

ECE 455: CYBERSECURITY

Lecture #4

Daniel Gitzel

In the news...

- **Firmware exploit for older iPhones. Requires physical access, in-memory only (cleared on reboot).**

Got a pre-A12 iPhone? Love jailbreaks? Happy Friday! 'Unpatchable tethered Boot ROM exploit' released

Coder claims iThings older than two years can be unlocked from Apple's clutches

By [Shaun Nichols in San Francisco](#) 27 Sep 2019 at 22:22

35 

Announcement

- **Microsoft Teams outage on 9/28 impacted Lecture 4**
 - Impacts lecture schedule
- **Quiz #1 will be posted by Wed 10/7 on Teams (take-home)**
- **Lab #1 (Buffer Overflows) will be posted by Wed 10/7 on Teams**

MORE ON BUFFER OVERFLOWS

Introduction

- **Typical memory exploit involves code injection**
 - Put malicious code at a predictable location in memory, usually masquerading as data
 - Trick vulnerable program into passing control to it
- **Possible defenses:**
 - Prevent execution of untrusted code
 - Stack “canaries”
 - Encrypt pointers
 - Address space layout randomization
 - Code analysis

Executable Space Protection

- **Mark all writeable memory locations as non-executable**
 - Example: Microsoft's Data Execution Prevention (DEP)
 - This blocks many code injection exploits
- **Hardware support**
 - AMD "NX" bit (no-execute), Intel "XD" bit (executed disable) (in post-2004 CPUs)
 - Makes memory page non-executable
- **Widely deployed**
 - Windows XP SP2+ (2004), Linux since 2004 (check distribution), OS X 10.5+ (10.4 for stack but not heap), Android 2.3+

What Does “Executable Space Protection” Not Prevent?

- **Can still corrupt stack ...**
 - ... or function pointers or critical data on the heap
 - As long as “saved EIP” points into existing code, executable space protection will not block control transfer
- **This is the basis of return-to-libc exploits**
 - Overwrite saved EIP with address of any library routine, arrange stack to look like arguments
- **Does not look like a huge threat**
 - Attacker cannot execute arbitrary code

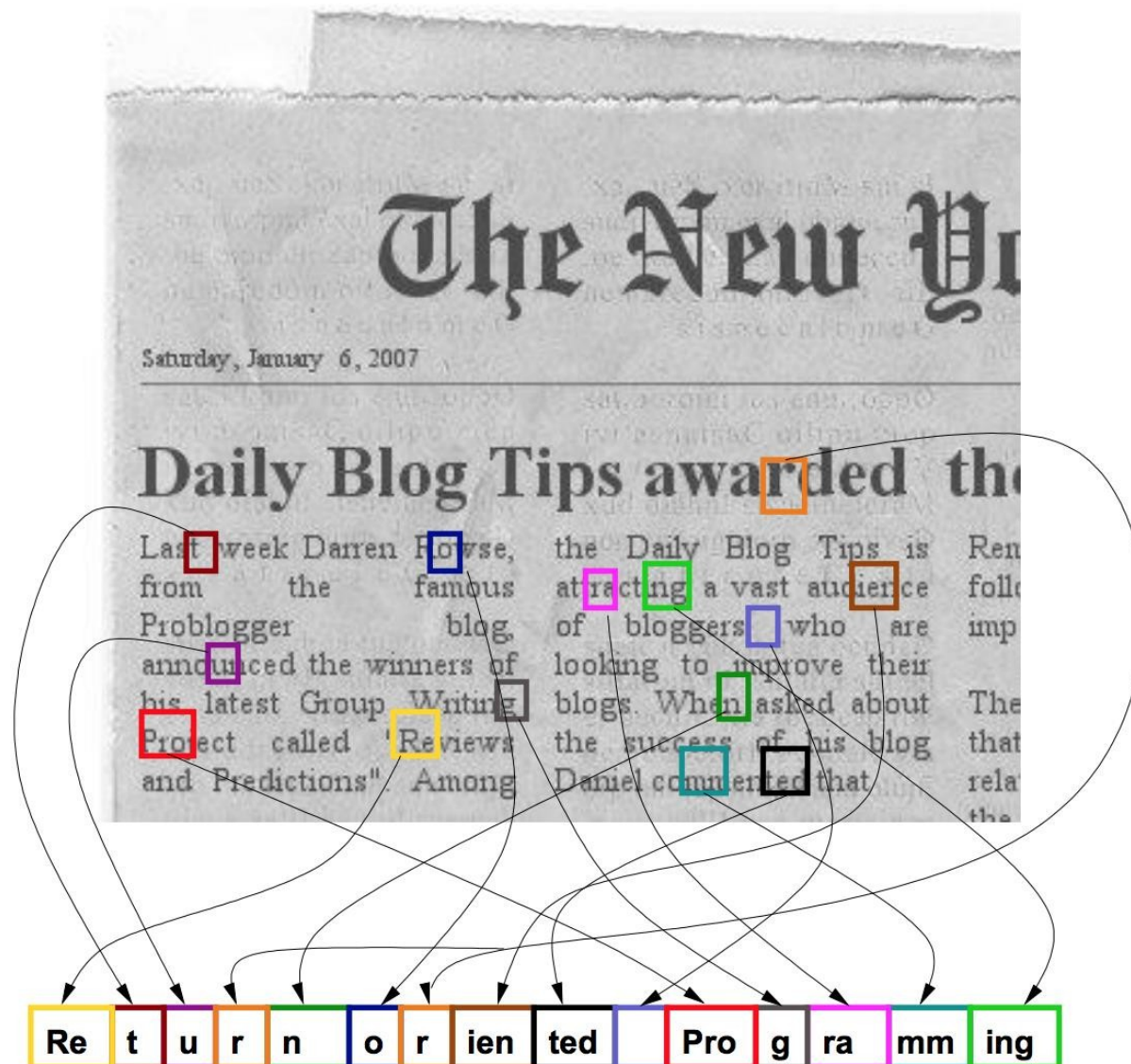
return-to-libc

- **Can still call critical functions, like exec**
- **Overwritten saved EIP need not point to the beginning of a library routine**
- **Any existing instruction in the code image is fine**
 - Will execute the sequence starting from this instruction
- **What if instruction sequence contains RET?**
 - Execution will be transferred... to where?
 - Read the word pointed to by stack pointer (ESP)
- **Guess what? Its value is under attacker's control!**
 - Use it as the new value for EIP
- **Now control is transferred to an address of attacker's choice!**
 - Increment ESP to point to the next word on the stack

Chaining RETs for Fun and Profit

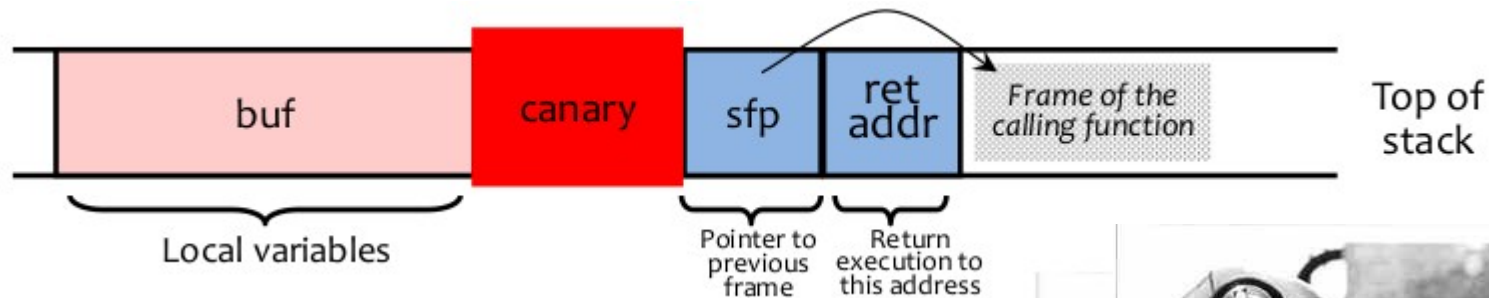
- **Can chain together sequences ending in RET**
 - Krahmer, “x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique” (2005)
- **What is this good for?**
- **Answer [Shacham et al.]: everything**
 - Turing-complete language
 - Build “gadgets” for load-store, arithmetic, logic, control flow, system calls
 - Attack can perform arbitrary computation using no injected code at all – return-oriented programming

Return-Oriented Programming



Run-Time Checking: StackGuard

- Embed “canaries” (stack cookies) in stack frames and verify their integrity prior to function return
- Any overflow of local variables will damage the canary



Run-Time Checking: StackGuard

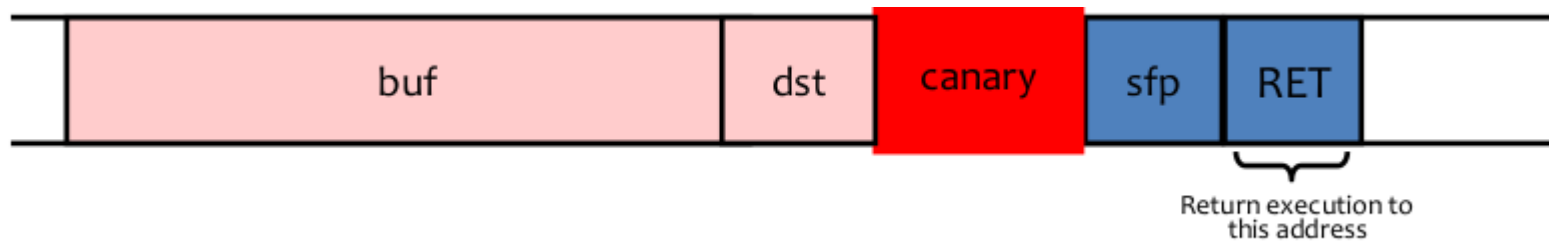
- **Embed “canaries” (stack cookies) in stack frames and verify their integrity prior to function return**
 - Any overflow of local variables will damage the canary
- **Choose random canary string on program start**
 - Attacker can't guess what the value of canary will be
- **Terminator canary: “\0”, newline, linefeed, EOF**
 - String functions like strcpy won't copy beyond “\0”

StackGuard Implementation

- **StackGuard requires code recompilation**
- **Checking canary integrity prior to every function return causes a performance penalty**
 - For example, 8% for Apache Web server at one point in time
- **StackGuard can be defeated**
 - A single memory write where the attacker controls both the value and the destination is sufficient

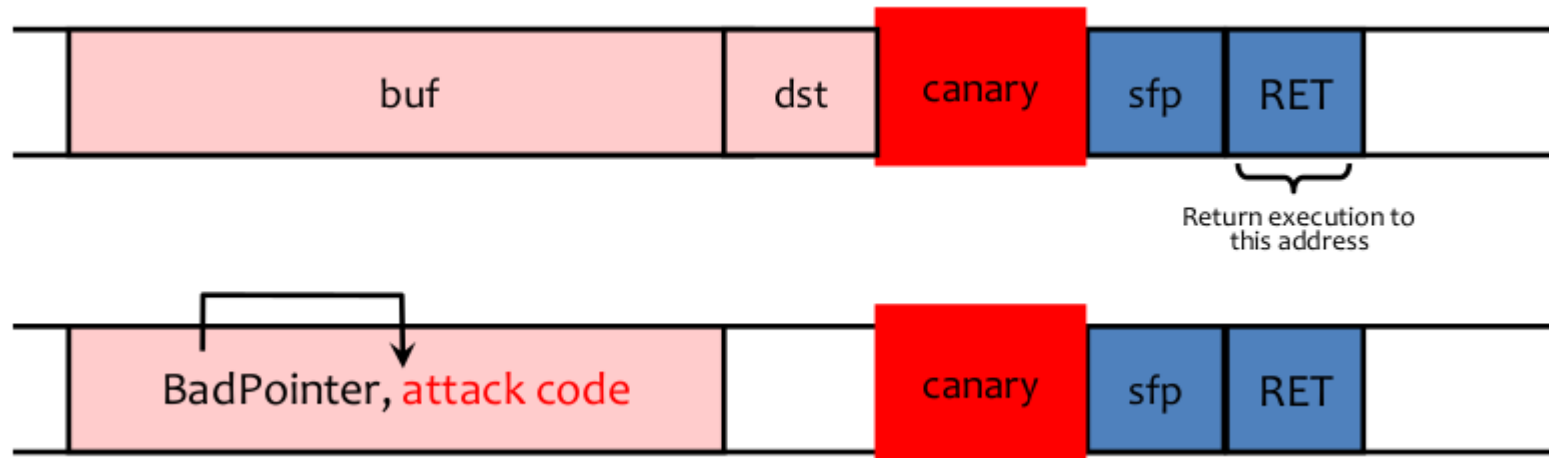
Defeating StackGuard

- **Suppose program contains strcpy(dst,buf) where attacker controls both dst and buf**
 - Example: dst is a local pointer variable



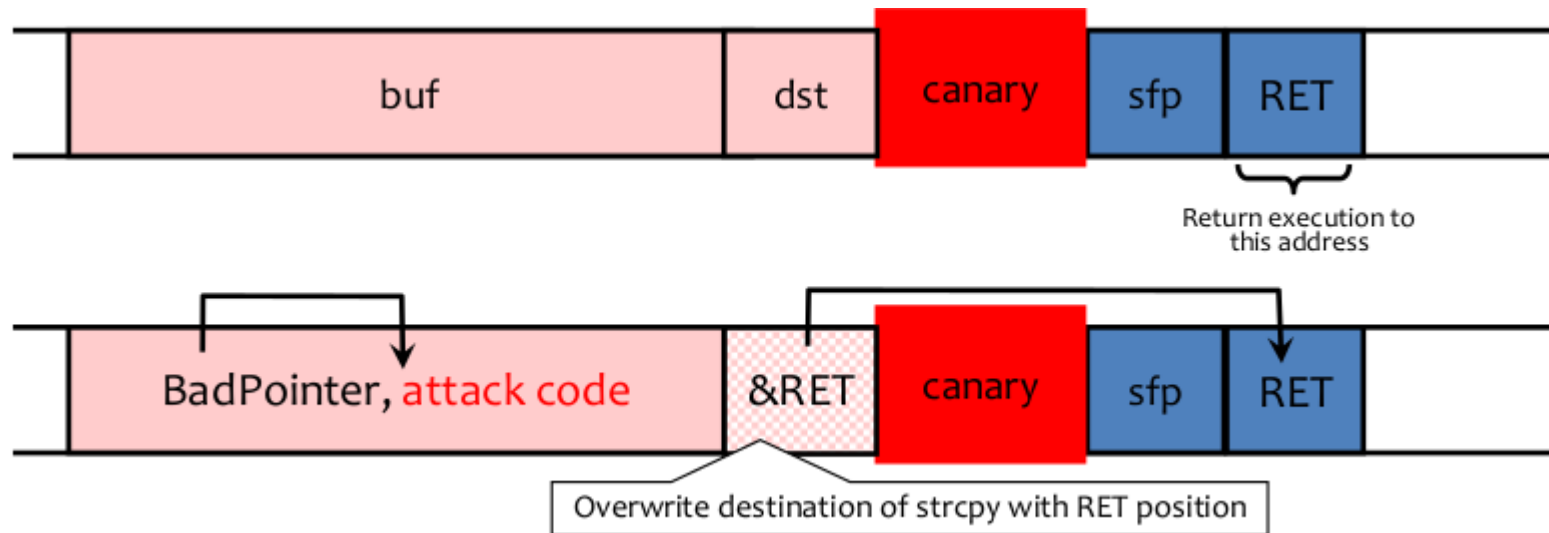
Defeating StackGuard

- **Suppose program contains strcpy(dst,buf) where attacker controls both dst and buf**
 - Example: dst is a local pointer variable



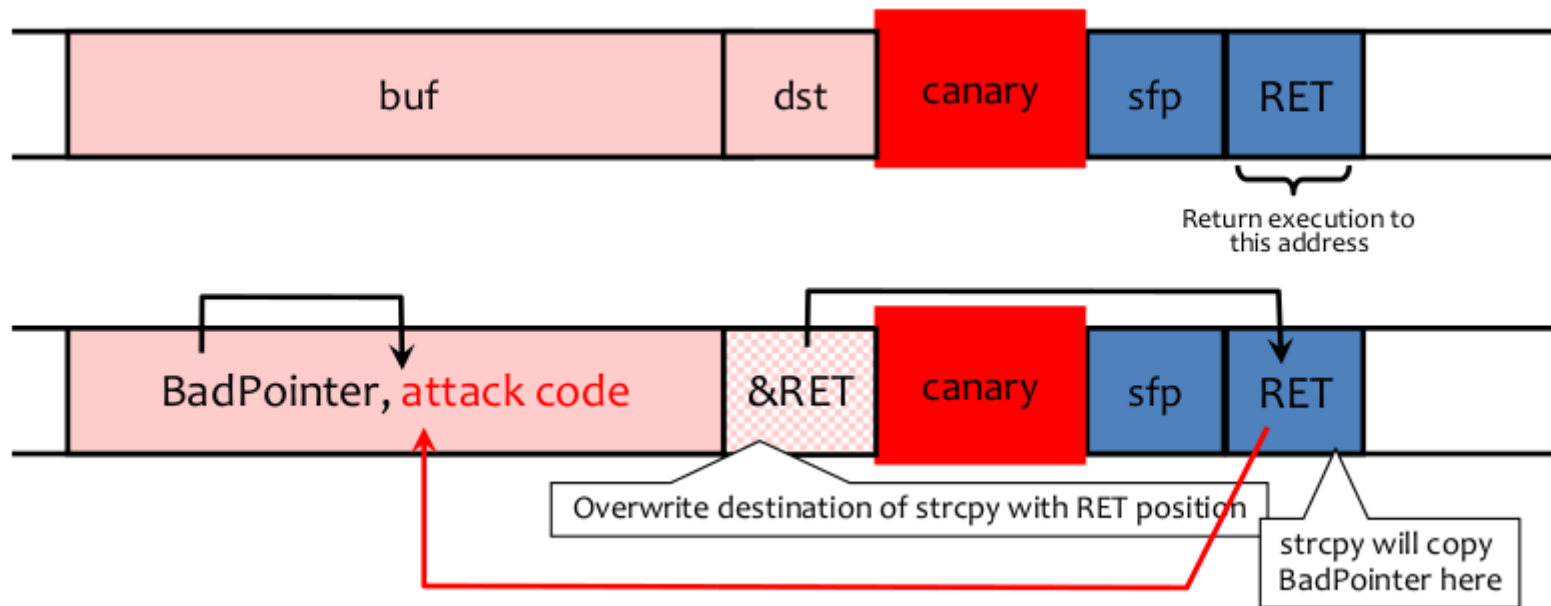
Defeating StackGuard

- **Suppose program contains strcpy(dst,buf) where attacker controls both dst and buf**
 - Example: dst is a local pointer variable



Defeating StackGuard

- **Suppose program contains strcpy(dst,buf) where attacker controls both dst and buf**
 - Example: dst is a local pointer variable



More on Defeating StackGuard

- **Attacker sets buf to contain (first) a pointer to another region in buf with the attack code, and then (second) the attack code**
- **Attacker sets dst, to contain the address where RET is stored (recall the assumption that the attacker can also set dst)**
- **When the strcpy happens, memory beginning at the address of RET is overwritten with the contents of buf**
 - This puts “BadPointer” in the location of RET
 - Recall that “BadPointer” is a value for the address at which the attack code starts (in buf)

ASLR: Address Space Randomization

- **Randomly arrange address space of key data areas for a process**
 - Base of executable region
 - Position of stack
 - Position of heap
 - Position of libraries
- **Introduced by Linux PaX project in 2001**
- **Adopted by OpenBSD in 2003**
- **Adopted by Linux in 2005**

ASLR: Address Space Randomization

- **Deployment (examples)**
 - Linux kernel since 2.6.12 (2005+)
 - Android 4.0+
 - iOS 4.3+ ; OS X 10.5+
 - Microsoft since Windows Vista (2007) (not by default)
- **Attacker goal: Guess or figure out target address (or addresses)**
- **ASLR more effective on 64-bit architectures**

ASLR Issues

- **NOP sleds and heap spraying to increase likelihood for custom code (e.g., on heap)**
- **Brute force attacks or memory disclosures to map out memory on the fly**
 - Disclosing a single address can reveal the location of all code within a library, depending on the ASLR implementation

NOP Slide

- **Attacks rely on jumping to a specific address and continue running from there.**
 - The injected code has been loaded to exact location.
- **Stack randomization makes the jump impossible to predict**
 - Attacker places a “NOP sled” in a large range of memory.
 - If the program jumps to anywhere into the sled, it will run all the remaining NOPs, doing nothing, and then will run the payload code, just after the sled.
- **The NOP sled makes the target address bigger: the code can jump anywhere in the sled, instead of exactly at the beginning of the injected code.**

Heap Spraying “Spray and Pray”

- **Ensure that the bytes can be accessed later as the vector of a separate attack.**
 - Malicious software can use a pointer reference to execute the arbitrary code.
 - If the heap is sprayed all over with the code to be executed, the chances that the pointer will reference the code is very high.
 - Therefore, the heap spray is not actually an exploit, but a way to give other exploits a higher chance of success.
- **Heap spraying was first used in web browser exploits and was first identified as a technique in the early 2001.**
 - Heap spraying attacks have been demonstrated using JavaScript, VBScript, and HTML5.

Other Possible Solutions

- **Use safe programming languages, e.g., Java**
 - What about legacy C code?
 - (Though Java doesn't magically fix all security issues...)
 - Static analysis of source code to find overflows
 - Dynamic testing: "fuzzing"
 - Modern compiler options, e.g., incorporate stack canaries

Another Type of Vulnerability

- **Goal: only open regular files**
 - What could go wrong?

```
int openfile(char *path) {  
    struct stat s;  
    if (stat(path, &s) < 0)  
        return -1;  
    if (!S_ISREG(s.st_mode)) {  
        error("only allowed to regular files!");  
        return -1;  
    }  
    return open(path, O_RDONLY);  
}
```

TOCTOU (Race Condition)

- **TOCTOU == Time of Check to Time of Use:**
- **Attacker can change meaning of path between stat and open (and access files he or she shouldn't)**

TOCTOU (Race Condition)

- In call to open, pass **O_NOFOLLOW** to not follow symbolic links
- Call **fstat** on open file descriptor
- Nice reference:
 - <https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html>

Another Type of Vulnerability

- **Consider this code:**

- What could go wrong?

```
char buf[80];
void vulnerable() {
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if (len > sizeof buf) {
        error("length too large, nice try!");
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dst, const void * src, size_t n);
typedef unsigned int size_t;
```

Another Type of Vulnerability

- **Consider this code:**

- Implicit cast, if len is negative, may copy large amount of input!

```
char buf[80];
void vulnerable() {
    int len = read_int_from_network();
    char *p = read_string_from_network();
    if (len > sizeof buf) {
        error("length too large, nice try!");
        return;
    }
    memcpy(buf, p, len);
}
```

```
void *memcpy(void *dst, const void * src, size_t n);
typedef unsigned int size_t;
```

Yet Another Example

- **Consider this code:**
 - What could go wrong?

```
size_t len = read_int_from_network();  
char *buf;  
buf = malloc(len+5);  
read(fd, buf, len);
```

Integer Overflow

- **Consider this code:**

- What if len is large (e.g., len = 0xFFFFFFFF)
- Then len + 5 = 4 (on many platforms)
- Result: Allocate a 4-byte buffer, then read a lot of data into that buffer.

```
size_t len = read_int_from_network();  
char *buf;  
buf = malloc(len+5);  
read(fd, buf, len);
```

Software Testing

- **So what should we do?**
- **How can we detect these vulnerabilities?**

Fuzz Testing

- **Generate “random” inputs to program**
 - Sometimes conforming to input structures (file formats, etc.)
- **See if program crashes**
 - If crashes, found a bug
 - Bug may be exploitable
- **Surprisingly effective**
- **Now standard part of development life-cycle**

General Principles

- **Check inputs**
- **Check all return values**
- **Least privilege**
- **Securely clear memory (passwords, keys, etc.)**
- **Fail-safe defaults**
- **Defense in depth**
 - Also: prevent, detect, respond
- **NOT: security through obscurity**

General Principles

- **Reduce size of trusted computing base (TCB)**
- **Simplicity, modularity**
 - But: Be careful at interface boundaries!
- **Minimize attack surface**
- **Use vetted component**
- **Security by design**
 - But: tension between security and other goals
- **Open design? Open source? Closed source?**
 - Different perspectives

Vulnerability Analysis and Disclosure

- **What do you do if you've found a security problem in a real system?**
 - A commercial website?
 - UW grade database?
 - Boeing 787?
 - TSA procedures?

Vulnerability Analysis and Disclosure

- **Suppose companies A, B, and C all have a vulnerability, but have not made the existence of that vulnerability public**
 - Company A has a software update prepared and ready to go that, once shipped, will fix the vulnerability; but B and C are still working on developing a patch for the vulnerability
 - Company A learns that attackers are exploiting this vulnerability in the wild
 - Should Company A release their patch, even if doing so means that the vulnerability now becomes public and other actors can start exploiting Companies B and C?
 - Should Company A wait until Companies B and C have patches?

BREAK

UNIX SECURITY MODEL

Unix Preliminaries

- **Unix (like the Internet) was developed for friendly environments like research labs or universities.**
- **Security mechanisms were quite weak and elementary; improved gradually.**
- **Several flavors of Unix; vendor versions differ in the way some security controls are managed & enforced.**
 - Commands and filenames used in this lecture are indicative of typical use but may differ from actual systems.
- **Unix designed originally for small multi-user computers in a network environment; later scaled up to commercial servers and down to PCs.**

Unix Design Philosophy

- **Security managed by skilled administrator, not by user.**
 - Command line tools and scripting.
 - Archaic syntax retained; those who know it, love it (saves keystrokes!).
- **Focus on:**
 - protecting users from each other.
 - protecting against attacks from the network.
- **Discretionary access control with a granularity of owner, group, other.**
- **“Secure” versions of Unix: Trusted Unix or Secure Unix often indicates support for multi-level security.**

Principals

- **Principals: user identifiers (UIDs) and group identifiers (GIDs).**
- **A UID (GID) is a 16-bit number; examples:**
 - 0: root
 - 1: bin
 - 2: daemon
 - 8: mail
 - 9: news
 - 261: peter
- **UID values differ from system to system**
- **Superuser (root) UID is always zero.**

Superuser

- **The superuser is a special privileged principal with**
- **UID 0 and usually the user name root.**
- **There are few restrictions on the superuser:**
 - All security checks are turned off for superuser.
 - The superuser can become any other user.
 - The superuser can change the system clock.
- **Superuser cannot write to a read-only file system but can remount it as writeable.**
- **Superuser cannot decrypt passwords but can reset them.**

Groups

- **Users belong to one or more groups.**
- **/etc/group contains all groups; file entry format: groupname:password:GID:list of users**
- **Example:**
 - infosecwww:*:209:carol,al
- **Every user belongs to a primary group; group ID**
- **(GID) of the primary group stored in /etc/passwd.**
- **Collecting users in groups is a convenient basis for access control decisions.**
 - For example, put all users allowed to access email in a group called mail or put all operators in a group operator.

Subjects

- **The subjects in Unix are processes; a process has a process ID (PID).**
- **New processes generated with exec or fork.**
- **Processes have a real UID/GID and an effective UID/GID.**
- **Real UID/GID: inherited from the parent; typically UID/GID of the user logged in.**
- **Effective UID/GID: inherited from the parent process or from the file being executed.**
- **POSIX compliant versions add saved UID/GID**

Objects

- **Files, directories, memory devices, I/O devices are uniformly treated as resources.**
- **These resources are the objects of access control.**
- **Resources organized in a tree-structured file system.**
- **Each file entry in a directory is a pointer to a data structure called inode.**

Inode

mode	type of file and access rights
uid	username of the owner
gid	owner group
atime	access time
mtime	modification time
itime	inode alteration time
block count	size of file
	physical location

Default Permissions

- **Unix utilities typically use default permissions 666 when creating a new file and permissions 777 when creating a new program.**
- **Permissions can be further adjusted by the umask: a three-digit octal number specifying the rights that should be withheld.**
- **Actual default permission is derived by masking the given default permissions with the umask: compute the logical AND of the bits in the default permission and of the inverse of the bits in the umask.**

Sensible umask Settings

- **022:** all permissions for the owner, read and execute permission for group and other.
- **027:** all permissions for the owner, read and execute for group and no permission for other.
- **037:** all permissions for the owner, read permission for group, no permissions for other.
- **077:** all permissions for the owner, no permissions for group and other.

Permissions: Order of Checking

- **Access control uses the effective UID/GID:**
 - If the subject's UID owns the file, the permission bits for owner decide whether access is granted.
 - If the subject's UID does not own the file but its GID does, the permission bits for group decide whether access is granted.
 - If the subject's UID and GID do not own the file, the permission bits for other (also called world) decide whether access is granted.
- **Permission bits can give the owner less access than is given to the other users; the owner can always change the permissions.**

Security Patterns

- **We will discuss how some general security principles manifest themselves in Unix.**
- **Controlled invocation: SUID programs.**
- **Access to the layer below: protecting devices.**
- **Physical and logical representation of objects: deleting files.**
- **Search-path**
- **Importing data from outside: mounting file systems.**

Controlled Invocation

- **Superuser privilege is required to execute certain operating system functions.**
 - Example: only processes running as root can listen at the “trusted ports” 0 – 1023.
- **Solution adopted in Unix: SUID (set userID) programs and SGID (set groupID) programs.**
- **SUID (SGID) programs run with the effective user ID or group ID of their owner or group, giving controlled access to files not normally accessible to other users.**

Displaying SUID programs

- When `ls -l` displays a SUID program, the execute permission of the owner is given as `s` instead of `x`:

```
-rws--x-x 3 root bin 16384 Nov 16 1996  
passwd*
```

- When `ls -l` displays a SGID program, the execute permission of the group is given as `S` instead of `x`:

```
-rwx--S-x 3 root bin 16384 Nov 16 1996  
passwd*
```

SUID to root

- **When root is the owner of a SUID program, a user executing this program will get superuser status during execution.**
- **Important SUID programs:**

<code>/bin/passwd</code>	change password
<code>/bin/login</code>	login program
<code>/bin/at</code>	batch job submission
<code>/bin/su</code>	change UID program

SUID Dangers

- **By tricking a SUID program owned by root to do unintended things, an attacker can act as the root.**
- **All user input (including command line arguments and environment variables) must be processed with extreme care.**
- **Programs should have SUID status only if it is really necessary.**
- **The integrity of SUID programs must be monitored (tripwire).**

Applying Controlled Invocation

- **Sensitive resources, like a web server, can be protected by combining ownership, permission bits, and SUID programs:**
- **Create a new UID that owns the resource and all programs that need access to the resource.**
- **Only the owner gets access permission to the resource.**
- **Define all the programs that access the resource as SUID programs**

Deleting Files

- **General issue: logical vs physical memory**
- **Unix has two ways of copying files.**
- **cp creates an identical but independent copy owned by the user running cp.**
- **ln creates a new filename with a pointer to the original file and increases link counter of the original file; the new file shares its contents with the original.**
- **If the original is deleted (with rm or rmdir) it disappears from its parent directory but the contents of the file and its copy still exist.**
 - Users may think that they have deleted a file whereas it still exists in another directory, and they still own it.
 - If a process has opened a file which then is deleted by its owner, the file remains in existence until that process closes the file

Deleting Files

- **Once a file has been deleted the memory allocated to this file becomes available again.**
- **Until these memory locations are written to again, they still contain the file's contents.**
- **To avoid such memory residues, the file can be wiped by overwriting its contents with a pattern appropriate for the storage medium before deleting it.**
- **But advanced file systems (e.g. defragmenter) may move files around and leave copies.**

Protection of Devices

- **General issue: logical and physical memory**
- **Unix treats devices like files; access to memory or to a printer is controlled like access to a file by setting permission bits.**
- **Devices commonly found in directory /dev:**

<code>/dev/console</code>	console terminal
<code>/dev/kmem</code>	kernel memory map device (image of the virtual memory)
<code>/dev/tty</code>	terminal
<code>/dev/hd0</code>	hard disk

Access to the Layer Below

- **Attackers can bypass the controls set on files and directories if they can get access to the memory devices holding these files.**
- **If the read or write permission bit for other is set on a memory device, an attacker can browse through memory or modify data in memory without being affected by the permissions defined for files.**
- **Almost all devices should therefore be unreadable and unwritable by “other”.**

Example

- **The process status command ps displays information about memory usage and thus requires access permissions for the memory devices.**
- **Defining ps as a SUID to root program allows ps to acquire the necessary permissions but a compromise of ps would leave an attacker with root privileges.**
- **Better solution: let group mem own the memory devices and define ps as a SGID program.**

Mounting Filesystems

- **General issue: scoping of identifiers**
- **The client may misinterpret the UID or GUID even if it tries to enforce access control.**
- **Problem: UID and GID are local identifiers; only globally unique identifiers should be used across network.**
- **NFS server trusts the client to enforce access control on the mounted filesystem.**

Environment Variables

- **Environment variables:** kept by the shell, normally used to configure the behavior of utility programs
- **Inherited by default from a process' parent.**
- **A program executing another program can set the environment variables for the program called to arbitrary values.**
- **Danger:** the invoker of setuid/setgid programs is in control of the environment variables they are given.
- **Usually inherited, so this also applies transitively.**
- **Not all environment variables are documented!**
- **Inheriting things you do not want can become a security problem.**

Examples

PATH	# The search path for shell commands (bash)
TERM	# The terminal type (bash and csh)
DISPLAY	# X11 - the name of your display
LD_LIBRARY_PATH	# Path to search for object and shared libraries
HOSTNAME	# Name of this UNIX host
PRINTER	# Default printer (lpr)
HOME	# The path to your home directory (bash)
PS1	# The default prompt for bash
path	# The search path for shell commands (csh)
term	# The terminal type (csh)
prompt	# The default prompt for csh
home	# The path to your home directory (csh)

Changing Root of the Filesystem

- **Access control can be implemented by constraining suspect processes to a sandbox environment; access to objects outside the sandbox is prevented.**
- **Change root command `chroot` restricts the available part of the filesystem:**
 - `chroot <directory> <command>`
- **Changes the apparent filesystem root directory from `/` to `directory` when command executes.**
- **Only files below the new root are thereafter accessible.**

Changing Root of the Filesystem

- **If you employ this strategy, make sure that user programs find all system files they need.**
- **System files are ‘expected’ to be in directories like /bin, /dev, /etc, /tmp, or /usr**
- **New directories of the same names have to be created under the new root and populated with the files the user will need by copying or linking to the respective files in the original directories.**

Search-path

- **General principle: execution of programs taken from a 'wrong' location.**
- **Users can run a program by typing its name without specifying the full path-name that gives the location of the program within the file system**
- **The shell searches for the program following the search-path specified by the PATH environment variable in the .profile file in the user's home directory.**

Searchpath

- **A typical searchpath:**
 - `PATH=.:\\$HOME/bin:/usr/ucb:/bin:/usr/bin:/usr/local:/usr/new:/usr/hosts`
- **Directories in the searchpath are separated by ‘:’**
- **first entry ‘.’ is the current directory.**
- **When a directory is found that contains a program with the name specified, the search stops and that program will be executed.**

Search-path

- **To insert a Trojan horse, give it the same name as an existing program and put it in a directory that is searched before the directory containing the original program.**
- **As a defense, call programs by their full path-name, e.g. `/bin/su` instead of `su`.**
- **Make sure that the current directory is not in the search-path of programs executed by root**

Network Services (telnet, ftp)

- **inetd daemon listens to incoming network connections**
- **Configuration file maps port numbers to programs**
- **When a connection is made, inetd starts the requested server program and then returns to listening for further connections.**
- **Entries in the configuration file have the format:**
 - service type protocol waitflag userid executable command-line
- **Example: entry for telnet**
 - telnet stream tcp nowait root /usr/bin/in.telnetd in.telnet

Telnet Wrapper

- **When inetd receives a request for a service, it consults the configuration file and creates a new process that runs the executable specified.**
- **Name of new process changed to the name given in the command-line field.**
- **Usually, the name of the executable and the name given in command-line are the same.**

Telnet Wrapper

- **This redundancy can be used for a nice trick:**
- **Point inetd daemon to a wrapper program.**
- **Use the name of the process to remember the name of the original executable; return to this executable after running the wrapper.**
- **Example: change configuration file entry for telnet to**
 - telnet stream tcp nowait root /usr/bin/tcpd in.telnetd
- **Program executed is now the TCP wrapper executable /usr/bin/tcpd.**

Telnet Wrapper

- **Wrapper performs access control, logging, ...**
 - Original application: IP address filtering.
- **Wrapper knows the directory it is in (/usr/bin) and its own name (in.telnetd) so it can call the original server program (/usr/bin/in.telnetd)**
- **Users see no difference and receive the same service as before.**
- **Design principle: add another level of indirection.**
- **TCP wrapper performing security controls is inserted between the inetd daemon and the server program.**

Protecting Root account

- **The root account is used by the operating system for essential tasks like login, recording the audit log, or access to I/O devices.**
- **The root account is required for performing certain system administration tasks.**
- **Superusers are also a major weakness of Unix; an attacker achieving superuser status effectively takes over the entire system.**
- **Separate the duties of the systems manager; create users like uucp or daemon to deal with networking; if a special users is compromised, not all is lost.**

Superuser

- **Systems manager should not use root as their personal account.**
- **Change to root from a user account using `/bin/su`; the O/S will not refer to a version of `su` that has been put in some other directory.**
- **Record all `su` attempts in the audit log with the user who issued the command.**
- **`/etc/passwd` and `/etc/group` have to be write protected; an attacker who can edit `/etc/passwd` can become superuser by changing its UID to 0.**

Trusted Hosts

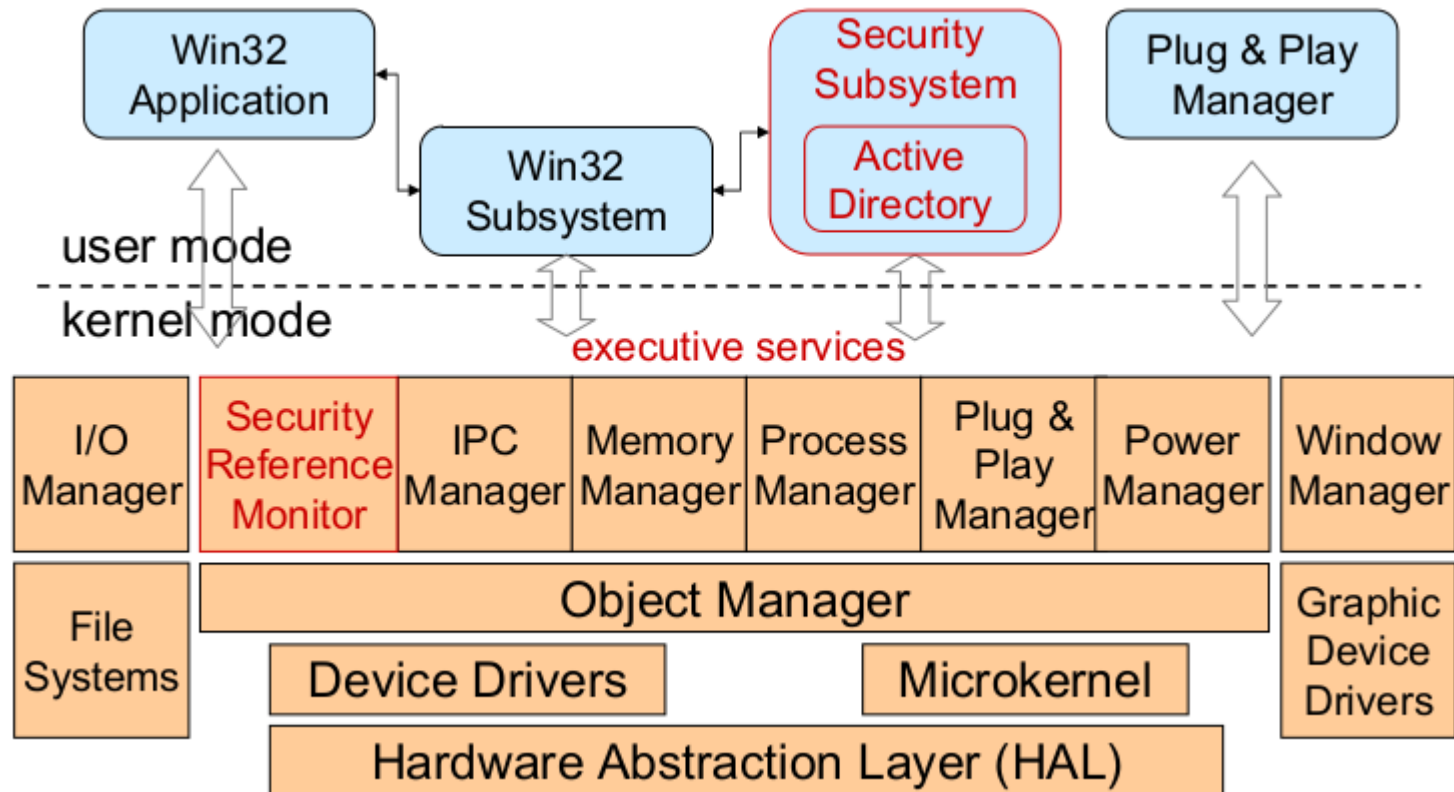
- **Users from a trusted host can login without password authentication; they only need to have the same user name on both hosts.**
- **Trusted hosts of a machine are specified in `/etc/hosts.equiv`.**
- **User names must be synchronized between hosts.**
- **Trusted hosts of a user are specified in the `.rhosts` file in the user's home directory.**
- **User can either access all hosts in the system or nothing; exceptions difficult to configure.**
- **With a growing number of hosts, synchronizing user names and `hosts.equiv` files becomes tedious.**

Audit logs

- **/usr/adm/lastlog** records the last time a user has logged in; displayed with finger
- **/var/adm/utmp** records accounting information used by the who command.
- **/var/adm/wtmp** records every time a user logs in or logs out; displayed with the last command.
- **/var/adm/acct** records all executed commands; displayed with lastcomm

WINDOWS SECURITY MODEL

Windows architecture



Windows Architecture

- **Two modes: user mode & kernel mode**
- **Security components in kernel mode:**
 - Security Reference Monitor
- **Security components in user mode:**
 - Log-on process (WinLogon)
 - Local Security Authority (LSA): deals with user logon and audit logs
 - Security Accounts Manager (SAM): accounts database, including e.g. passwords (encrypted)
- **Device drivers (often third party products) run in kernel mode.**

Registry

- **Registry:** central Windows configuration database.
- **Entries in the registry are called keys.**
- **Registry hive:** group of keys, subkeys, and values in the registry.
 - **HKEY_CLASSES_ROOT:** holds file extension associations; e.g., to specify that .doc files are handled by Word.
 - **HKEY_CURRENT_USER:** configuration information for the user currently logged on.
 - **HKEY_LOCAL_MACHINE:** configuration information about the local computer.
 - **HKEY_USERS:** contains all actively loaded user profiles on the system.
 - **HKEY_CURRENT_CONFIG:** information about hardware profile used by the local computer at system startup.

Windows Domains

- **Stand-alone Windows machines usually administered locally by users; impossible in large organizations.**
- **Domains facilitate single-sign on and centralized security administration.**
- **A server can act as domain controller (DC); other computers join the domain.**
 - A domain can have more than one DC; updates may be performed at any DC.
- **Domain admins create and manage domain users and groups on the DC.**
- **Domains can form a hierarchy.**

Access control

- **Access control in Windows applies to objects: files, registry keys (systems database), Active Directory objects, etc.**
- **More complex than access control in a file system.**
 - Access rights beyond read, write, execute.
- **Means for structuring policies in complex systems: groups, roles, inheritance.**
- **Identify principals, subjects, and objects.**
- **Access rules: where to find them, how they are evaluated.**

Principals

- **Principals: local users, domain users, groups, aliases, machines.**
 - Machine readable security identifier.
 - Human readable user name.
- **Domain users, groups, aliases, machines:**
 - `principal@domain = DOMAIN\principal`
 - E.g. `diego@europe.microsoft.com = EUROPE\diego`
- **Local users and aliases:**
 - `principal = MACHINE\principal`
 - `diego@europe.microsoft.com = MSRC-688432\Administrators`

Scoping of Principals

- **Local Security Authority (LSA):** each Windows machine has its own built-in authority; users created by the LSA are local users.
- **Local principals, administered locally, visible only to the local computer:**
 - e.g. local system (i.e. O/S), local users
- **Domain principals, administered by domain admins on a domain controller, seen by all computers in domain:**
 - e.g. domain users, Domain Admins alias
- **Universal principals: e.g. Everyone alias**

Security Identifiers

- **Security identifier (SID) format: S-R-I-SA-SA-SA-N**
 - S: letter S
 - R: revision number (currently 1)
 - I: identifier authority (48-bit)
 - SA: subauthority (32-bit)
 - N: relative identifier, unique in the authority's name space
- **E.g. Guest S-1-5-21-<authority>-501**
 - <authority>: 96-bit unique machine or domain identifier created when Windows or domain controller is installed
- **E.g. World (Everyone) S-1-1-0**

SID - Examples

- **SYSTEM S-1-5-18**

- OS runs locally as S-1-5-18; in its domain machine is known under a separate, domain specific, SID.

- **Administrator S-1-5-21-<local authority>-500**

- user account created during OS installation.

- **Administrators S-1-5-32-544**

- built-in group with administrator privileges, contains initially only the Administrator account.

- **Domain Admins S-1-5-21-<domain authority>-512**

- global group, member of the Administrators alias on all machines in a domain.

Creating an Authority

- **A new issuing authority gets a SID with identifier authority 5, followed by 21 and a 96-bit random number put into three subauthority fields.**
- **Design principle: authorities have (statistically) unique identifiers.**
- **SIDs include the identifier of the issuing authority (domain), so a SID cannot by mistake represent access rights in the scope of some other domain.**
- **Design principle: use randomness for creating unique name spaces.**

Creating a SID

- **SID constructed when a user account is created, fixed for the lifetime of the account.**
- **Pseudo-random input (clock value) used to construct a SID; you will not get the same SID if you delete an account and then recreate it with exactly the same parameters as before.**
- **SIDs for users and groups are unique and cannot be assigned again to another user or group.**
- **A principal cannot by mistake get permissions of a previous principal.**

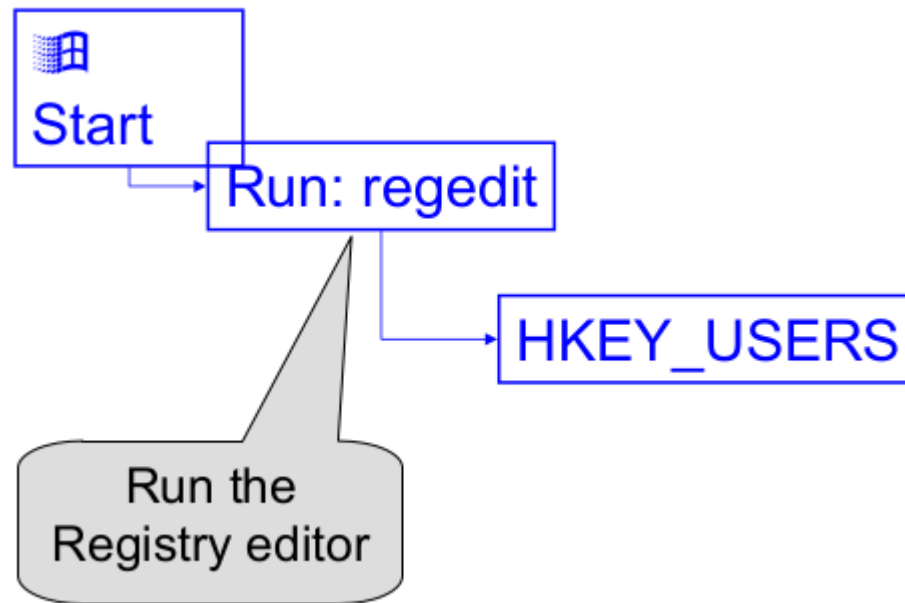
Where do Principals Live?

- **Information about principals stored in accounts and user profiles.**
- **User profiles stored in file system under \Documents and Settings\.**
- **Local accounts in Registry (under HKEY_USERS).**
- **Domain accounts at the Domain Controller, also cached locally.**
- **Domain controller authority knows the principal's password; can act as a trusted third party when principal authenticates itself to some other entity.**

Principals for Access Control

- **SID**: an individual principal
- **Group**: a collection of SIDs managed by the domain controller; a group has its own group SID, so groups can be nested.
- **Alias** (local group): collection of user and group SIDs managed by DC or locally by LSA; cannot be nested.
 - Used to implement logical roles: application developer refers to an alias Student, at deployment time appropriate SIDs are assigned to this alias.
- **Design principle: support instantiation of policies that refer to placeholder principals.**

Display SIDs on a Machine



Subjects & Tokens

- **In Windows, processes and threads are subjects.**
- **Security credentials for a process (or thread) stored in a token.**
- **Token contains a list of principals and other security attributes.**
- **New process gets a copy of the parent's token; can restrict it.**

Token Contents

- **Identity and authorization attributes:**
 - user SID, group SIDs, alias SIDs
 - privileges
- **Defaults for new objects:**
 - owner SID, group SID, DACL
- **Miscellaneous:**
 - logon session ID
- **Some fields are read-only, others may be modified.**

Privileges

- **Privileges control access to system resources.**
- **Uniquely identified by programmatic name (SeTcbPrivilege), have display name (“Act as part of the operating system”), cached in tokens as a locally unique identifier (LUID).**
- **Assigned to users, groups and aliases.**
- **Assigned on a per machine basis.**
- **Different from access rights, which control access to ‘securable objects’ (explained later).**

Performance and Reliability

- **Group and alias SIDs are cached in the token, as is the union of all privileges assigned to these SIDs.**
- **Token will not change even if a membership or privilege is revoked.**
- **Better performance.**
- **Better reliability as process can decide in advance whether it has sufficient access rights for a task.**

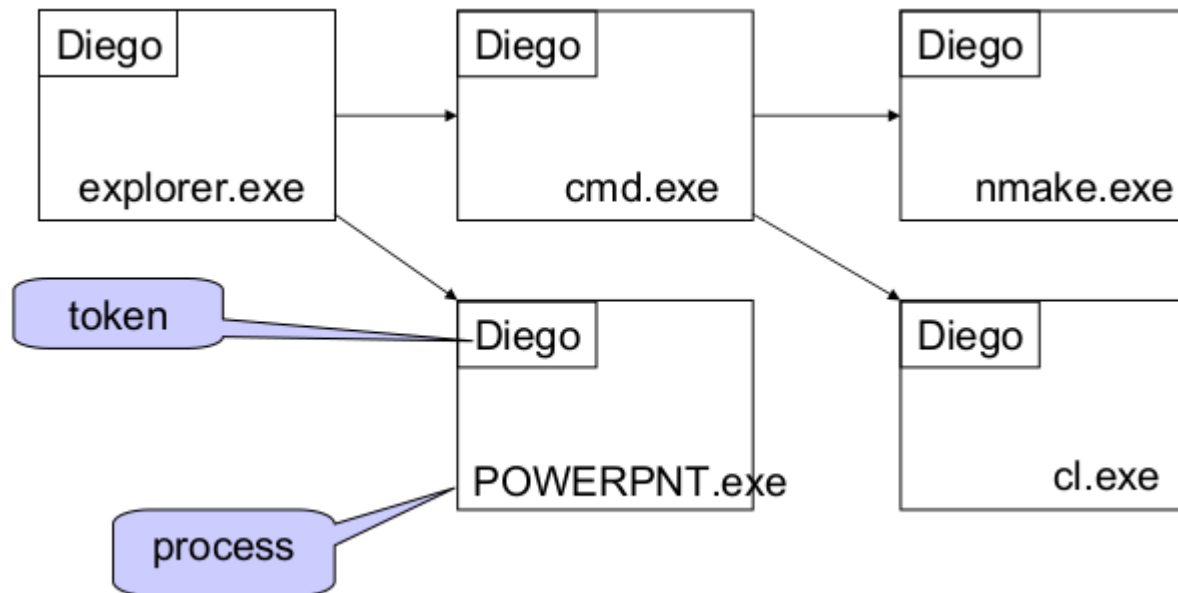
Creating Subjects

- **A machine is always running a logon process**
 - (winlogon.exe) under the principal SYSTEM.
- **When a user logs on to a machine,**
 - logon process collects credentials and presents them to the LSA,
 - LSA (lsass.exe) verifies the credentials,
 - logon process starts a shell (explorer.exe) in a new logon session under the user (principal).
- **Shell spawns processes to the same logon session.**
- **Log-off destroys logon session and all processes in it.**

Creating more Subjects

- **A process can spawn a new local process (subject) by calling `CreateProcess`.**
- **Each process has its own token: different processes within a logon session can have different credentials**
- **New process gets a copy of parent's token.**
- **Threads can be given different tokens.**
- **User's network credentials (e.g. password) are cached in the interactive logon session.**
- **Processes can create network logon sessions for that user at other machines; network logon sessions do not normally cache credentials.**

Processes in a Logon Session



Objects

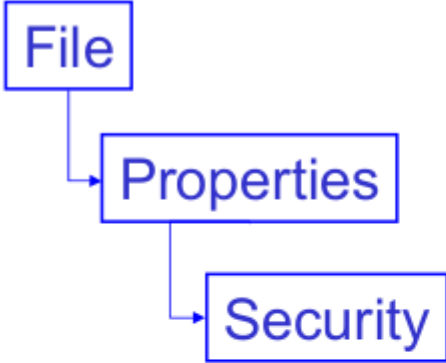
- **Securable objects have a security descriptor.**
- **Security descriptors for private objects have to be managed by the application software.**
- **Security descriptors for built-in objects are managed by the OS.**
- **Creating securable objects it is tedious but enables highly granular access control.**

Security Descriptor (SD)

Owner SID
Primary Group SID
DACL
SACL

- Owner: defined when object is created.
- Primary Group: for POSIX compatibility
- DACL: lists who is granted or denied access
- SACL: defines audit policy

Find Access Rights for a File





File

Properties

Security

Group or user name

 SYSTEM

 Dieter

Add Delete

Permissions for ...	Allow	Deny
Full Control	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Modify	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Read & Execute	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Read	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Write	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Special Permission	<input type="checkbox"/>	<input type="checkbox"/>

Advanced

Owner

- **Objects get an owner when they are created.**
- **Stored in the security descriptor the object.**
- **Ownership can also be obtained via the privilege 'Take ownership of files and other objects' (SeTakeOwnershipPrivilege).**
- **Owner (almost) always has READ_CONTROL and WRITE_DAC permission.**

Access Control Lists

- **DACL in security descriptor: List of access control entries (ACEs).**
- **ACE format:**
 - Access mask (access rights)
 - Flags
 - Type: positive (Access Allowed), negative (Access Denied), audit; whether ACE contains ObjectType
 - InheritedObjectType (since Windows 2000)
 - ObjectType (since Windows 2000)
 - Trustee: Principal SID the ACE applies to

Permissions (access rights)

- **Describe what one can do with an object.**
- **Permissions encoded as 32-bit masks.**
- **Standard permissions**
 - DELETE
 - READ_CONTROL: read access (to security descriptor) for owner, group, DACL
 - WRITE_DAC: write access to DACL
 - WRITE_OWNER: write access to owner
 - SYNCHRONIZE
- **Specific permissions can be tailored to each class of objects.**

Active Directory

- **Directory service in Windows 2000.**
- **Hierarchy of typed objects.**
- **Each object type has specific properties.**
- **Each object type has a unique GUID (globally unique identifier).**
- **Each property has its own GUID.**
- **Containers: Objects that may contain other objects.**
- **AD can be dynamically extended by adding new object types or new properties to existing object types.**

ObjectType

- **ObjectType: GUID defining an object type.**
- **ACEs without ObjectType are applied to all objects.**
- **Application can include ObjectType in its access requests; only ACEs with matching ObjectType or without an ObjectType will be evaluated.**
- **Control read/write access on object property: put GUID of the property in ObjectType.**
- **Control create/delete access on objects: put GUID of the object type in ObjectType.**

Example

- **Allows Server Applications to create RPC endpoints in any container of type RPC Services.**
- **ACE will be inherited into any container of type RPC Services.**

ACE1	
Access mask:	create child
Type:	ACCESS_ALLOWED_OBJECT_ACE
InheritedObjectType:	{GUID for RPC Services}
ObjectType	{GUID for RPC Endpoint}
Trustee (principal SID):	Server Applications