Lecture.14.Security.Attack.Examples.Protection.txt

- Buffer Overflow Attacks
  - Also known as a buffer overrun
  - Defined by the National Institute of Standards and Technology (NIST) as:
    - "A condition at an interface under which more input can be placed into a buffer or data-holding area than then capacity allocated, overwriting other information.
      Attackers can 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
  - i.e. Buffer Overrun Demonstration

```
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
  - i.e. Execution of Buffer Overrun Program

```
$ 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), st2(BADINPUTBADINPUT), valid(1)
```

- Buffer Overflow Stack
  - i.e. Figure of Stack After Buffer Overrun

```
|----------|------------|------------|-----------|
| Memory   | Before     | After      | Contains  |
```

Address	gets(str2)	gets(str2)	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	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

        - Aims to harden programs to resist attacks
    - Runtime defenses
        - 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:
    - Some techniques are:
        1. Choose a high-level language that does not permit buffer overflows/overruns
        2. Encourage safe coding standards
        3. Use safe standard libraries
        4. Include additional code to detect corruption of the stack frame

- Compile Time Techniques (1)
    - Choice of the 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 exeute 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 (2)
    - Language extensions and use of safe libraries
        - There may 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 by:
  - Altering the properties of regions of memory
    OR
  - Making predictions of the location of targeted buffers sufficiently difficult to thwart many types of attacks

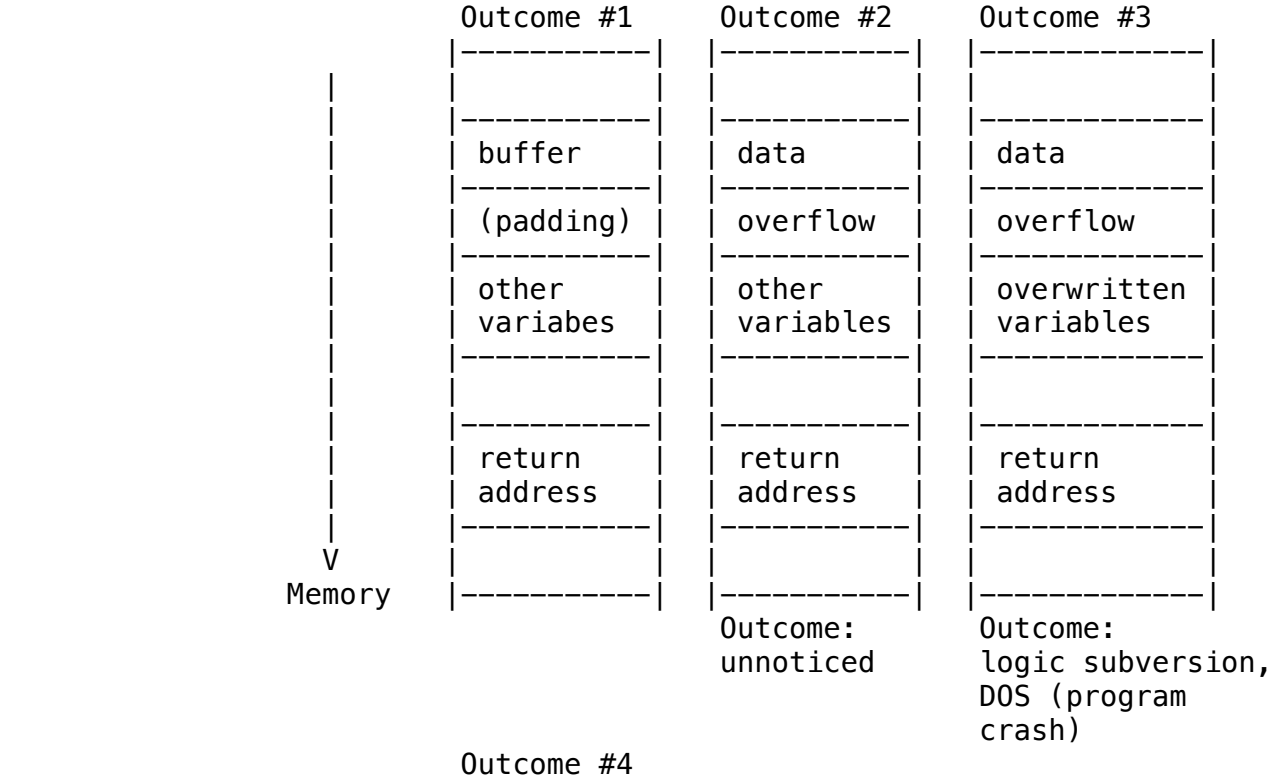
- Runtime Techniques (1)
  - Executable address space protection
    - A possible defense is to block the execution of code on the stack, on the assumption that executable code should only be found elsewhere in the processes address space
    - Extensions have been made available to Linux, BSD, and other 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

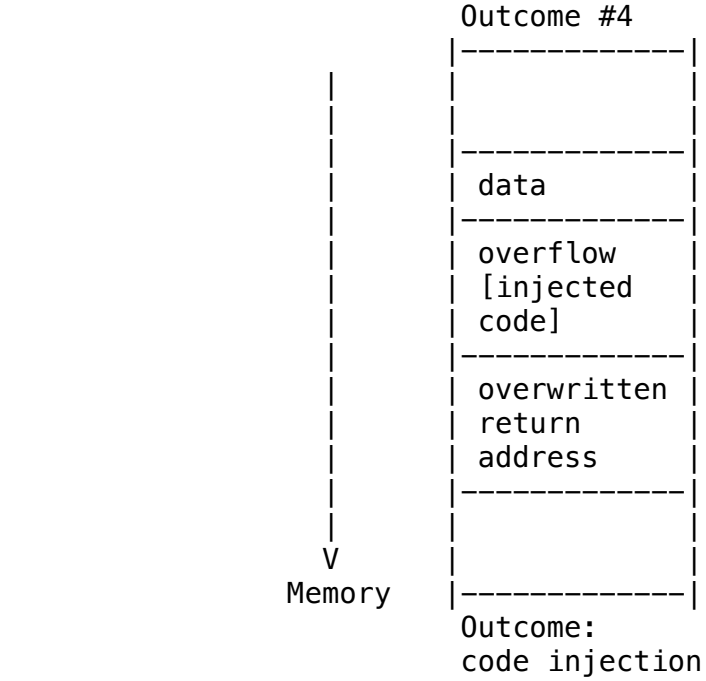
- Runtime Techniques (2)
  - Guard pages
    - Gaps are placed between the ranges of addresses used for each of the components of the address space
    - These gaps, or guard pages, are flagged in the MMU as illegal addresses, and any attempt to access them results in the process being aborted
    - A further extension places guard pages between stack frames or between different allocations on the heap

- Code Injection Attack
  - 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 allows 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
      - Script kiddies are cowards that use tools written by other people, and exploit security holes found by other people for their own gain
        - They put two and two, together, and act like they're intelligent

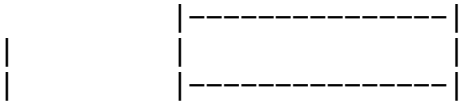
- Code Injection

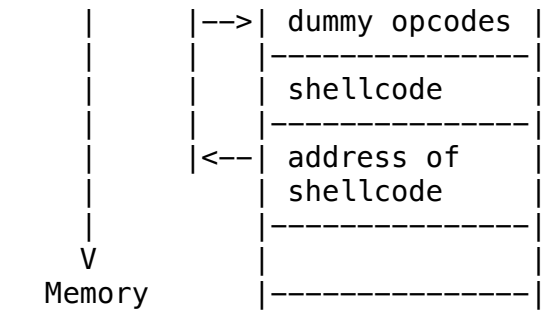
- Outcomes from code injection include:
  - i.e. Diagram of Code Injection





- Frequently use trampoline to code execution to exploit buffer overflow:
  i.e. Diagram





- Great Programming Required?
  - Yes
    - For the first step of determining the bug, and second step of writing the exploit code
  - 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 contains security exploits

- User Authentication
  - It is crucial to identify users correctly
    - Some protection systems depend on user ID
  - User identity most often established through passwords
    - Can be considered a special case for either keys or capabilities
  - Passwords must be kept secret
    - Frequent change of password, history to avoid repeats
    - Use of "non-guessable" passwords
    - Log all invalid access attempts
      - But not the password 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?
        - i.e. Plaintext, MD5, SHASUM, Etc.

- Passwords
  - Encrypt to avoid having to keep secret
    - But keep secret anyway

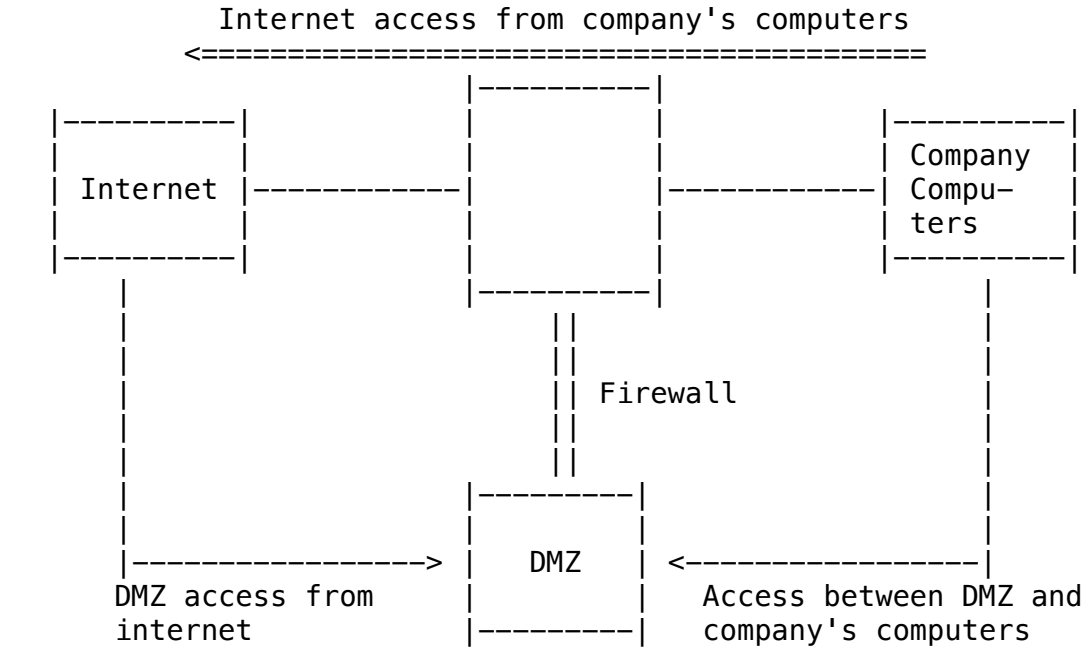
            - i.e. UNIX uses super-user only readable file
                - Located at `/etc/shadow`
        - Use an algorithm that is easy to compute but difficult to invert
            - i.e. One way algorithm
        - Only store the encrypted version of the password
            - Never ever store the decrypted version of the password
        - 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
        - Use a hardware device/calculator to generate the password
            - Changes very frequently
                - i.e. Key fob
    - Biometrics
        - Some physical attribute
            - i.e. Fingerprint, hand scan, face, etc.
    - Multi-factor authentication
        - Need two or more factors for authentication
            - i.e. USB "dongle", biometric measure, password, text via SMS

- 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
                - A zero-day attack, also known as 0-day, is an exploit that takes advantage of a bug that is new, and unknown to the vendors of the product
        - False-positives and false-negatives are a problem
    - Virus protection
        - Searching all programs, or programs at execution, for known virus pattern(s)
        - Or run in sandbox-mode, so it can't damage the system
    - Auditing, accounting, and logging of all or specific system

- Firewalling To Protect Systems & 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 applies rules to them
- i.e. 'This' program can execute 'that' system call

- Network Security Through Domain Separation Via Firewall
- i.e. Diagram of Network Security



- Computer Security Classifications
- U.S. Department of Defense (DOD) outlines 4 divisions of computer security: A, B, C, & D
- D = Minimal security
- C = Provides discretionary protection through auditing
- Further 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 (1)
    - By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers
        - Educate users about safe computing
            - i.e.
                - 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 emails are not a medium for secure and private communication
                - Etc.
        - Educate users about how to prevent phishing attacks:
            - i.e.
                - Don't click on email attachments or links from unknown, or even known senders
                    - i.e. Nigerian Prince
                - Always authenticate that a request is legitimate
                    - i.e. Contact your bank, via phone, to see if they really did send out an email containing an offer; forward them the email
        - Use secure communication when possible
            - i.e. Messaging services that use end-to-end encryption, and make sure that their code is open source
        - Physically protect computer hardware
        - Configure the operating system to minimize the attack surface
            - i.e. Disable all unused services
                - i.e. Bluetooth, GPS, Webcam, Microphone, Etc.

- Security Defenses Summarized (2)
    - Configure system daemons, privileges applicaions, and services to be as secure as possible
    - Use modern hardware and software, as they are likely to have up-to-date security features
        - i.e. Newer iPhones have hardware to thwart attacks
        - i.e. iOS 14 contains more security features than iOS 3
    - Keep systems and applications up-to-date and patched
    - Only run applications from trusted sources
        - Such as those that are code signed
            - Avoid running apps that you download off the internet
    - 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
        - i.e. Don't write down your password on a sticky note and

tape it next to your monitor

- Security Defenses Summarized (3)
  - 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
    - i.e. Use TrueCrypt, VeraCrypt, Bitlocker, etc. for your storage devices
  - Have a security policy for important systems and facilities, and keep them 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

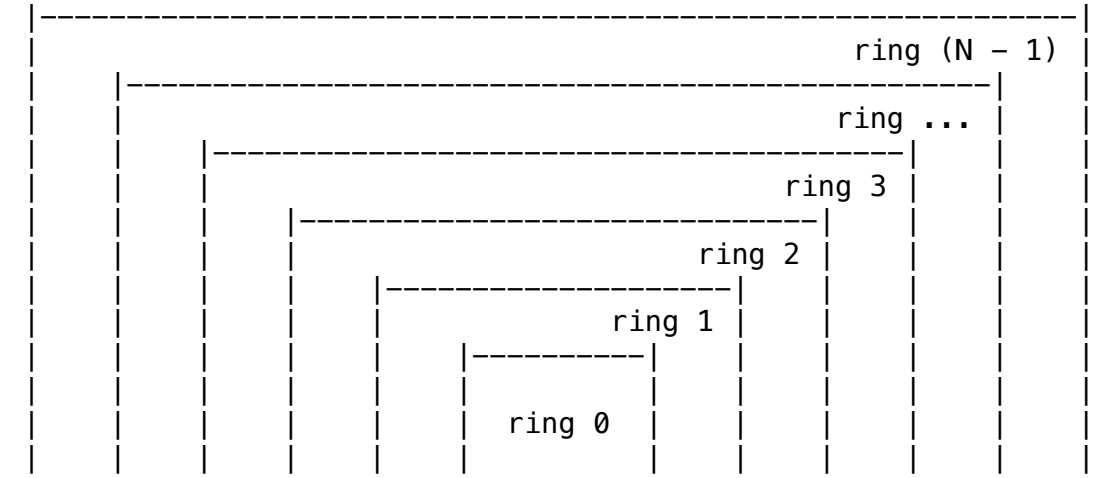
- Principles Of Protection (1)
  - Guiding principle: Principle of least privilege
    - Programs, users, and systems should be given just enough privileges to perform their tasks
    - Permissions that are properly set can limit damage if an entity has a bug, and exploits it
    - Can be static
      - i.e. During life of system, during life of process, etc.
    - Can be dynamic
      - Changed by process as needed
        - i.e. Domain switching, privilege escalation, etc.
    - Compartmentalization a derivative concept regarding access to data
      - Process of protecting each individual system component through the use of specific permissions and access restrictions
  - An implementation of the principle of least privilege is: "When working out of the terminal, do not work as 'root'. Only use `sudo` when it is necessary, and work as 'user' all other times."

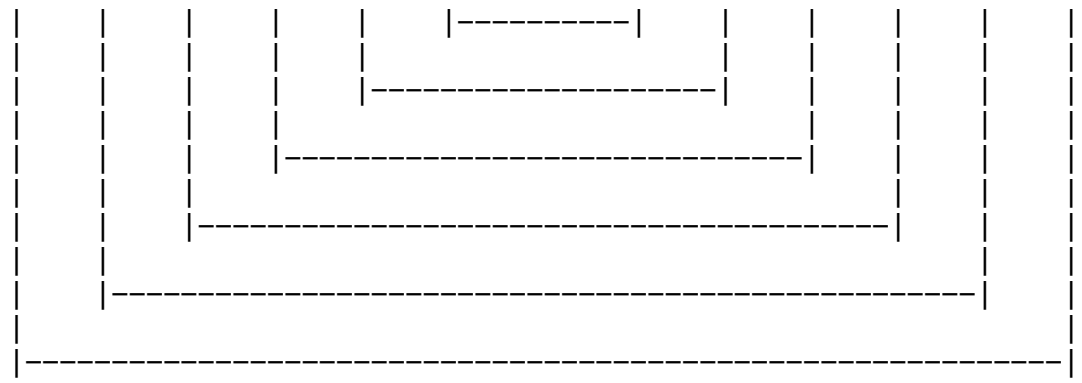
- Principles Of Protection (2)
  - Must consider "grain" aspect

        - Rough grained privilege management is easier and simpler, but least privilege now done in large chunks
            - i.e. Traditional UNIX processes either have abilities of the associated user, or of root
        - Fine grained management more complex, more overhead, but more protective
            - i.e. File ACL lists, RBAC, etc.
    - Domain can be user, process, procedure, etc.
    - Audit trail
        - Recording all protection orientated activites
            - It is important to understand what happened, why it happened, and catch things that shouldn't exist
    - No single principal is a panacea for security vulnerabilities
        - Need defense in depth
        - 'panacea' means "the ultimate, one and only, solution"

- Protection Rings
    - Components ordered by amount of privilege and protected from each other
        - i.e. The kernel is in one ring, and user applications are in another
        - Privilege separation requires hardware support
        - Gates used to transfer between levels
            - i.e. The 'syscall' Intel instruction
        - Also applies to traps and interrupts
        - Hypervisors introduced the need for yet another ring
        - ARMv7 processors added TrustZone (TZ) ring to protect crypo functions with access via new Secure Moniter Call (SMC) instruction
            - Protects NFC secure element and crypto keys
                - TrustZone even protects it from the kernel

- Protection Rings (MULTICS)
    - Let `D_i` and `D_j` be any two domain rings
        - If `j` < `i` => `D_i` is a subset of `D_j`
    - i.e. Diagram of Rings





- Android Use Of TrustZone
    1. Application in need of crytographic services uses Android frameworks
    2. Frameworks use vendor-supplied hardware abstraction layer (HAL) to communicate with daemon
    3. Vendor supplied privileged user mode daemon issues requests to driver
    - Only kernel is allowed to access TrustZone
        - The vendor driver must make a request

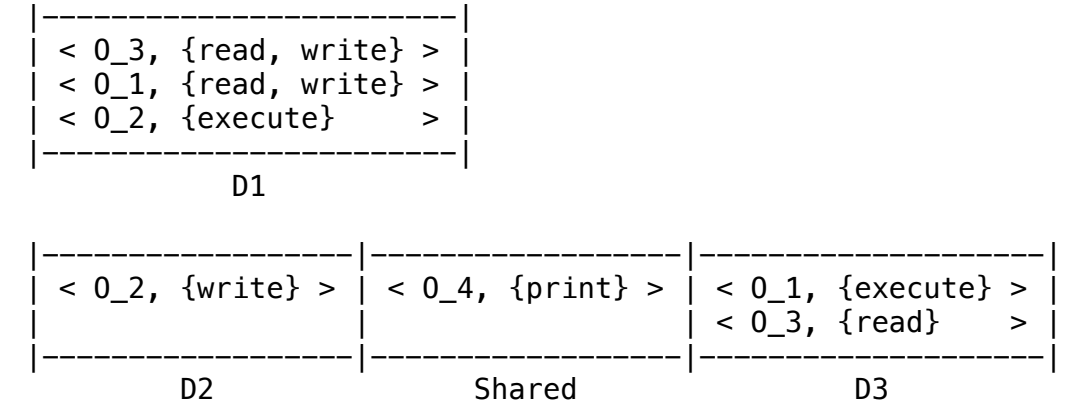
- ARM CPU Architecture
    - Consists of:
        - EL0
            - User (applications)
        - EL1
            - Kernel
        - EL2
            - Hypervisor
        - EL3
            - Secure monitor

- Domain Of Protection (1)
    - Rings of protection separate functions into domains and order them hierarchically
    - Computer can be treated as processes and objects
        - Hardware objects
            - i.e. Devices
        - Software objects
            - i.e. Files, programs, semaphores, etc.
    - Process, for example, should only have access to objects it currently requires to complete its task
        - This is the need to know principle
            - i.e. The calculator app that you downloaded on your iPhone doesn't need access to your contacts and personal pictures to crunch numbers

- Domain Of Protection (2)
    - Implementation can be via process operating in a protection

domain
  - Specifies the resources a 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
    - i.e. <object-name, rights-set>
  - Domains may share access rights
  - Associations can be static or dynamic
    - If dynamic, process can switch domains

- 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
  - i.e. Diagrams of Domain Structures



- Domain Implementation (UNIX)
  - Domain = user-id
  - Domain switching can be accomplished via the file system
    - Each file has associated with it a domain bit
      - i.e. '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 switching can be accomplished via password
    - `su` command temporarily switches to another user's domain when the other domain's password is provided
  - Domain switching can be done via commands
    - `sudo` command prefix executes specified command in another domain, if original domain has privilege or correct password is entered

- 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

  data directory (i.e. /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
- This is 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
  - i.e. AID INET, 3003
- A further enhancement by Android is to define certain UIDs as "isolated"
  - This prevents them from initiating RPC requests to any but a bare minimum of services

- Access Matrix
  - View protection as a matrix
    - This is the 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`
  - i.e. Example of Access Matrix

Domain \ Object	F1	F2	F3	printer
D1	read			
D2				print
D3		read	execute	
D4	read, write			

- Use of Access Matrix (1)
  - If a process in Domain `D_i` tries to execute "operation" on object `O_j`, then "operation" must be in the access matrix
  - User who creates object can define access column for that object
  - Can be expanded to dynamic protection
    - There are operations to add and delete access rights
    - Special access rights are:
      - `owner` of `O_i`
      - `copy` "operation" 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 (2)
    - Access matrix design separates mechanism from policy
        - Mechanism
            - Operating system provides access-matrix + rules
            - It 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 this does not solve the general confinement problem

- Access Matrix Of Figure (A) With Domains As Objects
    - i.e. Access Matrix Example

Domain	Objects							
	F1	F2	F3	laser printer	D1	D2	D3	D4
D1	r		r			sw		
D2				print			sw	sw
D3		r	x					
D4	r, w		r, w		sw			

    - Legend:
        - r = read
        - w = write
        - x = execute
        - sw = switch

- Access Matrix With Copy Rights
    - i.e. Access Matrix Example A

\ Object	F1	F2	F3

Domain \\			
D1	execute		write*
D2	execute	read*	execute
D3	execute		

- i.e. Access Matrix Example B

\\\ Object / Domain \\	F1	F2	F3
D1	execute		write*
D2	execute	read*	execute
D3	execute	read	

- Access Matrix With Owner Rights
  - i.e. Access Matrix Example A

\\\ Object / Domain \\	F1	F2	F3
D1	owner execute		write
D2		read* owner	read* owner write
D3	execute		

  - i.e. Access Matrix Example B

\\\ Object /	F1	F2	F3

Domain \\			
D1	owner execute		write
D2		owner read* write*	read* owner write
D3		write	write

- Modified Access Matrix Of Figure (B)
  - i.e. Access Matrix Example

	Objects							
Domain	F1	F2	F3	laser printer	D1	D2	D3	D4
D1	r		r			sw		
D2				print			sw	sw co
D3		r	x					
D4	w		w		sw			

  - Legend:
    - r = read
    - w = write
    - x = execute
    - sw = switch
    - co = control

- Implementation Of Access Matrix (1)
  - 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`
      - This tells the system to search the table for: < D_i, O_j, R_k > with `M in R_k`
      - In other words: get `R_k`, which is located at row `D_i` and column `O_j`, and see if it contains `M`
    - Problems with this implementation:
      - Table can be (very) large
        - Thus, it won't fit in main memory
      - It is difficult to group objects

            - i.e. An object that all domains can read
                - Which group does the object get placed into?

- Implementation Of Access Matrix (2)
    - Option 2: Access lists for objects
        - Each column implemented as an access list for one object
        - Resulting per-object list consists of ordered pairs
            - i.e. <domain, rights-set>
                - This defines all domains with non-empty set of access rights for the object
        - Easily extended to contain default set
            - If `M` in default set, then allow access

- Implementation Of Access Matrix (3)
    - 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 (4)
    - 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 allowed on them
        - Object represented by its name or address, called a capability
        - Execute operation `M` on object `O_j`
            - The process requests the operation and specifies capability as paramter
                - Possession of capability means access is allowed
        - Capability list associated with domain but never directly accessible by domain
            - Instead, it is a protected object that is maintained by the OS and accessed indirectly
            - Like a "secure pointer"
            - Idea can be extended up to applications

- Implementation Of Access Matrix (5)
    - Option 4: Lock Key
        - Compromise between access lists and capability list
        - Each object has a list of unique bit patterns
            - Called locks
        - Each domain has 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 (1)
    - Many trade-offs to consider
        - Global table (option 1) is simple, but can be large
        - Access lists (option 2) correspond to needs of users
            - Determining set of access rights for domain
                - It is difficult because it is non-localized
            - Every access to any object must be checked
                - This is slow because there are many objects and access rights
        - Capability lists (option 3) are useful for localizing information for a given process
            - But revocation capabilities can be inefficient
        - Lock key (option 4) is effective and flexible
            - Keys can be freely passed from domain to domain
            - Revocation is easy

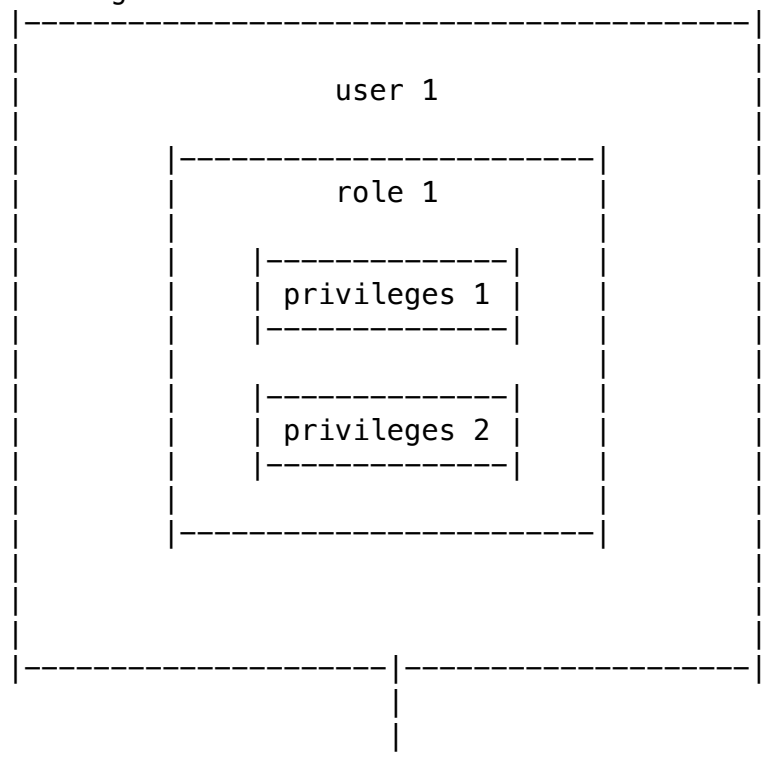
- Comparison Of Implementations (2)
    - Most systems use a combination of access lists and capabilities
        - First access to an object -> Access list searched
            - If allowed, capability created and attached to process
                - Additional accesses do not need to be checked
            - After last access, capability is destroyed
            - Consider file system with ACLs per file

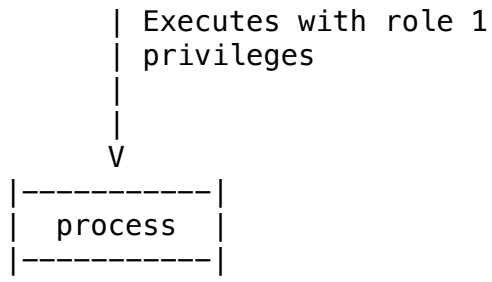
- Revocation Of Access Rights (1)
    - Various options to remove the access right of a domain to an object
        - Immediate VS. Delayed
        - Selective VS. General
        - Partial Vs. Total
        - Temporary VS. Permanent
    - In an access List, revocation is done by deleting the access rights from the access list
        - This is very simple
            - All you need to do is search the access list and remove the entry
        - Options include:
            - Immediate
            - General or selective
            - Total or partial
            - Permanent or temporary

- Revocation Of Access Rights (2)
    - In a capability list, the scheme is 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 with capability is 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
                    - Should it be the object owner, or others?

- Role Based Access Control
    - Protection can be applied to non-file resources
        - i.e. 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 granted access to privileges and programs
        - Enable role via password to gain its privileges
    - i.e. Diagram of RBAC





- Mandatory Access Control (MAC)
  - Operating systems traditionally had discretionary access control (DAC) to limit access to files and other objects
    - Examples:
      - UNIX file permissions
      - Windows access control list (ACL)
    - Discretionary is a weakness
      - Users/admins need to do something to increase protection
  - Stronger form is mandatory access control (MAC), which even root users cannot circumvent
    - Makes resources inaccessible, except to their intended owners
    - Modern systems implement both MAC and DAC
      - MAC is usually more secure, and provides more configuration options/settings
      - i.e. Trusted Solaris, TrustedBSD, SELinux, Windows Vista MAC
        - Note: TrustedBSD is used in macOS
  - At its core, MAC assigns labels to objects and subjects
    - When a subject requests access to an object, the policy is checked to determine whether or not a given label-holding subject is allowed to perform the action on the object

- Capability Based Systems (1)
  - Hydra and CAP were first capability based systems
    - It is now included in Linux, Android, and others
    - Based on POSIX.1e
      - Which never became a standard
    - Essentially, it slices up root powers into distinct areas
      - Each area is 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
        - Inheritable

- Capability Based Systems (2)
  - Three sets of bitmaps: Permitted, Effective, and Inheritable
    - Can apply per process or per thread
    - Once revoked, cannot be reacquired

        - Process or thread starts with all privileges, and voluntarily decreases set during execution
        - Essentially a direct implementation of the principle of least privilege
    - This is an improvement over 'root' having all privileges
        - However, it is inflexible
            - i.e. Adding new privilege(s) is difficult, etc.

- Capabilities In POSIX.1e
    - i.e. Chart of Capabilities

CAP_CHOWN	. . .	CAP_SETUID	. . .	CAP_NET_ADMIN
. . .	CAP_KILL	. . .	CAP_NET_RAW	. . .

        - With capabilities, `ping` can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP, but not extra privileges
        - 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

- Other Protection Improvement Methods (1)
    - System integrity protection (SIP)
        - Introduced by Apple in OS X 10.11 (El Capitan)
        - Restricts access to system files and resources
            - Also includes 'root' user
        - Uses extended file attributes to mark a binary to restrict changes, disable debugging and scrutinizing
        - Only code-signed kernel extensions allowed and configurably only code-signed apps
    - System-call filtering
        - Like a firewall, but for system calls
        - Can also be deeper
            - Inspects all system call arguments
        - Linux implements this via SECCOMP-BPF
            - BPF = Berkeley Packet Filtering

- Other Protection Improvement Methods (2)
    - Sandboxing
        - Running process in limited environment
        - Impose set of irrevocable restrictions early in startup of

process
  - This is done before 'main()'
- The process is unable to access any resources beyonds its allowed set
- Java and '.net' implement at a virtual machine level
- Other systems use mandatory access control (MAC) to implement sandboxing
- Apple was an early adopter, from Mac OS X 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

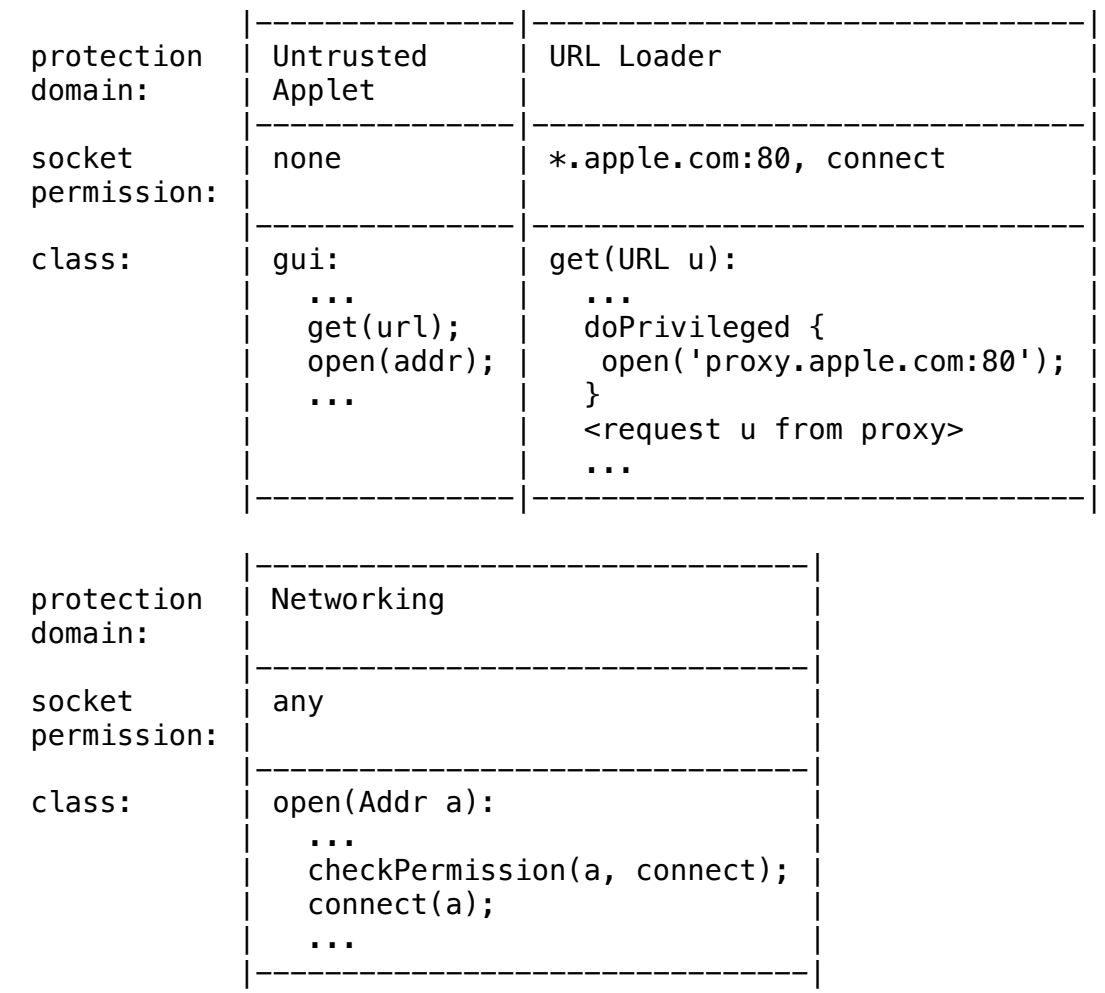
- Other Protection Improvement Methods (3)
  - Code signing allows a system to trust a program by script or using crypto hash to have the developer sign the executable
    - This is code as it was compiled by the author
    - If the code is changed (i.e. Signature is invalid), then (some) systems disable execution
    - Can also be used to disable old programs that are co-signed by the operating system vendor
      - i.e. On an iPhone, Apple has the power to invalidate signatures for apps
        - This means that the app will no longer run
          - When you tap the app, it will launch, and then immediately crash

- 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 2.0
  - 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
  - i.e. Diagram



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