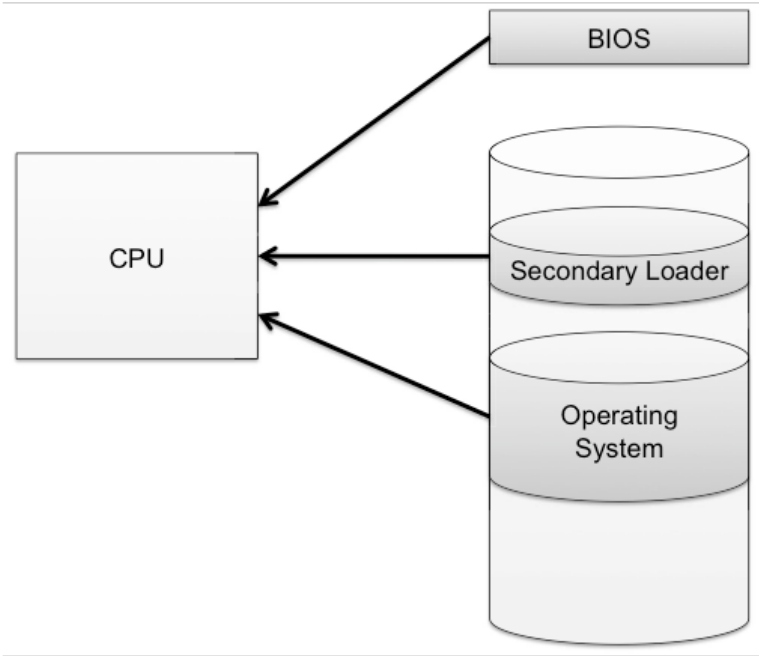
Operating Systems Security

The Boot Sequence

- The action of loading an operating system into memory from a powered-off state is known as **booting** or **bootstrapping.**
- When a computer is turned on, it first executes code stored in a firmware component known as the **BIOS (basic input/output system).**
- On modern systems, the BIOS loads into memory the **second-stage boot loader,** which handles loading the rest of the operating system into memory and then passes control of execution to the operating system.



BIOS Passwords

- A malicious user could potentially seize execution of a computer at several points in the boot process.
- To prevent an attacker from initiating the first stages of booting, many computers feature a **BIOS password** that does not allow a second-stage boot loader to be executed without proper authentication.

Hibernation

- Modern machines have the ability to go into a powered-off state known as **hibernation.**
- While going into hibernation, the OS stores the contents of machine's memory into a **hibernation file** (such as hiberfil.sys) on disk so the computer can be quickly restored later.
- But… without additional security precautions, hibernation exposes a machine to potentially invasive forensic investigation.

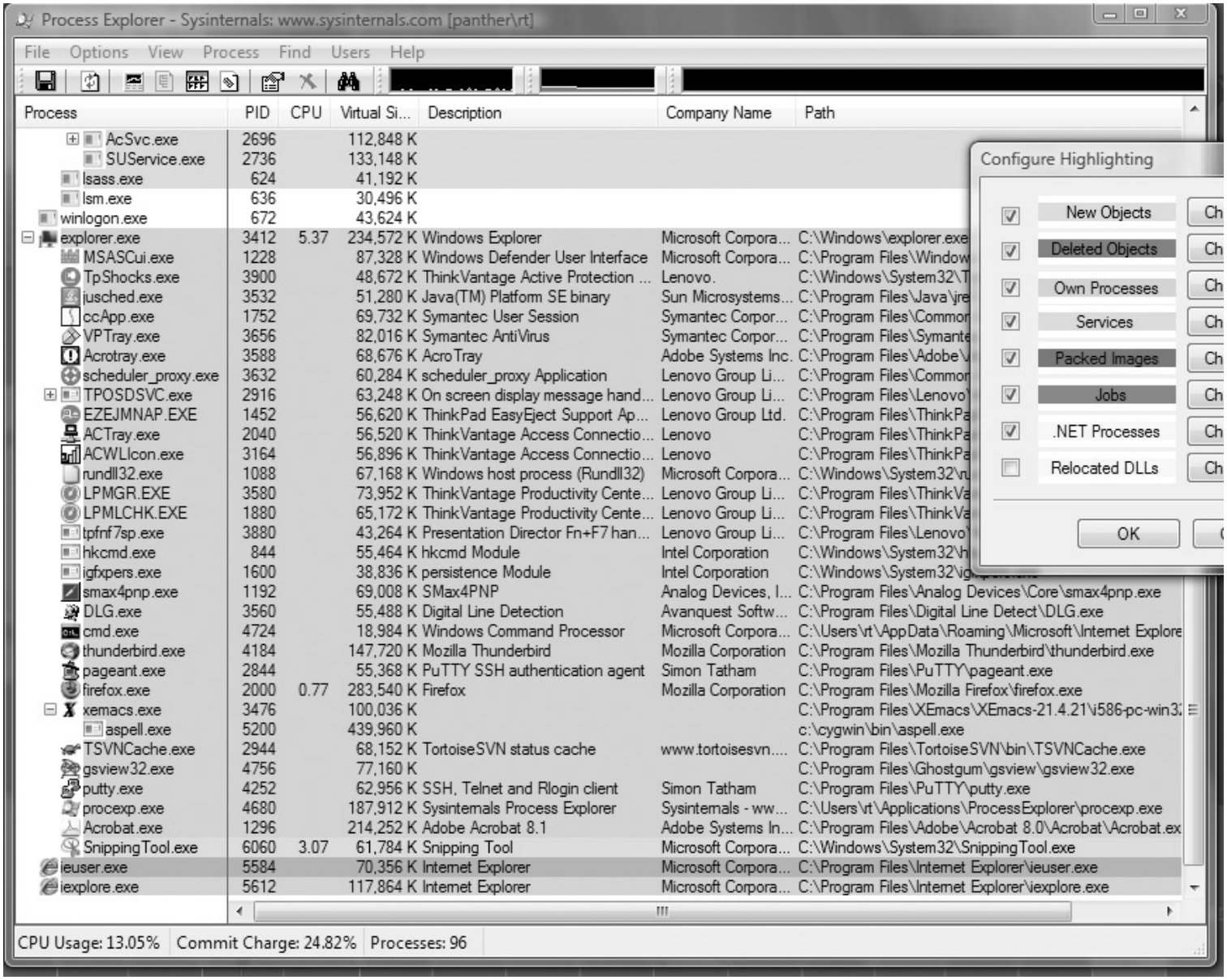
1. User closes a laptop computer, putting it into hibernation.

2. Attacker copies the hiberfil.sys file to discover any unencrypted passwords that were stored in memory when the computer was put into hibernation.

Event Logging

- Keeping track of what processes are running, what other machines have interacted with the system via the Internet, and if the operating system has experienced any unexpected or suspicious behavior can often leave important clues not only for troubleshooting ordinary problems, but also for determining the cause of a security breach.

Process Explorer



Memory and Filesystem Security

- The contents of a computer are encapsulated in its memory and filesystem.
- Thus, protection of a computer's content has to start with the protection of its memory and its filesystem.

Password Security

- The basic approach to guessing passwords from the password file is to conduct a **dictionary attack**, where each word in a dictionary is hashed and the resulting value is compared with the hashed passwords stored in the password file.
- A dictionary of 500,000 “words” is often enough to discover most passwords.

Password Salt

- One way to make the dictionary attack more difficult to launch is to use **salt**.
- Associate a random number with each userid.
- Rather than comparing the hash of an entered password with a stored hash of a password, the system compares the hash of an entered password and the salt for the associated userid with a stored hash of the password and salt.

How Password Salt Works

Without salt:

1. User types userid, X, and password, P.

2. System looks up H, the stored hash of X's password.

3. System tests whether h(P) = H.

Password file:

…
X: H
…

With salt:

1. User types userid, X, and password, P.

2. **System looks up S and H**, where S is the random salt for userid X and H is stored hash of S and X's password.

3. System tests whether h(S||P) = H.

Password file:

…
X: S, H
…

How Salt Increases Search Space Size

- Assuming that an attacker cannot find the salt associated with a userid he is trying to compromise, then the search space for a dictionary attack on a salted password is of size

  $2^B * D$,

  where B is the number of bits of the random salt and D is the size of the list of words for the dictionary attack.
- For example, if a system uses a 32-bit salt for each userid and its users pick passwords in a 500,000 word dictionary, then the search space for attacking salted passwords would be

  $2^{32} * 500{,}000 = 2{,}147{,}483{,}648{,}000{,}000$,

  which is over 2 quadrillion.
- Also, even if an attacker can find a salt password for a userid, he only learns one password.

Filesystem Security

General Principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder
- A file handle provides an opaque identifier for a file/folder
- File operations
  - Open file: returns file handle
  - Read/write/execute file
  - Close file: invalidates file handle
- Hierarchical file organization
  - Tree (Windows)
  - DAG (Linux)

Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Alternative model not covered in this lecture
  - Multiple levels of security for users and documents
  - Read down and write up principles

Closed vs. Open Policy

Closed policy

- Also called “default secure”
- Give Tom read access to “foo”
- Give Bob r/w access to “bar
- Tom: I would like to read “foo”
  - Access allowed
- Tom: I would like to read “bar”
  - Access denied

Open Policy

- Deny Tom read access to “foo”
- Deny Bob r/w access to “bar”
- Tom: I would like to read “foo”
  - Access denied
- Tom: I would like to read “bar”
  - Access allowed

Closed Policy with Negative Authorizations and Deny Priority

- Give Tom r/w access to “bar”
- Deny Tom write access to “bar”
- Tom: I would like to read “bar”
  - Access allowed
- Tom: I would like to write “bar”
  - Access denied
- Policy is used by Windows to manage access control to the file system

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)
- An ACE specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “Bob’s CS167 Grades”
  - Bob; Read; Allow
  - TAs; Read; Allow
  - TWD; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

Linux vs. Windows

- Linux
  - Allow-only ACEs
  - Access to file depends on ACL of file and of all its ancestor folders
  - Start at root of file system
  - Traverse path of folders
  - Each folder must have execute (cd) permission
  - Different paths to same file not equivalent
  - File's ACL must allow requested access
- Windows
  - Allow and deny ACEs
  - By default, deny ACEs precede allow ones
  - Access to file depends only on file's ACL
  - ACLs of ancestors ignored when access is requested
  - Permissions set on a folder usually propagated to descendants (inheritance)
  - System keeps track of inherited ACE's

Linux File Access Control

- File Access Control for:
  - Files
  - Directories
  - Therefore…
    - \dev\ : *devices*
    - \mnt\ : *mounted file systems*
    - What else? *Sockets, pipes, symbolic links*…

Linux File System

- Tree of directories (folders)
- Each directory has links to zero or more files or directories
- Hard link
  - From a directory to a file
  - The same file can have hard links from multiple directories, each with its own filename, but all sharing owner, group, and permissions
  - File deleted when no more hard links to it
- Symbolic link (symlink)
  - From a directory to a target file or directory
  - Stores path to target, which is traversed for each access
  - The same file or directory can have multiple symlinks to it
  - Removal of symlink does not affect target
  - Removal of target invalidates (but not removes) symlinks to it
  - Analogue of Windows shortcut or Mac OS alias

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use ls –l
- For the first character a - (hyphen) indicates a plain file, d a directory and l a soft link.
- The remaining 9 characters are split into 3 components of 3 characters each, to describe the user's, group's and others' privileges respectively.
- Within each 3 character triplet, the first indicates read permission, the second is for write permission and the third is for execute permission.
- If the r, w or x character is present, the permission is allowed, if the position is occupied by a - (hyphen) the permission is denied

Directory Rights

- On a directory, the x permission means something different from the ability to execute a file.
- Directories can't be executed. Here the x permission means the ability to search through, or *traverse* a directory to access subdirectories, whether or not you are allowed to read the directory being traversed. Being allowed to write to a directory enables files to be renamed, or deleted within it.

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use `ls -l`

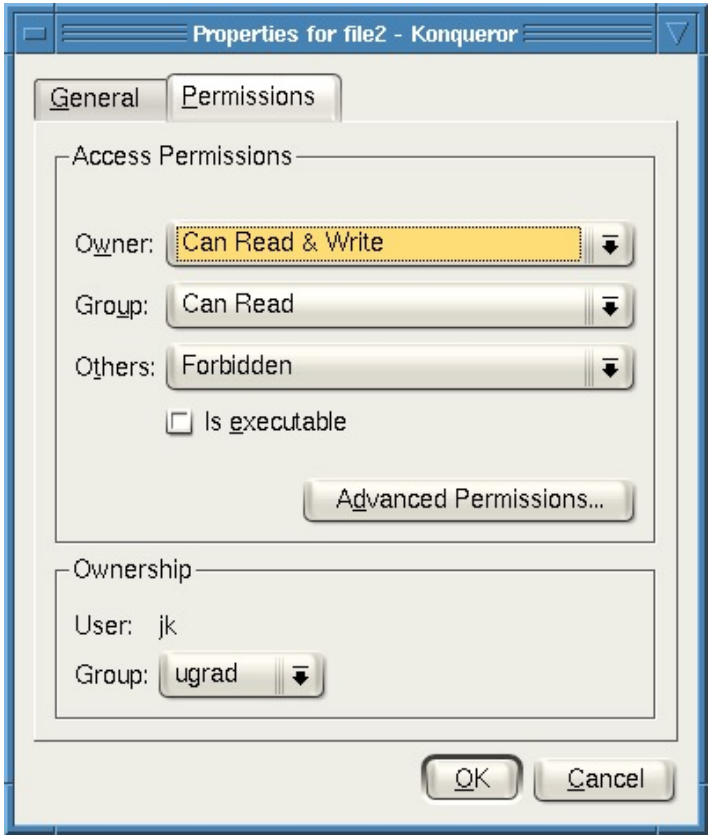
```
jk@sphere:~/test$ ls -l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

File Permission Examples

-rwxr-xr--	indicates user read, write and execute rights, group read and execute rights, and read rights only for others.
drwxr-xr--	indicates a directory which can be displayed and searched by user and group. User can also delete, create and rename files in the directory. Others may list the directory but not search through it or access files in it.

Working Graphically with Permissions

- Several Linux GUIs exist for displaying and changing permissions
- In KDE's file manager Konqueror, right-click on a file and choose Properties, and click on the Permissions tab:
- Changes can be made here (more about changes later)



Sharing data on a multiuser system

- Giving write access on your own directory isn't a safe way to share data. This gives the ability to do anything (accidently or deliberately) to any of the contents of the directory.
- If you want to share your files, allocate read and execute permission to the **directory** and read permission to your **files**. Others can then read them or copy them into their own directories.
- A multi-user open system in an academic environment is not very secure, but some protection can be given by putting files into a directory to which only you have access.

The chmod command

File permissions are assigned using the chmod command. This can be used in one of two ways. Some users prefer to give the octal 3 digit mode, others prefer to use the character equivalents.

Examples using octal modes:

chmod 421 f1 assigns permissions r---w---x to f1

chmod 750 f1 f2 assigns permissions rwxr-x--- to f1 and f2

Each octal digit adds 4 for read, 2 for write and 1 for execute permission. The first digit is for user, second for group and the third for others.

The chmod command (continued)

Examples using mnemonic modes

`chmod u=rwx f1`
`chmod g=rx f1`
`chmod o= f1`
	these three commands assign permissions rwxr-x--- to f1

`chmod -R go-w dir1`	removes write permission from group and others from directory dir1 including all objects contained in it and all its subdirectories etc.
`chmod u+x *.exe`	adds user execute rights to all files in current directory not prefixed by period (.) ending in .exe

Octal Notation Examples

644 or 0644	read/write for owner, read-only for everyone else
775 or 0775	read/write/execute for owner and group, read/execute for others
640 or 0640	read/write for owner, read-only for group, forbidden to others
2775	same as 775, plus setgid (useful for directories)
777 or 0777	read/write/execute to everyone *(dangerous!)*
1777	same as 777, plus sticky bit

Special Permission Bits

- Three other permission bits exist
  - Set-user-ID (“setuid”) bit
  - Set-effective-user-ID (seteuid) bit
  - Set-group-ID (“setgid”) bit
  - Sticky bit

Set-user-ID

- Set-user-ID ("suid" or "setuid") bit
  - On executable files, causes the program to run as file owner regardless of who runs it
  - Ignored for everything else
  - In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)

    -rw**s**r-**x**r-**x**: setuid, executable by all

    -rw**x**r-**x**r-**x**: executable by all, but not setuid

    -rw**S**r--r--: setuid, but not executable - not useful

Setuid example

rich@saturn:~$ ls -l /usr/bin/passwd

-rwsr-xr-x 1 root root 27132 2006-07-11 13:51 /usr/bin/passwd

When a user changes their own password using the passwd command, this program is run by the user, but runs with the UID of its owner (root).

Set-group-ID

- Set-group-ID ("sgid" or "setgid") bit
  - On executable files, causes the program to run with the file's group, regardless of whether the user who runs it is in that group
  - On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups
  - Ignored for everything else
  - In 10-character display, replaces 7th character (x or -) with s (or S if not also executable)

    -rw**x**r-**s**r-**x**: setgid file, executable by all

    drw**x**rw**s**r-**x**: setgid directory; files within will have group of directory

    -rw-r-**S**r--: setgid file, but not executable - not useful

Setgid example

rich@saturn:~$ ls -l /usr/bin/procmail

-rwsr-sr-x 1 root mail 68152 2005-05-03 03:10 /usr/bin/procmail

rich@saturn:~$ ls -l /var/mail

total 15652

-rw-rw---- 1 rich mail 15984976 2006-10-18 20:03 rich

-rw-rw---- 1 test mail 1810 2006-10-15 08:20 test

-rw-rw---- 1 rich mail 4333 2006-09-26 07:00 trap

When user1 uses procmail to send an email to user2, procmail needs to be able to write to the mail spool (/var/mail/user2) of the other user, which has group ownership of mail and group write access. The write access by user1 to user2's mailbox is restricted to what the procmail program is programmed to do, i.e. deliver a message.

Sticky Bit

- On directories, prevents users from deleting or renaming files they do not own
- Ignored for everything else
- The sticky bit is used on directories such as /tmp where all users have write access, but not to each others' files.
- In 10-character display, replaces 10th character (x or -) with t (or T if not also executable)

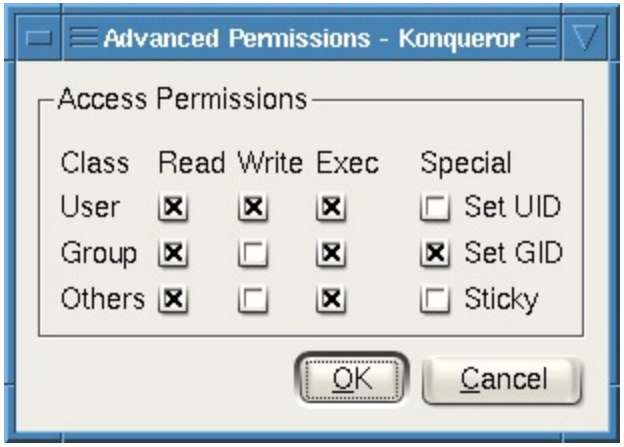
drwxrwxrwt: sticky bit set, full access for everyone

drwxrwx--T: sticky bit set, full access by user/group

drwxr--r-T: sticky, full owner access, others can read *(useless)*

Working Graphically with Special Bits

- Special permission bits can also be displayed and changed through a GUI
- In Konqueror's Permissions window, click Advanced Permissions:
- Changes can be made here (more about changes later)



Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn't touch most of environment and doesn't run login scripts
- su -
  - Logs in as root just as if root had done so normally

- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

Limitations of Unix Permissions

- Unix permissions are not perfect
  - Groups are restrictive
  - Limitations on file creation
- Linux optionally uses POSIX ACLs
  - Builds on top of traditional Unix permissions
  - Several users and groups can be named in ACLs, each with different permissions
  - Allows for finer-grained access control
- Each ACL is of the form *type*:[*name*]:*rwx*
  - Setuid, setgid, and sticky bits are outside the ACL system

Daemons and Services

- Computers run dozens of processes that run without any user intervention.
- In linux, these background processes are know as **daemons**.
- They are indistinguishable from any other process, are started by **init** process and operate at varying levels of permissions.
- They are forked before the user is authenticated, and able to run with higher permissions than any user, and survive the end of the login sessions.
- Examples are processes that control web servers, remote logins, and print servers.

Daemons and Services

- Windows have an equivalent class of processes known as services.
- They are easily distinguishable from other processes,
- are differentiated in monitoring software such as Task Manager.

Stack-based BoF

- The stack is the component of the memory address space of a process that contains data associated with the function calls.
- The stack consists of frames.
- A frame stores the local variables and arguments of the call and the return address of the parent call.
- This structure allows for the CPU to know where to return to when a method terminates.

Stack-based BoF (cond)

- In a BoF attack, an attacker provides the input to the program that is larger than a buffer can hold.
- This commonly occurs with the use of unchecked C library functions such as strcpy() and gets(), which copy user input without checking its length.

How to seize control of execution

- How to guess the location of the return address with respect to buffer?
- OS design makes this job challenging
  - Processes cannot access the address spaces of other processes, so the malicious code must reside within the address space of the exploited process
  - The address space of a given process is unpredictable and may change when a program is run on different machines.

How to seize control of execution- Vulnerabilities and Attack Method

- Vulnerability scenarios
  - The program has root privileges (setuid) and is launched from a shell
  - The program is part of a web application
- Typical attack method
  1. Find vulnerability
  2. Reverse engineer the program
  3. Build the exploit

Techniques to overcome above challenges

- NOP Sledding: is a CPU instruction that does not actually do anything except tell the processor to proceed to the next instruction.
- Jump-to-register or trampolining: a particular code such as Windows DLL tells the processor to jump to the address stored in on the processor's registers, such as ESP. If the malicious code is placed at the address pointed by ESP and overwrite the return address, the application will jump and execute the malicious code.

The Return-to-libc Attack

- If an attacker can determine the address of a C library function within a vulnerable process's address space, such as system() or execv.
- The attacker can force the program to call this function.
- Same technique, but replace the return address with the address of the desired library function.

Shellcode

- A malicious coded included in an exploit to spawn a terminal or shell, allowing them to issue further commands.
- It is executed directly on the stack by the CPU, it must be written in assembly language, low-level processor instructions, which vary by CPU architecture.
- BoF is commonlly used as a means of privilege escalation. For example, if a SETUID program (stack.c) is vulnerable to BoF attacks, the attacker can gain a shell with the permission of the process's owner.

Preventing Stack-based BoF Attacks

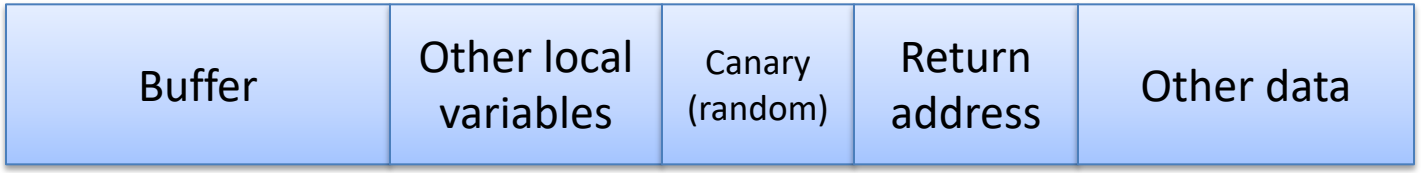
- The root cause does not come from OS but insecure programming practices.
- Programmers must be educated about the risks of insecurely coping user-supplied data into fixed size buffers.
- Use strncpy(buf, argv[1], sizeof(buf)) instead of strcpy(buf, argv[1])

Preventing Stack-based BoF Attacks -- strcpy() vs. strncpy()

- Function strcpy() copies the string in the second argument into the first argument
  - e.g., strcpy(dest, src)
  - If source string > destination string, the overflow characters may occupy the memory space used by other variables
- Function strncpy() copies the string by specifying the number **n** of characters to copy
  - e.g., strncpy(buf, argv[1], sizeof(buf))
  - If source string is longer than the destination string, the overflow characters are discarded automatically

Preventing Stack-based BoF Attacks -- using a random canary

Normal (safe) stack configuration:



Buffer overflow attack attempt:

- The canary is placed in the stack prior to the return address, so that any attempt to over-write the return address also over-writes the canary.
- The system regularly checks the integrity of this canary value. If it has been changed, it knows that the buffer has been overflowed and it should prevent malicious code execution.

Preventing Stack-based BoF Attacks (cond)

- PointGuard by Microsoft. It is a compiler extension, that address code which XOR-encodes any pointers, including the return address before and after they are used. Therefore, attacker cannot reliably overwrite the return address.
- Enforce no-execution permission on the stack segment of memory
- Address space layout randomization (ASLR) rearranges the data of a process's address space at random.

Race Condition

- A race condition is any situation where the behavior of the program is unintentionally dependent on the timing of certain event.
- We have a simple program that takes a filename as an argument.
- Checks whether the user running the program has permission to open that file, and if so, reads the first few characters of the file and prints them.

We assume this program is a SETUID program.

```
Int main(int argc, char * argv[]) {
        int file;
        char buf[1024];
        memset(buf, 0, 1024);
        if(access(argv[1], R_OK) != 0){
        //CHECK: zero means permitted, non-zero means denied
                printf("cannot access file. \n");
                exit(-1);
        }
        file = open(argv[1], O_RDONLY);
        //READ
        read(file, buf, 1023);
        close(file);
        printf(%s\n", buf);
        return 0;
}
```

What an attacker can do?

- Exploit the delay b/w the calls to access() and open()
- Attacker can provide an innocent file “/home/joe/dummy” as an argument, which he has access to it.
- After the access() returns 0, he quickly **replace** /home/joe/dummy with a symbolic link to a file that he does not have permission to read such as /etc/password.
- Because the program is SETUID, it can open any files accessible to the root user.

Time of Check/Time of Use (TOCTOU) Problem

- Any time a program checks the validity and authorization for an object, whether it be a file or some other property, before performing an action on that object, care should be taken.
- Two operations need to be performed atomically, which means they need to be performed as a single uninterruptible operation.
- Otherwise objects may be changed in between.

```
Int main(int argc, char * argv[]){
        int file;
        char buf[1024];
        uid_t uid, euid;
        memset(buf, 0, 1024);
        if(argc < 2) {
            printf("Usage: printer [filename]\n");
                exit(-1);
        }
        euid = geteuid();
        uid = getuid();
        /* Drop privileges */
        seteuid(uid);
        file = open(argv[1], O_RDONLY);
        read(file, buf, 1023);
        close(file);
        /* restore privileges */
        seteuid(euid);
        printf("%s\n", buf);
        return 0;
}
```

Drop privileges using seteuid before calling open(). open() uses euid instead of uid.