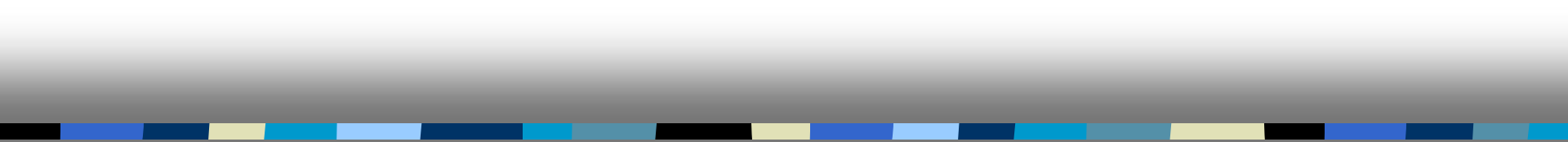


Computer Security

CS 426

Lecture 9



Access Control for Intrusion Prevention

Review

- Malware
 - trojan horses, viruses, worms, botnets, rootkits
- Two questions from last lecture
 - Buffer overflow vulnerabilities exploited by Morris worm is in gets, rather than fgets
 - In the Warhol worm: **the name Warhol is based on Andy Warhol's remark that "In the future, everyone will have 15 minutes of fame".**

Review: The Morris Worm

- Three ways to get a shell on a system
 - exploits buffer overflow in fingerd (gets)
 - exploits DEBUG option in sendmail
 - comes from a trusted host (.rhosts) through rsh
 - having cracked passwords on a trusted host
- After getting a shell
 - compiles a short loader program and runs it,
 - the loader program retrieves the worm main program, compiles and runs it
 - the main program tries to propagate to other hosts

Review: Nimda worm

- Spreads via 5 methods to Windows PCs and servers
 - e-mails itself as an attachment (every 10 days)
 - scans for and infects vulnerable MS IIS servers
 - copies itself to shared disk drives on networked PCs
 - PCs that later read the file is contaminated
 - appends JavaScript code to Web pages
 - surfers pick up worm when they view the page.
 - scans for the back doors left behind by the "Code Red II" and "sadmind/IIS" worms

How does a computer get infected with malware or being intruded?

- Executes malicious code (email attachment, download and execute trojan horses, use infected floppy/thumb drive)
- Runs buggy daemon programs that receive traffic from the network (e.g., ftpd, httpd)
- Runs buggy client programs (e.g., web browser, mail client) that receive input data from network
- Read malicious files with buggy file reader program
- Configuration errors (e.g., weak passwords, guest accounts, DEBUG options, incorrect access control settings, etc)

Review & Motivation

- Softwares have vulnerabilities
 - most prominent is memory errors (e.g., buffer overflow, ...)
- Vulnerabilities can be exploited
- Many layers of defense mechanisms exist
 - no mechanism is silver bullet
- Security implications often caused by issues in access control.

Can we fix access control?

Readings for this lecture

- Additional readings
 - Best Practices for UNIX chroot() Operations,
 - The Linux kernel capabilities FAQ

 - wikipedia topics: Introduction to virtualization, Operating system-level virtualization, FreeBSD jail,

What are some issues with UNIX access control?

- Designed for local users sharing the same machine, not with network attack and buggy software in mind
- Coarse granularity
 - access control is by user id; however, the user may be executing many different programs (some cannot be trusted), they run with the same privilege, but they shouldn't
- All powerful root

Solutions

- Virtualization to achieve confinement
 - compromising one process doesn't affect others
 - three kinds of virtualization technologies exist
- Breaking up the power of root
- Finer-grained process access control based on Mandatory Access Control
 - aiming at better achieving least privilege: a program does only what it is supposed to do, but not others,

Virtualization Technologies (1)

- Operating system level virtualization
 - runs a single kernel, virtualizes servers on one operating system,
 - e.g., chroot, FreeBSD jail, ...
 - used by service providers who want to provide low-cost hosting services to customers.

 - Pros: best performance, easy to set up/administer
 - Cons: all servers are same OS, some confinement can be broken

chroot

- The chroot system call **changes** the **root** directory of the current and all child processes to the given path, and this is nearly always some restricted subdirectory below the real root of the filesystem.
- chroot exists in almost all versions of UNIX,
 - creates a temporary root directory for a running process,
 - takes a limited hierarchy of a filesystem (say, /chroot/named) and making this the top of the directory tree as seen by the application.
- A network daemon program can call chroot itself, or a script can call chroot and then start the daemon

Using chroot

- What are the security benefits?
 - under the new root, many system utilities and resources do not exist, even if the attacker compromises the process, damage can be limited
 - consider the Morris worm, how would using chroot for fingerd affect its propagation?
- Examples of using chroot
 - ftp for anonymous user
- How to set up chroot?
 - need to set up the necessary library files, system utilities, etc., in the new environment

Limitations of chroot

- Only the root user can perform a chroot.
 - intended to prevent users from putting a setuid program inside a specially-crafted chroot jail (for example, with a fake /etc/passwd file) that would fool it into giving out privileges.
- chroot is not entirely secure on all systems.
 - With root privilege inside chroot environment, it is sometimes possible to break out
- process inside chroot environment can still see/affect all other processes and networking spaces
- chroot does not restrict the use of resources like I/O, bandwidth, disk space or CPU time.

Jail is an extension of chroot implemented in FreeBSD

- Jail provides decentralized administration (partitioning), similar to a virtual machine
- Restrictions in Jail
 - access to the file name-space is restricted in the style of chroot
 - the ability to bind network resources is limited to a specific IP address
 - the ability to manipulate system resources and perform privileged operations is sharply curtailed
 - the ability to interact with other processes is limited to only processes inside the same jail
 - e.g., ps shows only processes in same jail

Virtualization Techniques (2)

- Virtual machines: emulate hardware in a user-space process
 - the emulation software runs on a host OS; guest OSes run in the emulation software
 - needs to do binary analysis/change on the fly
 - e.g., VMWare, Microsoft Virtual PC

 - Pros: can run other guest OS without modification to the OS
 - Cons: worst performance

Virtualization Techniques (3)

- Paravirtualization
 - No host OS, a small Virtual Machine Monitor runs on hardware, guest OSes need to be modified to run
 - Requires operating systems to be ported to run
 - e.g., Xen

 - Pros: better performance compared with (2), supports more OSes compared with (1)
 - Cons: each guest OS must be modified to run on it, (each new version of the OS needs to be patched)

Roadmap

- Virtualization to achieve confinement
 - Does it solve all operating system access control problems?
 - When is it useful?
- Breaking up/limiting the power of the all-powerful root
 - POSIX/Linux Capability system
 - FreeBSD runlevel
- Finer-grained process access control based on Mandatory Access Control

Breaking up the all-powerful root

- The Linux kernel breaks up the power of root into multiple capabilities
 - 31 different capabilities defined in `capability.h` in Linux kernel 2.6.11

```
141 /* Allows binding to TCP/UDP sockets below 1024 */
142 /* Allows binding to ATM VCIs below 32 */
143
144 #define CAP_NET_BIND_SERVICE 10
145
146 /* Allow broadcasting, listen to multicast */
147
148 #define CAP_NET_BROADCAST 11
```

Some Capabilities and Their Meanings

Capability Name	Meaning
CAP_CHOWN	Allow for the changing of file ownership
CAP_CHOWN	Override all DAC access restrictions
CAP_DAC_READ_SEARCH	Override all DAC restrictions regarding read and search
CAP_KILL	Allow the sending of signals to processes belonging to others
CAP_SETGID	Allow changing of the GID
CAP_SETUID	Allow changing of the UID
CAP_SETPCAP	Allow the transferring and removal of current set to any PID
CAP_LINUX_IMMUTABLE	Allow the modification of immutable and append-only files
CAP_SYS_MODULE	Allow the loading of kernel modules
CAP_NET_BIND_SERVICE	Allow binding to ports below 1024

How the capability system work?

- Each process has three sets of bitmaps
 - effective capabilities the process is using
 - permitted capabilities the process can use
 - inheritable capabilities that can be inherited when loading a new program

How the capability system work?

- Each executable file has three sets of bitmaps
 - allowed can inherit these capabilities
 - forced get these capabilities
 - transfer these capabilities are transferred from permitted to effective, i.e., capability-aware or not
- When the file is loaded by exec:
 - new permitted = forced | (allowed & inheritable)
 - new effective = new permitted & transfer
 - new inheritable = inheritable

Why would capabilities be useful?

- A program that needs some but not all privileges of root does not need to be setuid root,
 - it just needs the corresponding capability bits in forced
- Remove some capabilities during system boot will make the system very difficult to penetrate.
 - Protect integrity of system utilities and log files
 - make system log files append-only and core system utilities immutable and remove `CAP_LINUX_IMMUTABLE`
 - this makes it virtually impossible for intruders to erase their tracks or install compromised utilities.

FreeBSD securelevel

- A security mechanism implemented in the kernel.
 - when the securelevel is positive, the kernel restricts certain tasks; not even the superuser (i.e., root) is allowed to do them.
 - one cannot lower the securelevel of a running system.
 - Why?
 - to lower the securelevel, need to change the securelevel setting in `/etc/rc.conf` and reboot.
 - When need to change the secure level?

Warning about securelevel, from FreeBSD FAQ

Securelevel is not a silver bullet; it has many known deficiencies. More often than not, it provides a false sense of security.

One of its biggest problems is that in order for it to be at all effective, all files used in the boot process up until the securelevel is set must be protected. If an attacker can get the system to execute their code prior to the securelevel being set (which happens quite late in the boot process since some things the system must do at start-up cannot be done at an elevated securelevel), its protections are invalidated. While this task of protecting all files used in the boot process is not technically impossible, if it is achieved, system maintenance will become a nightmare since one would have to take the system down, at least to single-user mode, to modify a configuration file.

Roadmap

- Virtualization to achieve confinement
- Breaking up/limiting the power of the all-powerful root
 - POSIX/Linux Capability system
 - FreeBSD runlevel
- Finer-grained process access control based on Mandatory Access Control

Main Idea of Finer-grained Access Control

- For each process, there is an additional policy limiting what it can do
 - in addition to the DAC restriction based on the user ids
 - typically specify the capabilities and the files that can be accessed
- The policy can be based on
 - the binary that was loaded
 - the source of data that the process has received
- The key challenge
 - how to specify the policy

Example systems of finer-grained access control

- Systrace
- The Linux Intrusion Detection System (LIDS)
- Security Enhanced Linux (SELinux)
 - initially developed by people in NSA
 - shipped with Fedora Linux distributions
- AppArmor
 - shipped with SUSE Linux distributions
- LOMAC (Low Water-Mark Mandatory Access Control)
- The Usable Mandatory Integrity Protection Model (UMIP)
 - developed by Purdue

We will discuss these after presenting the theory of
DAC and MAC.

Coming Attractions ...

- September 18:
 - Theory of Access Control: Access Matrices, Discretionary Access Control, Access Control List, Capabilities
- Readings:
 - Security Engineering: Chapter 7

