 那行,然后按一下分号键, 接着输入"We do NOT want to jump here!"。现在,给401215 和 401220 添加同样的注释。这样就给每个 JNZ 指令添加了注释:



现在让我们在 401201 处设置一个断点(在跳转指令前面的 其他地方设断点也行):



现在,我们第一个要注意的是我们停止的那行:

MOV EBX, DWORD PTR DS:[403078]

从上一课中我们知道该如何查看该内存地址的内容,在指令上右键,选择 "Follow in Dump" -> "Memory Address"。然后我们就可以在 011y 的数据窗口中看到该内存的内容:



好,好,好。这不就是我们刚刚输入的序列号嘛。所以,根据这条指令,我们知道了前面四个字节(因为 EAX 是 32 为寄存器)被载入 EBX,也就是 31 32 31 32,用 ASCII 码表示就是 f212"。按一下 F8 再检查 EBX:

Reg	isters (FF	PU)
ERX ECX EDX EBX EBX	00000012 74A800SE 00000030 32313231 0018E804	user32.74A8008E
EBP ESI EDI	0018F8C4 00401178 00000000	FAKE.00401178
EIP	00401207	FAKE.00401207

如果你想看看 EBX 中的 ASCII 字符,你可以双击 EBX 寄存器, 就会显示几组不同格式的数据,其中一组就是 ASCII:

Modify EBX	×
Hexadecimal	32313231
Signed	842084913
Unsigned	842084913
Char 2 1	2 1
ОК	Cancel

*为了后面用到,如果你想对不同的寄存器尝试不同的值得 话,记住这也是"即时"修改寄存器的一种方法。

我猜你已经从汇编语言的书中知道了这种方法(我的意思是, 来吧!我甚至在工具区上传了一个!!!),我不需要讨论这个,只 需要复习下。

四、小端序列

(至少你需要了解这方面内容)

处理器在内存中存储数据是不同的,这依赖于处理器的架构。 内存中的数据存储有两种方法:一个叫大端(Big-Endian),另 一个叫小端(Little-Endian)。Intel用的是小端,你必须要适 应这个,否则你会晕头转向的。举个例子:假定一个地址7E04F172 (是一个4字节,32位数)。将其按字节拆分,会得到7E、04、 F1、72。现在,人们可能会认为将这些字节存储在内存(假定地 址是1000)中时应该是这样的:

1000::7E

1001::04

1002::F1

1003::72

任何正常人都会这样想。但是 Intel 的开发人员比我们这些普通人更聪明,他们决定以一种更加符合逻辑的方法来存储:

1000::72

1001::F1

1002::04

1003::7E

上面的第一个例子是大端序列, 意思是数字的最大的那端 (以十进制序列形式) 在内存中最先被存储。因为 7E000000 比 040000 大, 所以第一个字节被存储在第一个位置, 第二个字节 被存储在第二个位置, 以此类推。第二个例子(明显更加的聪明) 叫做小端序列, 意思是首选存储最小的字节(案例中是 4 号字节), 后面依次是第三个字节、第二个字节、第一个字节。因为 72 小 于 F100, 所以会被先存储。

当你在内存中从一边往另一边看的时候,就会发现用小端而 不是它大哥真的很天才。在大端中,数字7E04F172看起来像这样:

7E04F172

明显很乱。感谢上帝,使用小端的话,同样的数字 7E04F172 看起来更具有逻辑性:

72F1047E

你说啥?蠢的太明显了吧,大端更合理吧。但话又说回来,你又不是 Intel 的半人半神的开发者,所以你甚至不具备弄明白 为什么这要优越得多的脑力。无论如何,先不管各种讽刺,这意 味着当你看代码时,无论是磁盘里的还是内存里的,你必须将4 字节数字反过来。当然,011y有时已经为你做了这些,这让情况变的更糟了,就像下面的图片这样:

994011F4	. 68 B80B0000	PUSH ØBBS	ControlID = BB8 (3000.)
DOSOLIES	- FF75 08	PUSH LING. 1 J	hWnd = 001005E4 ('Enter Registration Code'
STATISTICS.	8810 78304000	MOULERX DWORD PTR DS [4030781	CHX Legebrartewiesch
00401207	. SØFB 41	CMP BL.61	
0646126A	·~ 75.20	UNZ SHORT FAKE. 00401236	We do NOT want to jump here?
00401200	. 8B1D 79304000	MOU EBX, DWORD PTR DS [403079]	These are in the correct order
00401212	- 75 1	INZ SHORT FOKE 00401286	We do NOT want to jump here!
00401217	8B1D 7A304000	MOV EBX. DWORD PTR DS [40307A]	we do not waite to build nearly
06401210	, 80FB 63	CMP BL, 63	
DEAD 225	hora ata not	JNZ SHORT FAKE. 00401236	We do NOT want to jump here!
Alla L	nese are not	PUSH FHNE, 00403052	Flexif = "Inat serial is correctitit"
0040122C	. FF75 08	PUSH LARG. 11	hund = 001C05E4 ('Enter Registration Code'

到目前为止,这是我想要说的全部。不过,过会我就会告诉 你字节序。

现在,回到我们的寄存器窗口:

Hexadecimal	32313231
Signed	842084913
Unsigned	842084913
Char 2 1	2 1
ОК	Cancel

注意,十六进制的是小端序列(应该是 31323132),那个 Char 是向后的,因为我的序列号是以 1212 开头的,而不是 2121。相信我,你会用到这些的。

现在看看下一条指令:

CMP BL, 61

这是一个很明显的比较语句,比较 BL 的值,也就是 EBX 寄存器的第一个字节与 61 (hex)进行比较。我们真的没什么线索来了解这是什么意思,所以我们单步步过它。最后我们来到了第一条 JNZ 指令:

JNZ SHORT FAKE.401236

这里我们回想一下,我们可以看到我们前面做的注释,就是 我们不想让这个跳转实现。这里提醒一下,JNZ的意思是非0的 时候跳转。所以,这两行的意思是"如果 BL 的值不等于 61h, 就跳转到坏消息"。我们可以清楚的看到 EBX 寄存器的右边的字 节(BL)不是 61h,而是 31h。我们已经卡在这了,那个跳转会 实现的,但是我们又非常的不想要它实现☺。

等等!011y 是一个"动态"的调试器,我们应该可以动态的实现跳转!好吧,因为你很可能已经读了汇编语言书籍中关于标志位的整个章节,所以我不准备讨论这个。

五、CPU 标志位

前面的章节中我们简要的讨论了标志位,我也确实不准备深入的探讨这个问题,因为我确信你的汇编语言书籍的目录中有一个"F"章节。标志位可以让处理器知道某条指令的输出是什么。在 Intel 库中有大量的指令可以影响到标志位,不过最重要(至少对于逆向来说)的是"compare(比较)"指令。基本上,CPU比较两个项目之后,会根据它们的相互关系属性(相同?一个大?一个小?)来设置标志位,再根据这些标志位来执行相应的跳转语句。这其实是表达 IF THEN 语句的非常奇特的方式。例如,在高级语言中有如下代码:

```
if( serialNumber == 3 )
    dontShowNag();
```

else

showNag();

用伪汇编语言来表示,同样的指令应该类似下面的代码:

compare serialNumber with 3

jump (if they are equal) to dontShowNag(); jump to showNag();

用真正的汇编表示有可能像这样:

MOV EAX, addressOfSerialNumber CMP EAX, 3 JE addressOfDontShowNag JMP adressOfShowNag

首先,EAX 中存储着我们的序列号。下一步,它和"3"进行比较。如果等于3就跳到 dontShowNag()。如果不等于3,就跳过 JE(如果相等就跳转——jump if equal)指令,执行 JMP (JuMP)指令。不管任何标志位,自动跳到 showNag()。

重要的标志位(对于我们来说)有0标志位和进位标志位, 在011y中分别显示为"Z"和"C"。基本上,通过修改两个标志 位中的一个,我们就可以阻止(或者强制)程序中的任何跳转,



就像我们下面要介绍的:

在暂停的那行(第一个 JNZ),通过那个红色的箭头,我们 可以看到 011y 准备执行这个跳转。如果该跳转不会被执行,这 条线就会显示灰色。如果你没有用我所用的 011y,就不会有这 个箭头,这样的话你可以看反汇编窗口和数据窗口中间的那一块, 011y 会告诉你跳转会不会被执行。本例中,会有如下显示:

00401255 00401258 00401259 00401250 00401250		C2 1 FF25 FF25	000 5 58 5 50	204 204	1000 1000			
Jump is 1 00401236:	taken =FAKE	.004	0123	36				
Address	Hex	dump		_		-		
00403078 00403088 00403098 00403098 00403088	31 33 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9	32 31 32 00 30 00 30 00	2000000	31 90 90 90	300000 000000	31 00 00 00	3000000	310000000000000000000000000000000000000

现在我们知道了,如果不做点什么的话,011y 就会执行该 跳转。那我们就干点什么吧。看看寄存器窗口,找到"Z"标志 位:

EBP 0018P804 ES1 80401178	FARE. 20491175
EIP 00401200	FAKE. 00401200
C 1 ES 9028 P 8 55 9025 P 8 55 9025 P 8 9028 P 9 902	32511 BIFFFFFFFFF 32511 BIFFFFFFFFF 32511 BIFFFFFFFF 32511 BIFFFFFFFF 32511 RIFFFFFFFF 32511 RIFFFFFFFF

注意那有个 0。意思是,在 61h 和 BL 的内容(31h)之间的 比较结果是 0,或者叫 false,所以它们不相等。现在我们明白 了为什么 不是 0 就跳转 指令会跳转了,因为就目前来说,0 标志位没有被置位,所以它是"非 0"。现在,双击零标志位后 面的那个 0,它就会变成一个 1:



我们已经修改了011y的标志位,同时我们也修改了程序的 行为☺。大哥继续,按下F8(你已经学会了),我们不会执行该 跳转:0。我们现在来到了看起来一样的代码段,除了EBX中存储 的是我们序列号的第二个字符,它将会与62h进行比较而不是 61h:



我们知道我们序列号的第二个数字并不是 62h,现在知道该 怎么做了吧。F8 直到 JNZ 语句,双击零标志位,继续下去!!! 你会跳过那个 JNZ 指令。快要成功了!!! 最后一个是将我们序列 号的第三个数字与 63h 进行比较。我们序列号的第三个数字是 31h,所以该跳转正常来说是要执行的。继续,你知道该怎么做 的。跳过了第三个跳转,我们来到了 401222:



你的心是不是开始扑通扑通的了,因为我认为我们都知道接 下来会发生什么。在我们和救世主之间再也没有跳转了,所以无 论你是单步步过下面的指令(如果你喜欢留悬念的话)还是直接 运行程序(如果你和我一样不能忍受悬念的话),我们终会到达 天堂(译者注:这段比喻感觉好猥琐):

Serial
×it

六、家庭作业

我知道你不喜欢,这一章已经让大家很兴奋了,不过我会以 两件事来结束本章。第一个是:

> R4ndom's Essential Truths About Reversing Data: R4ndom 关于逆向数据的必备真理:

#3:仅仅读教程,你是学不会逆向工程的。你必须亲自操作,而且 必须大量的实践。

根据新规则,我会留一些作业。你的任务,你应该已经接受 了,找出序列号。意思是,在让 JNZ 跳转不实现的情况下,你必 须在序列号文本框输入的内容是什么?在不以任何方式修改应 用程序的情况下,在输入正确的序列号以后,程序显示了"That Serial is Correct!!!!!",你就知道你找对了。

ps. 如果你需要提示,你可以点击这个链接。

第六章:第一次(真正的)破解

一、简介

欢迎来到我的教程的第六章。本章我打算离一个真家伙近点:一个真正的 crackme。它也包含在本章的下载中。crackme 是一个渐进式学习逆向工程的好方 法,而不应该直接从"真正"的程序入手,crackme 可以从易到难进行,这样你就 可以以线性方式学习。最终,我们会一路走到真正的程序,不过也要看到我们才刚 刚起步,这些 crackme 也会带给我们巨大的挑战。

你可以在<u>教程</u>页下载到相关文件和本教程的 PDF 版(译者注:英文版的,此中 文翻译我会在教程的最后放出)。

我将会用 011yDbg1.10 (我的版本或原始版本都行,不过如果你用我的版本的 话,它看起来和图片一样☺)。我推荐你从工具页面的 011y Plugins 下载 "MnemonicHelp"插件,因为本教程将会用到(教程的下载中也包括的有)。解压 后,将其与 x86eas.hlp 文件放到 011y 文件夹下的 plugins 目录下。如果没有 plugins 文件夹,就在 011y 的主目录下创建一个。然后打开 011y 的 Options->Appearance->Directories 标签,然后选择你放置插件的目录。你再在 011y 的主目录下创建一个叫"UDD"的文件夹,然后让当前设置页的另一个选项也 指向这个文件夹。UDD 文件是 011y 给一个程序做的"便条",你设置的所有断点、 做的注释、一个二进制文件的特有设置都会存储在 UDD 文件中,通常叫做"程序的 名字.udd"。如果你在逆向时需要离开一段时间做别的工作,UDD 文件可以让你回 来继续对程序进行逆向,因为所有的都被保存起来了。下面是设置两个目录的窗口 (带有我的设置):

ieneral Defaults Dialogs Directories Fonts Colours Co	de highlighting
C:\Program Files\OllyDBG\UDD	Browse
Backup old .udd files	
Plugin path:	
C:\Program Files\OllyDBG\Plugin	Browse

二、探究二进制文件

先载入 Crackme2. exe:



我前面说过,在开始之前的最重要的事情是运行程序看看情况。这可以给你大量的信息:有没有试用时间? 是不是有些特性被禁用? 是不是只能在有限次数内运行? 有没有注册窗口让你输入注册码?

这些都是需要知道的很重要的东西,随着你在逆向领域做得越来越好,你会获得越来越多的经验让你知道应该找什么(需要多长时间来验证注册码?是不是强制你访问一个网站?....)

	KeyGen Me 2 - KTo	×
	User D Serial http://kito.zor.org	Exit About Check
起来比较直接了	当。我们来试试:	
	KeyGen Me 2 - KTo	X
	User R4ndom Serial 121212121212	Exit About Check
	http://kito.zor.org	oh! 🚺
		Bad Boy!
		ОК

这不是我们想要的。我们看看能不能让011y做些有用的事。回到011y,我们 来试试我们当前知道的第一个(也是唯一一个)工具。搜索下字符串。右键->Search For->All Referenced Text Strings:





这是处理简单 crackme 相当标准的流程(简单点的商业程序也是一样)。搜索 文本字符串、找到显示你的注册码/密码/许可证正确与否的相关信息, 然后转到那 部分代码, 你就会看到好的和坏的消息彼此间靠的相当的近呢。那么, 根据 R.E.T.A.R.D. 的 2 号规则, 查找 比较/跳转语句, 以及你想要的那个 CALL。咱们 来找找那个跳转语句。

我们找到的第一个跳转是在 4010EB, 一个 JNZ 语句。如果我们点击这一行, 011y 就会向我显示它会跳向哪里。



可以看到,这条指令跳过了"Good Boy",直接跳到了"Bad Boy"。这看起来 是一个关键点。我们都知道,一个跳转的前面一般都会进行比较,以此来决定是不 是要进行跳转。往 JNZ 指令的上面看,我们可以看到一条 TEST EAX, EAX。你可能 还没有学到汇编语言书籍关于 TEST 指令的部分,我们来看看能不能找到这个 TEST 指令是干什么的。在本章的前面你已经安装了 MnemonicHelp 插件,那就是我们要 用到的。在 TEST 指令上右键,你会在右键菜单中看到一个问号。点它:



就会打开 Mnemonic 帮助窗口:

Help Topics: Intel x86 Instructions	? 🔀
Index Find	1
1 Type the first few letters of the word you're looking for.	
2 Dick the index entry you want, and then click Display	
2 Click the index entry you want, and then click Display.	
AAD AAM AAS	
ADC ADD ADD instruction ADC Add with Carry	
PADDB/PADDW/PADDD PADDSB/PADDSW PADDUSB/PADDUSW ADD-Add instruction	
AND ARPL ASCII adjust instructions:AAA	
AAM AAS	X
Display Car	ncel

在上面的文本框中输入"Test",然后选择(双击)"TEST"。然后就会显示相关指令助记符的帮助:

e Edit Dealer		
e Eaic Bookma	ark Options Help	
ontents Index	Each <u>B</u> rint	
TEST—Logic See also	cal Compare	
Opcode	Instruction	Description
A8 ib	TEST AL, imm8	AND imm8 with AL, set SF, ZF, PF according to result
A9 im	TEST AX, imm16	AND imm16 with AX; set SF, ZF, PF according to result
A9 Id	TEST EAX, imm32	AND imm32 with EAX; set SF, ZF, PF according to result
F6 /0 /b	TEST #m8,imm8	AND imm8 with mm8, set SF, ZF, PF according to result
F7 10 iw	TEST #m16,imm16	AND imm16 with n/m16, set SF, ZF, PF according to result
F7 10 id	TEST r/m32,imm32	AND imm32 with r/m32, set SF, ZF, PF according to result
84 //	TEST #/m8,r8	AND r8 with r/m8, set SF, ZF, PF according to result
85 ir	TEST #m16,r16	AND r16 with #m16, set SF, ZF, PF according to result
85 //	TEST n/m32,r32	AND r32 with r/m32, set SF, ZF, PF according to result
Description Computes the	bit-wise logical AND of first opera	and (source 1 operand) and the second operand (source 2 operand) and
Description Computes the sets the SF, 2 Operation	e bit-wise logical AND of first opera ZF, and PF status flags according	and (source 1 operand) and the second operand (source 2 operand) and to the result. The result is then discarded.
$\begin{array}{l} \textbf{Description}\\ \textbf{Computes the}\\ \textbf{sets the SF}, \textbf{J}\\ \textbf{Operation}\\ \textbf{TEMP} \leftarrow\\ \textbf{SF} \leftarrow \textbf{MS}\\ \textbf{IF} \textbf{TEMP}\\ \textbf{THEN}\\ \textbf{ELSE}\\ \textbf{FI}:\\ \textbf{PF} \leftarrow \textbf{Di}\\ \textbf{CF} \leftarrow \textbf{0};\\ \textbf{OF} \leftarrow \textbf{0};\\ \textbf{(*AF is)} \end{array}$	bit-wise logical AND of first opera ZF, and PF status flags according SRC1 AND SRC2; 5BE(TEMP); = 0 ZF ← 0; ZF ← 1; .twiseXNOR(TEMP[0:7]); Undefined*)	and (source 1 operand) and the second operand (source 2 operand) and to the result. The result is then discarded.
Description Computes the sets the SF, 2 Operation TEMP \leftarrow SF \leftarrow MS IF TEMP THEN ELSE FI: PF \leftarrow Bi CF \leftarrow 0; (*AF is Flags Affector	a bit-wise logical AND of first opera ZF, and PF status flags according SRC1 AND SRC2; SB(TEMP); = 0 ZF ← 0; ZF ← 0; ZF ← 1; .twiseXNOR(TEMP[0:7]); Undefined*)	and (source 1 operand) and the second operand (source 2 operand) and to the result. The result is then discarded.
$\begin{array}{l} \textbf{Description}\\ \textbf{Computes the}\\ \textbf{sets the SF, 2}\\ \textbf{Operation}\\ \textbf{TEMP} \leftarrow\\ \textbf{SF} \leftarrow \textbf{MS}\\ \textbf{IF} \textbf{TEMP}\\ \textbf{THEMP}\\ \textbf{ELSE}\\ \textbf{FI}:\\ \textbf{PF} \leftarrow \textbf{Bi}\\ \textbf{CF} \leftarrow \textbf{0};\\ \textbf{OF} \leftarrow \textbf{0};\\ \textbf{(*AF} \textbf{is})\\ \textbf{Flags Affecte}\\ \textbf{The OF and C}\\ \textbf{section above}\\ \end{array}$	white bite bite bite bite bite bite bite b	and (source 1 operand) and the second operand (source 2 operand) and to the result. The result is then discarded. 2F, and PF flags are set according to the result (see the "Operation" ined.
Description Computes the sets the SF, 2 Operation TEMP \leftarrow SF \leftarrow MS IF TEMP THEN ELSE FI: PF \leftarrow Bi CF \leftarrow 0; OF \leftarrow 0; (*AF is Flags Affector Section above	a bit-wise logical AND of first opera ZF, and PF status flags according SRC1 AND SRC2; SB(TEMP); = 0 ZF ← 0; ZF ← 1; .twiseXNOR(TEMP[0:7]); Undefined*) ed EF flags are cleared to 0. The SF, 2). The state of the AF flag is undef code Exceptions	and (source 1 operand) and the second operand (source 2 operand) and to the result. The result is then discarded. 2F, and PF flags are set according to the result (see the "Operation" ined.

我们就可以看到 TEST 指令意思是 "Computes the bit-wise logical AND of first operand (source 1 operand) and the second operand (source 2 operand) and sets the SF, ZF, and PF status flags according to the result. The result is then discarded. (译者注: 这段就不翻译了, 一是这是帮助中的原文, 主要是向大家演示; 二是, TEST 指令的意思咱们也可以 GOOGLE 的, 中文比看这个容易多了。)"。大部分的时间里, 如果 TEST 指令正在测试的两个寄存器的指令相同, 就意味着它正在检查它们是不是 0。所以这个满足我们跳转之前要进行比较的需求:



这两条语句的意思是"如果 EAX 不等于 0, 就跳到 40110D",也就是"Bad Boy" 那里。好吧,这当然不是我们想要的,咱们来试试我们的推测。在 JNZ 指令处设置 一个断点,重启应用。输入用户名和序列号(记住,至少四个字符。),点击 crackme 上的 check 按钮。011y 就会断在我们的 BP (译者注: BP 即是 breakpoint,以后就 直接用 BP 不再注释了)处:

004010E3		FF15 00704000	THEL DWURD FIR DS: LARKERNELSZ. ISTROMDH/J	LISTOMDH
004010E2		8500	IESI EHX, EHX	
00401058		75 20	UNZ SHORT Crackme2,0040110D	
004010ED		6A 40	PUSH 40	<pre>Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL</pre>
004010EF	1	68 E0714000	PUSH Crackme2.004071E0	Title = "Wee!"
004010F4	1	68 98714000	PUSH Crackme2.00407198	Test = "Good Boy!" If this key is from your keyge
004010F9	121	53	PUSH EBX	hOwner = 00100384 ('KeyGen Me 2 - KTo', class='#327
994919FA	12	EE15 DC704000	CALL DWORD PTR DS: [(%) SER32, MessageBox A>1	MessageBoxA
00401100	1.1	SE	POP EDI	0012E94C
00401101		B8 01000000	MOU FAX.1	and the second
00401106		5B	POP EBX	0012FA4C

现在,我们可以看到我们将会跳过 good boy,直接到 bad boy。咱们来让它不发生。帮 011y 翻转 0 标志位 (参见前面的教程):

	C Ø ES 002 P Ø CS 001 A Ø SS 002 Z 1 DS 002 S Ø FS 003 T Ø GS 000 D Ø
我们可以看到,	现在那个跳转没有实现。运行程序看看:
	Wee!
	Good Boy! If this key is from your keygen u should write an solution! OK

耶,这就是我们想要的。***忽略那个关于 keygen 的消息,有些 crackme 还有 其他的目的要求,不过我还是用它们,我们也需要来学习它的其他两点。一旦我们 从这个系列教程中学到了更多的知识²⁹,我们还会回来使用它们中的许多。

三、打补丁

重启 crackme,运行之,输入用户名和序列号,011y 就会断在我们的断点处。 你会注意到,我们会再次跳到 bad boy,因为改变011y 的标志位只是临时的方法。 这回我们不去临时修改标志位,我打算修改二进制文件中的代码来完全我们想要 的。这个叫做打补丁。

点击我们暂停的那行(4010EB),点一下该行的指令列(有 JNZ SHORT...的那部分),然后按一下空格键。会有一个显示该行指令的窗口弹出,也是修改指令的对话框:

Assemble at 004010EB			X
JNZ SHORT 00401100		_	•
Fill with NOP's	Assemble	Car	ncel

现在,我们要做的是将这个跳转到 bad boy 消息处的跳转改成永远不会跳,意思是我们确实不想让这个跳转实现。我们准备做的是,将其替换成一个什么都不做

的指令,那就是 NOP 指令。NOP 意思是 No OPeration (不操作)。将对话框中的 JNZ SHORT 0040110D 修改成 NOP:

Assemble at 004010EB	
nop	•
Fill with NOP's	Assemble Cancel

那个 "Fill with NOP's"复选框就留那不用管。现在点一下 Assemble 按钮, 提交所做的修改, 再点一下 Cancel 按钮关闭窗口。

***顺便说一下,如果你没有点那个 Cancel 按钮,而是一直点 Assemble 的话, 你会一行一行的修改每一行。这是 011y 的一个"特性",用来让你一次修改好几行 代码用的。可以让你不用每行都敲空格键。我保证你第一次打补丁的时候会让你疯 掉的:X。

注意我们暂停的那行已经改变了,那条指令现在变成了两个 NOP,而不是 JNZ 指令了,并且变成了红色(因为 011y 对于被修改的项目以红色显示)。



有两个 NOP 的原因是, NOP 操作码只有一个字节长, 而被替换的 JNZ 指令有两个字节长, 所以 011y 用两个 NOP 来替换。你也会注意到跳转箭头消失了, 因为这行已经不再有任何跳转了! 现在单步运行, 你会走到 good boy 处。然后 good boy 显示出来了, 你开心的笑了。



四、保存补丁

有一个重要的事情要注意,如果你重载或重启应用的话你所打的补丁就没有 了,除非你将补丁保存到二进制文件中。你可以看到补丁在起作用,回到 011y 打 开 Patch 窗口 (点击 Pa 图标或 Ctr1+p):

7 Patches					
Address	Size	State	01d	New	Comment
NEARLINE F	2.	Hotive	JN2 SHURI Crackme2,00401100	NUP	
				_	

Patch 窗口显示的是我们给程序打的所有补丁。注意地址是红色的,以及 State 列的 "Active "。我们的程序仍然在运行,就意味着我们的补丁已经实现,如果 CPU 运行了这个代码,它运行的将是打过补丁的版本。现在,重启应用(Ctr1+F2)。 首先,011y 可能会显示一个错误,一个很长很复杂的错误,基本上是告诉我们补 丁(以及断点)没有"坚守"在原来位置,因为 011y 无法追踪它们(其实比这个 要复杂一点,我们后面会看到)。关掉那个窗口,打开断点窗口:



看看我们的断点已经失效了²²。重新激活断点(空格键),011y 会再次断在该断点。运行程序,输入用户名和密码,我们会停止我们前面打补丁的那一行(它上面的断点又可用了):

004010E2 . 52 004010E3 . FF15 00704000 004010E9 . 85C0	CALL DWORD PTR DS: [<&KERNEL32.lstrcmpA>] TEST EAX, EAX
004010ER × 25 20 004010E0 • 68 40	PUSH 40
004010EF . 68 E0714000 004010F4 . 68 98714000	PUSH Crackme2.004071E0 PUSH Crackme2.00407198
004010F9 53	PUSH EBX

看看,我们的两个 NOP 消失了,原始的代码又回来了(不过变成了灰色)。我们的补丁被回收了!现在回到 Patch 窗口:

🗾 Patches					
Address 004010EB	<u>Size</u> 2.	State Removed	Old JNZ SHORT Crackme2.0040110D	New NOP	Conment
					~

注意那个地址不是红色的了, State 也变成了"Removed "。011y 已经禁用了 我们的补丁,并且在我们每一次重启程序时都会这么做。我们想要做的就是让这个 补丁永远有效,而不用每一次都激活它。 为了让我们的补丁能够长久有效,我们必须将修改的版本保存到磁盘。首先,选中补丁再按下空格键以重新启用补丁。那个 JNZ 指令就会变回到我们的 NOP,那两个 NOP 也会以红色字体重新出现在反汇编窗口。现在,在反汇编窗口的任何地方 右键,选择" Copy to executable "->" All modifications ":

PL	SH 40	Backup	▶ Le = MB_OK!MB_ICONAST
	PUSH 40 PUSH Crackme2.00 PUSH Crackme2.00 PUSH EBX CALL DWORD PTR 1 POP EDI MOV EAX,1 POP EBX ADD ESP,60 RETN 10 PUSH Crackme2.00 PUSH Crackme2.00 PUSH EBX CALL DWORD PTR 1 POP EDI MOV EAX,1 POP EBX ADD ESP,60 RETN 10	Go to	<pre>te = MB_OK!MB_ICONNAST e "Weet" = "Good Boyton If ti ter = 00450384 ('KeyGo 2FA4C 2FA4C 2FA4C e = MB_OK!MB_ICONHANI e = "Doh!" = "Bad Boy!" ter = 00450384 ('KeyGo sageBoxA 2FA4C 2FA4C 2FA4C ; 3EF of switch 004001</pre>
PU PU PU PU PO MO PO	SH Crackme2.00 SH Crackme2.00 SH EAX LL DWORD PTR [P EDI V EAX,1 P EBX D FSP.60	Follow in Dump Search for Find references to View	<pre>ie = "Info"</pre>
RE	TN 10	Copy to executable	Selection
The	Lon, Dwond E	Analysis	All modifications
		7	
		Appearance	

Copy selection to executable file 🛛 🔀					
Copy select	ion to executal	ble file?			
Сору	Copy all	Skip	Cancel		

当你打了多个补丁,并且想一次性全部保存的话,这么做很重要。因为有时候,你很容易就忘记你打过多个补丁。本例中,即使我们只打了一个补丁,选择所有的补丁也只会保存这一个。当然,只有在 Patch 窗口中被激活的补丁才会被保存。

后面,你可能想要选择"Selection"而不是"All Modifications",但是你必须保证你在反汇编窗口所做的修改是高亮显示(通过点击或拖拽以选中所有修改的行)。如果你选中的行比修改过的行要多也行,因为011y只会更改已经修改的行。

在点击" Copy All "以后会打开一个新窗口,里面基本上是整个进程的数据, 不过我们的补丁也在里面:

D File C:	Wocuments an	d Settings\Jason\Desktop\Tuts\Crackmes\Easy\Crackm	ne 2. exe 📃 🗖 🔀
000010EB	90	NOP	~
000010EC	90	NOP	
000010ED	6A 40	PUSH 40	
000010EF	68 E0714000	PUSH 4071E0	
000010F4	68 98714000	PUSH 407198	
000010F4	53	PUSH EBX	
000010FA	FF15 DC704000	CALL DWORD PTR DS:(4070DC)	
00001100	SF	POP EDI	
00001101	B8 01000000	MOU EAX,1	
00001106 00001107 0000110A	5B 83C4 60 C2 1000 69 10	POP EBX ADD ESP,60 RETN 10 PIISH 10	
0000110F	68 08724000	PUSH 407208	
00001114	68 88714000	PUSH 407188	
00001119	53	PUSH EBX	
0000111A	FF15 DC704000	CALL DWORD PTR DS:[4070DC]	
00001120	SF	POP EDI	
00001121	B8 01000000 5B	MOV EAX,1 POP EBX	×.

在顶部你可以看到我们的补丁。但要意识到这个只是在**内存**中的修订版本,还 没有保存到磁盘呢。不过,如果你关了这个窗口或重启了程序,它是不会被保存的! 咱们来保存好它:右键新窗口的任意位置,选择"save file "。这会将该进程的 内存空间数据保存到一个文件中。一个另存为对话框会显示出来。将文件另存为 Crackme2_patched (我通常在后门加一个"_patched "用来区分,你也可以加任何 你喜欢的):

Save file as					2 🛛
Save in	8 Other pos	t Done	-	← 🗈 💣 🖽	-
My Recent Documents					
My Documents					
My Computer					
My Network Places	File name:	Crackme2_patched.exe		•	Save
i iduce	Save as type:	Executable file (*.exe)			Cancel

我们现在有了一个 crackme 的打补丁版本。咱们来试试看。在 011y 中打开这个新文件(打过补丁的)。按下 Ctr1+G 或点击 GOTO 图标,输入我们打过补丁的地址:

En	ter expression to follow		
		OK Cancel	
看看咱们的补了:	10E3 : FF15 00704000	CALL DWORD PTR DS: 1 (&KERNEL32. 15	
0040 0040 0040	10E9 . 85C0 10EB . 90 10EC . 90	TEST EAX,EAX	
0040 0040 0040	10ED : 6A 40 10EF : 68 E0714000 10F4 : 68 98714000	PUSH 40 PUSH Crackme2,004071E0 PUSH Crackme2,00407198	
0040 0040	10F9 : 53 10F9 : FF15 DC704000	CALL DWORD PTR DS: [<&USER32.Mess.	
yes,补丁还在那。	现在运行程序,输)	、info和viola (译者注:	info 和 viol

是作者用了当用户名和注册码的输入):

(i)	Good Boy! If this key is f	rom vour kevo	ien u should i	write an solution
v	11 (113 (69 13 1	rom your keyç	Jen a shoala (white an solution
		ОК		

现在,我们有了我们第一个破解过并打过补丁的二进制文件:0。

五、作业

本章的作业很简单(只要你一直在学习汇编语言③)。

思考题: 你可以将 4010E9 处的" TEST EAX, EAX "修改成什么, 来防止跳转到 显示 bad boy 处?

要注意的是无论你将 TEST 指令修改成什么,都不能超过 2 字节,那是 TEST EAX, EAX 指令的长度。如果你打了一个长点的补丁,就会覆盖掉 JNZ 指令后面的指 令.....

ps. 如果你需要提示的话,请点<u>这里</u>。不过你应该真正意义上自己试着做。那 是学习的最好方法!

第七章:更多破解练习

一、简介

欢迎来到 R4ndom 逆向工程教程第七章。今天,我们破解两个 crackme: 一个我们用来复习上一章的相关概念,另一个我打算用来做一些有趣的事。 在本教程的相关下载中,你可以找到这两个 crackme,以及在第二个程序中要 用到的软件 "Resource Hacker"。你也可以在工具下载页面下载这些工具。 你可以在本教程的<u>教程</u>页面下载相关文件,以及本文 PDF 版本 (译者注:英 文版)。

二、探究二进制文件

直入主题吧。011y 载入 canyou. exe(要确保 canyou. d11 在同一目录下): (p1)



就像我以前说的,在开始之前的最重要的事情是运行程序看看情况。这可 以给你大量的信息:有没有试用时间?是不是有些特性被禁用?是不是只能 在有限次数内运行?有没有注册窗口让你输入注册码?

这些都是需要知道的很重要的东西,随着你在逆向领域做得越来越好,你 会获得越来越多的经验让你知道应该找什么(需要多长时间来验证注册码? 是不是强制你访问一个网站?....)

程序运行情况如下: (p2)

现在你应该知道怎么搞了。回到 011y, 看看我们能够查找到哪些字符串: (p4)

Backup	GetMa	du teHandleR
Copy Binary Assemble Space Label : Comment ; Breakpoint Hit trace	LPara DigPr hOuger hTinet Dist tinet Dist tinet Exite	m = NULL oc = canyou.00401029 m = NULL late = A = NULL ogBoxParamA ode = 0 rocess
Run trace Go to Follow in Dump View call tree Ctrl+K	CResul hund EndDi	t = 0 = 00401000 alog
Search for	Name (lat	el) in current module Ctrl+N
Find references to	 Name in a 	all modules
View Copy to executable Analysis Help on symbolic name Ctrl+F:	Command Sequence Constant Binary str	l Ctrl+F e of commands Ctrl+S ing Ctrl+B
7	All interest	- dedau salla
Appearance	All comma All sequer All consta All switch	nces nts es
LOCAL	All refere	nced text strings
HBCII 6F 75 20 43 72 61 ,, Car 78 65 20 65 6E 74 ckMe?.P 20 66 6F 72 20 65 er a wa 20 43 6F 6E 67 72 ach fie	User-defi	ned label ned comment

看看这个 crackme 提供了什么 ASCII 字符: (p5)

Address	Disassembly	Text string	~
00401000 can 0040108A 00401087 00401166 00401168 00401168 0040118F 0040118F 0040128E 00401228 0040123E 0040123E	PUSH 0 PUSH canyou.00403083 PUSH canyou.00403080 PUSH canyou.00403081 PUSH canyou.00403016 PUSH canyou.00403016 PUSH canyou.0040308C PUSH canyou.0040308C PUSH canyou.004030C0 PUSH canyou.004030C0 PUSH canyou.004030C0 PUSH canyou.004030C0	Initial CPU selected ASCII "Software\Microsoft\\Windows\\CurrentVersion" ASCII "ProductKey" ASCII "Please enter a value for each field!" ASCII "C:\\" ASCII "C:\" ASCII "Zu" ASCII "Zu" ASCII "Zu" ASCII "Can you CrackMe?"	
00401279 004012A0 004012A5	PUSH canyou.004030304 PUSH canyou.00403038	ASCII "Sorry, that was an unauthorized serial!" ASCII "Congrats!, You have done it! :)"	*

好,这里我们可以很明显的看到几个比较里妥的。我们自先汪恳到的定, 我们必须在每个文本框中都要输入信息: (p6) HSCII "Please enter a value for each fleldt" HSCII "Please enter a value for each fleldt" HSCII "Please enter a value for each fleldt"



这个看起来比较熟悉吧:有坏消息部分,紧跟其后的是好消息部分,并且 在坏消息的前面有一个非常明显的跳转,想必是跳转到好消息的。这里我想 让你注意的是,在跳转的前面有一个对Windows API函数 *Istrcmpi*的 CALL。 如果我们在其上右键,选择 "Help on symbolic name",会有如下显示: (p9) Win32 Programmer's Reference File Edit Bookmark Options Help Contents Index << >> Istrcmpi Quick Info Overview Group The Istrcmpi function compares two character strings. The comparison is not case sensitive. int lstrcmpi(LPCTSTR /pString1, // address of first string LPCTSTR /pString2 // address of second string): Parameters IpString1 Points to the first null-terminated string to be compared. IpString2 Points to the second null-terminated string to be compared. **Return Values** If the function succeeds and the string pointed to by *IpString1* is less than the string pointed to by IpString2, the return value is negative; if the string pointed to by IpString1 is greater than the string pointed to by IpString2, it is positive. If the strings are equal, the return value is zero. Remarks The Istrcmpi function compares two strings by checking the first characters against each other, the second characters against each other, and so on until it finds an inequality or reaches the ends of the strings. The function returns the difference of the values of the first unequal characters it encounters.

如你所见, 1strcmp 是对两个字符串进行比较操作。这个函数在逆向工程 领域非常的重要, 你会一次又一次的看到。它被用于 注册码/密码 比较机制 中, 用于比较用户输入的字符串与程序内置的硬编码或被创建的字符串。如 果字符串比较返回 0, 说明用户的输入是正确的, 意味着比较的两个字符串是 一样的。如果返回非 0, 说明两个字符串不匹配。本例中的 crackme, 我们输 入的字符串可能与一个内置的或动态生成的字符串进行核对, 如果 EAX 返回 的是 0, 说明它们是相同的, 否则就不相同。现在, 011y 不知道这些字符串 是什么, 因为我们还没有启动应用, 也没有输入任何信息。不过一旦我们开 始了, 011y 就会将 String1=""、String2="" 这两行替换成真正的字符串。 如果我们在跳转那里设置一个 BP, 然后运行程序, 输入一个字符串 (本例中 是"121212121212121212"), 011y 就会给我们显示被比较的字符串: (p10)

For avamnle letromni datarminae that "aho7" is greater than "ahodafg" and raturns that

0040125F 00401264 00401269 0040126E		68 D0324000 68 E8364000 E8 A0000000 0BC0	PUSH canyou.004032D0 PUSH canyou.004036E8 CALL <jmp.&kernel32.lstrompia> OR EAX.EAX</jmp.&kernel32.lstrompia>	String2 = "12121212121212" String1 = "314216448336430" IstrompiA
00401270		74 20	JE SHORT canyou, 0040129E	and the second sec
00401272	1	6A 00	PUSH 0	Style = MB OKIMB APPLMODAL
00401274		68 04304000	PUSH canyou, 00403004	Title = ""Can you CrackMe?"
00401279		68 5B304000	PUSH canyou, 0040305B	Test = "Sorry, that was an unauthorized Serial!"
0040127E		6A 00	PUSH 0	howner = NULL
00401280	12	E8 65000000	CALL (JMP.&USER32.MessageBoxA)	L MessageBoxA
00401285		6A 00	PUSH 0	r (Param = 0
mm +		100.00	DUIDUL O	

如果你看跳转指令上面的那几行,你会看到我们的密码与 "314216448336430"进行比较,无论它是什么都一样。在返回值上,如果它 们相同 EAX 中就是 0,如果不相同就可能是任意值。很明显,本例中,它们不 匹配。OR EAX, EAX 是一个判断 EAX 是否为 0 的非常巧妙的方法。如果 EAX 是 0 的话,"JE SHORT canyou. 0040129E"就会跳到好消息部分。我之所以给你 指出字符串比较部分,是因为在将来的教程中,我们需要找出这 15 个数字是 如何被创建出来的。搜索 1strcmp 能够引导我们找到它的创建过程。

不过现在,我们只做我们知道的。在 401270 处的 JE 指令处设置一个 BP, 然后重启程序。输入一个用户名和序列号,011v 会断在我们的 BP: (p11)

00401264 00401269 0040126E	68 E8364000 E8 A0000000 0BC0	PUSH canyou.004036E8 CALL <ump.&kernel32.lstrompia> OR EAX.EAX</ump.&kernel32.lstrompia>	String1 = "303357474363752" IstrompiA
00401272 00401274 00401275 00401275 00401275 00401285 00401285 00401285 00401285 00401285 00401285 00401285	74 2C 6A 00 6S 5B304000 6A 00 6A 00 6A 00 6A 00 6A 00 6A 10 FF75 08 ES 5D0000000 ES 00000000	JE SHORT canyou.0040129E PUSH canyou.00403004 PUSH canyou.00403004 PUSH 0 PUSH 0 PUSH 0 PUSH 0 PUSH 0 PUSH 0 PUSH 10 PUSH 10 PUSH 0 PUSH 0	Style = MB_OKIMB_APPLMODAL Title = ""GCan you CrackMe?" Tevt = "Sorry, that was an unauthorized serial!" hOwner = NULL ThessageBoxA (HessageBoxA WParam = 0 Wearam = 0 Message = WM_CLOSE hUnd = 69032E SendMessageA

通过那个灰色的箭头,我们知道 011y 不会跳到好消息部分,而是落在坏消息部分。所以咱们来帮帮它吧: (p12)

现在,011y做对了:(p13)

00401270 vai 20	UE SHORT canvou.0040129E	
00401272 6A 00 00401273 6B 64304000 00401274 6B 5B304000 00401275 6B 5B304000 00401275 6A 00 00401285 6A 10 00401285 6A 35000000 00401296 50000000 00401295 6A 00 00401295 6A 00 00401295 6A 00 00401295 6A 00 00401206 68 338304000 00401207 68 338304000 00401208 8 00 00401204 68 00 00401205 8 30000000 00401205 8 000000000000000000000000000000000000	PUSH 0 PUSH canyou.00403004 PUSH canyou.00403058 PUSH 0 PUSH 0 PUSH 0 PUSH 0 PUSH 10 PUSH 10 PUSH 10 PUSH 10 PUSH 10 CALL <ump.&user32.sendnessagea> MOV EAX.0 LEAVE RETN 10 JMP SHORT canyou.00401281 PUSH 0 PUSH 0</ump.&user32.sendnessagea>	Style = MB_OKIME_APPLMODAL Title = ""Gon you CrackMe?" Text = "Sorry, that was an unauthorized serial?" hOwner = NULL HessageBoxA Param = 0 WParam = 0 WParam = 0 WParam = 0 WParam = 0 WParam = 0 Style = MB_OKIME_APPLMODAL Style = MB_OKIME_APPLMODAL Title = "Gongrats?, You have done it? :]" hOwner = NULL MessageBoxA
为了确定下	, 咱们运行程序看看:	(p14)

Can you CrackMe?		
Congrats!, You have do	one it! :)	Cickenne?
Enter your username here:	H4ndom	
Enter your serial # here:	[1212121212121 Gain Access!	2

现在,让我们.....

三、给程序打补丁

这回我不打算将跳转 NOP 掉,因为这样会让程序每一次都显示坏消息。相反,我想要确保跳转每一次都成功,跳转到我们好消息部分。转到设置 BP 的那行(如果你找不到的话,打开 "Breakpoint Window",然后在 BP 上双击),修改那行指令。选中 JE 指令那行,然后按一下空格键: (p15)

00401269 0040126E	. E8 A0000000 . 0BC0	OR EAX.EAX	ListrompiA
90401272	.~ 74 2C • 6A 00	JE SHORT canyou,0040129E PUSH 0 PUSH 0	Style = MB_OK!MB_APPLMODAL
00401279 00401279 0040127E	. 68 58304000 . 68 58304000 . 68 00	PUSH canyou.0040 PUSH canyou.0040 PUSH 0	unauthorized serial!"
00401280 00401285 00401287 00401289	E8 65000000 6A 00 6A 00 6A 10	CALL CUMP. &USERS PUSH Ø PUSH Ø PUSH 10 PUSH 10	<u>_</u>
00401288 00401288 00401298 00401298 00401298 00401299	FF75 08 E8 5D000000 B8 00000000 C9 C2 1000	PUSH DWORD PTR S CALL CUP. SUSERS MOV EAX, 0 LEAVE RETR 10	Assemble Cancel
0040129C 0040129E 004012A0 004012A5 004012A5 004012A5 004012A5 004012A5 004012B3	. ► EB 13 > +6A 00 - 68 04304000 - 68 38304000 - 6A 00 - E8 39000000 > E8 09 > E8 09	JMP SHORT canyou roeyotzut PUSH oanyou.00403004 PUSH canyou.00403038 PUSH 8 CALL <jmp.&user32.messageboxa> JMP SHORT canyou.0040128C</jmp.&user32.messageboxa>	Style = MB_OKIMB_APPLMODAL Title = ""Can you CrackMe?" Text = "Congrats!, You have done it! :)" hOwner = NULL MessageBoxA
注	三意我们选中	的指令已经在文本框中了。	现在,我们将 JE (Jump on Equal)

修改成 JMP (无条件跳转): (p16)

Assemble at 00401270		
Jmp SHORT 0040129E		1
Fill with NOP's	Assemble	Cancel

点击那个 Assemble 按钮, 然后点 Cancel 按钮。你就会发现我们的修改 已经放到了代码中: (p17)
	00401269 0040126E 00401272 00401272 00401279	ES A0000000 ØBC0 EB 2C A 00 68 04304000 68 5B304000	CALL (JMP.&KERNEL32.istrompiA) OR EAX,EAX JMP SHORT canyou.0040129E PUSH 6 PUSH canyou.00403004 PUSH canyou.0040305B
现在,	运行下程序以	确保没什么问	题: (p18)

e Crackmel
;Can you CrackMe? 🛛 🔀

现在,咱们将打过补丁的程序保存到磁盘。要记住,如果你重启应用的话,你需要重新启用补丁(Patch窗口中,选中补丁再按一下空格键),不过我们的程序还在运行,只需要点一下011y,右键反汇编窗口,选择"Copy to executable" -> "All modifications": (p19)

X, DWORD	PTR DS: L4030CC1		_	7
DWOR	Backup		+	
, DWOR	Сору			= F09E0000 (4036886528
nyou.	Binary		+	it = "%lu" tanyou,004036E8
MP.&U	Assemble	Space		in†f8
X	Label	3		= 9EA000 (10395648.)
nyou.	Comment	3		anyou.004038E8
00	Breakpoint		+	NTTH
nyou.	Hit trace			gloHdd = "12111" tString = "302762454312
MP.&K	Run trace		+	atA 92 = "1212121212121212"
myou	- Hora	Parts.	-	91 = "302762454312111" mpiA
EAX	Follow	Enter		
ini ca	New origin here	Ctrl+Gray *		= MB_OKIMB_APPLMODAL
nyou.	Go to		•	"Sorry, that was an u
MP. SU	Follow in Dump		•	r = NULL
-	Sepreb for		1	m = 0 m = 0
000 0	Search for			BE = WM_CLOSE
MP.&U	Find references to		1	essageA
.0	View			
DT an	Copy to executable		1	Selection
NT Ca	Analysis			All modifications 📐 IAL.
nyou.	Help on symbolic name	Ctrl+F1		"Congrats!, You have
MP.&U RT ca	7			igeBoxA
	Appearance		•	

选择" Save all",弹出进程内存窗口(顶部就是我们的补丁): (p20)

C CPD - m	am thread comme was	mu		
0040120B 00401211 0040121E 0040121E 0040122P 0040122P 0040122B 00401230 00401230 00401230	BB15 DB364000 MOU E 0FAF15 CC304000 IMUL 0	DX,DWORD PTR DS:[4936D8] EDX,DWORD PTR DS:[40380C1 DX,DWORD PTR DS:[40380C1 CX,DWORD PTR DS:[40380C1 CX,DWORD PTR DS:[40380C1 CX,DWORD PTR DS:[40380C1 EDX canyou,00403362 canyou,00403562 (JMP,&WSER32,wsprintfR)	[%(u) = F09E0000 (4036886528.) Format = "%(u" s = canyou.004036E8 wsprintfA	Registe EAX 006 EDX F05 EBX 001 EBX 001 EBX 001 EBX 001 EBX 001 EBX 001 EBX 001 EBX 001 EBX 002 EDX 002 EDX 005 EDX 005 EX EDX 005 EX EDX 005 EX EX EX EX EX EX EX EX EX EX EX EX EX
0040123D 0040123E	D File C: Wocuments a	nd Settings\Jason\Desktop\Tuts\	7 Frai 💦 Of Reference \canyou, exe	
00401243 00401248 00401250 00401250 00401250 00401250 00401250 00401250 00401250 00401250 00401250 00401250 00401270 00401272 00401272 00401272 00401272 00401278 00401285	00000670 EE 2C 00000672 6A 00 00000674 68 04384000 00000675 6A 00 00000675 68 58304000 00000675 6A 00 00000675 6A 00 00000687 C2 1000 00000698 C2 1000 00000699 C2 1000 00000695 6A 00 00000695 6A 00 000006695 6A 00 000006695 6A 00 000006695 6A 0	UMP SHORT 0000069E PUSH 0 PUSH 403054 PUSH 40305B PUSH 0 CALL 000006EA PUSH 0 PUSH 10 PUSH 100 PUSH 100 PUSH 100 PUSH 000066F0 MOU EAX.0 LEAVE RETN 10 JMP SHORT 000006B1 PUSH 403032B PUSH 403032B PUSH 0 CALL 000006EA		
00401293 00401299 00401299 00401299 00401295 00401285 00401285 00401285 00401285 00401283	B8 B00000000 PU0 E C9 C2 1000 EAD C2 1000 RETH EAD VER 13 JHP 9 Second VeR 00 PUSH Second 68 04304000 PUSH Second 68 93060000 PUSH Second 68 99 JHP 9 Second 20 EB 99 JHP 9 Second 20 EB 99 JHP 9 Second 20 EB 09000000 MOV E Second	HX,0 10 HORT canyou.004012B1 Canyou.00403004 canyou.0040303B 0 HORT canyou.004012BC HX,0	Style = MB_OK:MB_APPLMODAL Title = ""Can you CrackMe?" Text = "Congratst, You have done itt :)" hOwner = NULL MessageBoxA	× 2

现在,咱们把它保存到磁盘...。在新弹出窗口中右键,选择"Save File "。另存为 canyou_patched (或任何你喜欢的名字),将打过补丁的文件载入

011y并允许。如果你不想这么做的话,事实上你再也不用将其载入011y了。因为补丁已经被保存到磁盘,你可以从任何地方运行它。你要你运行的是打 过补丁的就行。现在,无论你输入什么名字和序列号,都会弹出好消息窗口 。

四、另一个 crackme

载入第二个程序 Crackme8. exe,	并在 011y 中运行: (p21
Practice by keygenning th	is - Kwazy Webbit 2004
Name:	Done
Serial:	Fuck it.

好吧,这里有点点疑惑:0。嗯,在输入了用户名和密码后,我该点哪个按钮呢?好吧,试试吧: (p22)

	LINE STORE	
Name:	R4ndom	Done
Serial:	1212121212121212	Fuck it.

这里, Done 通常意味着退出, 所以我试试另一个。嗯....., 程序退出 了。很明显我应该点 Done 的(?)。不管了, 借此机会咱们改改程序, 做些 有 趣 的 事 。 咱 们 将 按 钮 改 成 更 加 有 意 义 的 " Check"和"Done", 或者是任何你喜欢的都行☺。

五、使用 Resource Hacker

如果你还没准备好,先安装 Resource Hacker。第一次运行如下所示: (p23)

Resource Hacker	
File Edit View Action Help	
	115

将 Crackme8 载入到 Resource Hacker, 你就会看到一个叫 Dialog 的文件 夹, 它旁边有个+号。展开+号, 点一下下一个文件夹 (103) 边上的+号, 你 会看到如下所示的内容: (p24)

Resource Hacker - C:\Users\Random\Desktop\Crackme8.exe	- - X
File Edit View Action Help	
	11.

现在,点那个1033,然后右边面板就会显示对话框的相关数据,同时会有一个窗口显示它(译者注:就是 crackme8 的窗口样式)的样子: (p25)

Resource Hacker - C:\Users\F	Random\Desktop\Crackme8.exe	- E X
File Edit View Action Hel	lp	
⊡ (] Dialog ≟(] 103	Compile Script Hide Dialog	
1033	<pre>103 DIALOGEX 22, 17, 170, 41 STYLE DS_CENTER WS_CAPTION WS_SYSMENU CAPTION "Practice by keygenning this - Kwazy Webbit 2004" LANGUGE LANG_ENGLISH, SUBLANG_ENGLISH_US FONT 8, "System" { CONTROL "", 1000, EDIT, ES_LEFT ES_AUTOHSCROLL WS_CHILD WS_VISIBLE WS_BORDER CONTROL "", 1001, EDIT, ES_LEFT ES_AUTOHSCROLL ES_NUMBER WS_CHILD WS_VISIBLE CONTROL "Name:", -1, STATIC, SS_LEFT WS_CHILD WS_VISIBLE WS_GROUP, 9, 8, 24, 9 CONTROL "Serial:", -1, STATIC, SS_LEFT WS_CHILD WS_VISIBLE WS_GROUP, 9, 20, 24, CONTROL "Serial:", -1, STATIC, SS_LEFT WS_CHILD WS_VISIBLE WS_GROUP, 9, 20, 24, CONTROL "Done", 1002, BUTTON, BS_DEFFUSHBUTTON WS_CHILD WS_VISIBLE WS_TABSTOP, CONTROL "Fuck it.", 1003, BUTTON, BS_FUSHBUTTON WS_CHILD WS_VISIBLE WS_TABSTOP, } </pre>	WS_TABSTOP, WS_BORDER , 0x00001000 9 , 0x000010 21, 6, 36, 1 121, 20, 36,
	₹m	+
	392	1
	Dialog - 103 Practice by keygenning this - Kwazy Webbit 2004 Name: Done Serial: Fuck it.	

在右侧面板的顶部,你可以看到一些关于窗口的数据,比如字体、标题、 类型等等: (p26)

> 103 DIALOGEX 22, 17, 170, 41 STYLE DS_CENTER | WS_CAPTION | WS_SYSMENU CAPTION "Practice by keygenning this - Kwazy Webbit 2004" LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US FONT 8, "System"

在下面你可以看到对话框中所有元素的细节,包括"Name"、"Serial"标 签和两个按钮。咱们把这个对话框修改成我们喜欢的,好不好?首先将两个按钮的名字修改成"Check"和"Exit": (p27)

CONTROL "Serial:", -1, STATIC, SS_ CONTROL "Check", 1002, BUTTON, BS_ CONTROL "Exit.", 1003, BUTTON, BS_

现在,我们修改顶部的标题: (p28)

103 DIAL	OGEX 22, 17,	170, 41	
STYLE DS	CENTER WS	CAPTION	WS SYSMENU
CAPTION	"Super Sweet	Ultimate.	Crackme!"
LANGUAGE	LANG ENGLISH	, SUBLANC	E ENGLISH US
FONT 8,	"System"		
{			

CONTROL "", 1000, EDIT, ES_LEFT | ES_AUI

现在点击"Compile"按钮,就会看到我们的窗口更新了: (p29)

Super Sweet Ultimate Crackme	el
Name:	Check
Serial:	Exit.

好,相当不错。将其保存("File" -> "Save"),将新的 crackme 载入 011y (原始的 crackme 被 Resource Hacker 以 Crackme8_original 名字保存), 运行它: (p30)

04000 PUS	0 DWORD PTR DS:[<&KERNEL32.GetModuleHan	dlef CotModule = NULL
04000 CAL PUS 04000 CAL	Super Sweet Ultimate Crackme!	×
INT DB DB DB	Name:	Check
DB DB DB	Serial:	Exit.
DB DB DB		
DB 0	90 90	

啊,相当好。现在我们正式开始...

六、破解程序

现在你应该知道怎么做了。搜索文本字符串: (p31)

Address	Disassembly	Text string	
00401049 0040104E 004010EC 004010EC 004010EC 0040110E 00401112 00401131 00401131 00401136 00401142 0640118F	PUSH Crackme8.0040230C PUSH Crackme8.004022CC PUSH Crackme8.00402384 PUSH Crackme8.00402384 PUSH Crackme8.00402384 PUSH Crackme8.00402384 PUSH Crackme8.0040232C PUSH Crackme8.00402324 DE 00	ASCII "A problem has occurred" ASCII "There is no valid key for this name, please use another one." ASCII "PFFT." ASCII "You might want to enter a name?" ASCII "Thats not a proper serial Use a normal number between 1 and 2^32" ASCII "Thats not a proper serial Use a normal number between 1 and 2^32" ASCII "You got it?" ASCII "WRONG" (Initial CPU selection)	
1			*

我们了解了两件事:1)序列号必须是一个1到非常大的数字;2)我们知道了好消息和坏消息生成的地方。咱们转到好消息那:(p32)



双击进到相关领域。我们看到好消息的路径是从 401131 开始的,坏消息 从 40113D 开始。我们看到那个跳转指令(JE SHORT Crackme8.0040113D)在 401131 处,比较指令(TEST EAX, EAX)在 40112A 处。咱们在 40112F 处设置 BP, 然后运行程序。输入用户名和序列号后点击"Check"。011y 随后断在了 我们的断点处: (n33)

1.1111	V	(Pee)	
00401124 00401125 00401120 00401120 00401120 00401122	- 50 - E8 D6FEFFFF - 85C0 - 59 - 59 - 59 - 56	CUSH EHA CHIL Crackme8,00401000 TEST EAX,EAX POP ECX POP ECX PUSH ESI	
00401131 00401136	. 68 38234000 68 20234000	UE SHURI Crackme8.00401130 PUSH Crackme8.00402338 PUSH Crackme8.0040232C	ASCII "RIGHT" ASCII "You got itt"
0040113B 0040113D 00401142 00401142	·^ EB DA > A1 B4234000 • 68 24234000 • 69 08	UMP SHORT Crackme8.00401117 MOU EAX,DWORD PTR DS:[4023B4] PUSH Crackme8.00402324 PUSH Crackme8.00402324	Title = "WRONG"
00401149 0040114A 0040114B	• 59 • 99 • F7F9	POP ECX CDQ IDIV ECX	
0040114D 00401154 00401155 00401155	 FF3495 28204000 57 FF15 10204000 FE05 84234000 	PUSH DWORD PTR DS:[EDX*4+402028] PUSH EDI CALL DWORD PTR DS:[<&USER32.MessageBoxA>]	Text = "We're just as far as when we started :(" hOwner = 004506E8 ('Super Sweet Ultimate Crackmet',class='# MessageBoxA
00401155 00401161 00401168	. 833D 84234000 0E	CMP DWORD FIR DS:[14023B4],08	
0040116E	57	PUSH EDI	

我们可以看到,011y 依旧要跳过好消息部分,直达坏消息部分。你知道 了那个路径...,清除 0 标志位运行程序: (p34)

	Super Sw	eet Ultimate (Crackme!		25
SH	Name:	R4ndom		Check	
SH	Serial:	1234567		Exit.	
P		-	RIGHT	23	_
SH SHORT SH Cra SH Cra SH Cra V EAX, SH Cra SH OB	Crackme8 ckme8.004 ckme8.004 T Crackme DWORD PTR ckme8.004	.0040113D 02338 0232C 8.00401117 1 DS:L4023B4 02324	You got it!		RIGHT" You got it "WRONG"
P ECX Q IV ECX SH DWOI SH EDI L DWOI C DWOR	RD PTR DS	:[EDX#4+402 :[<&USER32.] [402384]	OK		"We're jus = 004506E8 BoxA

成功了!现在咱们快速的创建一个补丁。重启应用,找到断点(通过断点 窗口),在JE指令上点一下,再按一下空格键,NOP掉跳转指令,这样我们就 能够一直的直达好消息部分: (p35)

0040112E 00401131 00401136 00401138	. ~ 74 0C . 68 38234000 . 68 2C234000 . 68 PD	USH ESI JE SHORT Crackme8,0040113D PUSH Crackme8,00402338 PUSH Crackme8,0040232C JMP SHORT Crackme8,00401117	ASCII "RIGH ASCII "You
0040113D 00401142 00401147 00401149	> 4A1 B4234000 . 68 24234000 . 6A 0B . 59	Assemble at 0040112F	= "WF = 132.75
0040114H 0040114B 0040114D 00401154 00401155	· 99 · F7F9 · FF3495 28204 · 57 · FF15 1020400		
00401158 00401161 00401168 00401168	 FF05 B423400 833D B423400 ØF85 5BFFFFF 56 	Fill with NOP's	Assemble Cancel
0040116F 00401170	- E9 4EFFFFFF	PUSH EDI JMP Crackme8.004010C3	a

先点 "Assemble" 然后是 "Cancel"。右键然后选择 "Save to executable" -> "All modifications",再选择 "Copy all"。右键弹出的窗口,选择 "Save file"保存它。现在你有了一个打过补丁的并且修改过资源的 crackme, 你输入的任何序列号都会让他显示好消息。

七、值得思考的事

我想说的是, Resource Hacker 是一个有意思的非常有用的小程序。通过 它你不仅仅可以修改一个文件的许多东西(字符串、图标、标签、按钮、标 题),你也可以用它修改 Windows 自身的许多东西(开始按钮、上下文菜单、 计算机的"关于"对话框等)。事实上, Resource Hacker 正是我的版本的 011y 的图标修改工具!

第八章:参考引用框架

一、简介

我们现在要研究的 crackme,相比来说更具挑战性。它就是 Crackme3. exe。 咱们也会学习几个新技巧。

你可以在教程页下载相关文件以及本文的 PDF 格式版本。

二、探究二进制文件

启动 011y 并载入 crackme。它会载入、分析并暂停在第一行: (p1)



运行下程序看看什么样: (p2)



好吧,没啥东西。选择 "Help" ->" Register": (p3)

Me v1.0			GetModule Title = N Class = " FindWindo
egister Name Serial		Car	ncel
	Me v1.0 egister Name Serial	Me v1.0 egister Name Serial	Me v1.0

现在咱们来到了某个地方。奇怪了,怎么和我们的 FAKE 那么像。试着输入用户名和序列号看看程序有什么反应: (p4)

gister		
Name	R4ndom	-
Serial	12121212121212121	
()K Са	incel

嗯。这回弹出了一个显示坏消息的对话框: (p5)

No luck!	
⚠	No luck there, mate!
E	ок

有时候,对于一个比较小的程序,我喜欢向下多翻几页看看有没有什么有意思的东西。我向下翻了大概6页,然后我看到了一些相当有趣的东西: (p6)



看看在 MessageBoxA 函数前面的文本。如果你往 MessageBoxA 函数上面的 文本的左边看的话,你会看到一条黑线将函数的参数框起来: (p7)



011y 给你显示的是准备传递给函数的参数,就是被调用的那个函数的。本例中个,参数1是窗口的类型,参数2是窗口的标题("Good work!"),参数3 是窗口显示的文本("Great work..."),参数4是窗口所有者的句柄。最后, MessageBoxA函数被调用。你可以在MessageBoxA上右键,选择 Help on symbolic names"来查看传递给函数的参数以及返回值。

现在,我们对比着看下紧随其后的那部分: (p8)

Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL Title = "No luck!" Text = "No luck there, mate!" hOwner = 00401000 MessageBoxA

对这两个函数的调用有很大的不同。一个看起来真的不错,而另一个却不 是。我想我们大家都承认,我们宁愿要第一个调用。现在咱们要记住

R4ndom's Essential Truths About Reversing Data #2: R4ndom 关于逆向数据的必备真言 2:

2. 大部分的保护机制都可以被绕过,通过修改一个简单的跳转指令来跳转到"好 的"代码处,而不是"坏的"代码处(或者避免跳转跳过"好的"代码)。

如果你看两个函数的上面几行,你会看到几个 jmp 语句,它们决定了你将 走哪条路,好的路或者坏的路。99%的应用里的 99%的时间都是这样。窍门就是 找到这个跳转。(当然剩下的1%要难得多,不过我们不会接触。)我们的例子中, 在 401344 和 40134B 有跳转。现在,作为一个已经训练过的逆向工程师,这些 跳转很快就被略过(如果你想知道为啥,是因为它们和我们的消息框在不同的 函数中,所以它们不会跳过我们的坏消息或跳转至好消息处,后面会讨论这个)。 咱们来研究研究它们: (p9)



首先, 点一下 40134B 处的 JMP 指令。会看到一个红线指示该 JMP 将跳到哪, 我们看到它走的是一条错误的路! (p10)



它没有跳到我们的好消息那,也没有跳过坏消息,反而往上跳到前面的代码了。我们试试 401344 那个 JMP。这个事实上和那个指向的一样(仍然是错误的路),所以看起来我们的第一个猜测是错的。

顺便说一下,就像我早些说的,老鸟忽视这些跳转的原因是因为 011y 显示 函数的方式。如果你注意看第一列(地址)和第二列(操作码)之间的话,会 看到一些粗黑线。这些线是 011y 放进去的,用来区分函数(有时候 011y 无法 指出函数的起始点和结束点,所以就不会有这些线): (p11)



本例中,你可以看到那两个 JMP 是在我们的好消息和坏消息的上面的函数 中。因为它不会跳转到好消息或者坏消息处,它们真的对我们没有任何帮助。 这也告诉你另一件事,第一个消息框(好消息那个)和坏消息框不是在同一个 函数中。这些都告诉我们,这些函数都是在别的地方被调用的,并且在它们被 调用之前的某处决定了哪个被调用,是好的还是坏的。咱们看看怎么才能绕过 这些干扰.....

三、查找参考

在好消息函数的第一行,也就是 40134D 那行上右键。选择 Find References To" ->" Selected Command"(或者按 Ctrl+R): (p12)

084401325 BS 0000000 MOU EAX. 094401325 > 5F POP EDI POP EDI 094401325 > 5F POP EDI POP EDI 094401325 > 5F POP EDI POP EBU 094401327 > 5B POP EBUE EQUE EQUE EQUE 094401327 > 52 1000 RETN 10 EQUE 094401327 > 52 1000 F20300000 CTP EBUE 094401325 > 817D 10 F20300000 CTP EBUE 094401323 > 817D 10 F20300000 CTP EBUE 094401333 > 75 11 JUC SHOR SHOR	9 -81, 3F2 T CRACKME. 00401346	kerne 132.7C817077 kerne 132.7C817077 kerne 132.7C817077 kerne 132.7C817077
B84481335 > 68,00 PUSH 0 004481337 > FF75 08 PUSH E 004481337 > E8 73010000 OHL CJ 004481337 > E8 73010000 OHL CJ 004481336 > E8 00000000 MOV EAX 004481344 - EB DF MOV EAX 004481344 - EB D8 OHL CJ 004481344 - EB D8 OHL CJ 004481344 - EB D8 OHL CJ 004481346 - E8 08 OHL CJ 004481346 - E8 08 OHL CJ 004481347 - E8 D9000000 PUSH 08 004481354 - FF5 08 PUSH 08 004481354 - FF5 08 PUSH 08 014481364 - E8 D0000000 CHL CJ 014481365 - E8 D0000000 CHL CJ 014481364 - E8 D00000000 CHL CJ 014481365 - E8 D00000000 CHL CJ 014481365 - E8 D00000000 CHL CJ 014481365 - E8 BD0000000 CHL CJ 014481375 - E8 BD0000000	Backup Copy Binary Assemble Sp Label : Comment ; Breakpoint Hit trace Run trace Run trace New origin here Ct Go to Follow in Dump View call tree Ct Search for	<pre>ipace B_OK:MB_ICONEXCLAMATION:MB_APPLMODAL Good work!" iveat work: mate!\rNow try the next CrackMet" 00401000 xA = MB_OK B_OK:MB_ICONEXCLAMATION:MB_APPLMODAL No luck there, mate!" 00401000 xA = MB_OK Etrl+Gray * i10228 Ctrl+K</pre>
00400138F .	Find references to View Copy to executable	Selected command Ctrl+R Instediate constant
Local call from 0040124C	Analysis Detach Process	

R References in CRACKME:CODE to 0040134D		4D	
Address 0040124C 0040134D	Disassembly CALL CRACKME.0040134D PUSH 30	Comment (Initial CPU selection)	
			>

该窗口显示的是 011y 能够找到的 CALL 或 JMP 到*这个*地址的所有参考 (CALL 和 JMP)。现在,双击列表中的第一个(就是那个不是红色的),然后你 就会来到调用这个(好的)消息的那行: (p14)



在 40124C 那行你可以看到指令 CALL CRACKME. 0040134D。40134D 就是好消 息对话框的第一行。咱们在这里设置一个断点: (p15)



现在,咱们对另一个函数做相同的操作,也就是坏消息那个。转到 401362 那行,就是坏消息函数的第一行,右键选择 Find References To" -> "Selection (or ctr1-R)"。这会再一次调出参考窗口。双击第一条,我们就会来到调用坏 消息的地方: (p16)

00401238	. E8 9B010000	CALL CRACKME, 004013D8	
00401230	. 83C4 04	ADD ESP.4	
00401240	. 58	POP EAX	
00401241	3BC3	CMP EAX.EBX	
00401243	··· 74 07	JE SHORT CRACKME.0040124C	
00401245	. ES 18010000	CALL CRACKME, 00401362	-
0040124A	.~ EB 9A	JMP SHORT CRACKME.004011E6	
STREET, SQUARE, SQUARE	> E8 FC000000	CALL CRACKME, 0040134D	
00401251	.^ EB 93	JMP SHORT CRACKME,004011E6	
00401253	r. C8 000000	ENTER 0.0	
00401257	53	PUSH FBX	

有意思的是,它就在我们刚才设置的断点的上面 2 行! 咱们在这里也设置 一个断点: (p17)

00401230	. 83C4 04	ADD ESP,4	
00401240	. 58	POP EAX	
00401241	. 3BC3	CMP EAX.EBX	
00401243	.~ 74 07	JE SHORT CRACKME, 0040124C	
064913451	. ES 18010000	CALL CRACKME.00401362	
0040124A	.^ EB 9A	JMP SHORT CRACKME.004011E6	
88401340	> E8 FC000000	CALL CRACKME.0040134D	
00401251	.^ EB 93	JMP SHORT CRACKME.004011E6	
00401253	r. C8 000000	ENTER 0.0	
00401257	. 53	PUSH EBX	
00401258	56	PUSH EST	

***注意,有时候你选中一行然后查找参考,但是一个都没有。导致这个结果的原因有两种:1)你选择了错误的函数"入口点",也就是调用这个函数应该 call 或 jump 其他的地方,但是它们却调用了别的行,有可能就是你选择行的前面或后面那行。选择正确的行来查找参考需要花时间和技巧,不过要坚持下去。2)代码中没有明显指向这一行的指令。记住,程序运行时有许多数字被动态的操纵, call 或 jump 指向的地址也不例外。所以,如果 call 的地址是动态创建的话,所以 011y 就没有办法提前知道这行会被调用,所以 011y 也就不会将这个参考列出来。关于这个也是有方法的,不过这会我不打算讨论。

现在,如果我们看看这两个 CALL 的附件的话,会看到几个 jmp 指令。第一个,401243 的 JE SHORT CRACKME.0040124C。当然,你知道 JE 是啥意思,因为你已经读过汇编语言的书 (参见 R. E. T. A. R. D. 规则#1),不过为了证实,我们假定你不知道这个特别的助记符(指令)是啥意思。这就是插件 MnemonicHelp存在的原因。右键 JE 指令,在上下文菜单中选择"? JE": (p18)

Intel x86 Instructions			
File Edit Bookmar	k Options Help		
Contents Index	Back Print		
Jcc-Jump if	Condition Is Met	- D	
See also			
Opcode	Instruction	Description	- 18
77 cb	JA rel8	Jump short if above (CF=0 and ZF=0)	3
73 cb	JAE relâ	Jump short if above or equal (CF=0)	_
72 cb	JB re/8	Jump short if below (CF=1)	
76 cb	JBE rel8	Jump short if below or equal (CF=1 or ZF=1)	
72 cb	JC rel8	Jump short if carry (CF=1)	
E3 cb	JCXZ rel8	Jump short if CX register is 0	
E3 cb	JECXZ re/8	Jump short if ECX register is 0	
74 cb	JE re/8	Jump short if equal (ZF=1)	
7F cb	JG rel8	Jump short if greater (ZF=0 and SF=OF)	
7D cb	JGE rel8	Jump short if greater or equal (SF=OF)	
7C cb	JL re18	Jump short if less (SF<>OF)	
7E cb	JLE rel8	Jump short if less or equal (ZF=1 or SF<>OF)	
76 cb	JNA rel8	Jump short if not above (CF=1 or ZF=1)	
72 cb	JNAE re/8	Jump short if not above or equal (CF=1)	
73 cb	JNB rel8	Jump short if not below (CF=0)	
77 cb	JNBE rel8	Jump short if not below or equal (CF=0 and ZF=0)	
73 cb	JNC rel8	Jump short if not carry (CF=0)	
75 cb	JNE rel8	Jump short if not equal (ZF=0)	
7E cb	JNG re/8	Jump short if not greater (ZF=1 or SF<>OF)	
7C cb	JNGE re/8	Jump short if not greater or equal (SF<>OF)	
7D cb	JNL rel8	Jump short if not less (SF=OF)	
7F cb	JNLE re/8	Jump short if not less or equal (ZF=0 and SF=OF)	
71 cb	JNO re/8	Jump short if not overflow (OF=D)	
78 cb	JNP re/8	Jump short if not parity (PF=0)	
79 cb	JNS rel8	Jump short if not sign (SF=0)	

这个窗口比较长,因为有大量的跳转指令。如果我们向下看那个"JE"的话,会看到它是"Jump if Equal (ZF = 1)"。意思是如果 0 标志位被置 1 就跳转(或者被比较的两个项目相等)。前面的教程中我们复习过标志位,所以你应该知道,如果被比较的两个对象相等,JE 就会跳转。我们也能够发现,这个 JE 跳过了对坏消息的调用,并且紧随跳转的那条指令是对好消息的调用。如果 JE 没有跳,我们就会调用坏消息。所以,我们**想要**这个跳转实现,以便我们能够调用好消息。咱们操作下看看。在 JE 指令上设置一个断点,重启(或运行)应用。点击 crackme 中的"Help"->" Register",输入用户名和序列号,然后点OK: (p19)



哇! 等等! 显示了坏消息,并且 011y 也没有断下来? 也就是说 011y 永远 也不会运行到我们的断点! 这是咋回事呢。

事实上,这个在逆向工程领域里是可以用得着的。我向你保证,每一个专家级逆向工程师/破解者这时候都会想"我错过什么了吗?一个 0xcc 中断? IsDebuggerPresent (译者注:一个 Windows API)? NTF1ags? TLS 回调?", 然后白费力气去寻找一些过于复杂的解决方案。但是我们只是初学者,我们只有几个工具可以使用,其中一个就是搜索字符串,那就试试这个吧: (p20)

R Text strin	gs referenced in Crackme3:0	CODE	
Address 00401006 Crac 00401007 00401077 00401087 00401087 00401087 00401087 00401213 00401213 00401213 00401228 00401287 00401287 00401284 00401354 00401354 00401370 00401384	Disassembly PUSH 0 PUSH 0 PUSH Crackme3.004020F4 MOU DWORD PTR Ds:[4020881,Cr MOV DWORD PTR Ds:[4020881,Cr PUSH Crackme3.004020F4 PUSH Crackme3.00402115 PUSH Crackme3.00402155 PUSH Crackme3.0040217E PUSH Crackme3.0040217E PUSH Crackme3.0040217E PUSH Crackme3.0040217E PUSH Crackme3.0040217E PUSH Crackme3.00402174 PUSH Crackme3.00402169 PUSH Crackme3.00402169 PUSH Crackme3.00402169 PUSH Crackme3.00402169 PUSH Crackme3.00402169 PUSH Crackme3.00402169	Text string Initial CPU selection: ASCII "No need to disasm the codet" ASCII "No need to disasm the codet" ASCII "CrackMe v1.0" ASCII "CrackMe v1.0" ASCII "DLG_ABDUT" ASCII "DLG_ABDUT" ASCII "DLG_AEGIS" ASCII "BLG22121212" ASCII "R4ndom" ASCII "1212121212" ASCII "R4ndom" ASCII "Boreat work, matet\rNow try the next CrackMet" ASCII "Boreat work, matet\rNow try the next CrackMet" ASCII "No luckt" ASCII "No luck there, matet" ASCII "No luck there, matet"	
Saml			×

现在,你可以看到一些相当有趣的东西...。有两个"No luck!"坏消息, 但是只有一个好消息。也就是说,代码中的某个地方做了检查,如果没有通过 就会显示坏消息。这是一个在反逆向工程中非常流行的技术:找一个非常明显 的地方放好的/坏的消息,然后添加一个不是那么明显的检测。如果你看看代码 窗口我们的好消息和坏消息所在的位置,你会发现字符串"No luck!"是在 40136B,所以我们知道那不是我们要找的字符串。所以咱们双击下 4013AF 那个: (p21)



这个坏消息是在程序内存中完全不同的区! 我认为这个 crackme 是在太简单了! 好,咱们深呼吸然后想想 RETARD 规则 #2,找找 比较/跳转。本例中在 4013AA 处有个 JMP,当你点击它的时候,011y 会显示一个箭头刚好跳过了坏消息。看起来前途光明啊...。那就试试吧! 在那个 jmp 指令处设置一个断点,重 启应用并运行。

***你有可能得到错误的消息,就像我们上一章中断点被破坏那样。如果发生了,像上一次那样做就行了。打开 BP 窗口,在你运行程序前重新启用所有的断点:)(p22)



操蛋!!! 好吧,不起作用,所以我猜我们得深入挖掘了。咱们看看代码, 试试理解到底是什么个情况(该是你组合阅读大放异彩的时候了☺): (p23)

00401378 00401375 0040135 0040135 00401385 00401385 00401385 00401389 00401388 00401388 00401388 00401388 00401389 00401392 00401392 00401392 00401399 00401399 00401399	 E8 BD000000 C3 S87424 04 56 8406 84C0 74 13 3C 54 72 1F 3C 54 73 03 46 E8 EF E8 39000000 46 5E 20000000 81F7 78560000 8872 	CALL & UMP.&USER32.MessageBoxA> RETN MOV ESI,DWORD PTR SS:LESP+41 PUSH ESI MOV AL,BYTE PTR DS:LESIJ TEST AL,AL UE SHORT Crackme3.0040139C CMP AL,41 UB SHORT Crackme3.00401383 CML Crackme3.00401302 INC ESI UMP SHORT Crackme3.00401383 CALL Crackme3.00401302 INC ESI UMP SHORT Crackme3.00401383 POP ESI CALL Crackme3.00401302 XOR EDI,5678 MOV.FRX.FDI	Crackme3.00402188 Crackme3.00402188 Crackme3.00402188 Crackme3.00402188 Crackme3.0040218E
004013AC 004013AD 004013AF 004013B4 004013B4 004013B9 004013B9 004013C1 004013C2		JMP SHORT Crackme3,004013C1 POP ESI PUSH 30 PUSH Crackme3,00402160 PUSH Crackme3,00402169 PUSH LARS.11 CALL <ump.&user32.messageboxa> RETN XOR EDI.EDI</ump.&user32.messageboxa>	Crackme3.0040218E Title = "No luck!" Text = "No luck there, mate!" hOwner = 014103F4 ('CrackMe v1.0',class='No need MessageBoxA

好,我们知道了一件事,因为教程的前面我们学过,就是函数的开始和结束点。图片中你能通过蓝色箭头看到。所以,从函数的起始点开始,有一个循环首先检查 AL 是不是 0(TEXT AL, AL),然后循环将 AL 和一组数值(41,5a)进行比较。期间,有一些依赖于 AL 值得跳转。首先,咱们看看到底哪个跳转会调用我们的坏消息(有一个 JMP 指令刚好在坏消息前面,没有什么可以"空降" 直达它。所以,必须有什么东西跳过那个跳转来运行坏消息代码。最有可能的跳转是在 4013AC)。

点一下 4013AC,也就是坏消息框的第一条指令,右键该行选择 "Find References To" -> "Selected Address"。(我知道一旦你点了这行,就会显 示一个红色箭头,显示了哪条指令调用了它,但是我们怎么才能知道就没有别 的指令调用坏消息呢。找到所有的参考可以帮助我们确定,有可能只有一个。) 然后我们就再次看到了参考窗口: (p24)

Address	Disassembly	Comment	1
0040138B 004013AC	JB SHORT Crackme3.004013AC POP ESI	(Initial CPU selection)	

现在,双击第一个,咱们来看看哪一行正在调用坏消息: (p25)

0040137E 00401382 00401383 00401385	\$ 887424 04 . 56 > 8806 . 8400	MOV ESI, DWORD PTR SS:[ESP+4] PUSH ESI MOV AL, BYTE PTR DS:[ESI] TEST AL, AL	Crackme3.00402188
00401387	.~ 74 13	CMP AL. 41	
0040138B		JB SHORT Crackme3.004013AC	
0040138D 0040138F 00401391	. 3C 5A .∽ 73 03 . 46	CMP AL, 5A UNB SHORT Crackme3.00401394 INC ESI MMP SHORT Crackme3.00401393	Crackme3.00402188
00401392	> E8 3900000	CALL Crackme3.004013D2	
00401399	A FR F7	INC EST	Crackme3.00402188
00401390	> 5E	POP ESI	Crackme3.0040218E
0040139D 00401382	. 81F7 78560000	XOR EDI.5678	
004013A8	. 8BC7	MOU EAX, EDI	
004013HH	.~ EB 15	POP ESI	Crackme3.0040218E
004013AD	68 30	PUSH 30	
004013HF	68 60214000	PUSH Crackme3,00402160	Tevt = "No luck!"
004013B9	. FF75 08	PUSH LARG.11	hOwner = 014103F4 ('CrackMe v1.0', class='No need t
004013BC	E8 7900000	CALL KUMP.&USER32.MessageBoxA>	LMessageBoxA
00401301	1 US	REIN	

噢,原来是循环中的一个。注意参考窗口中旁边的那个红色的那行(我们 现在可以忽略它),只有一个到该地址的参考,所以我们可以保证 40138B 这行 是调用坏消息的唯一代码。所以 40138B 的 JB SHORT 4013AC 就是那个罪魁祸首。 咱们试着在它上面设置一个 BP,临时修改下看看能否绕过这个坏消息。在 40138B 设断点,重新运行程序: (p26)



嗯。箭头是灰色的,我们知道在这次的循环迭代中我们没有跳到坏消息那。 按下 F9 执行循环体: (p27)

00401383	> 8A06	MOU AL, BYTE PTR DS:[ESI]	
00401385	. 84C0	TEST AL, AL	
00401387	.~ 74 13	JE SHORT Crackme3,0040139C	
00401387	. 3C 41	CMP AL, 41	
94441389 96461387 96461387 96461397 96461392 96481394 96481392 96481394 96481395 96481396 96481397 964401392 964401392 964401384 96481386 96481386 96481386 96481386	. 3C 41 . 3C 5A . 3C 5A . 46 . EB EF . ES 39000000 . 46 . EB E7 . EB 20000000 . 81F7 78560000 . 81F7 78560000 . 88C7 . EB 15 . 63 60214000 . 64 60 . 65 60214000 . 65 60216000 . 65 60000000000000000000000000000000000	CHP HL, 41 JB SHORT Crackme3,004013AC CMP AL, 5A INS SHORT Crackme3,00401394 INC ESI JMP SHORT Crackme3,00401388 CALL Crackme3,004013D2 INC ESI JMP SHORT Crackme3,00401388 POP ESI CALL Crackme3,004013C2 XOR EDI,5675 MOV EAX,EDI JMP SHORT Crackme3,004013C1 POP ESI PUSH Crackme3,00402160 PUSH Crackme3,00402160 PUSH Crackme3,00402160 PUSH Crackme3,00402160 PUSH CRACKME3,00402160	Crackme3.0040218F Crackme3.0040218F Crackme3.0040218E Crackme3.0040218E Title = "No luck!" Text = "No luck!" Text = "No luck there, mate!" hDumper = 014203F4 ('CrackMe v1.0',class='No nee
004013C1	L> C3	RETN	The sease with
004013C2	\$ 33FF	XOR EDI, EDI	

啊! 第二次循环时它就要调用坏消息了。好,就让它那么干,看看我们跟踪的对不对。你可能注意到了,如果修改了0标志位,跳转仍然实现了。这是因为 JB 指令是跳转指令集中略有不同的那部分,它用进位标志位而不是0标志位(别担心,这些你的汇编语言书籍中全都有。)。所以双击那个进位标志位("C"): (p28)

CP 0 0	ESSOS

然后那个箭头就会变为灰色: (p29)

00401587 . 7 74 15	UE SHUKI Crackmes, 00401590	
00401389 . 3C 41	CMP AL. 41	
R0401388	JB SHORT Crackme3,004013AC	
0040138D . 3C 5A	CMP AL.58	
0040138F .v 73 03	JNB SHORT Crackme3,00401394	
00401391 46	INC ESI	Crackme3, 0040218F
00401392 A FR FF	JMP SHORT CrackmeS, 00401383	
00401394 > F8 39000000	CALL Crackme8, 00401302	and a second secon
00401399 . 46	INC ESI	Crackme3,0040218F
00401899 A EB E7	JMP SHORT Crackme3, 00401383	
0040139C	POP ESI	Crackme3, 0040218E
6646139D . F8 26666666	CALL Crackme3, 00401302	
00401302 81E7 78560000	X08 ED1, 5678	
00401308 8BC7	MOU FAX. FDI	
00401300 V FB 15	JMP SHORT Crackme3, 00401301	
0040138C >5E	POP ESI	Crackme3, 0040218E
00401300 69 30	PUSH 30	
0040138F 1, 68 60214000	PUSH Crackme3,00402160	Title = "No luck?"
004013B4 . 68 69214000	PUSH Crackme3, 00402169	Text = "No luck there, maret"
004013B9 . FF75 08	PUSH LARG. 11	hOwner = 014203F4 ("CrackMe v1.0".ctass="No
004013BC . E8 79000000	CALL KUMP. &USER32, MessageBoxA>	-MessageBoxA
004013C1 > C3	RETN	
004013C2 rs 33FF	XOR EDI.EDI	
004013C4 . 33DB	XOR EBX EBX	
004013C6 8A1E	MOV BL. BYTE PTR DS: [ESI]	

现在我们再次运行循环,看看循环中还有没有调用坏消息的。按 5 次 F9, 没有一次调用坏消息。事实上,在第五次 F9 之后,我断在了一个旧断点处,我 们首先想到的是补丁: (p30)

00401228	68 8E214000	PUSH Crackme3.0040218E	ASCII "R4NDOM"
00401220	E8 4C010000	CALL Crackme3.0040137E	
00401232 00401233 00401238	50 68 7E214000 E8 9B010000	PUSH EAX PUSH Crackme3.0040217E	ASCII "1212121212"
0040123D	. 83C4 04	ADD ESP,4	Crackme3.0040218E
00401240	. 58	POP EAX	
00401241	. 3BC3	CMP EAX,EBX	
00401245	·~ 74 07 E8 18010000	JE SHORT Crackme3.0040124C CALL Crackme3.004013	1
0040124C	-^ EB 9H -> ES FC000000	CALL Crackme3.04.11116 CALL Crackme3.00401140	
00401253 00401257	C8 00000	ENTER 0,0 PUSH EBX	and a second second
00401258	· 56	PUSH ESI	Crackme3.00402188
00401259	· 57	PUSH EDI	
0040125H 00401261 00401263	- 8170 0C 10010000 - 74 34 - 8170 0C 11010000	JE SHORT Crackme8.00401297 CMP CERC, 21, 110	
0040126A	.* 74 35	JE SHORT Crackme3,004012A1	
0040126C	. 837D 0C 10	CMP LARG,23,10	
00401270	·* 0F84 81000000	JE Crackme3.004012F7	
00401276	. 817D 0C 01020000	CMP CARG.23,201	
00401275	- 88 0000000	MOV EAX,0	Crackme3.0040218E
0040127F	> 5F	POP EDI	
00401285	: 5E	POP ESI	Crackme3.0040218E
00401286	: 5B	POP EBX	Crackme3.0040218E
00401287	: C2 1000	RETN 10	TOUC

那么,这就意味着我们已经成功的绕过了对坏消息的第一个检测,并且回 到了原来的检测点。咱们给第一个检测打个补丁,这样就再也不用操心它了, 就可以将注意力集中在主要的检测点。回到 40138B 的断点处,我们得想想怎么 给它打补丁而不让它跳转到坏消息那。记住,跳转是在第二次循环时实现的, 只有在 AL 的值小于 41 时才成立(相关指令是 CMP AL, 31, JB SHORT 4013AC)。 如果我们只 NOP 掉这个跳转会怎么样?它就再也不会跳了,我们一点也不用担 心它会跳到坏消息那⁽²⁾: (p31)

00401378	. E8 BD000000	CALL (JMP.&USER32.MessageBoxA)	MessageBoxA	
0040137E 00401382 00401383	\$ 887424 04 .56 .> 8806 .> 8806	MOU ESI, DWORD PTR SS: LESP+4] PUSH ESI MOU AL, BYTE PTR DS: LESI]	Assemble at 0040138B	
00401385 00401387 00401389	.~ 74 13 . 3C 41	JE SHORT Crackme3.0040139C CMP AL,41 JE SHORT Crackme3.0040138C	nop	•
0040138D 0040138F 00401391 00401392 00401394	.~ 3C 5A .~ 73 03 . 46 . EB EF . E8 39000000	CMP PL,54 UNB SHORT Crackme3,00401394 INC ESI UNP SHORT Crackme3,00401383 CHLL Crackme3,004013D2	Fill with NOP's	Assemble Cancel
00401399 00401398 00401390 00401390 00401390 00401388 00401388	*^ 46 EB E7 > 5E E8 20000000 81F7 78560000 • 88C7_	LINC ESI UMP SHORT Crackme3.00401383 POP ESI CALL Crackme3.004013C2 XOR EDI.5678 MOV ERX,EDI	Crackme3.0040218E	





右键,选择 "Copy to executable" -> "All modifications"。弹出内 存窗口,右键该窗口,选择 "Save File",将其另存为 crackme_patch1.exe: (p33)

D File C:\Use	ers\Random\De	sktop\Jason\Tuts\Intro to Reverse Engineering\8 Frame Of Reference\Crackme3.EXE	
000009922 5: 00000993 8: 00000995 8: 00000985 9: 00000988 9: 00000988 9: 00000988 9: 00000980 9: 00000990 3: 00000991 4: 00000994 6: 00000994 6:	6 A06 4C0 4 13 C 41 0 C 5A 3 03 6 B EF 8 39000000	PUSH ESI MOU AL, BYTE PTR DS; [ESI] TEST AL, AL JE SHORT 0000099C CMP AL, 41 NOP CMP AL, 5A JNB SHORT 00000994 JNB SHORT 00000994 JNC ESI JNC ESI JNC ESI JNC ESI CALL 000009D2	
00000999 00000990 00000990 00000990 00000902 00000902 00000902 00000902 00000902 5	6 B E7 E 3 20000000 1F7 78560000 BC7 B 15 E	INC ESI JMP SHORT 00000983 POP ESI CALL 000009022 XOR EDI, 5678 MOV EAX,EDI JMP SHORT 000009C1 POP ESI	

现在,在重新载入刚刚打过补丁版本前,我们要明白所有的补丁、注释和 (尤其是)断点都会被删除,因为所有的信息都存储在 Crackme3.udd 这个 UDD 文件中。我们将要打开的 Crackme3_Patch1,并没有和它相关的 UDD 文件。不过 还是有几个好消息的。本文的相关下载中包含有断点管理插件。如果你没有准 备好,那就将其拷贝到你的插件目录下,然后重启 011y。如果你一开始就安装 好了,那你就已经载入了它。现在打开断点窗口,右键并选择"Breakpoint manager"->" Export Breakpoints": (p34)

1	B Break	TEST AL,A	<u> </u>						80 94 90
00	Address 00401245 0040124C 0040138B	Module Crackme3 Crackme3 Crackme3	Active Always Always Always Always	Disassem CALL Cra CALL Cra JB SHORT	bly ckme3.004013 ckme3.004013 Crackme3.00	362 34D 34013AC	Comment	*	0 Crackme3. <mod 3B 32bit 0(FFFFF 33 32bit 0(FFFFF 3B 32bit 0(FFFFF 32 32bit 0(FFFFF 33 32bit 7EFDD00 9 32bit 7EFDD00</mod
00 85						Remove	Del		r ERROR_SUCCESS
14 14 8 00						Disable Edit condition Follow in Disasser	Space mbler Enter		3210
	T	TEST BL.B	L			Disable all	1	FCW 027F	Prec NEAR,53 Ma
		ADD EDI,E	BX BX	401301		Breakpoint Mana	ger 🕨	Import Breakpoint	s 🖌
	R	UMP SHORT ETN UB AL, 20	Crackme3.00	34013C6		Delete All BreakPo	pints	Export Breakpoints	
04		OV BYTE P ETN OR EAX, EA OR EDI, ED OR EBX, EB OV ESI, DW	TR DS:[ES]], X I X ORD PTR SS:[AL (ESP+4)	kernelS	Copy to clipboard Appearance	1 +		

保存文件,因为我们将会将其导入到新文件。现在,将新文件(打过补丁的)载入011y。它很可能会弹出一个消息框,告诉你断点被破坏了: (p35)

X Error In module 'Crackme3' OllyDbg encountered several corrupted breakpoints, where first byte of the actual command differs from that in the previous debugging session. This happens when program was recompiled or code is self-modifying (for example, self-extracting). For security reasons, OllyDbg will disable suspicious INT3 breakpoints. You can re-enable them in Breakpoints window. OK

点 OK 就行了。在我们的打过补丁的程序中打开断点窗口,很可能所有的(或 大部分)断点消息了。现在,右键并选择 "Breakpoint Manager" -> "Import breakpoints": (p36)

Address	Module	Active	Disassembly		Comment	
				Breakpoint Manager	•	Import Breakpoints
				Appearance		Export Breakpoints

现在你会看到我们原来的断点又回来了: (p37)

B Break	points				- E X
Address	Module	Active	Disassembly	Comment	×
00401245 00401245 00401388	Crackme3 Crackme3 Crackme3	Always Always Always	CALL Crackme3.00401362 CALL Crackme3.0040134D NOP		
	_				-

运行程序,011y 会断在我们的第一个断点也就是 401243 的 JE 指令处(如果你没有在该行设置断点,那就设一个。译者注: 吐一下槽,原作者太操蛋,从来都只打圆括号的左半个,右半个就不管了,我还得自己琢磨把右边的圆括号放哪)。重启应用并运行,你会断在这里: (p38)

00401240	3803	CMP EAX, EBX	UT UT
00401251 00401253 00401253 00401253	→ 74 87 = E8 18010000 → E8 96 → E8 FC000000 → E8 93 C8 000000 C8 000000	JE SHORT CRACKME.0040124C CALL CRACKME.00401362 JMP SHORT CRACKME.004011E6 CALL CRACKME.004011E6 CALL CRACKME.004011E6 UMP SHORT CRACKME.004011E6 ENTER 0.0	

现在,看那个灰色的箭头,就是从当前暂停行往下到 40124C。因为箭头是 灰色的,所以跳转不会实现。你也可以看反汇编窗口与数据窗口中间的那块, 它告诉你跳转**没有**实现: (p39)

0040129F1 00401284 00401284 00401286 00401286 00401285 00401285 00401285 00401285	→ EB E3 JHE SH > 3300 SOR SOR
Jump is 1 0040124C:	NOT taken =CRACKME.0040124C
Address	Hes dump
00402000 00402010 00402020 00402030 00402030 00402050 00402050 00402050 00402050	00 00 00 00 56 03 37 00 00 00 <t< th=""></t<>

这意味着,什么都不用做,程序不会跳到第二个调用,会直达第一个调用。 第一个调用会跳到坏消息那,所以我们真心不想让它发生。按一下 F8 单步步过。 就像 011y 告诉我们的,没有跳转,我们当前的位置就在调用坏消息的地方。按 一下 F7 单步步入那个 CALL,我们来到了坏消息所在函数的第一条指令处。现在, 如果我们按 F9 让程序运行,我们看到的正是意料之中的: (p40)

No luck!	
⚠	No luck there, mate!
E	ОК

咱们来看看能不能解决这个 (2)。重启应用,按F9运行之,选择"Help"->" Register",然后输入用户名和序列号。现在,当你按下 0K 按钮时,011y 会再次停在我们的第一个断点: (p41)



这回,咱们来帮助 011y 走正确的路。浏览下寄存器窗口,注意到 Z 标志位 是红色的。嗯,你知道该怎么做了: (p42)



注意,箭头是灰色的,显示跳转不会发生,不过现在变红了,在反汇编窗口和数据窗口之间的那个区域已经变成了"Jump will be taken (跳转将会发生)"。我们所做的就是告诉 011y 去修改标志位,该标志位用来判定两个东西是否相同,为了让它认为它们是相同的。所以现在,我们会跳过对坏消息的调用,转而去调用好消息!!!

咱们试试。按 F8 执行跳转,再按 F7 单步步入到那个 CALL。现在我们将跳转到好消息的起始处: (p43)



现在,按几次 F8,每按一次就观察一下堆栈窗口。你会看到 MessageBoxA 的参数被压入堆栈,本例中确实是好消息被压入堆栈。只要你单步步过 40135C 处的函数,新的消息对话框就会显示出来。我们已经破解了我们的第一个程序!!! (p44)



现在的问题是,我们只是临时性的修改了标志寄存器,当程序再次运行时, 它却不会再次的修改标志位,所以我们还是会得到坏消息。我们需要做的是通 过某种方式将修改保存起来,以便于每一次程序运行的时候,我们都能强制它 做跳转。这时候补丁要派上用场了。和我们以前做的一样:选中所有已修改的 行,右键并选择 Copy to executable"。在弹出的窗口中右键,点击 Save file"。 选一个名字,这就是你的打过补丁的版本: (p45)

Save file as					? 🛛
Save in: My Recent Documents Desktop My Documents My Computer	Ult pics	E exe	•		
My Network Places	File name: Save as type:	CRACKME1.EXE Executable file (*.exe)		-	Save Cancel

现在可以关掉数据窗口和 011y 了。打开你保存的打补丁版本文件夹,运行 打过补丁的程序。输入你的信息并验证: (p46)

😬 CrackMe v1.0	-		×	DIly2
File Help				Calc.EXE
	Goo	d work!		
		Greal Now	t work, mat try the nex OK	el t CrackMel

干的漂亮!你已经真正的破解了一个有些挑战的 crackme。

第九章:无相关字符串

一、简介

此次教程中我将会向我们的武器库中加入一个新的装备。如果搜索二进制 文件时发现没有可用的字符串你怎么办?我将会介绍一个新的 R. E. T. A. R. D. 规 则。此次教程(下一章也是)我们将研究"TDC"写的一个 crackme 叫 Crackme6, 相关下载里面包含有。总之,它不是一个硬骨头,不过我将会对其进行一些高 级分析,好为将来的教程做准备。

你可以在教程页下载相关文件以及本教程的 PDF 版本。

那么,咱们开始吧.....

011y 载入 Crackme6: (p1)



现在,我们已经知道操作程序了。运行程序看看情况: (p2)



嗯,看起来挺简单的。我输入了一个密码 1212121212,下面就是返回的情况: (p3)



相当直接呀。试试我们拿手的"字符串搜索",看看有什么: (p4)

Itemnessagen = 21. NULL mapA = 75143388 = 0 = BM_SETIMA(ID = 69 (105.	Go to Follow in Dump View call tree	۲ Ctrl+K	8 ESPUOZDI 8 Err 00000000000 8 Mask 111111	
TEFDE000 ItemMessageA	Search for	+	Name (label) in current module	Ctrl+N
NULL mapA	Find references to	+	Name in all modules	
= 75143388 = 0 = BM_SETIMA(ID = 6A (106, 72FDE000 IItemMessageA = 100. :NULL :NULL :nA = 75143388 = 1	View Copy to executable Analysis Help on symbolic name	۰ ۲ Ctrl+F1	Command Sequence of commands Constant Binary string	Ctrl+F Ctrl+S Ctrl+B
= WM_SETICON 7EFDE000	Detach Process		All intermodular calls	
	Process Patcher Analyze This!		All commands All sequences	
018FF8C 751 018FF90 7EF	Bookmark	+	All constants	
018FF98 777 018FF98 777 018FF9C 7EF	Code Rinner		All switches	/
018FFA0 776 018FFA4 008	IDAFicator		All referenced text strings	-
018FFAC 7EF 018FFB0 000	? CALL		User-defined label	
000 018FFB8 000 018FFBC 001	Script(S)		User-defined comment	
018FFC0 000 018FFC4 FFF 018FFC8 777 018FFC8 000	Script Functions	٠		

(p5)

Address Disassembly	Text string	
80401000 CALL CUMP. & conct 132. In FCC 00401103D PUSH Crackme6.0040303C 00401177 PUSH Crackme6.0040300F 00401203 PUSH Crackme6.0040300F 00401277 PUSH Crackme6.0040300F 00401279 PUSH Crackme6.0040301F 00401282 PUSH Crackme6.0040301F 00401282 PUSH Crackme6.0040300F 00401284 PUSH Crackme6.0040300F 00401205 PUSH Crackme6.0040300F 00401205 PUSH Crackme6.0040300F 00401315 PUSH Crackme6.00403045	AmenCon (Initial CPU selection) ASCII "TOC [#4]" ASCII "NLLJNN/KJAFJK." ASCII "NLLJNN/KJAFJK." ASCII "M3z(ji')Ifah@/Mnk/xnv." ASCII "M3z(ji')Ifah@/Mnk/xnv." ASCII "M3z(ji')Ifah@/Mnk/xnv." ASCII "M1z(ji')Ifah@/Mnk/xnv." ASCII "M1z(ji')Ifah@/Mnk/xnv." ASCII "NLLJNN/KJAFJK." ASCII "NLLJNN/KJAFJK." ASCII "NLLJNN/KJAFJK." ASCII "NLLJNN/KJAFJK." ASCII "NLLJNN/KJAFJK."	
		-

搞什么鬼这是!!! 这些一点用也没有啊☺。我们可以拿这些字符串干啥!?!? 明显,这个 crackme 将字符串加密了(或者是作者说一种很奇怪的语言:D)。好,是个好时候介绍

R4ndom's Essential Truths About Reversing Data #3: R4ndom 关于逆向数据的必备真理#3:

#3.不要依赖二进制文件当前已有的字符串。

不幸的是,在你开始研究真正的二进制文件(比如商业产品)时,它们中的大部分被以某种方式打包 以及/或 保护。干扰逆向工程师的一个最明显的方法是加密字符串。坦率地说,在逆向工程领域当我第一次研究一个感兴趣的新的二进制文件时,如果我搜索字符串并且搜出来了,我能够假定那个二进制文件很可能没有多少挑战。所以,你不能够依赖于那些东西(如果有当然更好。)。

二、模块间的调用

有鉴于此,我向你展示一个新的在没有字符串的情况下的技巧。大部分的 Windows应用程序使用一个标准的 API 集来完成特定的动作。例如,如果需要一 个简单的消息框的话就调用 MessageBoxA,当程序想要退出的时候就调用 TerminateProcess。因为大部分的应用都使用这些相同的 API,我们可以用这个 获利。例如,有些 API 可以用于从对话框的输入框(类似用户名和序列号)获 取文本。有可以被调用用来比较两个字符串的字符串比较函数(输入的密码和 程序中存储的密码相同吗?)。有读写注册表的 API(存储和读取你的注册状态)。

011y 提供了一种搜索所有被调用的 API 的方法。在反汇编窗口右键,选择 "Search for" -> "All intermodular calls": (p6)

052 057 058	Follow	Enter	idBitmapA>	•LoadBitmapA •lParam = 75143388 •wParam = 8	i .
05A 05P 061	Go to	•	P+81	Message = BM_SETI ControlID = 68 (1 build = ZEEDE000	MAGE 04.)
064	Follow in Dump		dDlgItemMessageA>	SendDlgItemMessag	eA
065 06E 071 076	View call tree	Ctrl+K	IS07C] IdBitmapA≻	hInst = NULL LoadBitmapA LParam = 75143388	
077 079 079	Search for		Name (label) in current mo	odule Ctrl+N	GE
080 083	Find references to	+	Name in all modules		ľ
488 986 996	View	+	Command	Ctrl+F	
095 096	Copy to executable	+	Sequence of commands	Ctrl+S	
898 890	Analysis	+	Constant		GE .)
A2 A7	Help on symbolic name	Ctrl+F1	Binary string	Ctrl+B	
1945	Detach Process		All intermodular calls	-	T
12	Process Patcher		All commands		
e	Analyze This!		All sequences		
5 30	Bookmark	+	All constants		90 94
10 20 30	Code Ripper		All switches		980
49	IDAFicator		All referenced text strings		40 94 98
79	? CALL		User-defined label		AC BØ B4
90 80 80	Script(S)		User-defined comment		BBC
9			10 00	0018	FFC4

011y 会弹出 Found intermodular calls 窗口: (p7)

R Found intermodular calls	
Address Disassembly	Destination
<pre>M0401007 CALL </pre> JMP.&conct iSc. In itCommonCom 00401007 CALL JMP.&kernel32.GetModuleHand 00401022 CALL JMP.&kernel32.ExitProcess> 00401045 CALL JMP.&user32.SetWindowTextA3 00401045 CALL JMP.&user32.SetWindowTextA3 00401045 CALL JMP.&user32.SetWindowTextA3 00401045 CALL JMP.&user32.SetWindowTextA3 00401046 CALL JMP.&user32.SetWindowTextA3 00401046 CALL JMP.&user32.SetWindowTextA3 004010485 CALL JMP.&user32.SetWindowTextA3 004011045 CALL JMP.&user32.SetWindowTextA3 004011045 CALL JMP.&user32.SetWindowTextA3 004011045 CALL JMP.&user32.SetWindowTextA3 004011045 CALL JMP.&user32.SetWindowTextA3 00401140 CALL JMP.&user32.SetWindowTextA3 00401140 CALL JMP.&user32.SetWindowTextA3 00401180 CALL JMP.&user32.SetWindowTextA3 00401180 CALL JMP.&user32.SetWindowTextA3 00401180 CALL JMP.&user32.SetWindowTextA3 00401182 CALL JMP.&user32.SetWindowTextA3 00401242 CALL JMP.&user32.SetWindowTextA3 00401240 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401280 CALL JMP.&user32.SetWindowTextA3 00401396 C	Thitis DPU selection: remeis: BiogostParanA ternelS2. ExitProcess wser92. LoadBitmapA wser93. SendDigItemHessageA wser93. SetDigItem wser93. SetUlgItem wser93. SetUlgItem wser93. SetBitmapA gdi32. CreateS0 LidBrush gdi32. CreateS0 LidBrush gdi32. CreateS0 LidBrush gdi32. CreateS0 LidBrush wser93. SetBkColor gdi32. SetBkColor gdi32. SetBkColor gdi32. SetBkColor gdi32. SetBitmapA wser93. SetWindowTextA wser93. SetWindowTextA wse
*	45

通常我做的第一件事是点一下"Destination",将列出的函数按字母顺序进行排序(而不是按地址排序): (p8)

R Found	d intermodular calls	Sec Andrewson
Address	Disassembly	Destination
00401000	CALL & JMP. & comet 132. In it Common Con	(Initial CPU selection

现在,如果你看第三列的话,你可以看到该 crackme 调用的所有 API: (p9)

R Found intermodular calls		*
Address Disassembly	Destination	
0040110E CALL JMP. &gdi32. CreateSolidBrush 00401141 CALL JMP. &gdi32. CreateSolidBrush 00401186 CALL JMP. &gdi32. CreateSolidBrush 00401350 CALL JMP. &gdi32. CreateSolidBrush 00401350 CALL JMP. &gdi32. CreateSolidBrush 00401365 CALL JMP. &gdi32. CreateSolidBrush 00401365 CALL JMP. &gdi32. CreateSolidBrush 00401366 CALL JMP. &gdi32. CreateSolidBrush 00401366 CALL JMP. &gdi32. CreateSolidBrush 0040126 CALL JMP. &gdi32. CreateSolidBrush 0040127 CALL JMP. &ger32. DialogBoxParamF 00401285 CALL JMP. &ger32. EndDialog2 00401285 CALL JMP. &ger32. EndDialog2 00401029 CALL JMP. &ger32. EndDialog2 00401029 CALL JMP. &ger32. EndDialog2 00401027 CALL JMP. &ger32. EndDialog2 00401026 CALL JMP. &ger32. EndDialog2 00401026 CALL JMP. &ger32. EndDialog2 00401026 CALL JMP. &ger32. EndDigitem/ 00401026	Jessilian gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush user32. DialogBoxParamA user32. DialogBoxParamA user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. GetDlgItem user32. GetDlgItem user32. GetDlgItem user32. LoadBitmapA user32. LoadBitmapA user32. LoadBitmapA user32. LoadBitmapA user32. LoadBitmapA user32. SendDlgItemMessageA user32. SendMessageA user32. SendMessageA user32. SendMessageA user32. SetBkColor gdi32. SetBkColor gdi	
*		1

这是一个小程序,所以调用的函数不是那么多。大部分的程序都有数百个。 不过通过这个列表,你可以了解到一个二进制文件的很多信息。你可以发现它 用一个对话框作为主窗口。它载入了一个自定义的位图。它修改了对话框中的 一些颜色。

在更大些的应用中,这个窗口的价值更高,因为它能告诉你这些事情:1) 是否有注册表相关 API 被调用用来存储和获取信息?是不是有 API 呼叫网站来 验证我们确实注册了?3)有没有读写一个可能存有注册码文件的 API?当我们 研究一个加壳的二进制文件时,这个窗口将更加重要(这个后面讨论。)。

尽管如此,有几个特定 API 逆向工程师总是会留意,因为这几个在保护机制中用的比较多。包括:

DialogBoxParamA GetDlgItem GetDlgItemInt GetDlgTextA GetWindowTextA GetWindowWord

LoadStringA IstrcmpA

wsprintfA

MessageBeep MessageBoxA MessageBoxExA SendMessageA SendDlgItemMessageA

ReadFile WriteFile CreateFileA

GetPrivateProfileIntA WritePrivateProfileStringA GetPrivateProfileStringA

不幸的是,这里没有包括你可能遇到的所有 API,不过幸运的是,大部分应 用使用下面的其中一个:

GetDlgItemTextA GetWindowTextA IstrcmpA GetPrivateProfileStringA GetPrivateProfileIntA RegQueryValueExA WritePrivateProfileStringA GetPrivateProfileIntA

如果你专注这 8 个 API 的调用,你就可以处理绝大多数的实例。还有别忘了, "Get help on symbolic name" 是可以给你提供帮助的。

现在,在 011y 查找出的 crackme 的调用列表中往下看,在那个简短的列表 中有两个 API:

GetDIgItem 和 GetDIgItemTextA

这两个函数是用来获取输入到对话框中文本框的文本。好吧,在我们的教程中,这只可能说明一件事,获取我们输入的密码。我们想要做的是,不管什么时候只要 011y 遇到两者中的一个就暂停。方法是,选中你想要关注的 API 那行,右键然后选择"Set breakpoint on every call to ____",这里的____是 API 的名称 (这里是 GetD1gItem): (p10)

ALL KUMP ALL KUMP ALL KUMP	.&user32.EndDialog> .&user32.EndDialog> .&kernel32.ExitProcess>	Wser32.EndDialog Wser32.EndDialog kerne132.ExitProcess	_
ALL (JMP) ALL (JMP) ALL (JMP)	.&user32.GetDlgItem> .&user32.GetDlgItemText .&kernel32.GetModuleHan	Follow in Disassembler	Enter
ALL CUMP	.&user32.LoadBitmapA> .&user32.LoadBitmapA>	Toggle breakpoint	F2
ALL CUMP	.&user32.LoadBitmapA> .&user32.LoadBitmapA> .&user32.LoadIcopA>	Conditional breakpoint	Shift+F2
ALL CUMP	.&user32.ReleaseCapture .&user32.ReleaseCapture	Conditional log breakpoint	Shift+F4
ALL CUMP	.&user32.SendDigitemnes .&user32.SendDigitemMes .&user32.SendDigitemMes	Help on GetDlgItem	Ctrl+F1
ALL KUMP ALL KUMP	.&user32.SendDlgItemMes .&user32.SendMessageA> .&user32.SendMessageA>	Set breakpoint on every call to GetDlgItem	
ALL CUMP	.&user32.SendMessageA> .&gdi32.SetBkColor>	Set log breakpoint on every call to GetDlgItem	
ALL COMP	.%gdi32.SetBkColor> .%gdi32.SetBkColor> .%gdi32.SetBkColor>	Set breakpoint on every command	
ALL CUMP ALL CUMP ALL CUMP	.&gdi32.SetBkMode> .&gdi32.SetBkMode> .&gdi32.SetTextColor> &gdi32.SetTextColor>	Set log breakpoint on every command	
ILL CUMP	.%gdi32.SetTextColor> .%gdi32.SetTextColor>	Copy to clipboard	÷
ALL COMP ALL COMP ALL COMP	.&user32.SetWindowlextH .&user32.SetWindowTextF .&user32.SetWindowTextF	Sort by	- E
ALL CUMP	.&user32.SetWindowTextF .&user32.SetWindowTextF &user32.SetWindowTextF	Appearance	E.
ALL CUMP	.&user32.SetWindowTextA)	Wser32.SetWindowTextA	

现在,我们看到011y已经在该行设置了一个BP: (p11)

R Found intermodular calls 📃 🗷			
AddressDisassembly0040114ECALL(JMP. &gdi32, CreateSolidBrush00401141CALL(JMP. &gdi32, CreateSolidBrush00401350CALL(JMP. &gdi32, CreateSolidBrush00401350CALL(JMP. &gdi32, CreateSolidBrush00401390CALL(JMP. &gdi32, CreateSolidBrush00401390CALL(JMP. &gdi32, CreateSolidBrush00401390CALL(JMP. &gdi32, CreateSolidBrush00401390CALL(JMP. &user32, DialogBoxParamP00401292CALL(JMP. &user32, EndDialog)00401292CALL(JMP. &user32, EndDialog)00401295CALL(JMP. &user32, EndDialog)00401295CALL(JMP. &user32, EndDialog)00401295CALL(JMP. &user32, EndDialog)00401295CALL(JMP. &user32, GetDlgItemTextP00401097CALL(JMP. &user32, LoadBitmapA)00401097CALL(JMP. &user32, LoadBitmapA)00401097CALL(JMP. &user32, LoadBitmapA)00401097CALL(JMP. &user32, LoadBitmapA)00401096CALL(JMP. &user32, SendDigItemTextP00401097CALL(JMP. &user32, LoadBitmapA)00401096CALL(JMP. &user32, SendDigItemTextP00401097CALL(JMP. &user32, SendDigItemTextP00401096CALL(JMP. &user32, SendDigItemTextP00401097CALL(JMP. &user32, SendDigItemTextP00401096CALL(JMP. &user32, SendDigItemTextP00401097CALL(JMP. &user32, SendDigItemTextP <td>Destination dd 92. CreateSolidBrush dd 92. CreateSolidBrush dd 92. CreateSolidBrush dd 92. CreateSolidBrush gd 92. CreateSolidBrush gd 92. CreateSolidBrush gd 92. CreateSolidBrush user32. DialogBoxParamR user32. DialogBoxParamR user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. CoadBitmapR user32. LoadBitmapR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendHessageR user32. SendHessageR user32. SendHessageR user32. SetBKColor dd 32. SetBK</td>	Destination dd 92. CreateSolidBrush dd 92. CreateSolidBrush dd 92. CreateSolidBrush dd 92. CreateSolidBrush gd 92. CreateSolidBrush gd 92. CreateSolidBrush gd 92. CreateSolidBrush user32. DialogBoxParamR user32. DialogBoxParamR user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. EndDialog user32. CoadBitmapR user32. LoadBitmapR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendDigItemMessageR user32. SendHessageR user32. SendHessageR user32. SendHessageR user32. SetBKColor dd 32. SetBK		
*			

我们也想在另一个 API GetD1gItemTextA 那暂停,那么点击选中它,右键 然后和前面一样进行操作: (p12)

R Found intermodular calls		
Address Disassembly	Destination	
0040114E CHL CHP & gdi32. CreateSolidBrush 0040114E CALL CHP & gdi32. CreateSolidBrush 0040118A CALL CHP & gdi32. CreateSolidBrush 0040118A CALL CHP & gdi32. CreateSolidBrush 0040138D CALL CHP & gdi32. CreateSolidBrush 0040138D CALL CHP & gdi32. CreateSolidBrush 00401383 CALL CHP & gdi32. CreateSolidBrush 00401289 CALL CHP & guser32. EndDialog 00401289 CALL CHP & & & ser32. EndDialog 00401387 CALL CHP & & & ser32. EndDialog 00401387 CALL CHP & & & & & & & & & & & & & & & & & & &	gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush gdi32. CreateSolidBrush user32. DialogBoxParamA user32. DialogBoxParamA user32. EndDialog user32. LoadBitmapA user32. LoadBitmapA user32. LoadBitmapA user32. SendDigItemMessageA user32. SendMessageA user32. SetBKColor gdi32. SetBkMode gdi32. SetBkMode gdi32. SetTextColor gdi32. SetBkMode gdi32. S	
*		

现在,不管什么时候只要 011y 遇到了对这两个 API 的调用,它都会断下来 (在调用被执行前)。咱们来试试看。重启 crackme 并运行。011y 会断在对 GetD1gItem 的调用处: (p13)

004040000	50 50010000		
004010HF	. ES 80040000	CHEL SUMP.&user32.LoadIconH2	LoadiconH
004010B4	. 50	PUSH EAX	[Param = 0
004010B5	. 6A 01	PUSH 1	WParam = 1
004010B7	68 8000000	PUSH 80	Message - WM SETICON
agagiaper	EE7E 00	DUCU DUDD DTD CC. FEDDAOT	blind = 980C9
00401000	, , , , , , , , , , , , , , , , , , , ,	POST DWORD FIN SOULEDFIELD	10010 - 96000
904010BF	. E8 B2040000	CHLL KUMP.&user32.SendmessageH>	• Sendnessager
00401004	. 6A 6B	PUSH 6B	ControlID = 68 (107.)
00401006	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hund = 00090DC8 ('TDC [#4]',class='#32770')
CONTRACTORS	. E8 84040000	CALL KUMP.&user32.GetDlaItem>	GetDlaIten
994919CE	93 80304000	MOU DWORD PTR DS: [4030801.FOX	
88481802	40 12	DICH 12	
00401000		DUCH Charlenet 00400040	
00401005	. 68 67304000	FUSH Crackmeb.00403067	
004010DH	. E8 3C040000	CHLL Crackme6.0040151B	The second
994919DF	. 68 69304000	PUSH Crackme6.00403069	Text = "V'z}/fa\x7Fz{/\x7Fcintit"
994919F4	. FE35 80304000	PUSH DWORD PTR DS: [403080]	hund = NULL
004010F0	F8 80040000	COLL (JMP. Sucer32 SettilindouTevt0)	SetMindouTevtB
004010EF	60 12	DUCU 10	-Decontradowienten
004010EF	, OH 12	PUOL Construct Contenance	
004010F1	. 68 69304000	PUSH Urackmee.00403069	

现在,因为我们还没有输入任何内容,我们对GetD1gItem取到了什么东西不感兴趣,好咱们继续(F9): (p14)



如果你看看周围,你会注意到我们已经来到正确的位置。搞笑的是,我们 起初搜索字符串的时候,没有一个是这些字符串中的。

三、破解应用

咱们快速浏览下附近的...。我们注意到有一个跳转(JB)跳过了第一个 "ACCESS DENIED",所以我们关注一下它: (p17)

00401261	. FF75 08	PUSH DWORD PTR SS: LEBP+81	hWnd = 00090DC8 ('TDC [#4]',class='#32770')
00101261	. E8 EF020000	CALL (JMP.&user32.GetDigItemTextA)	GetDigItenTextA
00401269	. 83F8 ØB	CMP EAX, 0B	
0040126C	. 72 10	JB SHORT Crackme6.0040127E	
0040126E	. 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
88401273	. FF35 80304000	PUSH DWORD PTR DS: [403080]	hWnd = 00120DC2 (class='Edit',parent=00090DC8
00401279	. E8 FE020000	CALL <jmp.&user32.setwindowtexta></jmp.&user32.setwindowtexta>	SetWindowTextA
9940127E	> +8500	TEST EAX, EAX	Crackme6.0040300F
00401280	.~ 75 10	UNZ SHORT Crackme6.00401292	
88481282	. 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"

有一个跳转(JNZ)跳过了第二个坏消息,所以我们也将其加入关注名单。 然后我们就会直接穿过到达好消息,所以基本上我们想要确保我们跳过了这两 个跳转: (p18)

0040125A 0040125F 00401261 00401269	. 68 5D304000 . 68 6B . FF75 08 . ES EF020000 . 83F8 0B	PUSH Crackme6.0040305D PUSH 68 PUSH 58 PUSH DWORD PTR SS:[EBP+8] CALL (JMP.&user32.GetDigItemTextA) CMP EAX.08	Buffer = Crackme6.0040305D ControllD = 6B (107.) hWnd = 00090DC8 ('TDC [#4]',class='#32770') GetDigitemTextA
0040126C 0040126E 00401273 00401279 00401279 0040127E	 72 10 68 00304000 FF35 80304000 E8 FE020000 8500 2500 	UB SHORT Crackme6.0040127E PUSH Crackme6.00403000 PUSH DWORD PTR DS:[403080] CALL <ump.&user32.setwindowtexta> TEST EAX.EAX UN2 SUPT Crackme6 00401293</ump.&user32.setwindowtexta>	Text = "ACCESS DENIED!" hWnd = 00120DC2 (class='Edit',parent=00090DC8) SetWindowTextA Crackme6.0040300F
00401282 00401282 00401287 00401280 00401292 00401293 00401298 00401298 00401295	 68 00304000 FF35 80304000 E8 EA020000 50 68 50304000 E8 84010000 0000 0000 	UNE SHORT Erackmed @dd8200 PUSH DWORD PTR DS:[403080] PUSH DWORD PTR DS:[403080] CALL (JMP.&user32.SetWindowTextA) PUSH ERX PUSH Crackmed.0040305D CALL Crackmed.0040305D OR ERX.ERX	Text = "ACCESS DENIED!" hund = 00120DC2 (class='Edit',parent=00090DC8) SetWindowTextA Crackme6.0040300F Crackme6.0040300F
004012A1 004012A6	68 0F304000 FF35 80304000	PUSH Crackme6.0040300F PUSH DWORD PTR DS:[403080]	Text = "ACCESS GRANTED!" hund = 00120DC2 (class='Edit',parent=00090DC8)

咱们试试,看看咱们是不是对的。再一次运行程序,我们应该断在 GetDlgItemTextA指令处(记住绕过第一个断点): (p19)

00401261 - FF75 08 00421254 - E8 EF020000	PUSH DWORD PTR SS:LEBP+81 CALL (JMP.&user32.GetDigItemTextA)	hWnd = 000A0DCB ('TDC [#4]',class='#32770') GetDlgItemTextA
00401269 . 63F8 08 0040126C . 9 10 0040126E . 68 00304000	JE SHORT Crackme6.0040127E PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401273 . FF35 80304000 00401279 . E8 FE020000 0040127E > 985C0	CALL (JMPA Baser32.SetWindowTextA) TEST EAX,EAX	SetWindowTextA
00401280 .~ 75 10 00401282 . 68 00304000 00401287 . FF35 80304000 00401280 . E8 EA020000	JNZ SHORT Crackme6.00401292 PUSH Crackme6.00403000 PUSH DWORD PTR DS:[403080] CALL KUMP.&user32.SetWindowTextA>	Text = "ACCESS DENIED!" hWnd = 00120DB4 (class='Edit',parent=000A0DCB) SetWindowTextB

因为这是一个 JB 跳转,所以我们需要翻转进位标志位: (p20)

1	Periodic la	00401
	01010 01010	ES SS S

这样就会强制跳转。现在我们将做另一个 TEST, 停在了 401280 处的跳转那。 注意, 我们的密码已经出现了注释列。: (p21)

0040125F 00401261 00401264 00401269	68 68 FF75 08 E8 EF020000	PUSH 68 PUSH DWORD PTR SS:[EBP+8] CALL <uhp.&user32.getdlgitemtexta> CMP F07 00</uhp.&user32.getdlgitemtexta>	ControlID = 68 (107.) hWnd = 000A0DC8 ('TDC [#4]',class='#32770') GetDlgItemTextA
0040126C 0040126E 00401273 00401279 00401279	. → 72 10 . 68 00304000 . FF35 80304000 . E8 FE020000 . 85 FE020000	UB SHORT Crackme6.0040127E PUSH Crackme6.00403000 PUSH DWORD PTR DS:14030600 OALL (JHP.&user32.SetWindowTextA) TEST EAX.EAX	Text = "ACCESS DENIED?" hWnd = 00120DB4 (class='Edit',parent=000A0DC8) SetWindowTextA
00401280 00401282 00401287 00401280 00401292 00401292 00401293 00401298 00401290	 25 IB 68 09304000 F55 80304000 E3 E8020000 56 68 50304000 E8 84010000 E8 84010000 B800 	UNZ SHORT Crackme6.00401292 PUSH Crackme6.00403000 PUSH DWORD PTR DS:(4030801) CALL (JMP.&user32.SetWindowTextA> PUSH CRA PUSH Crackme6.0040305D CALL Crackme6.00401421 OR E0X.E0X	Text = "ACCESS DENIED!" hUnd = 00120DB4 (class='Edit',parent=000A00D8) SetWindowTextA ASCII "12121212121
0040129F 004012A1 004012A6	 ✓ 75 1F 68 ØF304000 FF35 80304000 	UNZ SHORT Crackme6.004012C0 PUSH Crackme6.0040300F PUSH DWORD PTR DS:[403080]	Text = "ACCESS GRANTED!" hUnd = 001200B4 (class="Edit",parent=000A00C3)

我们想要那个跳转实现,因为它跳过了第二个坏消息,所以我们只需要继续单步直到到达 40129F 的 JNZ 指令: (p22)


好,这条指令将会跳过我们的好消息,所以我们想要阻止它跳。你知道该 怎么做: (p23)



现在运行程序 (F9), 看看我们已经成功的破解了程序: (p24)



四、家庭作业

作为一个挑战,试着给这个 crackme 打补丁,基于那个我们已经修改的标志位。在将打过补丁的程序保存后,你应该可以运行它,输入任何密码(少于 11个数字)它都会提示"Access Granted"。记住有几个补丁可以完成这个任务, 所以如果一个不起作用,那就找下一个。

加分题:给 crackme 打补丁,让它接受任意长的密码。

第九章续: "无相关字符串"的解决方案

一、简介

针对第九章的家庭作用,这里我提供几个解决方案。记住,有无数种方法 可以破解这个程序,而这只是一个很小的例子。如果你自己找到了一个方法, 那么恭喜你。如果没有的话,也别着急,我们将分多次解决它。

二、方案一

最简单的一种方法是给程序打补丁,只需要将 40129F 处的 JNZ 指令 NOP 掉就行:

00401292	2	68 50304000	PUSH EHX PUSH Crackme6, 00403050
00401298	1.	E8 84010000	CALL Crackme6.00401421 NOP
00401290		ØBCØ	OR EAX, EAX
0040129F	. 14	75 1F	JNZ SHORT Crackme6.004012C0
004012A1		68 0F304000	PUSH Crackme6.0040300F
004012A6		FF35 80304000	PUSH DWORD PTR DS: [403080]
004012AC		E8 CB020000	CALL < JMP.&user32.SetWindowTextA>
004012B1		6A 00	PUSH 0
00401283		FF35 80304000	PUSH DWORD PTR DS: [403080]
004012B9		E8 88020000	CALL (JMP.&user32.EnableWindow)
004012RE		EB 10	IMP SHORT Crackmed 00401200

这会强制程序每一次都直接空降到好消息那。

三、方案二

另一个可行性方案是,让EAX一直等于 0,将那个检测密码的 CALL 用 MOV EAX, 0 替换掉:

00401230 00401292 00401293 00401293	.>.	50 50 68 5D304000 88 0000000	PUSH EAX PUSH Crackme6.00403050 MOV EAX.0	ASCII "12121212"
0040129D		ØBCØ	OR EAX, EAX	
0040129F	•*	75 1F	JN2 SHORT Crackme6.004012C0	THE MOOFER CONTEN
004012H1		58 0F304000	PUSH LTACKMES, MAANSMAR	TELEVIT - MULTIESS GRUNTED
004012H6		FF 55 60304000	COLL Z INC Assemble at 0040120D	X 155-
664612RL	•	60 00	PIGE 0	
004012B3		FF35 80304000	PUSH DWORD PT	res:
00401289	1.1	E8 88020000	CALL KUMP, &us	
004012BE		EB 10	JMP SHORT Crail MOV Bes. U	
004012C0	>	68 00304000	PUSH Crackme6	ID!'
004012C5		FF35 80304000	PUSH DWORD PT	1551
004012CB		E8 AC020000	CALL (JMP.&us)	the second secon
00401200	>	6A 0E	PUSH BE IV Fill with NUP's	Assemble Cancel
004012D2		68 00304000	PUSH Crackme6	There and the
00401207		E8 3F020000	CHLL Crackmeb	
00401200		6H 0F	PUCH Creating COMPAGE	OCCTI NOCCECC COONTED.
00401202	•	50 0F304000	POSH Crackneb, 60404E4D	HOULT HUUESS GRHMED!

这个基本上将检测密码可行性的 CALL 整个删除了,程序将总是跳转到好消息那。

四、方案三

继续方案二的思想,我们将那个 CALL 留下,在它返回以后,我们再强制 EAX 等于 0。只需要将 OR EAX, EAX 替换成 XOR EAX, EAX 即可:

00401200	> 50	EHOZOOOO	PUSH EAX
88481293	. 68	5D304000	PUSH Crackme6.0040305D
00401298	E8	84010000	CALL Crackme6.00401421
8640129D	330	0	XOR EAX, EAX - Force EAX = 0
0040129F	.v 75	1F	JNZ SHORT Crackme6.004012C0
994012A1	. 68	0F304000	PUSH Crackme6.0040300F
004012A6	. FF3	5 80304000	PUSH DWORD PTR DS: [403080]
004012AC	. E8	CB020000	CALL (JMP.&user32.SetWindowTextA)
004012B1	. 6A	00	PUSH 0
004012B3	. FF3	5 80304000	PUSH DWORD PTR DS: [403080]
odc takaa	EO	000000000	COLL / IMP Anasy22 Eachtablishers

我喜欢这个解决方案,对此还有一定的讽刺意味(你只打了一个字节的补 丁,只加了一个字母☺)。(译者注:我觉得讽刺意味应该是,一个程序的保护机 制,加一个字母就搞定了,确实挺讽刺的。)

五、加分题

我希望加分题没有给你带来烦恼。移除密码长度限制的最简单的方法是替 换掉原始的跳转,如果密码太长的话,就用一个直接跳转到好消息的 JMP 替换 掉原始跳转。

10101261	. E8 EF020000	CALL (JMP.&user32.GetDigItemTextA)	4 GetDlgItemTextA
00401269	. 83F8 0B	UMP EHX, UB	
0040126C 0040126E 00401273 00401279 00401279	 68 00304000 FF35 80304000 E8 FE020000 85C0 	PUSH Crackmed.00403000 PUSH DWORD PTR DS:[403080] CALL CALL CHL ser32.SetWindowTextA TEST FAX.FQX 	Text = "ACCESS DENIED!" hWnd = 000D083A (class='Edit' SetWindowTextA
00401280 00401282 00401287 00401287	.~ 75 10 . 68 00304000 . FF35 80304000 . E8 EA020000	UN2 SHORT Crackme6.00401292 PUSH Crackme6.00403000 PUSH DWORD PTR Ds:14030801 CALL <ump.&user32.setwindowtexta></ump.&user32.setwindowtexta>	Text = "ACCESS DENIED!" hund = 000D083A (class='Edit' SetWindowTextA
00401292 00401298 00401298 00401290	 50 68 5D304000 E8 84010000 08C0 	PUSH EHX PUSH Crackme6.0040305D CRLL Crackme6.00401421 OR EHX,EHX	ASCII "12121212"
0040129F 004012A1 004012A6 004012A6 004012B1 004012B3 004012B3	 ✓ 75 1F ◆68 ØF304000 FF35 80304000 E8 CB020000 6A 00 FF35 80304000 FF35 80304000 F835 80304000 	UN2 SHORT Crackme6.004012C0 PUSH Crackme6.0040300F PUSH DWORD PTR DS:(403080) CALL <ump.&user32.setwindowtexta> PUSH 0 PUSH 0 PUSH DWORD PTR DS:[403080] CALL <ump.&user32.frablewindow></ump.&user32.frablewindow></ump.&user32.setwindowtexta>	Text = "ACCESS GRANTED!" hund = 0000083A (class='Edit' SetWindowTextA Enable = FALSE hund = 0000083A (class='Edit' FnableWindow
004012BE 004012C0 004012C5 004012C5 004012C8 004012D0	 → EB 10 → 68 00304000 → FF35 80304000 → E8 AC020000 → 6A 0E → 60 00204000 	UMP SHORT Crackme6.00401200 PUSH Crackme6.00403000 PUSH DWORD PTR DS:[403080] CALL (JMP.&user32.54010dowTextA) PUSH 0E	Text = "ACCESS DENIED!" hUnd = 000D083A (class='Edit' SetWindowTextA

这个相当的巧妙(在下一章中有更好的办法),不过确实有用。这样做的好 处是,不仅修补了程序让它总是接受你的密码,而且和上面的方法不一样的是, 它也移除了对密码的所有限制。

第十章:打补丁的层次级别

一、简介

本章我们会讨论给二进制文件打补丁的不同的层次级别。本章有点长而且 详细,涵盖较多的背景知识,有些还不简单。我想给你展示一个深入分析二进 制文件的例子,以及它需要什么。你可能大部分都不能够理解,不过它会给你 一个非常好的总览逆向工程的一个好的机会。这样在将来的教程中,你会有一 个参考框架。我们用上一章的那个 crackme 来研究,就是"TDC"写的 Crackme6, 相关下载中包括的有。

你可以在教程页现在相关文件和本文的 PDF 版本。

总之,从上一章我们就知道这个 crackme 不是一个硬骨头,不过这里我打算对它做高级分析,也为将来的教程做准备。现在坐好,准备一杯咖啡/香烟/ 巧克力棒/注射器,任何能让你坚持下去的东西都行,那咱们开始了.....。

二、破解的等级

逆向工程领域(尤其是破解)中关于打补丁的不同级别有几个不成文的规则。基本上可以分为四级(我保证,至少有一半的逆向工程师会因为那个数字和我吵起来☺)。当然,因为缩写神马的听起来都不错,所以我给四个级别的每 一个都想了一个缩写。事不宜迟,下面就是补丁级别的介绍及具体意义:

级别一:LAME

LAME 方法, 就是 Localized Assembly Manipulation and Enhancing, 这 个方法目前我们已经学习过。意思是找到代码中的第一个魔术 比较/跳转 指令, 然后将其 NOP 掉或强制它跳转。到目前为止,这个方法都很神奇的好用。当然, 我们都是在简单的 crackme 上做实验(有一半都是我专门为教程写的)。不幸的 是,外面的大部分应用都不会这么简单。用 LAME 方法,有许多东西都会出问题, 包括:

1、许多许多的应用会在程序的不同地方对程序是否已经注册进行检测,所以如果你仅仅打了一个补丁的话,并不意味着就没有其他的地方需要打补丁(我想我见过最多的分布检测点是 19 个)。并且有时候这些其他的检测点并不会起作用,除非某些特定的事件发生,所以你会发现自己又得回头对同一个程序进行搜索,以找到替代检测点并打补丁。

2、许多程序也会采用多种特别的技巧以避免 比较/跳转 指令组合的暴露。 无论是在 DLL 中执行、在另一个线程中执行还是以多态的方式修改,都有许多 种方法来实现。 3、有时候你将会修补大量的代码。你有可能会给七个检测点打补丁,将其他的检测点 NOP 掉等等。这会让你头昏脑涨的,而且对你来说也不是那么的优雅。

4、使用该方法你不需要学习太多东西,如果你正在阅读本系列教程,很可 能是因为你对相关主题感兴趣并有学习的欲望。

尽管如此,有时候最优雅的解决方案,通常也是最简单的解决方案,仅仅 是一个 比较/跳转 指令组合的补丁即可,所以别让我走错路并认为你不应该使 用它。事实上,我逆向过的许多程序中,我猜大概有 25-40%就是用像这样的一 个简单补丁搞定的。所以它是一个强大的方法⁽²⁾。

级别二: NOOB

NOOB方法,也就是 Not Only Obvious Breakpoints 方法,通常要比 LAMP 方法更深入一步。它通常涉及到要单步步入到 比较/跳转 指令组合的前面的那 个 CALL,以了解是什么让 比较/跳转 指令组合决定走这条路的。这样做的好处 是,你将有更多机会捕获到调用相同方法进行注册验证的其他部分代码,所以 给一处打补丁就可以真正的补好几处,也就是所有调用相同注册验证方法的那 几处。当然,该方法也有几个缺点,比如:

1、有时该方法用于超过一个注册验证的程序。例如,有一个用于比较两个 字符串的通用函数,它返回真或假。在我们序列号匹配的案例中,这就是打补 丁的地方,不过同样的方法被调用以比较两个不同的字符串,并且我们已经将 其打过补丁以让它始终返回 true(或者视情况也有可能是 false)结果会怎样?

2、该方法需要更多的时间和实验,以判定能够返回正确值的最好选择是什么。这个需要时间和技巧。

这是本章中我们将会用到的第一个方法。

级别三: SKILLED

SKILLED 方法,也就是 Some Knowledge In Lower Level Engineered Data 方法,和 NOOB 方法有点像,除了它需要你仔细审查程序并且将其完全逆向以研 究到底是什么情况。这样做有许多好处,比如理解所使用的任何技巧(像在内 存中存储变量以便于后面获取),提供更多的打补丁的地方以更简单并且少侵 入,从内部了解程序是如何工作的。它也给了你作为一个逆向工程师在将来会 用到的许多知识,更不用说你的汇编语言技能。

该方法的主要缺点是,它更难并且需要更多的时间。我建议你至少找几个 程序试试这个方法,因为没有什么能够比花时间深挖代码、堆栈、寄存器以及 内存能够让你成为更好的逆向工程师,尝试去感受下作者曾经试过的。本章的 最后我们将会用到这个方法。

级别四: SK1LL\$

思考下破解的圣杯, Serial Keygenning In Low-level Languages, Stupid 意味着你不仅要仔细研究并且准确找出注册进程是如何执行的,还要重建它。 这就得能够让新用户随意输入任何用户名, 然后 keygen 者的代码能够算出对于 该二进制文件管用的序列号。制作一个 keygen 的通常的方法是用程序自身的代 码来对付它, 意思是拷贝作者用来解密序列号的代码并用它来进行加密。这些 代码通常放置在某种专门用来接收被拆分的代码的程序中(它提供有 GUI 等类 似功能)。

skill\$(译者注:小标题中的第三个字母为1,这里又变成i,我不知道是 不是作者故意的)的最高境界是,如果不能够从应用中提取代码,就必须自已 编写代码来提供可用的序列号。意识是你必须完全理解程序是如何解密序列号 并将其与你输入的进行对比。你必须自己写程序来完成相同的功能,仅在逆向 领域中,有很多次是用汇编语言写的。

很明显,该方法的主要缺点是 skill\$的复杂难懂。

那么,鉴于我们对逆向工程级别的新的理解.....

三、用级别二来研究应用程序

重启应用并运行。给 GetDlgItemTextA 设置断点(参见上一章), 输入密码 (我输入的是"12121212")然后点"Check", Olly 就断在了 GetDlgItemTextA:



咱们来看看 GetDlgItemTextA:



需要重点注意的是:其中一个参数是一个指向缓冲区的指针,该缓冲区是 用来存储密码的(1pString)。返回值保存在 EAX 中,它保存的是字符串的长度:

Return Values

If the function succeeds, the return value specifies the number of characters copied to the buffer, not including the terminating null character.

If the function fails, the return value is zero.

你在 40125A 处看到那个指向字符串缓冲区的指针,它是 40205D (011y 加 了一个注释 "Buffer=",因为它能够猜测参数。译者注:作者写的是 40205D,

不过看图片实际上是 40305D,估计作者弄错了)。意思是该函数会拷贝我们的对 话框文本到一个以 40305D 开始的 buffer 中,将返回的字符串的长度保存在 EAX 中。所以,在本例中,我们输入的密码"12121212"将被获取到,返回的密码 长度保存在 EAX 中,这里是 8。现在,如果你看接下来的两行,你会发现这个值 与 0x0B(十进制的 11)进行比较,并且如果 EAX 比它小的话程序就会跳转。真 正的意思是,如果我们的密码长度(EAX)小于 0x0B(11个数字)就会跳转。 注意如果我们不跳的话,我们就会直接到坏消息那,所以实际上,这就意味着 我们的密码长度必须比 11小:



看吧!! 咱们已经了解了一些东西了,我们的密码最多只能有 11 个数字☺。 因为我们的密码少于 11 个数字,所以咱们继续并让跳转实现。(如果你输入的 密码大于 11 个数字,重启应用然后再输入一个小于 11 个数字的密码,再单步 到我们所在的位置。)

00401273 00401279 00401275	 FF35 80304000 E8 FE020060 85C0 	PUSH DWORD PTR DS:[403080] CALL KUMP.&user32.SetWindowTextA> TEST EAX.EAX	hWnd = 00120DB6 (class='Edit',pa SetWindowTextA
00401280	.~75 10	UNZ SHORT Crackme6.00401292	and the second of the second s
00401282 00401287 00401280	+ 68 00304000 FF35 80304000 E8 EA020000	PUSH Crackme6.00403000 PUSH DWORD PTR DS:[403080] CALL (JNP.&user32.SetWindowTextA)	Text = "ACCESS DENIED!" hWnd = 00120DB6 (class='Edit',pa SetWindowTextA
00401292 00401293 00401298 0040129D 0040129D	 68 5D304000 E8 84010000 08C0 75 1F 	PUSH Crackme6.0040305D CALL Crackme6.00401421 OR EAX, EAX UNZ SHORT Crackme6.00401200	ASCII "12121212"

接下来你注意 EAX 的值,它仍然保存着密码的长度,并被测试是否为 0,如 果不是 0 的话,就跳过第二个坏消息。那么现在我们知道了第一个坏消息是当 我们的密码长度小于 11 时显示,第二个坏消息是在密码为空的时候显示。

注意,在跳转实现的接下来的两行,是从 401282 开始的, PUSH EAX (密码 长度),地址 40305D (存储密码的 buffer)入栈。看看堆栈,可以看到确实如此:



首先要注意的是(在地址 18FAB0 处)是长度(8)被压入栈,其次是在地址 18FAAC 处的地址 40305D 被压入栈,011y 也向我们显示了"12121212"也就是我们的密码。现在我们知道了,我们的密码是存储在内存的 40305D 处。这一点在后面会很重要。后面,011y 会将这两个值叫做 ARG.1 和 ARG.2,因为它们是传递给函数的参数。这两个值入栈以后,我们就可以调用 401298 处的主要注册程序了(我们之所以知道这个,是因为所有重要的 比较/跳转 指令组合的前面都有个 CALL,所以它的结果将决定我们是跳到好消息还是坏消息):

00401292	> 50	PUSH EAX	ASCII "12121212"
00401293	- 68 5D304000	PUSH Crackme6.0040305D	
0040129D	. 08C0	OR EAX, EAX	
0040129F	.~ 75 1F	JNZ SHORT Crackme6.004012C0	
004012A1 004012A6 004012AC	68 0F304000 FF35 80304000 FF35 80304000 F8 CB020000	PUSH DWORD PTR DS: [403080]	<pre>Text = "ACCESS GRANTED!" hWnd = 00120DB6 (class='Edit',parent=000 SetWindowText8</pre>
004012B1	6A 00	PUSH 0	Enable = FALSE
004012B3	FF35 80304000	PUSH DWORD PTR DS: [403080]	hUnd = 00120DB6 (class='Edit'.parent=000

让 011y 就暂停在 CALL 那行,不过要注意 CALL 后面那行,40129D 处指令对 EAX 自身做了 0R 操作(该操作会根据 EAX 是否为 0 来设置 0 标志位),如果 EAX 不是 0 的话就会跳过好消息。这就意味着将在 401298 处调用注册程序,并在某 个时刻在 EAX 中保存一个值并将该值 RETN。返回值将会被检查是否为 0,如果 不是就显示坏消息。所以我们必须保证在这个 CALL 中,当它返回时 EAX 等于 0!! 如果我们能够做到的话,它就是我们需要的唯一一个补丁(密码被限制在 0 到 11 个数字之间也算一个,不过那是一个简单的补丁)。咱们继续,单步步入到 401298 处的注册程序,总览一下:



哇噢,看起来不少呢,尤其是你很可能对汇编语言只是个半吊子。但也不 是不可能。我常用的招是到程序的最后面,我们知道它返回时 EAX 必须等于 0, 看看是什么完成了这项工作以及是什么阻止了它发生,然后再回头用我们的方 法。向下滚动直到你看到函数的 RETN 指令:



这里,我们可以看到,我们肯定是想要在函数返回前避免401510处的指令将 EAX 的值设置为 1。你可以看到有一个红色箭头指向该行(译者注:不是作者 加的箭头,是那个细线的小箭头,在指令的边上),所以该跳转也需要被干掉。现在如果我们向上看看,我们能够看到 EAX 被设置为 0 的地方,也能看到函数 底部将其返回的路径:



如果我们看看 4014FB 那行, EAX 将会被置 0 (对自身做 XOR 操作), 跳转指 令将会跳过 401510 处的坏消息指令,相关的执行流将会返回 EAX 值为 0⁽²⁾。现 在我们跟一跟我们看到的第一个跳转(也就是会跳到 401510 处 MOV EAX, 1 这个 坏指令的跳转),看看从哪跳过来的:

S / 1 / 20	1.5	OFOF OFOOOOO	THE OWNER CONCIENCE
00401470	. 2	0F85 8E000000	UNZ Urackme6.0040151
00401482		89 09000000	MOV ECX,9
00401487	>	8A1401	MOV DL, BYTE PTR DS:
3040148A		321401	XOR DL, BYTE PTR DS:
3040148D		49	DEC ECX
3040148E		67:E3 02	JCXZ SHORT Crackmen
00401491	20	FB F4	MP SHORT Crackmen.
10401493	S.	66-8848 08	MOU CY, WORD PTR DS. I
00401497	1	66-91E1 EEEE	YOR CY GEFEE
36461490	0	66.01E9 0000	CMP CV DOOC
38481401		7E 64	INT CHOPT Cupekwee 6
38481402	• •	0000	MOU CL DUTE DTD DC. I
004014H0		0000 01	MOU OU DUTE DTD DO.
004014H5		8468 01	HOU CH, BYTE PIR DS:1
004014H8		66:8101 9235	HUD UX, 3592
104014HD	20	66:81F9 9HE5	CMP CX, 0E59H
004014B2	.~	75 49	JNZ SHORT Crackme6.8
304014B4		8138 08B0817A	CMP DWORD PTR DS: LEF
304014BA		75 4B	UNZ SHORT Crackme6.0
304014BC	1	66:3309	XOR CX,CX
394014BF	$\boldsymbol{\lambda}$	80F2 0A	FXOR DL.0A
30401402		8301 04	ADD ECX.4
00401405	10	83F9 0C	CMP FCX. 9C
00401408		ZE ES	ULE SHORT Crackmen.
39491400	8	8178 04 028F8D38	CMP DWORD PTR DS. LEG
20401401	1	75 20	INZ SHORT Crackmee 6
20401402	211	9000	MOU CL DI
384814D5		9201	YOP DI CI
38481407		0001	MOU DL CL
20401407	1	0HD1	YOD CL DL
50401409		32UH	MOU OL DUTE OTD DO I
0040140B		00000 00	HOU DL BYTE FIR US:1
004014UE	2	8450 05	HUV DE, BYTE FIR DSTL
004014E1		8601	XUHG UL, DL
004014E3		SOLA BE	CMP DL.08F
004014E6	• Y	75_1F	UNZ SHUKI Grackme6.8
304014E8		80F9 8D	CMP CL,8D
394014EB		-75 23	UNZ SHORT Crackme6.0
304014ED		8078 05 BF	CMP BYTE PTR DS: [EA>
304014F1		75 14	JNZ SHORT Crackme6.0
304014F3		25 FFFF0000	AND EAX, ØFFFF
304014F8	S	66:3300	XOR AX.AX
304014FB	.4	EB 18	JMP SHORT Crackme6.0
304014FD	2	SAD1	MOU DL.CL
204014FF	12	3208	XOR CL. DL
38481581	199	BØ Ø1	MOU AL.1
0401503	6	94 29	ADD AL 20
0401505	100	Sers	MOU CL OL
00401507	S	66+B9 9139	MOU CV 3891
3949159P	6	66-91F1 000F	YOR CY AFOD
20401510	S.	PP 01000000	MOULEON H
30401515	5	50 51666666	POP ENV
20401510	1	EQ	POP ECV
THE REPORT OF A DESCRIPTION OF A DESCRIP			EL A

40147C 就是那个坏跳转。我们想要阻止它跳,否则我们肯定会得到坏消息。 好的,我们现在已经有了关于这段程序的基本的知识,对于级别二的破解我们 就到这里,现在来打个补丁确保 EAX 总是返回 0。你会怎么做呢?我准备将它留 给你来做(这是本章结尾的作业。)。放心好了,我会给你答案的...。不过你要 明白这个级别的补丁已经开始时的补丁要好很多,一是我们只打了一个补丁, 二是如果这段程序被应用中的其他部分调用的话,我们仍然能够获得好消息。

现在,先停一会,考虑考虑你怎么来打这个补丁。记住,EAX 必须返回 0。 我让你做的原因是,有很多很多 NOOB 补丁可以完成这个任务,我想让你开始像 一个逆向工程师那样思考!如果你需要提示,那就看看结尾的作业那一块。如 果你能够解决,那你就是一个真正的 NOOB!!!

当你完成时,就准备转到更详细的分析,继续阅读吧.....。

四、步入到级别三

我知道你仍然是一个初学者,不过我还是想让你尝尝更深层次补丁的感觉。 如果你还没有准备好,或者完全失败了,也别气馁。这里只是给你一个想法。 我们会在将来的教程中学习这一块的所有东西。你可能会问,更加深入代码的 目的是什么,应用程序中调用这段程序的每一个地方都会被打上补丁吗?好吧, 对于新手来说,假如有不同程度的注册会怎么样,比如"Private"、"Corporate", "Enterprise"...。程序可能会根据内部的逻辑来做决定。另一个你想要更深 入学习的原因是,为它做一个keygen。你需要理解代码才能做。现在,咱们开 始打一个 SKILLED 级别的补丁。回到上面程序(译者注: 这里的程序是指那个 验证函数,本章大部分都是这个意思,读者应自己做分别)的起始处,实验一下:



首先,那有一些典型的寄存器的压栈操作以及在栈中为本地变量开辟空间的操作。ECX和EDX中的值被压栈,然后我们就可以在不用覆盖这些寄存器的情况下使用它们(函数返回时会将这些值出栈以将它们还原。译者注:这就是传说中的堆栈平衡,脱壳中ESP定律的原理)。然后我们就到了40142A,这里将栈中(我们输入的密码的地址)的本地变量拷贝到EAX中。如果你看寄存器窗口就会发现EAX的值是地址40305D,也就是我们密码所在的地址。下一行代码:

XOR DWORD PTR DS:[ECX+EAX], 1234567

该行的意思是,将 ECX (它的值是 0)和我们的密码首地址(密码存储在 40305D,记不记得?)相加,然后从该位置取 DWORD (4 字节)数据与十六进制 的 1234567 进行 XOR 操作。因为 ECX 是 0,将其与我们的密码首地址进行相加不 会有任何的影响,所以我们从密码的第一个数字的地址开始处理就行。简单点 来说,这行代码的意思是"取密码的前四个字节与 1234567 进行 XOR 操作,将 新值存储到内存中与我们密码同样的位置"。

我们可以观察到这一过程。首先,要确保我们依然暂停在 40142D 那行,看数据窗口的上面那一块,它会告诉你地址 ECX+EDX (40305D)是什么,以及它存储的值是什么 (32313231),用 ASCII 码表示就是 "2121" (要记得数据存储序列):



现在选中 "DS: [0040305D]=32313231" 这行,右键选择 "Follow in dump (数据窗口中跟随)",然后我们就能看到我们密码当前存储的实际内存内容:



现在,数据窗口显示的就是从 40305D 开始的内存内容。前面 8 个字节就是 我们的密码。记住,我们当前所在的行正准备取该地址的前四个字节(31,32, 31,32),然后与 0x1234567 进行 XOR 操作,之后再将结果存回到该内存区:

Address	He	t di	IMP	1		-	-	1	340	-	2.2		1.00				ASCII
0040305D	31	32	31	32	31	32	31	32	00	00	00	00	56	60	78	70	12121212
00403060	60	40	00	64	6D	17	00	61	00	00	00	60	00	00	00	00	.@.+.\$.0
0040308D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030AD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
094030BD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	•••••
004030DD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030ED	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040310D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040311D 0040312D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040313D	ØØ	00	00	00	00	00	00	00	ØØ	00	00	00	00	00	00	00	
0040314D 0040315D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

咱们继续,单步步过一次,然后你就会看到我们密码的前四个字节已经变了,和 0x1234567 进行 XOR 的结果:

Address	He	; di	AMD	2.4	-	-		-	10	-		10.2	-	122	-	201	ASCII
0040305D	56	77	12	33	31	32	31	32	00	00	00	00	56	60	78	70	Uw\$31212V
9949396D	2F	66	61	7F	78	7B	2F	7F	63	68	6E	70	6A	21	00	00	/faOzt/Ocjnijt
00403070	00	40	00	B4	00	17	00	01	00	00	00	00	00	00	00	00	.@.+. ‡ .0
9040308D	100	90	90	90	90	90	90	90	90	00	90	90	90	90	90	90	
00403090	00	00	00	00	00	00	00	00	00	00	90	00	00	00	90	00	
BOMBOODD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403000	00	00	00	00	00	00	00	00	00	00	00	60	00	00	60	00	******
004030hh	aa	ññ	aa	ñã	aa	ññ	ññ	ññ	80	ññ	ññ	ññ	ññ	ññ	ññ	ññ	
994939ED	00	ññ.	ññ	ññ	ññ	ñ0	ññ	ññ	00	00	ññ	ññ	ññ	ñ0	ññ	ññ	
004030FD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
9949319D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00403110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040312D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

好,继续下一行代码:

00401426 00401428 00401428 00401429 00401420	3309 600007 AUD > 813401 67452301	XOR ECX, ECX XOR EDX, EDX MOV EAX, LARG.11 FXOR DWORD FTR DS: LECX+EAX1, 1234567	Crackme6.0040305D
00401434 00401438 00401438 00401438	- 302401 0E - 83C1 04 - 83F9 08	AND BYTE PTR DS:LECX+EAX] OF ADD ECX, 4 CMP ECX, 8	
00401440 00401442 00401445	33C9 > 8A1401 0050 08	ADD BYTE PTR DS: (EAX+83, DL	

该行是 AND BYTE PTR DS:[ECX+EAX], OE。我们已经知道了 ECX+EDX 的 结果是 40305D,也就是我们以前密码的地址。现在,我们准备按 BYTE 与 0x0E 进行 AND 操作,并将结果存回该地址。这意味在,我们存储在 40305D 的以前的 密码的第一个数字(译者注:这里我感觉作者的表述不太准确,因为存储在 40305D 的是 XOR 后的数据,早不是我们输入的 12121212 了,大家要注意,仔细 观察数据区。)在与 0E 进行 AND 操作后再存回第一个位置。看看数据窗口的帮 助区域已经显示出来了:

004014B4	. 8138 Ø8BØ817A
DS: 0040	305D]=56 ('V')
Address	Heyaump
0040305D 0040306D 0040307D 0040307D 0040309D 0040309D	$\begin{array}{cccccccccccccccccccccccccccccccccccc$

它告诉我们将受到影响的地址是 40305D, 该地址的(当前)值是 56。继续, 单步执行一次, 你会发现第一个数字又变了:

Address	Hee	d	IMP	-												-	ASCII
0040305D	06	77	12	33	31	32	31	32	00	00	00	00	56	60	78	70	+W\$31212V
0040306D	2F	66	61	7F	78	7B	2F	7F	63	6A	6E	70	6A	21	00	00	/fa0zt/0cinii*
0040307D	00	40	00	B4	ØD	17	00	01	00	99	00	00	00	00	00	00	.0.1.\$.0
0040308D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040309D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
804030AD	00	00	00	00	00	00	00	00	00	80	00	00	00	00	00	00	
004030BD	00	00	ØØ.	00	00	00	ØØ.	00	ØØ.	00	ØØ.	00	00	00	00	BB	
304030CD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
aadasahh	aa	aa	00	99	an.	āā	00	aa	an.	aa	00	80	an	aa	00	aa	

现在咱们知道了 0x56 与 0x0E 进行 AND 操作的结果是 0x06 。咱们继续跋涉这段困难的代码:



ECX 增加了4(指向下面的四个字节),并和8进行比较。意思是这个循环 会运行两次,第一次 ECX 等于4,第二次等于8,然后跳出循环。这意味着我们 总共处理了8个字节。所以第二次循环时,我们将影响第二个4字节,也就是 将它们与0x1234567进行 AND 操作。你在单步运行时,注意观察第二个4字节:

Address	Hex du	IMP 1					1000				ASCII
0040305D	06.77	12 33	56 77	12 33	30	00 00	00	56 6	0 7A	70	*w*3Uw*3
0040306D	2F 66	61 78	76 70	GF 11	-53	6A 6E	70	6A 2	1 00	00	/faozt/ocjnijt
0040307D	00 40	00 B4	0D 17	00 01	00	00 00	00	00 0	0 00	00	,0,1,\$,0,,
0040308D	00 00	00 00	00 00	00 00	00	00 00	00	00 0	0 00	00	**************
00403090	00 00	00 00	00 00	00 00	00	00 00	00	00 0	0 00	00	
004030AD	00 00	00 00	00 00	99 99	00	00 00	00	00 0	0 00	00	**************
004030BD	00 00	00 00	00 00	00 00	00	00 00	00	00 0	0 00	00	****************
004030CD	00 00	00 00	00 00	00 00	00	00 00	00	00 0	0 00	00	

它们也会被修改。第五字节也会被再次修改,因为它将与 0x0E 进行 AND 操作。循环结束后,下一条指令也就是 401440 处的指令仅仅是将 ECX 值置 0:

00401464	. 0043 00	HOV CHA, CHING, LO
0040142D	> 813401 67452301	CXOR DWORD PTR DS:[ECX+EAX].1234567
00401424	000401 0F	OND DUTE DTD DC. FECYLEOVI OF
00401404	. 002401 00	HID DITE FIN DOLLEGATEROI, OL
00401438	. 83C1 04	ADD ECX.4
00401400	0000 00	CMP ECV 0
00401400	. 0312 00	CHF ECAYO
0040143E	.^ 75 ED	JN2 SHORT Crackme6.00401420 Erv _ n
00401440	9309	SOR FCX FCX
Cicket Charter		non Lon, Lon
00401442	> 8H1401	MUV DE, BYTE PTR DS:LECX+EHX1
00401445	0050 08	ODD BYTE PTR DS+FE0X+81 DL
00401440	. 0000 00	
00401448	. 41	INC ECX
00401449	384D 0C	CMP FCX FORG 21
00401442		ON ECONETIMETER
00401440	· 75 F4	UNZ SHUKI Crackme6.00401442
0040144F	3309	XOR FCX FCX
COACTAAL	· · · · · · · · · · · · · · · · · · ·	non con, con

现在咱们来看看接下来的几条指令:



首先,我们将我们(旧)密码的第一个(新的)字节拷贝到 DL 中(因为 ECX 再次被置 0,所以我们正在处理的是第一个数字,也就是 EAX 当前指向的数字)。 如果你看寄存器窗口,你会发现第一个字节(0x06)在 EDX 寄存器中:



然后我们将 DL 中的数字与 EAX+8 中的内容相加,也就是 EAX 的第八个字节, 并将结果存回第八个字节:

438	. 8301 04	ADD FCX.4
43B	83F9 08	CMP ECX 8
43F	-^ 75 FD	INZ SHORT Crackme6, 00401420 DL
449	BDA POSS	XOR FCXLECX
442	> 881401	MOV DL. BYTE PTR DS: CECX+EAX
445	. 0050 08	ADD BYTE PTR DS: LEAX+81.DL
448	. 41	INC ECX
449	. 3B4D 0C	CMP ECX. TARG. 21
44C	.^ 75 F4	UD 0000T 0 1 0 00101110
44E	. 3309	With the 8th digit of buffer and store there

这里,我们能够看到那个字节被修改了:

Address	Hea	t di	4mp			100											ASCII
00403050	Ø6	77	12	33	06	77	12	33	06	00	00	00	56	60	78	70	+w\$3+w\$3+
0040306D	2F	66	61	ŹĒ.	2Å	7B	2F	7F	63	SA.	6Ē	ŹĈ.	6Â	21	00	00	/faozf/ocinlit
00403070	00	40	ØØ.	B4	ØD.	19	00	01	00	£	00	00	00	00	00	00	.0.1.4.0
0040308D	ØØ	ØØ	00	00	ØØ	00	00	00	00	0.	90	00	00	00	00	00	
00403090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030AD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030BD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030CD	Ø Ø	ññ.	Ø Ø	ЙЙ.	ЙЙ	Ø Ø	Ø Ø	ØØ.	Ø Ø	Ø Ø	Ø Ø	ÑØ.	Ø Ø	D D	Ø Ø	Ø Ø	

它是将 buffer 中的第一个字节(6)与第八个字节(0)相加,结果得 6(译 者注: 不知道读者注意到没有,上面的图片中箭头明明指的是第九个字节好不 好,那是不是作者弄错了呢? 我觉得是有问题的,因为如果按从 0 开始数,那 么第一个字节就应该是 77,如果不从 0 开始数,那么第八个字节就应该是 33。 那该怎么办呢? 其实不用管第几个了,我们知道定位到该字节是按 EAX+8 的结 果算的,因为 EAX 的值是 40305D,所以 40305D+8=403065。40305D 就是 06 那个 字节,我们往右边数,数到 403065,刚好就是箭头指的那个 06)。如果我们密 码的长度大于 8,这就会让我们密码的第一个字节与我们密码的后面的那个数字 相加,不过因为我们的密码只有 8 位,所以这块内存被设为 0。接下来我们给 ECX 加 1 (因此转移到下一个字节),将其与密码的长度进行比较。这只是查看 我们是否已到达结尾。如果没有,就跳转到循环的开头再执行一次。这基本上 意味在,我们将循环遍历密码的所有数字,每个数字相加然后将结果存储在第 八个内存位置。现在我们明白了为什么密码只能是 11 个数字,所有空间加上 0 终止符一共只能接受 11 个字符。



你单步执行这个循环,就可以观察到内存的变化:

Address ,	Ho	, d	1 Mapi	-	-	_	-	-					-	-			F
0040305E	06	77	12	33	06	77	12	33	8F	00	00	00	56	60	78	70	4
00403060	2F	55	b1	CF.	2H	10	2h	65	63	68	6E	70	68	21	00	00	1
00403070	00	40	80	84	00	00	00	001	00	ê	00	00	00	00	00	00	1
0040309D	00	- 4	00	00	øø	ØØ	00	-	00	00	200	00	00	00	00	20	1
			1.1	-	100	-			10.0	SIL	цто)		2.59	1994	100	1.0	
NINANSHI T	au.	ыя	ии	ии	ИИ	ии	ИИ	88	ññ	ññ	00	ñà	Ø Ø	ññ	00	00	
004030DD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030ED	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030FD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	٠
00403100	00	00	00	00	00	00	00	00	00	00	00	00	00	ña	00	00	

在循环结束后,我们再一次将 ECX 设置为 0,并进入了与第一个循环相似的循环中,这次将每个四字节数据与 0x89ABCDE 进行 XOR 操作。



它也将所有的字节相加,再将结果保存在第九字节。这个循环将会一直执行,直到 ARG.2 等于 0 为止。ARG.2 是我们密码的长度(还记不记得它是调用该 函数前第二个被压入堆栈的?)所以,这些指令将会执行 8 次,密码中的每个 数字一次。在执行完这段代码后,你就会看到最终的结果:

Hadress	He	(O(4mp	1.000	-	-	100	-		-		1000	_	-	-	100	HSCIL
99493950	08	CB	88	3B	08	CB	88	ЗB	84	20	00	00	56	60	78	70	Chre; Die
00403060	2F	66	61	7F	78	7B	2F	7F	63	60	6E	70	6A	21	00	00	/fa0z(
0040307D	00	40	00	6E	05	07	00	01	00	00	00	00	00	00	00	00	.@.n.*·
00403080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040309D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030AD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030BD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
004030CD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
904030DD	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
RR4R3RED	na.	ØØ.	DO.	ØØ.	ØØ.	ØØ.	ØØ.	nn.	A0	ØØ.	ØØ.	DO.	A0	ØØ.	na.	ØØ.	0.00033



运行程序并观察所有的情况是非常非常的重要,因为这能让整个过程更加 清晰很多。花点时间来理解每一行,看看它将做什么,它准备将结果存储在什 么地方。你会发现,它其实不像听起来的那样难:)。别忘了,我们将要在 40147C 处的跳转做出我们的选择。下面我们总结下我们所做的:

1、我们将我们的密码的每一个四字节值与 0x1234567 进行 XOR 操作,再将 结果覆盖回我们的密码。

2、第一个字节与 0x0E 进行 AND 操作, 第五个字节也是一样。

3、然后我们将所有的字节值加起来,将结果存储在第八字节。

4、然后,我们再将 buffer 中的每个四字节值与 0x89ABCDEF 进行 XOR 操作, 再将结果存进这个 buffer。

5、我们再一次将 buffer 中的内容相加,将结果存储在第九个内存位置。

我们已将执行了此 crackme 保护机制魔法的大部分(*啧啧*)。现在我们将载入这两个值(buffer 内存内容的求和),一个在 EAX+8,另一个在 EAX+9,分别载入到 DL、DH,本例中 EDX 的结果就是 842C。然后,我们将这两个值与 42DE 进行比较:



为啥是 42DE 呢?好吧,这很可能是一个硬编码的密码。你思考下,如果你有一个特殊密码,用它来进行整个的 XOR 和 AND 操作,将得到魔数 42DE。我们的例子中,EDX 等于 842C:



我们没有输入那个魔术密码,所以我将实现该跳转,跳到坏消息代码那里:



当然,除非我们给011y帮点小忙:

因为我们不空降,所以 EAX 的值不会被置 1,并且该函数会立即终止。下一步,我们将 ECX 值置为 9,以便于我们访问 buffer 的第九个数字,将第九处内存位置的内容拷贝到 DL 中(这里是 0x2C),对其自身做 XOR 操作(让其等于 0), ECX 减 1 以指向前一个位置,这样做 9 次:



你可能有点疑惑,这没有改变 buffer 中的任何东西呀,那这个函数有什么 意义呢?好吧,咱俩都被迷惑了。看起来它所做的一切都是让 DL 一次又一次的 等于 0,这看起来几乎就是代码中一个圈套(或者是一个错误。)。总而言之, 不管这个代码运行还是不允许,这都没有什么不同,所以它就是死代码。我们 现在来到一组短点的代码,基本上是将 EAX 与 30AC 进行比较:



首先,它将我们前面求的和存在 ECX 中(第九处内存位置是 0x2C,第八处 内存位置是 0x84),将其与 0xEEEE 进行 XOR 操作,再与 30AC 进行比较。因为 ECX 不等于 30AC,所以我们将跳转:

Reg	isters (F	PUI
EAXXX	00403050 0000026A 00008400 00000001 0018F090	Crac

跳转到那里, ECX 再次被置为 1:



这基本上就是第二个密码检测点了。一个没有多少经验的逆向工程师(或 刚好将他/她给难住了)很可能立即就将这个JNZ给打了补丁,原因是上面将 我们转换的密码与 0x42DE 进行了比较。他们可能没有花时间来分析其他代码, 认为这个补丁就够了。不幸的是,这个补丁明显不够,因为应用程序对我们的 密码计算出来的值做了其他更多的操作,并且如果与新值不匹配的话就跳转。 该方法多次被用来作为一种检测技术,检测是否有人尝试给应用程序打补丁: 在没有任何补丁的情况下,如果我们的密码通过了检测并通过了第一个 JNZ,那 我们也应该能通过第二个。如果没有,那么我们就知道有人给第一个打了补丁, 所以我们就知道有人修改了代码。许多情况下,第二个跳转会跳到完全不同的 代码块,有些看起来令人难以置信的复杂,但是实际上又什么都没做,最后就 终止了。这是企图让逆向工程师做一些徒劳无功的事,让攻克保护机制变的更 难。这不是我们想要的,所以我们设置 0 标志位,然后继续,我们遇到了下面 两行代码:

004014A1 004014A3 004014A3	. 275 64 . 8808 . 8868 01	MOU CL. BYTE PTR DS: [ERX]	
004014A8 004014AD	. 66:81F9 9AE5	ADD CX, 3592 CMP CX, 0659A	C
004014B2 004014B4	.~ 75 49 . 8138 0880817A	UNZ SHORT Crackme6.004014FD CMP DWORD PTR DS:[EAX],7A81B008	
004014BA	.∨ 75 4B	JNZ SHORT Crackme6.00401507	

这是将我们密码 buffer 的第一和第二个内存内容拷贝到 CL 和 CH,本例中 是让 ECX 等于 CB08。与 3592 (十六进制) 相加后再与 E59A 进行比较。如果它 不等于该值就跳转:



这个和上面做的事情是一样的。完成了另一个检测,以确保我们是合法的 到达此处。我们明显不想让这个跳转实现,所以我们通过修改0标志位再次的 帮了011y的忙。然后我们顺利通过了另一个检测,这个是从4014A3到4014AD。 通过修改0标志位,我们也跳过了这个JNZ,最终来到了这里:

00401462	.V (5 47	UNC SHUNI CLACKWEDY004014FD
004014B4	. 8138 08B0817H	CMP DWORD PTR DS: LEAX1, 78818008
004014BA	.×75 48	UNZ SHORT Crackme6.00401507
004014BC	. 66:3309	XOR CX,CX
004014BF	> 80F2 0A	FXOR DL.0A
004014C2	. 83C1 04	ADD ECX.4
004014C5	. 83F9 0C	CMP ECX. OC
00401408	A 75 55	LIF SHORT Erackme6, 004014BE
00401400	8178 04 028F8D38	CMP DWORD PTR DS. [EQX+41 388DRE02
00401401	75 30	INZ SHORT Crackwood 00401510
00401402		MOULCE DE
00401405	• 0HUH 00D1	YOP DI CI
00401405	. 0201	MOU DL CL
00401407	• OHD1	HOV DE,CE
00401409	· 320H	YOR CL, DL
004014DB	. 8848 05	NUV CL, BYTE PTR DS:LEHX+51
004014DE	. 8A50 06	MOV DL, BYTE PTR DS: [EAX+6]
004014E1	. 86D1	XCHG CL, DL
004014E8	. SØFA BF	CMP DL, 0BF
004014E6	.∨ 75 1F	JNZ SHORT Crackme6.00401507
004014E8	. 80F9 8D	CMP CL,8D
004014EB	.~ 75 23	UNZ SHORT Crackme6,00401510
004014ED	. 8078 05 BF	CMP BYTE PTR DS: LEAX+51.0EF
004014F1	× 75 14	UNZ SHORT Crackme6, 00401507
094914F3	25 FEFF0000	AND FAX MEEEE
004014F8	66+3300	YOR OX OX
004014FP	- FB 18	IMP SHORT Crackwee 00401515
004014ED	N Pont	MOULDL CL
004014EE	0001	
00401501	- 020H	MOU OL 1
00401501	. 00 01	HOU HE, I
00401505	. 04 20	HOU HL,20
00401505	. SHUS	NOV CL, HL
00401507	> 966:89 9138	MUV CX, 3891
0040150B	. 66:81F1 HD0F	XOR CX, UFHD
00401510	> 88 01000000	MOV EAX,1
00401515	> 5A	POP EDX
00401516	. 59	POP ECX
00401517	. C9	LEAVE
00401518	L. C2 0800	RETN 8
0040151B	r\$ 55	PUSH EBP

第一行代码 CMP DWORD PTR DS:[EAX], 7A81B008 做了另一个检测。在 对密码做完了所有的操作以后 ,最后第一个四字节等于 7A81B008。如果不是, 我们就会跳到坏消息那:



所以还是 0 标志位来帮 011y 一把, 然后我们就来到了另一个检测群(为什 么不呢?), 首先对接下来的几个字节做了一些操作, 然后将其与 388DBF02 进 行比较, 并与各种内存中硬编码数字进行比较。这个在检测上有点矫枉过正了, 不过我认为作者可能觉得检测越多就越能保护好 crackme☺。绕过所有的跳转, 我们最后来到了我们想要的地方, 就是那个 4014FB 的 JMP 指令:



如果我们单步通过 RETN,我们将来到熟悉的地方,不过这次有点儿不同:

00401293	10	68 50304000	PUSH Crackme6.0040305D	
00401298	1.00	E8 84010000	CALL Crackme6.00401421	
0040129D		ØBCØ	OR EAX, EAX	
0040129F	×	1 5 IF.	JNZ SHORT Crackme6.004012C0	
004012A1	1.4	68 ØF304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
004012A6	1	FF35 80304000	PUSH DWORD PTR DS: [403080]	hWnd = 001A0DB4 (class='Edit',parent=00150DC8)
004012AC	10	E8 CB020000	CALL (JMP.&user82.SetWindowTextA)	SetWindowTextA
00401281		6A 00	PUSH 0	Enable = FALSE
004012B3		FF35 80304000	PUSH DWORD PTR DS: [403080]	hWnd = 001A0DB4 (class='Edit',parent=00150DC8)
004012B9	2	E8 88020000	CALL <ump.&user32.enablewindow></ump.&user32.enablewindow>	EnableWindow
004012BE		EB 10	JMP SHORT Crackme6.004012D0	
00401200	>	968 00304000	PUSH Crackme6,00403000	Text = "ACCESS DENIED!"
00401205		FF35 80304000	PUSH DWORD PTR DS: [403080]	hWnd = 001A0DB4 (class='Edit',barent=00150DC8)
004012CB		E8 AC020000	CALL (JMP.&user32.SetWindowTextA)	SetWindowTextA
The state of the state of the				

注意这次我们来到了好消息这些。这是因为我们组织应用程序将 EAX 设置为

现在,你可能会认为"太棒了,我们在这个新的深入分析中,在级别二层 面只用了一个补丁换来了9个(被设置0标志位的所有JNZ)",不过这却不是真 正的情况。我不仅理解了它是如何工作的(并且对于将来的逆向挑战也赢得了 大量经验),现在还能够打非常牢固的补丁,因为我们知道这个补丁无论在什么 情况下都会起作用。有一点没有提,那就是找到这个软件的真正的密码其实不 是很难,这样就绕过了任何需要打补丁的地方!这就是真正的逆向工程,它只 能靠**大量的**练习。并且应用程序越难破解,你就越能够从代码中获取更多的细 节。

再说一次,如果你失败了也不要担心。这次更多的只是提供一个相关方法的使用印象。我们将会再次的学习这些内容。同时呢,这里有一些...

五、作业

就是教程前面提到的,看你是否能够用 NOOB 技术给程序打补丁。这就意味着,找到一个方法步入对密码进行所有操作的 CALL,并找到一个绕过所有操作 的方法。你不需要理解对密码做的所有操作,仅仅是找到一个让程序跳过它并 且仍然能够得到好消息的方法。

如果你需要提示,请点击这里。

超级吊的加分题:你能够找到硬编码密码吗?

第十一章:用 NOOB 技术破解

一、简介

本章我们将再次讨论补丁程序,不过比典型的单个"我们遇到的第一个补 丁"要深入一点点。我们将从一个控制台程序开始,找到隐藏在其中的正确密 码。教程的相关下载中有。除此之外,你只需要 OllyDbg。

你可以在教程页下载相关文件及本文的 PDF 版。

那么,咱们开始吧...。

控制台程序是和其他 windows 下 32 位的程序一样。唯一的不同是它们不使 用 图 形 界 面 。 除 此 之 外 , 它 们 是 一 样 的 。 此 次 的 crackme 叫 CrackmeConsole. exe。咱们来运行一下看看情况:



好,看起来挺简单得。咱们来随便输个密码:



真失败! 按"N"结束程序吧:



好吧,我觉得至少我们有了足够的信息来开始研究它。GO,Olly载入应用。 开始,首先搜索字符串:

R Found strings are	
Address Disassembly	Text string
Display Pick Fig. Byte Pick Fig	<pre>Initial CPU selection) HSCII "bad cast" ASCII "Try ToryTot" HSCII "Dease enter the correct serial: " HSCII "The serie the the correct serial: " HSCII "The serie the the correct serial: " HSCII "The security failure detected!" HSCII "The security error of unknown cause has been detected which has\ncorrupt: _ HSCII "Ascin "Ascin "Suffic overnu detected!" HSCII "Suffer overnu detected!" HSCII "Suffer overnu detected!" HSCII "Ascin "Ascin "Ascin " HSCII "Try again? HTT" HSCII "In sorry but that is wrong?" HSCII "Try again? HTT" HSCII "Tong live the ARTHEMAT" HSCII "Tong live the ARTHEMAT" HSCII "Tong live the ARTHEMAT" HSCII "Tong at the tis correct!" HSCII "Tong at the tis correct!" HSCII "Tong at the tis correct to the everything AR" HSCII "Tong base::failbit set" HSCII "Tong bas</pre>

不是很难嘛! 双击坏消息 "l'm sorry, but that is wrong", 至少来到了 正确的地方:



好,咱们研究研究这个。我们看到一个从 4025C6 来的跳转,有红色箭头标出来了。我们也注意到,如果 4025D5 的 JE 指令没实现的话,我们也会得到坏消息。咱们来看看如果这个跳转实现的话会怎么样。点它:

004025CE	. 3BD5	CMP EDX, EBP	
00402500	. 0F95C0	SETNE AL	
00402503	. 3BC3	CMP ERX, EBX	
004025U5	V 0F84 CF020000	DUCH Crackmet, 004028HH	OCCLL WILL anyone but that is un-
00402506	68 002E4100	PUSH CrackmeC 00412E00	HOULT I'M SULTY DUE CHAE IS WIT
004025E5	ES A6F4FFFF	CALL CrackmeC, 00401890	
004025EA	. 83C4 08	ADD ESP,8	strend water in a strend with the strend to
004025ED	. 88F0	MOV ESI, EAX	kernel32.BaseThreadInitThunk
004025EF	· 68 08	PUSH 0A	and the second sec
004025F1	. SBUE	MUU ECX, EST	
004025F3	· LO DOFOFFFF	MOU ECY DUODD DTD DC. FEETA	
004025F8	8851 04	MOU EDX, DWORD PTR DS: LECX+41	
004025FD	. 8A4C32 08	MOV CL. BYTE PTR DS: [EDX+ESI+8]	
00402601	. 8D0432	LEA EAX, DWORD PTR DS:[EDX+ESI]	
00402604	• 33FF	XOR EDI, EDI	
00402606	· F6C1 06	TEST CL.6	
00402609 0640260P	·* 75 14 9840 29	MOLL FOX DWORD PTP DS+FEOX+291	
0040260E	. 8810	MOU EDX. DWORD PTR DS: (FAX1	the second s
00402610	3 BBC8	MOV ECX, EAX	kernel32.BaseThreadInitThunk
00402612	. FF52 20	CALL DWORD PTR DS: (EDX+2C)	and an additional to the second s
00402615	. 83F8_FF	CMP EAX, -1	
00402618	· 75 05	MOU EDI 4	
00402616	> 9P04	MOU FOY DUORD PTP DS+FEST1	
90402621	. 8B48 Ø4	MOU ECX. DWORD PTR DS: [E8X+4]	
00402624	. ØSCE	ADD ECX.ESI	
00402626	. 3BFB	CMP EDI, EBX	
00402628	.~ 74 16	JE SHORT CrackmeC.00402640	
0040262H	· 8841 08	MOU ERX, DWORD PIR DS: LECX+81	
00402020	0001 20 0807	IN FOX FOI	
00402632	3803	CMP FDY FRY	

滚动到它指向的地方(在下面几页):



这看起来就是我们想走的路。咱们回到上面看看周围地方:



4025D5是调到好消息那的,这就是我们想要实现的跳转。咱们点一下另一个跳转看看它将跳到哪去...。说不定前面也有个跳转可以跳到好消息那呢:

004025BF 004025C1 004025C4 004025C6 004025C6 004025C8	· 1800 • 8308 FF > 3803 FF > 3803 · 75 13 > 3805_	SBB EAX, EAX SBB EAX, -1 CMP EAX, EBX JNZ SHORT CrackmeC, 0040250B CMP EDX, EBP	kernel32.BaseThreadInitThunk
004025CC	· 72 0F • 33C0 38D5	XOR EAX,EAX	kernel32.BaseThreadInitThunk
004025D0 004025D3 004025D5 004025D5 004025D8 004025E0 004025E5 004025E4	. 0F95C0 . 3BC3 . 0F95C F020000 . 68 48224000 . 68 48224000 . 68 A02F4100 . 68 A6F4FFF . 83C4 08	SETNE AL CMP ERX, EBX JE CrackmeC, 004028AA PUSH CrackmeC, 00402248 PUSH CrackmeC, 00412EA0 CALL CrackmeC, 00401A90 ADD ESP.8	ASCII "I'm sorry but that is wrong?"

这个是到坏消息的:

00402580 0040258F 004025C1 004025C4	. 18C0 . 83D8 FF > 38C3	SBB EAX, EAX SBB EAX, -1 CMP EAX, EBX	kernel32.BaseThreadInitThunk
00402506	.~ 75 13	JNZ SHORT CrackmeC.004025DB	
00402508	> 3BD5	CMP EDX, EBP	
004025CA	.~ 72 ØF	JB SHORT CrackmeC.004025DB	A REAL PROPERTY AND A REAL PROPERTY AND A REAL PROPERTY.
004025CC	. 3300	XOR EAX, EAX	kernel32.BaseThreadInitThunk
004025CE	. 3BD5	CMP_EDX, EBP	
00402500	. 0F95C0	SETNE HL	
00402503	· 3863	UNP EHX, EBX	
00402505		UE LTACKMEL.004028HH	OCCUT MILE STAND AND ADDE TO DESCRIPTION
004025DB	20 000004100	PUCH Crackmet 0040E240	MSCII I M SOFTY DUE that is wrong!
BRARDEEL	- 60 H02F4100	COLL Cyschwell 00412FH0	and a second second second second
00402555	. LO HOF4FFFF	ODD ECD O	the state of the s
DOMOZOER	. 0304 00	HOD EST, O	I ICO D TI IT VITI I

这个也是,如果你接着点那些跳转指令,你会发现 4025D5 是唯一一个跳 到好消息的跳转。所以基本上,我们要阻止所有跳到坏消息的跳转实现,强制 跳到好消息的跳转成功跳转。如果我们接着往上滚动,就会在 402582 找到第一 个 call/compare (调用/比较) 指令:

00402579 00402578 00402578 00402570 00402580 00402581 00402581	. 38D3 .~ 75 03 . 83C8 04 > 53 . 50 	CMP EDX, EBX UNZ SHORT CrackmeC.00402580 OR EAX, 4 PUSH EBX PUSH EBX PUSH EBX COLL CrackmeC.00402587	kernel32.BaseThreadInitThunk
00402587 00402587 00402592 00402592 00402592 00402596 00402596 00402596 00402586 00402586 00402586 00402586 00402586 00402585 00402585 00402585 00402585 00402585	 Contrology 837C24 2C 10 887C24 18 73 04 807C24 18 885424 44 3803 886C24 28 74 26 3805 88CA 72 02 88CA 72 02 88CA 72 02 88CA 73 04 887424 34 73 04 807424 34 3300 3300 	CMP DWORD PTR SS: LESP+2C1, 10 MOV EDI, DWORD PTR SS: LESP+181 UNB SHORT CrackmeC, 00402596 LEA EDI, DWORD PTR SS: LESP+181 MOV EDX, DWORD PTR SS: LESP+181 MOV EDX, DWORD PTR SS: LESP+441 CMP EDX, EBX MOV ECX, EDX UE SHORT CrackmeC, 004025C3 CMP EDX, EBP MOV ECX, EDX UB SHORT CrackmeC, 004025AA MOV ECX, EBP CMP DWORD PTR SS: LESP+341 UNB SHORT CrackmeC, 004025B9 LEA ESI, DWORD PTR SS: LESP+341 UNB SHORT CrackmeC, 004025B9 LEA ESI, DWORD PTR SS: LESP+341 UNB SHORT CrackmeC, 004025B9 LEA ESI, DWORD PTR SS: LESP+341 XOR EAX, EAX	CrackmeC. <moduleentrypoint> kernel32.BaseThreadInitThunk</moduleentrypoint>

再往上滚动,就会发现有个跳转跳过了那个 CALL,但是仍然进行了比较:

0040255F 00402561 00402566 00402568 00402568 00402568 00402560	. 75 05 . BF 04000000 > 8806 . 8848 04 . 03CE . 38FB	MOV EDI,4 MOV EDI,4 MOV EAX,DWORD PTR DS:[ESI] MOV ECX,DWORD PTR DS:[EAX+4] ADD ECX,ESI CMP EDI,EBX	Jump past the call
004025671 00402571 00402579 00402579 00402579 00402579 00402579 00402580 00402580 00402581 00402582 00402582 00402582 00402592 00400	. 74 16 . 8841 08 . 8851 28 . 08C7 . 38D3 . 75 03 . 83C8 04 . 53 . 50 . E8 100A0000 . 887C24 2C 10 . 887C24 18 . 73 04 . 807C24 18 . 885424 44 . 38D3 . 886C24 28 . 74 26 . 38D5	LEA ENLAND PTR SS: LESP+181 MOV EDX, DWORD PTR DS: LECX+81 MOV EDX, DWORD PTR DS: LECX+81 OR EAX, EDI CMP EDX, EBX JN2 SHORT CrackmeC.00402580 OR EAX, 4 PUSH EAX PUSH EAX CALL CrackmeC.00402597 CMP DWORD PTR SS: LESP+181 JNB SHORT CrackmeC.00402596 LEA EDI, DWORD PTR SS: LESP+181 MOV EDI, DWORD PTR SS: LESP+181 MOV EDX, DWORD PTR SS: LESP+281 JE SHORT CrackmeC.00402508 CMP EDX, EBP	kernel32.BaseThreadInitThunk

这个行为不太正常,如果我们再往上滚动一点,就会发现另外一对调用/ 比较 指令对。我在这两个 CALL 上都设置了 BP:



好吧,咱们继续,在 Olly 中运行程序看看会发生什么。我将输入密码 "12121212":



Olly 断在了第一个 CALL:



单步调试,注意 42056F 处的跳转跳过了第二个 CALL。嗯,这倒给了我们 一个提示,第二个跳转可能不是校验密码的,不过有可能是某种验证程序,如

果我们的密码不符合某种规则,比如太短或者太长?不管是啥,咱们接着单步运行就行了:

89492540 . F6C1 06 09492550 .√ 75 14 09492552 . 8849 28 09492555 . 8810 09492555 . 8810 09492557 . 8808 09492557 . 8808	TEST CL,6 UN2 SHORT CrackmeC.00402566 MOV EAX,DWORD PTR DS;[EAX+28] MOV EDX,DWORD PTR DS:[EAX] MOV ECX,EAX FOLL DWORD PTR DS:[EDX+2C]	CrackmeC.00404D25 CrackmeC.0040E470 CrackmeC.0040B37A1	
Jumps past our call 00402560 . 8848 04 00402560 . 03CE 00402560 . 8878	P ERX,-1 SHORT CrackmeC.00402566 # EDI,4 # EAX,DWORD PTR DS:[ESI] MOV ECX,DWORD PTR DS:[EAX+4] ADD ECX,ESI CMP EDI.EBX	CrackmeC.0040E470 CrackmeC.00412FA0	
0040256F ~ 74 16 00402571 8841 08 00402574 8851 28 00402577 08C7 00402579 38D3 00402571 83C8 04 00402571 83C8 04 00402571 83C8 04 00402581 53 00402581 50 00402581 53 00402581 53 00402581 54 00402581 53 00402581 53 00402582 887C24 2C 10 00402596 ~ 73 04 00402596 885424 44 00402596 885424 44 00402596 38D3	JE SHORT CrackmeC.00402587 MOV EAX,DWORD PTR DS:[ECX+8] MOV EAX,DWORD PTR DS:[ECX+8] OR EAX,EDI CMP EDX,DWORD PTR DS:[ECX+28] OR EAX,4 JNZ SHORT CrackmeC.00402580 OR EAX,4 PUSH EBX PUSH EBX CALL CrackmeC.00402597 CMP DWORD PTR SS:[ESP+18] UNB SHORT CrackmeC.00402596 LEA EDI,DWORD PTR SS:[ESP+18] UNB SHORT CrackmeC.00402596 LEA EDI,DWORD PTR SS:[ESP+18] MOV EDX,DWORD PTR SS:[ESP+18] MOV EDX,DWORD PTR SS:[ESP+18] CMP EDX,DWORD PTR SS:[ESP+18] CMP EDX,DWORD PTR SS:[ESP+14] CMP EDX,DWORD PTR SS:[ESP+44]	CrackmeC.00412F40 CrackmeC.0040E470	

4025C6 这里, 咱们看到了罪魁祸首了, 就是它跳到了坏消息那:

00402586	. 72 02	UB SHORT CrackmeC.004025AA	os to bad boy
00402588	. 88CD	MOV ECX,EBP	
00402588	> 837C24 48 10	CMP DWORD PTR SS:LESP+48],10	
00402585	. 887424 34	MOV ESI,DWORD PTR SS:LESP+341	
00402583	. 73 04	UNB SHORT CrackmeC.004025E9	
00402583	. 807424 34	LEA ESI,DWORD PTR SS:LESP+341	
00402589	> 5369	AUR EHA, EHA	
00402588	• F3: A6	REPE CMPS BYTE PTR ES: (ED12, BYTE PTR DS:	
00402580	• 74 05	JE SHORT CrackmeC, 00402504	
0040258F	• 1800	SBB ERX, EAX	
0040258F	• 8308 FF	SBB ERX, -1	
004025C1	> 3803	CMP ERX, EBX	
004025C4	• 275 13	.NZ SHORT CrackmeC, 00402508	
004025C8 004025CC 004025CC 004025D0 004025D3 004025D3 004025D5 004025E0 004025E0 004025E5 004025E5 004025E5 004025E1 004025F1 004025F3 004025F3 004025F3	 > 38D5 > 72 0F > 3200 > 38D5 • 0F95C0 38C3 • 0F84 CF020000 > 468 40824000 • 68 A02F4100 • 68 A02F4100 • 83C4 08 • 88C4 • 907 • 8804 	CHP EDX,EBP JB SHORT CrackmeC.004025DB XOR EAX,EAX CMP EDX,EBP SETNE AL CMP EAX,EBX JE CrackmeC.00402248 PUSH CrackmeC.00402248 PUSH CrackmeC.00412FA0 CALL CrackmeC.004012FA0 CALL CrackmeC.004014FA0 ADD ESP.8 MOV ESI.EAX PUSH 0A MOV ECX,ESI CALL CrackmeC.00401ED0 MOV ECX,DWORD PTR DS:EESI1	ASCII "I'm sorry but that is wrong!"

咱们设置下0标志位,看看会怎么样:

	and the manage
0101100 0101100	ES 0023 CS 001B SS 0023 DS 0023 FS 003B GS 0000
ŏŏ	LastErr

继续单步,终于和跳到好消息的跳转碰头了,注意它实现了:



继续运行程序,我们发现我们已经找到了第一个潜在的补丁:



现在,给我们刚才设置O标志位的那个跳转打上补丁,这可能有用也可能 不起作用。这很难说。如果我们的密码太短会怎样?太长呢?是不同于我们所 输入的密码的(译者注:大概这个意思,我没弄明白作者啥意思。原文是 A different password than the one entered)。这个补丁不是一个非常好的补丁, 因为我们真的不知道我们到底做了什么,我们只知道在这种情况下会起作用。

二、深入挖掘

咱们靠近点看看这段代码,用上一章我学到的级别,试试不那么 LAME 的方法。向上滚动到我们打过补丁的那个跳转,就是跳到坏消息的那个,咱们来 试试看找出为什么我们没打补丁时它会跳转。注意,我已经在跳转那加了一个 注释,这样后面比较容易记住(回想下,选中该行,按一下";"来添加注释):



我们通常在注释前加上"###"以示区别,这样的话在将来,当用其他的工 具来向我们显示注释的时候,就更容易找到我自己得注释,因为它们比较突出。 当然你也可以按自己喜欢的方式做。

现在,咱们就来看看跳转的上面,看能否找到是什么让它跳转的。我在下 面已经标记出了跳转上面的第一个区块:



我们能看到有几个 SBB 指令和一个比较指令。对于我们来说,这里的这段 代码并不真正有什么意义,因为我们不知道它是干啥的,所以咱们网上看下一 个区,看看咱们能不能开始对它有所了解:

004025A2 004025A4 004025A6 004025A8 004025A8 004025A8 004025AF	3BD5 9BCA 9BCD 8BCD 837C24 48 10 8B7424 34	CMP EDX,EBP MOV ECX,EDX JB SHORT CrackmeC.004025AA MOV ECX,EBP CMP DWORD PTR SS:[ESP+483,10 MOV ESI,DWORD PTR SS:[ESP+484]	CrackmeC.0040E408
00402583 08402585 00402589 00402588 00402588 00402588 00402585	 ✓ 73 04 807424 34 33C0 F3: A6 ✓ 74 05 18C0 	LEA ESI, DWORD PTR . M9492569 LEA ESI, DWORD PTR SS: (ESP+34) XOR EAX, EAX REPE CMPS BYTE PTR ES: (EDI), BYTE PTR DS: JE SHORT CrackmeC.00402504 SED EOV EOV	CrackmeC.00412F40
004025C1 004025C4 004025C6 004025C6	83D8 FF 3BC3 * 75 13 3BD5	SBB EAX,-1 CMP EAX,EBX JNZ SHORT CrackmeC,0040250B CMP EDX,EBP	### Jump to bad boy
004025CR 004025CC 004025CE 004025D0 004025D0	72 0F 33C0 3BD5 0F95C0 2BC2	UB SHORT CrackmeC,004025DB XOR EAX,EAX CMP EDX,EBP SETNE AL CMP EDX EBY	CrackmeC.00412F40
00402505 00402508 00402550 00402550	✓ 0F84 CF020000	UE CrackmeC.004028AA PUSH CrackmeC.0040E248 PUSH CrackmeC.0040E248 PUSH CrackmeC.00412FA0	ASCII "I'm sorry but that is wrongt"

好,这里我们将会到达某个地方。可能你注意到第一个问题的是 REPE CMPS 指令。这是逆向工程的一个红色标志(译者注:原文是 This is a red flag in reverse engineering!,我不知道作者是啥意思,就直译了)! 咱们查查 REPE 看看是啥意思:

ents Index	Back Print		
EP/REPE/R	epz/repne/repnz	Repeat String Operation Prefix	
F2 AF	REPNE SCAS m32	Find EAX, starting	g at ES:[(E)D/]
escription epeats a stri the ZF flag epeat while structions. EPNE, REF refixes are s ndefined while ne REP pref	ing instruction the num is no longer met. The F zero), and REPNZ (rep The REP prefix can be ⁷ Z, and REPNZ prefixes ynonymous forms of th en used with non-string fixes apply only to one consthet looping const	ber of times specified in the count r REP (repeat), REPE (repeat while e eat while not zero) mnemonics are added to the INS, OUTS, MOVS, L s can be added to the CMPS and S is REPE and REPNE prefixes, resp g instructions. string instruction at a time. To reper-	register ((E)CX) or until the indicated condition qual), REPNE (repeat while not equal), REPZ prefixes that can be added to one of the string ODS, and STOS instructions, and the REPE, CAS instructions. (The REPZ and REPNZ bectively.) The behavior of the REP prefix is at a block of instructions, use the LOOP
Il of these re ecremented punter, and refixes also pecified stat ther by test struction.	peat prefixes cause th to 0 (see the following if the address-size attr check the state of the e. When both terminati ing the (E)CX register v	e associated instruction to be repea table). (If the current address-size a ibute is 16, the CX register is used. ZF flag after each iteration and term on conditions are tested, the cause vith a JECXZ instruction or by testin	ated until the count in register (E)CX is attribute is 32, register ECX is used as a) The REPE, REPNE, REPZ, and REPNZ inate the repeat loop if the ZF flag is not in the e of a repeat termination can be determined g the ZF flag with a JZ, JNZ, and JNE
Repeat Cor	nditions		
Repeat Pre	fix	Termination Condition 1	Termination Condition 2
REPE/REPZ		ECX=0	ZF=0
REPNE/REPNZ		ECX=0	ZF=1
e CMPS an repeating s gisters is pl andler. The bints to the struction. The struction. The system.	d SCAS instructions at tring operation can be s reserved to allow the st source and destination string instruction, and t his mechanism allows b occurs during the exec re is restored to the sta lo not use EFLAGS as INS and REP OUTS in execute.	ffect the ZF flag according to the res suspended by an exception or interr ring operation to be resumed upon a registers point to the next string ele he ECX register has the value it hel long string operations to proceed wi ution of a CMPS or SCAS instruction to prior to the execution of the instru- an input, the processor can resume estructions with caution. Not all I/O	sults of the comparisons they make. rupt. When this happens, the state of the a return from the exception or interrupt ements to be operated on, the EIP register d following the last successful iteration of the thout affecting the interrupt response time of on that is prefixed with REPE or REPNE, the uction. Since the SCAS and CMPS e the instruction after the page fault handler. ports can handle the rate at which these
/hen a fault FLAGS valu structions d se the REP structions e REP STOS	instruction is the faste	est way to initialize a large block of	=



现在,注意第一条指令,LEA ESI,DWORD PTR SS:[ESP+34],准备 将一个栈中得有效地址载入到 ESI 中。SS:表示堆栈,[ESP+34]表示的是栈中 的位置,本例中是 ESP 所指向位置前面的第 34 字节。LEA 指令意思是取地址, 而不是取内容。如果我们看那个中间区域(就是蓝色箭头指向的地方),可以发 现 SS:[ESP+34]等于地址 0012FE88,在这个地址存储的是我们的 ASCII 形式 的密码。单步步过该行,可以看到 ESI 被设置成我们的密码(当前是在栈上):



下一条指令将 EAX 设置为 0, 然后就是 REPE 指令。本例中,是将存储在 ESI 中的地址的内容与存储在 EDI 中的地址中的内容进行比较:



ECX 寄存器减一,比较就转到 ESI 和 EDI 的下一个内存位置,当 ECX=0 时循环结束。本例中,如果你往上看,会发现 ECX 被置为 8(就是我们密码的 长度),所以该循环会遍历我们密码的 8个数字,每一次将一个数字与 EDI 中相

关的数字进行比较。不过,等等...,我们正和谁比较呢?如果我们再看看寄存器窗口,我们会发现 EDI 指向的是堆栈中的一个地址,其中存储着几个 ASCII 字符 7。咱们到堆栈中看看。点击挨着 EDI 的那个地址,在其上右键选择 "Follow in stack (堆栈中跟随)":

EDI 0012	Increment	Diur
C 0 ES I	Decrement	Minus
A Ø SS I Z 1 DS I	Zero	
TO GSI	Set to 1	
0 0 Last EFL 0000	Modify	Enter
ST0 empt: ST1 empt:	Copy selection to clipboard	Ctrl+C
ST2 empti ST3 empti ST4 empti	Copy all registers to clipboard	
ST5 empti ST6 empti ST7 empti	Follow in Dump	-
FST 0000	Follow in Stack	
1.00.000.0	View MMX registers	
	View 3DNow! registers	
-	View debug registers	
	HW break [ESP]	
	(

堆栈窗口立即就跳转到相关地址处,也就是 0012FE6C 处。在该地址(我 们不能不注意到后面的也是一样)我们看到一串 "37"。查查 ASCII 码表就知道 37 就是 "7",就是我们在寄存器窗口中看到的 EDI 寄存器中的内容:

0012FE6C	37373737	
0012FE70	37373737	the second second second
8812FE74	00412000	CrackmeC.00412000
0012FE78	0012FF88	102 262 Mart 200 Mart 200
8012FE7C	00000008	
0012FE80	RARARA	
9912FE84	00000000	
0012FF88	32313231	
9912FE80	32313231	the second se
AR12FF98	00406400	Crackmer 00406400
0012EE94	00300680	01 400 100 100 100
BR12FF98	000000000	
BRIDEFOR	000000000	
OOMERE 20	00000000	

好吧,不需要像外科医生那样就能够发现我们输入的密码正在和硬编码的 全是"7"的字符串进行比较。堆栈中真切的只有 8个"7"(很走运,我们输入 的密码正好和硬编码密码的长度相同。)。这八个"7"与我们输入的密码一个一 个的进行比较。如果所有的 8 个都相等(也就是等于 7),我们就会执行下一个 跳转。嗯...,我们输入的密码被拿来和 8个"7"进行比较。给我的感觉就是 密码可能就是八个"7"。咱们来重启应用试试看:



此处应该有掌声...。



我们拿到了☺。所以,在我们通常打补丁的地方的稍远处我们发现了密码, 坦白的说这比给一个程序打补丁要好的多,因为我们不知道是真的打上了还是 没有。相比 LAME 级别,这就是 NOOB 级别补丁的好处。

三、最后一件事

我只是想举个例子,是分析代码及对代码进行注释。不幸的是,在写教程时,你需要在相当深的层次上理解相关应用。下面是核心区块的图片,我在其中加了注释:



如你所见,很多都是对应用程序工作方式的理解。
第十二章:一个难啃的 NOOB 例子

一、简介

本章我们将研究一个有点更具挑战性的程序。它叫 ReverseMe1,我写的。 我也会讨论一个 Olly 的插件 "ASCII 码表"。可以在<u>工具</u>页下载它。这个 ReverseMe 是用来说明为什么 LAME 补丁方式通常就是那么 lame (烂)的一 个极好的例子。

你可以在<u>教程</u>页下载相关文件及本文的 PDF 版。

二、准备开始

运行下程序看看:

This progra	am is not reg	jistered
Π		_
Check Seria	l le	Quit

我们能看到它说还没有注册,需要序列号。那就给它一个:

This program is not regist	tered
12121212	_
Check Serial Qu	iit

点 "Check Serial":



我们看到我们是错的(再一次)! Olly 载入应用,用咱们信得过的"搜索字符串":

Address 004010AC 004010AC 004010B1 004010BD 004010D5 004010DA 004010DA 004010DA	Disassembly PUSH 0 PUSH Register.00403000 PUSH Register.00403034 PUSH Register.00403004 PUSH Register.00403000 PUSH Register.00403050	Text string (Initial CPU selection) ASCII "Register me 1" ASCII "That is correct!" ASCII "This program is registered!" ASCII "Register me 1" ASCII "That is not correct." ASCII "This program is not registered!"	
00401101	PUSH Register.00403098	ASCII "12121212"	
00401116	MOV EAX,Register.00403098	ASCII "12121212"	

好哇,看起来前途光明呀。咱们来检查下"That is not correct"字符串:

004010AA 004010AA 004010AA 004010B1 004010B1 004010B1 004010B2 004010B2 004010B2 004010B2 004010B2 004010B2 004010B2 004010B2 004010B5 00401005 00400000000000000000000000	 C9 C2 1000 C3 1000 C4 00 68 00304000 68 2304000 68 2304000 68 24304000 68 24304000 68 20304000 E8 BA000000 C3 68 00304000 E8 0000000 FF35 94304000 E3 91000000 68 93304000 FF35 94304000 68 93304000 FF35 94304000 E3 9400000 E3 9400000 E3 94000000 E3 94000000 E3 940000000 E3 94000000000000000000000000000000000000	No. Lno. Lno. LERVE RETN 10 PUSH Register.00403000 PUSH Register.00403023 PUSH Register.00403023 PUSH Register.00403034 PUSH Register.00403034 PUSH BUD CRLL (JNP.&user32.MessageBoxA) PUSH BUD PUSH Register.00403004 PUSH Register.00403006 PUSH SED PUSH DWORD PTR DS:14080941 PUSH Register.00403096 PUSH 200 PUSH Register.00403096 PUSH PUSH Register.00403096	Style = MB_OKIMB_APPLMODAL Title = "Register me 1" Text = "That is correct!" hOwner = NULL MessageBoxA Text = "This program is registered!" ControllD = 3ED (1005.) hUnd = 00030368 ('Register Me 1',class='#32770') SetDigitemTextA Style = MB_OKIMB_APPLMODAL Title = "Register me 1" Nowner = NULL MessageBoxA Text = "This program is not registered!" ControlID = 3ED (1005.) hUnd = 00030368 ('Register Me 1',class='#32770') SetDigitemTextA ControlID = 3ED (1001.) hUnd = 00030368 ('Register Me 1',class='#32770') SetDigitemTextA
00401106 00401108 00401108 00401116 00401116 00401118 00401110 00401110	 68 £9030000 FF35 94304000 E8 6A000000 B8 98304000 B8 98304000 8800 66:3D 3433 √ 75 07 	PUSH WORD PTR DS:[408094] USH WORD PTR DS:[408094] HOL EAX,Register.00408098 HOU EAX,Register.00408098 HOU EAX,Sas4 CHP AX,3834 UTC SHORT Register.00401120	ControlTD = 369 (1001.) hWnd = 00030368 ('Register Me 1',class='#32770') GetDlgitewTextR ASCII "12121212"

咱们来到了问题的核心。因为每一个都是单独的方法,我们需要看看哪里 调用了它们,所以咱们要这么做:

04010C7 04010CD 04010D2	FF35 94304000 E8 BA000000 C3	PUSH DWORD PTR DS:[403094] CALL <_MP.&user32.SetDigItemTextA> RETN	hWnd = SetDig	00030368 ('Register Me altemTextA	1',class='#32770':)	ST3 empty 0. ST3 empty 0. ST4 empty 0. ST5 empty 0.	9 9 9
04010D3 04010D5 04010DA	\$ 6H 00 . 68 00304000 . 68 0E304000	PUSH 8 PUSH Register,00403000 PUSH Register,0040300E	TI	Backup	•		ST6 empty 0. ST7 empty 0.	0 32
04010DF 04010E1	. 6H 00 . ES A0000000	PUSH 0 CALL (JMP.&user32.MessageBoxA) PUSH Pariston 00403050	Me	Сору	+		FST 4020 Co FCW 027F Pi	ec NEAF
04010EB 04010EB	. 68 ED030000 . FF35 94304000	PUSH SED PUSH DWORD PTR DS:[403094]	Cd	Binary	+			
04010F6 04010FB	E8 91000000 C3	CALL (JMP.&user32.SetDigItemTextA) RETN	LSe	Assemble	Space			
04010FC 0401101	\$ 68 00020000 • 68 98304000 • 68 598304000	PUSH 200 PUSH Register.00403098	Cd By	Label	4			
0401108 0401108 0401111	. FF35 94304000 . E8 6A000000	PUSH DWORD PTR DS:[403094] CALL <jmp.&user32.getdlgitemtexta></jmp.&user32.getdlgitemtexta>	hlu	Comment	*			
0401116 040111B	. B8 98304000 . 8800	MOV EAX,Register.00403098 MOV EAX,DWORD PTR DS:[EAX]	AS	Breakpoint	*			
0401110 0401121 0401123	- 66:30 3433 75 07 - 88 0000000	UN2 SHORT Register.0040112A		Hit trace				
0401128 040112A		JMP SHORT Register.0040112F MOV EAX,1		Run trace				
040112F 0401130 0401135 0401137	C3 \$ E8 C7FFFFFF . 0BC0 .√ 74 33	RETN CALL Register,004010FC OR EAX,EAX JE SHORT Register,0040116C	re	New origin here	Ctrl+Gray *			
0401139 040113E	. B9 1F000000 . BE 00000000	MOU ECX, 1F MOU ESI, 0	4	Go to	*			
0401143 0401145 0401148	> 8886 70304000 8350 20	MOV AL, BYTE PTR DS: [ESI+403070]	rţ	Follow in Dump			1.1.1.1	
040114E 0401154	. 8886 70304000 . 46	MOV BYTE PTR DS:[ESI+403070],AL INC ESI		View call tree	Ctrl+K		1	
0401155 0401157 040115C 040115C	-^ E2 EE - 68 70304000 - 68 ED030000 - FE35 94304000	LLOOPD SHORT Register.00401145 PUSH Register.00403070 PUSH 3ED PUSH 3ED PUSH 3ED	Co	Search for	+	'IH\r"		
0401167 040116C	E8 2000000	CALL CALL CHP.&user32.SetDigItemTextA> RETN	Se	Find references to		Selecte	ed command	Ctrl
040116D 040116E 0401174	\$- FF25 24204000 \$- FF25 18204000	INT3 JHP DWORD PTR DS:[{&user32.DialogBoxParamA}] .MP DWORD PTR DS:[{&user32.EndDialog}]	ue ne	View	1			

Olly 弹出了 References 窗口:



我们能够看到有一个对该函数的调用。咱们双击它,看看它是啥样的:



这里,我们能看到坏消息是在 401078 处调用的,并且我们马上就能看到 40106A 处有个跳转指令跳到这里:



向上滚几行,我们就能看到一个 CALL,用来检测 程序/比较/跳转,和我 们前面看到的一样。从这里我们能够猜到,主要的检测程序是在 4010FC, 401063 处调用了它。在返回后,EAX 寄存器被检测其值是否是 0,如果不是就 跳到坏消息。



测试下我们的假设, 在 40106A 处设置断点, 然后重启应用。在输入一个序列号以后(我输入的还是"12121212"), 我们断在了调用序列号校验的那个 CALL 后面的跳转处:



现在咱们帮 Olly 走正确的路,所以我们不能让跳转实现(直接到调用好消息的 CALL 那):

	C	0	ES 002
	P	0	CS 001
	A	0	SS 002
	Z	1	DS 002
	S	0	FS 003
	T	Ø	GS 000
	D	0	
	Ô	Ø	LastEr
1			

点一下运行:



耶, so easy (妈妈再也不用担心我破不了了)!!点 OK:

This program is n	not registered	
12121212		
Check Serial	Quit	

噢, 艹%\$Q, 这里他爸的发生了啥, 你个阿西吧\$\$\$%^#!!!!很明显, 我们的程序没有注册成功。这说明我们肯定错过了啥。

三、进一步分析

重启应用,输入序列号,让Olly再次断在40106A:

00401068	ABC0	OR FOX FOX	
004010-00	.V/75 0C	UNZ SHORT Register,00401078	-
0040106C	. E8 3900000	CALL Register.004010AA	
00401071	. E8 BA000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.00401084	
00401078	> 7E8 56000000	MP CHOPT Pagiston 00401003	
0040107F	> 8170 10 FC030000	CMP LARE 31.3EC	
00401086	.v 75 1C	JNZ SHORT Register,004010A4	
00401088	. 6A 00	PUSH 0	
0040108A	. FF75 08	PUSH [ARG.1]	

看看这个,如果我们阻止 Olly 跳到坏消息那,直接执行 40106C 的那个 CALL,就是调用 4010AA。沿着那条路往下走,我们能看到它是相当的标准: 它弹出一个显示 "That is not correct"的消息框,然后将主窗口的标签修改成 "This program is registered!"。



PUSH 0 PUSH 0 PUSH Register.00403000 Style = MB_OK!MB_APPLMODAL Title = "Register me 1" 该 CALL 调用的是 401130, 所以咱们看看那个子程序。首先, 我们注意它 调用了 SetDlgItemTextA,不过有一个看起来很奇怪的字符串。咱们来一行一行 的执行。401130 有个 CALL 调用了 4010FC。往上看,我们看到这是一个序列 号校验子程序。然后 EAX 自身做了 OR 操作看是否为 0, 如果不是, 它执行了 许多看起来很怪异的玩意儿:

.&user32.SetDigItemTextA>



到目前为止,我们从这些收集到的信息是,在我们给程序打了补丁后它显 示了好消息,然后另一个 CALL 执行了,在这个 CALL 里,又有一个 CALL 再次 执行了序列号校验子程序,对结果做了同样的分析。这是一个备份检测点!现 在我们来看看如果我们在这个备份检测点失败的话会怎样(这里我们是可以让 它检测失败的,因为我们只给那个跳转打了补丁):



首先, ECX 被设置值为 1F(十进制是 31)***对不住了, 被切掉了一点(译 者注: 指的是上面图片中 MOV ECX, 1F 那行)***。然后 ESI 被赋值为 0, EAX 被清 0。然后就进入一个循环。咱们一步一步执行这个循环。第一行从 ESI+403070 拷贝了一个字节到 AL 寄存器中,我们知道 ESI 等于 0,所以地址 实际上就是 403070。咱们看看内存中这个地址里是什么。右键并选择 Follow in dump->constant,或者就右键 dump 窗口,选择 goto 并输入地址 403070。

Address	Hex dump	ASCII
00403070	78 44 45 5F ØC 5C 5E 43 4B 5E 4D 41	0C 45 5F 0C xDE\^CK^MA.E
00403080	42 43 58 0C 5E 49 4B 45 5F 58 49 5E	49 48 0D 00 BCX. ^IKE_XI^IH
00403090	00 00 40 00 B2 03 05 00 31 32 31 32	31 32 31 32@.###.12121212
00403000	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403000	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403000	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00
00403190	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00

如果仔细看的话,就会发现这就是上面传给 SetDlgTextItemA 的字符串参数。所以它就是将那串奇怪的字符串的第一个字符拷贝到 AL 中。

有件事你应该知道,许多汇编语言指令会按默认的使用方式使用某些寄存器,例如 ECX 被用来作为计数器,ESI 被用来作为源地址,EDI 被用来作为 目的地址。本例中就是这样的。

接下来,我们将该字符与 2C 进行 XOR,然后再将其存回原来的地址中:



最后,给 ESI(源址寄存器)加1,再做 LOOPD 操作。LOOPD 意思是 ECX 寄存器减1,然后循环直至 ECX 为0。也就是说,我们原来给 ECX 赋的值,十进制的31,就是循环的次数。

总的来看,该循环遍历奇怪字符串的每一个字符,将它们与 2C 进行 XOR 操作,再保存回原内存。这些操作将持续到 ECX 等于 0,或 31 次。单步执行一次 LOOPD 指令后回到顶部,然后看看数据窗口:

Address	He:	s di	IMP	-	-		-	-	-	-	-	100		-	-	200	ASCII			
00403070	54	44	45	5F	0C	50	5E	43	4B	5E	4D	41	0C	45	SF	00	TDE	CK/\	1A.E	
99493989	42	43	58	0C	5E	49	4B	45	SF	58	49	5E	49	48	ØD	60	BCX. ~IK	E_X1	I⊢	
00403090	Ø	NA	40	00	BZ	03	05	00	31	32	31	32	31	32	31	1 A	U.器#\$.121	212	12
98463646	UL.	90	00	90	90	00	00	00	00	90	90	90	90	90	00	200				
00403080	00	10	00	00	00	00	90	00	00	00	00	90	00	00	90	00				
00403000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				22
00403000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				
004030E0	00	00	00	aa	00	aa	00	00	00	00	00	00	00	aa	00	00				22
00403100	00	ññ	aa	ñã	aa	ña	aa	ññ	an	ññ	aa	ññ	aa	ña	aa	ññ	101111		100	11
00403110	00	ññ	ññ	ññ	ññ	ññ	ññ	ññ	ññ	ññ	BB	B B	ññ	B B	ññ	ññ				321
00.0000000	22	22	ΥΨ	44	22	22	YY.				24	22	22	22	ΥΨ	22				5 G L

你会发现,字符串的第一个数字已经变了。原来的字符被执行 XOR 操作后 变成了"T"。如果你单步执行这个循环几次的话,会看到数据窗口中的字符串 的变化。你也会发现传给 SetDlgTextItemA 的参数也变了:

00401137 04401137 04401132 04401145 04401145 04401145 04401145 04401145 04401157 04401157 04401157 04401157 04401157 04401157 04401157 04401157 04401161 04401161 04401162 04401174 04401174 04401192 04401195 0401195 04001195 0400195 0400195 0400195 0400195 0400000000000000000000000000000000000	 0000 74 33 89 1F000000 89000000 8300 8360 70304000 8360 70304000 8360 70304000 8360 70304000 8360 70304000 8200 204000 63 ED0300000 FF35 94304000 E3 20000000 FF35 94304000 E3 20000000 FF35 10204000 FF35 10204000 FF25 00204000 FF25 00204000	UP EHORE HA BE SHORT Register.0040116 MOU ECX, 1F MOU ECX, 1F MOU ECX, 1F MOU ECX, 1F MOU ECX, 1F MOU BEY.0 MOU BYTE PTR DS: LESI+403 INC ESI LOOPD SHORT Register.004 PUSH SED PUSH OWORD PTR DS: L403094 CALL (JMP.&user32.SetD191 RETM INTS JMP DWORD PTR DS: L403094 JMP DWORD PTR DS: L403094	C 4030701 070],AL 01145 1 temTextR> 2.DialogBoxParamA>1 2.EntD[sitem>1 2.GetD[sitem>1 2.GetD[sitem>1 2.SetFocus>1 2.SetFocus>1 132.EwitProcess>1 132.EwitProcess>1 132.GetModuleHandled	Text = "This program is but is by inder SetDigitemTextA user32.DialogBoxParamA user32.EndDialog user32.GetDigitemTextA user32.GetDigitemTextA user32.MessageBoxA user32.SetFlous kernel32.SetFlous kernel32.ExitProcess kernel32.GetModuleHandle	BCX`x0C^IK er Me 1',c	E_XI^IH\r" Lass="#32770")
00401145=	Register.00401145					
Address 00403070 00403090 00403090 00403080 00403080 00403080	Hex dunp 54 68 69 73 20 78 72 42 43 58 60 52 49 48 56 90 48 90 82 93 95 50 90 90 90 90 90 90 90 50 90 90 90 90 90 90 90 30 90 90 90 90 90 90	6F 67 72 61 60 28 69 78 28 45 5F 56 49 55 49 52 49 32 31 32 32 32 32 32 32 </td <td>Decti This program is </td> <td></td> <td>0012FAA0 0012FAA4 0012FAA8 0012FAA6 0012FAB0 0012FAB4 0012FAB4 0012FAB4</td> <td>00001075 REIDRI to fee 00127800 773404E7 RETURN to use 00050382 00000111 00000328 00050832 00050832 00050832</td>	Decti This program is 		0012FAA0 0012FAA4 0012FAA8 0012FAA6 0012FAB0 0012FAB4 0012FAB4 0012FAB4	00001075 REIDRI to fee 00127800 773404E7 RETURN to use 00050382 00000111 00000328 00050832 00050832 00050832

单步执行完这个循环,就会看到最后生成的消息,看起来相当属性呀,"This program is not registered!"。这和程序事实上还没有注册时主窗口中显示的消息是一样的:

00401143	. 3300	XUR EHX, EHX	
00401145	> 8886 7030400	MOV AL, BYTE PTR DS:[ESI+403070]	
0040114B	. 83F0 2C	XOR EAX, 2C	
0040114E	. 8886 7030400	MOV BYTE PTR DS:[ESI+403070],AL	
00401154	. 46	INC ESI	
00401155	-^ E2 EE	LOOPD SHORT Register,00401145	
00401157	. 68 70304000	PUSH Register,00403070	FText = "This program is not registered!"
0040115C	. 68 ED030000	PUSH 3ED	ControlID = 3ED (1995)
00401161	. FF35 9430400	PUSH DWORD PTR DS: [403094]	hund = 000503B2 ('Register Me 1', class='#32770')
00401167	E8 2000000	COLL (MP. Suser32, SetDigItemText0)	SetDialtemTevt9
0040116C	> 13	RETN	
00401160	CC.	INTO	
0040116E	E- EEDE 2420400	MP DUOPD PTP DC+ F/Succes22 DislogPouPasan0\1	usen22 DislogPouPanan0

可以看到这个字符串变成了传递给 SetDlgTextItemA 的值,事实上用之前在 那里的坏消息替换了已注册的好消息:

00401154	• 46	INC ESI	
00401155	•^ E2 EE	LOOPD SHORT Register.00401145	and the second
00401157	. 68 70304000	PUSH Register,00403070	flext = "This program_is not registered!"
0040115C	. 68 ED030000	PUSH SED	ControlID = 3ED (1005.)
00401161	. FF35 94304000	PUSH DWORD PTR DS: [403094]	hWnd = 000503B2 ('Register Me 1',class='#32770')
00401167	. ES 2000000	CALL <pre></pre>	SetDigItenTextA
0040116C	L> C3	RETN	
0040116D	CC	INTS	
0040116E	\$- FF25 24204000	JMP DWORD PTR DS:[<&user32.DialogBoxParamA>]	user32.DialogBoxParamA

下面就是主窗口中显示的:

	This program is not	registered
1	12121212	
	Check Serial	Quit

所以,现在我们知道了,给该应用打补丁的巧妙的方法是进入到序列号检测子程序,确保它总是返回正确的值,因为它不只是在第一次检测时被调用, 而且在显示成功后再次被调用。再提醒你一下,序列号检测的相关 CALL 被调 用,然对 eax 进行 0 测试。如果不是 0,就跳到坏消息,所以我们想让子程序返 回 0! 然后,序列号检测子程序再次被调用,如果它再次返回 0,那么我们的第 二次检测就通过了:



那么,咱们去序列号检测子程序那,看看能对它做些什么。子程序的开始 调用了 GetDlgItemTextA,我们猜它就是获取我们输入的序列号。你可以在 401101(它指向的是放置文本的 buffer)的参数上右键,在数据窗口中跟随它:

004010FC 004010FC 00401101 00401106 00401108 00401111	68 00020000 68 98304000 68 E9030000 FF35 94304000 E8 6A000000 E8 6A000000	PUSH 200 PUSH Register,00403098 PUSH 3E9 PUSH DWORD PTR DS:[403094] CRLL_{UPP.&user32.GetDigItemTextA>	Count = 200 (512.) Buffer = Register.00403098 ControlID = 3E9 (1001.) hWnd = 005F0410 ('Register Me 1',class='#32770') GetDigItemTextA
00401118 00401118 00401110 00401121 00401123 00401128 00401128	85 98304000 8500 66:3D 3433 75 07 85 00000000 EB 05 88 01000000 C3 05 02000000 C3 05 02000000	MOV EAX, DWORD PTR DS:[EAX] CMP RX, SB34 UNZ SHORT Register.0040112A MOV EAX,0 UNF SHORT Register.0040112F MOV EAX,1 RETN	Get entered serial

我们单步步过 GetDlgItemTextA 指令后,就能在 buffer 中看到我们的序列 号了:

Address	Hex dump	ASCII
00408098	31 32 31 32 31 32 31 32 00 00 00 00 00 00 00 00 00 1	12121212
004030A8	00 00 00 00 00 00 00 00 00 00 00 00 00	
084030B8	00 00 00 00 00 00 00 00 00 00 00 00 00	
00403008	00 00 00 00 00 00 00 00 00 00 00 00 00	
00403008	00 00 00 00 00 00 00 00 00 00 00 00 00	**************
004030E8	00 00 00 00 00 00 00 00 00 00 00 00 00	
004030F8		
00403108		
00400110		
00403120		
00100110	00 00 00 00 00 00 00 00 00 00 00 00 00	

在它被保存到 buffer 后,该 buffer 的起始地址被拷贝到 EAX 中,随后该地 址中的内容被拷贝到 EAX 中。就是将我们密码的前四个字节拷贝到 EAX 中。 然后这几个字节与 3334 进行比较,如果不匹配,EAX 就被填充为 1 (坏消息), 否则就填充为 0 (好消息):

00401111	. E8 6A000000	CALL <jmp.&user32.getdlgitemtexta></jmp.&user32.getdlgitemtexta>	GetDigItemTextA
00401116 00401118 00401110 00401121 00401123	 B8 98304000 8800 66:3D 3433 75 07 88 00000000 	MOU EHX, Negister, 00403048 MOU EAX, DUORD PTR DS:[EAX] CMP AX, 3334 UNZ SHORT Register 0040112A MOU EOX 0	EAX == 0
00401123 00401129 0040112F 0040112F	→ EB 05 → B8 01000000 → C3 ► E8 C7FFFFFF	MP SHORT Register.0040112F MOU EAX,1 RETN COLL Register 004010FC	EAX == 1

我们可以看到,做主要决定的是 401121 的 JNZ 指令:



这一行决定了在返回前, EAX 到底是 0 还是 1。所以我们要做的就是保证 EAX 总是等于 0:

004010EB 004010F0	. 68 ED030000 . FF35 94304000	PUSH SED PUSH DWORD PTR DS: [403094]		ControlID = SED hWnd = 003B0D44	(1005.) ('Register Me	2 1', class='#3
004010F6 004010FB 004010FC	L: C3 C3 C\$ 68 00020000	CHLL KUMP.&user32.SetDigitem RETN PUSH 200	Assemble at 0040	1122		x
00401101 00401106 0040110B	. 68 98304000 . 68 E9030000 . FF35 94304000	PUSH Register.00403098 PUSH 3E9 PUSH DWORD PTR DS:[403094]	nop			- +*#E
00401116 00401118 00401118 00401110	. B8 98304000 . 8800 . 66:3D 3433	MOV EAX, Register, 00403098 MOV EAX, DWORD PTR DS:[EAX] CMP AX, 3334	Fill with NOP's		Assemble	Cancel
00401122	90 BS 00000000	NOP MOV EAX,0			Assemble	Cancer
0040112A 0040112F	> B8 01000000	MOV EAX, 1				
00401130	. 0BC0	OR EAX, EAX				

所以现在,代码将总是直接给 EAX 赋 0 值,然后直接跳转到返回处。运行 下程序看看:



注意在对序列号检测子程序调用后,我们自然而然的就跳转到好消息那了:



在第二个检测点,我们也跳到了好消息那:

Regis	ter Me 1	X
Γ	This program is regist	tered!
	12121212	_
	Check Serial	Quit
	YEAH !!!!!!	

我们现在已经找到了一个注册该程序的补丁,无论你输入什么序列号都行 。

祝贺你!

四、ASCII 码表插件

你需要做的一件事是找出密码是什么(或对密码有什么样的要求)。给你些帮助,下载并安装"Ascii Table"插件,将其拷贝到插件目录。重启 Olly 后,选择"plugin"->"Ascii table"就会显示一个表格。尽管它还有很多地方需要改进,不过它能让你快速查询 ASCII 值:

ASCII T	able		1.0	1	 ×
3D	061	075	00111101	-	*
3E	062	076	00111110	>	
SF	063	077	00111111	2	
40	064	100	01000000	e	
41	065	101	01000001	A	
42	066	102	01000010	в	
43	067	103	01000011	C	
44	068	104	01000100	D	
45	069	105	01000101	E	
46	070	106	01000110	E	
47	071	107	01000111	G	
48	072	110	01001000	H	
49	073	111	01001001	1	-
4A	074	112	01001010	J	-
4B	075	113	01001011	K	
4C	076	114	01001100	I.	
4D	077	115	01001101	М	
4E	078	116	01001110	N	
4 <i>E</i>	079	117	01001111	0	
50	080	120	01010000	P	
51	081	121	01010001	2	
52	082	122	01010010	R	
53	083	123	01010011	s	
54	084	124	01010100	T	
55	085	125	01010101	U	
56	086	126	01010110	X	
57	087	127	01010111	W	
58	088	130	01011000	X	
59	009	101	01011001	1	
DA ED	090	102	01011010	4	
50	092	134	010111011	L.	
50	092	135	01011100		
58	094	136	01011110	1	
SE	095	137	01011111		
60	096	140	01100000	1.8	
61	097	141	01100001	a	
62	098	142	01100010	b	
63	099	143	01100011	e	
64	100	144	01100100	d	-

如果有人想要主动更新或重做这个插件,我将永远感激。第一,那些文 本不应该被选中,也不应该可编辑(我为什么要编辑 ASCII 码表?)。第二,让 窗口大小可变真是件好事。如果有人做了,请告诉我,我欠你一辈子。

第十三章:破解一个真正的程序

一、简介

本章我们打算不训练了,咱们来破解一个真正的程序。这个程序有个时间 限制,过了这个时间,这个程序就不能用了。我们准备给它打补丁,让它认为 是注册过的。目标文件在下载中有(我没有提及程序的名字,因为教程的目的 不是为了拿到一个"破解版"程序,只是为了学习)。与所有的商业软件一样, 如果你真的打算用它们,你真的应该考虑购买它。人们在软件中投入了大量的 时间,他们应该得到补偿。为了不让这个系列教程成为关于"获得破解版软件" 的东东,我试着找了一个没有人真想要的程序,所以我下载了这个软件,它是 上周 Download. com 中拥有最少下载量的软件。作为一个完全诚实的人,在本 章中破解了这个程序以后,我很喜欢这个程序,所以我买了一个注册码,现在 我心安理得的用它(译者注:作者真是活雷锋,其实咱们都是搞技术的,或多 或少都写过代码,尊重软件作者,为他们的劳动付费,其实就是尊重自已。实 在不愿意花钱的,就用免费替代软件行了,我一般喜欢用开源免费软件。多说 了几句哈)。只是告诉你,你不能通过下载量来判断一个应用。

你可以在教程页下载相关文件及本文的 PDF 版。

好,咱们继续...。

二、研究该应用

首先安装软件。安装完成后,会弹出下面这个窗口:

TreeDBNotes Pro 4 (4.25) S	Setup
TRIE	Completing the TreeDBNotes Pro 4 (4.25) Setup Wizard
	TreeDBNotes Pro 4 (4.25) has been installed on your computer.
	Click Finish to close this wizard.
	Run TreeDBNotes Pro 4 (4.25)
TREEDBNOTES PRO	Visit Web Site: http://www.mytreedb.com/
	< Back Finish Cancel

让"Run the app"保持勾选状态,看看会遇到什么:



好吧,看起来不是很好啊。我们注意到这里有几个字符串可能会有帮助, "unregistered"、"evaluation"、"registered"等等。点OK,然后弹出主界面:



注意,标题栏中显示的是"unregistered"。通常,我注意一个程序的另一个地方就是它的关于对话框。它通常都包含有字符串,以及用于逆向的思路。

这时候,我们寻找关键字、可识别的方法调用,以及类似的东西。这样的工作 你做的越多,就会有更多的线索。



这里我们又看见了"unregistered"。我通常做的下一件事是,找找看有没 有什么地方用来输入注册码。如果"搜索字符串"这招不好用的话,那么对于 渗透来说这是一个好入手点:



下面是输入注册码的地方:

Registration	
Enter the registration information to register	r the full version
Registration Name	
	Paste
Registration Code	
	Paste

输入一个试试,看看什么情况:

gistration	
Registration Enter the registration information t	o register the full version
Registration Name R4ndom	Paste
Registration Code	Paste
	OK Cancel





唉! 我好像从来就没有输对过☺。好吧,对于我们当前搜集到的信息来说, 我们有一个相当好的方法, Olly 载入程序:

009F6098	\$	55	PUSH EBP	
009F6099	1.1	SBEC	MOU EBP, ESP	
009F609B	1	53U4 FØ	PUSH EBX	
009F609F		B8 204F9F00	MOV EAX, TreeDBNo, 009F4F20	
00956004	1	881D D0060200	MOU EBX, DWORD PTR DS: [8206D0]	TreeDBNo, 00921BE8
009F60AF		3309	XOR ECX, ECX	11 CODDIOL CONCLEDIO
009F60B1	•	B2 01 01 70F29700	MOU EOX DWORD PTP DS+197E2701	
009F60B8		E8 275EAAFF	CALL TreeDBNo, 0049BEE4	the submer least states
009F60BD		8B15 5405A200	MOV EDX, DWORD PTR DS: [A20554]	TreeDBNo.00A26764
009F60C5		A1 5405A200	MOV EAX, DWORD PTR DS: LA20554]	Kernetoz, base infedurito
009F60CA		8800	MOV EAX, DWORD PTR DS: [EAX]	
009F60D1		A1 5405A200	MOV EAX, DWORD PTR DS: [A20554]	
009F60D6		8B00	MOU EAX, DWORD PTR DS: [EAX]	
009F60DA	1	FF92 8000000	CALL DWORD PTR DS:[EDX+80]	
009F60E0		8B03	MOV EAX, DWORD PTR DS: [EBX]	
009F60E7	1.	8B03	MOV EAX, DWORD PTR DS: [EBX]	and the second second second
009F60E9		BA CC619F00	MOV EDX, TreeDBNo. 009F61CC	ASCII "TreeDBNotes"
009F60F3	-	8800 7C09A200	MOV ECX, DWORD PTR DS: [A2097C]	TreeDBNo.00A2632C
009F60FB	1	8815 D4AF8E00	MOV EDX, DWORD PTR DS: [8EAFD4]	TreeDBNo.008EB020
009F6106	1	SBOD OCFFA100	MOU ECX, DWORD PTR DS: [A1FF0C]	TreeDBNo.00A26874
009F610E	1	8815 98229D00	MOV EDX, DWORD FTR DS: [9D2298]	TreeDBNo.009D22E4
009F6119	1	8800 0000A200	MOV ECX, DWORD PTR DS: LA200001	TreeDBNo.00A267A4
009F6121	1	8815 <u>D0999800</u>	MOV ERA, DWORD FTR DS: LEBAJ	TreeDBNo.00989A1C
009F612C	1	SBOD FS00A200	MOU ECX, DWORD PTR DS: [A200F8]	TreeDBNo.00A248F0
009F6184	1	8815 58148800	MOV EDA, DWORD FTR DS: [881458]	TreeDBNo.008B14A4
009F613F		880D 1403A200	MOU ECX, DWORD PTR DS: [A20314]	TreeDBNo.00A263E4
009F6147	1	8B15 28FB9100	MOV EDX, DWORD PTR DS: [91FB28]	TreeDBNo.0091FB74
009F6152		880D 5802A200	MOV ECX, DWORD PTR DS: [A20258]	TreeDBNo.00A263F0
009F615A	1	8815 88019200	MOU EDX, DWORD PTR DS: [9201B8]	TreeDBNo.00920204
009F6165	-	SBOD SCFFA100	MOU ECX, DWORD PTR DS: [A1FF5C]	TreeDBNo.00A26360
009F616D		8B15 ACE78E00	MOU EDX, DWORD PTR DS: [8EE7AC]	TreeDBNo.008EE7F8
009F6178	2	880D 2802A200	MOU ECX, DWORD PTR DS: [A20228]	TreeDBNo.00A26358
009F6180		8815 <u>A8E68E00</u>	MOV EDX, DWORD PTR DS: [8EE6A8]	TreeDBNo.008EE6F4
009F618B		880D 3408A200	MOU ECX, DWORD PTR DS: [A20B34]	TreeDBNo.00A2676C
009F6193		8815 E8EA9700	MOU EDX, DWORD PTR DS: 197EAE81	TreeDBNo,0097EB34
009F619E	1	A1 0000A200	MOV EAX, DWORD PTR DS: [A20000]	
009F61H3	1	E8 1E71FAFF	CALL TreeDBNo.0099D2C8	
009F6188		A1 5405A200	MOU EAX, DWORD PTR DS: [A20554]	
009F61B1	:	E8 7EAØAAFF	CALL TreeDBNo.004A0234	
009F61B6		8803	MOV EAX, DWORD PTR DS: LEBX]	

你可能已经注意到了,这看起来和我们已经见过的大部分应用有点不一样。 看起来有辣么多的 CALL 指令,没有那些典型的 Windows 设置的玩意儿(像 RegisterClass...)。这是一个好的标志,说明程序是用 Delphi 写的。Delphi 在程序中会使用大量的 CALL。我们可以通过运行一个 ID 程序来确定,不过我打算在后面的教程中讨论。也有一些专门的工具用来处理 Delphi 程序,不过幸运星是本章我们不需要用专用工具(虽然我们会接触到它们。)。

三、寻找补丁

试试咱们的字符串搜索。右键,选择 "Search for" -> "All referenced text strings",将会弹出搜索窗口。滚动到顶部然后右键,选择 "Search for text":

Address	Disassembly	Text string		
00401000 00401006 0040101B	DD TreeDBNo.00401004 ASCII "Boolean" ASCII "False"	Initial	Follow in Disassembler	Enter
0040102E 0040102E 00401046	ASCII "True" ASCII "WideChar" ASCII "Char" OSCII "Smallist"		Search for text	
0040105H 00401072 0040108A 0040109E	ASCII "Integer" ASCII "Integer" ASCII "Byte"		Set breakpoint on every command	ł
004010B2 004010C2 004010C2	ASCII "Extended" ASCII "Cardinal" ASCII "Int64"		Set log breakpoint on every comm	hand
004010F6 00401106	ASCII "Double" ASCII "Currency" ASCII "WardBool"		Copy to clipboard	
0040112C 00401132	ASCII "False" ASCII "True"		Appearance	>
0040113E 0040114A 0040115A	ASCII "String" ASCII "WideString" ASCII "Variant"			_
0040116A 0040117A 00401180	ASCII "OleVariant" ASCII "HRESULT" DD TreeDBNo 00401108	OSCIL 07 "T	Object"	
004011AC 004011D9	DD TreeDBNo.004011D8 ASCII "TObject"	ASCII 07,"T	Object"	
004011E6 004011ED 004011F8 00401206 00401226 00401226 00401236	HSCII "TUDJect" DD TreeDBNo.004011D8 ASCII "System" ASCII "IInterface" ASCII "System" ASCII "Dispatch"	ASCII 07,"T	Object"	
00401255 004012B0 004012D0 004012D0 004012FD	ASCII "System" DD TreeDBNo.004012FC DD TreeDBNo.004012FC ASCII "TInterfacedObjec"	ASCII 11."T	InterfacedObject" InterfacedObject"	

弹出文本搜索对话框。现在我们注意到"registration"和"registered"很早就用到了,所以咱们就搜它们。通常在这种情况下,因为是第一次搜索,我会搜"regist",因为包含了这两个单词,而且也从来没有让我失望过(我猜没有多少程序会使用单词 "register" (29)。不要勾选 "Case sensitive",选中 "Entire scope",然后点 OK:

Enter text to search for		×
tegister		•
Case sensitive		
F Entire scope		
	OK	Cancel

第一个命中的看起来没啥前途,按 CTRL+L 继续搜:

AAAAA LIGH PRU P. BRU AA JAAAAA	11644 841 108	
H2002 MOV EDX, TreeDBNO, 004H20/C	ASCII "Default"	
A207C ASCII "Default",0		
A241C PUSH TreeDBNo. 004A2524	ASCIT "MAINICON"	
A2524 ASCII "MAINICON",0	10011 1111000	
A2FAF MOV EAX, TreeDBNo.004A32AC	ASCII "voltest3.dll"	
A2FD0 PUSH TreeDBNo,004H32BC	ASCII "RegisterAutomation"	
A32BC ASCII "RegisterAutomati"		
A32CC ASCII "on".0	the second se	
A3D90 ASCII ".",0	and a set of the second	
A4E01 PUSH TreeDBNo.004A4E24	ASCII "User32.dll"	
04F24 0SCTT "Heeps22 dil" 0	HSUII "SetLageredwindowHttributes"	
A4E30 ASCII "SetLayeredWindow"		
A4E40 ASCII "Attributes",0	and the second second second	
A4F00 PUSH TreeDBNo.004A4F1C	ASCII "TaskbarCreated"	
04F2C OSCII "need diction any" 0		
A4F3C ASCII "stream end".0		
A4F4C ASCII "file error",0		
A4F58 ASCII "stream error",0		
DAFASI DELTT "Hats error" U		100

注意,这次找到的就是我们第一次搜到的。因为第一次命中的是在字符串 被压到堆栈的地方,第二次才是字符串 "RegisterAutomation"在内存中的真实 的数据。因为在第二列中没有指令所以可以分辨出来,反而有个 ASCII 字样。 你遇到的大多数字符串都有两个版本,一个是字符串被访问的地方,另一个就 是字符串真正所在的地方:

Address	Disassembly	Text string	
004A2002	MOV EDX, TreeDBNo, 004A207C	ASCII "Default"	
004A2334 004A241C	ASCII "TApplication",0 PUSH TreeDBNo.004A2524 OSCII "MOINICON" 0	ASCII "MAINICON"	
004A2FAF 004A2FD0	MOV EAX, TreeDBNo.004A32AC PUSH TreeDBNo.004A32BC	ASCII "voltest3,dil" ASCII "RegisterAutomation"	
004832BC	ASCII "RegisterAutomati" ASCII "on".0		-
004A3D90 004A4E01 004A4E11 004A4E11 004A4E24	ASCII ".",0 PUSH TreeDBNo.004A4E24 PUSH TreeDBNo.004A4E30 ASCII "User32.dli",0 OSCII "Schlamaddida"	ASCII "User32.dll" ASCII "SetLayeredWindowAttributes"	
00404E30 00404E40 00404F00 00404F1C	ASCII "Attributes",0 PUSH TreeDBNo.004A4F1C ASCII "TaskbarCreated",0	ASCII "TaskbarCreated"	
004A4F2C 004A4F3C 004A4F4C	ASCII "need dictionary",0 ASCII "stream end",0 ASCII "file error",0		
004A4F58	ASCII "stream error",0		-

如果你再按一次 CTRL+L,我们会遇到另一个没前途的字符串。一直按 CTRL+L 直到来到下面这个地方:

Address D	isassemblu	Text string	
009A9F9F P 009A9FAB M 009A9FC4 M 009A9FC1 M 009A9FC1 M 009AA134 A 009AA134 A 009AA158 A 009AA158 A	USH TreeDBNo.009AA134 OV EAX,TreeDBNo.009AA144 OV EDX,TreeDBNo.009AA158 OV EDX,TreeDBNo.009AA158 OV EDX,TreeDBNo.009AA164 OV EDX,TreeDBNo.009AA134 SCII "SNoteS",0 SCII "IDFolder",0 SCII "IDFolder",0 SCII "IDFolder",0	ASCII "sNotes" ASCII "IDFolder" ASCII "ID" ASCII "NoteData" ASCII "sNotes"	-
0099ABBA9 M 0099ABBCC U 009AABBCC U 009AABBCC U 009AABBCC U 009AABEC U 009AACCC U 009AACC4 U 009AAC24 U 009AAC24 U 009AAC24 U 009AAC24 U	OU EDX, TreeDBNo.009AABCC OV EDX, TreeDBNo.009AAC14 NICODE "TreeDBNo" NICODE "tes Pro" NICODE "4.25 (Re" NICODE "9istered" NICODE "1",0 NICODE "TreeDBNo" NICODE "tes Pro" NICODE "tes Ston" NICODE "tegister" NICODE "register"	UNICODE "TreeDBNotes Pro 4.25 (Registered)" UNICODE "TreeDBNotes Pro 4.25 (Unregistered)"	
009AACC8 P	USH TreeDBNo.009AAD70 NU Fry TreeDBNo.009AB50r	UNICODE ": "	τ

这回看起来好多了。它将会在程序启动过程中的某个时刻出现,它会检测 我们有没有注册,然后根据检测的结果来决定窗口的标题栏显示注册还是没注 册。这是我们开始干活的好地方。双击有"registered"的那行,咱们就会跳到 相应的代码那:



首先我们能看到字符串是在 9AABA9 那,还能看到字符串存储在内存的 9AABCC 处。第二,要注意到是两个字符串是在同一个方法中,在它们的上面 有个一个条件跳转。点击 9AABA5 处的条件跳转:



我们能够看到如果结果相等,就跳到"Unregistered"那里。很明显,不能 让它跳。咱们在 JE 指令那设置一个 BP, 启动应用:

009AAB94 009AAB95 009AAB98 009AAB98 009AAB98 009AAB98 009AAB9C 009AAB9C	L. C3 8D40 00 \$ 55 8BEC 53 8BDA 88DA 88BB 88150000 00	RETN LER EAX, DWORE PTR DS: CERXI PUSH EBP MOV EBP,ESP PUSH EBX,EDX CMP BVTE PTR DS: CEAX+15B8],0 LE SHORT TOPEDEND, 009040806	TreeDBNo. <moduleentrypoint></moduleentrypoint>
009948847 009948849 00944888 00944883 00944885 00944885 00944885 00944885 00944885 00944885 00944853 00944853 00944853 00944853 00944853 00944855 00944855 00944855	BEC3 BEC3 E3 E569409 E3 E569405FF 58 50 C3 BA 14AC9400 E3 464945FF 55 55 C3 C3 0	HOU EAX, EEX HOU EAX, TreeDBNo, 009AABECC CALL TreeDBNo, 00405508 POP EBX POP EBP RETN HOU EAX, EEX HOU EAX, EEX HOU EAX, TreeDBNo, 009AAC14 CALL TreeDBNo, 00405508 POP EBP RETN DE 00 DE 00	UNICODE "TreeDBNotes Pro 4.25 (Registered)" kernel32.7747ED6C kernel32.7747ED6C UNICODE "TreeDBNotes Pro 4.25 (Unregistered)" kernel32.7747ED6C kernel32.7747ED6C

Olly 就会断在那行,你会发现我们就要跳到坏消息那了。咱们得修改下:

CRANST	010000	ESSS DESS	0023 001B 0023 0023 003B 003B	3251 3251 3251 3251 3251 3251 NULL
0	00	Las	stErr	ERRO

运行程序。Olly 会再次断在同一行,并准备跳到坏消息那。咱们再次就 O 标志位置 O, 然后运行程序。又来了一遍,清除 O 标志位后,我们最终得到如下的反馈:



所以那样做是不起作用的。给那个检测点打补丁不会注册成功,如果你点 OK并再一次将标志位置 0,你会发现主窗口的"unregistered"没有了:

TreeDBNotes Pro 4.25 (Register	ed) (9-Jan-2012)
Open demo file View a demo document	
Open Open an existing document	
New Create new document	
Visit Homepage Visit program homepage	
Buy Registration Key Buy a registration key for the program	

那么,我们知道至少没有跟踪错。我们准备做的是步入到下一"层",做深入的研究。重启应用,然后断在了我们的断点处,咱们再多做些研究:



在比较的前面并没有 CALL, 但是在 JE 指令的前面 9AAB9E 处有个比较:

CMP BYTE PTR DS: [EAX+15B8], 0

所以,这个比较的结果决定了我们是注册还是未注册。EAX+15B8只是一个内存地址,在这里它是一个全局变量,因为是以 DS: 打头的。我们希望这是程序检测注册与否的唯一一个检测点。如果不是的话,我们就需要找出其他检测点的位置。点选比较指令,就可以看到 EAX+15B8i 是什么:

DS:10	18011103=00	1	_
00100			

在地址上右键,选择 "Follow in dump":

Address	Hex dump	ASCII
01AC111C 01AC113C 01AC113C 01AC113C 01AC114C 01AC114C 01AC114C 01AC114C 01AC114C 01AC114C 01AC119C 01AC119C 01AC119C 01AC119C 01AC112C 01AC114C 01AC114C 01AC114C 01AC121C 01AC122C	90 90<	0.0. 20 ↓ 23 ↓ =B%093 ↓ 4=4 2 ↓ 24 ↓ 2 ↓ 26 0 m= #6.920 ↓ 26 0 m= #6.920 ↓ 26 0 m= #6.920 1 H. dr%0 X2 I. dr%0 0 1 H. dr%0 X2 I. dr%0 0

你的地址几乎肯定和我的不一样。这个没事。跟着教程,将我的地址替 换成你的地址,一样跑的好好的。

这里我们能看到该地址被用于检测注册与否,就是 1AC111C 处(至少我这 里是这样)的第一个 00。意思是如果此内存位置的内容是除了 0 以外的任何数 据,那么就假定我们已经注册。这也意味着,程序中可能有别的子程序检查该 内存位置,这就是为什么主窗口显示 "Registered",而程序的其他部分知道我 们没有注册。因为我们只是在检查了内存内容后绕过了这个子程序的自然流程, 其他子程序的检测却没有绕过。 首先,咱们把这个内存地址设置为非0,那么我们知道至少这个子程序将会按照我们想要的方式工作。在比较那行(9AAB9E)设置一个断点,将其他断点 删掉。重启应用后 Olly 就断下来了。在比较那行上右键,选择 "Follow in dump" -> "Memory location",因为 Olly 会在我们重启应用的时候重置数据窗口。你可能已经注意到了,比较指令检查的内存地址这次变了:

Address	Hex dump	ASCII
01891111C 0189112C 0189113C 0189114C 0189115C 0189115C 0189117C 0189117C 0189118C 0189118C 0189118C 0189118C 0189118C 0189118C 0189118C 0189112C 0189128C 0189118C 0189128C 000000000000000000000000000000000000	00 00 00 00 FF FF FF 00 <td< th=""><th>0.0 20 ±3" ±3" ±3" ±4°± &2↓₩0°=₩0.940 ±4°±0.</th></td<>	0.0 20 ±3" ±3" ±3" ±4°± &2↓₩0°=₩0.940 ±4°±0.

第一次是 1AC111C, 现在是 01B9111C。你的和我的会不一样, 你只需要 注意第二次的就行, 存储 已注册/未注册 标志的内存地址不同。

点击数据窗口中的 "00"(在我的数据窗口中是 1B9111C), 右键选择 "Binary" -> "Edit":

Edit data a	rt 01D4111C
ASCII	[]
UNICODE	?
HEX +00	3 0
I ⊂ Keep	size OK Cancel

咱们输入01:

Edit data	a at 009F7000	×
ASCII UNICODE	© DE ?	
HEX +01	1 01	
I ⊂ Keep	ep size	Cancel

注意数据窗口中的内容已经更新了:

Address	Hex dump	ASCII
Address 9189111C 9189112C 9189113C 9189114C 9189114C 9189114C 9189114C 9189114C 9189114C 9189114C	Hex dump 01 00 00 0F FF FF FF 00 0	ASCII 0
0189118C 0189110C 0189110C 018911EC 018911EC 0189120C 0189120C	00 00 00 00 00 00 00 00 00 00 00 00 00	¶)H.duŋ0NXzI. duŋ6 0 .l♥₩♥. ¶. 0

继续运行程序直到再一次断下来。你会发现内存中的内容又变回了 0,我们 将再一次跳到坏消息那。这意味着程序的某个地方,做了第二次检测并将注册 与否的标志重置为 0。我们需要做的就是找出在哪里重置的,确保不会再被重置。 要这样做的话,在该内存位置设置一个硬件断点,当程序向该内存位置写数据 时让 Olly 断下来。之所以选择"写",是因为某个地方向该内存写了 0。

重启应用直到它断下来。右键比较指令,选择"Follow in dump",因为 Olly 又重置了数据窗口。用二进制编辑方法将第一个内存位置修改为 01。注意 它现在的地址又换了:



右键数据窗口中的第一个值,选择 "Breakpoint" -> "Hardware, on write" -> "byte":

Address	Hex d	lump			at the set of the	ASCII			0012FDA0	. 042
0188111C 0188112C 0188113C 0188114C 0188114C 0188116C 0188116C 0188117C 0188119C 0188119C		Backup Undo selection Copy Binary	Alt+BkSp	* * * *	0 00 00 00 00 0 00 00 00 0 00 00 00 0 00 0	00 			0012FDH4 0012FDA8 0012FDB0 0012FDB0 0012FDB4 0012FDB4 0012FDB6 0012FDB6 0012FDC6 0012FDC6 0012FDC6 0012FDC4 0012FDC4	90 90 90 90 90 90 91 90 91 90 91 90 90
01BB11BC 01BB11CC 01BB11DC 01BB11EC 01BB11FC 01BB120C 01BB121C 01BB122C	G1-9000000	Breakpoint Search for Go to		* * *	Memo Memo Hardw	ory, on access ory, on write vare, on access			0012FDD0 0012FDD4 0012FDD8 0012FDD0 0012FDE0 0012FDE4 0012FDE4 0012FDE4 0012FDE4	04E 00E 04E 00E 00E 00E 00E
A THAIR	~	Hex Text Short Long		* * * * *	Hardw Hardw	vare, on write vare, on execution	•	Byte Word Dwor	l rd	

在逆向一个程序时,我通常留在硬件断点,因为它们很难被应用检测到。 我选择 "byte" 是因为我们想追踪的就一个字节。

运行程序。Olly 会再次断在普通断点, 你会发现我们输入的 01 仍然在那里, 所以到目前为止还不错。再运行, Olly 会断在一个新地方:



如果你看 OllyDbg 左下角的话,会发现我们断在了硬件断点。

四、给程序打补丁

现在,咱们来研究研究这块代码。第一条指令是将 DL 与我们刚才编辑的内存内容进行比较,如果相等就跳到 9ADC02,然后就返回了。如果不相等,就将 DL 的内容存储到我们编辑的内存中。我们已经知道了 DL 等于 0,因为我们看到内存中的值从 01 变成了 00。所以这基本上就是另一个注册检测点,并且如果它检测失败就会将 已注册/未注册 标志置 0。如果成功,就什么都不做。现在咱们将硬件断点删掉,选择"Debug"->"Hardware breakpoints" (译者注: 这里的 Debug 指的是菜单中的),将硬件断点删除。咱们在 9ADBF4 处设置另一个硬件断点,这样我们可以在该段代码运行前断下来:

DB 00 CMP DL, BYTE PTR DS: LEAX+15B81 JE SHORT TreeDBNo.009ADC02 MOV BYTE PTR DS: LEAX+15B8], DL RETN NOP PUSH EBP MOV EBP, ESP ADD ESP, -8 PUSH EBX PUSH EDI PUSH EDI PUSH EDI MOV LLOCAL.1], EDX MOV EBX, EAX MOV EBX, EAX MOV EBX, EAX MOV EBX, EAX MOV EBX, EAX	Backup Copy Binary Assemble Label Comment	ہ پ Space :	ESI 0000000 EDI 01B0FB64 EIP 0099DC02 TreeD C 1 ES 0023 32bit P 1 CS 0018 32bit A 1 SS 0023 32bit S 1 FS 0028 32bit T 0 GS 0000 NULL D 0 D 0 LastErr ERROR EFL 00200297 (NO,B ST0 empty -??? FFF ST1 empty -??? FFF	SNo.009ADC02 0(FFFFFFF) 0(FFFFFFF) 0(FFFFFFF) 0(FFFFFFF) 7FFDF000(4000) _SUCCESS (0000000 NE, BE, S, PE, L, LE) F 0000003S 00330 003300
MOV EDX, DWORD PTR DS: [508080]	Breakpoint	•	Toggle	F2
IMUL EDI, LARG.21 ADD EDI, LOCAL.21	Hit trace	•	Conditional	Shift+F2
MOV ESI, DWORD PTR DS: [EBX+CA0 MOV DWORD PTR DS: [ESI+C], EDI	Run trace		Conditional log	Shift+F4
MOV EAX,ESI CALL TreeDBNo.004E49B0 MOV EAX,EBX	New origin here	Ctrl+Gray *	Run to selection	F4
POP EDI	Go to		Memory, on access	
POP EBX	Thread	¥.	Memory, on write	1
POP ECX POP EBP RETN 8 NOP PUSH EBP	Follow in Dump View call tree	► Ctrl+K	Hardware, on execution	

你或许会纳闷,我为什么不在这里设置一个普通断点。因为我先试过了! Olly 根本就不会断下来好嘛! 有几个愿意可能会导致该问题的发生: 这段是多 态代码,所以我们的 BP 丢了,程序检测到软件断点所以把它删了,断点在一个 Olly 不会自动追踪的区块...。不管怎样,就是这么个结果。我们需要设置硬件 断点而不是软件断点。不保证硬件断点一定管用,因为软件有可能专门对它们 进行检测。不过它是一个更可靠的设置断点的方法,所以通常来说还是比较好 用的。

在后面的章节中我们会更多的学习反调试技巧。

重启应用,我们会再次断在新的硬件断点处:



好,现在咱们来思考思考啊。这个子程序是在咱们原来的断点前面被调用。 这个子程序检测我们是否注册,如果没有就将[EAX+15B8]地址处的内容设置为 0,如果注册了就置为 01 (或者任何非 0 的数据)。然后我们原来的子程序被调 用,就是那个在窗口标题中输出"Registered"或"Unregistered"的子程序, 它也是根据内存中的数据是 0 还是 1 来决定输出。如果我们确保任何时候只要 该子程序运行时那个内存位置中都是 1,那么任何其他子程序来检测内存中内容 时看到的都只能是 1,也是就认为我们已经注册了。

如果我们只是将子程序修改成总是在内存中的合适位置放置1的话会怎么样? 咱们来试试看。

下一个问题就是怎么做最简单。好,我们已经有了在 9ADBFC 处被用某些 值(DL)填充的内存位置,所以我们只需要在上面的某个地方将 DL 改成 1。问 题是将 DL 改成 1 需要在当前指令的长度上加一个字节,这样做会覆盖 RETN 语句。如果我们将比较/跳转指令替换成将 DL 置为 01 的指令怎么样。那样的话, 在最后一行, DL 将被拷贝到我们的内存位置!下面就是我们的做法,选中比较 /跳转那两行指令:

009ADBF3		00	DB 00	
009ADBF4 009ADBFA	15	3A90 B8150000 74 06	CMP DL, BYTE PTR DS: EEAX+15B81	
009ADBFC 009ADC02 009ADC03 009ADC03		8890 B8150000 C3 90 55	MOU BYTE PTR DS:/ERX+15B81, DL RETN NOP PUSH EBP	

右键选择 "Binary" -> "Fill with NOPs":

009HDBF0 009ADBF1 009ADBF2 009ADBF3	80 80 80 80	06 60 DB 00 DB 00 OB 00	CHHK ' I'		E	DI 00989A1C IP 009ADBF4 0 ES 0023	TreeDBNo TreeDBNo 3251t 00
009ADBFA 009ADBFC 009ADBFC 009ADC02 009ADC03 009ADC03	r\$ 3A90 88150000 ~ 74 06 8890 88150000 C3 90 r. 55	CMP DL.BYTE PTR DS:[ERX+15 JE MORT TyseDBMD, DD9AD MOU BYTE PTR DS:[ERX+15 RETN NOP PUSH EBP	Backup Copy	*		1 CS 001B 0 SS 0023 1 DS 0023 0 FS 003B 0 GS 0000	3251t 0(3251t 0(3251t 0(3251t 7F NULL
009ADC05 009ADC07	: 88EC : 83C4 F8	MOV EBP, ESP ADD ESP, -8	Binary		Edit	Ctrl+E	RROR_SU
009ADC0B 009ADC0C	· 55 · 56 · 57	PUSH ESI PUSH EDI	Assemble	Space	Fill with 00's	~	10 March
009ADC0D 009ADC10	: 894D F8 : 8955 FC	MOU FLOCAL.23,ECX MOU FLOCAL.13,EDX	Label	:	Fill with NOPs	-	
009ADC15 009ADC15 009ADC18 009ADC18	8845 FC 8815 <u>80805000</u> E8 9564A5FF	MOV EAX, LLOCAL.13 MOV EAX, LLOCAL.13 MOV EDX, DWORD PTR DS: L5 CALL TreeDBNo, 00404088	Comment	7	Binary copy		10000000
009ADC23 009ADC2A	. 0FB6B8 19010000 . 0FAF7D 0C	MOVZX EDI, BYTE PTR DS: [IMUL EDI, LARG. 2]	Breakpoint	1	2	ST 4000 Co	321 321
009ADC2E 009ADC31	. 037D F8 . 38B3 A00C0000	ADD EDI, ILOCAL.21 MOV ESI, DWORD PTR DS: [E	Hit trace		ŧ	CW 1372 Pr	NEAR, 6
DO MOUSE	1 0712 00	HOW DWOND FIN DOLLEDING	Runtrace	5 I			

然后就像下面这样:

009ADBF2 009ADBF3	00 00	DB 00 DB 00	
009ADBF4 009ADBF5 009ADBF6 009ADBF7 009ADBF8 009ADBF8 009ADBF9 009ADBF8	90 90 90 90 90 90 90 90	NOP NOP NOP NOP NOP NOP	
009ADBFC 009ADC02 009ADC03 009ADC03 009ADC04 009ADC04	8890 B8150000 C3 90 55 88EC 88EC	MOU BYTE PTR DS:[EAX+15B8], DL RETN NOP PUSH EBP MOU EBP.ESP	

这一步不是必须的,不过这能让你更容易的看清自己正在干啥(译者注:这样可以防止不小心多添加或少添加字节)。

现在选中 9ADBF4 处的 NOP,按一下空格键。弹出汇编窗口,输入 MOV DL,1:

009ADBF3 009ADBF4 009ADBF5 009ADBF5 009ADBF7 009ADBF7 009ADBF8	80 90 90 90 90 90 90	DB 00 DB 00 NOP NOP NOP NOP NOP			
009ADBF9 009ADBFA 009ADBFB 009ADBFC 009ADC02 009ADC03 009ADC03 009ADC03 009ADC04 009ADC05 009ADC05 009ADC06 009ADC06	90 90 8890 B8150000 C3 90 55 8BEC 83C4 F8 53 56	NOP NOP MOV BYTE PT RETN NOP PUSH EBP MOV EBP,ESP ADD ESP,-8 PUSH EBX PUSH ESI	Assemble at 009ADBF4	Assemble	Cancel
009ADC0C 009ADC0D 009ADC10 009ADC10 009ADC13 009ADC15 009ADC15	57 894D F8 8955 FC 88D8 8845 FC 8815 R0805000	PUSH EDI MOV ELOCAL, MOV ELOCAL, MOV EBX, EAX MOV EAX, ELO MOV EDX, DUO	CR. 1] CR. 1] CR. PTR DS: [5080801	TreeDBNo. 0050B0FC	

先点 Assemble, 然后点 Cancel。结果就像下面这样:



现在,无论什么时候调用这个子程序,内存中的那个标志位(译者注:这 里的这个标志位和寄存器的标志位不一样啊,读者要能够分辨的了)都会被置1 而不是 0。因为我们仍然暂停在子程序的第一行,所以你可以单步执行以观察 DL 被置 1 的过程,然后 1 就被存入内存中(你可能需要到数据窗口的合适的地 址观察,因为 Olly 很可能再一次将数据窗口重置过)。现在运行程序,Olly 会断 在我们原来的断点:

009AAB95 009AAB96 009AAB99 009AAB9B 009AAB9B 009AAB9C	5	8040 00 55 88EC 53 88DA 89BA 8088 88150000 00	LEA EAX, DWORD FTR DS:[EAX] PUSH EBP MOU EBP,ESP PUSH EBX MOU EBX,EDX CMP BYTE PTR DS:[EAX+15B8].0	
009AABA7 009AABA9	:	BRC3 BA CCAB9A00	MOV EAX, EBX MOV EDX, TreeDBNo. 009AABCC	UNICODE "TreeDBNotes Pro 4.25 (Registered)"
009AABB3 009AABB3	•••	58 50	POP EBX POP EBP	042233A8 042233A8
009AABB5 009AABB6 009AABB8	5	C3 98BC3 BA 14AC9A00	RETN MOV EAX,EBX MOV EDX,TreeDBNo.009AAC14	UNICODE "TreeDBNotes Pro 4.25 (Unregistered)'
009AABBD 009AABC2 009AABC3	:	E8 46A9A5FF 5B 5D	CALL TreeDBNo.00405508 POP EBX POP EBP	042233A8 042233A8
009AABC5		00	DB 00	

我们能看到我们将直达正确的字符串。继续运行,我们将断在我们修改过 的注册检测子程序,它会如愿的放一个 01 到我们的内存地址中。这会回头再往 前这样来回几次直到最后:



现在我们注册成功了!!! 继续运行程序(打开一个 demo 文件), Olly 会在 注册子程序中断下来几次,不过每次它都会走正确的路。不久你就会看到主窗 口:



你会看到我们仍然是已注册状态。点击显示关于对话框:



恭喜你!你已经成功的完成了第一次破解。

别忘了将它保存到磁盘。打开硬件断点窗口("Debug" -> "Hardware breakpoints"),点断点边上的 Follow 按钮。然后我们就来到了我们打补丁的地方。选中所有我们修改过的行,右键选择"Copy to executable"。在弹出的窗口中右键,选择"Save to disk"。以原来的名字保存它。现在退出 Olly,运行程序体验它的全部,以及注册成功的骄傲与自豪!!!

第十四章: NAG 窗口(我不是在说你妈)^{举①}

一、简介

Nags,或者叫 Nag 窗口,是普通的消息对话框。它弹出来是提醒你你的试用结束了、你需要注册、关于访问网站的提醒...。基本上任何事它都要唠叨,而且还是不必要的(像大多数的 boss 一样。)。许多免费软件之所以免费,是因为它们充满了 nag (广告、限时试用、重定向)。商业软件通常也有这些玩意儿,提醒你"你只剩下 18 天来使用此产品"等等。在逆向工程领域,除掉 nag 窗口 是一个中心主题,有时候也提出了很多挑战。本章我'们将会研究两个有 nag 的程序。我们将会绕过它们,之后它们就再也不会显示了,然后再打上补丁,这样它们就永远不会回来了。

我也会介绍一个新的 Olly 插件,叫 IDAFicator。它有许多特点和设置。你可以在<u>工具</u>页下载该插件。因为它有如此多的特性,我也在下载中包含了 IDAFicator 作者写的教程。我强烈推荐你看看教程,因为该插件有许多非常酷的特性。

你可以在<u>教程</u>页下载相关文件以及本文的 PDF 版。

二、第一个应用程序

我们将要研究的第一个二进制文件是 Nag1. exe。程序一运行就会弹出 nag:



它明摆着告诉你这就是一个 cracker 写的⁴。不管怎么样,点了 OK 就可以 看到主窗口:

注①:标题中的"我不是在说你妈"可不是骂人的,因为 nag 有"唠叨"的意思,而英文的标题就是"Nags",没有其他多余的字,所以你懂的。

🔤 Brought to you by TDC, 2005
Remotethe CAG by TDCh
Remove the NAG by TDC
Status: Nag not removed! You have to remove the nag and make the program say it is registered when clicking Re-Check.
-={ :: HINTS :: }=- Close About Re-Check

注意,它说"Nag not removed!"。我情不自禁的就点了那个"Hints"按钮,然后给了一些非常详细的信息(译者注:我咱们没觉得很详细):



Gee,谢谢。Olly载入应用,咱们试试老方法——搜索字符串:

R Found strings are	
Bottoress Disascembly 004010000 PUSH 5 004010000 PUSH 5 004010000 PUSH 50 004010000 PUSH 1004030010 004010000 PUSH 1004030010 004010000 PUSH 1004030001 004010000 PUSH 1004030001 004010000 PUSH 1004030001 004010000 PUSH 1004030001 004011000 PUSH 1004030010 004011000 PUSH 1004030010 004011100 PUSH 1004030010 00401100 PUSH 1004030010 00401100 PUSH 1004030	Text string Initial DEW Selection: ASCII "Mag" not renoved" ASCII "Dity crack! Nag removed not registered!" ASCII "Elease register this software for support and you'll receive the full version!" ASCII "Elease register this software for support and you'll receive the full version!" ASCII "Dity crack! Nag removed not registered!" ASCII "Dity crack! Good Job!" ASCII "Dity crack! Good Job!" ASCII "Thank you!" RSCII "Thank you for registering this software!" RSCII "Thank you for registering this software!" RSCII "Bout" ASCII SE, "emove the NAG by TDC\r\n\n.:: Coded by\t: TDC\t\t\t:\t\r\n.:: Also known as\t: The Dutch Cr" RSCII "Whints" RSCII "Using" RSCII "Using" RSCII "Using" RSCII "Using"
×□	

运气还不错。你可以在 4010AE 处看到 nag 窗口中的文本。双击它,咱们 就跳到了 nag 窗口被创建的地方:

$\begin{array}{cccccccccccccccccccccccccccccccccccc$	CHP EVTE PTR 05:14032801.3 UE SHORT Nail 00401066 CHP EVTE PTR 05:14032801.2 UE SHORT Nail 00401067 CHP EVTE PTR 05:14032801.2 UE SHORT Nail 0040106D PUSH 1017 Nail 0040106 CHL <uhp.suser32.setulgitemtexta> OFL <uhp.suser32.setulgitemtexta> PUSH 40 PUSH 101.004030861 PUSH 1017.Nail 0040106E PUSH 1017.Nail 00401105 CHEL <uhp.suser32.setulgitemtexta> UHP N00TT Nail 00401149 CHP LARG.3,6F DHZ SHORT Nail 0040114F CHP EVTE PTR DS:14032801.3</uhp.suser32.setulgitemtexta></uhp.suser32.setulgitemtexta></uhp.suser32.setulgitemtexta>	Text = "Dirty orack! Nag removed not registered!" DentrolID = 73 (115.) hWnd = 7EFDE000 SetDigitenTextA Style = MB_OKIMB_ICONASTERISK:MB_APPLMODAL Title = "INAG] Please register this software!" Text = "ClubleSHIT] Please register this software for support and ye hOwner = 7EFDE000 HessageBoxA Text = "Clean orack! Good Job!" BontrolID = 73 (115.) hWnd = 7EFDE000 SetDigitenTextA
	5030 103324600 03 74 12 0324600 02 74 19 0324600 02 74 19 0324600 02 74 19 0324600 01 74 19 0324600 01 74 19 68 1304000 68 1304000 16 16 68 1304000 16 16 68 1304000 16 16 68 1304000 16 16 68 1304000 16 17 68 1304000 17 17 68 1304000 17 16 68 1304000 17 18 75 03 18 17 74 19 10000 18 817D 0C 11610000 18 817D 0C 116 16 75 590000000 337D	<pre>SB30 B8324000 03 CMP EVTE PTR 05:14032801.3</pre>

嗯,它上面有一个有趣的字符串,不过咱们现在先不管。咱们看看 4010A7 处,也就是调用 MessageBoxA 函数的第一行,看看哪里调用了它:

0040107B 00401082 00401088 00401088 00401088 00401089 00401094 00401096 00401096 00401090 00401090	. 803D 80324000 03 ~ 74 12 803D 80324000 02 ~ 74 18 . 803D 80324000 01 . 74 27 > 68 21304000 64 73 . FF75 08 . F75 08 . F75 08 . F75 08	CHP BYTE PTR DS:[4032B0],3 UE SHORT Nag1.00401096 CHP BYTE PTR 05:[4032B0],2 UE SHORT Nag1.00401087 CHP BYTE PTR DS:[4032B0],1 WE SHORT Nag1.0040108D PUSH Nag1.00403021 PUSH Nag1.00403021 PUSH 13 PUSH 13 PUSH 14 PUSH 13	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) Wind = 7EFDE000 SeeFDItexText0
00401030 004010A0	E8 68010000	CALL CALL CALL	-SetDigitemTextA
00401007	> +68 40	PUSH 10	range = MB OKIMB ICONASTERISKIMB APPLMODAL
004010A9	. 68 61304000	PUSH Nag1.00403061	Title = "[NAG] Please register this software!"
004010AE	. 68 86304000	PUSH Nag1.00403086	Text = "[BULLSHIT] Please register this software for support and yo
004010B3	• FF75 08	PUSH LARG.11	howner = 7EFDE000

我们能看到 40108B 处的 JE 指令调用了它,而且刚好在一个比较指令的后面。好吧,这个场景我们已经很属性了☺。咱们在 JE 指令那设置一个 BP:

00401071 00401076 00401076 00401078 00401082 00401084	E8 94010000 E8 34010000 803D 80324000 03 74 12 803D 80324000 02	CALL AJP.&user32.SetDlgItemTextA> CALL Nag1.004011AF CMP BYTE PTR DS:(4032801,3 JE SHORT Nag1.00401096 CMP BYTE PTR DS:(4032801,2	LSetDigItenTextA
08481880 09401894 03491895 03401895 03401895 03401890 03401900 03401900 03401905 03401905 03401905 03401905 03401905 03401905 03401905 03401805 0340000000000000000000000000000000000	 ✓ 74 1H ✓ 8630 B0324000 01 ✓ 74 27 ✓ 68 21304000 ← 6A 73 ← FF75 08 ← E8 68010000 ← E8 68010000 ← E8 681304000 ← 68 61304000 ← 68 613040000 ← 7575 08 ← E8 49010000 ← E8 49010000 ← 7575 08 ← E8 49010000 ← 7575 08 ← E8 44010000 ← 7575 08 	dE SHURI Nagi 004010H/ CMP BYTE PTR DS:14032800,1 dE SHORT Nagi 0040106D PUSH Nagi.0040106E PUSH Nagi.00403036 PUSH Nagi.0040304 PUSH Nagi.0040304 PUSH Nagi.0040304A PUSH PUSH 73 <	Text = "Dirty crack! Nag removed not registered!" ControlID = 73 (115.) hUnd = ZEFDE0808 SetDigItenTextA Style = HB_OK:HB_ICONASTERISK:HB_APPLMODAL Title = "CNAGJ Please register this software!" Text = "CBULLSHIT] Please register this software for support and y hOwner = ZEFDE0800 MessageBoxA Text = "Clean crack! Good Job!" DontrolID = 73 (115.) hUnd = ZEFDE080 SetDigItemTextA

运行程序。然后我们断在了那个 BP, 能够看到我们将跳转到 nag 窗口那, 所以不能让它跳:

CPAZSE	010000	ES 002 CS 002 SS 002 DS 002 FS 005 GS 002
D	0	LoctEr

接着运行程序:

Gin	TAGAN TECH
Remain	
Rer	move the NAG by TDC
Status: Dirty crac	ck! Nag removed not registered!
Status: Dirty crac You have to remo say it is regis	ck! Nag removed not registered! ove the nag and make the program stered when clicking Re-Check.
Status: Dirty crac You have to remo say it is regis	ck! Nag removed not registered! ove the nag and make the program stered when clicking Re-Check.
Status: Dirty cran You have to remo say it is regis	ck! Nag removed not registered! ove the nag and make the program stered when clicking Re-Check.

这就是"Dirty crack!"的出处,显然咱们的补丁打的还不够。重启应用, Olly 断在了 BP 那。再次将 0 标志位清零:

CPAZSED	0100000	ES 002 CS 002 SS 002 DS 002 FS 005 GS 002
D	0	LastEr

咱们单步执行两次到下一个跳转那。你可能已经猜出来了,这个跳转应该 是跳到好消息那的,而不是到坏消息那:

00401082 00401081 00401081 00401081 00401081 00401098 00401098 00401098 00401098 00401098 00401086	 √74 12 833D B0324060 02 √74 18 803D B0324060 01 68 21384000 69 738 FF75 08 F86810000 68 4384000 68 4384000 68 4384000 68 4384000 68 4384000 68 449010000 68 44304000 69 73 FF75 08 F8 4010000 69 73 FF75 08 59 0600000 50 0600000 	UE SHORT Nagl.00401096 CMP BYTE PTR 05:1403280J.2 UE SHORT Nagl.00401007 CMP BYTE PTR 05:1403280J.1 UE SHORT Nagl.0040106D PUSH Nagl.00403021 PUSH 1040 CHLL (JMP.&user32.SetDlgItenTextA) JMP SHORT Nagl.004010CE PUSH Nagl.00403066 PUSH Nagl.00403066 PUSH Nagl.00403066 PUSH Nagl.00403066 PUSH Nagl.00403066 PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE PUSH Nagl.004010CE	Text = "Dirty prack! Nag removed not registered?" ControlID = 73 (115.) hWnd = 002B0668 ('Brought to you by TDC, 2005',class='#32770') SetDigitenTextA Style = NB_OKINB_ICONASTERISKINB_APPLMODAL Title = "INAGI Please register this Software!" Text = "DBLENTIT Please register this Software for support and you hOwner = 002B0668 ('Brought to you by TDC, 2005',class='#32770') MessageBoxFi Text = "Clean prack! Good Job!" ControlID = 78 (115.) hWnd = 002B0668 ('Brought to you by TDC, 2005',class='#32770') SetDigitenTextA
004010CC	. ≥ 949010000	UHEL CONFIGNESSION SCHOOL CONFIGNESSION	- DefrigitemiextH
004010CC	.~ EB 00	JMP SHORT Nagi G040100C	
004010CE	> E9 06000000	JMP Nagi G0401189	
004010D3	> 817D 0C 11010000	CMP LARG 21,111	
004010DA	.~ 0F85 B9000000	JM2 Nagi G0401199	
004010E0	. 837D 10 6F	CMP LARG 31,6F	
004010E0	. 25 59	JM2 SHORT Nagi G040114F	

咱们打个补丁,让它直接跳:


运行程序,可以看到咱们弄对了:



很明显,现在这个补丁就可以解决这个程序,咱们回到 40108B 那(咱们原 来将 0 标志位清零的地方)给它打上补丁让它永远不会跳转。保存这两个补丁 程序就会很好的运行。不过我也想向你展示(我以前提到过,如果我没有提到 的话,那我应该提到的),通常总是有别的方法来给程序打补丁。重启应用,在 我们的 BP 处断下来:



这些指令与下面这些(用高级语言表示)类似:

```
if (contents of 4032B0 == 3)
    jump "Dirty Crack"
else if ( contents of 4032B0 == 2)
    jump to "Show Nag Screen
else if (contents of 4032B0 == 1)
    jump to Good Boy Msg
else
    Display "Dirty Crack"
```

我们知道 nag 窗口默认是要显示的,4032B0 处内存总是等于 2,因为跳转 得实现才行。如果我们跳过整个 if/then 语句,直接跳到好消息怎么样?所以 如果我们将最开始的第一个跳转替换成跳到好消息的跳转,那么我们就只需要 一个补丁就行。试试看:



运行下程序:

Gin	t and	dealar-	Bur'	DC
Remoi	0 0100	216		- 0
F	lemove the	NAG by T	DC	
Status: Clean	crack! Good	Job!		
You have to re say it is re	move the n gistered wh	ag and m ien clicking	ake the Re-Ch	program leck.
You have to re say it is re	move the n gistered wh	ag and m ien dickini	ake the Re-Ch	program leck.

可以看到结果是一样的。另外,可以思想下更加优雅的方法,"4032B0中的内容总是等于2,不过要显示好消息的话它就需要等于1,那么为什么不在内存中就放一个1呢,那样的话就会一直显示好消息了呀?"你应该试试这个。 重启应用,点一下数据窗口,转到4032B0,用二进制编辑将它改成1。起作用了没? 需要记住的另一件事是,总是有别的方法可以找到我们正在寻找的代码块。 比如,如果本例中我们不能用字符串,我们就可以用 搜索模块间调用 (译者 注:相关内容在第九章):

R Found intermodular calls	- • •
Address Disassembly	Destination
00401000 PUSH 0 00401020 CALL 00401176 CALL 00401184 CALL 00401026 CALL 00401026 CALL 00401002 CALL 00401002 CALL 00401003 CALL 00401043 CALL 00401050 CALL 00401050 CALL 00401053 CALL 00401053 CALL 00401053 CALL 00401053 CALL 00401053 CALL 00401060 CALL 00401060 CALL 00401107 CALL 00401110 CALL 00401121 CALL 00401132 CALL 00401132 CALL <td>(Initial CPU selection) user82. DialogBoxParamA user82. EndDialog user82. EndDialog kernel32. ExitProcess user82. GetDlgItem kernel32. GetModuleHandleA user82. LoadIconA kernel32. IstrompA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA</td>	(Initial CPU selection) user82. DialogBoxParamA user82. EndDialog user82. EndDialog kernel32. ExitProcess user82. GetDlgItem kernel32. GetModuleHandleA user82. LoadIconA kernel32. IstrompA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. MessageBoxA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA user82. SetDlgItemTextA
1	

注意有四个对 MessageBoxA 的调用。右键其中一个,选择"Place a breakpoint on every call to MessageBoxA"。当你运行程序时,在显示任何 东西之前,我们会停在下面这行代码:

	74 27 74 27	JE SADET Nail 06401080 PUSH Nail 06403021 PUSH LERG.11 PUSH LERG.11 SHORT Nail 06403062 PUSH Nail 06403066 PUSH Nail 06403066 PUSH Nail 06403066 PUSH Nail 06403066 PUSH LERG.11 CHLL SHORT Nail 064010CE PUSH Nail 064010CE USH LERG.11 CHLL SHORT Nail 064010CE USH Nail 064011CE SHORT Nail 064010CE PUSH Nail 064011CE	Text = "Dirty orack! Nag removed not registered!" ControlID = 73 (115.) hUnd = 0030063E ('Brought to you by TDC, 2005',class='#32770') SetDigitenTextA Style = ME_OKIME ICONASTERISKIME_APPLMODAL Title = "ILNAGI Please register this software!" Text = "IBULSHIT] Please register this software for support and you howner = 0030063E ('Brought to you by TDC, 2005',class='#32770') Text = "Clean orack! Good Job!" ControlID = 73 (115.) hUnd = 0030082E ('Brought to you by TDC, 2005',class='#32770') SetDigitenTextA
--	---	--	--

是不是很熟悉?它就是 nag 消息框!! 所以要记住总是有不只一种方法可以 完成一些事情。不久我们将会学习其他的技术(像窗口消息处理),会给你更大 包的技巧。

三、第二个应用程序

现在咱们来看看 Nag2. exe。看起来差不多,不过我们将用不同的方法来解决它。启动程序的时候,我们看到了意料中的 nag:



在点了 OK 后,我们看到了主窗口:



此时我关了程序,将其载入到 Olly 中:

4010F3	\$ 6A 60	PUSH 68	
4010F5	. 68 18524000	PUSH Nag2.00405218	
4010FH	. E8 81000000	CHLL Nag2.00401E80	
401104	. BF 94000000	MOULEON FOI	
401106	ES DERERARA	CALL Nag2, 00401FE0	
40110B	. 8965 E8	MOV DWORD PTR SS: [EBP-18], ESP	
40110E	. 8BF4	MOV ESI, ESP	
401110	. 893E	MOV DWORD PTR DS:[ESI],EDI	a second and a second a second
401112	. 56	PUSH ESI	pversionInformation = NULL
101113	. FF15 10504000	MOLL ECY DWORD PTR DS: 1(&KERNEL32, GetVersionEXH.	GetVersionEXH
aditic	8900 B8724000	MOU DWORD PTR DS+ [4072B81 FCX	
401122	8846 Ø4	MOU EAX, DWORD PTR DS: [ESI+4]	and the second s
401125	. A3 C4724000	MOV DWORD PTR DS: [4072C4]. EAX	kernel32.BaseThreadInitThunk
40112A	. 8B56 08	MOV EDX, DWORD PTR DS: [ESI+8]	
40112D	. 8915 C8724000	MOV DWORD PTR DS: [4072C8], EDX	Nag2. (ModuleEntryPoint)
401133	. 8876 ØC	MOV ESI, DWORD PTR DS: [ESI+C]	A Real Property of the second s
101136	. 81E6 FF7F0000	HNU ESI, /FFF	
401130	. 8935 BL724000	CMP ECV 2	
401142	· 00F 2 02	IF SHOPT Non2 00401159	
1277.13	DIOF COCCOCC		

首先,咱们来看看有没有字符串。这里我想提的一件事是 IDAFicator 插件。 在众多的添加功能中,它在程序顶部提供了一组按钮,让搜索字符串变得更加 的简单。当点击字符串按钮 (Str)时,它会显示 ASCII 和 Unicode 两种字符串, 并自动的将光标移到顶部,这样你就不用自己滚动到顶部了。下面是那些按钮 的样子:

le Tr Sr 👩 🚺 ?	↔ ‡ Str HBP ⊕ 📃

第一个按钮(有左右箭头的那个)是向前和往后。比如,你点击一个 CALL, 然后按一下 enter 键转到该 CALL,点一下第一个图标你会回到 CALL 指令那。 右击你就会前进一步。第二个按钮会尝试找出当前函数的开头,右击则会尝试 找到结尾。下一个就是字符串按钮。再下一个是硬件断点按钮。它会弹出一个 很漂亮的对话框来显示你设置的所有硬件断点。非常的便利。十字图标打开你 的应用程序所在的文件夹,列表图标会弹出一个对话框以便于你输入多行汇编 代码,如果你打算修改 exe 文件的大部分代码时可以用这个。你会注意到一个 叫 "Breakpoints" 的新的菜单项,它会弹出一个下拉菜单,里面是许多用到的 API,你可以自动对它们设置断点:



最后,有一个上下文菜单可以让你恢复隐藏的菜单,我们会在后面的章节 中讨论。

咱们继续,点击新的按钮栏中的字符串("Str")按钮:



在第七行,我们看到了 nag 上面的文本,双击该行:

0040106B 0040106C 00401071		5E B8 01000000 C2 1000	POP ESI MOU EAX, 1 RETN 10	kernel32.7697339A
99491974	1	69 40	PUSH 40	Stule = MB OK!MB ICONASTERISK!MB APPI MODAL
0040107A		68 50514000	PUSH Nag2.00405150	Title = "Nag!!"
0040107F		68 00514000	PUSH Nag2,00405100	Text = "Oh, do u like this program? You are using the
00401084		51 FE15 C9504000	PUSH EUX	hUwner = NULL
0040108B	1	BS 01000000	MOV EAX.1	-Hessageboxh
00401090		C2 1000	RETN 10	and the surface based on the second second second
90401093	>	885424 Ø4	MOV EDX, DWORD PTR SS: [ESP+4]	_Case 10 (WM_CLOSE) of switch 00401004
00401097		5H 00	PUSH 0	$\int \frac{1}{2} \int $
0040109A	1	FF15 CC504000	CALL DWORD PTR DS: [<&USER32.EndDialog>]	EndDialog
00401080	>	B8 01000000	MOV EAX, 1	Default case of switch 00401021
004010A5		C2 1000	RETN 10	
004010H8		00	1015	

看到了 nag 的实现方法。是一个自包含的方法(在它的上面和下面各有一个 RETN 指令),所以我们知道它是在某个地方被调用的。点击 401074 那行,也就是该自包含方法的第一行,看看它是从哪被调用:



可以看到是被 401012 的 JE 指令调用。咱们在这里设置一个断点,运行程序:

00401000 50401004 00401007 00401007	. 884424 08 . 83E8 10 .~ 0F84 86000000 . 2D 00010000	MOV EAX, DWORD PTR SS:[ESP+8] SUB EAX, 10 JE Nag2.00401093 SUB EAX, 100	Switch (cases 10111)
33431312	.×r74 60	JE SHORT Nag2.00401074	
00401014	. 48	DEC EAX	kernel32.BaseThreadInitThunk
00401015	.~ 74 05	JE SHORT Nag2.00401010	
00401017	. 3300	XOR EAX.EAX	kernel32.BaseThreadInitThunk: Default case of switch (
00401019	. C2 1000	RETN 10	
00401010	> 0FB74424 0C	MOUZY EAX, WORD PTR SS: LEEP+C1	Case 111 (WM COMMEND) of switch 00401004
00401021	. 2D E9030000	SUB EAX. 3E9	Switch (cases 3E93EA)
00401026	× 74 22	JE SHORT Nag2, 00401049	FERTIES (FEETE FEITE FERT
00401020	40	DEC EOV	keypel 22 DecoThyond Teit Thurk

咱们断在了 JE 指令处。注意它没有调用我们的 nag 窗口代码。原来是我们 刚好在 windows 的处理的中间。我们会在另一章深入讨论消息处理,不过目前 我们只需要知道所有的 GUI Windows 程序都有一个消息处理程序,并且 Windows 通过它来发送各种消息。根据到达的消息,我们可以添加自己的代码 来覆盖 Windows 的普通处理流程。例如,当我们点击窗口上的"X"时,Windows 就会通过消息处理程序发送一个消息说"嘿,用户想要关闭这个窗口"。我们可 以让消息通过,这样的话Windows 就会处理它并关闭窗口,或者我们可以"捕 获"这个消息,做任何我们想做的(可能弹出一个对话框说"你还未保存,确 定退出?")。

我们的断点刚好在这个的中间,所以已经来到的第一个消息与应用程序想要覆盖的以便于显示 nag 窗口的消息不匹配:

00401	100	·~ 0F84 8600000	UE 11392.00401035	
00401	000	. 20 00010000	UE CHERT N-22 00401074	
88481	014	10	DEC EOV	
88481	014	· 74 OF	LE CUODT Name ORADIGIC	
00401	017	0000	VOD EOV EOV	Default area of cuitab 00401004
00401	010	. 0300	PETN 10	Default case of Switch 00401004
00401	017	> 0EP24494 0C	MOUZY DOY WOOD DTD CONFECTACT	Case 111 (MM COMMOND) of switch 00401004
00401	DIC L	20 F0000000	CUP FOY 259	Switch (asses 259, 250)
88481	0.26	20 29030000	E CUOPT Noo2 00401040	SWITCH (Cases SEP. SEH)
00401	020	19 22	DEC EOV	
00401	820	· 75 75	NZ CUOPT Non2 00401000	and the second
88481	027	· OD4424 04	MOLL EOV DUDDD DTD CC+FECD+43	Case 250 of suitab 00401021
00401	02D	. 604424 64	DUCH 40	ACTULA - MD OVIMD TONOCTEDICVIMD ODDI MODOL
00401	021	. 60 0CE24000	DHCH Naa2 00405200	Tisla = "Toto"
00401	802	, 00 00524000 20 00514000	DUCH N==2 00400400	Tout - " KillNes As I did this one for
00401	000	. 60 00514000		house EEEEE20
00401	000	FETE COEGAGOO	COLL DWODD DTD DC. F/MICEDO2 MaccaseDow031	Message Rev 0
00401	040	. PP15 C0504000	MOLLEOV 1	HESSAGEDOXH
88481	042	. 02 1000	DETN 10	the second se
00401	840	· 62 1000	DIEL ECT	Non2 00401000: Coco 259 of quitab 00401021

咱们继续,按F9运行程序,然后断在同一个BP,不过这一次跳转会实现,将显示 nag 窗口。咱们让Olly 别显示它:



现在,如果我们留着这个断点,通过这个消息处理程序将会发送超过 34 个 消息。你可以留着这个 BP 然后运行 34 次(这种情况下,在某一刻你会看到有 窗口出现,按钮被绘制等),你也可以删除断点然后就运行一次。这样的话,对 nag 窗口的调用就不会再次执行,所以删除断点再运行程序比较好:



然后就看到了主窗口:



注意初始的 nag 窗口已经消失了。

四、给程序打补丁

通常我们所要做的是给跳转到 nag 的 JE 指令打补丁,将其 NOP 掉。这样的话,它就是再也不会跳了,不过我想告诉你另一个种方法也可以实现。我们知道当正确的消息来到消息处理程序时(这里指的是第二个消息),我们的 nag 代码将会被调用。好吧,如果我们允许跳到 nag 那,但是将 nag 改成再跳回去 会怎么样?



这里,将会跳转到 401074 的 nag 指令,但是我们让它又立即跳回了初始 跳转的后面那行(401014)。基本上,我们的程序会跳转,然后又跳回到下一行:



将401012 处的 JE 指令 NOP 掉与添加一个跳回到 401074 的跳转真的没啥 不同,不过我想让你开始注意到总是有多个打补丁的方法,有时候 NOP 掉一个 CALL 并不是最好的方法。记住,这个二进制文件是你的,你可以添加任何你想 添加的代码,所以别害怕修改它,尤其是在学习的时候。

运行程序,可以看到 nag 窗口同样被绕过了:



现在保存补丁。选中修改的代码(如果你选多了也没事。译者注: 就是说你选中了那些没有修改的也没关系。),选择"Copy to executable" -> "Selection":

10401048 . 887424 08 MOU ESI, DU 10401041 . . 887424 08 MOU ESI, DU 10401051 . . . 878514000 PUSH 10 10401051 10401051 10401052 10401052 10401052 10401064 10401065 10401064 10401064 10401064 10401064 10401064 10401064 10401064 10401075 <th>Copy Binary Undo selection Assemble Label Comment Breakpoint Hit trace Run trace Follow New origin here Go to</th> <th>Alt+BkSp Space ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;</th> <th>HB_ICONHANDIMB_APPLMODAL id u forget this one? :P" a0 mB_ICONASTERISKIMB_APPLMODAL b u like this program? You are usin a068 -OSEJ of switch 00401004 of switch 00401021</th>	Copy Binary Undo selection Assemble Label Comment Breakpoint Hit trace Run trace Follow New origin here Go to	Alt+BkSp Space ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;	HB_ICONHANDIMB_APPLMODAL id u forget this one? :P" a0 mB_ICONASTERISKIMB_APPLMODAL b u like this program? You are usin a068 -OSEJ of switch 00401004 of switch 00401021
Hex dump 10405000 14 97 76 97 76 27 43 9 10405000 09 14 97 76 97 76 27 76 9 10405000 09 14 97 76 90 92 76 76 15 9 10405020 12 97 76 80 08 97 76 14 9 10405020 120 77 76 12 12 97 76 14 9 9 10405020 182 12 97 76 12 12 77 15 9 10405 14 9 7 14 14 9 14	Follow in Dump Search for Find references to View	*	9018F928 9018F924 9018F924 9018F928 75196 9018F928 90850 90850 90850 90850
00405040 19 £3 97 76 31 £3 97 76 98 51 9 00405050 C0 11 97 76 83 51 97 76 F9 CA 9	Copy to executable		Selection 00000
10405060 87 35 97 76 FD 49 97 76 6E 18 9 10405070 9C 17 97 76 91 D1 99 76 55 51 9 10405070 EC 17 97 76 91 D1 99 76 59 51 9	Analysis		All modifications
014456900 24 44 97 76 24 30 95 77 25 17 9 014456900 24 44 97 76 18 11 97 76 19 14 14 97 76 12 17 77 16 14 17 77 17 97 76 12 17 97 16 12 97 76 12 17 97 76 12 17 97 16 12 97 76 12 17 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 76 12 97 10 10 10 10 10 10 10 10 10 10	Help on symbolic name Detach Process	Ctrl+F1	0013F942 00000 0015F94C 00401 0018F956 0013F956 0013F 0013F958 00401 0013F958 00401 0013F956 00401

在新窗口上右键,选择"Save file":

D File C:\Use	ers\Random\De	sktop\Rand	om\Tutorials\Intro to Reverse Er	gineering\13	Nags\Nag2.exe	×
0000045C F 00000462 6 00000464 5 00000465 F 00000468 5	F15 C8504000 A 01 F15 CC504000 E	CALL DWORD PUSH 1 PUSH ESI CALL DWORD POP ESI	PTR DS: [4050C8] PTR DS: [4050CC]			Â
bbbbbble46 bbbbble471 C bbbbble471 C bbbbble471 C bbbbble471 C bbbbble471 C bbbbble472 Bbbbble472 Bbbbble472 Bbbbble472 bbbbble472 Bbbbble472 Bbbbble472 Bbbbble472 bbbbble472 Bbbbble472 Bbbbble472 Bbbbble472 bbbbble484 S Bbbbble484 S bbbbble484 S Bbbbble484 S bbbbble484 S Bbbbble484 S bbbbbble484 S Bbbbble484 S bbbbble484 S Bbbbble484 S bbbbble484 S Bbbbble484 S bbbbble484 S Bbbbble484 S bbbbble4848 B Bbbbble4848 B bbbbbble48490 C Bbbbble48497 B bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb	8 01000000 2 1000 B 9E 0 A 40 8 50514000 1 50514000 1 F15 C6504000 8 0100000 2 1000 85424 04 A 00	MOU EHX RETH 10 JMP SHO NOP PUSH 40 PUSH 40 PUSH 40 PUSH 40 CALL DW MOU EAX RETH 10 MOU EAX RETH 10 MOU EAX	Backup Copy Binary Assemble Search for) 		
00000499 5 04 PL 0504000 00 00000 00 0 11 11 11	2 DO EDX, DWORD P ISH O ISH EDX ALL DWORD PTR I JU EAX, 1 TTN 10 ITS ITS ITS	PÜSH ED TH SSELE DSEL<&US	Save file Go to offset View image in Disassembler Hex	Ctrl+G	tch 00401004	*

我将它保存为不同名字的文件,这里是 Nag2_partial. exe。等会你会明白 我为什么把它取名为 partial:

Favorites Earline Desktop Downloads Recent Places	TDC		
 Libraries Documents Music Pictures Videos 	Nag1.exe	Nag2.exe	
File name: Nag	2_partial.exe		
Save as type: Exec	utable file (*.exe)		
Hide Folders			Save

OK。咱们继续,Olly载入这个打过补丁的程序试试看。我们直接跳到了主窗口,所以我们知道那个补丁起作用了。现在点击 exit,但是弹出了这个:



噢。我才这里作者是真的很确定(译者注:确定你没有干掉这个 nag)。咱们来找找这第二个 nag。回到字符串窗口那,可以发现这个 nag 的文本也在那:

R Found strings are	
Rddress Disassenbly Bddress D	Text string (Initial CPU selection) RSCII "Info" RSCII "Info" RSCII "Info" RSCII "Main" tid u forget this one? :P" RSCII "Main" tid u forget this one? :P" RSCII "Concest" RSCII "Concest
30	* 1

许多许多许多应用程序会这么干。它们启动的时候有个 nag,关闭了之后又 来一个。大部分时候,你在搜索第一个 nag 的文本字符串时,你也应该自动找 找其他的。在该文本上双击:



我们在这里看到了该 nag 的方法。点击该方法的第一行,我们能够看到第 二个 nag 正是在第一个 nag 被调用之后调用的。但是它用了一个不同的消息来 触发它(很可能是窗口销毁消息)。所以当这个消息到达时,发出用户选择"Exit" 的信号,第二个 nag 就会被调用。

你可能首先会想到"我们为什么不能像上一个那样,放置一个跳转让它再跳回去呢。"好吧,你再仔细看看这个方法,我们能看到它调用了第二个 nag, 但是然后它立即调用了 EndDialog。所以跳回去不会起作用,因为我们的对话框 永远不会被关闭:



所以接下来你可能会想"咱们就把 401026 的 JE 指令修改成跳到 EndDialog, 跳过显示 nag 的 MessageBoxA 的指令"。这个想法不错,咱们来 试试:

00401017	· 3300	XOR EAX, EAX	kernel32.BaseThreadInitThunk; Default case of switch (
0040101C 00401021 00401026	> 0FB74424 0C . 2D E9030000 . 74 22	MOVZX EAX, WORD PTR SS: (ESP+0] SUB EAX, SE9 VE SHORT Nag2, 0040104A	make this
00401029 00401029 00401028 00401028 00401036 00401036 00401036 00401036 0040103C 00401042 00401042 00401047 00401044 00401044 00401044	. 48 . 75 75 . 884424 04 . 6A 40 . 68 80524000 . 68 80514000 . 50 . FF15 C8504000 . B8 01000000 . C2 1000 . 956 . 887424 08	UEC EAX JN2 SHORT Nag2.004010A0 MOV EAX,DWORD PTR SS:[ESP+4] PUSH 40 PUSH Nag2.0040520C PUSH Nag2.00405180 PUSH EAX CALL DWORD PTR DS:[<&USER32.MessageBoxA>] MOV EAX,1 MOV EAX,1 PUSH ESI MOV EAX,1 PUSH ESI MOV EAX,1 DUBRD PTR SS:[ESP+8]	TadInitihunk Case 3ER of switch 00401021 Style = MB_OKIMB_ICONASTERISK:MB_APPLMODAL Title = "Info" Text = "KillNag \n I did this one for your newt hOwner = 76973388 MessageBoxA Case 3E9 of switch 00401021
0040104F 00401056 00401056 00401055 00401055 00401062 00401064 00401064 00401065 00401066 00401065 00401065 00401071	64 10 68 78514000 68 58514000 56 FF15 C8504000 6A 01 56 FF15 CC504000 5E B8 01000000 C2 1000	PUSH 10 PUSH Nag2.00405178 PUSH Nag2.00405158 PUSH ESI CALL DWORD PTR DS:[<&USER32.MessageBoxA>] PUSH 1 PUSH 1 PUSH ESI CALL DWORD PTR DS:[<&USER32.EndDialog>] POP ESI POP ESI POP ESI POP ESI Nov EAX,1 RETM 10 PUSH 10 POP DS: POP ESI POP	Style = MB_OKIMB_ICONHAND:MB_APPLMODAL Title = "Nagt" Text = "Oh, did u forget this one? :P" hOwner = NULL HessageBoxA Result = 1 hUnd = NULL EndOialog kernel32.7697839A
ajum	np to here instea	d 148 Nag2.00405150	Title = MB_OKIMB_ICONASTERISKIMB_APPLMODAL

将 401026 的 JE 指令修改成跳转到 401062,也就是跳到 EndDialog 的第 一行:

00401007 0040100D 00401012 00401014 00401015 00401017 00401017	.~ 0-84 85000000 . 2D 00010000 .~ 74 60 .~ 74 05 . 33C0 	UE N992-00401095 SUB EAX, 100 JE SHORT Nag2,00401074 DEC EAX JE SHORT Nag2,0040101C XOR EAX,EAX	kerne 132. BaseThreadIr kerne 132. BaseThreadIr	iitThunk iitThunk; Default case of switch (
0040101C 00401021 00401026	> 0FB74424 0C . 2D E9030000 > 74 3R	MOUZX EAX, WORD PTR SS: LESP+C3 SUB EAX, 3E9 JE SHORT Nag2, 00401062	ssemble at 00401028	
00401029 00401029 0040102B	. 48 . 75 75 . 884424 04 . 69 40	DEC EHX WhZ SHORT Nag2.004010A0 MOV EAX, DWORD PTR SS: [ESP+4] PUSH 40	JE SHORT 00401062	2
00401031 00401036 00401038 0040103C 0040103C	68 0C524000 68 80514000 50 FF15 C8504000 88 01000000	PUSH Nag2.00405200 PUSH Nag2.00405180 PUSH EAX CALL DWORD PTR DS:[<&USER32.MessageB(HOV_ERX,1	Fill with NOP's	Assemble Cancel
0040104A 0040104B 0040104F 0040104F 00401051 0040105B 0040105B 0040105B 00401065 0040000 0040000 0040000 0040000 0040000 0040000 0040000 0040000 00400000000	. C2 1000 56 887424 08 69 10 68 78514000 56 FF15 C3504000 69 01 56 FF15 CC504000 56 FF15 CC504000 C2 1000 C2 1000 C	NETR 18 PUSH ESI MOV ESI, DWORD PTR SS: [ESP+8] PUSH Nag2.00405178 PUSH Nag2.00405158 PUSH ESI CALL DWORD PTR DS: [<&USER32.MessageBoxF PUSH 1 PUSH 51 PUSH 51 POP ESI MOV ERX,1 RETN 10 JMP SHORT Nag2.00401014 NOP	Case 3E9 of switch 00 Style = MB_OKIMB_ICON Title = "Nagf" Text = "Oh, did u for hOwner = NULL MessageBoxA Result = 1 hUnd = NULL Endbialog kernel32.7697339A	H01021 HAND!MB_APPLMODAL ∵get this one? ∶P″

运行程序:

Nag2_partial.exe	23
Nag2_partial.exe has stopp	ed working
Windows is checking for a solution	to the problem
(
	Cancel

看起来前途灰暗呐。我们明显做错了什么。下面是我们准备做的:运行没 有补丁的程序,单步执行它,看看是什么情况。然后再运行带补丁的程序,再 看看两者之间有什么不同。重启应用并点击"Exit",我们会断在打补丁的地方 (因为我们重启了应用,所以补丁消失了):



单步运行几行,当你单步步过对 MessageBoxA 的调用时,你就会看到 nag 窗口:



再单步两次, 直到对 EndDialog 调用的那个 CALL:



咱们来看看堆栈。在堆栈中我们看到四个项目:我们窗口的句柄、对话框的返回值、指向第一行代码(401000)的指针、返回到 user32 的地址。



重启应用,当我们来到打补丁的地方后将其激活(打开补丁窗口,选中后按一下空格键):



现在我们将跳过 nag 消息框。单步执行直到来到对 EndDialog 的调用处:

0040105C 00401062 00401064		FF15 C8504000 6A 01 56	CALL DWORD PTR DS:[<&USER32.MessageBoxA>] PUSH 1 PUSH 1 PUSH ESI	HessageBoxA Result = 1 hund = 00401000
00401065	1.1	FF15 CC504000	CALL DWORD PTR DS: [<&USER32.EndDialog>]	EndDialog
0040106B 0040106C	:	5E B8 01000000	POP ESI MOV EAX,1	Nag2_par.00401000
00401071	in	C2 1000	REIN 10 IMP SUGPT Noo2 now 00401014	Core 119 of cuitob 00401004

看看堆栈,有窗口的句柄、返回值、返回到 user32,但是没有了指向代码的第一行也就是 401000 的指针!!!

2010100	BB461888	NUNC = 00401000
0018F92	00000001	Result = 1
0019	751962FA	RETURN to User32,751962FA
0018592	000E11HH	
0018F-24	00000359	
0018F938	000B1148	
0018F93C	00401000	Nag2_par.00401000
0018F940	DCBRABCD	
0010F944	000000001	
0018F94C	00401000	Nag2_par.00401000
0018F950	-0018F9CC	
0018F954	751BF943	RETURN to user32.7518F943 from user82.751962D7
00185958	00401000 000E1100	hagz_par.00401000
0018F960	00000111	
0018F964	000003E9	
0018F968	000B1148	
00185950	56F6H79B	
0018F974	000000111	
0018F978	00A377C0	

如果你向上滚动看看第二个 nag 的 CALL,你会发现在消息框被创建之前, ESI 被压栈了。这是一个指向我们代码的指针。在调用消息框前,它刚好被压入 堆栈,虽然它也可以在这之后被压栈。所以我们丢失了一个重要的 push 操作, 为了正确的运行 EndDialog 程序需要它。问题是,我们有一些想要的初始化代 码,然后是一个我们不想要的对 nag 窗口的 CALL,之后又是一个我们想要的对 EndDialog 调用的 CALL:

0040101C 00401021	> 0FB74424 0	MOUZX EAX, WORD PTR SS: [ESP+C] SUB EAX, 359	Switch (cases 3E93EA)
00401028 00401029 00401028 00401028	48 • 75 75 • 884424 04	DEC EAX JNZ SHORT Nag2_par.004010A0 MOV EAX, DWORD PTR SS:[ESP+4] PUSU 40	Case 3EA of switch 00401021
00401031 00401036 00401038	68 0C52400	PUSH Nag2_par.0040520C PUSH Nag2_par.00405180 PUSH EAX	Title = "Info" Text = " KillNag \n I did this one for your new hOwner = NULL
0040103C 00401042 00401047 00401047	. BS 0100000 . C2 1000 > 56	MOU EAX, 1 RETN 10 PUSH ESI	Nag2_par_00401000; Case 3E9 of switch 00401021
00401048 0040104F 00401051 00401056	. 887424 08 . 6A 10 . 68 7851400 . 68 5851400	NUV ESI,DUURD PIR SS:[ESP#8] PUSH 10 PUSH Nag2_par,00405178 PUSH Nag2_par.00405158	user32.751962FH Style = MB_OK!MB_ICONHAND!MB_APPLMODAL Title = "Nag!" Text = "Oh, did w forget this one? :P"
0040105B 0040105C 00401062 00401062	56 FF15 C8504 6A 01	PUSH ESI CALL DWORD PTR DS: [<&USER32.MessageBoxA>: PUSH 1 PUSH 51	hOwner = 06401000 MessageBoxA Result = 1 bund = 00401000
00401065 0040106B 0040106C	FF15 CC504	00 CALL DWORD PTR DS:[<&USER32.EndDialog>] POP ESI MOU EAX,1	LEndDialog Nag2_par.00401000
00401071	> EB 9E	JMP SHORT Nag2_par.00401014	Case 110 of switch 00401004

好吧,那咱们就去除掉不想要的代码。选中 MessageBoxA 指令(从 40104F 到 40105C) 然后右键,选择 "Binary" -> "fill with NOPs":

1040102B 1040102F 10401031 10401036 10401038 10401038 10401042 10401042 10401044 10401048	 884424 04 6A 40 68 80524000 68 80514000 50 FF15 C85040 B8 01000000 C2 1000 S6 887424 08 	HOU ERX, DUORD PTR SS:(ESP+41) PUSH 40 PUSH Nag2_par.00405200 PUSH 8ag2_par.00405200 PUSH ERX 00405130 PUSH ERX 100 CPTR DS:(24USER32.Message HOU ERX,10 PUSH ESI HOU ERX.50000 PTR SS:(ESP+81)	Case BER of suitch 004 Title = ME.OKIMB_ICONA Title = "Info" Title = "Info" Howner = NULL HessageBoxA Nag2_par.00401000; Cas	M01021 ISTERISK:MB_APPLMODAL ∖n I did this one for e 3E9 of switch 004010	your newt 21	EDI 000000 EIP 004010 C 0 ES 00 P 1 CS 00 A 0 SS 00 A 0 SS 00 Z 1 DS 00 S 0 FS 00 T 0 GS 00	00 26 Nag2_par. 28 32bit 0(F 23 32bit 0(F 28 32bit 0(F 28 32bit 0(F 53 32bit 7EF 28 32bit 0(F
1040104F 10401051 10401056 10401058	. 6A 10 . 68 78514000 . 68 58514000 . 56	PUSH 10 PUSH Nag2_par.00405178 PUSH Nag2_par.00405158 PUSH ESI	Backup Copy			0 0 LastE EFL 000002 ST0 empty	rr <mark>ERROR_SUC</mark> 46 (NO,NB,E, 0.0
1040105C 10401062 10401064 10401064	. FF15 C85040 . 6A 01 . 56 . FF15 CC5040	PUSH 1 PUSH ESI CRU DWORD PTR DS:[{&USER32.EpdDial	Binary	*	Edit	CI	trl+E
1040106B	5E 88 01000000	POP ESI MOU EAX.1	Assemble	Space	Fill with 0	0's	00000
10401071 10401074	. C2 1000 >> EB 9E	RETN 10 JMP SHORT Nag2_par.00401014	Label	4	Fill with N	NOPs 🔸	00000
10401076 10401077 10401078	90 90 . 6A 40 68 50	NOP NOP 51 44 ASCII "jehPoe",0	Comment	4	Binary co	ру	4R,53
1040107F 10401084	. 68 00514000 . 51	PUSH Nag2_par.00405100 PUSH ECX	Breakpoint			1	
10401085 1040108B	. FF15 C85040	MU UHLL UWUKU MIK DS: [<&USER32.Message MOV EAX, 1	Hit trace				
0401093 0401093 0401097	> 885424 04 - 68 00	MOV EDX, DWORD PTR SS: [ESP+4] PUSH 0 PUSH FOX	Run trace	OK.			

然后,砰!再也没有对 nag 的 CALL 了:



现在当你在运行程序的时候,会发现程序可以正常关闭了。你可以保存补 丁,并且没有 nag 窗口了。。

第十五章:调用栈的使用

一、简介

本章我们会删除一个"真正"的程序的 nag 窗口。为了试图帮助作者,因为他们花了大量的时间来创造这些应用,我试着挑出一个能将伤害降到最低的应用。这次,我用 Google 搜索了下"Cracked Software",这个程序有着最高的点击率,包括教程、序列号、keygen,应有尽有。因为获得该软件的破解版是如此的简单,我想不管怎么获得它都不太可能有麻烦。不过我们请求你,如果你确实喜欢它,那就买它。

我也会添加一些技巧到咱们的逆向兵器库中。有一点需要注意,如果你是 在 64 位 Windows7 下学习本章 (像我一样),Olly1.1 版甚至我的版本,调用栈 这招就不好用了。我的建议是,做我所做的:用 Olly2.0 版来学习新的技巧(取 得正确的地址),然后再转到我所用的 Olly 来做其他的。或者就用 Olly2.0 版, 它有很多很好的特性,它可以在 64 位操作系统下工作。(译者注:既然有了 Olly2.0,那我们为什么在平时用的时候还是以 1.1 的版本居多呢?因为 1.1 版 的 Olly 拥有大量的插件,这给破解带来了很大的便利,而 2.0 版的插件则弱了 很多。)

你可以在教程页下载本章的相关文件。

二、探究该应用

该程序有一个40天(可能和圣经有关?)的使用限制,40天后会弹出一个 nag窗口。相信我,根据以往的经验,它肯定很啰嗦。不幸的是,因为 nag窗 口40天(40个晚上?-对不起。)内都不会出现,所以你有两个选择:你可以安 装该应用,然后等上40天再阅读本章;或者你可以将系统时间设置成今天加上 41天后练习本章,然后再将日期设置回今天。你要确保在练习本章前做了其中 的一个,否则它不匹配。

	evaluation cop	yy)	
ile Commands To	ools Favorites	s Options Help	
Add Extract To	Test V	/iew Delete Fit	nd Wizard Info Repair
Name	am Files\WinR/ Size	AR	Modified
R.		File folder	
Formats		File folder	7/13/2012 8:44
Default.SFX	102.912	SFX File	1/17/2007 3:36
Descript.ion	1.063	ION File	9/18/2006 9:13
File Id.diz	509	DIZ File	1/17/2007 3:36
License.txt	4,494	Text Document	10/18/2005 6:1
Order.htm	3,271	Firefox HTML Doc	10/22/2006 11:
Rar.exe	316,928	Application	1/17/2007 3:36
Rar.txt	71,951	Text Document	12/30/2006 6:1
RarExt.dll	129,024	Application extens	1/17/2007 5:36
RarExt64.dll	43,008	Application extens	12/11/2006 2:1
RarExtLoader.exe	44,032	Application	12/23/2006 5:3
RarFiles.lst	1,088	MASM Listing	4/11/2006 12:0
rarnew.dat	20	DAT File	7/13/2012 8:45
ReadMe.txt	1,687	Text Document	5/12/2005 6:01
TechNote.txt	9,232	Text Document	9/19/2006 7:16
Uninstall.exe	99,840	Application	1/17/2007 3:37
📄 Uninstall.lst	639	MASM Listing	1/17/2007 5:36
UnRAR.exe	203,264	Application	1/17/2007 3:36
UnrarSrc.txt	90	Text Document	5/12/2005 6:02
WhatsNew.txt	9,906	Text Document	1/17/2007 3:36
WinCon.SFX	80,384	SFX File	1/17/2007 3:36
WinRAR.chm	243,916	Compiled HTML	1/17/2007 3:37
WinRAR.exe	929,792	Application	1/17/2007 3:36
	67 073	CEV El-	1 (17 (2007 2.26

过了一会...。

Please note that WinRAR is not free software. After a 40 day trial period you must either buy a license or remove it from your computer.	Buy online
	How to buy
	Close
	Help

nag 窗口弹出来了。在使用的时候它出现了很多次。这真是很烦人。我们在顶部也可以看到"evaluation copy":



我打算介绍两种方法来获取相关注册码。

三、第一种方法

Olly 载入并启动程序:

00401000	34	CEB 10	JMP SHORT WinRAR.00401012	the second se
00401002 00401003 00401003 00401005 00401005 00401005 00401007 00401009 00401009 00401009 00401009	-	28 43 28 28 443 28 443 443 443 443 443 445 445 445	DB 66 DB 62 DB 28 DB 28 DB 28 DB 28 DB 43 DB 43 DB 43 DB 443 DB 4	CHAR 'f' CHAR 'b' CHAR ':' CHAR ':' CHAR '+' CHAR '+' CHAR 'H' CHAR 'O' CHAR 'O' CHAR 'O' CHAR 'K'
0040100C 0040100D 00401012 00401017 00401017 00401020 00401022 00401027 00401027 00401027 00401029 00401029 00401025		90 E9 CØ 21 4A 00 A1 B3214A00 C1E0 02 A3 B7214A00 52 6A 00 E8 DDFF0900 8BD0 E8 12600900 5A F8 14530900	NOP HSCII "84",0 MOV EAX,DWORD PTR DS:[4A21B3] SHL EAX,2 MOV DWORD PTR DS:[4A21B7],EAX PUSH 0 CALL <jmp.&kernel32.getmodulehandl MOV EDX,EAX CALL WinRAR.00497040 POP EDX CALL WinRAR.00497040 POP EDX</jmp.&kernel32.getmodulehandl 	kernel32.BaseThreadInitThunk WinRAR. <moduleentrypoint> Comodule = NULL Kernel32.BaseThreadInitThunk kernel32.763BED6C</moduleentrypoint>
00401034 00401039 00401039 00401040 00401040 00401044 00401044 00401048 00401048 004010452 00401052		E8 08600900 6A 00 E8 98720900 59 68 5C214A00 6A 00 E8 87FF0900 A3 88214A00 6A 00 F9 03560900	CALL WinRAR.00497044 PUSH 0 CALL WinRAR.004982D8 POP ECX PUSH WinRAR.004A215C PUSH 0 CALL <jmp.&kernel32.getmodulehand1 MOV DWORD PTR DS:[4A218B],EAX PUSH 0 JMP.WinRAR.0049E0EC</jmp.&kernel32.getmodulehand1 	<pre>[Arg1 = 00000000 WinRAR.004982D8 kernel32.763BED6C [phodule = NULL GetModuleHandleA kernel32.BaseThreadInitThunk</pre>

启动应用后, 等 nag 出现。它一出现(在关闭它以前), 切换到 Olly, 点击暂停按钮(靠近 play 的那个):

*				
File View	Debug	Plugins	Options	Window
Running				
C *G.P.	U* - main t	hread, mo	dule WinR	AR
0040100 0040100 0040100 0040100 0040100		10	UB 60 DB 60 DB 60 DB 30 DB 40	SHORT Win

现在,我们要找出那个 nag 窗口是从哪来的,最后找出是谁让它显示的。 我们当然也可以搜索字符串或模块间调用,不过我向你保证,这些技巧对于外 面的大部分应用都不管用。所以咱们学习另外一个技巧...。

四、调用栈

调用栈是 Olly 尝试跟踪让我们到达某处位置的代码,从而试着找出哪个函数被调用。它也尝试向你显示被传递给函数的参数。所有的这些都可以通过右下角的"普通"的堆栈来完成,不过通过调用栈用来查看这些数据要更好用。 要记住 Olly 在这方面不是很完美,你不能把这个窗口的所有东西都当做福音(糟糕,我又犯了同样的错误。译者注:我也不知道这句话啥意思。)。要做很多猜测。当然,有很多次,这个窗口是空的。通常是因为 Olly 完全糊涂了,或者是在逆向一个 VB 程序(VB 程序在调用函数的方式上和真正的程序不太一样)。

要查看调用栈,如果你用的是我的版本的话,点击"St"按钮:



如果用的是原版的 Olly,点击工具栏中的"K"按钮。似乎"Call"这个词 在作者的母语中是以"K"打头的:

K Call st	K Call stack of main thread					
Address	Stack	Procedure / arguments				
0012C118 0012C11C 0012C150 0012C174 0012C174 0012C1C4 0012C1C4 0012C1C6 0012C1C0 0012C1C0 0012C104 0012E5A4 0012E5A4 0012E5A8 0012E6A8 0012E6A8 0012E6A8 0012E6BC 0012E6EC	761B66C9 761D3824 761D3827 761D3876 761ECF3E 0043781D 00400000 00446974 0007047C 00482E30 00000000 761BC4E7 761BC5E7 761BC5E7 761BC5E7 761BC5E7 761BC5E7 0012E6C0 00442E34 0012E6C0	Includes ntdll.KiFastSystemCallRet user32.WaitMessage user32.761D3875 user32.761D3875 user32.01alogBoxParamA> hInst = 00400000 pTemplate = "REMINDER" hOwner = 0007047C ('WinRAR - WinRAR (evaluation c',class=' DIgProc = WinRAR.0043E30 IParam = NULL Includes WinRAR.0043F81D ? wser32.761BC424 ? user32.761BC434 ? user32.761BC534 ? user32.761BC534 ? user32.761BC534 ? user32.761BC534 ? user32.761BC534 ? user32.761BC534 ? user32.761BC534 ? user32.761BC424 pMsg = WM_TIMER hw = 7047C ("WinRAR - WinRAR (evaluation":				
	-					

有几件事需要点出来...。最近的调用是在顶部,和堆栈类似。"Includes" 意思是该指令在那个 CALL 中有涉及到,不过 Olly 不能够准确的知道是怎么回 事。问号的意思是 Olly 对该行没有把握,所以你得为自己带盐(译者注:原话 是"take it with a grain of salt",意思是需要进行分析,要斟酌斟酌,不能全 信。)。

在咱们的例子中,可以看到一个 ntdll 函数、几个 user32 函数、对 DialogBoxParamA 的带参数调用、又是几个对 user32 的调用,底部是对程序 WinRAR 本身函数的调用。下面是对这些内容的思考: WinRAR 在地址 442C44

处调用了 DispatchMessageA,这里用一个消息来显示对话框。然后 User32 调用了 DialogBoxParamA 函数来显示对话框,标题是 "evaluation copy",还有其他几个参数。然后 User32 显示对话框并等待我们的输入,它使用 WaitMessage 来做这个。



这个窗口里重要的是调用显示对话框的那个 CALL 以及应用自己的 CALL。 通常使用调用栈时,从顶部开始,找到你感兴趣的可以用来找到代码区块的第 一项。如果这个不好用,继续向下找,检查每一个函数调用,直到你"回到" 代码足够的远,以找出决定该函数是否被调用的那个 比较/跳转 指令组合。通 过双击那行,咱们来试试检查下对 DialogBoxParamA 调用的那个 CALL:



我们来到了调用 DialogBoxParamA 的地方。我在执行设置以及调用显示对 话框这些指令的开始处设置了一个 BP。在它的上面,有几个条件跳转映入眼帘。 如果你再向上滚动,你会发现有几个写着 Case XX (WM-Something) of switch 0043F0A4"的注释,这里的 XX 是一个十六进制数:



这是 Olly 显示 switch 语句的方式。如果你往上滚动,你会发现这真是一个 相 当 大 的 switch 语 句 。 如 果 你 有 Windows 编 程 经 验 , 你 可 以 认 出 "WM-SOMETHING"句子是 Windows 消息,你也可以认出这一整块代码是作 为 Windows 消息的消息处理过程。如果你对这些一无所知也没关系,在后面的 章节中我们会非常细致的讲解 windows 消息处理过程。目前来说,我们只对涉 及到对话框调用的部分感兴趣。下面,我们可以看看这整个分支 (case):



你会发现它是在处理 WM-TIMER 消息的区块。这很能说明一些问题。为什么我们的对话框是在一个对计时器超时的消息处理中?我们马上就会看到...。

还要注意在对话框被调用后,有几个条件跳转和比较语句:



这些跳转执行的代码依赖于我们点了对话框中的什么。如果你点的是" Close",它会跳到关闭窗口的代码等。

滚动到 switch/case 语句块的起始处,能够发现那里有一个初始的比较和跳转指令:



这个跳转跳过了打开 nag 窗口的那个 CALL(还有其他很多代码也被跳过)。 咱们来看看这个初始的 比较/跳转 是啥。在该行上右键,也就是有 "Case 113 (WM_TIMER)"的那行,选择 "Goto":

	D 1	CS 0023 32010 01	FFFFFFFF)	-	
Case II3 (WEIINER) of Switch 0043r0H4	Backup	•	FFFFFFF) FFFFFFFF)	77 57 19 4	
	Сору				
	Binary	+	CCESS (00000000)	1000	
	Assemble	Space		2012	
	Label	+			
ASCII "rarkey"	Comment	3			
	Breakpoint	÷.,	0 ESPU02		
	Hit trace		0 Err 0 0 1 0 0 0 4 Mask 1 1 0 0		
	Run trace	÷.		- NO	
Param = NULL	New origin here	Ctrl+Gray *			
DigProc = WinRAR.00482E30 hOwner = 002F04C0 ('WinRAR - WinRAR (e	Go to		Origin	*	
hInst = 00400000 DialogBoxParamA	Thread		Previous	Minus	
-DratogooxParane	Follow in Dump	•	Expression	Ctrl+G	
CIParam = 0	Search for		Previous procedure	Ctrl+Minus	
WParam = 0 Message = WM_CLOSE	Find references to		Next procedure	Ctrl+Plus	
hWnd = 2F04C0 PostMessageA	View	+	Switch base		
	Copy to executable	•	Default case		
	Analysis	*	Case 1 (WM_CREATE)		
Timeout = 10. ms h0bject = 00000230 (window) WaitErSingleDiect	Detach Process		Case 2 (WM_DESTROY)		
	Process Patcher		Case 5 (WM_SIZE)		
	Analyze This!		Case 7 (WM_SETFOCUS)		
9912	Bookmark		Case 10 (WM_CLOSE)	1.5.2	
	Code Dinner		Case 15 (WM_SYSCOLORCH/	ANGE)	
00120 00120 00120	TDAFirster		Case 2B (WM_DRAWITEM)		
00120	2 CMD		Case 2C (WM_MEASUREITEM	0	
00120	(CIVIP		Case 4E (WM_NOTIFY)		
00120	Script(S)		Case 53 (WM_HELP)		
●≪♥+\$¬. 00120 00120	Script Functions	•	More cases		
00120 00121 00121	Dump debugged process		JE from 0043F177		

在弹出的下拉菜单中你可以看到,Olly 向我们显示了可以被这个 switch 语 句处理的好几个 case。点击 "More cases...",会弹出一个对话框向我们显示 全部的 case:



点击其中几个, 然后点击 "Follow", 你会跳到处理相应 case 的代码。你 会发现, 所有这些 case 的开始都是一个 比较/跳转 组合。意思就是汇编语言 处理 switch/case 语句, 是作为巨大的 if/then 语句来处理的。有点像下面这 样 (用伪代码来表示):

if (msg != WM_CREATE)
 jump to next if
Do WM_CREATE code
Jump to end
if (msg != WM_DESTROY)
 jump to next if
Do WM_DESTROY code
Jump to end
if (msg != WM_SIZE)
 jump to next if
Do WM_SIZE code
...

所以在每一个 case 的起始处,都要检测该 case 是不是用于处理特定的消息,如果不是就跳到下一个比较。如果是,就忽略跳转,直接转到处理消息的 代码部分。 现在,因为我们的 case 是 WM_TIMER 消息,我们可以知道(通过在 Google 搜索 WM_TIMER 消息),这个是用来处理计时器超时时的消息。也就是说,在 某个地方计时器必须被启动。向上滚动(多滚一点),我们看到了罪魁祸首:



那么现在,我们可以猜测怎么来覆盖这个 nag 窗口了...。

五、给程序打补丁

最简单的方法是,当计时器超时时,让消息处理过程什么都不做。做这个的最简单的方法是保证我们每一次都跳过这个 case。那转到该 case (113-WM_TIMER)的起始处,在这里它会检测是不是正确的 case,把它改成总是跳转:

004387566 . BA 60694A00 MOU EDX.WINBAR.004A6960 004387568 . 33C9 XOR ECX.ECX 004387561 . 38C6 MOU EAX.ESI	Assemble at 0043F79E
Didd Stress Stres Stres Stres	Fill with NOP's Assemble Cancel
3350 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 7670 3350 76700 76700 76700	
下面就是打过补了之后的样子:	
0943F788 B8 604D4C00 MOV EAX, WINRAR, 004024D60 0043F78D E8 4ACD0100 CALL WinRAR, 0045C4DC 0043F792 ~ E9 3C140000 UMP WINRAR, 004408D3 0043F797 > 833D F0764C00 00 0043F792 ~ EB 7D UMP SHORT WINRAR, 004468D3	D.0 Case 113 (WM_TIMER) of switch 0043F0A4
0043577R0 8095 R4FAFFFF LEA EDX, DWDRD FTR SS: LEA 0043577R0 .<	9 -55C1

现在,无论什么时候该消息过程得到了计时器超时的消息,它都会简单的 忽略它。运行程序,过一会你会发现那个 nag 窗口再也不会出现了:

			1 🗒 🔊 (4) 🖓
Add Extract To	Tort 1		Winard Info Penair
Add Extract To	Test V	new Delete Fil	
1 C:\Progra	am Files\WinR/	AR	
lame 🔂	Size	Туре	Modified
a contraction of the second se		File folder	
Formats		File folder	7/13/2012 8:44
Default.SFX	102,912	SFX File	1/17/2007 3:36
Descript.ion	1,063	ION File	9/18/2006 9:13
File_Id.diz	509	DIZ File	1/17/2007 3:36
License.txt	4,494	Text Document	10/18/2005 6:1
Order.htm	3,271	Firefox HTML Doc	10/22/2006 11:
Rar.exe	316,928	Application	1/17/2007 3:36
Rar.txt	71,951	Text Document	12/30/2006 6:1
RarExt.dll	129,024	Application extens	1/17/2007 5:36
RarExt64.dll	43,008	Application extens	12/11/2006 2:1
RarExtLoader.exe	44,032	Application	12/23/2006 5:3
RarFiles.lst	1,088	MASM Listing	4/11/2006 12:0
rarnew.dat	20	DAT File	7/13/2012 8:45
ReadMe.txt	1,687	Text Document	5/12/2005 6:01
TechNote.txt	9,232	Text Document	9/19/2006 7:16
Uninstall.exe	99,840	Application	1/17/2007 3:37
Uninstall.lst	639	MASM Listing	1/17/2007 5:36
UnRAR.exe	203,264	Application	1/17/2007 3:36
UnrarSrc.txt	90	Text Document	5/12/2005 6:02
WhatsNew.txt	9,906	Text Document	1/17/2007 3:36
WinCon.SFX	80,384	SFX File	1/17/2007 3:36
WinRAR.chm	243,916	Compiled HTML	1/17/2007 3:37
WinRAR.exe	929,792	Application	1/17/2007 3:36
Zip.SFX	67,072	SFX File	1/17/2007 3:36
Contractor	and a second	Cauge Ca	

这个程序仍然会在标题栏显示 "evaluation",后面的章节我们会回到这个话题(修改这个有点复杂,你可以试试。这是学习的最好方法!)。不过目前,即使它显示 "evaluation",它也工作的很好并且永远不会过期。好吧,确切的说这不是真的,它仍然会过期,不过它却什么都不会做。确定你保存了补丁(和我们前面几章一样)以保存所做的修改。

六、第二种方法

现在,我向你介绍我们能够用来找出对话框区块的另外一个方法(除了调用栈)。如果你还没有的话,先下载一个 Resource Hacker。你可以在工具页获

得它。Resource Hacker 可以让你查看和操纵一个 PE 文件内的资源。当我们 讨论 PE 文件的构成时会更深入的讨论资源,不过就目前来说我们只需要知道应 用程序所使用的任何资源(包括按钮、对话框、位图、图标、文本字符串)都 存储在文件的独立区块,和代码是分开的。真正的,看看我所说的最好的方法 是打开 Resource Hacker,载入几个程序看看。

那么咱们就这么干...。打开 Resource Hacker,载入我们的应用:



左边的树形列表显示了应用程序的各种资源。可以看到它包含有位图、图标、菜单,以及最重要的对话框:



该应用有很多对话框。咱们继续,点击第一个,就是 ABOUTRARDIALOG:

Resource Hacker - CAProgram F	iles\WinRAR\WinRAR.exe		×
File Edit View Action Help			
Configurement C	<pre>BitestWinRAR(WinRAR.exe</pre>	, 175 WS_VISIBLE WS_CAPTION WS_SYSM S_BITMAP SS_NOTIFY SS_REALSIZEIM CC_SS_LEFT WS_CHILD WS_VISIBLE Dialog - ABOUTRARDLG About WinRAR Copyright © 1993-% by Alexander Roshal Translated by your name	ENU AGE SS_SUNKEN WS_CHILD WS_VI: WS_GROUP, 49, 43, 125, 8 D WS_VISIBLE WS_GROUP, 49, 54, VISIBLE WS_GROUP, 49, 62, 125, 1 ILD WS_VISIBLE WS_GROUP, 49, 7 OUP, 49, 92, 125, 16 9, 112, 125, 1 WS_TABSTOP, 8, 42, 32, 32 S_NOPREFIX WS_CHILD WS_VISIBLI ILD WS_VISIBLE WS_GROUP, 49, 1 ILD WS_VISIBLE WS_GROUP, 49, 2 9, 145, 125, 1 E WS_TABSTOP, 139, 151, 34, 14 34, 175, 137
		40 days trial copy	
	638	L	

Resource Hacker 向我们显示了该对话框的相关数据,包括标题(显示在窗口标题栏上的),该对话框的相关按钮,以及它的各种设置。它也打开了一个窗口向我们准确的显示了该对话框的样子,这里的是关于对话框。在点了左侧一堆目录以后,就找到了我们想要的:



是不是看起来挺面熟的? 注意对话框的名字是"REMINDER"。有时候 Windows 用名字来引用一个对话框,有时候用 ID。这里它用名字"REMINDER"。 现在我们知道了所有我们需要的,Olly 载入应用,转到"search for strings"。 咱们来搜索"REMINDER":

30401000 30401002 30401003	\$~	EB 66 62	R Found strings are	
3401004		3A	Address Disassembly	Text string
401005		28	00401000 JMP SHORT WinRAR, 00401012	Initial CPU selectioni
401007 401008 401009 401009		28 48 4F 4F	0040100D ASCII "0+1J",0 0040124B MOV EDX,WinRAR.004A21CC 0040170C ASCII "SaveFilePos",0 00401754 ASCII "ComprDataIO",0	ASCII "CMT"
0100B 0100C 0100D	5	4B 90 E9	00401704 ASCII "Unpack", 0 00401704 ASCII "BL", 0 0040170C ASCII "DL", 0 00401724 ASCII "PL", 0	earch for
01017 0101A 0101F		C1E A3 52	00401A82 DD WinRAR.004000 00401AA4 ASCII "Array <wch 00401B2C ASCII "Array<wch 00401B2C ASCII "Array<wch< td=""><td><u> </u></td></wch<></wch </wch 	<u> </u>
01020 01022 01027 01027		6A E8 8BD F8	00401B3C H3CII "Array <cha 00401C70 ASCII "Array<cha 00401C70 ASCII "Array<cha 004020C0 ASCII "RawWrite"</cha </cha </cha 	itive
0102E	1	5Å	00402110 ASCII "Array (uns Finite score	na
0102F		ES	00402120 HSCII "hary",0	
01039	1	6A	00402194 ASCII "har> *".0	UK. Cancel
0103B		ES	004025C2 PUSH WinRHR.004H	
01040	1	68	004028C8 ASCII "File",0	
01046		6A	00402914 PUSH WinRAR.004A21E4	ASCII "stdin"
01048		E8 02	0040304D MOU EDX. WinRAR, 004H21CC	UNICODE ""
00401048 00401040 00401052		E8 A3 6A	00402CD6 PUSH WinRAR.004A21CC 0040304D MOV EDX,WinRAR.004A21EA 00403121 MOV EDX,WinRAR.004A21EA 00409289 DSCI1 "Environment" 0	ASCII "CMT" UNICODE "" UNICODE ""

咱们在列表中看到它了:

Address	Disassembly	Text string
0043F6A3	MOU EDX, WinRAR, 004A6912	ASCII "HELPFileMenu"
0043F6L6	MOU EDX, WINKHK, 004H691F	DSCIT "HELFLOMMANDSMENU"
0043F712	MOV EDX, WinRAR, 004A693E	ASCII "HELPFavoritesMenu"
0043F738	MOV EDX, WinRAR.004A6950	ASCII "HELPOptionsMenu"
0043F766	MOV EDX, WinRAR.004A6960	ASCII "HELPHelphenu"
0043F7D8	MOV EDX, WinRAR, 004A696D	ASCII "rarkey"
00435600	PUSH WINKHK,004H6974	HOULT "YONNY"
004405F5	PUSH WinRAR, 00486982	ASCII "http://www.rarlab.com"
00440686	PUSH WinRAR.004A6998	ASCII "ABOUTRARDLG"
004406AA	MOV EDX, WinRAR, 004A69A4	ASCII ""
00440716	MOV EDX, WinRAR. 004A69B0	ASCII "Detailed"
0044071B	MOU ERX, WINRAR, 00446947	HSCII "FileList"
0044074H	MOU FOX WINKHK.004H69B0	OSCIL "Filelist"

双击它,我们来到了与使用调用栈同样的区块:

004357881	. 68 60404600	1100 EHX. WINKHK. 00404060	
0049E29D	E9 40CD0100	COLL WINDOR 00450400	
00405700	- CO 4HODOTOO	IND UL DOD OCAMODDO	
0043F792	.V E9 30140000	JAR WINKHK.00440BD3	A second s
0043E792	> 833D F0764C00 00	CMP DWORD PTR DS: [4C76F0],0	Case 113 (WM TIMER) of switch 0043E004
BRADEZOE	75 70	INT CUART WINDER ARADEAT	and the surface of the second second second
D043F19E	~ 13 TU	ONC SHORT WINNER, 0045F610	
BRASELING	. 8095 H4FHFFFF	LEH EDX, DWORD FIR SS: LEBP-5501	
0043F706	R8 08504000	MOU FOX WINROR 00405008	
DOMOFTOD	. 0000	YOD FOU FOU	
0043FTHB	. 3369	AUK ELA, ELA	
0043F7AD	. E8 2E7A0100	CALL WinRAR,004571E0	
DDdgEZE2	9030 95694000 00	CMP BUTE PTP DS+ F4048951 0	
DEADERDO	. 0000 50004H00 00	INT OUODT WE DOD COMPTOND	
0043E7B9	. 75 62	UNC SHURT WINKHK.0043F810	
0043F7BB	. 803D 8C854C00 00	CMP BYTE PTR DS: [4C858C].0	
0043E7C2	v 75 59	INT SHORT WINDOR 0043E810	
DOTOL TOP	0000 04004000 00	OND DUTE DTD DO: 54D00041 0	
9643E764	. 8030 24204800 00	UNP BYTE FIR DS: L4B20241,0	
0043F7CB	.v 75 50	JNZ SHORT WinRAR,0043F81D	
QQ49E2CD	ODOE GAEGEEEE	I FO FOY DWORD PTP CC. [FPP-FFC]	
DOTOFTED.	. 0000 447 47111		
0043F7U3	. E8 8804FUFF	CHLL WINKHK, 0040BC60	
0043F7D8	. BA 6D694A00	MOU EDX.WinRAR.0048696D	ASCII "rarkey"
88425700	P9 04000000	MOLL ECY 4	
0045FTDD	. 57 00000000	HOV LONG OF COMPANY	
0043F7E2	. E8 81F9FCFF	CHLL WINKHR, 0040F168	A second state of the second
0043F7E7	. 8500	TEST EAX.EAX	kernel32.BaseThreadInitThunk
19940E7E9	74 99	E CUNDT HISDOD GRADEOID	in the second seco
SO4OFTE2		DE OHONT WITHANY OUGO DID	
UU43F7EB	. H1 20804000	MOV EHX, DWORD PIR DS: 14080201	
0043E7E0	. 83F8 28	CMP EBX.28	
B640E7E9	7E 04	IC CUOPT WEAPOD 0040E7E9	
0043FFF3	·* /F 04	OG SHORT WITTHIN DOASFIES	a service and the state of the service and the service of the serv
0043F7F5	. 8500	TEST EHX, EHX	kernel32.Baselhreadinitihunk
0043E7E7	-× 7D 24	JGE SHORT WinBAR, 0043E810	
00405750	X CCOE 954000 01	MOU DUTE DTD DC. FANCOGET 1	
BB40FTF2	/ CODS 20004H00 01	HOV DITE FIR DOLLAHOODOJ, I	and a state of the
80431-866	. 6H UU	PUSH 0	r Param = NULL
9943E892	68 302E4800	PUSH MinRAR, 00482E30	$P_{\rm e}$ = MinB9R, 00482E30
00405007	EESE ACADACOO	DUCH DHODD DTD DC. FACADACT	houses - NULL
99451397	. FF35 40404000	FUSH DWORD FIR DS:L4040401	howher - Hocc
0043F800	. 68 74694H00	PUSH WINKHR, 004H6974	Displace - "REMINDER"
0043F812	. FF35 48204B00	PUSH DWORD PTR DS: [4B2048]	hInst = NULL
DRADEOTO.	E9 17100400	COLL / IMD OLICEDOD DislooDou Doosmally	DislocPouPsysm0
00400010	. 20 1,100000	CHEL NONFIGODENSZIDIATOSODAFATANA/	auta togoostarana
00435810	> 803D 94684H00 00	UNP BYTE FIR DS:L4H68941,0	
0043F824	·v 74 1C	JE SHORT WinRAR,0043F842	
00425026	0000 E4764000 00	CMD DUTE DTD DC+F4C74E41 0	
00101020	. 0000 14104000 00	NR CUOPT HE DOD CONCEAS	
9043F820	.v r5 13	JNZ SHUKI WINKHK, 0043F842	
0043F82F	. C605 94684A00 00	MOU BYTE PTR DS: [4A6894].0	
DOADEOOL	60 00	PIICH 0	CIParian - 0
20402000		DUCH O	
0043F838	. 6H 00	PUSH 0	wraram = 0
0043F83A	. 68 10	PUSH 10	Message = WM CLOSE
BB425920	56	PHEN FOT	hided = NULL
00401000	. Fo por porcos		tional - Hore
0043F83D	• E9 D91D0600	UHLL NUMP.&USENSZ.FOSTNESSageH/	Fostnessager
0043F842	> 833D F0764C00 00	CMP DWORD PTR DS: [4C76F0].0	
DDdgEgag	v 25 20	IN7 SHORT MUKROR 0043F979	
00401042		OND DUODD DTD DO-5407(501 C	
0043F84B	. 8330 E8764000 00	CHP DWORD FIR DS: L4C76E81, M	
0043F852	.~ 75 24	UNZ SHURT WinRAR, 0043F878	
DD43E854	833D 08774000 FF	CMP DWORD PTR DS: [4077081, -1	
00100000	74 1D	UE CHODT WE DOD COMPOZO	
0093F85B	· 74 1B	UE SHUKI WINKHK.0043F878	Territoria de la construcción de la
0043F85D	. 6A 0A	PUSH ØR	\mathbf{r} Timeout = 10. ms
DOARESEE	FE35 09774000	PUSH DWORD PTR DS+F4C77081	bObject = NULL
Sector Sector	1100 00114000	T CONTENAND THE DOLETON COOL	The state of the local state of the state of

事实上,你可以看到传递给 DialogBoxParamA 的其中一个参数就是 "REMINDER"。如果资源是通过 ID 而不是名字来标示,我们可以通过右键反 汇编窗口,选择"Search for"->"Command"来找到它。然后在弹出的对话 框中输入"PUSH xx", xx 是资源的 ID (十六进制的)。这也会将你带至对话框 的调用 CALL。

七、最后一件事

如果你看看众多的破解二进制文件的教程,会发现有一个方法是简单的通 过 Resource Hacker 来删除对话框来实现的。本例中这是管用的,你再也不会 看到 nag 窗口,但是这个方法不总是好用,因为它完全依赖于程序是如何处理 丢失的资源的。之所以介绍这个简单的技术,是因为它总是值得试试的。

ps:别忘了将你的日期改回去☺。

第十六章(上): Windows 消息的处理

好,在干掉了两个病毒(一个是我身体上的,另一个是电脑的)以后,我 最终还是上传了最新的教程。本章是其中的三分之一,所有三章处理的都是同 一个 crackme (相当难的一个),是 Detten 写的 Crackme12。第一部分我们将 学习 Windows 消息是怎么工作的。第二部分是关于自修改代码,该部分我们也 会破解该应用。在第三及最后那部分我们介绍暴力破解。如你所猜测的,在第 三部分我们将爆破这个二进制文件。每一部分都会继续前面部分的研究结果。

这个系列的三部分都比较有挑战性,不过我保证如果你花时间并自己动手 实践,你会获得逆向领域中的很关键的知识。记住,如果有任何问题,就在<u>论</u> <u>坛</u>里随意发问。我也会在每一章的最后布置作业,让你为下一章做好准备。课 后作业是真正学习的地方。

一如既往,你需要的相关文件可以在<u>下载</u>页下载。对于第一部分,下载中包括 crackme,以及一份 Windows 消息备忘单。

那么,事不宜迟,咱们开始吧...。

一、Windows 消息简介

本章我们将会讨论 Windows 消息,以及处理它们的过程。几乎所有的程序, 除了用 Visual Basic *唉*、. NET 或 java 写的程序以外,任务都是通过使用消 息驱动回调过程来完成的。意思就是,与 DOS 时代程序不同,在 Windows 中 你只需设置窗口,提供各种你想要显示的设置、位图、菜单项等,然后你再提 供一个循环运行到程序结束。这个循环的唯一责任是从 Windows 接收"消息", 然后再将提发送到我们应用的回调函数。这些消息可以是任何东西,从移动鼠 标到点击一个按钮,到点击"X"来关闭一个应用。当我们在做一个 Windows 程序时,我们要在 WinMain 过程中编写一个无限循环,以及一个无论消息什么 时候到来都可以调用的地址。这个地址就是回调(函数的地址。译者注:这几 个字是我补充的)。然后该循环用我们提供的地址将其接收到的消息发送给我们 的回调函数,在回调函数中我们决定是否对特定的消息做处理,或只让 Windows 处理它。

例如,我们想要显示一个带有 OK 按钮的警告消息框。我们只关心 OK 按钮 被点击的消息。我们不关心用户是否移动了窗口 (WM_MOVE 消息),或者是 点击了窗口 OK 按钮以外的某处 (WM_MOUSEBUTTONDOWN 消息)。不过当 OK 按钮被点击的消息传来的时候,那就是我们要做些什么的时候了。所有我们 不想处理的消息,Windows 会为我们处理。对于我们想要处理的消息,我们只 需重写相关消息的处理,做我们想做的。

设置窗口以及包含循环的主过程叫做 WinMain,如果是窗口的的话回调函数通常被叫做 WndProc,如果是对话框的话回调函数通常被叫做 DlgProc,虽然这些名字可以任何其他的东西。

我在下载中包含了一个所有 Windows 消息的指南,你在学习本章时应该打 开看看。你可以在<u>教程</u>页下载到所有的文件。你也可以在<u>工具</u>页面下载 windows 消息备忘单。

二、载入应用

Olly 载入 Crackme12. exe, 咱们来看看:

00401000	Cá	6A 00	PUSH 0	Pomeda Le = NULL
00401002		E8 C5040000	CALL <jmp.&kernel32.getmodulehandlea></jmp.&kernel32.getmodulehandlea>	GetModuleHandleA
00401007		A3 28304000	MOV DWORD PTR DS: [403028], EAX	kernel32.BaseThreadInitThunk
0040100C	1.0	6A 00	PUSH Ø	FlParam = NULL
0040100E		68 2B104000	PUSH crackme1.0040102B	DigProc = crackme1.0040102B
00401013		6A 00	PUSH 0	hOwner = NULL
00401015		68 20030000	PUSH 320	pTemplate = 320
0040101A	1.2	FF35 28304000	PUSH DWORD PTR DS: [403028]	hInst = NULL
00401020	1.5	E8 8F040000	CALL (JMP.&USER32.DialogBoxParamA)	DialogBoxParamA
00401025	1.0	50	PUSH EAX	FExitCode = 761F3388
00401026	L.	E8 98040000	CALL (JMP.&KERNEL32.ExitProcess)	ExitProcess
0040102B	r	55	PUSH EBP	Children and the
0040102C	10	8BEC	MOU EBP. ESP	
0040102E		817D 0C 10010000	CMP [RRG. 2], 110	
00401085	L.v.	75 37	JNZ SHORT crackme1,0040106E	
00401037	1.2	C705 48304000 000	MOV DWORD PTR DS: [403048].0	
and the second sec	1.1	TITE STORESTER AND		

这个一个标准的应用程序,在使用一个对话框作为主窗口,看起来像是用 C 或 C++写的。

如果程序使用的是常规窗口而不是对话框窗口,它看起来会不一样。参 见下面的。

注意参数是被压入堆栈,以及对 DialogBoxParamA 的调用。这个将对话框 设置成程序的主窗口(而不是普通窗口,不过别太在意这些细节,这真的没什 么关系)。咱们看看有关 DialogBoxParamA 的帮助怎么说:

Nin32 Programmer's Reference				
File Edit Bookmark Options H	Help			
<u>Contents</u> Index Back <u><</u>	<u>></u> >			
DialogBoxParam	Quick Info	Overview	Group	1
The DialogBoxParam function of box, the function passes an appli <u>WM_INITDIALOG</u> message. An a	reates a modal cation-defined v application can	l dialog box fror value to the dial use this value t	n a dialog bo og box proc o initialize di	ox template resource. Before displaying the dialog edure as the <i>IParam</i> parameter of the ialog box controls.
int DialogBoxParam(HINSTANCE hInstance, LPCTSTR IpTemplateName, HWND hWndParent, DLGPROC IpDialogFunc, LPARAM dwInitParam);	// handle to // identifies // handle to // pointer to // initializat	o application ins dialog box tem o owner window o dialog box pro ion value	tance plate cedure 🔫	
Parameters				
hInstance Identifies an instance of the r	nodule whose e	executable file o	ontains the	dialog box template.
IpTemplateName Identifies the dialog box temp the name of the dialog box te the parameter specifies a res identifier. You can use the M	plate. This para mplate or an in ource identifier AKEINTRESOL	meter is either t teger value that , its high-order <u>JRCE</u> macro to	he pointer to specifies th word must b create this v	o a null-terminated character string that specifies he resource identifier of the dialog box template. If e zero and its low-order word must contain the value.
hWndParent Identifies the window that ow	ns the dialog bo	ox.		
IpDialogFunc Points to the dialog box proc function.	edure. For more	e <mark>information a</mark> t	out the dialo	og box procedure, see the <u>DialogProc</u> callback
dwInitParam Specifies the value to pass to	the dialog box	in the IParam	narameter of	f the WM INITDIALOG message
	, the dialog bes		parameter er	
Return Values	m value ie the v	aluo of the pDo	sult paramot	tor encoified in the call to the EndDialog function
used to terminate the dialog box.	in value is the v	alue of the line	Suit paramer	ter specified in the call to the Endbratog function
If the function fails, the return valu	ue is -1.			
Remarke				
The DialogBoxParam function u	uses the Create	WindowEx fu	nction to cre	ate the dialog box. DialogBoxParam then
对于我们的目的来	よい マノ	► CALL ∄	的最重要	E的东西是 DI GPROC 的地址。
7是我们应用程序的原	可调函数的	内地北。月	↓ ~ <i>~ ~ ~ ~ ~</i> 月 干 分 珥	里所有的 windows 消息 回头
后方汇编化码 能够深	青晰的看到	副这个抽题	11 1 / C-1 - •	
	マーバーリン目 エ PUSH Ø	うくり てほた	1	CoModule = NULL
0401002 . E8 C5040000 10401007 . R3 28304000 10401007 . R3 28304000 10401007 . G8 28104000 10401002 . G8 28104000 10401013 . GA 00 10401013 . GA 00 10401013 . FF35 28304000	NALL CUMP.&KER MOV DWORD PTR PUSH 0 PUSH 0 PUSH 0 PUSH 320 PUSH DWORD PTF	NEL32.GetModu DS:[403028],E/ 00401028 CDS:[403028]	leHandleA> AX	Gettlodu LeHandleA kernel32. BaseThreadIn itThunk IParam = NULL DigProc = crackme1.0040102B hOwner = NULL pTemplate = 320 hInst = NULL
10401020 · E8 8F040000 · 10401025 · 50 · 10401025 · 50 · 10401026 · F8 98040000	CALL KUMP.&USE	R32.DialogBoxf	ParamA>	ExitCode = 761F3388
1040102B . 55 1040102C . 8BEC	PUSH EBP MOV EBP,ESP	-		

这里它是 40102B。咱们过去看看它长啥样。这将是...。

三、主对话框消息处理回调函数

这里我们能够看到它的开头:

00401025	120	50	PUSH	EAX	FERItCode = 761F3388
00401026	ι.	E8 98040000	CALL	KUMP.&KERNEL32.ExitProcess>	ExitProcess
0040102B	F .	55	PUSH	EBP	
00401020		9170 OC 10010000	CMP	LBP, ESP	
00401035	1.4	75 37	UNZ	SHORT crackme1,0040106E	
00401037		C705 48304000 000	MOU	DWORD PTR DS: [403048],0	
00401041		C705 38304000 ADI	MOU	DWORD PTR DS: [403038], 0DEAD	
00401046		C705 40304000 424	MOU	DWORD PTR DS: 14030303, 00EHD	and the second se
0040105F		C705 4C304000 003	MOÙ	DWORD PTR DS: [40304C]. crackme1.0040	ASCII "An error occured"
00401069	.~	E9 DE010000	JMP	crackme1.0040124C	Construction of the second sec
0040106E	>	8370 00 10	CMP	CHOPT appakent 99491991	And an and the second sec
00401074	.*	FF75 08	PUSH	[ARG, 1]	Chlind = 7EFDE000
00401077		E8 32040000	CALL	<pre><ump.&user32.destroywindow></ump.&user32.destroywindow></pre>	DestroyWindow
0040107C	:*	E9 CB010000	UMP	crackme1.0040124C	and the second second
00401081	1.v	0F85 85010000	UN2	crackme1,00401243	
0040108E		8B45 10	MOU	EAX, [ARG.3]	
00401091		8855 10	MOU	EDX, LARG. 31	
00401094	2	66: 0BD2	OR D	EDA, 10 IX. DX	
0040109A		0F85 AC010000	UNZ	crackmel.0040124C	
00401000		66:83F8 65	CMP	AX,65	
00401004	•*	69 01	PLICH	SHURI CTACKMEL,00401082	
004010A8		E8 F8010000	CALL	crackme1.004012A5	
004010AD	.*	E9 40010000	UMP	crackme1.004011F2	
004010B2	2	55:83F8 55	UNP-	SHORT enackmel 00401004	
004010B8		6A 02	PUSH	2	
004010BA		E8 E6010000	CALL	crackme1.004012A5	
004010BF	1	E9 2E010000	CMP	orackme1.004011F2	
00401008	1.4	75 ØC	UNZ	SHORT crackme1.004010D6	
004010CA		6A 03	PUSH	3	
00401000		E8 U4010000	IMD	crackme1.004012H5	
004010D6	Š	66:83F8 68	CMP	AX.68	
004010DA		75 ØC	UNZ	SHORT crackme1.004010E8	
004010DC		6H 04	PUSH	4	
004010E3	1.	E9 0A010000	JMP	crackme1.004011F2	
004010E8	N	66:83F8 69	CMP	AX,69	
004010EC	.*	75 ØC	DITCH	SHORT crackme1,004010FR	
004010F0		E8 B0010000	CALL	crackme1,004012A5	
004010F5	.~	E9 F800000	UMP .	crackme1.004011F2	
004010FA	>	66:83F8 6A	CMP	RX,6R SHOPT empolyment 0040110C	
00401100		6A 06	PUSH	6	
00401102	1	E8 9E010000	CALL	crackme1.004012A5	
00401107	1	E9 E6000000	CMP	orackme1,004011F2	
00401110		75 00	UNZ	SHORT crackme1.0040111E	
00401112		6A 07	PUSH	7	
00401114	1	E8 80010000	IMP	crackme1.004012H5	
0040111F	Š	66:83F8 6C	CMP	AX.6C	
09401122		75 ØC	JNZ	SHORT crackme1.00401130	
00401124		6H 08 F8 20010000	COLL	S crackmel 00401205	
0040112B	i.v	E9 C2000000	JMP	crackme1.004011F2	
00401130	2	66:83F8 6D	CMP	AX,6D	
00401134	.*	75 ØC	DUCU	SHORT crackme1,00401142	
00401138		E8 68010000	CALL	crackme1.004012A5	
0040113D		E9 B0000000	UMP	crackme1.004011F2	
00401142	>	66:83F8 6E	CMP INZ	SHORT experiment 00401154	
00401148	i.	6A ØA	PUSH	ØÅ	
00401148		E8 56010000	CALL	crackme1.004012A5	
0040114F	1	E9 9E000000	CMP	orackme1,004011F2	
00401158		75 00	JNZ	SHORT crackme1.00401166	
0040115A	1	6A 0B	PUSH	0B	
00401150		E8 44010000	IMP	crackme1.004012H5	
00401166	×	66:83F8 70	CMP	AX,70	

这是一个相当普通的 DlgProc。它通常就是一个真正的大的 switch 语句, 虽然在汇编形式下,变成了一个真正的大的 if/then 语句。如果你通读了我的 上一章,这个看起来应该比较熟悉,本例中它唯一的不同是,Olly 无法指出 case 标签 (ie. Case 113 (WM_TIMER))。

这个过程在这里有一个原因:是为了响应我们感兴趣的 windows 消息。如果你仔细看的话,你会看到一堆比较和跳转语句。这是将每一部分代码与 Windows 发送过来的消息 ID 进行核对。如果代码匹配了其中一个比较语句,该 代码就会运行(译者注:感觉作者这几句怎么那么别扭呢)。否则,它就会尝试 所有的比较,没有匹配的话,它就会被发送给 Windows,让 Windows 来处理。 咱们来深入的看看这个过程。继续运行程序:

1	2	3	A	В	С
4	5	6	D	E	F
7	8	9	Clea	ar A	.bout

至少可以说它是一个很奇怪的 crackme。咱们来把玩把玩。你会发现你可以持续点击按钮,不过什么都不会发生,虽然它有一个"clear"按钮。看起来 它希望咱们输入一个指定的代码,如果我们不那么做,程序就什么都不做。

现在咱们在 DlgProc 代码的起始处也就是 40102B 设置一个 BP。重启应用, 我们能观察到有消息来了:

00401020	. E8 8F040000	CALL (JMP.&USER32.DialogBoxParamA)	-DialogBoxParamA
00401026	L: ES 98040000	CALL (JMP.&KERNEL32.ExitProcess)	-ExitProcess
COMPTORE	r. 55	PUSH EBP	A STREET AND A STR
0040102C	. SBEC	MOV EBP, ESP	
0040102E	. 817D 0C 10010000	CMP [ARG.2],110	
00401085	.~ 75 37	JNZ SHORT crackme1.0040106E	
00401037	. C705 48304000 000	MOV DWORD PTR DS:[403048],0	
00401041	. C705 38304000 ADI	MOV DWORD PTR DS: [403038], 0DEAD	
0040104B	. C705 3C304000 ADI	MOV DWORD PTR DS: [40303C], 0DEAD	
00401055	. C705 40304000 424	MOV DWORD PTR DS: [403040], 42424242	
0040105F	. C705 4C304000 00	MOV DWORD PTR DS:[40304C], crackme1.00403000	ASCII "An error occured"
00401069	.~ E9 DE010000	JMP crackme1.0040124C	and the same of the state of the
0040106E	> 837D 0C 10	CMP [ARG, 21, 10	
00401072	.~ 75 00	JNZ SHORT crackme1.00401081	NAME OF TAXABLE
00401074	. FF75 08	PUSH [ARG.1]	filled = 0040102B
00401077	. E8 32040000	CALL (JMP.&USER32.DestroyWindow)	DestroyWindow
0040107C	.~ E9 CB010000	UMP crackme1.0040124C	
00401081	> 8170 00 11010000	CMP 1HR6.21,111	
00401088	.~ 0F85 85010000	UNZ crackme1.00401243	
0040108E	· 8845 10	MOV EHX, LHRG. 31	
00401091	. 8855 10	MUV EDX, CHRG. 31	
00401094	. LIEH 10	SHK EDX, 10	
00401097	. 66:0BU2	UK DX, DX	
00401094	.~ 0F85 HC010000	UNZ crackmel.00401240	

一启动应用,我们立马就断在了那个 BP。你会发现在我们开始进行第一次 比较的地方有几条指令

40102E CMP [ARG. 2], 110

如果你在本章相关下载中的 Windows 消息备忘单中查询 ID 110 的话,就 会发现 110 是 InitDialog 的编码:


这个消息给了我们的应用一个机会来初始化一下东西。如果你单步执行,并且消息是 INITDIALOG 的话,我们会直接到 401037 执行相关指令:



看看下面的信息区,可以看到 ARG. 2 不是 110 而是 30:

004010B2 004010B6		66:83F8 66 75 0C	CMP AX.66 UNZ SHORT crackme1.004010C
Stack SS	:[0012	FC54]=00000030	3
Address	Hex c	ARG.2 = 3	0 = WM_SETFONT
00402010	7H 78	6a 6al 7a 3a 6	

在我们的表中, 30 是 set font(设置字体)消息。所以这是 Windows 发送的第一个消息。

下一个是和 10 进行比较,在咱们的消息列表中是 WM_CLOSE:

0040102C	55 8BEC	PUSH EBP MOU EBP, ESP Compare with 1	0, which is WM_CLOSE
0040102E	. 817D 0C 10010000	CMP [ARG.2],110	
00401035	.~ 75 37	JNZ SHORT crackme1.0040106E	
00401037	. C705 48304000 00	(MOV DWORD PTR DS:[403048],0	
00401041	. C705 38304000 AD	MOV DWORD PTR DS: [403038], 000AD	
0040104B	. C705 3C304000 AD	MOV DWORD PTR DS: [40303C], UDERD	
00401055	. C705 40304000 42	MOV DWORD PTR DS: [403040],42424242	and the second second at
0040105F	. C705 4C304000 00	MOV DWORD PTR DS: [40304C], crackme1,00403000	ASCII "An error occured"
00401069	.V E9 DE010000	JMP crackme1.0040 C	And the construction of the second
0040106E	> 98370 0C 10	CMP [ARG.2],10	
00401072	.~ 75 00	UNZ SHURT crackme1.00401081	and a second second second second second
00401074	. FF75 08	PUSH LHRG.11	fhwnd = 002703HH ('Crackme#12 by)
00401077	. E8 32040000	CHLL (JMP.&USER32.DestroyWindow)	DestroyWindow
00401070	·~ E9 CB010000	JMP crackmel.00401240	the second se
00401081	> SI/D 0C 11010000		
00401088	·~ 0F85 85010000	UNC CRACKMEL,00401245	

所以当关闭按钮被点击时,这段代码就会被执行。下一个要比较的是 111, 它是 WM_COMMAND:



WM-COMMAND 包揽了好几种 Windows 消息,通常与资源相关联,例如 点击按钮、选择菜单或点击工具栏中的图标。此外,对于一个 WM-COMMAND 消息来说,在 ARG. 3 中存储了一个整型数据,用来帮助弄清楚命令消息。例如, 如果你点击一个按钮,就会传来一个 WM-COMMAND 消息,并且 ARG. 3 中有 可能保存有按钮的 ID。如果你正在用一个徒手绘画程序画画, ARG. 3 可能保存 有当前鼠标的 X 和 Y 坐标。



仔细看这个,能够发现过程处理的其他消息只剩 WM_COMMAND 了(真的,每一个 WM_COMMAND 都有可能是一个不同的"类型")。如果你单步执行,就会发现对于当前的消息 QM_SETFONT,没有与之有关的代码可以执行,

只是在我们的过程的结尾返回了。这是在告知 Windows,我们希望 Windows 来处理这个消息,而不是由我们来处理:



这回它是一个 WM_COMMAND 消息。向下单步执行到 401081 检测该消息 的比较指令处, 咱们再仔细看看 WM_COMMAND 的处理程序:

00401077 .E8 32040000 CALL CMP & USERS2. DestroyWindow> 0040107C E9 CB010000 UMP crackme1.0040124C 00401081 > 817D 0C 11010000 CMP LARG.21.111 00401085 0F85 B5010000 UM2 crackme1.00401243 00401095 8845 10 MOV EDX, LARG.31 00401094 .CIEA 10 SHR EDX, 10 00401097 66:0BD2 OR DX, DX 00401097 66:0BD2 OR DX, DX 00401094 VF8 AC010000 UM2 crackme1.0040124C	1
0040107C .~ E9 CB010000 UMP crackme1.0040124C 00401031 > S17D 0C 11010000 CMP crackme1.0040124C 00401031 > S17D 0C 11010000 UM2 crackme1.00401243 0040103E . WF85 B5010000 UM2 crackme1.00401243 0040109E . SB45 10 MOV EAX_LARG.31 00401094 . CLEA 10 SHR EDX,10 00401097 . 66:0BD2 OR DX,DX 00401097 . 0F85 AC010000 UM2 crackme1.00401240	
00401081 > > S17D 0C 11010000 CMP LARG.21,111 00401081 .v 0F85 B5010000 UH2 crackme1.00401243 00401091 .8845 10 MOV EAX.1ARG.31 00401094 . B855 10 MOV EAX.1ARG.31 00401094 . C1EA 10 SHR EDX.10 00401097 . 66:0BD2 OR DX.DX 00401097 . 0F85 AC010000 UR2 crackme1.00401240	
00401033 .~ 0F85 B5010000 UN2 crackme1.00401243 0040103E . 8845 10 MOV EDX, LARG.31 00401091 . 8855 10 MOV EDX, LARG.31 00401091 . 2855 10 MOV EDX, LARG.31 00401094 . C1EA 10 SHR EDX, 10 00401097 . 66:0BD2 OR DX, DX 00401094 . VF85 AC010000 . M2 crackme1.00401240	
004010SE . 8845 10 MOV EAX, LARG. 31 00401091 . 8855 10 MOV EDX, LARG. 31 00401094 . C1EA 10 SHR EDX, 10 00401097 . 66:0BD2 OR DX, DX 00401097 . 0F85 AC010000 JNZ 00401097 . 0F85 AC010000 JNZ	100
00401091 . 8855 10 MOV EDX, LARG. 31 00401094 . C1EA 10 SHR EDX, 10 00401097 . 66:08D2 OR DX, DX 00401097 . 06401097 . 67:08D2 00401097 . 07:055 AC010000 JNZ crackme1.00401240	
00401094 . CIER 10 SHR EDX,10 00401097 . 66:0BD2 OR DX,DX 00401097 0F85 AC010000 JNZ crackme1.00401240	
00401097 . 66:0BD2 OR DX, DX 0040109A 0F85 AC010000 JNZ crackme1.00401240	
0040109A 0F85 AC010000 JNZ crackme1.00401240	
00401000 . 66:83F8 65 CMP 8X.65	
00401004 .v 75 0C JNZ SHORT crackme1.00401082	
99491996 . 68 91 PUSH 1	
99491998 . F8 F8919999	
09401000 .x E9 40010000 .MP crackmet 004011E2	
99491982 > 66:8358 66 CMP 92.66	
09401086	
99491988 69 92 PISH 2	
99491989 . E8 E6019999	
994919PF F9 2F019999	
99401904 > 66:83E8 67 CMP 9X-67	
99491908	
P04010C0 60 03 PISH 3	
69401900 F8 04019999 COL crackme1, 60401295	
00401001 V E9 10010000 VMP crackme1 004011E2	
99491906 > 66:8358 68 CMP 92.68	
00401000 V 75 0C UN2 SHORT crackmet 004010E8	
PUSH 4	
0040100F 58 C2010000 COL crackmet 00401205	
004010F3 - F9 00010000 - MP crackme1 004011F2	
004010F8 > 66:83F8 69 CMP 9X 69	
1994010EC V Z5 0C VIZ SHORT crackme1, 004010EB	
Redefiger L 60 85 Pist 5	
004010F0 . F8 80010000	
004010F5 E9 F800000 UMP crackme1.004011F2	

注意它将 ARG.3 拷贝到 EAX 和 EDX。然后它对 EDX 完成了 16 位 SHR (右移位)操作。然后对该值做 OR 操作,如果它不是 0,就跳转。基本上这是

在检测参数的两个高位字节是否是 0(你正在读汇编语言的书,不是吗?)。EDX 的低位两字节保存了被影响到的资源 ID。本例中,它不是 0,所以我们跳过剩 下的代码,然后从回调函数返回:



这里我们能看到正在处理的是 111,或者叫 WM_COMMAND 消息:



00402000 CD 2B 12 76 F3 D8 12 76 E2 BB 13 76 00 00402010 7A 70 3A 77 A3 3B 3B 77 42 CF 3C 77 F4

这里我们能看到那个跳转:



再次运行程序,我们有停在了我们设置的 BP。这回我们处理的是 WM_INITDIALOG 消息:



咱们运行对话框初始化部分的前面几行代码:

Augulu Louise	r. 55	PUSH EBP	
0040102C	. SBEC	MOV EBP, ESP	It's 110 /WM INITDIALOG
0040102E	. 817D 0C 10010000	CMP [ARG.2],110	in a file firm nation read
00401035	.~ 75 37	JNZ SHORT crackme1.0040106E	so we run this code
00401037	. C705 48304000 00	MOV DWORD PTR DS: [403048],0	so we full this code
00401041	. C705 38304000 ADI	MOV DWORD PTR DS:[403038],0DEAD	
0040104B	. C705 3C304000 ADI	MOV DWORD PTR DS: [40303C], 0DEAD	
00401055	. C705 40304000 424	MOV DWORD PTR DS: [403040], 42424242	
0040105F	. C705 4C304000 00:	MOV DWORD PTR DS:[40304C], crackme1.00403000	ASCII "An error occured"
00401069	E9 DE010000	JMP crackme1.0040124C	
0040106E	> 9837D 0C 10	CMP [ARG.2], 10	

在这个特殊的 crackme 中,刚好这部分代码比较重要。咱们看到有几个整型被存进以 403038 为起始位置的内存中(颠倒顺序进行访问,403038 是最低位)。咱们先在数据窗口中看看:

Address	Hes	: du	amp			-				-	-	-		-			ASC	II				
00403038	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					 	
00403048	ññ	00	D D	ññ.	Ø Ø	ññ.	Ø Ø	ññ.	R R	Ø Ø	Ø Ø	ññ.	Ñ Â	ÑØ.	ññ.	R R	002		11			
00403058	aa	ññ	ññ.	ññ	ññ	ññ	ññ	ññ	ññ	ññ	ññ.	ññ	ññ.	ññ	ññ.	ñā	111			1.5		
00400069	ãã	ãã	ãã	ãã	ãã	ãã	ãã	ãã	ãã	aa	ãã	aa	ãã	ãã	ãã	ãã						
00400000	00	aa	aa	aa	aa	aa	aa	aa	aa	aa	aa	aa	00	aa	aa	aa						
00403070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		•••	•••	•••	 	••••
00405066	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1000	••			 	
00403098	00	90	90	00	90	90	00	00	00	00	00	00	00	90	90	90		• •		• •	 	
004030H8	90	90	90	96	99	99	00	96	90	90	90	90	90	90	00	NN					 1.0	
004030B8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2.2.2		2.2			
00403008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00						
00403008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					 	
904039F8	RR.	ññ.	ññ.	BB	ññ.	Ø Ø	ññ.	ññ.	RR.	Ø Ø	RR.	ññ.	RR.	ññ.	ññ.	BB	2.5	20	00			003
004030E8	ññ	ñ0	ññ	ññ.	ññ	00	ññ	ññ.	ññ	00	00	ññ	ññ	60	ññ	00	199.5		2.3	12		
00402109	aa	ãã	aa	aa	aa	aa	ãã	aa	aa	00	aa	aa	00	aa	aa	ãã						
88482110	aa	60	00	00	aa	00	00	00	00	00	00	00	00	00	00	00		•••				
00403110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		•••	•••	•••	 	• • •
00408128	00	00	00	00	00	60	00	00	00	00	00	00	00	00	00	90		• •	• •		 	
00403138	90	90	90	90	96	90	00	90	96	90	00	90	96	90	96	90		••	••	••	 	

看,在咱们运行这几行前,它被初始化为 0。现在,单步步过第一个 MOV 指令,什么也没发生,不过一个 0 被拷贝到 403038 处。单步步过下一条指令,能够看到产生的效果:

00401000	
00401037	. C705 48304000 00(MOV DWORD PTR DS:[403048],0
00401041	. C705 38304000 ADI MOV DWORD PTR DS: [403038], 0DEAD
0040104B	. C705 3C304000 AD[MOU DWORD PTR DS: [40303C], 0DEAD
00401055	. C705 40304000 424 MOU DWORD PTR DS: [403040], 42424242
0040105F	. C705 4C304000 00\$ MOU DWORD PTR DS: [40304C], crackme1.00403000

可以看到 0x0DEAD 被拷贝到内存中(以小端序列的形式):

Address	He	< du	4mp	-	-		-	-			-
00403038	AD	DE	00	00	00	00	00	00	00	00	00
00403048	00	00	00	00	00	00	00	00	00	00	00
00403058	00	00	00	00	00	00	00	00	00	00	00
00403065	80	00	10	00	00	00	00	00	00	00	00
0040307	00	00	- B	00	00	00	00	00	00	00	00
0040308.	00	00	83	00	00	90	00	00	00	00	00
00403098	00	00	00	00	00	00	00	00	00	00	00
004030A8	00	00	00	00	00	00	00	00	00	00	00
00402000	00	GIG.	GG.	00	00	00	00	00	00	GG.	00

单步步过下一行,它做了同样的事情,不过是在地址 40303C 处:

Address	He	; di	Imp	-	-	-						
00403038	AD.	DE	00	00	AD.	DE	00	00	00	80	00	00
00403048	00	00	00	00	00	00	00	00	00	00	00	00
00403058	00	00	80	90	la.	88	90	00	00	00	00	00
00403068	00	00	00	00	9	00	1	00	00	00	00	00
00403078	00	00	00	00	10	00	1.5	00	00	00	00	DE
00403088	00	00	00	00	00	00	EL.	00	00	00	00	00
00403098	00	00	00	BA	00	00	00	00	00	00	00	00
004030A8	00	00	00	00	00	00	00	00	00	00	00	06
004030B8	00	00	ØØ	00	00	80	00	00	00	88	00	Ø
	000	-	200	68	200	-	100	1000	100	-	100	200

事实是,以十六进制的双字写入彻底的暴露它对该 crackme 是很重要的☺ (译者注:为啥?)。接下来,在403040 处值 42 被写入了 4 次。在 ASCII 数 据区可以看到 42 的 ASCII 值是 "B":

Address	Hex dump		ASCII
00403038 00403048 00403058 00403058 00403088 00403088 00403088 00403088 00403088 00403088 00403088 00403088	AD DE 00 00 AD DE 00 </th <th>00 42 42 42 42 10 00 00<!--</th--><th>2 00 00 * *</th></th>	00 42 42 42 42 10 00 00 </th <th>2 00 00 * *</th>	2 00 00 * *

最后, 整数 403000 被拷贝到 40304C。Olly 可以分辨出是一个指向以 403000 为起始位置的一段代码或数据(记住是小端序列):

Address	Hex	du	mp				_	_		-						1	ASCII				
00403038	AD	DE	00	00	AD	DE	00	00	42	42	42	42	00	00	00	00	4	.BBI	BB.	2.	12
00403048	00	00	00	00	00	30	40	00	00	00	00	00	00	00	00	00					5
00403058	00	00	00	00	00	30	00	00	00	00	00	00	00	00	00	00				**	
00403068	00	00	00	00	00	10	00	98	00	00	00	00	00	00	00	00					5
00403078	00	00	00	00	00	00	00	10	00	00	00	00	00	00	00	00					1
00403088	00	00	00	00	01	90	00	10	00	00	00	00	00	00	00	00	******				
00403098	00	00	00	00	60	00	00	00	00	00	00	00	00	00	00	00					
004030A8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					
004030B8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					-

最终我们跳到了结尾并返回,等待发过来的下一个消息:



多点 F9 几次(10) 你会看到主对话框窗口被创建出来:



这地方非常的有意思,因为在你点击 F9 时,每点一次在对话框中就会有新 的东西出现(大概运行6次以后),就会接收到一个消息来在对话框中画资源。 下一个消息是 135,或者叫 WM_CTLCOLORBUTTON:

004010D6 004010DA 004010DC	> 66 -> 75 - 69	:83F8 68 ØC 04	CMP AX,68 JNZ SHORT crackme1.0040 PUSH 4
Stack SS	:[0012F9	90]=0000013	5
Address	Hex du	135 = W	M CTLCOLORBTN
00403048	00 00		

在窗口中画了一个按钮:



下一个是一个写着"2"的按钮:

Crackme#12 by Detten	×
2	
	About

这时候按 F9, 你会真切的看到对话框被构建, 一次一个按钮。观察到来的 所有消息以及在表中查询它们是相当的有趣。你会看到有很多消息到来。如果 有那个你不知道的, 就 Google 搜索它, 然后就会得到关于它的相关描述。直至 最后, 底部的 label 控件会被绘制出来, "No access" 文本会被写入进去。整 个窗口就快完成了。在窗口彻底完成前, 我还得按 F9 大概 35 次:



那么现在你知道了一个对话框是如何构建的。设置对话框的基本设置,标题以及整体外观,传进一个回调函数的指针(地址),用来处理所有从 Windows 发送过来的消息。然后 Windows 发送一系列的消息,一次一个的发给回调函数, 给我们机会让我们在我们收到所渴望的消息时运行代码。在对话框被构建完成 后,Windows 就进入一个内部循环,就坐在那等我们干些什么。只要我们一有 动作,一个带有已经发生的动作的 ID 的消息就会被发送给回调函数。然后我们 就可以决定对该消息做些什么或者忽略它,让 Windows 来处理它。

你需要注意的最后一件事是,如果该应用是在 Olly 中运行的,只要在窗口 上移动鼠标就会导致 Olly 暂停在处理新消息的消息处理过程的开始处。 Windows 告诉我们的消息处理过程有鼠标在窗口上移过。基本上,你对对话框 做的任何动作都会发送消息给消息处理过程。

四、作业

1、你能不能找出点击一个按钮后会发生什么,尤其是以 403038 为起始位置的内存内容方面。不同的按钮做的不一样吗?你能够理解代码正在修改这些内存位置了吗?

2、猜一猜密码有多长。

第十六章 (中):何谓自修改

发起帖子的原因是YYSniper没有继续更新后面的文章,我只是按照我看完原版后的理解进行翻译,如果翻译的不对请见谅!并且我会进行程序分析后给出下面的理解。

致谢 YYSniper 提供开始的翻译教程。

转载请注明来自吾爱破解论坛@52PoJie.cn

开始正文了:

现在,我们已经看到了基本的消息处理程序回调函数,让我们看看是否可以使用这个来破解这个crackme。我们可以看到,只有三个消息,这个应用程序处理;110(INITDIALOG),10(destroy_window),和 111(WM_COMMAN)。任何其他消息都被忽略。我们已经经历了初始化对话框的代码,我们真的不在乎销 毁窗口的代码,当我们关闭应用程序时才被调用。因此,任何值得注意的事情发生在wm_command节。所以 让我们只停留在那一段Olly。删除任何旧的断点然后设置一个新的在地址40108e,或经过ID 111比较/跳:



然后F9跑起来,这时打开程序,移动窗口点击窗口都不会被断下,因为这些消息已经被忽略,交给Windows处理了。好了,按下窗口中的"1"按钮,这时OD断在了我们设置的断点上,现在我们看看arg.3的值为0x65

00401088 0040108A 0040108F	. 6A 02 . E8 E6010000 . E9 2E010000	0x2 call crackn jmp crackme
堆栈 55:	[9918FA5C]=999999	65
eax=0040	1028 (Crackmel.00	401026)
地址	HEX 数据	能够能行。
00402000	2F 43 EA 74 45WW	2.52b3#e59 7
00402010	D6 C4 BC 76 9C B	9 BD 76 0C C

如果我们用Resource Hacker打开Crackme,然后打开主窗口,我们会发现0x65对应的10进制为101,在 Resource Hacker中显示的ID就是"1"按钮。

File Edit Vew Aldein	Halp	
B C Decg	Here Timud	
400 • • • • • • • • • • • • • • • • • • •	<pre>000 DIALOU 5. 0, 143, 45 HITLE WE_VISIBLE WE_CAPTION WE_FILMENT CAPTION "Linkswerkell by Densen" LANDAGE LAND TWILLER, SUBLAND ENGLISH UN FORT 10, 765 SANA DESIT CONTROL *0. 765 SANA DESIT CONTROL *0. 765 SANA DESIT CONTROL *0. 765 SANA DESIT CONTROL *0. 104, NUTTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 104, NUTTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 104, NUTTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 104, NUTTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 104, NUTTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE_VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON BE CENTER WE_CHILD WE VISIBLE CONTROL *0. 101, NUTON, DE FORMETTON</pre>	LE 1 H5 TANSTOF, 120, 52, 28, 14 1 H2 TANSTOF, 32, 12, 16, 17 HE TANSTOF, 32, 12, 16, 17 HE TANSTOF, 22, 22, 16, 17 HE TANSTOF, 22, 22, 16, 17 HE TANSTOF, 12, 12, 16, 17 HE TANSTOF, 13, 14, 17 HE TANSTOF, 12, 12, 16, 17 HE TANSTOF, 12, 12, 13, 16, 17 HE TANSTOF, 12, 12, 33, 16, 17 HE TANSTOF, 13, 17 HE TANSTOF, 13, 17 HE TANSTOF, 13, 16 HE TANSTOF, 13, 17 HE TANSTOF, 13, 16 HE TANSTOF, 13, 16

这个值进入到arg.3中,他只是这个按钮的ID,因此我们向下看两行会发现一个比较出现了,和这个ID进行比较,然后继续往下看,就会明白这里的比较和跳转就是与程序的按钮ID进行比较的

00401094	. C1ER 10	SHR EDX.10
00401097	. 66:0BD2	OR DX-DX
0040109A	.v 0F85 AC010000	JNZ crackme1.0040124C
004010A0	. 66:83F8 65	CMP AX, 65
00401004	.v 75 0C	JNZ SHORT crackhel.00401082
004010A6	. 68 01	PUSH 1
004010A8	. E8 F8010000	CALL orackme1.004012A5
034010AD	.v E9 40010000	JMP crackme1.004011F2
00401082	> 66:83F8 66	CMP AX.66
084010B6	.v 75 8C	JMZ SHORT crackme1.004010C4
004010B8	. 6A 02	PUSH 2
004010BA	. E8 E6010000	CALL crackme1.004012A5
004010BF	.~ E9 2E010000	JMP crackme1.004011F2
004010C4	> 66:83F8 67	CMP AX, 67
004010C8	.~ 75 0C	JMZ SHORT crackhel.004010D6
004010CA	. 6A 03	PUSH 3
004010CC	. E8 D4010000	CALL crackme1.004012A5
004010D1	.~ E9 1C010000	JMP crackme1.004011F2
00401006	> 66:83F8 68	CMP AX, 68
004010DA	.v 75 0C	JNZ SHORT crackme1.004010E8
004010DC	. 6A 04	PUSH 4
004010DE	. E8 C2010000	CALL orackme1.004012A5
004010E3	.~ E9 0A010000	JMP crackme1.004011F2
004010E8	> 66:83F8 69	CMP AX, 69
004010EC	.v 75 0C	JNZ SHORT crackhel.004010FA
004010EE	. 6A 05	PUSH 5
004010F8	. ES B0010000	CRLL crackme1.004012R5
884818F5	.~ E9 F8000000	JMP crackme1.004011F2
004010FH	> 66:83F8 6H	CMP HX, 6H
BBABIOFE	.* 15 80	JHZ SHUKI Crackmel.0040110C
88461188	. 6H 06	PUSH 6
00401102	. E8 9E010000	CHLL Crackmel.004012H5
00401107	.V E9 E6000000	UNP CrackMe1.004011F2
00401100	266:8378 66	WE CHOPT descel codeline
00401110	. 40 07	DUCU 7
00401112	- 6H 0/	COLL assalue1 00401205
00401114	- E0 00010000	IMP oppokent 004011E2
00401115 00401115	X 66.0250 6C	CMP DV SC
00401122	25 00	INZ SHORT crackhel 09401120
00401124	69 98	PIISH 8
00401126	E8 70010000	C911 crackme1, 09401295
0040112B	E9 C2000000	JMP crackme1,004011F2

不管你输入的是1或者2,都会进入到同一个Call中,那么我们到这个Call去看看到底处理了什么。

004012A5	г\$	55	main ebp
004012A6		8BEC	mov ebp,esp
004012A8	-	60	pushad
004012A9	1.4	880D 4030400	mov ecx, dword ptr ds:[0x403040]
004012AF	1.201	8B1D 3C30400	mov ebx, dward ptr ds:[0x40303C]
00401285	1241	A1 38304000	mov eax, dword ptr ds:[0x403030]
004012BA	1.00	807D 08 01	cmp byte ptr ss:[ebp+0x8],0x1
004012BE	-~	75 10	jnz Xcrackme1.004012D0
00401200	1.00	81C1 4805000	add ecx, 0x54B
004012C6	1.9	ØFAFD8	imul ebx,eax
00401209	1.20	3301	xor eax,ecx
004012CB	-~	E9 17010000	jmp crackme1.004013E7
004012D0	>	807D 08 02	<pre>cmp byte ptr ss:[ebp+0x8],0x2</pre>
004012D4	1.50	75 12	jnz Xcrackme1.004012E8
004012D6	-	81E9 3302000	sub ecx, 0x233
004012DC	-	6BDB 14	imul ebx,ebx,0x14
004012DF	1911	03C8	add ecx,eax
004012E1	ræi	2308	and ebx,eax
004012E3	-~	E9 FF000000	jmp crackme1.004013E7
004012E8	>	807D 08 03	<pre>cmp byte ptr ss:[ebp+0x8],0x3</pre>
004012EC	-~	75 ØF	jnz Xcrackme1.004012FD
004012EE	191	05 82050000	add eax, 0x582
004012F3	1911	6BC9 16	imul ecx,ecx,0x16
004012F6	1.201	3308	xor ebx,eax
004012F8	-~	E9 EA000000	jmp crackme1.004013E7
004012FD	>	807D 08 04	<pre>cmp byte ptr ss:[ebp+0x8],0x4</pre>
00401301	-~	75 ØF	jnz Xcrackme1.00401312 吾爱破解论坛
00401303	121	2303	and eax, ebx
00401305	- 200	81EB 2212110	sub ebx, 0x111222 www.52pojie.cn

现在我们来到了004012A5这个Call里面了,我们现在打开之前在初始化(WM_INITDIALOG)中的内存数据,数据地址为 00403038,然后我们在数据窗口中可以看到

地址	HE:	、雙	据		-			-	-								ASCII
00403038	ĤĎ	DE	60	80	AD	DE	00	00	42	42	42	42	00	00	66	00	
00403048	00	00	00	99	68	30	40	00	08	00	88	80	00	00	00	89	
00403058	80	80	80	00	08	88	88	88	88	80	00	08	88	00	80	60	www.52poile.cn
08493968	60	60	60	08	00	00	00	60	88	00	00	88	08	00	00	00	

地址中显示了初始化的数据:两次"DEAD",已经4个"42",代码中我们先把4个"42"移动到ecx,然后把两个"DEAD"分别移动到ebx和eax

Regi	isters (FPU)
EAX	00000EAD
ECX	42424242
EBX	ABABDEAD
ESP	0012FA78
EBP	0012FA98
ESI	00000111
EDI	0012FB20

Next, we do a series of compares to findout which button we pushed based on the value that was pushedonto the stack. Here, SS:[EBP+8] is directly accessing this pushed value.Since we clicked the first button,we will perform the first set ofinstructions:接下来,我们做了一系列的比较,找出我们按下按钮后被压入栈的值。在这里,自从我们点击了第一个按钮SS: [EBP+8]直接访问此栈的值.我们将执行第一次操作:

00401285 0040128A 0040128E	. A1 38304000 . 807D 08 01	MOV EAX, DWORD PTR DS: [403038] CHP BYTE PTR SS: [EBP+8], 1 PT SMORT or solved - 90401200
88481208 88481206 88481209 88481209 88481209	 81C1 48050000 0FAFD8 33C1 E9 17010000 802D 08 02 	ADD ECX.648 IMUL EBX.EAX XOR EAX.ECX UMP crackme1.004013E7

下面执行的操作如下; Ecx加上0x54B 后ecx为4242478D, 然后EBX乘以EAX(0xDEAD x0xDEAD)后ebx为C1B080E9。最后我们进行XOR操作Ecx, 然后跳转到4013E7.单步不过这个跳转我们来到:

00401303	· OFAFD9	IMUL EBX, ECX
00401308	> 807D 08 0E	CMP BYTE PTR SS: [EBP+8]. 0E
00401300	.v 75 0D	JNZ SHORT crackmel.004013DB
004013CE	. 35 55550500	XOR EAX, 55555
004013D3	. 81EB 51735800	SUB EBX, 587351
00401309	.~ EB 0C	JMP SHORT crackme1.004013E7
00401SDB	> 807D 08 0F	CHP BVTE FTR SS: LEBP+81, OF
004013DF	.~ 75 86	JAC SHUKI Crackme1.004013E7
004013E1	. 0303	HUU EHX, EBX
00401353	. 0309	HOD EDA, ECA
00401357	> FERE 44304000	INC DWORD PTR DS+[402044]
984913FD	. 83 38384888	MOU DWORD PTR DS: [4030381, E0X
004013F2	. 891D 3C304000	MOU DWORD PTR DS: [40303C].EBX
084013F8	. 890D 40304000	MOV DWORD PTR DS: [403040].ECX
004013FE	. 61	POPAD
004013FF	. C9	LEAVE
00401400	L. C2 0400	RETN 4
08401403	\$ 55	PUSH EBP
00401404	. SBEC	MOU EBP, ESP
00401406	. 50	PUSH ERX

我们来到算法的最后,如果你往回看,你会发现所有按钮按下后都会做一系列加、乘和XOR的处理只是不同的 按钮各不相同,然后来到4013e7处。在这里我们把403044加一,把EAX移动到403038,把EBX移动到 40303c,把ECX移动到403040,然后出栈返回到主函数中。

004010/4	. FFID 00	FUSH LHRG.11	1. 11 0013
00401077	. E8 32040000	CALL (JMP.&USER32.DestroyWindow)	Des
00401070	E9 CB010000	JMP_crackme1_00401240	1.222
00401001	> 9170 AC 11010000	CMP COPC 21 111	
00401001	2000 0C0100000		
00401088	·* 0F85 85010000	JNC Crackme1,00401243	
SOUTH THEFT	. 8845 10	MOU EAX, [ARG. 3]	
86461691	. 8B55 10	MOV EDX. CARG. 3]	
00401094	. C1E9 10	SHR EDX. 18	
00401097	66+0BD2	OR DX DX	
00401000	0595 00010000	INZ opport 00401240	
0040100H	.V 0F05 HC010000	OND ON CO	
004010H0	. 66:8318 65	CHP HX, 65	
00401004	·* 75 0C	UNC SHURT crackmel.004010B2	
004010A6	. 6A 01	PUSH 1	
00401008	. E8 F8010000	CALL crackme1,00401285	
00401000	E9 40010000	JMP_crackme1,004011F2	
864916B2	> 66:83F8 66	CMP DX.66	
BRARIADE	75 80	INT SHOPT exactment 004010C4	
00401000	. 60 00	DUCU 2	
004010B8	. DH 02	FUSH 2	
004010BH	. E8 E6010000	CHLL crackmel.004012H5	
004010BF	.~ E9 2E010000	JMP crackme1.004011F2	
004010C4	> 66:83F8 67	CMP AX.67	1
00401008	.v 75 0C	JNZ SHORT crackme1,0040;404	-
RR4R1RCO	60 03	PUSH 3	1
00401000	E9 04010000	COLL exact 0040120E	1
00401000	. E0 04010000	MD sealest 00401152	1
And a second state of the			

然后一个跳转直接跳到4011F2处:



这里把内存地址403048(0)与3进行比较,然后再把内存地址403044和0xA比较,而403044是我们前面的

Call中写入的,我们暂时把这个值猜测为长度,如果403044不是0xA的话,我们会跳走。然后来到暴力破解信息处,同样的,如果403048等于3的话也会来到暴力破解信息处。(译者注:其实这里就已经能够猜到密码为10位,如果连续3次都输错就会出现暴力破解信息)。

100702171	/ 6330 46304000 03	CHP DRUND FIR DELENDERED.	
004011F9	.v. 73 20	JNB SHORT crack#e1.00401228	
804011FB-(. 833D 44304000 0A	CHP DWORD PTR DS: [403044], 09	
00401202	·* 75 30	JHZ SHORT oracknel.00401241	
00401204	. E8 43828888	CALL gracknel.0040144C	The second se
00401209	. 68 00304000	PUSH crackme1.00403000	ASCII "An error occured"
0040120E	. FF75 08	PUSH EARS. 11	nation of the tablets
80401211	. ES ED010000	CRLL cracknel.00401403	
00401216	. C705 44304000 00	MOU DWORD PTR DS: [403044].0	
00401220	. FF05 48304000	INC DWORD PTR DS: [403048]	
88481226	.~ EB 19	JMP SHORT crackme1.00401241	
88481228	> 68 11304000	PUSH cracknel.00403011	"Tent = "Trying to bruteforce?"
0840122D	. 6A 03	PUSH 3	ControllD = 3
0840122F	. FF75 88	PUSH EARG. 1]	hand = 00080528 this stand has been
08401232	. E8 89828888	CALL (JMP.&USER32.SetDigItenTentA)	SetDigItenTentA
00401207	. C705 44304000 00	(MOV DWORD PTR DS: [403044], 0	A REPORT OF THE PARTY OF THE PA
00481241	>> EB 09	JHP SHORT crackhel.00401240	
80481243	> 88 0000000	HOU ERX.0	

让我们继续按下"2"按钮,来到我们的断点位置:

0040106E 00401072 00401074 00401077 00401077 00401070	> 837D 0C 10 • 75 0D • FF75 08 • E8 32040000 • E9 CB010000 > 817D 0C 11010000	CHP CARG.23,10 MNZ SHORT crackme1.00401081 PUSH [ARG.13 CALL (JHP.&USER32.DestroyWindow) JHP crackme1.0040124C CHP CPD 21.11	Child = 00
80481888	- 0F85 B5010000	JNZ crackne1,00401243	
0.0010.002	. 8B45 10	MOU EAX, [ARG. 3]	1
00401091	. 8B55 10	MOU EDX, CARS. 31	
00401094	. C1EA 10	SHR EDX, 10	
00401097	. 66:0BD2	OR DX, DX	
0040109A	.v 0F85 AC010000	JM2 crackme1.0840124C	
00401000	. 66:83F8 65	CHP AX,65	
004010A4	.~ 75 OC	UN2 SHORT crackme1.00401082	and the second second
004010R6	. 6A 01	PUSH 1	11 A A E I - LT
004010A8	. E8 F8010000	CALL crackme1.004012A5	and the second se
004010AD	.v E9 40010000	JMP crackme1.004011F2	0200000 0
00401082	> 66:83F8 66	CMP AX, 66	

参数arg.3为66

AN401130	1 . 75 AC	1107
Stack SS EAX=0000	:[0018F9E8]=00000066 0066	-
Address	Hex dump 喜爱旅艇	Wir.
00403038	20 99 42 42 29 80 80 60 60 60 60 60 60 60 60 60 60 60 60 60	C1 8E 38 86

这与我们之前的推测是一致的

00401086	· 68 01	PUSH 1
00401088	. E8 F8010000	CHLL crackme1.004012H5
004010AD	.~ E9 40010000	JMP crackme1.004011F2
004010B2	> 66:83F8 66	CMP AX, 66
004010B6	.~ 75 00	JNZ SHORT crackme1.004010C4
00401088	. 6A 02	PUSH 2
00401050	. E8 E6010000	CALL crackme1,00401285
004010BF	F9 2F010000	MP crackme1, 004011F2
99491964	> 66.83F8 67	CMP QX 67
00401009	75.00	INZ SUOPT enselvent 00401004
00401000		DUCU O
004010CH	. OH 03	FUSH S
004010CC	. ES D4010000	CALL crackme1.004012A5
00401001	.~ E9 1C010000	JMP crackme1.004011F2
00401006	> 66:83F8 68	CMP BX.68
00/01000		INT SHOPT encokenet 00401059

接下来的操作与按下"1"按钮一样来到了004012A5这个Call中,只是原来的"DEAD"和"42424242"不一样了,以为之前"1"进行过计算了。然后又来到



403048还是为**0**(这里作者问如果等于**3**了会跳到暴力破解信息中是什么原因,上面我的分析是连续**3**次都输错 密码就会出现暴力破解),而**403044**已经变成**2**了,还是不做任何处理进行回到主窗口循环,一定是要我们做 什么动作。(译者注:其实我们在分析的时候可以输入**10**次后按看**403044**和**0**xA比较后的代码,看做了什么)

正式破解Crackme

现在我们已经明白这个程序是怎么工作的了,让我们来打补丁,对于这个程序我们需要一点直觉。根据这个程序的流程,我们没有看到更多的比较和跳转跳到正常流程,真的,我们唯一能看到的都是跳转到暴力破解,地址为4011F9。一个跳转是重4010B2到4011A6是"about"按钮的。"Clean"的跳转会把DEAD和42424242重设到初始化内存中,在地址4011CA处,如果点击"about"然后跟踪代码,你会发现只是打开about窗口,然后回到主函数循环中。"clean"也是做同样的动作,因此还是会跳到暴力破解信息处。现在这里有一个小的直觉进来。每一次我们检查了地址403048,看看我们是否应该跳转到蛮力消息,内容是零和跳转从来没有采取。然而,在地址4011fb处比较403044地址值达到0x0a会跳转。我们也知道,每一次通过循环,403044的值的增加,所以我们可以假设这个计数器记录多少我们按下按钮: (译者注:其实这里我们之前就已经分析了)



当然,你的第一个想法是'是的',但是代码导致错误信息!"。是这样吗?所有的代码是否加载指向一个消息的 指针,该消息说有一个错误,但这显示吗?在代码中没有显示.....或许根本不是这样的。这段代码看起来很可 疑,所以让我们分析它。我们知道需要输入10次后才能会进入到错误信息。所以让我们在地址401204,设置 断点然后删除之前的所有断点并重新启动应用程序(当然我们也可以数10次后断下^_^):



好了,现在已经点击了10次按钮,并且断在了我们的断点处,让我们进去看看做了什么?



嗯,这里会先Call VirtualProtect,在读取完成VirtualProtect后基本可以确认是用来修改内存数据的.举个例 子,如果要修改一段内存,就要给这段内存赋予可写属性,使用VirtualProtect后就可以给这段内存赋予写属 性。现在你可以去修改这段内存了,实际上是在这个领空上进行修改。在这段代码中是自修改代码把内存数据 进行相关计算后写入内存然后call VirtralProtect后恢复原来的属性。看来,这个应用程序正在做类似的事情。 对virtualprotect的最后一个参数是你想要改变属性的内存位置,和第三个参数是在你想改变的部分的字节长 度。在这种情况下,我们可以看到,起始地址为401407,长度为0x1f4(500)。我们也可以看到,第二参数 page_readwrite,使这段可写、可读。让我们看看这段地址,在401407开始,看看会发生什么变化

004013F8		8980 48384888	MOU DWORD PTR DS: (403040].ECX	
004013FE	+	61	POPAD	
004013FF		C9	LERVE	
00401400		C2 8466	REIN 4	
00401403	9	SD	HOLL EPP FOR	the second se
00401404		SEL	DUCH COV	kernel 52 DeseThread to le Thurk
00401405	+	ED OF	HD CHOT constrant 00401440	Kerneloz, base inreadinit inunk
and a rang	+*	GO OF	VIE SHORT CERCKHET CONDIANS	
00401400		90	NOR	
100401400	1.0	42	08.42	CHOR 'R'
69491400		88	DE ED	Sector S
00401400	1	02	0B 82	
0040140E		35 00 43 00	ASCII "5/C",0	
00401412	1.0	01	06.01	
00401413		89	08 89	
00401414		02	08.02	
00401415		88	DB 85	
00401416	+	C2 8488	RETN 8884	
00401419	1.00	82	OB 05	CU205 151
00401416		35	OB OB	CHERK '5'
00401410	1.1	45	DEC EDI	
00401410		15 50000000	ODC EOV OBACCOCO	
00401422	2	C2 0498	PETH SROA	
00401425		82	08.93	
00401426		35	DB 36	CHOR 151
88481427		0E	DE DE	circle o
00401428		60	08.00	
00401429		17	DB 17	
69401429		10	0E 10	
00401428		89	DE 89	
00401420		02	08.03	
00401420	1	83	08.85	
00401425		C2 0488	RE IN CERM	
00401431		20	08 82	CLOD IEF
00401402		16	100 22	Clark 2
00401454	1.11	45 45 00	OSCIL MEEM A	
08481437		89	08.89	
08481438	1.	82	08 82	
00401485		59 58	ASCII "ZX"	
00401438		0466E788	00 EBE76604	
0040143F		40600388	00 88088040	a demonstration of the second s
00481441		E8 78888888	CALL (JHP.&USER32.SetDigItenTentA)	SetDigitenTextB
00401448	2	C9	LEAVE	
00401445	- 1	CZ 0808	HE IN S	subject and man and an end of the second second
B040144C	F ^s	58 5000000	PUSH ERX	kernel32.Baselh/dict lya
00401440		68 58384888	PUDM GEBOKWE1-00403050	puld rolect = crackiel.udiddubd
00401452	•	69 E4010000	DIRU IT	THE TOTECT = PHOE, REPORTS IE
00401459	1	68 07144000	PIEH crackent 99491497	Gitterst - coackest 00401407
00401402		00 07144000	CONT OF BOX META COMOTHON	Harters - CLarkine I' Codo Ladi

Hmmmmm. That looks really suspicious. Itdoesn't look like code at all. Let's keep going and see what theapp changes in this section of memory. Stepjust past the call to VirtualProtect:嗯,这一段确实有惊喜, 看上去不像常规代码,让我们继续看看程序是怎么改变内存的。单步步过Call VirtualProtect:

08401440	rs 50	PUSH ERX	AND RECORD AND ADDRESS OF ADDRESS
00401440	. 68 58304880	PUSH crackwe1.00403058	polidFretect = crackwel.00403058
00401452	. 6A 84	PUSH 4	thewFroteot = PAGE_READWRITE
00401454	. 68 F4010000	PUSH 1F4	Size = 1F4 (500.)
08401459	. 68 07144000	PUSH orackme1.00401407	Address = crackee1.08401407
-0048145E	. E8 6F000000	CALL (.RP. BKERMEL32. VirtualProtect)	UlptualProtect
00401463	. A1 38304000	HOU EAX, DWORD FTR DS: [4838383	21.000 00000
00401469	, 3105 07144000	XOR DWORD PTR DS: [401407], EAX	
0040146E	. 8030 07144000 52	CMP BYTE PTR DS: [401407], 52	
08491475	75 18	JC SHORT crackme1.0040148F	
00401477	- A1 3C304000	HOU ERK, DWORD PTR DS: [40303C]	
00401470	. 3105 36144000	XOR DWORD PTR DS: [4014383, ERX	
08401462	. A1 40304000	MOU ERX, DWORD PTR DS: (403040]	
00401487	. 3105 3F144000	XOR DWORD PTR DS: 140143F1, ERC	
06401480	.V EB 06	JPP SHORT cracknel.00401495	
00401465	2 3105 07144000	XOR DWORD PTR DS: 14014071, ERX	and an and a second state and an and a second state
00481495	2 68 58384888	FUSH crackme1,00403058	D(1) G.LUARCA C GUSCHWET REARCHER
00401494	. 6H 10	PUSH 14	Cleve pore de la Mile - station
00401490	. 68 F4010000	PUSH IF4	5130 = 1F4 (544.2-
00401491	. 68 87144888	PUSH orackne1.00401407	Hodress = gracknel.00401407
00401660	. E8 27000000	CHEL COTP. EXERNEL.32. VIPtua (Protect)	VictualFrotec'
INDUCTIONS	. 58	POP ENX	

现在,第一件事情是把403038的值mov到eax,然后与401407的值进行XOR,然后把结果存放到401407的地 址中。等等,401407是我们上面设置了可写属性的,所以我们可以进行写操作,然而403038是重DEAD通过 按下不同的按钮后进行不同的计算后的值。因此,这一系列的指令是改变,基于什么按钮,并在其中的顺序,

他们被按下的内存空间等等。走到地址为401475的JNZ跳转然后让我们看看地址401407:

004019FF . C9 LEAUE 00401405 . S5 PUSH EBP 00401403 . S50 PUSH EBP 00401404 . S80 PUSH EBP 00401405 . S80 PUSH EBP 00401406 . S80 PUSH EBP 00401405 . S80 PUSH EBP 00401405 . S80 PUSH EBP 00401406 . S80 PUSH EBP 00401405 . S80 DE 00 00401406 . S80 DE 02 00401407 . S80 DB 02 00401412 . S80 DE 02 00401412 . S8 DE 02 00401412 . CHAR 'S' 00401425 . C2 0488 00401426 . C2 0488 00401427 . C2 0488 <th></th> <th></th> <th></th> <th></th> <th></th>					
00401403 5 C2 0400 RET 4 00401403 555 PUSH EBP 00401404 500 PUSH ERK 00401407 900 NOP 00401407 900 NOP 00401408 42 DB 42 00401407 900 DB 62 00401412 01 DB 62 00401413 89 DB 62 00401414 62 DB 88 00401413 89 DB 89 00401414 62 DB 88 00401415 83 DB 92 00401416 01 DEC EDI 00401417 95 DEC EDI 00401418 01 DE 95 00401425 02 DE 95 00401425 02 DE 95 00401425 02 DB 95 <th>884813FF</th> <th></th> <th>9</th> <th>LEQUE</th> <th></th>	884813FF		9	LEQUE	
00401403 * C2 0400 PUSH EBP 00401403 * S5 PUSH EBP 00401407 * E3 DC PUSH EAX 00401407 #2 DB #2 CHAR 'B' 00401417 #36 DB #2 CHAR 'B' 00401417 #39 DB #2 CHAR 'S' 00401415 #33 #2 B8 CHAR 'S' 00401415 #33 #2 #2 CHAR 'S' 00401415 #33 #2 #2 #4 00401416 C2 #48 B4 #2 EB 00401410 #15 528990283 PDC EDR </th <th>00401400</th> <th>1 6</th> <th>2 0400</th> <th>DE TN 4</th> <th></th>	00401400	1 6	2 0400	DE TN 4	
004401403 \$ 55 PUSH EBP 004401406 590 HOU EBP, ESP 004401406 590 PUSH EAX 004401406 590 NOP 004401406 590 NOP 004401407 900 NOP 004401408 42 DB 42 004401406 38 DB 42 004401407 38 DB 42 004401407 38 DB 42 004401407 82 DB 42 004401412 01 DB 90 004401413 89 DB 90 004401414 62 DB 92 004401414 82 DB 93 004401414 92 DB 93 004401418 91 DE 01 004401419 02 DB 93 004401418 91 DE 01 004401418 91 DE 01 004401422 C2 0488 RETM 8804 004401422 02 DB 93 004401422 02 DB 93 004401423 17 DB 10	00401400		2 0400	BEIN T	
00401404 . 88EC MOU EBP, ESP 00401406 . 50 PUSH EAX 00401407 .* E3 DC UECX2 SHORT crackme1.004013E5 00401407 .* E3 DC UECX2 SHORT crackme1.004013E5 00401407 .* E3 DC UECX2 SHORT crackme1.004013E5 00401408 42 DB 42 CHAR 'B' 00401407 88 DB 90 NOP 00401407 88 DB 90 CHAR 'B' 00401407 88 DB 90 CHAR 'B' 00401410 . 35 90 43 98 PSCII "SJC", 0 CHAR 'B' 00401412 91 DB 92 DB 92 CHAR 'S' 00401414 92 DB 92 DB 92 CHAR 'S' 00401416 . C2 9488 RETN 9804 CHAR 'S' 00401410 .15 528990283 PCC EDI CHAR 'S' 00401412 .02 DB 92 CEC EDI 09401425 02 DB 92 CHAR 'S' 09401426 35 DB 92 CHAR 'S' 09401426 35 DB 92 CHAR 'S' 09401426 35 <t< th=""><th>60401403</th><th>\$ 5</th><th>5</th><th>PUSH EBP</th><th></th></t<>	60401403	\$ 5	5	PUSH EBP	
00401406 • 500 PUSH ERX 00401407 •* E3 DC JECN2 SHORT crackme1.004013E5 00401407 90 NOP 00401408 42 DB 42 00401400 82 DB 42 00401400 82 DB 42 00401400 82 DB 82 00401412 01 DB 92 00401412 01 DB 92 00401413 89 DB 82 00401414 92 DB 82 00401415 83 DB 82 00401416 .CHAR 'S' 00401417 92 00401418 91 00401418 01 00401416 .CHAR 'S' 00401417 45 00401418 01 00401416 .CHAR 'S' 00401416 .CHAR 'S' 00401417 46 00401426 35 DB 92 00401426 35 DB 92 00401426 35 DB 93 00441427 02 DB 93 00441428 00	00401404		REC	MOU ERP ESP	
00401405 . 50 DC UEXX SHORT crackme1.004013E5 00401407 .* E3 DC UEX SHORT crackme1.004013E5 00401407 .* E3 DC UEX SHORT crackme1.004013E5 00401407 .* S3 0D 43 00 NOP 00401410 S5 0D 43 00 RSCII "5/C",0 00401412 01 DB 02 00401414 02 DB 02 00401416 C2 0488 RETN 8504 00401418 01 DB 02 00401418 01 DB 02 00401418 01 DB 02 00401410 .15 52890283 ADC ERX, 83028952 00401425 02 DB 02 00401425 02 D	00101101		DEC	THOSE LOFF LOF	
00401407 .* E3 DC JECX2 SHORT crackme1.004013E5 00401407 90 HOP DE 42 CHAR 'B' 00401408 42 DE 42 CHAR 'B' 00401400 88 DE 98 CHAR 'B' 00401400 88 DE 98 CHAR 'B' 00401412 01 DE 90 CHAR 'B' 00401413 89 DE 98 CHAR 'S' 00401414 92 DE 92 CHAR 'S' 00401415 83 OE 89 CHAR 'S' 00401416 . C2 0488 CE 20 00401416 . C2 0488 CHAR 'S' 00401416 . C2 0488 CHAR 'S' 00401412 . C2 0488 CHAR 'S' 00401420 . CHAR 'S' CHAR 'S' 00401420 . C2 0488 CHAR 'S' 00401420 . C2 0488 CHAR 'S' 00401420 . DE 00 CHAR 'S' 00401420 . DE 00 CHAR 'S' 00401420 . DE 00 <th>00401406</th> <th>. 5</th> <th>0</th> <th>PUSH EHX</th> <th></th>	00401406	. 5	0	PUSH EHX	
00401409 90 HOP 00401408 42 D8 42 D8 42 00401408 88 D8 98 D8 98 00401408 82 D8 98 D8 98 00401408 82 D8 99 D8 92 00401413 99 D8 92 D8 92 00401413 99 D8 92 00401414 83 D8 92 00401415 83 D8 92 00401416 C2 0488 D8 92 00401417 92 D8 92 00401418 01 D8 01 00401419 C2 0488 CHR *5* 00401410 C2 0488 CHR *5* 00401418 01 D8 01 00401426 62 D8 92 00401427 02 D8 92 00401426 35 D8 92 00401427 92 D8 92 00401428 90 D8 92 00401429 10 D8 10 00401429 10 D8 10 00401428 89 D8 92 0040142	00401407	. + E	53 DC	JECKZ SHORT crackme1,004013E5	
00401407 90 H0P 00401408 42 DE 42 CHAR 'B' 00401400 88 DE 82 DE 82 00401400 02 00 43 00 ASCII "SJC",0 CHAR 'B' 00401412 01 02 08 89 08 89 08 89 00401412 01 02 08 89 08 89 08 89 00401414 02 02 89 08 89 08 89 08 89 00401415 02 04 89 08 99 08 99 08 99 00401416 C2 0488 08 92 08 92 08 99 08 99 00401416 C2 0488 08 95 CHAR '5' 08 91 00401425 02 9488 08 92 08 92 08 93 00401425 02 0488 08 92 08 93 08 93 00401426 35 08 93 08 93 08 93 00401426 35 08 93 08 93 08 93 00401426 35 08 93 08 93 08 93 00401426 35 08 93 08 93 08 93 <th>00401400</th> <th></th> <th>0</th> <th>HOD</th> <th></th>	00401400		0	HOD	
0240140R 42 DB 42 CHAR 'B' 0240140E 88 DB 42 DB 42 0240140E 35 0D 43 00 ASCII "SJC",0 02401412 01 DE 89 DB 82 02401412 01 DE 89 02401413 39 DE 89 02401414 02 DE 89 02401415 83 DE 89 02401414 02 DE 89 02401415 83 DE 89 02401416 . C2 0488 RETN 8804 02401417 . CHAR 'S' 02401418 01 DE 92 02401410 . 15 52890283 ADC ERX, 83028952 02401420 . C2 0488 RETN 8804 02401426 02 DE 96 02401427 02 DE 96 02401428 10 DE 10 <td< th=""><th>00401402</th><th>2</th><th>10</th><th>110</th><th></th></td<>	00401402	2	10	110	
08481400 08481400 08481400 08481400 08481400 08481412 42 08481400 08 09481412 DB 88 0B 82 09 09481412 DB 88 0B 82 08 83 08	0040140R	9	10	NOP	
0040140C 88 DB 98 DB 98 DB 97 0040140E 35 0D 43 00 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401412 01 DB 99 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401413 99 DB 99 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401415 92 DB 99 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401416 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401416 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401416 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401416 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401416 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 00401422 .C2 0488 PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 004401426 .C488 PS PSCII "5/C",0 PSCII "5/C",0 PSCII "5/C",0 004401426 .C488 <	0040140R	4	12	DB 42	CHOR 'B'
0040140C 38 02 08	00401400		6	00.00	Crimina D
00440140E .35 0D 43 00 PSCII "5/C",0 004401412 01 0B 02 0B 03 004401412 01 0B 02 0B 03 004401412 01 0B 02 0B 03 004401414 02 0B 03 08 03 004401415 83 0B 0B 02 0B 03 004401415 .C2 0488 RETN 9804 04 04 04 004401416 .C2 0488 RETN 9804 04 04 04 04 05 05 04	00401400		D .	PD 00	
0040140E . 35 00 43 00 ASCII "5/C",0 00401412 01 DB 01 00401414 02 DB 02 00401415 83 DB 02 00401416 . C2 0488 RETN 9804 00401416 . C2 0488 DB 02 00401416 . C2 0488 DE 02 00401416 . C2 0488 DE 02 00401416 . C2 0488 DE 02 00401410 . 15 52890283 ADC EAX, 83028952 00401420 . C2 0488 RETN 9804 00401425 . 02 0488 004401426 . S5 DB 02 004401426 . CHAR '5' 004401426 . CHAR '5' 004401426 . CHAR '5' 004401426 . C2 0488 004401428 . 00 004401429 . C2 0488 004401420 . 08 02 004401420 . 02 004401420 . 02 004401429 . 02 004401429 . 02 004401420 . 02 004401420 . 02	00401400	6	32	08 82	
00401412 01 08 01 08 01 08 01 00401412 89 08 89 08 89 00401414 62 08 83 00401415 83 08 02 00401416 C2 0488 08 02 00401416 C2 0488 08 02 00401416 C2 0488 08 02 00401410 45 52890283 00401410 15 52890283 ADC ERX, 83028952 00401425 02 0488 00401426 35 D8 02 00401427 08 02 D8 02 00401428 09 D8 02 00401429 17 D8 17 00401420 83 02 00401421 02 08 02 00401422 02 08 02 00401428 02 </th <th>6040140F</th> <th></th> <th>5 00 43 00</th> <th>DSC11 **51C** 0</th> <th></th>	6040140F		5 00 43 00	DSC11 **51C** 0	
004014122 01 08 01 00401413 02 08 02 00401415 83 08 03 00401416 . C2 0488 RETN 3804 00401419 02 08 02 00401419 02 08 02 00401410 . C2 0488 RETN 3804 00401410 . 4F 0E 01 00401422 . C2 0488 RETN 3804 00401425 02 08 02 00401426 35 0B 02 00401426 36 0B 02 00401426 36 0B 02 00401426 36 0B 02 00401426 36 0B 02 00401428 0D 0B 02 00401429 17 0B 17 00401420 83 0B 93 00401420 82 0B 93 00401420 82 0B 93 00401420 83 0B 93 00401421 10 0B 17 00401422 02 0B 93 00401431 92 0B 93 00401432 <th>00101110</th> <th></th> <th>00 40 00</th> <th></th> <th></th>	00101110		00 40 00		
00401413 89 0B 89 0B 89 0B 89 00401415 83 0B 82 0B 82 0B 82 00401415 . C2 8488 RETN 8604 CHAR '5' 00401416 . C2 8488 RETN 8604 CHAR '5' 00401416 . C2 8488 0B 92 CHAR '5' 00401410 . 15 52890283 ADC EAX, 83028952 CHAR '5' 00401422 . C2 9488 0B 92 0B 92 00401422 . C2 9488 0B 92 0B 92 00401426 95 0B 95 0B 95 00401426 95 0B 96 0B 96 00401426 95 0B 96 0B 96 00401427 0E 0B 97 0B 17 00401429 17 0B 17 0B 17 00401429 18 0B 93 00401420 82 0B 92 00401421 89 0B 93 00401422 02 0B 93 00401421 82 0B 93 00401421 83 0B 93 00401421 64 0B 93	00401415	6	91	DB 81	
00401414 02 DB 02 DB 02 DB 03 00401415 83 DB 02 DB 03 CHAR '5' 00401416 . C2 0488 RETN 9804 CHAR '5' 00401418 01 DB 01 CHAR '5' 00401410 . 4F DEC EDI CHAR '5' 00401426 . C2 0488 RETN 9804 CHAR '5' 00401426 . C2 0488 RETN 9804 CHAR '5' 00401426 . C2 0488 RETN 9804 CHAR '5' 00401426 . C2 0488 DE 02 CHAR '5' 00401426 . C2 0488 DE 02 CHAR '5' 00401426 . C2 0488 DE 02 CHAR '5' 00401428 . CD . CHAR '5' . CHAR '5' 00401429 . CE . DE 02 . CHAR '5' 00401429 . CE . DE 02	00401413	8	39	DB 89	
00401111 85 005 83 00401415 . C2 0488 08 02 00401416 . C2 0488 08 02 00401417 92 08 02 00401418 01 08 03 00401410 . 4F 0B 01 00401410 . 15 52890283 ADC EAX, 83028952 00401425 02 0488 00401426 35 0B 02 00401426 02 08 02 00401427 02 08 02 00401426 35 0B 02 00401427 0E 0B 02 00401428 0D 0B 02 00401429 17 0B 17 00401429 17 0B 12 00401429 17 0B 18 00401429 17 0B 93 00401420 83 0B 92 00401421 82 0B 93 00401422 02 08 92 00401431 02 08 93 00401431 92 08 93 00401431 92 08 93 00401432	00401414	e e	12	DD DO	
00401416 . C2 0488 RETN 9804 00401416 . C2 0488 RETN 9804 CHAR '5' 00401418 01 08 95 CHAR '5' 00401418 01 08 02 CHAR '5' 00401410 . 15 52890283 ADC ERX, 63028952 CHAR '5' 00401426 02 08 02 CHAR '5' 00401428 00 08 00 08 00 00401429 17 08 17 08 17 00401420 83 08 93 08 93 00401420 83 08 93 08 93 00401420 83 08 93 08 93 00401431 02 08 93 08 93 00401432 05 08 93 08 93	0010111		-	00 00	
00401416 . C2 0488 RETN 9804 00401419 02 08 02 08 02 00401418 35 08 01 08 01 01 00401418 01 02 01 02 01 <	00401415	8	53	UB B3	
00401419 02 08 02 08 02 08 03 00401418 01 0B 01 0B 01 0B 01 00401410 . 4F 0EC EDI 0EC EAX, 83028952 0E 02 00401422 . C2 0488 0E 02 0E 02 0E 02 00401426 35 0E 02 0E 02 0E 02 00401426 35 0B 02 0E 02 0E 02 00401426 35 0B 05 0E 02 0E 02 00401426 35 0B 02 0E 02 0E 02 00401426 35 0B 02 0E 00 0E 02 00401427 0E 0B 02 0E 00 0E 00 00401429 17 0B 17 0B 17 0E 02 00401420 83 0E 93 0E 93 0E 93 00401420 83 0E 93 0E 93 0E 93 00401421 02 0E 93 0E 93 0E 93 00401428 35 0B 93 0E 93 0E 93 00401431 02 0E 93 0E 93 0E 93 00401432	00401416	. 0	2 0488	RETN 8604	
00481418 35 08 35 08 35 CHAR '5' 00401418 01 02 01 02 01 02 01 02 01 00401410 4F 0EC EDI 0EC EDI 02 01 02 01 00401425 02 02 0488 0E 02 02 02 02 01 00401426 35 0E 02 0E 02 0E 02 02 01 00401426 35 0E 02 0E 02 0E 02 0E 02 00401426 35 0E 02 0E 02 0E 02 0E 02 00401426 35 0E 02 0E 02 0E 02 0E 02 00401428 00 0E 02 0E 02 0E 02 0E 02 00401429 17 0E 17 0E 17 0E 17 0E 18 0E 02 00401429 17 0B 18 0E 02 0E 02 0E 03 0E 03 00401420 83 0B 93 0B 93 0E 93 0E 93 0E 93 00401431 02 0E 02 0E 02 0E 93 0E 93 0E 93 00401432 35 0E 93	00401419	0	12	08.02	
0040141H 36 UB 35 UH 75* 0040141B 01 UB 01 UB 01 0040141C .4F DEC EDI 06401410 00401421 .C2 0488 ADC EAX, 83028952 06401426 00401426 02 08 02 08 02 00401426 05 0B 02 08 02 00401426 05 0B 02 0B 02 00401426 05 0B 02 0B 02 00401426 05 0B 02 0B 02 00401428 00 0B 02 0B 02 00401429 17 0B 17 0B 17 00401420 02 0B 99 0B 99 00401420 02 0B 93 00401420 02 0B 93 00401421 02 0B 93 00401422 02 0B 93 00401421 02 04 93 00401421 33 0B 93 00401431 02 08 95 00401432 35 0B 95 00401433 16 08 95 00401433	00101110			50 0C	CUOD 151
00401418 01 DE 01 00401410 .4F DEC EDI 00401410 .15 52890283 ADC EAX, 83028952 00401425 02 00401426 02 00401426 02 00401426 02 00401427 0E 00401428 00 00401429 17 00401429 17 00401428 00 00401429 17 00401428 00 00401429 17 00401429 17 00401429 18 00401429 17 00401428 89 00401429 16 00401429 17 00401429 18 00401429 16 00401420 83 00401421 82 00401431 02 00401431 02 00401432 35 00401433 16 00401433 16 00401433 16 00401433 16 00401433 16 00401433 16 00401433 16 00401433 16 00401433 16	0040141H	- 4	55	LIB 35	CHHK 5
0040141C . 4F DEC EDI 0040141C . 15 52890283 ADC EAX, 83028952 00401422 . C2 0488 RETH 9804 00401426 35 DE 35 CHAR '5' 00401426 35 DE 35 CHAR '5' 00401426 35 DE 35 CHAR '5' 00401426 00 DB 0E 00 00401429 17 DB 17 08 17 00401429 17 DB 18 09 00401420 89 DB 99 00 00401420 02 DB 93 08 00401420 83 DB 99 00 00401420 83 DB 93 00 00401420 83 DB 93 08 00401421 82 DB 95 08 00401431 02 DB 95 08 00401432 35 DB 95 08 00401432 35 DB 95 08 00401432 35 DB 95 08 00401433 16 08 02 00401433	0040141B	6	31	DB 01.	
06401410 . 15 52890283 ADC EAX, 63028952 06401422 . C2 0488 RETH 8804 06401426 02 DB 02 08 02 06401428 00 DB 02 08 00 06401429 17 DB 17 08 10 06401428 89 DB 03 08 03 06401420 83 DB 03 09 02 06401420 83 DB 03 09 02 06401431 02 DB 03 08 02 06401431 02 DB 03 08 02 06401431 02 DB 03 08 02 06401431 02 DB 03 04 02 06401432 35 D8 35 CHPR '5' 08 0401433 04 02 04 02 08 0401433 04 02 04 02 08 0401433 05 08 02	00401410	. 4	F	DEC EDI	
00401412 . 15 52670255 HOC EHA, 6026752 00401422 . C2 0488 AETH 8804 00401426 02 08 02 00401426 35 DB 05 00401426 35 DB 06 00401427 0E DB 06 00401428 0D DB 06 00401429 17 DB 17 00401429 17 DB 18 00401420 02 DB 09 00401420 02 DB 09 00401420 02 DB 02 00401420 02 DB 02 00401421 02 DB 09 00401422 02 DB 02 00401421 02 DB 02 00401421 02 DB 03 00401431 02 DB 05 00401432 35 DB 35 00401432 35 DB 35 00401432 35 DB 16 00401433 16 DB 16 00401433 05 08 02 00401433 16 08 02 <th>00101110</th> <th></th> <th>E E0000000</th> <th>ODC FOY DOGDOOD</th> <th></th>	00101110		E E0000000	ODC FOY DOGDOOD	
00401422 . C2 0488 RETH 8804 00401426 02 DB 02 00401426 35 DB 35 00401426 35 DB 0E 00401426 00 DB 0E 00401428 00 DB 0E 00401429 17 DB 17 00401429 10 DB 17 00401428 89 DB 99 00401429 10 DB 18 00401420 02 DB 99 00401420 02 DB 93 00401420 02 DB 93 00401420 02 DB 93 00401420 02 DB 93 00401431 92 DB 93 00401431 92 DB 95 00401432 35 DB 95 00401433 16 DB 16 00401433 16 DB 16 00401433 16 DB 16	00401410		5 52070203	HUC EHA, 00020952	
00401426 02 08 02 08 02 08 02 08 02 00401426 35 08 35 08 35 08 02 08	00401422	. 0	2 0488	RETM 8B04	
00401426 35 0B 35 CHAR '5' 00401426 0E 0B 0E 0B 0E 00401428 0D 0B 0E 0B 0E 00401429 17 0B 10 0B 10 00401428 89 0B 99 0B 99 00401420 02 0B 02 08 02 00401420 83 0B 33 0B 33 00401421 02 0B 02 08 02 00401431 02 08 02 08 02 00401432 35 0B 35 CHAR '5' 00401431 02 08 02 08 02 00401432 35 08 35 08 35 00401432 35 08 35 08 35	80491425	0	12	DB 02	
00401420 00 00 00 00 00 00 00401427 0E DB 0E DB 0E 00 0E	301101000		C.	00.00	CHOD ICI
B3481427 DE DE <thde< th=""> DE DE <t< th=""><th>00401420</th><th>9</th><th>0</th><th>00.00</th><th>CHAR D</th></t<></thde<>	00401420	9	0	00.00	CHAR D
00401428 00 08 00 08 00 00401429 17 08 17 08 17 00401429 10 08 10 08 10 00401428 89 08 89 08 99 00401420 02 08 02 08 02 00401420 83 08 33 08 36 00401431 02 08 02 08 02 00401431 02 08 02 08 02 00401431 02 08 02 08 02 00401432 35 08 95 CHPR '5' 00401432 36 08 16 08 16 002011 "55" 08 95 02 16	88481427	6	1E	DB BE	
B0401429 17 DB 17 00401429 10 DB 10 00401428 10 DB 10 00401428 89 DB 99 00401420 02 DB 02 00401420 83 DB 93 00401431 02 DB 02 00401431 02 DB 95 00401432 35 DB 95 00401433 16 DB 16 00401433 45 45 00 02011 "EE" 0	00401428	P	D.	DR 60	
004014220 10 00 11 004014221 10 00 10 004014228 89 00 89 00401420 02 08 02 00401420 83 08 33 00401421 83 08 02 00401431 02 08 02 00401432 35 08 95 00401433 16 08 16 00401433 16 08 16	00401400	4	2	DB 12	
00401428 10 08 16 00401428 89 08 99 00401420 92 08 92 00401420 83 08 93 00401421 62 08 92 00401422 62 08 92 00401431 92 08 95 00401432 35 08 95 00401433 16 08 16 008 163 08 16 08 16	00401425			00 10	
0040142E 89 DB 99 0040142C 02 DB 02 0040142C 02 DB 02 0040142D 83 DB 93 0040142E . C2 048B RETN 9804 00401431 02 DB 02 0802 00401432 35 DB 95 CHPR '5' 00401432 35 DB 16 DB 16 00401434 45 45 08 02 (CTT) 16 08	0040142H	1	.0	DB 16	
0040142C 02 DB 02 0040142D 83 DB 83 0040142E C2 048B REIN 8804 00401431 02 DB 05 00401432 35 DB 95 00401433 16 DB 16 00401434 45 45 00 02011 "EE" 0	0040142B	8	39	08.89	
0040142D 83 DB 83 0040142D 83 DB 83 0040142E . C2 048B RETN 8804 00401431 02 DB 02 DB 16 00401433 16 DB 16 DB 15 00401434 45 45 00 05°CTT 15°E** 0	00401420	0	12	08 02	
0040142D 83 0040142E C2 0488 RETN 8804 00401431 02 06 02 08 02 08 02 00401432 35 08 95 CHPR '5' 00401433 16 08 16 08 16 00401433 45 45 00 05011 "55" 0	00101400		10	00 00	
0040142E C2 0488 RETN 8804 00401431 02 D8 02 00401432 35 D8 35 00401433 16 D8 15 00401434 45 45 00 02011 "EE" 0	00401420	8	53	DB 23	
00401431 02 08 02 04 01432 05 04 01432 05 06 02 04 01432 05 06 02 04 01432 05 06 02 04 01432 05 04 01432 05 06 02 <th< th=""><th>0040142E</th><th>. 0</th><th>2 048B</th><th>RETN 8804</th><th></th></th<>	0040142E	. 0	2 048B	RETN 8804	
00401432 35 DB 35 CH98 '5' 00401433 16 DB 16 DB 16 00401434 45 45 99 OSCIT "EE" 9	88481431	0	12	08.02	Loc The Ross L
00401452 35 UB 55 00401433 16 DB 16 00401434 45 45 99 OS TT "EE" 9	00101400		(F	00.05	C1100 1C1
00401433 16 DB 16 00401494 45 45 00 00011 "FE" 0	00401452	-	00	06 05	CUBRIS . P.
BRARIARA 45 45 88 DOCT "FE" R	00401433	1	6	DB 16	OLD THE REAL OF
	00401424	6	15 45 00	OSCIT "EE" D	

(译者注:我的OD里面显示的和他不一样,我的是jo,而作者的是JECXZ,我百度了下解释为:JECXZ(ECX 为 0则跳转);jo则是溢出跳转

00401400	L.	C2 0400	retn Øx4
00401403	\$	55	ebp
00401404	- 41	8BEC	mov ebp,esp
00401406	- 41	50	eax
00401407	-	70 70	jo Xcrackme1.00401485.要破解论法
00401409	-	34 0C	xor al, 0xC
0040140B		42	inc edx www.52pojie.cn

跳转相关的标志位:

11	10	9	8	7	6	5	4	3	2	1	0
OF	DF	IF	TF	SF	ZF		AF		PF		CF
溢				符	卖	未	辅	1	奇	4部	进
出				묵	.4.	用	助	题	偶	屈	位

)你可以看到401407的内存数据已经被改变了,但是这里有个没见过的指令: JECXZ SHORT

Crackme1.004013E5.程序加了一个条件跳转到自己的代码! (译者注:没明白什么意思) 这方法是在内存位置修改操作码或者裸数据。现在回到我们的指令位置,下一个是用401407和0x52进行比较,如果不等则跳转到40148F。看上面的图片,我们可以看到401407处操作码是"E3"是不等于0x52的,因此我们跳走。这个跳转就是恢复之前VirtualProtect,把之前可写的内存恢复到只能执行不能写的状态:



but before this you may have noticed thatmemory location 401407 was XOR'ed again at address 40148F.

Looking again ataddress 401407 we see that it was changed again:在此之前你应该注意到了,内存401407 和eax (40148F)做了Xor操作.再去看看401407你会发现数据有被修改了。

88481488	υ;	C2 0400	RETN 4	
00401403	- 5	55	PUSH EBP	
00401404		SBEC	MOU EBP, ESP	and the second sec
00401406		50	PUSH EAX	
00401407	.~	EB SF	JMP SHORT crackme1.00401448	
88481489		90	NOP	
88481488		36	NUP	0100 101
6646146B		42	00 42	CHHR . B.
00401400		85	00 00	
00401400		2E 00 49 00	OCCAT HEACH O	
00401402	•	35 60 45 66	NOC11 5/C .0	
00401412		00	00.01	
00401414		82	08 82	
88491415		83	DB 83	
00401416		C2 8488	RETN 8804	
00401419		02	DB 92	and the second second
8840141A		35	DB 35	CHAR '5'
0040141B		01	DB 01	
00401410		4F	DEC EDI	法法 网络白豆属
0040141D		15 52890283	ADC EAX, 83828952	and the second second
00401422		C2 0488	RETN 8884	1 Y 1 Y 1 Y 1 Y 1 Y 1 Y 1 Y 1 Y 1 Y 1 Y
00401425		02	DB 82	Latting and

现在这里的JECXZ已经被JMP跳转替代了。事实上,程序改变这个内存两次,一次是JECXZ,另一次则是JMP(译者注:这里我把JECXZ替换为jo)。然后我们回到主窗口循环中:

00401163 004013F2 004013F2 004011F5 004011F8 00401178	C 2765 44304400 00 HOU 00000 PTR D5: [4030441, 0 9 0330 48304400 03 DHP 000R0 PTR D5: [4030441, 0 7 73 20 045 54007 crackwel, 00401228 9 0330 44304000 04 CHP 000R0 PTR D5: [4030441, 0 7 75 30 045 54007 crackwel, 00401442 56 43020000 045 54007 crackwel, 00401442 56 43020000 045 54007 crackwel, 00401442	OSTIT "Do arrow occured"
01401205 01401215 01401215 01401225 01401225 01401225 01401225 01401225 01401225 01401225 01401225 01401225 01401225 01401225 01401255 01401550 01401555 01401555 014015550 014015550 014015550 014015550 014015550 014015550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005550 014005500 014005500 01400550000000000	FF75 B8 PCER FF75 B9 C705 443544000 D04 MOAD D0401403 C705 443544000 D04 MOAD D0401403 FF75 48304000 DMF SAUCH CAUCH D0401403 FF81 00 DMF SAUCH D1600 PTR D5114030401 FF85 00 DMF SAUCH CAUCH D0401403 FF85 01 DMF SAUCH CAUCH D0401403 FF75 08 PUSH CAUCH D0403241 D1405 FF75 08 PUSH CAUCH D0403243 D1405 FF75 08 PUSH CAUCH D1405 D1405 D1405 C705 44304000 004 D0000 D1405 D1405	Time = "Trying to bruteforce?" Controll0 = 3 News = 000F08E2 ('Crachest2 by Derict', elayer'ss SetDigitenTestA

我们放了一个值(F08E2(译者注:这个值与你按下的按键不同会不同,我的与作者的就不一样))到栈然后 Call另外一个常规地址401403.步进这个Call我们看见了这个功能:

88481488	L.	C2 0400	RETH 4	
00401403	5	55	PUSH EBP	
00401404		8BEC	MOU EBP, ESP	
88481486		50	PUSH ERX	
88481487	.~	EB 3F	JMP SHORT crackme1.00401448	
00401489		90	NOP	
88481488		38	NOP	CHOD 101
00401405		42	00 00	CHHK .B.
00401400		00	00 00	Let have a
00401400	1.5	35 00 42 00	DECTT "EAC" D	and the second second
88481412	•	81	08.01	And a state of the state
00401413		89	DB .89	

现在我们来到了程序修改内存的地方。我们之前就知道401407这里的这个跳转了,让我来到JMP看看会把我们带到那里去。

PRANTARE		50	PUSH FOY	
COMPRESSION IN	1000	ER SE	HP SHORT crack pet, 00401449	
004014001		60		
00401400		90	HOD	
00401400		10	100	cuico tot
00401400		92	UB 42	CHARGE, B.
00401400		RR	DE SE	
0040140D		82	08.02	
8040140E		35 9D 43 90	ASCII "57C", 8	
00401415	1.1	01	QB 01	
00401413		89	DB: 09	
69401414		82	DB 82	
00401415		83	0B 63	
00401416		C2 0488	RETN BROW	
00401419		82	DR 02	
00401410		26	00.00	CHOR .E.
00401310		01	00 01	Custo 5
00401410		AF	DEC EDI	
00401410		IF FOODOOD	DEC EDI	
00401410		15 52670283	HUC EHX, BIRECOMOS	
86401422		L2 0488	HOLIN SERVE	
00401425		82	DB B5	billion and
00401426		35	DB 35	CHAR 15'
00401427		6E	DB DE	
00401428		0D	DB 00	
08401429		17	0B 17	
08481438		10	D6 10	
00401426		89	DB- 89	
08401420		82	DR R2	
00401420		83	DR 83	
0040145E		C2 D4PR	RETH SSINE	
00401401		82	00.90	
00401422		SE	TIP OF	YUOD TET
00401472		16	ND 92	August 5
00401435		10 45 00	CODEL HERE O	
00401424		45 45 00	HOGII EE . O	
00401437		69	DB- 83	
00401438		82	UB 02	
00401439		5H 58	ASCII "ZW"	
80481488		0466E7BB	DD BBE76604	
004014DF		4DBD088B	00 68686040	a second of the second s
00401443		E8 78000000	CALL (UMP.&USER32.SetDigItenTextR)	SetDigItenText8
69491448	2	+C9	LEAVE	
00401449		C2 8988	RETH 8	
00401440	11	50	PUSH ERX	and the second se
09401440		68 59394999	PUSH on acking 1, 00403050	Collident and a macked 09499950
00401452	1.1	69 84	PUSH	HeyProtent - Rom Standers
00401454	1.	69 E4010000	PUSH IFA	FIRE - 1EA TERA I
00404400		40 07144000	DUCH prostingt 00401407	Distants - exection 1, 00401407
basarder	1.	E9 (E000000	TOUL C HD ANEDNEL 22 III anu al Destroyal	Ulternal Fred the
00401460		01 20204000	HOLL FOY DUODO DTD DC. FA00000	evillagered.et.
00401469		2105 07144000	VOD DHODD DTD DS+14014021 EDV	

奇怪,他直接跳到了返回。所以看来这并没有真正做什么。我们现在回到主程序:

00401209	E8 43020000	AL GROUP CLACKER 1.00401241	ASCII "An error occured"
00401209	68 00304000	PUSH Crackwel.00403000	
00401205	FF75 08	USH EM90.11	
00401211	E8 ED010000	PUSH CARACKWEL.00403000	
100401226 00401226 00401226 00401226 00401227 00401237 00401237 00401237 00401241 00401241 00401248 00401347 00401347 00401347	. Cr05 44394000 000 . FF05 44394000 . EB 19 . 69 11394000 . 69 03 . FF75 08 . EB 99020000 . C705 44304000 000 . EB 09 . B8 00000000 . C2 1000 . C2 1000 . EB 09 . C2 10000000 . C2 10000000 . C2 10000000	RUC DURUPD FIN DS1 E44304431 RIP SHORT Crackwei.004030431 RIP SHORT Crackwei.00403011 USH Crackwei.00403011 USH Cracks.10 CRLL (.RPS.10 CRLL (.RPS.10) CRLL (.RPS.10) CRLL (.RPS.10) CRLL (.RPS.10) CRLL (.RPS.10) CRL (.RPS	Text = "Trying to bruteforce?" ContruitD = 3 Not - 000F00E2 ('Crackmesi2 by Detten', SetDisitemText0

接下来是把计数器置零。然后把另一个计数器加1.现在我们明白暴力破解信息是怎么检查的了:如果我们输入 10个密码错误超过3次,403048计数器就大于3就跳到暴力破解信息。如果想再试请继续。保存401204的断点 存在,点击10个按钮:

1	2	3	A	В	C
4	5	6	D	E	F
7	8	9	Clea	ar A	bout

出现了暴力破解的信息。现在,确保401204的断点存在然后重启程序。以确保计数器被清零。

因此我们知道程序更多的信息:1、密码为10位数。(译者注:早前猜测)2)如果连续三次错误,则会出现暴力破解信息,出发重新启动程序。3)每一次按下按钮403038,40303C和403040内存中的数据都会被不同的

算法修改。每一个按钮的算法都不一样。4) 直到我们按了10次按钮后来到一个Call,进入后发现它会检查代码 和改变跳转指令,在地址401407处。5) 如果密码不对,这个跳转就会生成并且就返回到我们的主窗口循环 中。6)因此,输入正确的密码,必须改变这跳转到别的东西,或者跳到不同的内存位置,来到我们想去的地 方,或者在这个内存的这个段修改代码以创造好消息,而不是创建一个跳转。这听起来似乎更合理,因为它只 是简单地变成了一个跳转到一个新的位置,这个奇怪的,非功能性的代码是什么?知道了这一切,我们知道我 们必须要在程序自修改前清空这个段,即从地址40144c。让我们再来看看这段



我们可以确认一件事就是在40146e处与0x52的比较是重要的。它基本上告诉我们,程序已作出的代码的变化 是正确的变化。但是0x52操作码是什么意思呢?好吧我们百度下(作者使用Google),我们发现操作码0x52 是"PUSH EDX",如果不是呢,这就是Bug。如果我们强制设置这个指令为"PUSH EDX"看看会发生什么?试试 看吧,在40146E处设置断点。然后来到401407处,并且修改操作码为0x52:

00401400 00401403 00401404 00401406 00401406 00401407 00401407	m	C2 0400 55 38EC 50 52 90	PUSH EBP MOV EBP, ESP PUSH EAX PUSH EXX	_	ntdil.KiFastSy	stenCallRet	
00401409 00401400 00401400 00401400 00401400 00401400 00401402 00401412 00401412		90 90 42 88 02 35 00 43 00 01 89	NOP DB 42 DB 86 DB 02 ASCII "SVC", 0 OP 01 DB 97	Assemble at 0	00401408		
00401414 00401415 00401415 00401419 00401419		82 83 C2 0488 02 35	D6/ 02 D6/ 83 RC 10/ 8804 D6/ 02 D6/ 35	Fill with N	0P's	Assemble	Cancel
00401418 00401410 00401410 00401422	:	01 4F 15 52890283 C2 0488	DEC EDI ADC ERX, ESE20952 RETN 2884	-			5?onije m

现在,单步走,我们能通过这个跳转

00401455 00401455 00401463 08401463	••••	E8 6F000000 A1 38304000 3105 07144000		CALL (JMP. &KERNEL32.UirtualPro MOV EAX, DWORD PTR DS:[403038] XOR DWORD PTR DS:[401407], EAX	UirtualProtect
234104462		8030 07144000	5	CMP BYTE PTR DS: [401407], 52	
00401475	.4	75 18		JNZ SHORT crackme1.0040148F	
00401477		A1 3C304000	-	MOU EAX. DWORD PTR DS: [40303C]	
8040147C		3105 3B144000		XOR DWORD PTR DS: [401438]. EAX	
88481482	1.1	A1 40304000	-1	MOU ERX. DWORD PTR DS: [403040]	
00401487		3105 3F144000		XOR DWORD PTR DS: [40143F1, ERX	and the second s
00401480		EB 06	1	JMP SHORT crackme1,00401495	
8848148F	3	3105 07144000		XOR DWORD PTR DS: [401407], EAX	
00401495	5	68 50304000		PUSH crackme1.00403050	[DUIdFrotest = crack
001012001		60 10		DUCU 10	Maufinebase - DOCE I

现在把40303C的移动到eax,然后把40143B与eax进行Xor,这时什么地址?我们看看

004010FC		01	FUEHU
004012FF	1.1	C9	LEQUE
00401400	L.	C2 0400	DETTY 4
00401400		02 0400	Dela Trop
00481483		55	PUSH EBP
08401484	÷.	SBEC	MOU EBP.ESP
00401406		50	PUCH EDY
00401400		00	FOR ERA
00401407		52	PUSH EDX ntdll.KiFastSystemCaliRet
00401408		90	NOP
00401409		98	140P
00401400		66	100
00401469		30	100° c
00401400		42	DB 42 CHRK 'B'
0048140C		88	08.88
00401400		92	DB 02
00401400		00 00 10 00	CO OL HE LER D
0040140E		35 80 43 88	HSCII "SJUT.8
00401412		01	08 81
00401413		89	08.89
00401414		22	00.00
00401414		02	00.02
88481415		83	DB 83
88401416		C2 048B	RETH B804
00401419		02	DP 92
00401417		00	00 02
00401418		35	UB 35 UHHK 5
88481418		01	DB 81
8848141C		4F	DEC EDI
00401410		15 52000202	000 EDV 03030055
00401410		10 02070200	HOC CHALGODZOPOL
96461455		C2 048B	RE10, 5504
00401425		02	OB 82
00401426		25	DB 95 CHOR '5'
00401405		0E	DD OF
00401421		DE	UD OC
00401428		00	DB 60
00401429		17	08 17
00401420		10	DR 10
00401420		òà	00.00
00401420		07	00 07
00401420		02	08 82
00401420		83	DB 63
8848142E		C2 8488	RETH SEDA
00401431	•	02	10.02
00401401		02	
86481435		35	DB 35 CHAR 'S'
00401433		16	DB 16
00401494	120	45 45 00	OSCIT MEEN O
00404457		40 40 00	NO. SO
00401437		07	00 07
00401438		02	DB 82
00401439		5A 58	ASCII "ZW"
08481420		9466F788	DD 68676694
00401400		40000000	00 0000000
0040145		4060088B	DU 2BGCEDAD
00401443		E8 7800008	CALL KUMP. SUSER32. SetDigiteMTek SetDigiteMTektR
00401440	- 3	00	LCOUC

就像我们看到的一样,这就是我们被跳过的一个内存段的尾部。Xor后我们来到:

00401437	89	08 89	
00401438 .	02	0B 02	
00401439 .	5A 58	ASCII "ZX"	
00401435 .	408002730 40800888	00 88888040	The state of the second s
00401443	E8 78000000	CALL <ump.&user32.setdigitentex< td=""><td>SetDigitententh</td></ump.&user32.setdigitentex<>	SetDigitententh
00401448 >	C9	LEAVE	and a more than the
00401449	C2 0800	REIN S	

我们改变了40143B的值,下一步又修改了40143F的值。



这些位置并没有被改变成正确的代码,所以它并没有真正帮助我们,但看到这是应用程序改变的最后一件事, 这是重要的。现在继续走,我们已经改了 PUSH EDX了,让我们看看这个内存段做了什么.返回主窗口循环 后,来到401211地址处的Call。

004013FF 00401400	L:	C9 C2 0400	RETN 4	
00401403	5	55	PUSH EBP	
00401404 00401406 00401407 00401407 00401408 00401409	:	88EC 50 52 90 90	MOU EBP, ESP PUSH EAX PUSH EDX NOP	ntdll.KiFastSystemCallR
00401409 00401408 00401400 00401400		90 42 88 02	110P DB 42 DB 88 DB 88	CHAR 'B'
0040140E 00401412 00401413 00401414 00401415		35 00 43 00 01 89 02 83	ASCII "57C", 0 DE 01 DE 89 DE 89 DE 83	
00401416 00401419 00401419 00401419		C2 048B 02 35 01	RETH 9804 D8 02 D8 35 D8 01 D8 01	CHAR '5'
00401410 00401422 00401425 00401425 00401426 00401426		15 52890283 C2 0488 02 35 0F	ADC EAX, 83828952 RETH 8884 DB 82 DB 35 DB 35	CHAR '5'
00401420 00401429 00401428 00401428 00401428 00401420 00401420		00 17 10 89 02 83 C2 0498	DE 00 DE 17 DE 10 DE 89 DE 89 DE 02 DE 83	17 25 W 4

我们来到了程序自修改的段了,从我们修改PUSH EDX处开始把。告诉OD代码已经被修改了,我们重新分析代码:

00401975	L:	C2 0400	LEAVE 4			· Registers (FPU)
00401404		55 DEEC	HOU LEP, COP			Los vincorrelas
00401406 00401407		50	PUCH ERM	Backup	• 1	12/ 0012FPR0
00401409		22		Сору		Est apprenting
00401408 0040140C		42 08	12 =	Binary		EIP 80481495 COALPRAT (1948)1488
00401400		02 35 90 43 90	(per) marrie	Undo selection	Alt+BkSp	FØ CI WIL SAL WITTHEFT
00401410		-99 82	118 1	Assemble	Space	The of this care antennet
00401416		63 C2 9400	RETH IS	Label		T & Tal allow All Treatment
00401417		95 91	10%	Comment		O R LATER ENCE DICESS IN
00401410	12	16 52990280	DEC ED1	Breakpoint		117 x222 2-3
00401422		62 92	and some	Hittrace		III IIII III
00401427 00401429 00401429		8E 80 17	100	Run trace		and ender the second
00401420 00401420 00401420		10	18.11	Goto		Far and Cons 1 6 4 Ere & B
00401420 0040142E		程 0408	RETH	Follow in Dump		Alte ello, tuestapertel ante
00401452		16	3	View call tree.	Ctrl+K	
00401434 00401437 00401439	1	45 45 90 89 82	Decil stars a	Search for		
00401439	1	64 58 0466.000	DO DESCRIPTION	Find references to		
004014 (F)	-	E8 7900000	CALL	View		
00401849	ri	C2 0000	PUSH ERK	Copy to executable		
00401440	1	60 62904000 60 94 65 F4010000	PUSH 4 PUSH 4	Analysis		Analyse code Ctrl+A
(0)401.457 (0)401.452 (0)401.452 (0)401.86.2		60 07144000 E0 6F000000 R1 30304000 R105, 07144000	POSH crackes1.004014 CREAT SUPPLACEMELS2. MOV EAN, DUGED PTR DS VOP EAN, DUGED PTR DS. LAN	Detach Process		Remove analysis frame munde
* EEP=0012F	444	an animal .	and a second sec	Process Patcher		Remove object scan from module

分析后看上去舒服了。



这就是我们常规的代码了,都做了什么呢? EDX+1把EDX移动到EAXXOR EAX,100430D,然后存放到EDX 中,EDX+4,把EDX移动到EAX然后与52154F01进行XOR后放回EDX中,EDX+4,把EDX移动到EAX然后与454516进行XOR后放回EDX中,EDX+4,把EDX移动到EAX然后与454516进行XOR后放回EDX中,出栈EDX,出栈EAX。AL+0x66

最后,我们都是不正确的解密从地址40143d几个内存位置。但是Call到SetDlgItemTextA并不是他们中的一个,这个指令没有改变的意义。一般前一个call到SetDlgItemTextA,我们已经看到,参数压栈,所以我们可以 假设当我们输入正确的密码,从40143d 401442说明会包含几个推指令(可能是3个)。

现在的大问题是EDX应该指向哪里?我们在这里有几个选择,应该指向哪里是经验告诉我们的。一个有经验的 逆向工程师可能会记得那串"一个错误发生"和认为"我们从来没有使用该字符串。我们看到它只是一个诱饵,从 未被使用过。也许这就是将解密..."。另一个提示,告诉我们,这是一个可行的解决方案是字符串被推到堆栈 上,但从未使用过。为什么?这里是一个堆栈的图片时,我们进入这个代码:

0018F9CC	00401216	RETURN TO	crackme1.00401216 from crackme1.00401408
0018F9D0 0018F9D4	00030708	ASCII "An	error occured"
0018F9DC 0018F9DC 0018F9E0	768A62FA 00030708	RETURN to	user32.768A62FA
0018F9E4 0018F9E8	00000111 00000065		
0018F9F0 0018F9F0	0040102B	RETURN to	crackme1.00401028 from <jmp.&kernel32.exit< td=""></jmp.&kernel32.exit<>
0018F9F8 0018F9FC	00000001		学校 吃小 化二
0018FA00	0040102B	RETURN to	crackmel.00401028 from (JMP. &KERNELS2.Ex1)

所以验证我们的猜测,我们希望EDX能指向这个字串。最简单的方法就是加载EDX时在内存中做偏移直到"An error occured"字符串放置,即地址403000。问题是,会占用太多字节。再看我们的代码,只有3个字节,也就是3个nop位置来修改EDX指向"An error occurred"字串,好了,保持头脑清醒,和记住堆栈上的字串,或许我们可以在堆栈上让EDX指向我们要的字串。

通常情况下,我们可以这样加载栈上的数据MOV EDX, [EBP + some_#] or MOV EDX, [EBP - some_#] 问题是那个才是我们要的偏移值?步过这两个指令,知道我们到达401408

80401403 00401404 00401406	\$ 55 • 8BEC • 50	PUSH EBP MOV EBP.ESP PUSH EAX	
88481487	. 52	PUSH EDX	
88481488	30	TOP	
88481489	. 90	NOP	
0940140H	. 30	HUP FOU	
00401408	2 42	HOU FOY DUODO DTD DO TEDUT	
00401400	· 6602	HOU EHA, DOURD PIK USILEDAJ	
00401402	. 35 00430001	MOUL DHODD DTD DC. (EDV1 EDV	
00401415	· 0702 04	DOD EDV 4	
00401410	0002 04	MOU EOV DWORD PTP DS. FEDY1	
89481415	35 014F1552	YOR FOX S2154F81	
0040141E	8982	MOU DWORD PTR DS+ (EDX1. EQX	
00401421	8302 84	ADD EDX.4	
09401424	. 8B02	MOU ERX, DWORD PTR DS: LEDX1	
00401426	. 35 ØEØD1710	XOR ERX, 1017000E	
68401428	. 8982	MOU DWORD PTR DS: (EDX).ERX	
08401420	. 8302 04	ADD EDX.4	
00401438	. 8802	MOV EAX, DWORD PTR DS: LEDX1	
00401432	. 35 16454500	XOR ERX, 454516	
00401437	. 8902	MOU DWORD PTR DS:[EDX],EAX	100000
00401439	. 5A	POP EDX	0008E3C8
0040143R	. 58	POP ERX	0008E3C8
00401438	. 84 66	ADD AL.66	
00401430	. 2800	SUB CH, BL	
0040143F	- 70 CR	JBE SHORT crackme1.0040140B	Concernance of
00401441	. 48	DEC EDX	
Middle and 2	19	I FMUE	

看暂存器中EBP指向的地址,字串比18f9c0高12字节。(栈是先进后出的,所以比他高)



因此我们的指针指向的位置应该是:MOV EDX, [EBP + 0x0C]让我们看看多少个字节被偏移了

00401403 00401404 00401406 00401407 00401407 00401408	8 55 . SBEC . 50 . 52 . 52 . 52 . 52 . 52 . 52 . 52 . 52	PUSH EEP NOV EEP,ESP PUSH EAX PUSH EOX NOV EOX,DUCHD PTR \$5:(EEP+C) INC EDX	crackne1.00403000					
00401140E 00401415 00401415 00401415 00401415 00401416 00401416 00401421 00401421 00401421 00401421 00401421 00401425 00401425	0401400 > +2 0401400 . 5502 0401400 . 5502 0401401 . 5902 0401401 . 5902 0401415 . 5902 0401415 . 5902 0401415 . 5902 0401415 . 5902 0414141 . . 0401415 . 5902 0414141 . . 0401420 . . 0401420 . . 0401420 . . 0401420 . . 0401420 . . 0401420 . . 0401420 . .	THO EAST TRANSFER OF DESTEEDING HOW CANOD PTR DS:(EDN1), EAST ROD EDN. HOW EAST, DUGOD PTR DS:(EDN1) KOR EAST, S2154701 HOW EAST, S2154701 H	Assemble at 00401408					
00401430 00401437 00401437 00401437 00401439 00401436 00401436	8802 95 16454500 8902 58 58 04 66 2800 2800	HOU ERX, DARED PTR DS:(EDX) XOR ERX, 44550 HOU DARED PTR DS:(EDX), ERX POP ERX POP ERX SUB CH.EL	0000EBCS 0000EBC9	- TR (77 - 1 -				

看上去正好。现在我们单步走看看会发生什.首先在地址401408处, EDX 装载了我们要的字串。

Registe	rs (FPU)	<
EAX 000 ECX 693 EDX 000 EBX 000 ESP 001 EBP 001	00065 50000 03000 ASCII "An error occured" 00001 8F9C0 8F9C8	1
ESI 004 EDI 000	0102B crackme1.0040102B 00000	
EIP 004	0140B crackme1.00401408	
C Ø ES	0028 32bit 0(FFFFFFF)	04

Edx + 1, 现在指向了第二个字符n, 然后从n开始4字节移动到eax, 然后eax和0x100430D进行Xor后保存到

Address	Hei	i di	IMP	-													ASCII	
00403000	41	6E	20	65	72	72	6F	72	20	6F	63	63	75	72	65	64	An error occured	Ī
00403010	66	6F	72	63	65	3F	80	60	60	00	40	00	60	60	60	88	force?	
00403030	60	88	00	60	88	00	00	00	08	E3	88	00	00	88	CF	66	**********************	
80403858	80	66	92	42	80	88	80	60	60	88	66	60	80	80	90	88	0w88	
00403060	00	00	00	69	00	00	00	00	00	00	00	00	00	00	00	00		
00403070	88	00	88	88	00	88	88	60	60	88	60	88	88	00	88	00		
88403898	66	60	66	66	60	ĕĕ	60	60	ĕĕ	õõ	ĕĕ	00	00	ee	66	00		
00403000	88	88	80	88	00	88	00	00	80	88	00	00	88	00	00	88		
00403000	00	00	00	õõ	00	ĕĕ	00	00	ĕĕ	õõ	õõ	60	00	00	00	00		
00403000	89	00	00	88	88	00	00	88	00	88	00	00	88	88	00	00		
004030F0	80	00	60	00	80	60	00	80	00	60	00	60	88	00	60	88		
00403100	00	80	00	00	00	00	00	00	00	00	00	00	88	00	00	00		
00403110	60	00	80	00	00	80	00	00	80	60	00	88	60	00	80	00		
00100100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	(20040000000000000000000000000000000000	

移动后

Address	Het	d d	IND							1							09011	
00403000	41	63	63	65	73	72	6F	72	20	6F	63	63	75	72	65	64	Accessor occured	ī
00403010	66	54 6F	72	63	69	SF 3F	60	20	66	61	20	62	66	66	69	65	force?	
00403030	80	00	00	00	00	00	00	00	08	E3	00	00	00	00	CF	66		
00403040	30	77	42	42	0A 0A	00	00	00	00	00	88	00	00	30	40	00	0wBB00.	
00403060	00	00	00	00	õõ	00	õõ	00	õõ	00	00	00	00	ĕĕ	õõ	00		
00403070	00	00	00	00	00	00	00	88	00	00	00	00	00	00	00	88		
00403090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
004030R0	88	00	00	00	00	00	88	60	00	90	00	00	00	00	00	00		
00403000	00	00	00	00	00	00	00	60	00	00	66	00	60	00	00	00		
00403000	88	00	00	68	00	88	00	00	00	88	00	00	00	80	88	00		
004030E0 004030F8	90	00	00	00	00	00	00	60	80	00	00	60	00	00	00	00	***************	
00403100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	******73* (3017499).	
00403110	99	00	00	89	90	00	00	99	90	88	00	00	88	90	99	88	************	
00403130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		

嗯嗯,字串已经被修改了。我们继续,然后edx偏移4字节后,装载到eax然后与0x52154f01进行Xor后保存到 edx偏移后的地址。

Address	Het	t di	IMP			-							_	_			ASCIT
00403000	41	63	63	65	73	73	59	67	72	6F	63	63	75	72	65	64	Access groccured
00403010	66	54	72	79	69	6E	67	20	74	6F	20	62	72	75	74	65	they want the branch
06463620	66	6F	72	63	65	3F	66	66	66	66	40	66	66	00	66	66	force?
00403030	99	90	00	66	99	00	00	99	88	E3	99	00	66	66	CF	66	angeren Blannef
06463040	30	66	92	42	UH	00	00	00	66	90	99	99	66	30	40	90	0wBB
00405050	04	00	66	00	00	00	00	90	00	90	90	00	00	00	00	00	•
00403050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	*************
00405070	00	00	00	60	00	00	60	00	00	00	00	00	00	00	00	00	
00403030	80	00	00	66	80	00	88	80	00	66	80	00	66	80	66	00	
08463000	ññ	ññ	66	88	ññ	66	ññ	ññ	ññ	66	AA	ññ	ññ	ññ	ññ	ññ	
00403080	õõ	00	00	00	ěě.	00	00	88	00	00	00	00	00	00	00	00	
00483800	00	88	00	00	80	00	00	88	00	00	00	00	00	00	60	00	
88483808	90	00	00	00	00	00	80	00	00	00	00	00	00	00	00	00	

后续也是一样的操作,只是Xor的数据不一样,最后得到

Address	Hex dump	11320
00403000	41 63 63 65 73 73 20 67 72 61 6E 74 65 64 20 2	Access granted 1
00403010	66 6F 72 63 65 3F 00 00 00 00 40 00 00 00 00 00 00	force?
00403030	00 00 00 00 00 00 00 00 08 E3 00 00 00 00 CF 66	0.00 BT=f
00403050	04 00 00 00 00 00 00 00 00 00 00 00 00 0	•
00403060	00 00 00 00 00 00 00 00 00 00 00 00 00	
00403080	00 00 00 00 00 00 00 00 00 00 00 00 00	
00403090	80 80 80 80 80 80 80 80 80 80 80 80 80 8	
00483080	00 00 00 00 00 00 00 00 00 00 00 00 00	
80403009	00 00 00 00 00 00 00 00 00 00 00 00 00	
00403050	80 80 80 80 80 80 80 80 80 80 80 80 80 8	

我们发现字串对了,尽管我们还没有明白原理.只是知道正确的String是什么了.问题是,既然我们输入了错误的

密码,最后的陈述是不正确的解密,我们的信息将不会被显示.我们必须要找到在调用SetDlgItemTextA时往栈中压入什么参数,查找SetDlgItemTextA的帮助文档,这里有3个参数:

LPCTSTR lpStrin	g // text to set
int nIDDIgItem,	/ide 哲爱破解论坛
HWND hDlg, //	handwww.5266jie.cn

第一个参数:简单,PUSH [EBP + 0x0c]就是我们的字串指针。、第二个和第三参数有点麻烦:但是幸运的是 我没有参考.因为这里有一个失败时调用SetDlgItemTextA的参数。

00401219 01401220 00401220 00401220 00401220 0040123F 0040123F 00401237 00401237	C2065 44304600 000 1000 Dators PTR Dist 4450441,0 FF05 45304000 UNC Dators PTR Dist 4450441,0 FF05 45304000 OFF 0 actions 0 00483011 64 03 FF75 08 PUSH Cracinel.00483011 FF75 08 PUSH C205 44304000 000 FUSH 0252.5et01g1temText62 C705 44304000 000 FUSH 02544100441,0 FF05 04504000 000 FUSH 0254410441,0 FF05 04504000 000 FUSH 0254410441,0 FF05 04504000 000 FUSH 0254410441,0 FF05 04504000 000 FUSH 0254410441,0 FF05 04504000 FUSH 02570 Cardenau D040124C	Tent = "Trying to bruteform?" = Contact[] = 3 him = 00070700 ('Cracknes12 by Detten', class="s30 SetDigItenText0
0040123F 00401232 00401237 00401241	- FF75 00 E8 5930000 CALL 497 USER32, SetDigitenTeuter - C705 44504000 CM MOU DUCED PTR DS:(445044), U - E8 09	himg = 00070700 ('Cracknew12 by Detten', class: SetDigItenTects

我们发现ControlID为3,Window Handle为707AA第二个参数:PUSH 3第三个参数有点麻烦,我们看下栈数据



第三个参数: PUSH [EBP + 8]好了,现在开始补丁吧

00401400	C2 8488		
00401404	SEFC	MOU ERP. ESP	
00401406	. 50	PUSH ERX	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
00401407	- 52	PUSH EDX	crackme1.00403000
00401405	8855 GC	HOU EDX, DWORD PTR SS: [EBP+C]	crackme1.00403000
00401400	> 42	INC EDX	crackme1.0040300D
9040140C	. 8802	MOU EAX, DWORD PTR DS: [EDX]	
00401402	. 35 80438881	XOR ERX, 1004300	
00401418	. 8762	ODD EDV 4	
00401410	9902	HOU EQY, DWORD PTR DS+ FEDX1	
22401412	35 R14E1552	XOR FOX, S2154F01	
0040141F	. 8982	HOU DWORD PTR DS: (EDX), EAX	
00401421	8302 84	ADD EDX. 4	
00401424	, 8802	MOU EAX, DWORD PTR DS: LEDX1	
00401426	, 35 0E001710	XOR EAX, 1017000E	
00401428	. 8902	MOU DWORD PTR DS: [EDX], EAX	
60481450	+ 83C2 84	HOD EDX, 4	
00401426	+ 8802 05 16 AF AF 00	YOD FOW ACAELS FIR USILEDAJ	
00401432	+ 35 15454500	HOLL DHODD DTD DS. LEDY1 EDY	
COMPANY AND	50	POP FOX	00095308
99401439	58	POP ERX	0008E3C8
08401430	FF75 8C	PUSH DWORD PTR SS(LEEP+C)	crackme1,00403000
8840143E	68 83	PUSH 3	
00401440	FF75 88	PUSH DWORD PTR SS: [EEP+8]	
80401443	. E8 7800000	CALL (UMP. &USER32. SetDigitenTextA)	SetDigItewTextP
00401448	. 09	LERVE	the state of the s
00401440	LZ 0800		

好了补丁完成了。(译者注:看作者的分析可能和我们的不一样,因为10次输入按键不同导致的。所以按照作者的思路还是可以正常破解的)作业:只用按一个按钮就显示成功信息。加分作业:启动就显示成功信息。 http://pan.baidu.com/s/1bEvITG 密码: e5ve

其实我希望能找到一个高压缩的软件,被你们提醒了,就放上网盘链接了。

谢谢支持,有时间会继续做后面的翻译的,我也是新手,翻译是按照我理解的方式去做的,可能和原版不同。 大家共同学习吧!

第十六课(下):再谈暴力破解

先说说上篇文章最后留的作业这是我做的,可能不是最好的方法第一个是按一次按钮出现破解成功信息。这个简单,把代码中和0xA比较改为和0x1比较就OK了。第二个也不是很难,初始化显示成功在初始化数 据"DEAD""42424242"和错误信息后的跳转,直接跳到Call正确信息的地方。

注意:1、 需要把3次错误检查的地方Nop掉,否则点三次按钮还是会出现暴力破解信息。2、 点第二次的时候会回到一个错误发生字串,这里我还没想到要怎么处理,有思路的请提供想法,谢谢! (其实用奇偶的方法可以做,但是代码空间可能够)

以上就是作业的思路,有兴趣的再去试试看,可能你的方法比我好。欢迎提供不同的思路。

翻译都是我理解的方式进行描述,可能和原文不一致。本教程中文版只在吾爱破解论坛 首发。转载请注明来自 吾爱破解论坛@52pojie.cn

正文开始了

暴力破解是一种方法,这中方法是你可以从程序中找到爆破点然后直达我们需要的地方,虽然通过常规的加密/ 解密你已经知道这个程序的输入和输出,但是你不知道他的解密过程,而是直接打了一个补丁就完事。而这种 方法与通过输入用户名和序列号的方式是不同的。如果你以前下载过破解软件,是通过输入用户名和序列号来 让程序工作的,它有可能是通过暴力破解来的。

这种工作方式是通过常规的加密/解密输入的用户名和序列号,你尝试不同的输入直到有一次匹配成功,例如: 我们输入"12121212",程序通过常规解密后得到"j6^^gD7-L",我们使用不同的输入,得到的结果是不同的。 我们就是要想办法知道"12121212"是怎么变成"j6^^gD7-L"的,然后让我们的输入的序列号能让程序启动,或 者能够注册成功。

请记住,这中方法只适用于程序内部检查用户名和序列号,不适用于网络验证。

和大家说的一样,暴力破解不是难事,首先你至少知道一门编程语言这样就可以自己编写暴力破解的程序,此 教程主要是汇编语言,因为要分析代码中的算法。作者自己也会两门其他编程语言____,因此你就可以在高级 语言中做暴力破解算法(译者注:我只会**C++**初阶)

另一个就是要明白用户名和序列号是怎么变成输出的,这样做的原因是它减少了操作次数,所以我们必须尝试。如果我说我们必须在密码栏输入"SECRET"导致输出为"MESSAGE"。这有无限多的方法。但是如果我说把用户名做异或操作后的值很有价值,这样会减少很多方法。

破译密码

还记得前面的教程中我问你是否能破译密码吗,修改什么地方能出现成功,下面就是所有的程序代码: (译者注: 这里后面的解释好像ecx和eax反了,我自己是吧eax当作a,ebx当作b,ecx当作c,然后我放上我的解释)

004012A9 mov ecx, dword_403040 004012AF mov ebx, dword_40303C 004012B5 mov eax, dword_403038 004012BA cmp [ebp+arg_0], 1 004012BE jnz short loc_4012D0 004012C0 add ecx, 54Bh 004012C6 imul ebx, eax 004012C9 xor eax, ecx 004012CB jmp loc 4013E7 004012D0 cmp [ebp+arg_0], 2 004012D4 jnz short loc_4012E8 004012D6 sub ecx, 233h 004012DC imul ebx, 14h 004012DF add ecx, eax 004012E1 and ebx, eax 004012E3 jmp loc_4013E7 004012E8 cmp [ebp+arg_0], 3 004012EC jnz short loc_4012FD 004012EE add eax, 582h 004012F3 imul ecx, 16h 004012F6 xor ebx, eax 004012F8 jmp loc_4013E7 004012FD cmp [ebp+arg_0], 4 00401301 jnz short loc_401312 00401303 and eax, ebx 00401305 sub ebx, 111222h 0040130B xor ecx, eax 0040130D jmp loc_4013E7 00401312 cmp [ebp+arg_0], 5 00401316 jnz short loc_401324 00401318 cdq 00401319 idiv ecx 0040131B sub ebx, edx 0040131D add eax, ecx 0040131F jmp loc_4013E7 00401324 cmp [ebp+arg_0], 6 00401328 jnz short loc 401339 0040132A xor eax, ecx 0040132C and ebx, eax 0040132E add ecx, 546879h 00401334 jmp loc 4013E7 00401339 cmp [ebp+arg_0], 7 0040133D jnz short loc_401351 0040133F sub ecx, 25FF5h 00401345 xor ebx, ecx 00401347 add eax, 401000h 0040134C jmp loc 4013E7 00401351 cmp [ebp+arg 0], 8 00401355 jnz short loc_401367 00401357 xor eax, ecx 00401359 imul ebx, 14h 0040135C add ecx, 12589h 00401362 jmp loc_4013E7 00401367 cmp [ebp+arg 0], 9 0040136B jnz short loc 401378 0040136D sub eax, 542187h 00401372 sub ebx, eax 00401374 xor ecx, eax 00401376 jmp short loc 4013E7 00401378 cmp [ebp+arg 0], 0Ah 0040137C jnz short loc 40138A

```
; b *= a
; a^= c
; ***** Button 2
; c -= 233h
; b *= 14h
; c += a
; b &= a
; ***** Button 3
; a += 582h
; c *= 16h
; b ^= a
; ***** Button 4
; a &= b
; b -= 111222h
; c ^= a
; ***** Button 5
; a /= c, divisionrest --> (r)
; b -= r
; a += c
; ***** Button 6
; a ^= c
; b &= a
; c += 546879h
; ***** Button 7
; c -= 25FF5h
; b ^= c
; a += 401000h
; ***** Button 8
; a ^= c
; b *= 14h
; c += 12589h
; ***** Button 9
; a -= 542187h
; b -= a
 ; c ^= a
; ***** Button 10
```

0040137E cdq 0040137F idiv ebx ; a /= b, division rest -->(r)00401381 add ebx, edx ; b += r ; a *= r 00401383 imul eax, edx ; c ^= r 00401386 xor ecx, edx 00401388 jmp short loc 4013E7 0040138A cmp [ebp+arg_0], 0Bh ; ***** Button 11 0040138E jnz short loc_4013A3 00401390 add ebx, 1234FEh ; b += 1234FEh 00401396 add ecx, 2345DEh ; c += 2345DEh 0040139C add eax, 9CA4439Bh ; a += 9CA4439Bh 004013A1 jmp short loc_4013E7 004013A3 cmp [ebp+arg 0], 0Ch ; ***** Button 12 004013A7 jnz short loc 4013B2 004013A9 xor eax, ebx ; a ^= b 004013AB sub ebx, ecx ; b -= c ; c *= 12h 004013AD imul ecx, 12h 004013B0 jmp short loc_4013E7 004013B2 cmp [ebp+arg_0], 0Dh ; ***** Button 13 004013B6 jnz short loc 4013C8 004013B8 and eax, 12345678h ; a &= 12345678h 004013BD sub ecx, 65875h ; c -= 65875h 004013C3 imul ebx, ecx ; b *= c 004013C6 jmp short loc 4013E7 004013C8 cmp [ebp+arg_0], 0Eh ; ***** Button 14 004013CC jnz short loc_4013DB 004013CE xor eax, 55555h ; a ^= 55555h 004013D3 sub ebx, 587351h ; b -= 587351h 004013D9 jmp short loc 4013E7 004013DB cmp [ebp+arg_0], 0Fh ; ***** Button 15 004013DF jnz short loc 4013E7 004013E1 add eax, ebx ; a += b 004013E3 add ebx, ecx ; b += c 004013E5 add ecx, eax ; c += a

在他的教程中为我做了大部分工作 (当我发现我已经完成了三分之二后).***

现在我们知道了每一个按钮都做了什么操作了。接下来我们需要的是输入和输出。这是我们已经知道的数据 了,在代码自修改段中有官方(译者注:程序自己的算法,我们分析的代码)的算法,然后进行一些列合法的 异或操作。具体就是与变量a、b、c进行异或后保存,然后第二个以后的数据都是和之前异或后的数据进行再 次异或。

地址 401407的值EB 3F 90 90 与a 异或后为 528B550C(这个值是我们之前修改出来的)然后反向求出 a为 B9B4C59C地址 40143B的值04 66 E7 BB与b 异或后为FF 75 0C 6A 然后可反向求出b 就是直接与结果异或 就OK了地址 40143F 的值 4D BD 08 8B与c 异或后为03 FF 75 08 然后可反向求出c

我们最终要做的是尝试修改的每一个组合,通过点击按钮模仿每一个尝试手动可能的组合,当我们做了10次按 钮操作后,我们可以看到a\b\c中的值,这个就是正确的值。(译者注:不明白为什么是正确的值,是我们补丁 后的程序?)

这个程序的作者提供了前两个值是7和9.给出的原因是,如果你的电脑比较慢的话,要把所有有可能的值都试一次的话要花费太多时间,在不知道前两位的情况下我用一台8核的电脑花了大约1小时才破译出密码。知道前两位的情况下只花了大约1分钟。通常在破解程序时我们不会有任何的提示(当然),我在破译程序中有包含两

个已知的数。

下面是C语言写的破译程序

```
#include <iostream>
using namespace
                    std;
void brute( void )
{
   char finalAsciiSerial[11] = "";
             i, varA, varB, varC,tempVar, tempSerial[10];
   int
   // we know the first number is '7'
   for
         (tempSerial[0] = 7;tempSerial[0] <= 7; tempSerial[0]++)</pre>
    {
     // and we know the second number is '9'
     for
           (tempSerial[1] = 9;tempSerial[1] <= 9; tempSerial[1]++)</pre>
     {
     for
           (tempSerial[2] = 1;tempSerial[2] <= 15; tempSerial[2]++)</pre>
     {
      for
             (tempSerial[3] = 1;tempSerial[3] <= 15; tempSerial[3]++)</pre>
      {
        for
               (tempSerial[4] = 1;tempSerial[4] <= 15; tempSerial[4]++)</pre>
         {
        cout << ".";</pre>
                             // Update display
               (tempSerial[5] = 1;tempSerial[5] <= 15; tempSerial[5]++)</pre>
        for
        {
          for
                 (tempSerial[6] = 1;tempSerial[6] <= 15; tempSerial[6]++)</pre>
          {
          for
                 (tempSerial[7] = 1;tempSerial[7] <= 15; tempSerial[7]++)</pre>
          {
                   (tempSerial[8] = 1;tempSerial[8] <= 15; tempSerial[8]++)</pre>
            for
            {
                   (tempSerial[9] = 1;tempSerial[9] <= 15; tempSerial[9]++)</pre>
            for
            {
                 // Reset variables
                 varA = 0xDEAD;
                 varB = 0xDEAD;
                 varC = 0x42424242;
                 // Apply each digit
                 for
                      (i = 0; i < 10; i++)
                 {
                      switch (tempSerial)
                      {
                      case 1:
                          varC += 0x54B;
                          varB *= varA;
                          varA ^= varC;
                          break ;
                      case 2:
                          varC = varC - 0x233 +varA;
                          varB = (varB * 0x14)& varA;
```

```
break ;
case 3:
   varA += 0x582;
   varC *= 0x16;
   varB ^= varA;
   break ;
case 4:
  varA &= varB;
   varB -= 0x111222;
   varC ^= varA;
   break ;
case 5:
   if (varC != 0) // Watch divide by zero!
   {
      varB -= (varA %varC);
      varA /= varC;
      varA += varC;
   }
   break ;
case 6:
   varA ^= varC;
   varB &= varA;
   varC += 0x546879;
   break ;
case 7:
   varC -= 0x25FF5;
   varB ^= varC;
   varA += 0x401000;
   break ;
case 8:
   varA ^= varC;
   varB *= 0x14;
   varC += 0x12589;
   break ;
case 9:
   varA -= 0x542187;
   varB -= varA;
   varC ^= varA;
   break ;
case 10:
   if (varB != 0) // Watch divide by zero!
   {
      tempVar = varA %varB;
      varA /= varB;
      varB += tempVar;
      varA *= tempVar;
      varC ^= tempVar;
```

```
}
         break ;
     case 11:
         varB += 0x1234FE;
         varC += 0x2345DE;
         varA += 0x9CA4439B;
         break ;
     case 12:
         varA ^= varB;
         varB -= varC;
         varC *= 0x12;
         break ;
     case 13:
         varA &=0x12345678;
         varC -= 0x65875;
         varB *= varC;
         break ;
     case 14:
         varA ^= 0x55555;
         varB -= 0x587351;
         break ;
     case 15:
         varA += varB;
         varB += varC;
         varC += varA;
         break ;
     }
}
// stop if serial equals propervalues
if ((varA == 0x9CC5B4B9)
                 &&(varB == 0xD1EB13FB)
                 &&(varC == 0x837D424E))
{
     // Convert to ASCII
     for (i = 0; i < 10; i++)
     {
         if (tempSerial <= 9)</pre>
         {
            finalAsciiSerial= tempSerial + 0x30;
         }
         else
         {
            finalAsciiSerial = tempSerial +0x37;
         }
     }
     cout << "\n\n*****
                           Bruteforced serial: ";
                       cout<< finalAsciiSerial << "\n";</pre>
     return;
```

```
}
int main()
{
    cout << "Bruteforcerby R4ndom\n\n";
    brute();
    cout << "\nBruteforcing done...\n";
    return 0;
}</pre>
```

首先,建立我们的变量a\b\c,然后我们知道第一个和第二个是7和9,后面的是在1-15之间,然后我插入了一个"点"字串在控制台中输出,我不太喜欢程序没有任何反应,能看到解密的动作,证明程序没有当掉。接下来,我们执行的变量的修改取决于哪个键被按下,就像我们在上面显示的一样。当输入10个数据时(因为长度是10位),我们会检查这三个变量,看它们是否与我们程序中比较的数据一致,如果一致,我们则停下来,把这个数据转换为ASCII码,然后把它显示在程序上。如果不一致则继续进行下一条数据。下面是控制台破译过程和结果:

C/Windows/system32/cmid.exe	

***************************************	*************************************
***************************************	*********************************
***************************************	***************************************
***************************************	***************************************
***************************************	***************************************

Bruteforced serial: 79DE624C4B	
ruteforcing done	五受破解论
wees any key to continue	百友吸附比
	The second secon

让我输入破译后的密码看看程序怎么运行



我们现在已经破译了这个程序。

译者注:算法中要注意5和7里面,5和A是一样的东西

00401312	>	807D 08 05	cmp	byte ptr ss:[ebp+0x8],0x5			
00401316		75 OC	jnz	Xcrackme1.00401324			
00401318	-	99	cdq		将eax扩展为64位,放到edx中		
00401319		F7F9	idi	ecx .	过里idiv ecx 意思就是eax除以ecx,	商放在eax中,	余数放在edx中。
0040131B	ιŵο.	2BDA	sub	ebx,edx			
0040131D	4	0301	add	eax,ecx			
0040131F		E9 C3000000	jmp	crackme1.004013E7			
00401324	>	807D 08 06	cmp	byte ptr ss:[ebp+0x8],0x6			
00401328	- G. Q.	75 OF	jnz	Xcrackme1.00401339			
0040132A	14-1	3301	xor	eax,ecx			
0040132C	i i i i i	2308	and	ebx,eax			
0040132E	64 J	8101 7968540	add	ecx,0x546879			
00401334		E9 AE000000	jmp	crackme1.004013E7			
00401339	>	807D 08 07	cmp	byte ptr ss:[ebp+0x8],0x7			
0040133D		75 12	jnz	Xcrackme1.00401351			
0040133F	14	81E9 F55F020	sub	ecx,0x25FF5			吾受破解论 坛
00401345	÷41.	33D9	xor	ebx,ecx	the second se		
00401347	640	05 00104000	add	<pre>eax,crackme1.<moduleentrypoint></moduleentrypoint></pre>	moduleentrypoint就是PE程序入口 4	01000	www.52pojie.ch

由于工作关系,后面的文章可能会比较晚才进行翻译,因为我要正在破解了,才能写出我理解的流程,才能在这里告诉大家。

第十七章(上):如何应对 Delphi 二进制代码(上)

翻译都是我理解的方式进行描述,可能和原文不一致。

本教程中文版只在吾爱破解论坛 首发。

转载请注明来自吾爱破解论坛@52pojie.cn

正文开始了

本次教程中,我们将讨论用Delphi写的程序是怎么工作的。Delphi写出的程序和其他语言写出的程序有很大不同。你会说有很多个调用(不仅仅是一个经典程序)我们将讨论其他技术。

本教程需要下载逆向的两个软件,一个是DeDe,一个是ExeinfoPE,均可在本教程中下载。

当然还是需要用到OD的,请自行下载。

Delphi

大多数程序都包含许多窗体,基本的窗口和对话框。你使用窗体进行绘制,如添加一个按钮,对话框等,这是 你喜欢的方式。你唯一要做的就是告诉Delphi编译器你希望程序做什么样的动作,例如:当你按下按钮后,希 望打开一个打开文件的对话框。被教程中你就是要告诉Delphi编译器通过你的代码打开一个对话框,只是简单 的代码就可以做到。

这些窗体,以及他们所关联的一切(字串,大小,颜色)都保持到资源中,看上去像C++程序,但是实际上有 很大不同。一个有趣的现象是,Delphi通过名称来控制这些资源,也就是说你所调用的特定资源的名称被硬编 码编译到可执行文件中的名称。名称看上去就是资源。这样有好处也有坏处,好处是你可以很容易的找到这些 名称与资源的管理,坏处是他们都存在同一个地方,在逻辑上看不到正常的代码。所以要找到按钮关联的资源 会比较麻烦。

多数程序以分析你就知道,这是C++的,Delphi写的程序在幕后有很多不同,代码看上去就和以前的不同,这就是麻烦的一个原因。

第一个目标

你的第一个问题会问,我们怎么知道这是一个Delphi编译的程序?多数的解释都只说了大概,这是我们就需要使用软件来确认了。打开ExeInfoPE。这个程序可以查看程序是否带壳,带来什么壳,如果不带壳我们就知道 是用什么语言来写的了。当你第一次使用ExeInfoPE时,会出现下面的图像:

E B	einfo PE - ver.0.0.3.0	by A.S.L - 617	sign 2011.07.29	L	
	(File :)				H R .
Info PE	Entry Point :	<	EP Section : First Bytes : SubSystem :		> 86MB
	File Offset :				- S Plug
	Linker Info :				About
	File Size :	<	Overlay :	1	Options
IXe	Diag				Exit
	Lamer Info - Help Hint	- Unpack info		0 ms.	Rip

然后打开我们要查看的程序, DelphiCrackme.exe, 然后就可以看见如下图:

Ex	Eile : Delohi	Crackme.exe	L - 61	7 si	gn 2011.07.2	19 (
	Entry Point :	000596D8	00 (<	EP Section :	.itext	> 86MB
H	File Offset :	00057CD8			First Bytes :	55.88.EC.83.C4	- S Plug
OI	Linker Info :	2.25			SubSystem :	Windows GUI	About
H	File Size :	00066A00h	<		Overlay :	NO 00000000	Options
X	Image is 32	bit executab	le		RES/OVL : 4	/0% 1992	Exit
9	Borland Delp	hi 2006/2007 - 1	www.bor	land	d.com		Pin
	Lamer Info -	Help Hint - Unpa	ick info			16 ms.	
	Not packed ,	try disassemble	OllyDbg) (v	/ww.ollydbg.d	le) or WD32dsm	Prodit 22

我们看到了该程序是由Delphi写的。也可以看到这个程序是没有加壳的。把这个放一边,使用OD加载该程序,你会看到:
00459608	5	55	PUSH EBP	
00459609		SBEC	MOU EBP.ESP	
R6459608		83C4 FR	800 ESP18	
004596DF	12	B8 947F4500	MOU F9X, De Joh I Co. 88457594	
864596E3		ES CCCDEGEE	COLL DetobilCr. 00406484	
004E94E9		01 00094500	MOLL FOY DWORD PTP DS. [45P9001	
00403020		0000	MOU FOY DUODD DTD DC. FEOVI	
OD4576ED	•	CO COOPERE	TOV EHA, DWORD FTR US: LEHAJ	
OU457BEF	1.5	ES BULUFFFF	CHLL Deiphiltr.00455450	D LLL D CONFEEDO
00459614		8800 948H4500	HUV ECX, DWORD PTR DS: 1458H941	Delphilr.0045F5CC
004596FH		H1 HCB94500	HOV EHX, DWORD PTR DS: 14589HCJ	
004596FF	1.0	8800	MOV EAX, DWORD PTR DS: [EAX]	a transferration
00459701		8B15 547D4500	MOU EDX, DWORD PTR DS: [457054]	DelphiCr.00457DA0
80459787		E8 6CCDFFFF	CALL DelphiCr.00456478	Contraction and the state of the
00459700		A1 ACB94500	MOV EAX, DWORD PTR DS: [4589AC]	
00459711		8800	MOV EAX, DWORD PTR DS: [EAX]	
00459713		E8 E0CDFFFF	CALL DelphiCr.004564F8	
00459718		E8 77AEFAFF	CALL DelphiCr.00404594	
00459710	1.2	8040 00	LEA EAX, DWORD PTR DS: LEAX1	
00459720		0000	ADD BYTE PTR DS: [EAX1. AL	
00459722		0000	ODD BYTE PTR DS. (FOX1 OL	
09459724	5	0000	ODD BYTE PTR DS+ (FOX1 OF	
00402124		0000	ODD BUTE PTP DS+FEOVI OL	
66469720		0000	ADD DUTE DTD DC. FEOVI OF	
00457720		0000	ADD DYTE PTR DOLLEHAJ, HL	
00407728		0000	HOD DYTE FTR DOLLEHAJ, HL	
00452720	1.1	0000	HUD BYTE FTR USILEHAL, HL	
8845972E	1.0	0000	HUD BYTE FTR US: LEHXJ, HL	
00459730		0000	HUD BYTE FIR DS:LEHXI, HL	
00459732		0000	HOU BYTE PIR DS:LEHXJ, HL	
00459734		0000	ADD BYTE PTR DS: LEAX], AL	
00459736		0000	ADD BYTE PTR DS: [EAX], AL	
00459738	1.0	0000	ADD BYTE PTR DS: [EAX], AL	
0045973A		0000	ADD BYTE PTR DS: [EAX], AL	
00459736		0000	ADD BYTE PTR DS: (EAX), AL	
0045973E		0000	ADD BYTE PTR DS: [EAX], AL	
00459740		0000	ADD BYTE PTR DS: (EAX) AL	
86459742		0000	ADD BYTE PTR DS: [EAX1. AL	
00459744		0000	ADD BYTE PTR DS: (FAX1. A)	
08459746	1.2	0000	ODD BYTE PTR DS+ FEQX1 OL	
00459748	1.1	0000	ODD BYTE PTR DS+FEOX1 OL	
88459740		0000	ODD BYTE PTR DS+ (FOY1 OL	
00450740	1.1	0000	OOD PUTE PTP DS. (EOVI OL	
OGAE074E		0000	ODD DUTE DTD DO. FEOVI OF	
00402146		0000	ADD DUTE DTD DC. (EOVI OL	
00402700		0000	HOD DYTE FIR DOLLEHAJ, HL	
00403752		0000	ADD DYTE PTR DS: LEHAJ, HL	
00459754		0000	HUD BYTE PIK USILEHAJ, HL	
00459756	1.00	0000	HUD BYTE FIR DS:LEHXJ, HL	
00459758		0000	HUD BYTE PTR US: LEHXJ, HL	
00459758		0000	HUD BYTE PTR US: [EAX], AL	
0045975C		0000	HOD BYTE PTR DS: [EAX], AL	
0045975E	1.4	0000	ADD BYTE PTR DS: [EAX], AL	
00459760		0000	ADD BYTE PTR DS: [EAX], AL	
00459762		0000	ADD BYTE PTR DS: (EAX), AL	
00459764	1	0000	ADD BYTE PTR DS: [EAX]. AL	
004E0766		0000	ODD DUTE DTD DC. FEOVI OL	

你会发现这和我们以往的程序不太一样。

查看Delphi程序的资源

资源是一个最大的区别,作为一个逆向工程师,这时就把程序使用Resource hacker打开看看吧。这时就发现 和以前的不同,多了一个叫RCDATA的新文件夹,点开看看



一般来说,最重要的子文件夹(资源部分)是TFORM段。在Delphi程序中。这些是窗口和对话框,在这个特殊的Crackme中,只有一个TFORM。在Resource Hacker打开TFORM1下面带小花图样,里面的信息告诉我们这个程序的所有数据。比如尺寸,颜色和位置,标题,办好多少个按钮等等。

通常情况下我们先看到标题栏,它告诉我们在窗口中显示的信息。在本教程中,"Delphi-MsgBosxes"是最要的,在程序中可能会调用TFORM1、TFORM2...很难知道哪一个窗体调用TFORM1.看着文字进行分析,例如,标题会告诉我们是否已经注册,关于则给出关于信息。

最后,对我们来说,最重要的在底部的按钮对象。重要的原因是按钮被按下后使用我们的用户名和序列号进行

注册,并确认是否注册成功。当按钮被按下时,我们要知道程序做了什么动作。本例中,按钮 是"Button1Click"。像前面说的一样,Delphi是通过ASCII与按钮进行关联的,因此当程序按下这个按钮后就会 按照"Button1Click"的方法进行动作。

通过Resource hacker得到的信息中,我们知道这个程序只有一个按钮,标题是"Delphi-MsgBoxex",按钮的句 柄为"Button1Click'。

让我们使用一个对于Delphi程序很重要的软件。

使用DeDe

DeDe打开一个Delphi编译后的程序,然后完整的呈现出来。显示出所有我们已经知道的资源信息和所有的调用,实现方法的地址,实现方法的名称,如果你愿意它还可以反编译,当然也可以修改它。让我们打开 DeDe,然后加载Crackme。

File Dumpers Tools Options About Image: Classes Info Units Info Forms Procedures Project Exports Classes Name Unit Name SelfPrt DFM Offset Unit List (from PACKAGEINFD)	DeDe 3.50.02 (c) 1999-2002 by D	aFixer		
Process Classes Info Units Info Forms Procedures Project Exports Unit List (from PACKAGEINFD) Unit List (from PACKAGEINFD) Readv	File Dumpers T	ools Options	About		
Classes Info Units Info Forms Procedures Project Exports Class Name Unit Name SelfPit DFM Offset Unit List (from PACKAGEINFD) Unit List Unit List (from PACKAGEINFD) Unit List (from PACKAGEINFD) Ready Example Unit List (from PACKAGEINFD)	1		Proc	ess	
Class Name Unit Name SeliPit DFM Offset Unit List (from PACKAGEINFD)	Classes Info Unit	ts Info Forms	Procedures Proje	ect Exports	
Peady	Class Name	Un	it Name Se	alfPrt DFM Offset	I
Ready					Unit List (from PACKAGEINFO)
Ready					
	Ready	-	1		一 一 一 一 一 一 一 一 一 一 一 一 一 一 一 一 一 一 一

首先,我要打开我们要逆向的程序。有两种方式,一个是通过程序的按钮找到我们要逆向的程序,另外就是直接把程序拖进DeDe,然后选择"YES"开始反编译程序。此时DeDe会弹出信息框询问是否已经加载。



同时我们的程序也跑起来啦



有时,DeDe会弹出一个信息框询问程序是否已经完全加载完成,然后继续进行分析。本例中只需要点击OK按钮,允许DeDe进行分析。DeDe将会关闭程序,然后询问是否使用更多的方式进行分析。

nitial dumps has finished	. Do you want to start extended analisys now?	
	<u>Yes</u> <u>N</u> o	1
Information		_
Extended analisys I type of unresolved file	ooks for unresolved procedures, tries to determine ds/members of the forms and other classes and	

我通常选择yes,然后不再弹窗。然后DeDe分析完程序:

DeDe 3.50.02 (c) 1999-2	2002 by DaFixer				- 0 - X
C:\Users\Random\Desktop\[DelphiDialogbox -	Process	DelphiD	DialogboxExample	
Classes Info Units Info	Forms Procedures	Project Ex	ports		
Class Name	Unit Name	SelfPrt	DFM Offset	Version: <unknown< th=""><th>version></th></unknown<>	version>
.02 .1 .56 .74 .84 A Boolean Byte Cardinal Char EAbort EAbort EAbstractError EAssertionFailed BitsError EClassNotFound ECiassNotFound EComponentError EConvertError EFCreateError EFCreateError EFCreateError EFCreateError EFCreateError EFCreateError		00410818 00406554 00417BE8 00407EA0 00407EC4 00401042C 00401004 00401058 00401080 0040102C 00407D34 00407CD4 00407CD4 00411ED0 00411ED0 00411F88 00407AB0 00411CB8 00411CB8 00411EB4 00411C50	00000000 00000000 00000000 00000000 0000	Unit List (from PACKAGEI) ActiveX ActnList Classes Clipbrd CommDlg Consts Controls Dialogs DIgs ExtCtrls FlatSB Forms Graphics GraphUtil HelpIntts ImageHlp ImgList Imm IniFiles Math Menus Messages MultiMon	NFO)
EHelpSystemException		0041B948 00407D90	00000000	Registry RegStr + RTLConsts	
Ready 182 sec.	DelphiDialogboxExam	420352 bytes			

DeDe默认显示类信息,我们通过"ClassInfo"表格可以看到,如果需要可以进行排序,但是我们需要的"FORMS"表:

DeDe 3.50.02 (c) 1999-2002 by DaFixed		- • ×
File Dumpers Tools Options About	t.	
C:\Users\Random\Desktop\DelphiDialogbox	Process DelphiDialogboxExample	e
Classes Info Units Info Forms Proc	edures Project Exports	
Class Name Offset		DFM Editor
1Form1 00066640	object Form1: TForm1	
	Left = 581 Top = 396	
	BorderStyle = bsToolWindow	
	ClientHeight = 89	
	ClientWidth = 127 Color = clBtnFace	
	Font Charset = DEFAULT_CHARSET	
	Font Height = -11	
	Font.Name = 'Tahoma' Font.Style = []	
	OldCreateOrder = False Position = poDesigned	
	PixelsPerlnch = 96	
	object Label1: TLabel	
	Left = 24 Top = 68	
	Width = 77	
	Caption = 'Unregistered'	
	Font.Charset = DEFAULT_CHARSET Font.Color = cfWindowText	
	Font Height = -11	
	Font Style = [tsBold]	
	ParentFont = False end	
	object Button1: TButton	
	Top = 8	
	Width = 113 Height = 49	
	Caption = 'MsgBox' TabOrder = 0	
	OnClick = Bulton1Click	
	end	
		-
	*	日日市にあり
Ready 182 sec. DelphiDialog	boxExam 420352 bytes	

这里,我们看到的和Resource Hacker一样。我这样做是为了以后可以直接略过Resource Hacker,直接使用 DeDe。现在点击"Procedures"表。这在DeDe中是很重要的。

DeDe 3.50.02 (c) 1999-2002 by DaFixer			- • ×
File Dumpers Tools Options About C:\Users\Random\Desktop\DelphiDialogbox	Process DelphiDia	logboxExample	
Classes Info Units Info Forms Procedures	Project Exports		
Unit Name Class Name Unit TForm1	Fvents Controls		DPR OFFS
	Event	BVA	Hint
	Buttom Liick	00437700	0013
Ready 182 sec. DelphiDialogboxExar	n 420352 bytes		And a subsection of the sector

这里DeDe显示了TFORM1的实现方法和地址。这是一个简单的程序,只有一个按钮,因此只有一个CallBack。现在我们记住这个地址"457F0C",显示用OD打开程序,看看我们能做什么。

找到补丁位置

如果进行字串搜索能看见:

R Found	d strings are		
Address 00459000 004590E8 00459477 004594F1 004594F1 004596R4	Disassembly SUB DWORD PTR D3:04505841.1 MOV EDX.DelphiCr.00459130 PUSH DelphiCr.00459488 PUSH DelphiCr.0045950C PUSH DelphiCr.0045951C PUSH DelphiCr.004596C8	Text string (Initial CPU selection) ASCII "0x" ASCII "TaskbarCreated" ASCII "Delphi Picture" ASCII "Delphi Component" ASCII "TaskbarCreated"	
			a - Maria Sta

(译者注:其实我刚开始使用字串搜索能找到,也可以打补丁,运行正常,不明白作者什么意思) 找到的Call在Delphi中都是无用的。(译者注:不明白)

R Found	l intermodular calls	
Address	Disassembly	Destination
88459608	UNP 71800000	(Initial CPU selection)
00459076	CALL DelphiCr.0040130C	De lph i Cr. 0040130C
0045903E	CALL DelphiCr.00403634	Detch (Cr. 00403634
00459047	CALL DelphiCr.00403664	DelphiCr.00403664
00459040	CALL Delphicr.00403728	Deinhi Cr. 00403728
004595B1	CALL DelphiCr.0040382C	DelphiCr.0040382C
004590CB	CALL DelphiCr.00404428	De Iph (Cr. 00404428
00459005	CALL DelphiCr.00404450	DelphiCr. 00404450
004594A6	CALL DelphiCr.00404450	DelphiCr, 00404450
00459698	CHLL DelphiCr.00404450	Delph (Cr. 00404450
004590ED	CALL DelphiCr.004046DC	DelphiCr, 004046DC
00459210	CALL DelphiCr.00405CE0	Belph ICr. 00405CE0
004590FC	CALL Delphicr.00405E68	Delph107.004050F0
004593DD	CALL DelphiCr.00406468	DelphICr.00406468
004593EA	CALL DelphiCr.00406468	DetphiCr. 00406468
00459404	CALL DelphiCr.00406468	Delph1Cr.00406468
00459411	CALL DelphiCr.00406468	Delph ICr. 00406468
0045941E 0045942B	CHLL DelphiCr.00406468	De IDA ICA, 00406468
00459438	CALL DelphiCr.00406468	DelphiCr.00406468
00459445	CALL DelphiCr.00406468	Delph1Cr.00406468
004590F2	CALL DelphiCr.00400328	Beloh (Cr. 00400404
00459101	CALL DelphiCr.0040C42C	DelphiCr.0040C42C
0045910B	CALL DelphiCr.0040CC08	DelphiCr. 0040CC08
00459228	CALL DelphiCr.0040DSCC	De Loh I Cr. 0040D5CC
0045913D	CALL DelphiCr.0040E334	Delph1Cr.0040E334
00459163	CALL DelphiCr.00410808	Delphich, 00410808
00459594	CALL DelphiCr.0041388C	DelphiCr.00413BBC
00459462	CALL DelphiCr.00413C78	De philo: 00413078
00459560	CALL DelphiCr.00413C78	DeiphiCr.00413C78
00459609	CALL DelphiCr.00413C78	DelphiCr.00413C78
00459472 004594CE	CALL Delphilr,00413004	DelphiCr. 00413004
0045957A	CALL DelphiCr.00413CC4	Delph (Cr. 00413004
0045958A	CALL DelphiCr.00413CC4	DelphiCr.00413CC4
00459623	CALL DelphiCr.00413CC4	DelphiCr.00413CC4
00459643	CALL DelphiCr.00413CC4	DelphiCr.00413CC4
88459674	CALL Delphicr.00413004	Detch ICT. 00413004
004594BF	CALL DelphiCr.00413D18	DelphiCr.00413D18
0045956A	CALL DelphiCr.00413D18	DelphiCr. 00413D18
00459395	CALL DelphiCr.00413D18	DeiphiCr.00413018
00459389	CALL DelphiCr.00413E04	DelphICr.00413E04
00459688	CALL Delphicr.00413F30	Delph (Cr. 00413F30
00459269	CALL DelphiCr.0041465C	DeiphiCr. 0041465C
0045936B	CALL DelphiCr.0041465C	Delph ICr. 00414650
00459370	CALL DelphiCr.0041465C	DelphiCr. 00414650
00459502	CALL DelphiCr.0041B768	DelphiCr.00418768
0045929D	CALL DelphiCr.0041CFF4	DelphiCr. 0041CFF4
00459334	CALL DelphiCr.0041EAA8	DelphiCr.0041EAA8
00459349	CALL DelphiCr.0041EAAS	DelphiCr.0041EAA8
00459201	CALL Delphiltr.0042638C	DelphiCr. 00426380
0045935A	CALL DelphiCr.004265E4	De iph (Cr. 004265E4
RANEOSTA	CULL DATABLE & 00435024	IT-1-817- 00492074

然后看到一大堆的Call

通常情况下,我们停在错误的地方,然后打补丁使程序进入到对的地方,而在Delphi中则不容易找到,需要进入大概15个call之后来到真正代码的地方。

但是如果我们直接去到DeDe中的Call的位置,我们可以看到

00-457E00	2 53	PUSH EBX	
00457F0D 00457F0F 00457F11	. 88D8 . 33C0 . 3C 01	MOV EBX.EAX XOR EAX.EAX CMP AL,1	
00457F15	B8 487F4500	MOU EAX, DelphiCr. 00457F48	ASCII "Registeredt"
00457F1F 00457F1F	BA 487F4500	MOU EDX, De Lph ICr. 00457F48 MOU EDX, DWORD PTR DS: [EBX+364]	ASCII "Registered!"
00457F2A 00457F2F	. E8 451DFEFF . 58	CALL DelphiCr.00439C74 POP EBX	DelphiCr.0043B79A
00457F30 00457F31	> B8 5C7F4500	MOU EAX, DelphiCr. 00457F5C	ASCII "Unregistered?"
00457F36 00457F38	. E8 3544FDFF . 58	POP EBX	DelphiCr.0043B79A
00457F3D 00457F3E	00	DB 00 DB 00	
00457F8F 00457F40	. FFFFFFF	DB 00 DD FFFFFFF	
00457F44 00457F48	. 08000000 . 52 65 67 69 73 74	DD 00000008 ASCII "Registered!",0	
00457F58 00457F58	. 0D000000 . 55 6F 72 65 67 69	DD 0000000D BSCII "Uppedisteredt".0	91.05.2
00457F6A 00457F6B	00	DB 00 DB 00	

哈哈,这里就是我们要的地方啊,让我们设断点跑起来看看。



注意,这个程序提示我们没注册,标题为"Delphi-MsgBoxes"。只有一个按钮,与我们之前的分析一致。来到 我们的断点处看看

现在只需要改一下就可以来到正确的地方。



(译者注: 这次程序实际不需要用到DeDe和ExeInfoPE,直接用查找字串参考就可以高低,只是作者希望我们使用DeDe和ExeInfoPE来分析Delphi程序)

第二个程序也是一样的方法,主要是使用DeDe和ExeInfoPE来处理Delphi程序。后面的等有时间再进行翻译。

第十七章(下):如何应对 Delphi 二进制代码(下)

翻译都是我理解的方式进行描述,可能和原文不一致。

本教程中文版只在吾爱破解论坛 首发。

转载请注明来自吾爱破解论坛@52pojie.cn

正文开始

接上章的第二个程序破解。

如果是从第一章一直看过来的朋友,一些简单的解释我这里就不再描述,你懂的。

使用ExeinfoPE确定程序是否加壳,什么语言写的

Ex Ex	einfo PE - ver.0.0.3.0 by A.S	.L - 617 s	ign 2011.07.2	29	×
	File : exif2htm.exe				H R
	Entry Point: 000A044C	00 <	EP Section :	CODE	> 86MB
H	File Offset: 0009F84C		First Bytes :	55.8B.EC.83.C4	- S Plug
E O	Linker Info: 2.25	0	SubSystem :	Windows GUI	About
A	File Size : 000BEC00h	<	Overlay :	NO 0000000	Options
Xe	Image is 32bit executab	le	RES/OVL : 4	/0% 1992	Exit
2	Borland Delphi (2.0 - 7.0) 1 Lamer Info - Help Hint - Unpa	992 - www. ick info	.borland.com	0 ms.	
	Not packed , try disassemble	OllyDbg (www.ollydbg.c	de) or WD32dsm	

发现没有壳, Delphi写的。

运行程序,弹出

ex	of2htm	X
	This is a shareware program, it means that you may try it free of charge, but if you like it and want to use it, you must buy it. product.	You will not see this message anymore when you buy the
	ок	吉茨都岡市地 1996年3月1日日

点击OK后出现

r						
ript editor HTML optio	ns					
Tag	Value					
Par Newline Sp Column Horiz D cript editor:	pace File name Full name !! efault ▼ Vert Default ▼ W	Size B KB MB	Image Imag Bg	e Width 0 🕃 Brdr	Height 0 🕃 📗	No patha

我们可以看见标题是未注册的,然后有一个注册按钮和关于按钮,点击关于后出现



我们发现是未注册的副本,点击注册按钮后出现

Name:				
T				
Code:				
	-	 	_	

出入用户名和密码后出现

Name:	
R4ndom	
Code:	exif2htm
12121212	Please restart E2H to check your reg info

好了,到这里我们知道程序的运行大致流程了。

打开DeDe分析程序

	ens options model					
C:\Users\Random\Des	ktop\exif2htm.exe 💽 🖻	Process	exif2htr	n		
Classes Info Units I	Info Forms Procedures	Project Exp	ports			
Class Name	Unit Name	SelfPrt	DFM Offset		Version:D7	
N .02		00415758	00000000	EU	Init List (from PACKAGEINFO)	
K .06		0041DCD0	00000000	IZ	ActiveX	
N .1		00406880	00000000	4	ActnList	
.2		0048F8F8	00000000	E	Buttons	
.24		0040839C	00000000	0	Jasses Nobrd	
.27		00443558	00000000	Č	ComCtris	
4.3		00490070	00000000	C	CommCtrl	4
.34		004083B4	00000000	6	CommDig	
4		00490D90	00000000	C	Consts	
.44		004083CC	00000000	0	Contrins	
.54		004083E4	00000000	C	Controls	L
6.6		00490E7C	00000000		Dialogs	
.74		00408474	00000000	e	IPTC	
.84		00408498	00000000	0	Digs	
.86		004412E8	00000000	F	ExtActos	
.9		0043BB68	00000000	Ē	ExtDigs	
.96		0044176C	00000000	F	latSB	
A		00412480	00000000		orms	
Boolean		00401004	00000000	0	ands	
Byte		00401070	00000000	F	telpIntfs	
Cardinal		00401098	00000000	1	mgList	
Char		0040102C	00000000		niFiles	
EAbort		004079BC	00000000	J	Consts	
EAbatratError		00000M00	0000000	- i	peq at the	191 7 2
Ready 7 sec.	exif2htm.exe	781312 bytes			10.22	11111111111

点击Procedures

DeDe 3.50.02 (c) 19	99-2002 by DaFixed	2	
File Dumpers Tool	ls Options Abou	ut exif2htm	
	top vexil2num.exe	Plocess	
Classes Info Units In	fo Forms Pro	cedures Project Exports	
Class Name	Offset		DFM Editor
TForm2	000BE5A4	Left = 218 Top = 122 Borderloons = [biSystemMenu, biMinimize] BorderStyle = bsSingle Caption = 'Exil2htm' LientHeight = 545 ClientWidth = 746 Color = clBtnFace DragMode = dmAutomatic Font Charset = DEFAULT_CHARSET Font Color = cfWindowText Font Height = -12 Font Name = 'MS Sans Serif' Font Style = [] OldCreateOrder = False Position = poScreenCenter DnCreate = FormCreate PixelsPerInch = 96 TextHeight = 13 object files: TListView Left = 10 Top = 10 Width = 726 Height = 111 ColumnS = < item Caption = 'File' Width = 696 end2 ColumnClick = False ReadOnly = True RowSelect = True PopupMenu = fileistpopup TabOrder = 0 ViewStyle = vsReport OnClick = filesClick end object savebut: TBitBtn Left = 660 Top = 510 Width = 75	
		4	Status child
Ready 7 sec.	exif2htm.exe	781312 bytes	TAN STRONG LA

发现有两个FORM,分别查看,发现1是主窗口,2是注册界面

DeDe 3.50.02 (c) 1999	-2002 by DaFixer			- 0 ×
File Dumpers Tools	Options About			
C:\Users\Random\Desktop	p\exif2htm.exe 👻 🧟	Process	exif2htm	
Charles halo	I Part		turne 1	
Classes into Units into	Forms Procedu	res Project	Expons	1.000
Class Name	Difset	The A		DFM Editor
TFom2		ect Form2 TFOrm2 FFOrm2 eft = 272 op = 188 ctiveControl = inp1 orderCons = [biSyst orderStyle = bsSing aption = Thegister' ientHeight = 171 ientWidth = 351 clore = cBUnFace ant Charset = DEFA ant Color = cRVindo ant Height = 13 ant Style = [] IdCreateOrder = Fal osition = poScreen(kelsPerinch = 96 estHeight = 13 cliticabelOrder = Fal osition = poScreen(kelsPerinch = 96 estHeight = 30 Height = 21 EditLabelWidth = 3 EditLabelWidth = 3 EditLabelWidth = 3 EditLabelHeight = 1 EditLabelHeight = 1 EditLabelCaption = TabOrder = 0 nd bject inp2: TLabeler Left = 25 Top = 80 Width = 300 Height = 21 EditLabelCaption = TabOrder = 1 nd bject BitBtn1: TBitB Left = 94 Top = 125 Width = 75 Height	emMenu] e ULT_CHARSET wText ns Seill' se Conter dEdt 1 3 "Name:" dEdt 8 3 "Code:" m ck n	
				吾常研究论片
Ready 7 sec.	exif2htm.exe	781312 byte	s	IVE POUNDARY

然后查看注册按钮的地址

DeDe 3 50,02	(c) 1999-2002 by DaFixer		- 0 X
File Dumpers	Tools Options About		
C:\Users\Random	NDesktop/exif2htm.exe	Process exif2htm	
Classes Info L	Units Info Forms Proceed	ures Project Exports	
Unit Name	Class Name	TForm2	DPR OFFS
Unit1 Unit2	TForm1 TForm2	Events Controls	
		Event	RVA Hint
			吉芬被拼论坛
Ready 7 sec.	exif2htm.exe	781312 bytes	William position of

然后使用OD来到4A0070的位置看看



设置断点, 让程序跑起来

004R006D	00	08 00	
0049006E	BBCØ	HOU ERK, ERM	kernel32.BaseThreadInitThunk
00460070	· 55	PUSH EBP	
00400072	. SBEU	NOD EEV EEV	
00400075	- 5507 E1	DICH EPV	
00400075	51	PUSH FCX	
86466677	. 51	PUSH ECX	
66496678	51	PUSH ECX	
00400079	. 51	PUSH ECX	
004A007A	. 53	PUSH EBX	
0049007B	. 56	PUSH ESI	and the second se
00480070	. 88D8	MOU EBX. EHX	kernel32.BaseThreadInitThunk
00400007.E	. 3300	AUK EHA, EHA	Kernelsz, Baseinreadinitinunk
66400881	68 53014000	PUSH avit2htm. 00400153	the second se
89469986	. 64: FE30	PUSH DWORD PTR ES: [E6X]	
804A0089	. 64:8920	MOU DWORD PTR FS: CEAX1.ESP	
00449009C	. 8D55 FC	LEA EDX, LLOCAL. 11	
0040008F	. 8883 F8020000	MOU EAX, DWORD PTR DS: LEBX+2F81	
68440095	. E8 7879FBFF	CALL exif2hth.00457A14	
UU4HUU9H	. 8370 FC 00	CMP (LUCHL.11.0	
0044009E	.V 0F84 8000000	LEO EDV (1000) 21	
08480884	9893 FC82888	MOU FOX DWORD PTR DS+ (FRX+2FC)	
88488880	. E8 6279FBFF	CALL exit2htn. 88457814	
094900B2	. 837D F8 00	CMP (LOCAL, 2], 0	
884R8886	.~ 74 78	JE SHORT exif2htm.004A0130	
804A00B8	. B2 01	MOU DL.1	
00400000	- H1 CC794100	MOU ERX, DWORD FTR DS: L4179CC]	
BB4NBBBF	. E8 2037F6FF	MOU EST EOV	kannal 22 PacaThurad In it Thurk
664000C4	8055 F4	LEG EDX. FLOCOL 31	Kernetoz, basemieduli termunk
69486829	8883 F8020000	MOU EAX, DWORD PTR DS: [EBX+2E8]	
884988CF	. E8 4079FBFF	CALL exif2htm.00457A14	
864A990D4	. 8855 F4	MOV EDX, [LOCAL. 3]	
66460607	. SBC6	MOV EAX, ESI	
00400000	· 8808	NUV ELX, DWORD PTR DS: LEAXI	
BBADGEDE	9055 F8	LEG EDV FLOCOL 41	
004000E1	8883 FC82888	MOU ERX, DWORD PTR DS: LEBX+2EC1	
654A68E7	. E8 2879FBFF	CALL exif2htn.00457A14	
004400EC	. 8855 F0	MOV EDX, LLOCAL. 41	
894RIBBEF	. 8BC6	MOU ERX, ESI	
UE4FIGUE1	· 8808	HUU ELX, DWORD PTR DS: [ERX]	
0044000F3	- PP15 58	MOLLEDY DUDED PTR DS+14095E01	av 162hts 98400059
BB46BBEC	8812	MOU EDX, DWORD PTR DS: CEDX1	an it circle ognicoo
004DODFE	8045 EC	LEA EAX, CLOCAL, 51	
864A0101	. B9 68014A00	MOU ECX, exif2htm.004A0168	ASCII "reginfo.dat"
00400106	. E8 A547F6FF	CALL exif2htm.00404880	and the second se
00490108	. 8855 EC	MOU EDX. CLOCAL 51	
BB49019E	· 9806	HOU SON DWOOD DTD DO FEOND	
B9409112	- 6808 FE51 74	COLL DWORD PTR DS: (ECY+741	
86408112	- 88C6	MOU FRX FSI	
00440117	. E8 F836F6FF	CALL ex (f2htm, 00403814	the second se
0049011C	. B8 7C014A00	MOV EAX, exif2htm.004A017C	ASCII "Please restart E2H to check your reg info "
00496121	 E8 BA1FF9FF 	CALL exif2htm,004320E0	
89440126	- H1 30AD4A00	MOV ERX, DWORD PTR DS: E4RAD301	

点击注册,输入信息后点击OK,OD断在了我们的断点处

C: 55 SBEC	HOU FRP. FSP				*	Regis
- 33009 - 51 - 51 - 51 - 51 - 51 - 51 - 51 - 51	XOR ECK, ECX PUSH ECX PUSH ECX PUSH ECX PUSH ECX PUSH ECX PUSH ECX PUSH EBX PUSH EBX PUSH ESI		exif2hti exif2hti exif2hti exif2hti exif2hti exif2hti exif2hti	₩, 00450D20 ₩, 00450D20 ₩, 00450D20 ₩, 00450D20 ₩, 00450D20 ₩, 00450D20 ₩, 00450D20		
- 88D8 - 33C0	MOU EBX, EAX				-	EIP 0
: 68 53014000	PUSH EBP PUSH ex If 21	Exif2htm UNREGISTERED)			
- 6418928 - 6418928 - 8055 FC - 8883 F8020000 - E8 7A79F8FF	MOU DWORD F LEA EDX, LU MOU EAX, DWO CALL ex 1621	File				
- 8370 FC 00 - 0F84 8C000000 - 8055 F8 - 8883 FC020000	JE exif2htr LEA EDK, LL MOU EAX, DW					
- E8 6279FBFF - 837D F8 00	CALL exif 2					
- 74 78 B2 01 B1 CC794100	MOU DL, I					
- E8 2037F6FF 88F0	MOV ESI, EA	Script editor HTML option	s			
. 8055 F4 . 8883 F8020000 F8 4079FPFF	MOV EAX, DWO		r			
8855 F4 98C6	MOU EDK. ILI	Tag	1	Register		
- 8808 - FF51 38	CALL DWORD			(1971)		
8883 FC820000	MOU EAX, DW			Name:	_	
. 8855 F0 . 88C6	MOV EDX. LL			R4ndom		
- 8808 - FF51 38	MOV ECX, DWO CALL DMORD		-	Code:		
. 8815 F8954800 . 8812	MOU EDX. DW	Par Newline Spa	ice I	12121212		
- 8045 EC - 89 68014800	MOV ECX, ex					- 11
· BB55 EC	MOU EDR. LL	Column Honz Der	ault 🔻	CK Cancel		
- 8808 - FF51 74	HOU ECK, DWO	Script editor:		Calicor		
: 88C6 : E8 F836F6FF	MOV EAX,ESI				-inter i de	and I
- 88 7C014000 - E8 BA1FF9FF	CALL exif2				= 1 10	1.11
	- 355 - 33C9 - 51 - 51 - 51 - 51 - 51 - 51 - 51 - 51 - 51 - 53 - 56 - 88D8 - 33C0 - 64: 530 - 64: 530 - 64: 5920 - 64: 5920 - 64: 5920 - 64: 5920 - 683 - 5002000 - 8055 FG - 883 - 5002000 - 8055 FA - 883 - 5002000 - 8055 FA - 882 - 500 - 8055 FA - 8855 FA - 8856 - 88808 - 751 74 - 8856 - 8808 - 751 74 - 8856 - 8808 - 751 74 - 8857 - 7014400 - 88 FA - 7014400 - 701450 - 701450 - 701450 - 701450 - 701450 - 701450 -	SBC HOU EBP, ESP 33C9 XOR ECK, ECX 51 PUSH ECX 53 PUSH ECX 54 S8D8 56 PUSH EEX 3300 XOR EAX, EAX 3300 XOR EAX, EAX 3300 XOR EAX, EAX 64:1730 PUSH ENX, EAX 64:1730 PUSH ENX, EAX 64:1720 PUSH EXX, EAX 9855 FF CHX, EAX, DUV 9870 FC 06 CHP LUCORL 9870 FC 06 CHP LUCORL 9870 FC 06 CHP LUCORL 9870 FC 80 CHP LUCORL 9870 FS 08 EA EDX, LU 9870 FS 08 CHP LUCORL 9870 FS 64 MOU CHX, LW 8055 F4 CHX LW/LW	SBEC HOW EBF, ESP 33C9 XOR ECK, ECX 51 PUSH ECX 53 PUSH ECX 54 PUSH ECX 55 PUSH ECX 56 PUSH ECX 57 PUSH ECX 5802 PUSH EC	SBEC HOV EBF, ESP 33C9 XOR ECX, ECX S1 PUSH ECX S2 PUSH ECX S3C9 PUSH EX S3C0 PUSH EX S3C9 PUSH EX S3C0 PUSH EX S3C0 PUSH EX S3C9 PUSH EX S3C9 PUSH EX S3C9 PUSH EX S3C9 PUSH OWORD (S415767 CHP CLOCAL S3270 FC 000 E S4279FBFF S3270 FC 000 CHP CLOCAL S4279FBFF CHP CLOCAL S4279FBFF CHL CAL, EX, DW S4279FBFF CHL CAL, EX, DW S655 F4 HOV EX, LL	BBBCC IOV EBF ESF EEX 33259 PUSH ECX PUSH ECX PUSH ECX 51 PUSH ECX PUSH ECX PUSH ECX 53 PUSH ECX PUSH ECX PUSH ECX 54 PUSH ECX PUSH ECX PUSH ECX 55 PUSH ECX PUSH ECX PUSH ECX 56 PUSH ECX PUSH ECX PUSH ECX 56 PUSH EXX PUSH EXX PUSH EXX 565 PUSH EXX PUSH EXX PUSH EXX 565 PUSH EXX PUSH EXX PUSH EXX 565 PUSH EXX PUSH EX	Second Seco

现在我们用OD进行分析,当运行到4a0095处时,我们能看到栈中显示的是我们的用户名,如果你已经步过了 就看不到了



这里对用户名做了什么,99%是做检查,以此作为是否注册的依据,然后继续走,来到4a00B2处,在信息窗口可以看到我们输入的密码或者序列号



与用户名呼叫同一个call,然后一直来到4A0101处

004R00EC 004R00EF 004R00EF 004R00EF 004R00EF 004R00EF 004R00EF 004R00EF	. 8855 F0 . 8866 . 8808 . FF51 38 . 8815 F0954800 . 8812	MOV EDX, ELOCAL.41 MOV EDX, ESI MOV ECX, DWORD PTR DS: [EAX] CALL DWORD PTR DS: [ECX+38] MOV EDX, DWORD PTR DS: [4A95F0] MOV EDX, DWORD PTR DS: [4A95F0]	exif2htm.004AAC58
004A00FE	. 8D45 EC	LEA EAX. [LOCAL.5]	ASCII "reginfo.dat"
004A0101	. B9 68014000	MOV ECX. ex if2htm.004A0168	
004A0106	. E8 A547F6FF	CALL ex if2htm.00404880	
004A0105	- 8855 EC	MOV EDX. [LOCAL.5]	Miner
004A0105	- 8866	MOV EAX, ESI	
004A0110	- 8808	MOV ECX, DWORD PTR DS: [EAX]	
004A0112	- FF51 74	CALL DWORD PTR DS: [ECX+74]	
004A0112	- 8866	MOV EAX, ESI	

这里看上去很重要,一个文件被创建了,步过两行来到4A010B处,发现在当前文件路径下创建了一个 Reginfo.dat文件。然后步过4A0112这个Call后,文件被创建了



使用txt文件打开这个文件



现在在OD中查找字串RegInfo.dat,能找到

00499837 MOU EDX, exif2htm.0049AE2C 00499837 MOU EDX, exif2htm.0049AE48 00499AB36 MOU EDX, exif2htm.0049AE5C 00499AB36 MOU EDX, exif2htm.0049AE5C 0049AB35 MOU EDX, exif2htm.0049AE30 0049AB34 MOU EDX, exif2htm.0049AE30 0049AD7F MOU EDX, exif2htm.0049AE30 0049AD7F MOU EDX, exif2htm.0049AE30 0049AD39 MOU EDX, exif2htm.0049AE7C 0049AE3C ASCII "Unregistered Use" 0049AE3C ASCII "Loregistered Use" 0049AE5C ASCII "http://o	ASCII "dd.mm.yyyy" ASCII "hh:nn:ss" ASCII "h.97 ASCII "I September 2008" ASCII "reginfo.dat" ASCII "reginfo.dat" ASCII "This is a shareware program, it means that you may try it free ASCII " UNREGISTERED"
---	---

在这里设置断点,重新运行程序,段了这里49AC00

06496683	. E8 947EE6EE	CALL exif2htm,00402040	
00490BB8	8845 F8	MOU FOX DWORD PTR SS+FERP-181	
RR490ERR	8055 EC	LEG EDX DUORD PTR SSICEPP-141	
00490PPF	ES ESECTET	COLL av 1425 FM 00409620	
00400000	ODEE EC	MOULEDY DWODD DTD CC. [EDD-14]	
00434603	· 0005 EC	HOU FOX DWORD FIR SSILEDF-141	
0049HBUB	. B8 58HC4H00	HUV EHX, EXITZATA, 004HHUS8	
RR4AHRCR	. E8 289HF6FF	LHLL 8X1+2htm.004045F8	
0043HBD0	. 8045 F4	LEA EAX, DWORD PIR SS: LEBP-CJ	
0049ABD3	 E8 CC99F6FF 	CRLL exif2htm.00404584	
0049ABD8	. 8D45 FØ	LEA EAX, DWORD PTR SS: [EBP-10]	
0049ABDB	 E8 C499F6FF 	CALL exif2htm.00404584	
0049ABE0	. B2 01	MOU DL, 1	
0049ABE2	. A1 CC794100	MOU EAX, DWORD PTR DS: [4179CC]	
0049ABE7	. E8 F88BF6FF	CALL exif2htm.004037E4	
0049REEC	. 8945 F8	MOU DWORD PTR SS: (EBP-81, EAX	
0049ABEF	. 3300	XOR EAX.EAX	
004968E1	55	PUSH EBP	
00490BF2	68 64904900	PUSH exif2htm,00490064	
66490BE7	64+FF30	PUSH DWORD PTR EST FERX1	
00490RE0	64:8920	MOU DWORD PTR ES (FOX1 FSP	
00490RED	8045 F4	LEG FOX DUORD PTR SSIFERR-101	
00490000	B9 900E4900	MOULECX av if 2htm 00490E90	OSCIT "regipto dat"
00490005	9915 59004099	MOLLEDY DWORD PTR DS+ [400C59]	Hours reguirorday
00400000	E9 00900000	COLL av i 625 th 00404900	
00494000	ODEE EA	MALL ENTERING OF PTP CONFERENCE	0042000C
004590010	· 0005 L4	MOH FOY DUODD DTD CC. (COD-01	ex17200430090
0049H015	. 0040 FO	MOUL FOX DWORD PTR DO LEOPTOJ	
004994016	. 0000	TOV ECA, DUORD FIR DSILEHAJ	
0049HC18	· FF51 68	CHEL DWORD FIR DS: LECATES	
0049HC1B	. 8845 18	TUO EHX, DWURD PTK SS: LEBP-81	
UU49HCIE	. 8810	MOU EDX, DWORD PTR DS: LEHXJ	
0049HC20	. FF52 14	CHLL DWORD PTR DS:LEDX+141	
0049HC23	. 83F8 02	UMP EHX, 2	
0049AC26	.~ 0F8C 2E010000	JL exif2htm.0049RD5A	
0049AC2C	. 8D4D E0	LEA ECX, DWORD PTR SS: [EBP-20]	
8049AC2F	. BA 0100000	MOV EDX, 1	
0049AC34	. 8B45 F8	MOV EAX, DWORD PTR SS: [EBP-8]	
8049AC37	. 8B18	MOV EBX, DWORD PTR DS: [EAX]	

滚动代码上下看看,看是否能发现有用的信息。

然后继续单步执行,来到49AC58

0049RC48 0049RC4B 0049RC4D 0049RC50 0049RC51 0049RC55	FF53 0C 8845 DC 58 E8 86F3FFFF 94C0	CALL DWORD PTR DS: [EBX+C] MOV EAX, DWORD PTR SS: [EBP-24] POP EDX CALL exif2htm.00499FDC TEST AL, AL	exif2htm.0041C110 0018FD94
0049AC58	.* 0F84 FC000000	JE exif2htm.0049AD5A	
80499C5E	. SD4D D8	LEA ECX, DWORD PTR SS: [EBP-28]	
9949HL61	. BH 01000000	MOU FOX DUODD PTP SS (FPP-91	
88499669	. 8B18	MOU EBX. DWORD PTR DS: (EAX)	and the second second
0049AC68	. FF53 0C	CALL DWORD PTR DS: [EBX+C]	exif2htm.0041C110
0049RC6E	. 8845 D8	MOV EAX, DWORD PTR SS: [EBP-28]	CUCKED CONTRACTOR
00498071	. ES FEF/FFFF	CHLL exif2htm.0049H474	
00490078	PESA DCOODOO	IE evif2htm.00490050	
0049AC7E	. C605 4CAC4A00 00	MOV BYTE PTR DS: [4AAC4C].0	
00498085	. 8D4D D4	LEA ECX, DWORD PTR SS: [EBP-2C]	
00498088	. 3302	XOR EDX, EDX	
99490090	- 8845 F8	MOULERY DWORD PTR DS (EDV)	and the second second
00499C8F	. FF53 0C	CALL DWORD PTR DS: LEBX+C1	exif2htm.0041C110

这里跳走了,看看跳到哪里去了



这里看上去不太好,不是我们想要的地方,我们知道怎么改让它不跳走,继续,来到49AC78

0049AC61 0049AC66 0049AC69 0049AC68 0049AC68 0049AC68 0049AC71 0049AC71	- BH 0100000 - 8845 F8 - 8818 - FF53 0C - 8845 D8 - E8 FEF7FFF - 84C0	MOV EAX, DWORD PTR SS: [EBP-8] MOV EAX, DWORD PTR DS: [EAX] CALL DWORD PTR DS: [EAX] CALL DWORD PTR DS: [EBX+C] MOV EAX, DWORD PTR SS: [EBP-28] CALL exif2htm.0049A474 TEST AL, AL	exif2htm.0041C110
0049AC78 0049AC78 0049AC85 0049AC88 0049AC88 0049AC88 0049AC88 0049AC89 0049AC95 0049AC95 0049AC95 0049AC95 0049AC47 0049ACA7 0049ACA7 0049ACA2	. 0F84 DC000000 C605 4CAC4A00 00 8040 D4 33D2 8845 F8 8818 FF53 0C 8855 D4 88 3CAC4A00 E8 3CAC4A00 E8 3CAC4A00 E8 3CAC4A00 8845 F8 8845 F8 8818 FF53 0C	JE exif2htm.0049AD5A MOV BYTE PTR DS:[4AAC4C],0 LEA ECX,DWORD PTR SS:[EBP-2C] XOR EDX,EDX MOV EAX,DWORD PTR DS:[EAX] CALL DWORD PTR DS:[EAX] CALL DWORD PTR DS:[EBX+C] MOV EDX,DWORD PTR SS:[EBP-2C] MOV EAX,exif2htm.00404SF8 LEA ECX,DWORD PTR SS:[EBP-30] MOV EDX,1 MOV EAX,DWORD PTR SS:[EBP-30] MOV EAX,DWORD PTR SS:[EBP-31] MOV EAX,DWORD PTR DS:[EAX] CALL DWORD PTR DS:[EAX]	exif2htm.0041C110 exif2htm.0041C110

这里还有一个跳转,看看到哪里的,和刚刚那个一样,跳到了不好的地方。继续不跳转一直走就会发现,启动 程序的弹窗没有了,标题上的未注册也没有了

现在我们能让这个程序使用任何用户名和密码进行注册了,我们成功了

cript editor HTML option	16	
Tag	Value	
Par Newline Sp	ace File name Full name Size B	KB MB Image
	Fault - Vort Default - 11	H Ba

到这里第十七章结束,作者的目的是教我们使用DeDe和ExeInfoPE这两个软件来分析Delphi写的程序,这样对新手来说比较容易。

当然我在没有看教程的时候就使用查找字串参考也能找到入口点,按照以往的教程知道怎么打补丁了。但是还 是按照作者的意图进行转述,作者描述的比较罗嗦一点。我就直接精简了,因为如果从第一章一直看下来的朋 友一定明白我说的是什么。

后续章节会抽时间继续,直到完成。

第十八章:时间限制与硬件断点之间的博弈

翻译都是我理解的方式进行描述,可能和原文不一致。

本教程中文版只在吾爱破解论坛 首发。

转载请注明来自吾爱破解论坛@52pojie.cn

正文开始

试用时间和内存断点

解释:使用时间通常来说是一个程序提供一段时间让用户使用,超过这段时间后程序将不再工作或者减少功能 工作,一般会给出30天的试用。当一个逆向工程师在破解这一类的程序时会去找到注册码,如果我们比较懒的 话,就找到程序中的试用时间,让这个时间无限话。

教程中,为了保护作者和软件,我下载的软件在CNET网站上的下载次数只有2次,估计是作者已经不再提供维护和支持了,本程序的原名为: Secrre Data-Hide a File into an image.exe,而我将它改为: SecureData.exe。而且不含附带的DLL文件,因此该程序并不能真正的工作。但是足够本教程的教学。

打开程序后会弹出窗口



显示了可以试用的次数,点击OK后来到主界面



点击About后

5 Secure Data - Hide	a file into an image		×
Features	Hide File	Get Hidden	About
SECUE Protect y Copyright © 2012 Aladd Secure Data version (1.0 Registered to IS NOT R Key R To get a registration key Pu Help	RE DATA your data using a in Software. All rights re o) REGISTERED YET egister y please purchase a key urchase o contents Help	secure means	

输入Key后,点击Register出现

😓 Secure Data - Hide a	a file into an image		
	ET'		
Features	Hide File	Get Hidden	About
SECUR Protect y Copyright © 2012 Aladdi Secure Data version (1.0 Registered to IS NOT R Key 12121212 R To get a registration key Pu	AE DATA Your data using a Secure Data - In Secure Data - In The er EGI EGI EGI EGI EGI EGI EGI	secure means valid Key ntered key is invalid. Please	purchase a key
Help	contents		1
	Help		
			日安岐Webras Advertige

好了,目前我们已经知道程序的大致流程了

让我们用OD来打开程序

408684	\$ E8 304A0000 CALL SecureDa.00400009	
4086A9	.^ E9 16FEFFFF JMP SecureDa.004084C4	
14086HE	> 55 PUSH EBP	
4086R1	SIEC 28030000 SHE ESP 328	a second a second se
408687	. A3 C3D64100 MOV DWORD PTR DS: [41D6C8]	.EAX kernel32.BaseThreadInitThunk
4086BC	. 890D C4D64100 MOV DWORD PTR DS: [41D6C4]	I,ECX
108602	. 8915 C0D64100 MOV DWORD PTR DS: [41D6C0]	J.EDX SecureDa. <moduleentrypoint></moduleentrypoint>
H086C8	. 891D BCD64100 MOV DWORD PTR DS: [41D6BC]	I,EBX
1026D4	. 8935 B8054100 NOU DWORD PTR DS: 1410686.	L SI
198609	66:8015 F0064101 MOU WORD PTR DS: [4106F0]	S8
1086E1	. 66:8000 D4D6410 MOV WORD PTR DS: [41D6D4]	ČŠ
1086E8	. 66:8C1D B0D6410(MOV WORD PTR DS: [41D6B0]	DS
4086EF	. 66:8005 ACD6410 MOV WORD PTR DS:[41D6AC],	ES
1086F6	 66:8025 R806410 MOV WORD PTR DS: [4106R8], 66:8020 0406410 MOU WORD PTR DC: [410604] 	FS
198794	90 PUSHED	.05
408705	. 8F05 D8D64100 POP DWORD PTR DS: [41D6D8]	kernel32.77E5E06C
10870B	. 8845 00 MOV EAX, DWORD PTR SS: LEB	2]
40870E	. AS CCD64100 MOV DWORD PTR DS: [41D6CC]	J.EAX kernel32.BaseThreadInitThunk
08713	. 8845 04 MOU ERX, DWORD PTR SS: [EB	+4] ntdll.77D3377B
400710	- HS D0064100 HOV D00HD FTH DS:L410600	J, EHA Kernel32. BaseInreadInitinunk
40871E	. 93 DCD64100 MOU DWORD PTR DS: [41D6DC]	LEAX kernel32. BaseThreadInitThunk
08723	. 8885 E0FCFFFF MOV EAX, [LOCAL, 200]	
08729	. C705 18D64100 0 MOV DWORD PTR DS: [41D618.	1,10001
108733	. A1 D0D64100 MOU EAX, DWORD PTR DS: [41]	06003
408738	. H3 CCU54100 MOU DWORD PTR DS: [4105CC]	EHX kernel32.BaseThreadinitThunk
498747	. C705 C4054100 0 MOU DWORD PTR DS: C4105C4	

搜索字串后,我们可以看到

R Found strings are	
Address Disassembly	Text string
HOUTES DISASSENDU 00406501 PUSH SecureDa.004 004050550 PUSH SecureDa.004 00405851 PUSH SecureDa.004 00405850 PUSH SecureDa.004 00405850 PUSH SecureDa.004 004058500 PUSH SecureDa.004 00405500 PUSH SecureDa.004 00405508 PUSH SecureDa.004 00405508 PUSH SecureDa.004 00405508 PUSH SecureDa.004 00405508 PUSH SecureDa.004 00406508 PUSH SecureDa.004 00406507 PUSH SecureDa.004 00406500 HUU ECX, SecureDa.004 00406500 PUSH SecureDa.004 00406500 PUSH SecureDa.004 00406507 PUSH SecureDa.004 00406507 PUSH SecureDa.004 00406500 PUSH SecureDa.004 00406579 PUSH SecureDa.004 004066779 PUSH SecureDa.004 004067178 PUSH SecureDa.004 00406718 PUSH SecureDa.004 00407138 PUSH SecureDa.004 00407138 PUSH SecureDa.004 00407255 PUSH SecureDa.	<pre>intext string UNICODE "OK" UNICODE "Software\Windows Data Count\\data" UNICODE "Secure Data" UNICODE "Secure Data" UNICODE "Software\Windows Data Count\\data" UNICODE "Software\Windows Data Count\\data" UNICODE "data flag" UNICODE "flag" UNICODE "Please select a file" UNICODE "Please select a file" UNICODE "Please select a file" UNICODE "Please select an image" UNICODE "Please select an image" UNICODE "Please select where you want to save the new image file" UNICODE "Data flag" UNICODE "Bitmap Image File" UNICODE "Please select the image that contains your hidden data" UNICODE "Please select the image that contains your hidden data" UNICODE "Please select the image that contains your hidden data" UNICODE "Please select the image that contains your hidden data" UNICODE "These select the image that contains your hidden data" UNICODE "These Select a file" UNICODE "These Select the image that contains your hidden data" UNICODE "These Select the image that contains your hidden data" UNICODE "These Select the image that contains your hidden data" UNICODE "These Select the image that contains your hidden data" UNICODE "These Select the image that contains your hidden data" UNICODE "These Select the image these Select the image Select the image Select the image Select the Secure Data Help.chm" UNICODE "Secure Data Help.chm" UNICODE "Secure Data Help.chm" UNICODE "Secure Data Help.chm"</pre>
*	the second s

我们没有看到和试用信息一样的字串,但是注意红色框

试用期

逆向工程师必须要知道的一个重要信息,在程序中必须要保存试用期的次数或者天数,这样来说试用期一定是 保存在什么地方,通常情况下是保存到文件或者硬件中。

大多数时候,保存这个数据的地方很容易找到,只是逆向工程师会把自己往复杂的方向去想。最简单的方法就 是看字串和搜索注册表或者文件路径。比如:

Software\\AppName\\Key

或者

AppName\DataFileName.ini or AppName\DataFileName.dat

多数会采用引用window系统变量,如%WINDOWS%,这表示在Windows目录。

当然,如果程序不是很大,你还可以一个Call一个Call的看。或者注意一下windows API,CreateFileExA,

RegSetValueExA等等,确保数据是保存到文件还是保存到注册表中。而本程序使用了2套做法,在硬盘上创建 了文件并且将它隐藏。

看会字串搜索结果

R Found strings are		
Address Disassembly Tex	t string	
Address Disassembly Tex 00404D71 MOU EBX, SecureDa. 004 UN1 00404EA4 PUSH SecureDa. 004 UN1 00404EA4 PUSH SecureDa. 004 UN1 00404EC PUSH SecureDa. 004 UN1 004050E7 PUSH SecureDa. 004 UN1 00405915 PUSH SecureDa. 004 UN1 00405915 PUSH SecureDa. 004 UN1 004055912 PUSH SecureDa. 004 UN1 004055913 PUSH SecureDa. 004 UN1 004055814 PUSH SecureDa. 004 UN1 004055812 PUSH SecureDa. 004 UN1 004055813 PUSH SecureDa. 004 UN1 004055814 PUSH SecureDa. 004 UN1 004055819 PUSH SecureDa. 004 UN1 004055819 PUSH SecureDa. 004 UN1	t string CODE "SecureData" CODE "Encrypting data" CODE "The hiding process was done successfully" CODE "OK" CODE "Decrypting data" CODE "Decrypting data" CODE "Decrypting data" CODE "Decrypting data" CODE "Decrypting data" CODE "The saving process was done successfully" CODE "OK" CODE "OK" CODE "Software\\Windows Data Count\\data" CODE "Secure Data" CODE "data flag" CODE "data flag" CODE "data flag" CODE "data flag" CODE "data flag" CODE "data flag" CODE "Software\\Windows Data Count\\data" CODE "Software\\Windows Count\\data" CODE "Soft	ly %d runs,
0040656D PUSH SecureDa.004 UN 00406770 PUSH SecureDa.004 UN 00406778 PUSH SecureDa.004 UN	CODE "All Files" CODE "Please select an image" CODE "Images of type ipg and bmp"	

这里不会讲解注册表的树形结构,我们直接打开注册表,使用Window +R键,输入Regedit,打开注册表

File Edit View Favorites Help		-		
Computer	Name	Туре	Data	
HKEY_CLASSES_ROOT HKEY_CURRENT_USER HKEY_LOCAL_MACHINE HKEY_USERS HKEY_CURRENT_CONFIG	(Default)	REG_SZ	(value not set)	
	*		m	

然后打开HEKY_LOCAL_MACHINE,找到字串Windows Data Count,然后可以看到



我们看到Data Flag显示为7,我们把这个值改为100

Edit DWORD (32-bit)	Value 🗾
Value name:	
data flag	
Value data:	Base
100	C Hexadecimal
1	Decimal
	OK CANARA CO
	- www.sabable.co

然后保存后显示



现在重新打开程序

Secure D	ata - Unregistered Copy
5	You can freely use and test the Secure Data software for only 99 runs, and after that the software will ask you to enter a registration key
	OK 書意報酬金之 2000年3月2月1日

看上去修改成功了,这比想象中容易

分析程序

另一种方式就是我们通过修改程序代码达到同样的效果。这样做的好处就是不用每次快到时间就去修改注册 表。记住,当我们运行程序的时候请记住还剩下的次数,让我们去看看这个地方程序到底做了什么,首先这个 字串开始于406078



这是一大段程序,我们可以看到这是windows程序的初始化消息



这里先把一个ID为0x41D的窗口句柄作为GetDlgItem的参数,让我们用Resource Hacker看看资源

Resource Hacker - C	\Users\Random\Desktop\SecureData.exe	
File Edit View Actio	n Help	
E — ☐ Bitmap E — ☐ Icon □ — ☐ Dialog	Compile Script Hide Dialog	
Image: Constraint of the system 153 Image: Constraint of the system 1033 Image: Constraint of the system 1001 Image: Constraint of the system 1002 Image: Constraint of the system 1003 Image: Constraint of the system 1004 Image: Constraint of the system 1005 Image: Constraint of the system 1007 Image: Constraint of the system 1009 Image: Constraint of the system 1011 Image: Constraint of the system Version Info Image: Constraint of the system 24	STYLE DS FIXEDSYS DS MODALFRAME WS_POPUP WS_CAPTION CAPTION "Secure Data - Unregistered Copy" LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US FONT 8, "MS Shell Dlg", FW_NORMAL, FALSE, 1 { CONTROL "OK"_ 1052 RUITON_ BS_DEFPUSHBUTTON WS_CHIL CONTROL "text", 1053, STATIC, SS_LEFT WS_CHILD WS_CONTROL 144, -1, STATIC, SS_ICON WS_CHILD WS_VISIE }	D WS_VISIBLE WS_TABSTOP. 1 VISIBLE WS_GROUP, 35, 7, 212 SLE, 7, 7, 20, 20
	238	

查看GetDlgItem的用法后,我们把断点设在40604C处,然后重启程序,来到我们的断点处,单步分析,发现GetDlgItem的返回值保存到EDI中。

下一条是把内存418FC4的数据放到ESP+C中,跟踪这个地址我们看到数据是403980.如果你继续跟踪就会发现这是一个Callback。我们假设这是一个对话框。

Address	Hex dump ASCII
00418FC4 00418FD4	80 39 40 00 54 00 65 00 6D 00 70 00 6F 00 72 00 C90.T.e.m.p.o.r. 61 00 72 00 79 00 46 00 69 00 6C 00 65 00 00 00 a.r.y.F.i.l.e
00418FE4 00418FF4	54 00 65 00 6D 00 70 00 6F 00 72 00 61 00 72 00 T.e.m.p.o.r.a.r. 79 00 46 00 00 00 6C 00 65 00 2E 00 62 00 6D 00 y.F.i.l.eb.m.
00419004	70 00 00 00 00 50 00 72 00 55 00 70 00 61 00 72 00 pP.r.e.p.a.r. 69 00 6E 00 67 00 20 00 64 00 61 00 74 00 61 00 i.n.g. d.a.t.d.
00419034 00419034	63 00 75 00 72 00 65 00 44 00 61 00 74 00 61 00 c.u.r.e.D.s.t.s.

再往下两行,把内存41E084保存到eax中,而这个值为8,这个值和试用次数是匹配的。这就是我们要找的。

最后,我们在去看看字串

UNICODE "You can freely use and test the Secure Data software for only 2d runs, and after that the software w" user32.75FDF212

这个看上去是C语言中格式化,如果你熟悉C/C++语言的话。

printf("My IQ is a whopping %d", 18);

我们就看看这个%d是在那里被赋值的,来到

00406881 00406082 00406088 00406088 00406088 00406085 00406085	- 51 - E8 C9E3FFFF - 887424 1C - 83C4 0C - 56 - 57 - 515 64724199	PUSH ECX CRLL SecureDs.00404450 MOV ESI DWORD PTR SS:CES ADD ESP.0C PUSH ESI PUSH EDI PUSH EDI PUSH EDI	P+1C1	Text = 0006049C ??? hUnd = 00080344 ('te:	xt',class='Static',parent=0006045	-
* Stack SS:[(ESI=000604)	0018F444]≕02131700, 9C	(UNICODE "You can freely	use and test th	e Secure Data software fo	r onl 8 runs, and after that the	5
Oddress He	ex dump.		LIGSCIT.	1	0018F428	Sam

然后程序跑起来,我们看到试用信息,点击ok按钮后来到主界面。

打补丁

你先想到的是为什么不直接在地址406072的地方把试用次数改大,比如

MOV EAX, DWORD PTR DS:[41E084]

改为:

MOV EAX, 99

但是不能这样做,这样做虽然程序会显示试用次数很大,但是其他地方会检测到数据小于1时,程序将不再工作,所以我们必须要找到程序去改变这个值的地方,在程序修改后我们再进行打补丁(修改)。

所以我们要在这里设置硬件断点



就是这个内存被写入的时候断下来

00406072		Binary		L41E0841				
00406078 0040607C	Label		:	CESP+101	UNICODE "Yo	ou can freely use and t ~7212		
00406082 00406087	Breakpoint		•	Memory, on acces	s			
00406088 0040608E 0040608F 00406090 00406096 00406096 00406098		Search for Find references	Ctrl+R	Memory, on write Hardware, on acce	092 ('text',class='51			
004060A0 004060A2		View executable file		Hardware, on write	e 🔸	Byte		
004060A3 004060A8		Copy to executable file		Hardware, on exec	ution	Word		
004060AD 004060B2		Go to	,	1		Dword		
004060B8 004060B8	~	Hex		Incore.	Case 111 ()	Contractor of Switcon		
0040606F 004060C1 004060C5		Text		1060E5				
004060C7 004060CB		Short		1060ES				
004060D1 004060D3		Long		1060E5				
004060D7 004060D9		Float		1060E5				
004060DD 004060DF 004060E3		Disassemble		14060E5	1			
004060E5 004060E7 004060E8		Special		USER32.EndDialog>]	Result = 0 hWnd = 0001108E ('Secure Data EndDialog			
<		ILLY						
DS: E0041E08 EAX=0001105		CheckVmp						
Address He	Appearance A 10 01 00 01 00 00 00 00 00 00 00 00 00			ASCII				
0041E094 6A 0041E0A4 00 0041E0B4 00				00 00 j)0.0d)0 00 00 00 00		101-101-00 000 000 000 000 000		

现在我们只剩下6次试用次数了,我们设置了硬件断点,不论在任何地方对这个地址执行写操作,我们都会断

下来。

重启程序, OD会断在我们设置的硬件断点处



看图片顶部,我们可以看到注册表不是创建一个键值,而是打开。然后判断是否能打开,如果不能打开则跳到 坏消息处。这就说明需要管理员进行操作,如果不出错405A96的call就会把值读取出来保存到ESP+C中,如果 有第二个则保存到ESP+8

00403521	000r	HOU FOR DUODD DTD DC- FECTA	
00403530		HUV ECA, DWURD FIR DELEGIJ	And the second sec
00403532	. 83EC 08	SUB ESP, 8	<pre>(Httempt to open key</pre>
00403535	. 57	PUSH EDI	
00403536	. 8B3D 14704100	MOV EDI, DWORD FTR DS: [<&ADVAPI32.RegQueryV	advapi32.RegQueryValueExW
0040353C	. 6A 00	PUSH 0	CpBufSize = NULL
0040353E	. 68 00	PUSH 0	Buffer = NULL
09493549	8D4424 0C	LEG EGX, DWORD PTR SSt [ESP+C1	
00402544	50	PUSH EOV	allalueTupe - 0000001
GOAGOE AE	60 00	DUCH 0	Deservised - NULL
00400540	- OH 00		Heserved - HOLL
00403547	. 68 E4914100	PUSH SecureDa.004191E4	Valuename = "data flag"
0040354C	. 51	PUSH ECX	hKey = 76882020
0040354D	• FFD7	CALL EDI	RegRueryValueExW
0040354F	. 8500	TEST EAX, EAX	< Is there an error?
00403551	.~ 74 09	JE SHORT SecureDa.0040355C	(No. so keep going
00403553	> 3300	XOR EAX. EAX	(ues, so bug out
00403555	SE	POP EDI	SecureDa, 00405098
00403556	8304 88	ODD ESP 8	cecur course loone
004000000	02 0400	DCTN A	and the second se
004035557	> 02 0400	MOULEON A	
00403550	2 68 64666666	CMD DUODD DTD CONTECODIAL FOU	And there a making analy
00403561	. 374424 04	CHP DWORD FIR SSILESF43, EHA	<pre>< wrong return type:</pre>
00403565	. 75 EU	UNC SHURT SecureDa, 00403553	< yes, so bug out
00403567	. 8B0E	MOU ECX, DWORD PTR DS: LESIJ	
00403569	. 8D5424 08	LEA EDX, DWORD PTR SS: [ESP+8]	< 18FE68
00403560	. 52	PUSH EDX	
0040356E	. 894424 0C	MOU DWORD PTR SS:[ESP+C],EAX	<pre>< 18FE68 = length of buffer (4)</pre>
00403572	. 8B4424 14	MOV EAX. DWORD PTR SS: [ESP+14]	< 18FE70
00403576	. 50	PUSH ERX	<pre>< 18FE7C = buffer for return value</pre>
00403577	60 00	PUSH 0	
00403579	60 00	PUSH 0	and the second sec
0040957P	69 E4914100	PUSH SecureDa 004191F4	UNICODE "data flag"
00403570	. 60 24714100	DICU CCV	kannaloo 7600000
00403500	· 51	COLL EDT	Kernersz.robbzozb
00403561	· FFDC	UHEL EDI	A HOPPING - Sharpen Particle 1944
00403503	· F708	REG EHX	< ISPERC = number trials left
00403585	. 1800	SBB EHX, EHX	and a second a second a second a second
00403587	. 8300 01	HOD EHX, 1	
0040358R	. 5F	POP EDI	SecureDa.00405A9B
0040358B	. 83C4 08	ADD ESP, 8	
0040358E	L. C2 0400	RETN 4	< # trials left = ESP + C
80493591	00	INT3	and a second beat of the second
00100500		1.5450	

我已经分析了代码并给了注释,建议你自己分析代码

我们发现这个值(6)已经搬到了405AAD内存里面了

2Da. 00405ABD PTR SS: [ESP+8]
DS: [41E084], EAX
DS: [41E088], ECX
reDa.00405AFC PTR SS:[ESP+4]
PTR \$8:[E8P+10]
UsiveTypy = RESIDNORD

最后,我们在检查一下其他值,然后关闭注册表控制。

00400000	. 30 00000000		
00405805	. 5E	POP ESI	
00405806	.~ 74 45	JE SHORT SecureDa.00405B4D	
00405B08	. 3D 05000080	CMP EAX, 80000005	
00405800	.~ 74 3E	JE SHORT SecureDa.00405B4D	
00405B0F	. 3D 01000080	CMP EAX, 80000001	
00405B14	.~ 74 37	JE SHORT SecureDa.00405B4D	
00405B16	. 3D 02000080	CMP EAX, 80000002	
0040581B	.~ 74 30	JE SHORT SecureDa.00405B4D	
00405B10	. 3D 03000080	CMP EAX, 80000003	
00405B22	.~ 74 29	JE SHORT SecureDa, 00405B4D	
00405824	. 50	PUSH EAX	CWKey = 00000005
00405B25	. FF15 08704100	CALL DWORD PTR DS: [<&ADVAPI32.RegCloseKey>	RegClassifier
00405B2B	. 83C4 0C	ADD ESP, OC	Concernance and the second
0040582E	. C3	RETN	and a substant descent of the
8846582F	> 6A 30	PUSH 38	Style = MB_OKIMB_ICO

那么问题来了,哪个位置是做补丁最好的地方呢?回看代码,发现在405ADD被赋值后就没做过任何变化了。 如果我们把405AAD修改为MOV DWORD PTR DS:[41E084],FF,这会导致程序代码不正常。因为会被删除下 两条命令。

00405AA3 00405AA5 00405AAA	- 33C9 - 3D 40010000 - 0F94C1	XOR ECX, ECX CMP EAX, 140 SETE CL
00405AAD	C705 84E04100 FF	MOU DWORD PTR DS: [41E084], 0FF
00405887	90	NOP
904958B9	90	NOP
00405ABA	90	NOP
00405HBB	V EB SF	MOU FOX ENORD FTP CC. FCCP.41
00405A6D	69 04	PUSH 4
00405AC3	. 805424 10	LEA EDX, DWORD PTR SS: LESP+164
00405AC7	. 52	PUSH EDX
00405AC8	. 6H 04	PUSH 4

那么就在前面405A9F处进行修改吧

00405A6E 00405A96 00405A9B 00405A9D 00405A63 00405A63 00405A65	. C74424 0C FFFFF . E8 95DAFFFF . 85C0 .~ 74 1E . 884424 08 . 33C9 . 30_40010000	R MOV DWORD PTR SS:[ESP+C],-1 CALL SecureDa.00403530 TEST EAX,EAX JE SHORT SecureDa.00405ABD HOV EAX,DWORD PTR SS:[ESP+8] XOR ECX,ECX CMP EAX,140	SecureDa.00403530
00405AAA 00405AAD 00405AB2 00405AB2 00405AB5 00405AB5 00405AB5 00405AB5	. 0F94C1 A3 84E04100 83C1 01 890D 88E04100 .✓ EB 3F > 884424 04	Assemble at 00405A9F	
00405AC1 00405AC3 06405AC7 00405AC8 00405AC8 00405ACC 00405AC1	- 6H 04 - 8D5424 10 - 52 - 6A 04 - 6A 00 - 68 E4914100 - 68 E4914100	Fill with NOP's Assemb	ble Cancel "data flag"
00405AD6 00405AD7 00405ADB 00405AE1 00405AE5 00405AE5	- 50 - 397424 24 - FF15 0C704100 - 384C24 04 - 51 - FF15 04704100	PUBH EHA MOV DWORD PTR S3:[ESP+24].ESI CALL DWORD PTR DS:[<&ADVAPI32.RegSetValue MOV ECX.DWORD PTR S5:[ESP+4] PUSH ECX CALL DWORD PTR DS:[<&ADVAPI32.RegFlushKey	EL RegSetValueExM Chkey = 1E580000 ChegFlushKey

然后我们可以看到步过这个命令后,内存41E084的值变成FF了

	_		_						_		_
Address	He	t d	Imp							-	
0041E084	FF	00	00	00	90	00	00	00	00	00	8
0041E094	00	00	00	00	00	68	00	00	99	00	8
0041E0A4	00	100	00	00	99	00	00	00	80	00	8
0041E0B4	00	0.	00	00	00	00	00	00	00	00	8
0041E0C4	00	00	10	00	00	00	00	00	00	00	0
0041E0D4	00	00	00	00	00	00	00	00	88	00	6
0041E0E4	00	00	00	1.0	00	14	199	間的	-89	10	6
0041E0F4	00	00	00	00	00	00	00	00	00	00	6
0041E104	00	00	00	00	00	38	60	00	100	99	6
9941F114	00	88	NO.	NN	RR.	RR.	N P	RR.	99	PP	P

然后跑起来,试用信息为:

Secure Data - Unregistered Copy	
You can freely use and test the Secure Da 255 runs, and after that the software will a registration key	ta software for only ask you to enter a
ОК	普查爾爾伯德國

现在我们的程序不管打开多少次,这里总显示为255了。

虽然说去掉NAG弹窗,直接进行破解是最好的,但是我会结束这次的教程。不去做破解的原因有2,1是通过逆向,你知道程序工作原理以及设计程序时避免这样的检测。2有时你不能破解一个程序,对你来说是下一个好事。
第十九章:打造补丁生成器

翻译都是我理解的方式进行描述,可能和原文不一致。

本教程中文版只在吾爱破解论坛 首发。

转载请注明来自吾爱破解论坛@52pojie.cn

正文开始

此次教程主要讲述补丁程序,补丁程序就是在程序中找到了补丁位置以及补丁方式,不修改程序,通过制作补 丁程序来达到我们想要的效果。(注册或者显示好消息等),逆向工程师就会把这些补丁放到另一个副本中。 通常情况下补丁程序是个小程序,是用来修改原始程序的,当运行补丁程序后,会去按照你制作的补丁对原始 程序进行补丁,然后原始程序就能按照补丁后的效果运行了。

举个例子,现在你在网上下载了一个有试用期的程序,当试用结束时,你找到的补丁并且应用后,试用信息没 有了。通过这次的教程我会教你怎么制作补丁程序,然后使用补丁程序精准的打上补丁,这样你就可以把这个 补丁程序放到网上,其他用户则只需要到软件官方下载程序,使用你的补丁程序进行打补丁后,程序就能正常 使用了。

另一种类似补丁程序是把程序进行加载,在没有打补丁的程序上是不能进行加载的。

此次教程中我们进行补丁程序的名字叫Saturday Night Creackme。这个程序进行破解是很简单的,我将用 dUP2进行补丁程序制作,还需使用到CFF Explorer。当然最后我会提供这两个程序。

打开程序

当我们打开程序后,我们可以看到一个彩色图像

如果有音箱,请打开,程序会进行音乐播放。







然后点击 dance!,我们可以看到



对程序进行补丁,使用OD打开

Paused			Wi Ha Cp Pa St Br Re Tr Sr C 1?
C CPU -	main thread, module	Saturday	
C CPU - 00401002 00401002 00401002 00401002 00401002 00401019 00401019 00401019 00401029 00401029 00401029 00401029 00401029 00401035 00401044 00401045 0040105 0040005 0040005 0040005 00005 0005 0005 0005 0005 0	main thread, module \$ 6A 00 A3 AB634000 A3 AB634000 A9 S0 E8 14070000 A3 AF634000 A9 S0 E8 14070000 A3 AF634000 A8 S97F0000 A9 B97F0000 A9 B97F0000 A3 B9634000 A9 B9634000	Saturday PUSH 0 CALL <ump.&kerne132.getmodulehandlea> MOV DWORD PTR DS:[4063AB],EAX PUSH 3 PUSH EAX CALL <ump.&user32.loadloona> MOV DWORD PTR DS:[4063AF],EAX PUSH 7F89 PUSH 0 CALL <ump.&user32.loadcursora> MOV DWORD PTR DS:[4063B3],EAX PUSH 0 CALL <ump.&gd132.createsolidbrush> MOV DWORD PTR DS:[4063B5],EAX PUSH 0 CALL <ump.&gd132.createsolidbrush> MOV DWORD PTR DS:[4063C3],EAX PUSH 0 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 6 PUSH 1 PUSH 6 PUSH 1 PUSH 6 PUSH 6 PUSH 1 PUSH 6 PUSH</ump.&gd132.createsolidbrush></ump.&gd132.createsolidbrush></ump.&gd132.createsolidbrush></ump.&gd132.createsolidbrush></ump.&gd132.createsolidbrush></ump.&gd132.createsolidbrush></ump.&user32.loadcursora></ump.&user32.loadloona></ump.&kerne132.getmodulehandlea>	Context Contex
00401068	. A3 C7634000	HOU DWORD PTR DS: [4063C7], EAX	kernel32.BaseThreadInitThunk

查找字串参考

R Found strings are	
Address Disassembly 00401000 PUSH 0 004012E1 PUSH Saturday.004 004012EE PUSH Saturday.004 004012FD PUSH Saturday.004 004013EA PUSH Saturday.004 004013EA PUSH Saturday.004 004013EA PUSH Saturday.004 00401666 PUSH Saturday.004 00401603 PUSH Saturday.004 00401603 PUSH Saturday.004 00401604 PUSH Saturday.004 00402325 ASCII "-: de".0 00402338 ASCII "-: de".0 00402340 ASCII "-: de".0	Text string (Initial CPU selection) ASCII "Saturday Night Crackme - by haik!r" ASCII "About" ASCII "Saturday Night Crackme - by haik!r\n\r ASCII "." ASCII "Jalid ! You win :]" ASCII "Julid ! You win :]" ASCII "Invalid ! Maybe think about MP3Stego"
*	

可以看到好消息和坏消息

看下从什么地方来到了坏消息处。

004016B1	A EB EC	LUMP SHORT Saturday.0040169F	
00401605	7 01FB 00000027	NZ CUODT Caturday 00401604	
00401605	01EE E0040000	CMP EQT 4E0	
00401666	- 75 11	INZ SHOPT Saturday 00401604	
00401602	69 EC614000	PUSH Saturday, 004061EC	FTeut - Muslid & You win 17"
00401608	60 08	PICH 9	
00401600	EE75 08	PUSH TOPP 11	hild = ZEEDE000
00401600	E8 7400000	COL (MP. Suser32, SetDialtenTevt0)	SetDigItenTevt0
00401602	- EB 11	MP SHORT Saturday 004016F5	- De optigi venten ve
00401604	3 468 64614000	PUSH Saturday, 00406104	Flour - "Invalid & Maube think about MP3S
00401609	68 98	PUSH 8	ControlID = 8
994916DB	. EE75 08	PUSH FORG. 11	hlind = 7FEDE000
004016DE	E8 63000000	CALL (JMP.&user32.SetDigItemTextA)	SetDigItemText8
004016E3	. EB 00	JMP SHORT Saturday, 004016E5	
004016E5	> 5B	POP EBX	kernel32.760DED6C
004016E6	. 5E	POP ESI	kernel32.760DED6C
004016E7	. 5F	POP EDI	kernel32.760DED6C
004016E8	. C9	LEAVE	
004016E9	L. C2 0400	RETN 4	and the second
004016EC	\$- FF25 A4504000	JMP DWORD PTR DS: E<&user32.CallWindowProcA	user32.CallWindowProcA
004016F2	\$- FF25 90504000	JMP DWORD PTR DS: [<&user32.DialogBoxParamA	user32.DialogBoxParamA
004016F8	\$- FF25 94504000	JMP DWORD PTR DS:[<&user32.DrawTextA>]	user32.DrawTextA
004016FE	\$- FF25 98504000	JMP DWORD PTR DS:[<&wser32.EndDialog>]	wser32.EndDialog
08401784	\$- FF25 9C504000	JMP DWORD PTR DS: [<&user32.FillRect>]	user32.FillRect

找到这个跳转

UU4U1535 00401535 00401537 00401537 00401538 00401539 00401539 00401530 00401541 00401554 00401544 00401551 00401554	\$ 55 . 88EC . 57 . 56 . 53 . 6A 14 . 68 90624000 . 6A 08 . FF75 08 . E3 CB010000 . 8035 90624000 . 8035 90624000 . 83F8 11 . 0F85 7A010000	MOV EBP, ESP MOV EBP, ESP PUSH EDI PUSH ESI PUSH EBX PUSH 14 PUSH Saturday.00406290 PUSH 3 PUSH 14 EAL (JMP.&user32.GetDigitemTextA) LEA ESI,DWORD PTR DS:[406290] CMP EAX,11 JN2 Saturday.004016D4	Count = 14 (20.) Buffer = Saturday.00406290 ControlID = 8 hWnd = 7FFDE000 GetDigitemTextA
8849155E 9049155E 80491561 88491567 90491567 88491568 88491568 88491566 90401572 88491572 88491578	 ØFB646 09 83F8 2D 9856 6D010000 33C9 33D2 ØFB61C31 83FB 40 ØF8D 5C010000 83FB 30 ØF8E 53010000 	MOUZX EAX,BYTE PTR DS:[ESI+9] CMP EAX,2D UNZ Saturday.004016D4 XOR ECX,ECX XOR EDX,EDX MOUZX EBX,BYTE PTR DS:[ECX+ESI] CMP EBX,40 UGE Saturday.004016D4 CMP EBX,30 ULE Saturday.004016D4 CMP EBX,30	Saturday. <moduleentrypoint></moduleentrypoint>

在这个跳转上设置一个断点, 让程序跑起来, 断下来, 后面的请自己分析, 我这里直接跳转到好消息4016C3 处。



应用这个补丁,让程序跑起来,我们成功了



好了,我们要使用dUP2进行补丁程序制作。

dUP2和其他补丁程序不一样,它只有两种方式进行补丁,1是偏移补丁,2是搜索和替换补丁。偏移补丁:你 要知道补丁相对程序的偏移地址,然后进行补丁。使用OD可以很容易找到补丁位置,然后进行补丁。所以我 要找到在OD中补丁位置与文件起始地址的偏移值,使用dUP2进行补丁。

第二种方式,搜索和替换补丁,你要知道补丁位置的命令码,对它进行搜索然后替换为补丁后的命令码,然后 生成补丁程序。

dUP2也可以对注册表进行补丁,像上一章中修改注册表,每隔255次进行一次补丁,就像生产了一个密钥,如 果你经常使用软件的话255次很快就会结束。

最后,dUP2可以生成自定义皮肤,这样看上去程序界面比较美观。

启动dUP2,我们可以看到如下界面:

dUP 2		
Project Module In	nfo	
Patch Script Settings		
Module	Description	
		書意味時です
		ممتنا والعبو الأصادة فارعان

选择project->new:

atch Info		- - X
Patcher Caption	Patch	
Application		
Filename (s)		
URL		visit
Author	Someone	
Release Date	August 27, 2012	today
Release Info		*
		τ
About Box Message	created with dUP2 http://diablo2oo2.cjb.net	*
Scrolltext		17
Show this dial	og when create a new project	
Run patch with	administrator rights	
No Backup by	default	Save

这里我们可以输入一些信息,比如标题,作者,哪个程序的补丁等,填写完成后

Patch Info	
Patcher Caption	Patch
Application	Saturday Night Patcher
Filename (s)	Saturday Night Crackme.exe
URL	http://www.TheLegendOfRandom.com visit
Author	R4ndom
Release Date	August 27, 2012 today
Release Info	*
About Box Message	For more patches, go to
Scrolltext	· · · · · · · · · · · · · · · · · · ·
Run patch with	og when create a new project
No Backup by	default Cancel Save
	· 「 」 、 、 、 、 、 、 、 、 、 、 、 、 、

点击save后来到

dUP 2 - *New Proje Project Module Int	t*0	
Patch Script Settings		
Module	Description	
[Patch Info]	Saturday Night Patcher	
	- Aller	
	1. 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2	
		等使很好的

通常到这一步我会把这个工程保存下来以便后面使用,这样就不用再次填写上面的信息了。

现在我们要做补丁。右击后选择 Add 然后 Offset_Patch,界面会多出现一行

oject Module In	fo
atch Script Settings	
Module	Description
[Patch Info]	Saturday Night Patcher

双击后打开:

larget File			
ile:			
Old Filesize CRC32:	New Filesiz	e RC32 check	Patchmode File Offset
Don't check original (Keep original file time Disable WOW64 FS F	old) bytes Try patc Fix PE C Redirector	hing used file heckSum	 Virtual Address (VA) Relative Virtual Address (RVA)
Compare Files			
Driginal File: Patched File:			Compare
RAW Offset	Old Byte	New Byte	Add and Edit Offset Original Byte Patched Byte Edit Add Remove Clear List MemCheck [for Loaders] Enable MemCheck 2 Memory/Value: Up tems
File to load in Ollydbo			
		Follow i	in Ollydbg Cancel Save

这个界面就显示了补丁信息,我们点击... 按钮找到要补丁的程序

补丁模式可以选择Virtual Addresses 或者 RVA,而我们使用默认。

Offset	Patchdata	-			
Tar	get File				
File	Saturday Ni	ght Crackme.exe			
	Old Filesize	00057E00	New Filesize		
	CRC32:	57BF47F2	Enable CRC3		
	Don't chee	ck original (old) bytes	Try patching		
	Keep origi	inal file time	Fix PE Check		
	Disable W	OWEA ES Dedirector	STREET BUILD		

对比文件可以把远程程序和已经补丁后的程序进行对比,找到不一样的地方。

主要修改Add 和 Edit组。这就是我们填写偏移地址,原始数据和补丁数据的地方。

制作补丁程序

第一件事是找到地址并记下,原始值和补丁值。重新使用OD打开程序,来到401554处,记住原始的命令码



然后应用我们的补丁后看看变化



我们可以看到7A变成了69,然后我们把这一行和前一行的命令码记住: 83f8110f857A010000.

现在使用CFF Explorer打开程序,点击Hex Editor

🕳 CFF Explorer VII - [Saturday Night Crackme.exe]																	
File Settings ?																	
A H D	Saturd	ay Nig	tht Crac	kme.	exe												
10 × 01	1	-	-18	1		-	-										
🛱 🛅 File: Saturday Night Crackme.ex	40	10	-8	1.8		~	12										
e	Offset	T	0 1	2	3	4	5	6	7	8	9 A	В	C	D	E	F	Ascii
- Dos Header	0000000	10	4D 5A	90	00	03	00 1	00 0	00 0	04 00	0 00	00	FF	FF	00	00	MZ . L J VV
- 🔁 🔳 Nt Headers	0000001	0	B8 00	00	00	00	00 1	00 (00 4	40 00	0 00	00	00	00	00	00	,@
🗟 File Header	0000002	20	00 00	00	00	00	00 0	00 (00 0	00 00	0 00	00	00	00	00	00	
- 🗐 🗐 Optional Header	0000000		00 00	00	00	00	D0 1		00 0	00 00 01 D	0 00	00	C8	00	00	00	= 0 = ' fi TfiTh
Data Directories [x]	0000000	0	69 73	20	70	72	6F	67	72 6	61 6I	0 20	63	61	6E	6E	6F	is program canno
- Section Headers [x]	0000006	0	74 20	62	65	20	72	75 6	6E 2	20 6	9 6E	20	44	4F	53	20	t.be.run.in.DOS.
- Comport Directory	0000007	0	6D 6F	64	65	2E	OD I	OD (OA 2	24 00	0 00	00	00	00	00	00	mode\$
- Besource Directory	0000008	10	FF 65	A4	DB	BB	04 (CA	88 H	BB 0	4 CA	88	BB	04	CA	88	yeguss-Ess-Ess-E
Address Converter	0000000	6	35 1E 70 02	D9	88	FA BA	04 0	A CA	88 4	52 6	4 108	68	BB	04	CA	88	L T & F Richwift
Deserver Weller	0000001	iõ l	00 00	00	00	00	00 1	00 0	00 0	00 00	0 00	00	00	00	00	00	TIT BIRIONS EI
Beperidency Walker	0000000	0	00 00	00	00	00	00 1	00 (00 5	50 4	5 00	00	4C	01	04	00	PEI -
Hex Editor	0000001	0	B0 75	8E	45	00	00 0	00 (00 0	00 00	0 00	00	EO	00	OF	01	*u∎Eà.¤
- Videntifier	0000000		08 01	05	00	00	36 1		00 0	10 E	5 05	00	00	00	40	00	a
- Simport Adder	00000010	in L	00 10	00	00	00	02 1			14 01	1 00	00	00	00	00	00	r
- 🔍 Quick Disassembler	0000011	.ŏ	04 00	00	ŭŬ	00	00 1	00 0	00 0	00 20	0 06	00	00	04	ŏŏ	00	
- 🐎 Rebuilder	0000012	20	00 00	00	00	02	00 1	00 (00 0	0 00	0 10	00	00	10	00	00	
- 👆 Resource Editor	0000013	0	00 00	10	00	00	10 0	00 0	00 0	00 00	00 0	00	10	00	00	00	
🖵 🐁 UPX Utility	0000014		00 00	00	00	48	39 1		00 1	10 50	0 00	00	64	00	00	00	à H9
	0000016	0	00 00	00	00	00	00 1	00 1	00 0	0 00	0 00	00	00	00	00	00	.a 117
	0000013	n	00 00	00	00	00	00	nn i	00 0	10 01	1 00	nn	nn	ññ	00	00	

点击放大镜进行搜索刚刚记下的命令码

🖝 CFF Explorer VII – [Saturday Night Crac	kme.exe]	
File Settings ?	Saturday Night Crackme.exe	
File: Saturday Night Crackme.ex e Bos Header B Nt Headers B Sta Meader	Find String	Find
Optional Header Optional Header I Data Directories [x] Section Headers [x] Import Directory	Hex 83F8110F857A010000 Status: Bytes found	Find

点击 Find 后可以找到

Offset	- 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000950	0.0	83	F8	11	0F	85	7A	01	00	00	OF	B6	46	09	83	F8	.124012
00000960	2D	0F	85	6D	01	00	00	33	C9	33	D2	0F	B6	10	31	83	-@[m3É3C@¶ 1]
00000970	FB	40	0F	8D	5C	01	00	00	83	FB	30	0F	8E	53	01	00	û@⊅ \ . û0⊅ S .
00000980	00	33	CO	E8	48	FE	FF	FF	83	FA	01	0F	85	43	01	00	.3Ae byylu QIC
00000990	00	33	D2	41	83	F9	09	20	D2	B9	01	00	00	00	83	F9	.30Å 1 0 ¹ 1 ú
000009A0	09	74	20	0F	B6	5C	31	FT	83	EB	30	6B	DB	0A	OF	B6	.t.¤¶\1ÿ∣ë0k0.¤¶
000009B0	04	31	83	E8	30	03	C3	41	F7	F9	85	D2	74	E0	E9	11	-1]è0-ĂA÷ù Otàé∢
000009C0	01	00	00	B9	0A	0.0	00	00	33	D2	0F	B6	10	31	83	FB	¹3C⊉¶ 1∎û
000009D0	38	0F	8D	FD	00	00	00	83	ΓB	30	OF	8E	F4	00	00	00	8⊊ ý û0⊋ ô
000009E0	B8	0A	00	00	00	E8	54	FD	FF	FF	83	FA	01	0F	85	E1	, èTýyylú ¢lá
000009F0	0.0	0.0	00	33	D2	41	83	F9	11	7C	CF	0F	B6	46	0A	83	3OA∣ù∢ Io¶F.∣
000000A00	E8	30	OF	B6	5E	0B	83	EB	30	03	C3	OF	B6	5E	0F	83	ĕ0¤¶^rtë0-Ť¶^¤t
00000A10	EB	30	0F	B6	4E	10	83	E9	30	03	D9	B9	0A	00	00	00	ë0⊅¶N+1é0 -U1
00000A20	41	83	F9	0F	7D	1F	0F	B6	14	31	83	EA	30	OF	B6	7C	A 003 0191 6001
00000A30	31	01	83	EF	30	03	D7	3B	DO	74	E5	3B	D3	OF	85	91	1 110 -×;Đtå;C¤[′
00000A40	00	00	00	EB	DB	0F	B6	16	0F	B6	4E	0B	0F	AF	D1	81	euq¶-q¶Ndq N
00000A50	FA	BE	0A	00	00	75	7D	6A	00	6Å	00	6A	03	6A.	00	6A	ú¾u}j.j.j-j.j
000000460	01	68	00	00	00	80	68	FF	61	40	00	E8	E8	00	00	00	hhya@.èè
00000A70	83	F8	FF	74	5F	A3	A7	63	40	0.0	6A	00	68	Α3	63	40	løyt_fSc@ j hfc@
08400000	00	68	FF	00	00	00	68	A4	62	40	00	FF	35	A7	63	40	hy haid yosce
00000A90	00	E8	F2	00	00	00	33	D2	33	F6	BB	01	00	00	00	0F	.èò303öx
000000330	B6	82	24	62	40	0.0	83	58	0.0	7.4	0.8	05	15	D8	03	FO	TIDO La tra PLA

我们的7A偏移了956个字节,现在回到dUP2进行修改

Add and Edit	
Offset	956
Original Byte	7 A
Patched Byte	69
Edit	Add
Remove	Clear Us

然后点击Add 后来到

atched File:		
RAW Offset	Old Byte	New Byte
00000956	7 A	69
		等要的(4)。 2000年初日 -

点击 Save 后就把补丁保存到我们的程序里面了

现在选择 Project ->Create Patch后,会在程序的同级目录生成一个Patch.exe,运行这个Patch后

	Saturday Night Patcher	
[Filename]	Saturday Night Crackme.exe	
[URL]	http://www.TheLegendOfRandom.com	_
[Author]	R4ndom	_
[Release Date]	August 27, 2012	-
Release Info		
		-

点击Patch按钮后

Patch	
	Saturday Night Patcher
[Filename]	Saturday Night Crackme.exe
[URL]	http://www.TheLegendOfRandom.com
[Author]	R4ndom
[Release Date]	August 27, 2012
Release Info	
Backup Fil C:\Users\R OK	e: andom\Desktop\Saturday Night Crackme.ex
Make Backu	p Patch About Exit

我们可以看到Patch成功的信息。然后运行程序,显示成功了



注意patch程序必须和原始程序放在同一目录才有效。

使用搜索和替换方式

打开刚刚保存的dUP2工程,右击选择 Add->Search and replace Patch,双击后出现:

Search and Replace Patch		10	
Target File			
compare files Add Edit	Remove Clear List	Up Down chec	k occurrence String2Hex
Search Pattern	Replace Pattern	+	Comment
		_	
Patch compressed PE File Dis	able WOW64 File System Re to original file time	director 🔲 Fix PE C	heckSum
MemCheck [for Loaders]			Cancel
Enable MemCheck 2 Memory	Address (RVA):	MemoryValue:	Save
			an said a sai
		_	https://www.commentat

点击 Add 按钮

			U
ace Pattern			-
			0
nent		Occurrence	
	*	Patch All	
		or	Cancel
	-	# Match only	

现在把刚刚的原始命令码和补丁命令码填到相应位置

Search Pattern			
83 F8 11 OF 85 7A 01 00 00			9
Replace Pattern			
83 F8 11 OF 85 69 01 00 00			9
Comment		Occurrence	
	*	Patch All	
		or	Cancel
	-	# . Match only	Save

选择 patch All, 然后保存

(译者注:注意这里一个字节空一位否则保存不了)

aturday Night Crackme.exe				
compare files Add Edi	t Remove Clear List Up	Down chec	k occurrence St	ring2Hex
earch Pattern	Replace Pattern		Comment	
3 F8 11 OF 85 7A 01 0	. ?? ?? ?? ?? ?? 69 ?? ?.	ALL		
Patch compressed PF File	isable WOW64 File System Redirect	or Fix PF C	heckSum	
	eep original file time			
Try patching used file K				ancel
Try patching used file K lemCheck [for Loaders]			-	ancer

再次点击Save来到主界面。但是现在我们要进行界面修改,附件中有SSECS cUP skin.zip

dUP 2 - Saturday Nigh	t Crackme.dUP2			- 🗆 🗙
Project Module Info				
Patch Script Settings				
custom patcher skin [*.res	s file]			
C:\Users\Random\Des	ktop\Random\Website\	dUP2 Official Skin\MAIN.res	1	
custom window shape [R	RGN]			
C:\Users\Random\Des	ktop\Random\Website\	dUP2 Official Skin\MAIN.rgn		
custom patcher/loader icc	n			
C:\Users\Random\Des	ktop\Random\Website\	dUP2 Official Skin\Lips.ico		
custom patcher dialog col	lors			
Custom colors		Dialog Background	00000000	
Statics Background	0000000	ReleaseInfo Back.	0000000	
Statics Text	00FFFFFF	ReleaseInfo Text	00FFFFFF	load
LogBox Background	0000000	Button Background	0000000	
LogBox Text	0000FFFF	Button Text	00FFFFFF	save
transparent patcher dialo	g		0	Language
Transparent Patcher W	lindow		0 240	
add music (tracker module	e) to the patcher			
📝 use small ufmod pla	yer for XM music		play >>	stop
compress with external p	acker [%s = patcher.e	exe]		
when edit patchdata			(- 240
nide main window	make all window	s transparent		240
tade in/out patcher windo	w custom	scrolltext font		
Fade Out in 500	nilliseconds texts	ize 🗌 bold t	textcolor 000000FF	田田 吉爱 文団 に
14				and the second second

按照我上面的设置然后进行ProjectàCreate Patch,就生成了新的补丁程序,界面如下;



第二十章(上):玩转 VB 程序 - 第一部分

翻译都是我理解的方式进行描述,可能和原文不一致。

本教程中文版只在吾爱破解论坛 首发。

转载请注明来自吾爱破解论坛@52pojie.cn

正文开始

本章主要讲解VB编写的程序要怎么破解,因为这是一个大工程,所以我分为两个章节进行说明。

我们来看下我所提供的两个VB程序,被教程中的所有文件都会提供下载。

介绍VB编程语言与其他编程语言的区别,这个大家自行百度。

主要说明VB语言可以转换为p_code,这个语言可以跨平台移植,在Mac、Linux上都可以正常调用(如果是VB 写的DII)。

教程开始:

当你第一次使用OD打开一个VB编写的程序时,你会马上看到VB调用(MSVBVM50)DLL的Call,然后停在这 里等待事件响应,因此这样就会让逆向程序员感觉有所不同。第一件事是你会发现call的栈中信息很少;因为 多数程序的运行时间都是放在DLL中,而我们不关心DLL,只关心程序的callback、方法、时间。

00401170 00401175 00401178	** • •	68 D4674000 E8 F0FFFFF 0000	PUSH AFKayAs004067D4 CALL <jmp.&msubum50.#100> ADD BVTE PTR DS:[EAX],AL</jmp.&msubum50.#100>	
0040117C 0040117E 00401180 00401180	• • • •	0000 0000 3000 0000	ADD BYTE PTR DS: [EAX], AL ADD BYTE PTR DS: [EAX], AL XOR BYTE PTR DS: [EAX], AL ADD BYTE PTR DS: [EAX], AL	
00401184 00401186 00401188	:	3800 0000 0000	CMP BYTE PTR DS: (EAX), AL ADD BYTE PTR DS: (EAX), AL ADD BYTE PTR DS: (EAX), AL	Beginning of VB program

另一个不同是字串表是查不到有用的东西的,因为多数的消息窗口和其他windows控件都是放在资源文件中, OD不会显示出来,不像逆向C\C++程序时可用查找字串参考。

R Found	l strings are		
Address	Disassembly	Text string	
00401000 00401058 00401058 00401064 00401098 00401098 00401098 00401098 00401098 00401098 00401125 00401125 00401356 00401356 00401356 00401356 00401356 00401356 00401356 00401482 00401482 00401482 00401483 00401585 00401583 00401583 00401583 00401583 00401583 00401583 00401585 00401583 00401777 00401733 004017860 00401786	DD msvbum60ubaVarSub DD msvbum60adj_fdiv_m16i DD msvbum60adj_fdiv_m16i DD msvbum60adj_fdiv_m32i DD msvbum60adj_fdivr_m32i ASCII "10".0 ASCII "2000-",0 ASCII "2000-",0 ASCII "2000-",0 ASCII "Form1",0 ASCII "Form1",0 ASCII "Leimorackme",0 ASCII "1t",0 ASCII "fff",0 ASCII "fff",0 ASCII "fff",0 ASCII "ffff",0 ASCII "ffff],0 ASCII "ffff],0 ASCII "fff],0 ASCII "fff],0 ASCII "fff],0 ASCII "fff],0 ASCII "fff],0 ASCII "fff],0 ASCII "fff],0 ASCII "ff],0 ASCII "ff]	ASCII "P® «%" ASCII "P® «%" ASCII "OWP=" ASCII "P® «%" ASCII "P® «%"	selection
			WIND - 1 - 407 - 2

另外一个比较麻烦的地方是,VB调用RegisterWindowEx和MessageBox是在自己的DLL中调用,我们不能下API断点。

R Found i	ntermodular calls	
Address	Disassembly	Destination
00401310 C 00402EAC 00402E73 00402F99 0040302F 0040302F 00403082 00403082 00403082 00403082 00403082 00403082 00403082 00403082 0040308 00403181 00403181 00403181 00403253 00403275 00403275 00403275 00403275 00403295 00403542 00403508 00403535 00403535 00403535 00403535 00403535 00403535 00403535 00403535 00403535 00403535 00403535 00403542 00403542 00403542 00403542 00403542	<pre>CALL DWORD PTR DS:[<&MSUBUM60vbaHresuitCheckObj>] CALL DWORD PTR DS:[<&MSUBUM60vbaHresuitCheckObj>] CALL DWORD PTR DS:[<&MSUBUM60.#\$533] CALL DWORD PTR DS:[<&MSUBUM60vbaFreeUarList}] CALL DWORD PTR DS:[<&MSUBUM60vbaFreeUbj}] CALL DWORD PTR DS:[<&MSUBUM60vbaFreeUbj}] CALL DWORD PTR DS:[<&MSUBUM60vbaHresuitCheckObj]] CALL DWORD PTR DS:[<&MSUBUM60vbaHresit]] CALL DWORD PTR DS:[<&MSUBUM60vbaHresit]] CALL DWORD PTR</pre>	<pre>Initial CPU selection: msubum68ubaHresultCheckObj msubum68ubaHresultCheckObj msubum68.rtcRandomNext msubum68.rtcRandomNext msubum68.rtcRandomNext msubum68ubaRSIntI2 msubum68ubaRSIntI2 msubum68ubaRSIntI2 msubum68ubaFreeUarList msubum68ubaFreeUarList msubum68ubaFreeUarList msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaFreeObj msubum68ubaHresu msubum68ubaHresu msubum68ubaUarHoue msubum68ubaUarHoue msubum68ubaUarHoue msubum68ubaUarForInit msubum68ubaUarForInit msubum68ubaUarForInit msubum68ubaItVar msubum68ubaStrUarVa1 msubum68ubaStrUarVa1 msubum68ubaStrI2 msubum68ubaStrI2 msubum68ubaStrI0ve msubum68ubaYENI msubuFoffee msubuFoffee msubuFoffee msubuFoffee msubuFoffee msu</pre>
00403585 00403585 00403504 00403500	CALL DWORD PTR DS:[<&MSUBUM60UbaVarCat>] CALL DWORD PTR DS:[<&MSUBUM60UbaVarMove>] CALL DWORD PTR DS:[<&MSUBUM60UbaFreeOtj>] CALL DWORD PTR DS:[<&MSUBUM60UbaFreeOtj>]	Meubum60ubaVarCat Meubum60ubaFreeStrList Meubum60ubaFreeObj



你可看到,这里没有有用的API,没有有用的字串。

在看代码前,我们把用p-code编译CrackmeVB1.exe加载到OD,然后把代码窗口拉到最上面,看看VB程序的 架构。我们可以看到DLL的很多功能

00401000		ER778472	DU m	sybym60ybaVarSub
00401004		0705A272	DD M	subum60. UbaStrI2
00401008		8693A372	DD m	subum60. Cloos
0040100C	1	F9098372	DD M	subum60, adi fotan
00401010		FE699472	DD m	SUBUM60, UbaUarMoue
00401014		31689472	DD m	SUNUMER, UNAFreeUar
00401018	1.1	9R600272	DD M	subum68, ubal enBatr
00401010		80000172	DD m	subum68. rtoRab
00401020		62720472	DD m	subum60, ubaEreeUarList
00401024		B0020372	DD m	subum60, adi fdiu m64
00401028		C39E9172	DD M	subum60, ubaEreeObilist
00401020	1.0	B7709272	DD m	subum60 rtcAnsiUalueBstr
00401030		41090372	DD m	subum68, adi forem1
00401034		74020172	DD m	subum60 ubaHyesu LtCheckOhi
00401039		0R600272	DD m	subun60. Ubal enllan
00401030		6E020372	DD m	subum60 adi fdiu m32
00401040		00930472	00 00	subum60, UballayForInit
00401044		85C00172	DD m	subum60 stoRandomNext
00401049		32010172	DD IN	subum60, rtoMandonnent
00401040		E19E0172	00 //	subum60 uba0biSet
00401050		06030372	00 00	subum60 adi fdiu m16i
00401054		08000172	00 /	subum60 uba0biSetOddref
00401059		06040372	DD m	subum60 adi fdium m16i
00401050		FE940372	00	subum60 Claip
00401060	1	2F700272	00 00	subum60 stoMidChaollan
00401064		E0620372	DD m	cubum60 ubaChketk
00401068		74980072	DD IN	SUDUMER FUENT SINK OddRef
00401060		E6970472	DD m	subum60 uballarTstFo
00401070		F6090372	00 00	subum60 adi fratan
00401074		87980072	00 m	SUNUMAR FUENT SINK Release
00401078		93959372	00 00	subum68. Clant
99491970		85900072	DD M	subum60 FUENT SINK QueruInterface
00401080		60760472	DD m	subum60, uballandul
00401084		DE470272	DD m	subum60 ubaEvcentHandler
00401088		89060372	00 00	subum60 adi forem
00401080		B0030372	00	subum60 adi fdiur m64
00401090		13750472	00 0	subum68, ubaFPEycention
00401094		48199272	00	subum60, ubaStrUarUal
00401098		70690272	DD m	subum60. uballarCat
00401090		28940372	00	subum68. Cilog
00401000		37020172	00	suburea ubaNeu2
00401040		SUNCHINE	100 1	SAPALION - APRILEME

这就是运行领空中所有用到的API call了。

A second s

代码窗口往下拉一点点,我们可以看到跳转表。这和多数windows程序的一样。

00401190	DF3C4000	DD Crackme .00403CDF	The second s
00401100	- FF25 64104000	JMP DWORD PTR DS: (<&MSUBUM60, vbaChkstk>)	msubum60. ubaChkstk
00401106	\$- FF25 84104000	JMP DWORD PTR DS: [<%MSUBUM60. vbaExceptHandler>]	msubum60. ubaExceptHandler: Struc
004011AC	> FF25 90104000	JMP DWORD PTR DS: [<&MSUBUM60. UbaFPException>]	msubum60. ubaFPException
004011B2	- FF25 50104000	JMP DWORD PTR DS: [<&MSUBUM60, adi fdiv m16i>]	msubum60, adi fdiu m16i
00401168	FF25 3C104000	JMP DWORD PTR DS: [<&MSVBUM60. adi fdiu m32>]	msubum60. adi fdiu m32
004011BE	- FF25 A8104000	JMP DWORD PTR DS: [<&MSUBUM60, adi fdiv m32i>]	msubum60. adi fdiu m32i
904011C4	FF25 24104000	JMP DWORD PTR DS: [<&MSUBUM60. adi fdiv m64>]	msubum60. adi fdiu m64
004011CA	FF25 B8104000	JMP DWORD PTR DS: [<&MSUBUM60, adj fdiv r>]	msubum60. adi fdiv r
00401100	FF25 58104000	JMP DWORD PTR DS: [<&MSUBUM60. adj fdior_m16i>]	msvbvm60adj_fdivr_m16i
88481106	FF25 B4104000	JMP DWORD PTR DS: [<&MSUBUM60adj_fdivr_m32>]	msvbvm60adj_fdivr_m32
004011DC	FF25 AC104000	JMP DWORD PTR DS: [<&MSUBUM60adj_fdivr_m32i>]	msvbvm60adj_fdivr_m32i
004011E2	FF25 8C104000	JMP DWORD PTR DS: [<&MSUBUM60adj_fdivr_m64>]	msvbvm60adj_fdivr_m64
004011E8	FF25 70104000	JMP DWORD PTR DS: E<&MSUBUM60adj_fpatan>1	msvbvm60adj_fpatan
004011EE	FF25 88104000	JMP DWORD PTR DS: [<%MSUBUM60adj_fprem>]	msvbvm60adj_fprem
004011F4	FF25 30104000	JMP DWORD PTR DS: [<&MSUBUM60adj_fprem1>]	msvbvm60adj_fprem1
004011FA	FF25 0C104000	JMP DWORD PTR DS:[<&MSUBUM60adj_fptan>]	msvbvm60adj_fptan
00401200	FF25 D4104000	UMP DWORD PTR DS: [<&MSUBUM60Clatan>]	msvbvm60CIatan
00401206	FF25 08104000	JMP DWORD PTR DS: [<&MSUBUM60CIcos>]	msvbvm60CIcos
00401200	FF25 E8104000	UMP DWORD PTR DS: [<&MSUBUM60CIexp>]	msvbvm60CIexp
00401212	FF25 9C104000	JMP DWORD PTR DS:[<&MSUBUM60CIlog>]	msvbvm60CIlog
00401218	FF25 5C104000	JMP DWORD PTR DS: E<&MSUBUM60CIsin>J	msvbvm60CIsin
0040121E	FF25 78104000	JMP DWORD PTR DS: [<&MSUBUM60CIsqrt>]	msvbvm60CIsqrt
00401224	FF25 E0104000	UMP DWORD PTR DS: E<&MSUBUM60CItan>]	msvbvm60CItan
0040122H	- FF25 DC104000	UNP DWORD PTR DS: L<&MSUBUM60allmut>1	msubum60allmul
00401230	- FF25 EC104000	JMP DWORD PTR DS: EK&MSVBUM60vbaFreeObj>1	msvbvm60vbaFreeObj
00401236	FF25 4C104000	JMP DWORD PTR DS: [<&MSUBUM60UbaObjSet>]	msubum60uba0bjSet
00401230	FF25 20104000	JNF DWORD FIR DS:E(&MSUBUM60UbaFreeUarList)]	msvbvm60vbaFreeVarList
00401242	- FF25 00104000	UNP DWORD FIR DS: E<&nSUBUNE0UBaR8Int12>1	MSUDUM60UDaK8In112
00401248	- FF25 10104000	JUL DWORD FIR DS: L<&nSVB0060. #588>3	MSVDVM60.rtcKgD
0040124E	- FF25 44104000	UNP DWURD PTR DS: L<&nSVB0060. #593>1	MSVDVM60.rtcKandom"2/1
00401254	- FF25 34104000	JNF DWURD FIR US:L(&ASVBVA60VbaHresultCheckObj>1	msvDvm60vDaHresultCheckUbj

然后我们往下拉,来到VB程序的资源信息中,这里有包含按键、回调。一个比较值得注意的地方是,这里是直接使用回调名称进行回调的,比如程序上有个MyCallBackButton,在代码中可以直接看到这个名字。这样就可以在代码中找到callback的地方。



继续往下拉,就能发现callback的处理代码了。这里使用了常规的callback,和你看到的一样,这里没有说明是 什么的callback,我们可以通过改变映射文件进行调用。



最后,我们来到导入表或者IAT。我们能得到更多有用的信息。



当然上面的信息并不能完成这次的破解。我们需要使用其他工具

VB Decompiler Lite

这个软件有两个版本,一个是Lite,一个是Pro。Lite版本是免费的,所以我们有附带该软件。VB Decompiler 可以把p-code编译的VB程序进行反编译,反编译为VB语言。这样我们就可以看到VB程序的资源文件。好,现 在打开VB Decompiler程序。

le Tools Plugins Help	
FileName:	(mm) [Decompile]
bjects Tree:	
	E
	-
Decompiler by DatFix Software	

打开 CrackmeVB1.exe程序,然后点击Decompile按钮

VB Decompiler Lite v9.0		X
File Tools Plugins Help		_
FileName: C:\Users\Rando	m\Desktop\Random\Tutorials\Intro to Reverse Engineering\20 Working with VB Bir	naries Decompile
Objects Tree:	Native Code	
Project Forms Form1 SerControls Code Form1 Module1 Module2 Form2 Form2 API	<pre>Type=EXE Reference=*00020430-0000-C000-000000000 Form=Form1.frm Module=Module1; Module1.bas Module=Module2; Module2.bas Form=Form2.frm Startup="Form1" IconForm="Form1" HelpFile="" ExeName32="andre" VersionCompanyName="?" MajorVer=0 RevisionVer=0 RevisionVer=0 RevisionVer=0 Command32 = "" HelpContextID = "0" CompatibleMode = "0" AutoIncrementVer = 0 ServerSupportFiles = 0 CompilationType = 0 FavorPentiumPro(tm) = 0 CodeViewDebugInfo = 0 NoAliasing = 0 BoundsCheck = -1 FlPointCheck = 0 FDIVCheck = 0 UnroundeFP = 0 StartMode = 0 Retained = 0 ThreadPerObject = 0 MaxNumberOfThreads = 1</pre>	46]#2.0#0#\\
	[MS Transaction Server] AutoRefresh = 1	善爱破解论。 www.S2pojie.o

然后可以看到许多信息了,注意左边的Forms文件夹,这里有两个,所有的资源文件都在这里面,我们一个一个看,一个是主窗口,一个是关于窗口。

Form 2 - "About"	🙊 Leimcräckme)
Leimcrackme Autore: nessuno (l'autore si vergogna di questo coso e non vuole che il suo nome venga ulteriormente infangato.) Anzi dai ve lo dico il vero autore sono io,	Whoever tries this out and new big loser this isn't a big deal, before doing anything else if it go and hide yourself for ev Name: Serial: <u>Check!</u>	ver solves it means it's a , so u must do that 'u don't manage to solve ver and ever! Amen. <u>About</u>

你启动程序的时候已经发现这里点击的About按钮后出现的是其他信息,而且about窗口上的OK按钮是不可点击的。如果你跟着我的教程走,你就会知道怎么来进行破解了。

双击Form2后,我们可以看到资源信息了



这里我们可以看到一个按钮的字串是OK,另外一个信息的是这里的callback被叫做Command1。

双击Form1,我们来到主窗口



现在我们在破解前知道了许多信息。一个重要的按钮叫 Check!,他的callback叫Command2,接着往下看,你可以看到Code中有对应的callback,而且知道偏移地址(Command2_Click_402FD0),如果你双击后就可以 看到VB代码

Hielvame: C: Users (Random (Desktop (Rando	Native Code
- Droject	Private Sub Command2 Click() 1402FD0
 Forms Form1 Form2 UserControls Code Form1 Form_MouseMove_403140 Command1_Click_402F40 Command2_Click_402FD0 Label1_MouseMove_403200 ShowAboutBox_402E60 Module1 Module2 Form2 API 	<pre>loc_00402FD0: push ebp loc_00402FD1: mov ebp, esp loc_00402FD1: mov ebp, esp loc_00402FD1: push 004011R6D ;vbaExceptHandler loc_00402FD1: push eax fs:[00b] loc_00402FE1: push eax loc_00402FE2: mov fs:[00000000h], esp loc_00402FE2: push ebx loc_00402FE2: push edi loc_00402FFE: mov var_C, esp loc_00402FFE: mov var_S, 00401110h loc_00402FFE: mov eax, eei loc_00402FFE: mov eax, eei loc_00402FFE: mov eax, eei loc_00402FFE: mov var_4, eax loc_00403001: mov var_4, eax loc_00403001: mov var_4, eax loc_00403008: mov ecx, [esi] loc_00403008: mov ecx, [esi] loc_00403008: mov ecx, [esi] loc_00403001: xor eax, eax loc_00403011: xor eax, eax loc_00403012: lea edx, var_24 loc_00403015: mov var_34, eax loc_00403015: mov var_34, eax loc_00403022: push edx loc_00403022: push edx loc_00403022: mov var_34, eax loc_00403022: mov var_34, eax loc_00403022: mov var_24, edi loc_00403025: mov var_24, edi loc_00403025: mov var_24, edi loc_00403026: mov var_24, edi loc_00403027: call [0040144h] ; Rnd(arg_2) 'arg_1 loc_0040308: lea eax, var_34 loc_0040308: lea eax, var_34 loc_0040308: lea eax, var_34 loc_0040308: lea eax, var_34 loc_0040308: push eax</pre>

现在我们知道Check按钮的偏移地址了,然后使用OD打开程序,来到402FD0处

00402FCE 00402FCF	90 90 > 55	NOP NOP PUSH EBP	inning of calback
00402FD1 00402FD6 00402FD6 00402FD6 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FE2 00402FF2 004000F7 00000F7 0000F7 0000F7 0000F7 0000F77 00000F7	 88EC 83EC 0C 68 A6114000 64:A1 00000000 50 64:8925 0000000 83EC 74 53 56 57 8965 F4 C745 F8 1011400 8875 08 88C6 83E0 01 8945 FC 83E6 FE 56 8975 08 880F 	MOV EBP.ESP SUB ESP.0C PUSH EAX. MOV EAX.DWORD PTR FS:[0] PUSH EAX MOV DWORD PTR FS:[0].ESP SUB ESP.74 PUSH EBX PUSH EDI MOV DWORD PTR SS:[EBP-C].ESP MOV DWORD PTR SS:[EBP-8].CrackmeV. MOV DWORD PTR SS:[EBP-4].EAX AND EAX.1 MOV ESI.FFFFFFE PUSH ESI MOV DWORD PTR SS:[EBP-4].EAX AND ESI.FFFFFFE PUSH ESI MOV DWORD PTR SS:[EBP-4].EAX AND ESI.FFFFFFE PUSH ESI MOV CWORD PTR SS:[EBP+8].ESI MOV CX.DWORD PTR SS:[EBP+8].ESI	.00401110 CrackmeV.0040240A CrackmeV.0040240A
00403000 00403010 00403012 00403015	 FF51 04 33C0 8D55 DC BB 04000280 	CALL DWORD PTR DS:[ECX+4] XOR EAX,EAX LEA EDX,DWORD PTR SS:[EBP-24] MOV.EBX,88020004	CrackmeV.0040240A
0040301R 0040301F 00403022 00403023 00403023 00403026 00403029	BF 0H000000 8945 DC 52 8945 CC 8945 BC 8945 BC 8945 E4 8950 DC	MOV EDI, GH MOV DWORD PTR SS: [EBP-24], EAX PUSH EDX MOV DWORD PTR SS: [EBP-34], EAX MOV DWORD PTR SS: [EBP-44], EAX MOV DWORD PTR SS: [EBP-1C], EBX MOV DWORD PTR SS: [EBP-1C], EBX	CrackmeV.0040240A MSVBVM60.72953E28 CrackmeV.00402466 CrackmeV.00402466

如果你在这里设了断点,运行程序,输入用户名和序列号,点击Check后就会停在这里了,后面的破解就很简 单了。不再进行详细教程。

VB Decompiler Pro

我想告诉你们使用p-code进行查找callback,这样就必须使用VB Decompiler Pro,当然这个不是免费软件,所以不附带(译者注:我在52Pojie中找到,并附上),使用VB Decompiler Pro

打开CrackmeVB1,然后反编译。



这里我们就能看到p-code方法和callback了,所有地址都显示了要做什么东西。

使用OD打开程序并来到402FD0处



设置断点,然后跑起来,来到4030AA处,背景颜色变了,这是从p-code中知道的。

2 Leimcrack	me 🗾 📈	004403069 DFE0 004403060 AB 80 004403060 PF85 BC0000000 TEST RL, 00 004033060 PF85 BC0000000 MC CrackmeU, 00403	12F
Whoever to	ies this out and never solves it means it's a	00403073 . FF07 00403075 . D945 84 00403075 . D800 08114000 FTN D8: 00403075 . 50 00403075 . 50 0040300000000000000000000000000000000	BP-7C] (401108]
big loser before doin it go and	this isn't a big deal, so u must do that g anything else if u don't manage to solve hide yourself for ever and evert Amen.	Od40307F DFE0 FS15W HX Od403081 A8 0D TEST AL, 00 00403083 ~ 0685 A6000000 NC CrackmeV.00403 00403089 FD7 CALL ED1 00403086 D945 88 FLD DWORD PTR S5:11 00403082 D900 08114000 FNUL DWORD PTR DS:	12F MSUBUM66 1997-783 14011083
Name:	R4ndom	004483094 50 004483095 DFE0 FE0 FE0 FE0 FE0 FE0 FE0 FE0 FE0 FE0	nanges background color
Senal	12121212	0046305F • FTD7 0046306F • FTD7 0046306F • FT15 1C104000 00463062 • FF15 1C104000 CALL EDI PUSH EAX CALL DI	2<&MSUBUM60, #288>3 MSUBUM66
	heck! About	0044030R9 56 0044030R9 56 0044030R9 FF53 64 0044030R0 9500 TEST ERX, ERX	CEBX+643 MSVBUM66
	1. 2 1	004030AF . DBE2 FCLEX 00403061 . 70 0F USE SHORT CrackmeU 00403065 . 68 64 PUSH 64 0403065 00403065 . 68 90254000 PUSH CrackmeU.00403	004030C2

在4030E3处,弹窗坏消息。



我们看这里的代码可以发现有比较跳转的代码



把断点设在4030E1处,重启程序,点击Check后,OD断下来,改变0标志位。这里不会跳出坏消息。这样来看就是4030C0这个Call中进行了判断和处理坏消息。我们把断点设在4030C0处,重启程序,然后一步一步调试。

004022C0	E EE	PHCH EPP	
00400200	ODEC	MOULEDD COD	
00400201	· ODEC OD	NUV EDF, ESF	
00403203	. 83EL 08	SUB ESP, 8	
00408206	. 68 H6114000	PUSH CUMP.&MSVBVM60VDaExceptHandler2	SE handler installation
004032CB	. 64:A1 00000000	MOV EAX, DWORD PTR FS: [0]	
00403201	. 50	PUSH EAX	
00403202	. 64:8925 0000000	MOV DWORD PTR FS: [0].ESP	
00403209	. 81EC 58010000	SUB ESP, 158	
0040320F	53	PUSH FBX	Crackmell, 00405834
00403250	. 56	PIICH FST	01 0001000100001
130001.00		PUCH EDT	MOUDIMER UNPOTATTO
00400201	OOCE ED	MOU DWOOD DTD CC. (CDD. 01 CCD	HOVEVIDEODANOTITETE
00405252	· 0705 F0	MOU DUORD FTR SSILEDFTOJ,ESP	
004032E5	. L(45 FL 4011400	HUV BWUKU PIK SSILEBF-4J, Cracknev, 00401140	
004032EC	. H1 10504000	MOV ERX, DWORD PTR DS: [405010]	The second second second second
004032F1	. 33FF	XOR EDI,EDI	MSUBUM60UbaRSIntI2
004032F3	. 3BC7	CMP EAX, EDI	MSUBUM60ubaR8IntI2
004032F5	. 897D EØ	MOV DWORD PTR SS: [EBP-20], EDI	MSUBUM60. UbaRSIntI2
004032F8	. 897D DØ	MOU DWORD PTR SS: (EBP-301, EDI	MSUBUM60, ubaR8IntI2
004032FB	8970 00	MOU DWORD PTR SS: [EBP-401, EDT	MSUBUM60, ubaRSInt 12
994932FF	8970 B0	MOU DWORD PTR SS (FERP-501 FOT	MSUBUM60 ubaRSInt12
00400201	. 007D 0C	MOLI DIJOPD PTP CC. (EDD_EA1 EDT	MCUDUM60 ubsPOInt 12
00400301	. 077D HC		MCUDUMCO what DOTA + 12
00405304	. 07/U Ho	HOU DWORD FIR SSILEDF-SOJ.EDI	MOUDUMCOUDaRoInt12
00403300	. 897U H4	HUV DWORD FIR SS:LEBP-SCJ.EDI	nsvBvn60vDakSint12
0040330H	. 8970 94	MOV DWORD PTR SS:LEBP-6C1.EDI	MSUBUM60UbaR8Int12
00403300	. 8970 84	MOV DWORD PTR SS: LEBP-7CJ, EDI	MSUBUM60VbaR8IntI2
00403310	 89BD 74FFFFFF 	MOV DWORD FTR SS: [EBP-8C], EDI	MSUBUM60vbaR8IntI2
00403316	. 89BD 64FFFFFF	MOV DWORD PTR SS: [EBP-9C], EDI	MSUBUM60vbaR8IntI2
0040331C	. 89BD 54FFFFFF	MOU DWORD FTR SS: [EBP-AC], EDI	MSUBUM60. ubaR8IntI2
00403322	89BD 44FFFFFF	MOU DWORD PTR SS: (EBP-BC1, EDI	MSUBUM60, ubaR8Int12
00403928	8980 34FFFFFF	MOULDWORD FTR SSIFERP-CC1 FDT	MSUBUM60 ubaRSInt12
80400000	OODD SAFEFEEF	MOLI DWORD PTP CS. (EPP-DC1 EDT	MSUDUM60 ubspotst12
00400322	· 0700 24FFFFFF		MCUDUMCO
00405034	. 070U 14FFFFFF	HOU DUORD FIR SSILEDFECS, EDI	novbonedobacointiz
0040333H	. 89BU 04FFFFFF	HOU DWORD FIR SS:LEEP-FUJ,EDI	MSVBVM60VDaR8Int12
00403340	 89BD F4FEFFFF 	MOV DWORD PTR SS:[EBP-10C],EDI	MSUBUM60UbaR8Int12
00403346	. 89BD E4FEFFFF	MOV DWORD PTR SS: [EBP-11C], EDI	MSUBUM60UbaR8IntI2
0040334C	 89BD C4FEFFFF 	MOV DWORD PTR SS:[EBP-13C],EDI	MSUBUM60VbaR8IntI2
00403352	 89BD B4FEFFFF 	MOV DWORD PTR SS: [EBP-14C], EDI	MSUBUM60UbaR8IntI2
00403358	. 89BD R4FEFFFF	MOV DWORD PTR SS: [EBP-15C].EDI	MSUBUM60, ubaRSIntI2
0040335E	× 75 15	JNZ SHORT CrackmeU, 00403375	Concernence - Concernence
99493369	68 10504000	PUSH CrackmeU, 00405010	
00403365	68 481E4000	PUSH Crackmell, 00401E48	
00402260	EE15 00104000	COLL DMORD PTP DS+F/9MSUPUM69 UbsNau231	MCUDUMER UbaNau2
004000276	01 10504000	MOLL EQY DUODD DTD DC+ [405040]	hovboridevbanewz
00400070	. HI 10504000	HOU COULDHOLD FIR DOLLADOID	
00403375	> 8868	HUV ECX, DWORD FIR DS:LEHXI	
08403377	. 50	PUSH EHX	
00403378	. FF91 08030000	CALL DWORD PTR DS: [ECX+308]	and the second s
0040337E	. 8B35 10104000	MOV ESI, DWORD PTR DS: [<&MSUBUM60vbaVarMove]	MSUBUM60vbaVarMove
00403384	. BB 0900000	MOV EBX,9	and the state of t
08403389	8055 94	LEA EDX. DWORD PTR SS: [EBP-6C]	
00403380	8040 B0	LEG ECX, DWORD PTR SS: [EBP-50]	
0040338F	8945 90	MOU DHORD PTR SSICERP-641, FOX	
00403392	995D 94	MOULDWORD PTP SSIFEPP-601 ERY	Crackmell 00405834
00400352	EED6	COLL EGT	/ MCI DI MCG what an Maria
00403336		HOLL FON DWODD DTD DC (4050101	(enovovnos,_voavarnove)
00403334	. HI 10504000	HOV EHA, DWORD PIK DS: 14050101	HOUDINGO
0040339C	· 3807	CHP EHX, EDI	MSOBON60
0040339E	.* 75 15	UNC SHORT CrackmeV.004033B5	
004033A0	. 68 10504000	PUSH CrackmeV.00405010	
00403305	68 481F4000	PUSH Crackmell, 99491E48	

我们慢慢看,就能看到Call VB DLL的方法,下拉代码窗口可以看到4033644处

00403620 00403632 00403632 00403637 0040363C 0040363C 00403642 00403642 00403643	E8 E3000000 E8 DE000000 E8 DE000000 E8 D4000000 8D4D C0 8D4D C0 8D55 D0 51 52	CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 LEA ECX,DWORD PTR SS:[EBP-40] LEA EDX,DWORD PTR SS:[EBP-30] PUSH ECX PUSH ECX	
00403644	. FF15 6C104000	CALL DWORD PTR DS: E<&MSUBUM60vbaVarTstEq>]	MSUBUM60UbaUarTstEq
0040364A	. 66:85CØ	TEST AX, AX	A CONTRACTOR OF
00403640	· 74 0D	JE SHORT CrackmeV.0040365C	
0040364F	. E8 CC000000	CHLL CrackmeV.00403720	
00408654	· 95	WHIT Desakualt 004026EE	
00400050	. 50 FE364000	IMP CHOPT Conscional 00403600	
00400000	S ES BERSARA	COLL Crackmell 00403020	
00403661	9B	WAIT	
00403662	68 FE364000	PUSH CrackmeU, 004036FE	
00403667	.~ EB 61	JMP SHORT CrackmeV.004036CA	
00403669	. 8D45 A8	LEA EAX, DWORD PTR SS: [EBP-58]	
0040366C	. 8D4D AC	LER ECX, DWORD PTR SS: [EBP-54]	
0040366F	. 50	PUSH ERX	
00403670	. 51	PUSH ECX.	
99493671	. 6H 02	PUSH 2	MOUDING A STATE
00403673	. FF15 B0104000	CHLL DWORD FIR US: L\&HSVBVH60VDarreestrLis	hovevnoevoarreestriist
00403679	- 0304 00 9040 04	FO FOY MADD DTD SC FEDDLED	
0040367E	EE15 EC104000	COLL DWORD PTR DS: [(&MSUBUM60, ubaErceObi)]	MSURUM60, ubaErceObi
88483685	. 8095 14FFFFFF	LEA EDX. DWORD PTR SS: [EBP-EC]	1012110011201166003
0040368B	. 8D85 24FFFFFF	LEA EAX, DWORD PTR SS: [EBP-DC]	
00403691	. 52	PUSH EDX	
00403692	. 8D8D 34FFFFFF	LEA ECX, DWORD PTR SS: [EBP-CC]	

vbaVarTstEq看上去像strcmp。然后前面传来2个参数。然后在40364F处还有一个call,点进去看看



现在我们必须要跳转到403644处。这样就能得到好消息。所以我们修改40344F的JE跳转

00403435 00403435 00403439 00403439 00403439 00403439 00403441	. 50 . 8055 E0 . 51 . 52 . FF15 40104000 . 9220 90104000	PUSH EAX LEA EDX, DWORD PTR SS: LEBP-20] PUSH ECX PUSH EDX CALL DWORD PTR DS: K&MSUBUM60,VbaVarForIn MOL EDI DWORD PTR DS: K&MSUBUM60,VbaVarForIn	it MSUBUM60ubaUarForInit
00403447 00403447 00403440 0040344F 00403455 90403455	 SB1D C4104000 SB1D C4104000 S5C0 ØF84 C9010000 A1 10504000 S5C0 	MOV EBX, DWORD PTR DS: [<&MSUBUM60vbaUarAd TEST EAX, EAX UE CrackmeV.0040361E MOV EAX, DWORD PTR DS: [405010] TEST FAX.FAX	Jumps to goodboy
0040345C 0040345E 00403463 00403463	 75 15 68 10504000 68 481F4000 FF15 A0104000 	UNZ SHORT CrackmeV.00403473 PUSH CrackmeV.00405010 PUSH CrackmeV.00401F48 CALL DWORD PTR DS:[<&MSVBVM60vbaNew2>]	MSUBUM60ubaNew2

这就是我们希望代码走到的地方


现在把断点设到40344F,跑程序,修改0标志位,强制跳转到40361E处

00403611 00403612 00403613 00403619	. 52 50 . FF15 E4104000 .^ E9 2FFEFFFF	PUSH EDX PUSH EAX CALL DWORD PTR DS: [<&MSUBUM60vbaVarForNex UMP CrackmeV.0040344D	t MSUBUM60vbaVarForNext
00403623 00403623 00403623 00403632 00403632 00403637 00403637	- E8 E3000000 - E8 E3000000 - E8 D5000000 - E8 D5000000 - E8 D4000000 - SD400 C0	CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 CALL CrackmeV.00403710 LEA ECX, DWORD PTR SS: [EBP=40]	e land here after rcing the jump
0040363F 00403642 00403643 00403643 00403640 00403640 00403654 00403655 00403655 00403655 00403655	. 8055 D0 . 51 . 52 . FF15 6C104000 . 66:85C0 . 74 00 . E8 CC0000000 . 98 . 68 FE364000 . FR 6F	LEA EDX.DWORD PTR SS:[EBP-30] PUSH ECX PUSH EDX CALL DWORD PTR DS:[(&MSVBVM60,vbaVarTstEq) TEST AX,AX JE SHORT CrackmeV.0040365C CALL CrackmeV.00403720 WAIT PUSH CrackmeV.004036FE JMP SHORT CrackmeV.004036FE	MSUBUM60vbaVarTstEq

然后来到40364D处后,我们得到了好消息,这就是破解这个程序的方法



有兴趣的可以去找到VB调用API的部分与C/C++的API对比,其实很容易找到相同的东西。

附件其他VB程序请自己练习,这里就不在进行讲解了