Protection & Security

Tyler Bletsch
Duke University

Slides are adapted from Brian Rogers (Duke)
Protection

- OS manages resources for users & user processes
  - Files, memory regions, I/O channels, CPU
- Protection is a critical part of this management
  - Ensure that resources can only be used with proper authorization from the OS
- Reasons for protection
  - Prevent users from malicious access to resources
  - Ensure processes use system resources only as consistent with allowed policies
- Protection is a *mechanism*
  - Mechanism to enforce policies that define how resources should be used
  - As opposed to a *policy* (definition of how resources should be used)
  - Policies may adapt and change over time (or between different applications)
  - Thus mechanisms should be general to allow flexibility
Basics of Protection

• Most protection mechanisms based on key principle
  – Principle of least privilege
  – Users, processes, etc. have the minimum level of access to resources and privileges needed to accomplish intended task
  – “Need to know basis”

• Minimizes damage from failed or compromised pieces
  – They can only affect a minimal set of components in the system

• OS designs provide support for this
  – System calls and services for apps to specify fine-grained permissions & controls
  – Apps enable and disable permissions as needed
  – Also applies to users (separate accounts, permissions)
Protection Domains

• Think of all resources as an object
  – E.g. in UNIX – “everything is a file”
  – Hardware objects: CPU, memory spaces, disk, keyboard, display
  – Software objects: files, directories, programs

• Different objects have different possible operations, e.g.
  – Read & write memory regions; Read from a keyboard input
  – Execute on a CPU
  – Create, delete, open, close, read, write, append files

• Protection mechanism operates based on domains
  – Application or user has permission to perform certain operations on certain objects
  – Again, can change dynamically
Protection Domains (2)

• Processes operate within a protection domain
  – Specifies objects (resources) the process has permission to access

• Access Right
  – A permission for a process to perform an operation on an object
  – Domain just a collection access rights
  – Access right can exist in more than one domain at a time
  – Example…

• Access rights can be static or dynamic for a process
  – Dynamic domains achieved via either:
    • A mechanism to change domain access rights
    • A mechanism for domain switching
      – Create new domain with proper new access rights; then switch to it
  – For example, user – supervisor mode we discussed for interrupts
UNIX Example of Domains

• In UNIX, each user belongs to a domain

• When the domain is changed, the user ID is changed

• Happens via a protection mechanism using the file system
  – Remember, in UNIX, “everything” is a file
  – Every file has an owner ID and a domain bit (setuid bit)
    • "set user ID upon execution"
    • Set just like file read/write/execute permissions
  – When a user executes a file:
    • If setuid bit is on, user ID is changed to the owner of the file
    • If setuid bit is off, user ID does not change
  – Temporary user ID change ends after process exits

• Allows a privileged component to be used by general users
  – E.g. an application that accesses the network or change user password

• What if a user creates a file with user ID of root & setuid on?
Access Matrix

- Domain protection model maps nicely to a matrix
- Rows = domains; columns = objects
- A matrix entry lists the access rights
- Provides a general mechanism for specifying policies
  - Enforce specific access rights for a user or process in a domain
- Example
Access Matrix – Additional Functions

• Base access matrix allows
  – Defining and enforcing strict access control policy

• How can we provide dynamic rights?
  – Domain switching
    • Add access matrix entries to enforce “switch” operation rights
    • Example
  – Allow controlled changing of the access matrix entries
    • Encode this permission in the access matrix as well!
    • New operations: copy, owner, control
Copy, Owner, Control

• Copy
  – Copy an access right from one domain to another for an object
  – Variants:
    • Move right instead of copy (transfer)
    • Copy with and without propagation of the copy right (limited copy)

• Owner
  – Allows creation of new rights in the access matrix
  – Process in a domain can add & remove any right for that object

• Control
  – Applies only to domain objects
  – Allows removal of rights in the access matrix
  – Process in a domain can remove any right from that row
Access Matrix Implementation

• In a real system, the matrix will be very sparse
  – But the way it is accessed & used cause special considerations

• Global Table
  – List of <domain, object, rights> tuples
  – Search for “right” when a process in “domain” accessed “object”
  – Drawbacks:
    • Table is huge
    • Objects may have “global” rights, but still listed in every domain

• Access Lists
  – Maintain a list per object with <domain, rights> (column-based)
  – Can extend with a “default” set of rights for each object
Security

• Protection is a mechanism for internal problem
  – Controlled access to programs, data

• Security deals with the external environment
  – Protection can be thwarted if security is compromised
  – Protection works well only if users behave as intended
  – E.g. if a user password is stolen or cracked

• In a secure system…
  – All resources (objects) are used only according to policy
  – Cannot be achieved in reality, but strive to limit violations
Security and the OS

- Why is security important for systems programming?
- Many attacks target an “escalation of privilege”
  - This often involves attempting to gain “root” privilege in a system
  - For purposes of reading or tampering with data
    - Data not otherwise accessible via protection mechanism
  - Systems programmers should be aware of:
    - Types of attacks
    - Mechanisms by which attacks are attempted and operate
Some Definitions

• Threat: potential for a security violation
  – E.g. a **vulnerability** in a program

• Attack: attempt to break security
  – E.g. an **exploit** is an attempt to utilize a program vulnerability

• Categories of security violations
  – Confidentiality breach: data theft
    • E.g. credit card, account information; very common goal
  – Integrity breach: unauthorized modification of code or data
  – Availability breach: unauthorized destruction of data
    • E.g. deleting customer account info or defacing a website
  – Theft of service: unauthorized use of resources
  – Denial of service: prevent legitimate use of a system
Domains of Security Measures

- Physical
  - Need to secure the sites where computer systems reside
  - Only authorized administrators / users have physical access

- Human
  - Social engineering may trick authorized users into performing an inadvertent breach of security
  - Phishing obtain information
  - Executing malicious code

- OS
  - Mechanisms to protect from accidental or purposeful breaches

- Network
  - Protect network-transmitted data from interception or tampering
Types of Threats

• Threats to Running Programs
  – Trojan Horse
  – Trap Door
  – Logic Bomb
  – Stack (or Buffer) Overflow
  – Viruses

• Threats to System and Network Resources
  – Worms
  – Port Scanning
  – Denial of Service
Trojan Horse

• Malware code that misuses its environment
  – Often disguised as legitimate software
  – User unknowingly is tricked into executing the malware code
  – Often takes advantage of access rights of the executing user

• Example: Take advantage of search paths on UNIX OS
  – PATH environment variable specifies order of locations to search for executable files (e.g. ‘ls’ command)
  – PATH usually has things like: /bin:/usr/bin:/usr/local/bin
  – Sometimes also has things like “.” (current directory)
  – Malicious user creates a Trojan program
    • With a common command name (e.g. ‘cd’)
  – Unknowing user goes into the directory with this program and executes what they think is the “normal” cd command
  – execute Trojan code
Trojan Horse (2)

• Emulate a login prompt
  – User enters a login ID and password
  – Trojan code captures user ID and password
  – Trojan code prints a login failure message & exits
    • Returning to the real login prompt
  – User thinks they have mistyped password; suspects nothing
  – Reason behind the `<ctrl>+<alt>+<delete>` Windows
    • Non-trappable key sequence
    • Trojan code cannot intercept this signal and ignore it

• Spyware
  – Code contained along with software to display ads
  – Or capture information and send it somewhere for mining
    • This is called a covert channel (e.g. by opening a network daemon)
  – Fundamental violation of principle of least privilege
Trap Door

• A security hole purposely left in legitimate software
  – Can be exploited by those with knowledge of the vulnerability

• Financial code might include tiny rounding errors
  – Route rounded money to a specific bank account
Trap Door (2)

- Tweak authentication procedures for an application
  - E.g. obscure user name and ID password combo is always valid
- Extra sneaky scenario:
  - Embed the trap door for an application in compiler(s)
  - Compiler checks to see if it is compiling the specific application
  - If so, it inserts the trap door code
  - Inspection of the application source code reveals no issues!
  - Additionally, if the application is re-compiled, problem still exists!
Logic Bomb

• Malware that is triggered only under certain conditions
  – E.g. causes a security incident only at a particular time
  – Application may run normally for a long period of time

• What kind of conditions? Almost anything…
  – Certain year, month, day, time
  – If certain information is present on the system
  – Check a database to see if the programmer is still employed
Stack (Buffer) Overflow

• Previous threats involve:
  – A programmer that can create malicious programs
  – A way to install or place malicious code in the system

• What if this is not possible?
  – How could an attacker execute malicious code?

• By far the most common way is through stack overflow
  – Takes advantage of the stack mechanism to:
    • Allow injection of malicious code
    • Force a process to execute the code

• As we will see, specific to architecture, OS, & application
Stack Overflow Mechanism

- Stack organized as frames
  - Every function call creates & pushes a new frame on stack
  - Function return pops frame from stack

- Frames may contain
  - Return address (PC)
  - Address of previous frame
  - Space for function args
  - Space for function local vars
void func(int a, int b, char *c) {
    char buff1[5];
    char buff2[10];
    strcpy(buff2, c);
}

int main(int argc, char *argv[]) {
    func(1, 2, argv[1]);
}

Stack Overflow Mechanism

Stack After Calling func()

New Stack Frame

main:
    <snip>
    0x4 push $1
    0x8 push $2
    0xC push $3
    0x10 call func  #pushes pc=0x14
    <snip>
Stack Overflow Mechanism (2)

```c
void func(int a, int b, char *c) {
    char buff1[5];
    char buff2[10];
    strcpy(buff2, c);
}

int main(int argc, char *argv[]) {
    func(1, 2, argv[1]);
}
```

- What if buffer c has more than 10 bytes?
- Lack of bounds checking means other stack addresses will be overwritten
- Including the Return PC!!!
- When a function exits (‘ret’ instruction)
  - This “Return PC” is placed in the CPU program counter
If a stack buffer is filled based on user input:
- A lack of bounds checking means the return PC can be changed

What can the PC be changed to?
- Where have we learned about PCs that point to specific code?
- Or the attacker can fill the buffer with code!
- Return PC will point back to the written buffer

For example, code to execute a shell:

```c
int main(int argc, char *argv[]) {
    execvp("/bin/sh", "/bin /sh", NULL);
    return 0;
}
```

If this is interesting, read “Smashing the Stack for Fun and Profit”: http://insecure.org/stf/smashstack.html
Prevention Mechanisms

• Historically this type of attack has dominated incidents
  – Lack of bounds checking is the vulnerability
  – Stack overflow is the exploit

• What can be done? Lots of R&D on this:
  – strncpy()!! (i.e. more careful programming)
  – Non-executable stacks
    • But attackers just become more sophisticated
    • Use chains of system calls to perform attacks
  – Address space randomization
  – Stack meta-data to record modifications to stack information
  – And on and on
Virus

- Malware embedded in a legitimate program
- Replicates itself and actively spreads
  - Key distinction from the threats we have thus far seen
- Typically spread via social engineering
  - Users execute programs via spam email or internet downloads
- One common source
  - Microsoft office files, as they can execute macros
  - Attacker can embed malicious macro code in a file
- A delivery program (usually a Trojan horse) contains a program called a virus dropper
  - Virus dropper executes and injects virus into the system
  - E.g. copies to memory & starts executing virus program
Some Common Virus Categories

• **File**
  – Virus appends to a file
  – Changes start of program to jump to virus
  – After executing, returns control to program so virus is not noticed

• **Boot**
  – Infect boot sector; execute every time system is booted
  – Also infects other bootable media such as USB
  – Viruses don’t appear in the file system

• **Macro** (mentioned on last page)

• **Source code**
  – Modifies source of programs to include the virus & help spread
More Common Virus Categories

- Lots of variants geared to avoiding detection
- Polymorphic
  - Changes each time it is installed to change its “signature”
  - Helps defeat virus scanners that look for patterns in code
- Stealth
  - Virus modifies parts of a system that check for virus existence
  - E.g. modify the read() system call
- Tunneling
  - Virus bypasses detection by installing itself in interrupt handler or device drivers
- Encrypted
  - Virus includes decryption code to decrypt itself and then execute
Worms

- Essentially a virus that uses network resources to spread
- Does not attach to existing program like a virus
- Often consist of 2 pieces
  - “Grappling hook”
    - Initial code that is established and executes on a machine
    - It communicates with an established machine & requests the worm
  - Main worm malware
Morris Worm

- Robert Morris – 1998, graduate student at Cornell
- First internet worm
- Worm attempted 3 network attack methods
  - Try to exploit ‘rsh’ to execute a task remotely
    - Searched for host-login name pairs in special files
  - Exploit vulnerability in ‘finger’ program
    - Buffer overflow of a stack frame to execute /bin/sh
  - Exploit vulnerability in ‘sendmail’ program
    - Debugging option commonly left enabled by system admins
    - Worm mailed itself & executed grappling hook program
Port Scanning

• Attempt TCP/IP connections to host on range of ports
• Each attempt tries to connect to a vulnerable service
  – E.g. sendmail
• If successful, this indicates an exploit opportunity
  – E.g. to exploit a buffer overflow
Denial of Service

- Not gained at stealing or modifying information
- Goal is to disrupt legitimate use of a system or service
- Usually network-based attacks
  - E.g. initiate but do not complete many TCP/IP connections
  - Results in no new legitimate connections being serviced
- Distributed Denial of Service (DDOS)
  - Launched from multiple sites at a common target
  - Sometimes the multiple sites are infected machines (zombies)
- Sometimes security measures introduce DOS vulnerabilities
  - E.g. increasing delays between unsuccessful login attempts
Rootkits

- Malware that acquires privileged access to the OS
  - Also maintains that access
  - By hiding its presence from normal OS activity

- Goals of a rootkit
  - Run (without restriction) on a target system
    - Use social engineering or vulnerabilities in protection (e.g. ACLs)
  - Remain invisible to security software, OS, users
  - Perform malicious action (called the payload)
    - Steal information or access to resources; install other malware
How do Rootkits Hide?

• Processes (including security software) depend on the OS to provide information about the environment
  – E.g. through APIs that expose system calls

• Rootkit software can monitor these API questions to OS
  – Rootkit intercepts questions related to its existence
    • E.g. an ‘ls’ of the directory where the rootkit program exists
    • E.g. an ‘ls’ of the /proc/ directory containing info on all processes
    • E.g. a ‘read’ request on a file modified by the Rootkit
Rootkit Subversion Mechanisms

• “Hooking” OS APIs
  – Change address of OS APIs by pointing to own malware code
  – Can be done for user or supervisor mode
  – In response to system calls, the code at modified address is run

• Hide in unused disk space
  – Unused disk space is not visible to normal file system APIs
  – Modify device driver(s) to execute rootkit code when loaded

• Infect the Master Boot Record (MBR)
  – Control what is loaded into memory before OS is even booted