Security Attack Examples and Protection

Bojan Nokovic

Based on:
"Operating Systems Concepts", 10th Edition Silberschatz Et al.
"Operating Systems: Internals and Design Principles", 8th Edition by W Stallings

Apr. 2021

Buffer Overflow Attacks

Also known as a **buffer overrun**

Defined in the NIST (National Institute of Standards and Technology) as:
"A condition at an interface under which more input can be placed into a buffer or data-holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system"

One of the most prevalent and dangerous types of security attacks!

Basic Buffer Overflow C Code

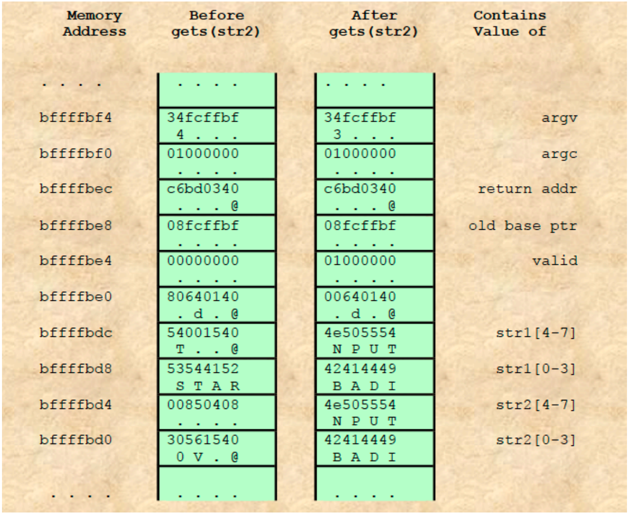
```c
int main(int argc, char *argv[])
{
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

Buffer Overflow Example Runs

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

Buffer Overflow Stack



Memory Address	Before gets(str2)	After gets(str2)	Contains Value of
. . . .	. . . .	. . . .	
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

Exploiting Buffer Overflow

To exploit any type of buffer overflow the attacker needs:

- To identify a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attackers control
- To understand how that buffer will be stored in the processes memory, and hence the potential for corrupting adjacent memory locations and potentially altering the flow of execution of the program

Defenses

Countermeasures can be broadly classified into two categories:

Compile-time defenses, which aim to harden programs to resist attacks

Runtime defenses, which aim to detect and abort attacks in executing programs

Compile-time Defenses

Aim to prevent or detect buffer overflows by instrumenting programs when they are compiled

1. Choose a high-level language that does not permit buffer overflows
2. Encourage safe coding standards
3. Use safe standard libraries
4. Include additional code to detect corruption of the stack frame

Compile-time Techniques

Choice of programming language

- Write the program using a modern high-level programming language that has a strong notion of variable type and what constitutes permissible operations on them
- The flexibility and safety provided by these languages does come at a cost in resource use, both at compile time and also in additional code that must execute at runtime

Safe coding techniques

- Programmers need to inspect the code and rewrite any unsafe coding constructs
- Among other technology changes, programmers have under-taken an extensive audit of the existing code base, including the operating system, standard libraries, and common utilities

Compile-time Techniques

Language extensions and use of safe libraries

- There have been a number of proposals to augment compilers to automatically insert range checks on pointer references
- Libsafe is an example that implements the standard semantics but includes additional checks to ensure that the copy operations do not extend beyond the local variable space in the stack frame

Stack protection mechanisms

- An effective method for protecting programs against classic stack overflow attacks is to instrument the function entry and exit code to set up and then check its stack frame for any evidence of corruption
- Stackguard, one of the best-known protection mechanisms, is a GNU Compile Collection (GCC) compiler extension that inserts additional function entry and exit code

Runtime Defenses

Can be deployed in operating systems and updates and can provide some protection for existing vulnerable programs

These defenses involve changes to the memory management of the virtual address space of processes

1. Alter the properties of regions of memory
2. Or make predicting the location of targeted buffers sufficiently difficult to thwart many types of attacks

Runtime Techniques - 1

Executable address space protection

- A possible defense is to block the execution of code on the stack, on the assumption that executable code should only be found elsewhere in the processes address space
- Extensions have been made available to Linux, BSD, and other UNIX-style systems to support the addition of the no-execute bit

Address space randomization

- A runtime technique that can be used to thwart attacks involves manipulation of the location of key data structures in the address space of a process
- Moving the stack memory region around by a megabyte or so has minimal impact on most programs but makes predicting the targeted buffer's address almost impossible

Runtime Techniques - 2

Guard pages

- Gaps are placed between the ranges of addresses used for each of the components of the address space
- These gaps, or guard pages, are flagged in the MMU as illegal addresses and any attempt to access them results in the process being aborted
- A further extension places guard pages between stack frames or between different allocations on the heap

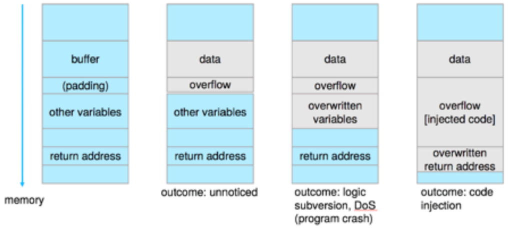
Code Injection Attack

Code-injection attack occurs when system code is not malicious but has bugs allowing executable code to be added or modified

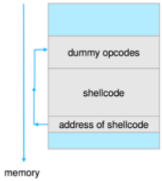
- Results from poor or insecure programming paradigms, commonly in low level languages like C or C++ which allow for direct memory access through pointers
- Goal is a buffer overflow in which code is placed in a buffer and execution caused by the attack
- Can be run by script kiddies - use tools written but exploit identifiers

Code Injection (cont.)

Outcomes from code injection include:



Frequently use trampoline to code execution to exploit buffer overflow:



Great Programming Required?

For the first step of determining the bug, and second step of writing exploit code, yes

Script kiddies can run pre-written exploit code to attack a given system

Attack code can get a shell with the processes owner's permissions

- Or open a network port, delete files, download a program, etc

Depending on bug, attack can be executed across a network using allowed connections, bypassing firewalls

Buffer overflow can be disabled by disabling stack execution or adding bit to page table to indicate "non-executable" state

- Available in SPARC and x86
- But still have security exploits

User Authentication

Crucial to identify user correctly, as protection systems depend on user ID

User identity most often established through passwords, can be considered a special case of either keys or capabilities

Passwords must be kept secret

- Frequent change of passwords, history to avoid repeats
- Use of "non-guessable" passwords
- Log all invalid access attempts (but not the passwords themselves)

Passwords may also either be encrypted or allowed to be used only once

- Does encrypting passwords solve the exposure problem?
  - ▷ Might solve sniffing
  - ▷ Consider shoulder surfing
  - ▷ Consider Trojan horse keystroke logger
  - ▷ How are passwords stored at authenticating site?

Passwords

Encrypt to avoid having to keep secret

- But keep secret anyway (i.e. Unix uses superuser-only readably file `/etc/shadow`)
- Use algorithm easy to compute but difficult to invert
- Only encrypted password stored, never decrypted
- Add "salt" to avoid the same password being encrypted to the same value

One-time passwords

- Use a function based on a seed to compute a password, both user and computer
- Hardware device / calculator / key fob to generate the password
  - ▷ Changes very frequently

Biometrics - Some physical attribute (fingerprint, hand scan)

Multi-factor authentication

- Need two or more factors for authentication
  - ▷ i.e. USB "dongle", biometric measure, and password

Implementing Security Defenses

Defense in depth is most common security theory - multiple layers of security

Security policy describes what is being secured

Vulnerability assessment compares real state of system / network compared to security policy

Intrusion detection endeavors to detect attempted or successful intrusions

- Signature-based detection spots known bad patterns
- Anomaly detection spots differences from normal behavior
  ▷Can detect zero-day attacks
- False-positives and false-negatives a problem

Virus protection

- Searching all programs or programs at execution for known virus patterns
- Or run in sandbox so can't damage system

Auditing, accounting, and logging of all or specific system or

Firewalling to Protect Systems and Networks

A network firewall is placed between trusted and untrusted hosts

- The firewall limits network access between these two security domains

Can be tunneled or spoofed

- Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
- Firewall rules typically based on host name or IP address which can be spoofed

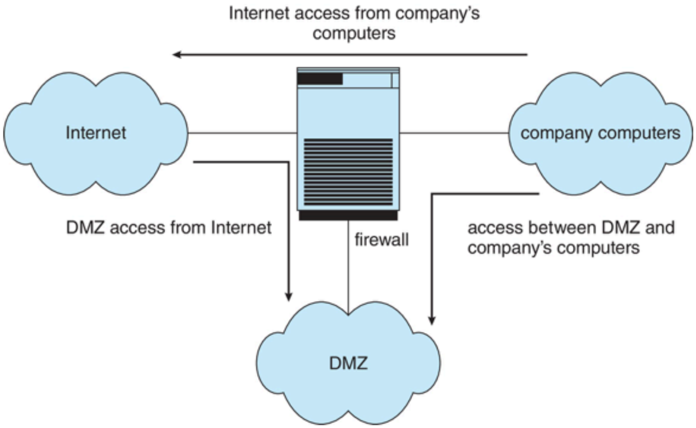
Personal firewall is software layer on given host

- Can monitor / limit traffic to and from the host

Application proxy firewall understands application protocol and can control them (i.e., SMTP)

System-call firewall monitors all important system calls and apply rules to them (i.e., this program can execute that system call)

Network Security Through Domain Separation Via Firewall


Internet access from company's computers
Internet
company computers
DMZ access from Internet
firewall
access between DMZ and company's computers
DMZ

Computer Security Classifications

U.S. Department of Defense outlines four divisions of computer security: A, B, C, and D

D - Minimal security

C - Provides discretionary protection through auditing

- Divided into C1 and C2
  - ▷ C1 identifies cooperating users with the same level of protection
  - ▷ C2 allows user-level access control

B - All the properties of C, however each object may have unique sensitivity labels

- Divided into B1, B2, and B3

A - Uses formal design and verification techniques to ensure security

Security Defenses Summarized

By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers.

- Educate users about safe computing-don't attach devices of unknown origin to the computer, don't share passwords, use strong passwords, avoid falling for social engineering appeals, realize that an e-mail is not necessarily a private communication, and so on
- Educate users about how to prevent phishing attacks-don't click on email attachments or links from unknown (or even known) senders; authenticate (for example, via a phone call) that a request is legitimate
- Use secure communication when possible
- Physically protect computer hardware
- Configure the operating system to minimize the attack surface; disable all unused services

Security Defenses Summarized (cont.)

- Configure system daemons, privileges applications, and services to be as secure as possible
- Use modern hardware and software, as they are likely to have up-to-date security features
- Keep systems and applications up to date and patched
- Only run applications from trusted sources (such as those that are code signed)
- Enable logging and auditing; review the logs periodically, or automate alerts
- Install and use antivirus software on systems susceptible to viruses, and keep the software up to date
- Use strong passwords and passphrases, and don?t record them where they could be found

Security Defenses Summarized (cont.)

- Use intrusion detection, firewalling, and other network-based protection systems as appropriate
- For important facilities, use periodic vulnerability assessments and other testing methods to test security and response to incidents
- Encrypt mass-storage devices, and consider encrypting important individual files as well
- Have a security policy for important systems and facilities, and keep it up to date

Protection

Protection

Goals of Protection

In one protection model, computer consists of a collection of objects, hardware or software

Each object has a unique name and can be accessed through a well-defined set of operations

Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

Principles of Protection

Guiding principle - principle of least privilege

- Programs, users and systems should be given just enough privileges to perform their tasks
- Properly set permissions can limit damage if entity has a bug, gets abused
- Can be static (during life of system, during life of process)
- Or dynamic (changed by process as needed) - domain switching, privilege escalation
- Compartmentalization a derivative concept regarding access to data

  ▷ Process of protecting each individual system component through the use of specific permissions and access restrictions

Principles of Protection (Cont.)

Must consider "grain" aspect

- Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
  ▷For example, traditional Unix processes either have abilities of the associated user, or of root
- Fine-grained management more complex, more overhead, but more protective
  ▷File ACL lists, RBAC

Domain can be user, process, procedure

Audit trail - recording all protection-orientated activities, important to understanding what happened, why, and catching things that shouldn't

No single principle is a panacea for security vulnerabilities - need defense in depth

Protection Rings

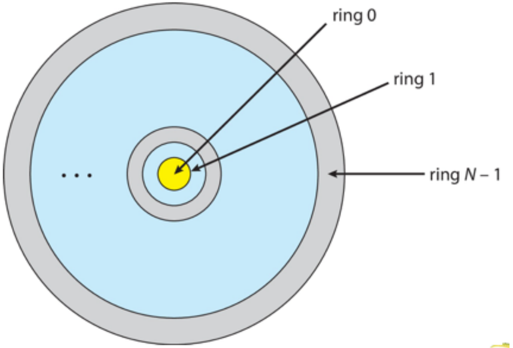
Components ordered by amount of privilege and protected from each other

- For example, the kernel is in one ring and user applications in another
- This privilege separation requires hardware support
- Gates used to transfer between levels, for example the syscall Intel instruction
- Also traps and interrupts
- Hypervisors introduced the need for yet another ring
- ARMv7 processors added TrustZone(TZ) ring to protect crypto functions with access via new Secure Monitor Call (SMC) instruction
  ▷ Protecting NFC secure element and crypto keys from even the kernel

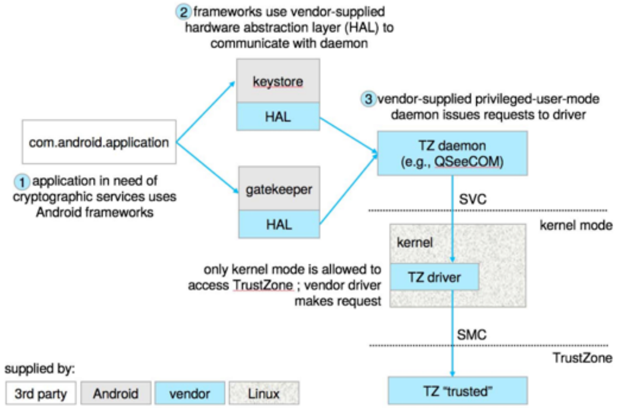
Protection Rings (MULTICS)

Let D_i and D_j be any two domain rings

If j < I $\Rightarrow D_i \subseteq D_j$

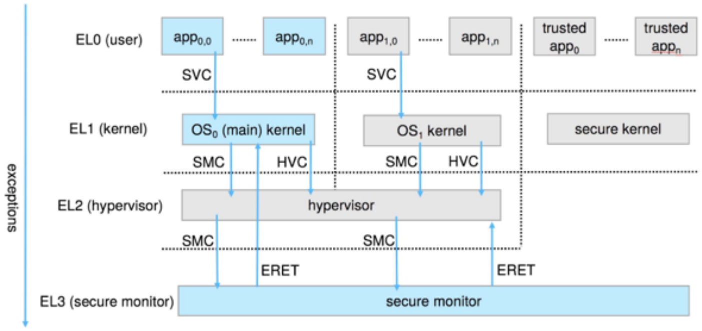


Android use of TrustZone



2 frameworks use vendor-supplied hardware abstraction layer (HAL) to communicate with daemon
keystore
HAL
3 vendor-supplied privileged-user-mode daemon issues requests to driver
com.android.application
TZ daemon (e.g., QSeeCOM)
1 application in need of cryptographic services uses Android frameworks
gatekeeper
HAL
SVC
kernel mode
kernel
TZ driver
only kernel mode is allowed to access TrustZone ; vendor driver makes request
SMC
TrustZone
supplied by:
3rd party
Android
vendor
Linux
TZ "trusted"

ARM CPU Architecture



Domain of Protection

Rings of protection separate functions into domains and order them hierarchically

Computer can be treated as processes and objects

- Hardware objects (such as devices) and software objects (such as files, programs, semaphores)

Process for example should only have access to objects it currently requires to complete its task - the need-to-know principle

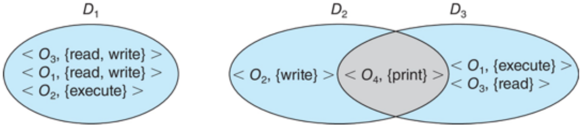
Domain of Protection (Cont.)

Implementation can be via process operating in a protection domain

- Specifies resources process may access
- Each domain specifies set of objects and types of operations on them
- Ability to execute an operation on an object is an access right
  - ▷ <object-name, rights-set>
- Domains may share access rights
- Associations can be static or dynamic
- If dynamic, processes can domain switch

Domain Structure

Access-right = <object-name, rights-set> where rights-set is a subset of all valid operations that can be performed on the object Domain = set of access-rights



Domain Implementation (UNIX)

Domain = user-id

Domain switch accomplished via file system
▷ Each file has associated with it a domain bit (setuid bit)
▷ When file is executed and setuid = on, then user-id is set to owner of the file being executed
▷ When execution completes user-id is reset

Domain switch accomplished via passwords

- `su` command temporarily switches to another user?s domain when other domain?s password provided

Domain switching via commands

- `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (Android App IDs)

Distinct user IDs are provided on a per-application basis

When an application is installed, the installd daemon assigns it a distinct user ID (UID) and group ID (GID), along with a private data directory (/data/data/<appname>) whose ownership is granted to this UID/GID combination alone.

Applications on the device enjoy the same level of protection provided by UNIX systems to separate users

A quick and simple way to provide isolation, security, and privacy.

The mechanism is extended by modifying the kernel to allow certain operations (such as networking sockets) only to members of a particular GID (for example, AID INET, 3003)

A further enhancement by Android is to define certain UIDs as "isolated", prevents them from initiating RPC requests to any but a bare minimum of services

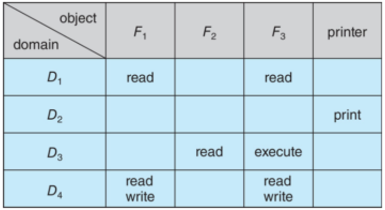
Access Matrix

View protection as a matrix (access matrix)

Rows represent domains

Columns represent objects

`Access(i, j)` is the set of operations that a process executing in *Domain*$_i$ can invoke on *Object*$_j$



object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix

User who creates object can define access column for that object

Can be expanded to dynamic protection

- Operations to add, delete access rights
- Special access rights:
  - ▷ *owner* of O_i
  - ▷ *copy* op from O_i to O_j (denoted by "*")
  - ▷ *control* - D_i can modify D_j access rights
  - ▷ *transfer* - switch from domain D_i to D_j
- *Copy* and *Owner* applicable to an object
- *Control* applicable to domain object

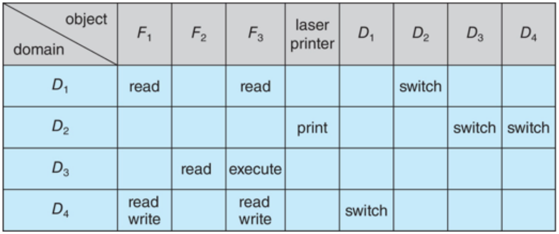
Use of Access Matrix (Cont.)

Access matrix design separates mechanism from policy

- Mechanism
  - ▷ Operating system provides access-matrix + rules
  - ▷ If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
- Policy
  - ▷ User dictates policy
  - ▷ Who can access what object and in what mode

But doesn't solve the general confinement problem

Access Matrix of Figure A with Domains as Objects



domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with *Copy* Rights

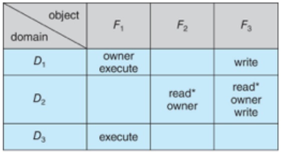
object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

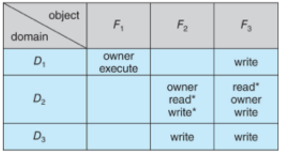
(b)

Access Matrix With *Owner* Rights



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

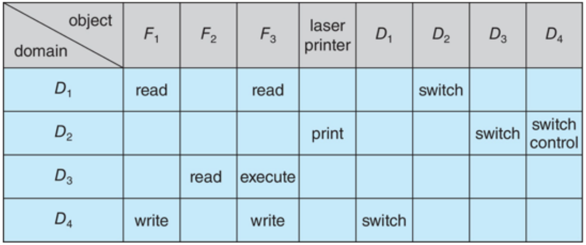
(a)



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Modified Access Matrix of Figure B



object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Implementation of Access Matrix

Generally, a sparse matrix

Option 1 ? Global table

- Store ordered triples <domain, object, rights-set> in table
- A requested operation M on object O_j within domain D_i -> search table for $< D_i, O_j, R_k >$
  - ▷ with M $\in R_k$
- But table could be large -> won't fit in main memory
- Difficult to group objects (consider an object that all domains can read)

Implementation of Access Matrix (Cont.)

Option 2 - Access lists for objects

- Each column implemented as an access list for one object
- Resulting per-object list consists of ordered pairs `<domain, rights-set>` defining all domains with non-empty set of access rights for the object
- Easily extended to contain default set -> If M $\in$ default set, also allow access

Implementation of Access Matrix (Cont.)

Each column = Access-control list for one object
Defines who can perform what operation

- Domain 1 = Read, Write
- Domain 2 = Read
- Domain 3 = Read

Each Row = Capability List (like a key)
For each domain, what operations allowed on what objects

- Object F1 - Read
- Object F4 - Read, Write, Execute
- Object F5 - Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

Option 3 - Capability list for domains

- Instead of object-based, list is domain based
- Capability list for domain is list of objects together with operations allows on them
- Object represented by its name or address, called a capability
- Execute operation M on object Oj, process requests operation and specifies capability as parameter
  ▷ Possession of capability means access is allowed
- Capability list associated with domain but never directly accessible by domain
  ▷ Rather, protected object, maintained by OS and accessed indirectly
  ▷ Like a "secure pointer"
  ▷ Idea can be extended up to applications

Implementation of Access Matrix (Cont.)

Option 4 - Lock-key

- Compromise between access lists and capability lists
- Each object has list of unique bit patterns, called locks
- Each domain as list of unique bit patterns called keys
- Process in a domain can only access object if domain has key that matches one of the locks

Comparison of Implementations

Many trade-offs to consider

- Global table is simple, but can be large
- Access lists correspond to needs of users
  ▷ Determining set of access rights for domain non-localized so difficult
  ▷ Every access to an object must be checked
    – Many objects and access rights -> slow
- Capability lists useful for localizing information for a given process
  ▷ But revocation capabilities can be inefficient
- Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (Cont.)

Most systems use combination of access lists and capabilities

- First access to an object -> access list searched
  - ▷ If allowed, capability created and attached to process
    - – Additional accesses need not be checked
  - ▷ After last access, capability destroyed
  - ▷ Consider file system with ACLs per file

Revocation of Access Rights

Various options to remove the access right of a domain to an object

- Immediate vs. delayed
- Selective vs. general
- Partial vs. total
- Temporary vs. permanent

Access List - Delete access rights from access list

- Simple - search access list and remove entry
- Immediate, general or selective, total or partial, permanent or temporary

Revocation of Access Rights (Cont.)

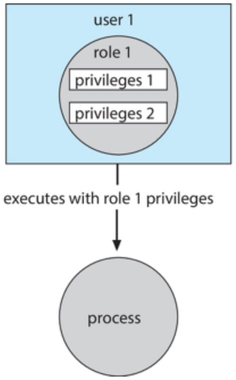
Capability List - Scheme required to locate capability in the system before capability can be revoked

- Reacquisition - periodic delete, with require and denial if revoked
- Back-pointers - set of pointers from each object to all capabilities of that object (Multics)
- Indirection - capability points to global table entry which points to object - delete entry from global table, not selective (CAL)
- Keys - unique bits associated with capability, generated when capability created
  ▷Master key associated with object, key matches master key for access
  ▷Revocation - create new master key
  ▷Policy decision of who can create and modify keys - object owner or others?

Role-based Access Control

Protection can be applied to non-file resources Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege

- Privilege is right to execute system call or use an option within a system call
- Can be assigned to processes
- Similar to access matrix



Users assigned roles granting access to privileges and programs

▷ Enable role via password to gain its privileges

Mandatory Access Control (MAC)

OS traditionally had discretionary access control (DAC) to limit access to files and other objects (for example UNIX file permissions and Windows access control lists (ACLs))

- Discretionary is a weakness - users / admins need to do something to increase protection

Stronger form is mandatory access control, which even root user can't circumvent

- Makes resources inaccessible except to their intended owners
- Modern systems implement both MAC and DAC, with MAC usually a more secure, optional configuration (Trusted Solaris, TrustedBSD (used in macOS), SELinux), Windows Vista MAC)

At its heart, labels assigned to objects and subjects

- When a subject requests access to an object, policy checked to determine whether or not a given label-holding subject is allowed to perform the action on the object

Capability-Based Systems

Hydra and CAP were first capability-based systems Now

included in Linux, Android and others, based on POSIX.1e (that never became a standard)

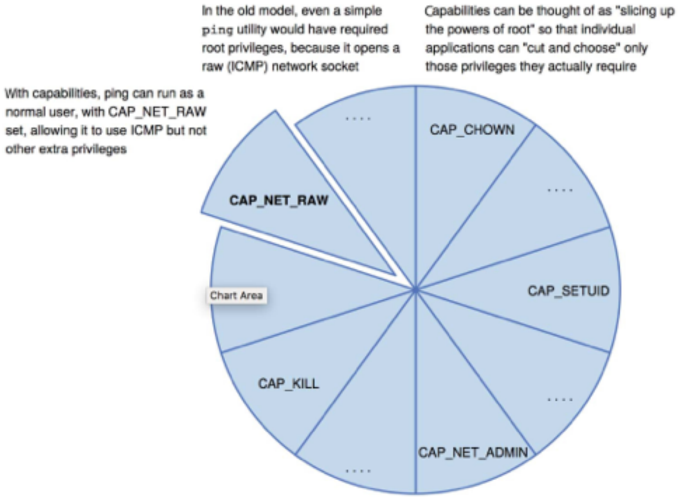
- Essentially slices up root powers into distinct areas, each represented by a bitmap bit
- Fine grain control over privileged operations can be achieved by setting or masking the bitmap
- Three sets of bitmaps - permitted, effective, and inheritable

Capability-Based Systems (Cont.)

▷ Can apply per process or per thread
▷ Once revoked, cannot be reacquired
▷ Process or thread starts with all privs, voluntarily decreases set during execution
▷ Essentially a direct implementation of the principle of least privilege

- An improvement over root having all privileges but inflexible (adding new privilege difficult, etc)

Capabilities in POSIX.1e


In the old model, even a simple `ping` utility would have required root privileges, because it opens a raw (ICMP) network socket
Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require
With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges
....
CAP_CHOWN
....
CAP_NET_RAW
Chart Area
CAP_SETUID
CAP_KILL
....
CAP_NET_ADMIN
....

Other Protection Improvement Methods

System integrity protection (SIP)

- Introduced by Apple in macOS 10.11
- Restricts access to system files and resources, even by root
- Uses extended file attribs to mark a binary to restrict changes, disable debugging and scrutinizing
- Also, only code-signed kernel extensions allowed and configurably only code-signed apps

System-call filtering

- Like a firewall, for system calls
- Can also be deeper - inspecting all system call arguments
- Linux implements via SECCOMP-BPF (Berkeley packet filtering)

Other Protection Improvement Methods (cont.)

Sandboxing

- Running process in limited environment
- Impose set of irremovable restrictions early in startup of process (before `main()`)
- Process then unable to access any resources beyond its allowed set
- Java and .net implement at a virtual machine level
- Other systems use MAC to implement
- Apple was an early adopter, from macOS 10.5's "seatbelt" feature

  ▷ Dynamic profiles written in the Scheme language, managing system calls even at the argument level

  ▷ Apple now does SIP, a system-wide platform profile

Other Protection Improvement Methods (cont.)

Code signing allows a system to trust a program or script by using crypto hash to have the developer sign the executable

- So code as it was compiled by the author
- If the code is changed, signature invalid and (some) systems disable execution
- Can also be used to disable old programs by the operating system vendor (such as Apple) cosigning apps, and then invaliding those signatures so the code will no longer run

Language-Based Protection

Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources

Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable

Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

Protection in Java 2

Protection is handled by the Java Virtual Machine (JVM)

A class is assigned a protection domain when it is loaded by the JVM

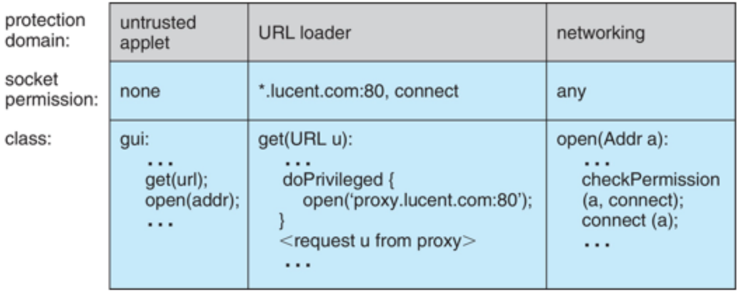
The protection domain indicates what operations the class can (and cannot) perform

If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

Generally, Java's load-time and run-time checks enforce type safety

Classes effectively encapsulate and protect data and methods from other classes

Stack Inspection



protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	. . .
	get(url);
	open(addr);
	. . .
		get(URL u):
		. . .
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		. . .
			open(Addr a):
			. . .
			checkPermission
			(a, connect);
			connect (a);
			. . .

Thank you !

Operating Systems are among the most complex pieces of software ever developed !