MODULE 1

Security:

1. Confidentiality: Access to systems or data is limited to authorized parties
2. Integrity: When you request data, you receive the correct data
3. Availability: The system or data is there when you want it

**Reliability (keeps data confidential, only authorized access/modification, correct & meaningful when you want it)

Privacy:

- "informational self-determination" – you control information about you
- ex: PIPEDA – personal information protection and electronic document act – Canada's private-sector privacy legislation

Assets – things that we want to protect (data, hardware, software)
Vulnerabilities – weaknesses in the system that be exploited to cause harm
Threats – a loss or harm that may befall a system

Interception	- some unauthorized party has **gained access** to an asset - can go undetected – no loss necessary
Interruption	- an asset of the system becomes **lost**, **unavailable** or **unusable**
Modification	- when an unauthorized party **accesses** and **tampers** with an asset
Fabrication	- an unauthorized **party creates a fake** message/entry which is **inserted** into the system

Attack – an action which exploits a vulnerability to execute a threat
Control/Defense – removing or reducing a vulnerability

- you can control a vulnerability to prevent an attack and defend against a threat
- METHODS: (PDDDR or P3DR)
    - Prevent it
    - Deter it (make it harder)
    - Deflect it (make yourself less attractive)
    - Detect it
    - Recover from it
- We can "defence in depth" – do many things to defend against same threat

*Principle of **Easiest** Penetration* – the system is only as strong as its weakest link
*Principle of **Adequate** Protection* – don't spend too much if it's not worth it

DEFENCE OF COMPUTER SYSTEMS (**5**):

1. Cryptography
2. Software controls (password, OS separating stuff, virus scanners, firewalls)
3. Hardware controls (fingerprint readers, smart tokens, firewalls, intrusion detection)
4. Physical controls (locks, guards, offsite backups)
5. Policies and procedures

MODULE 2 – PROGRAM SECURITY

Flaw – a problem with a program
- **Fault** – a mistake "behind the scenes" (in code, data, specification, process, etc..) – potential problem (programmer/specifier/inside view)
- **Failure** – when something actually goes wrong (user/outside view)

Security flaw – a problem that affects security in some way

FIXING FAULTS:
- **Patching** – making small edits to fix a fault
    - Can make things worse

FINDING FAULTS
- Work backwards to find the fault
- Intentionally try to cause failures to find faults
- **Fuzzing** – brute-force bad inputs (state machines)

<u>Unexpected behaviours</u>
- Most go ignored (as long as the program does what it is supposed to do, it doesn't matter if it does extra things)
- This can be very bad for security/privacy
- Should ONLY do what is specified and nothing else

TYPES OF SECURITY FLAWS
- **Intentional/Inherent**
    - Malicious – intentionally inserted to attack systems
    - Non-malicious – often features that are meant to be in the system and are correctly implemented, but nonetheless can cause a failure when used by an attacker
- **Unintentional**
    - Mistakes that are non-malicious and cause failures

<u>UNINTENTIONAL</u>

Heartbleed Bug
- TLS Heartbeat mechanism keeps SSL/TLC connections alive even when no data is being transferred (OpenSSL)
- The length of data and data is sent by one party and it is echoed back by the other
- This can be exploited by the sender party sending a smaller length and a bigger data buffer, so they echoer will send back that buffer, passing some of its own portion of memory
- Missing **bounds check**

Apple's SSL/TLS Bug

- The additional "goto fail" statement that causes an immediate exit and success, making the TLS connection succeed, even though the full verification process has not taken place
- The attacker can then trick the OS into receiving certificates that it should be rejecting

Buffer overflows
- Copying data/strings without checking if it fits first
- **Smash the Stack** – an attacker may write data past the end of an array in the stack and overwrite things like the return address (to jump into shellcode)
- **Types**: single byte, heap instead of stack, jump to other parts of code
- **Defences**: bound check, padding b/w return address and data (Canaries - OS), non-executable stack

Integer overflow
- If the upper limit is reached, this can be exploited to be used to access lower memory addresses

String vulnerabilities
- Unfiltered string input is used as format string in printf(), fprintf(), sprint(), …

Incomplete mediation
- **Mediation** – when an application ensures that the user has entered meaningful request
- This occurs when application accepts incorrect data from the user
- **Client-side mediation** – not enough to have client side input checks because the user could potentially interact with the server directly (turn off JavaScript, etc…)
- **Server-side mediation** – user-input check and make sure client has not modified data in state

TOCTTOU
- Time-Of-Check To Time-Of-Use
- When a resource is checked for a particular value and then that resource is changed before it is used
- So the file that was checked for writing privileges could be a different fie that gets written to
- **Defences**: when performing a privileged action on behalf of another party, make sure all the information relevant to the access control decision is constant

<u>INTENTIONAL</u> (malicious)
Malware
Software written with malicious intent
Needs to be <u>executed</u> to cause harm
- Virus
    - Adds itself to benign program/files (needs user activation)

    - Infection, spreading, payload (cause really bad stuff to happen), signatures, polymorphic (some viruses make modified copies of itself)
    - Behaviour based protection (limited to virus list) for viruses that are polymorphic, base rate fallacy (more false positives than true positives which may cause true ones to be overlooked)
- Worms
    - Malicious code spreading with no or little user involvement
    - Use security flaws in widely deployed software and then immediately starts searching for other victims to infect
    - There may or may not be a payload
    - Morris worm (1st, buffer overflow, backdoor, dictionary attack on passwords), Code Red worm (buffer overflow in Microsoft IIS web server), Slammer Worm (buffer overflow in SQL, infection through single UDP packet made it very quick), Stuxnet (targeted SCADA systems on Windows, USB installation), Flame (cyber espionage to collect sensitive information – very complex)
- Trojans
    - Malicious code hidden in seemingly innocent programs that are downloaded
    - Gains control by getting user to run code of the attacker's choice (by providing some code that the user wants to run)
        - PUP – potentially unwanted programs
    - **Scareware**, **ransomware**
- Logic bombs
    - Malicious code hidden in programs already on your machine
    - Written by insiders and are meant to be triggered sometime later in the future
    - Has pretty serious payload
- Web bugs
    - An object (1x1 pixel image) embedded in a web page which is fetched from a different server form the one that served the page itself
    - Used to send information about the user to third parties without consent

Back doors / trapdoor

A set of instructions designed to bypass the normal authentication mechanism and allow access to the system to anyone who knows that the backdoor exists

- Sources: testing, maintenance, legal reasons

Salami attacks

An attack made up of smaller, often considered inconsequential attacks

Privilege escalation

An attack in which the privilege level of the attacker is raised

- Occurs when part of systems with higher privilege is tricked into running commands
- <u>Vertical</u> (grant higher privileges by kernel alterations) vs <u>horizontal</u> (assume the identity of someone else to gain access)

Rootkit
Set of software tools that enable an attacker to gain unauthorized access and hide its own existence (or the presence of another application such as a virus)

- Stealth: acquired by cleaning log messages, modify ls or ps commands, modify kernel
- Sony XCP

Man-in-the-middle Attacks
The program you're communicating with isn't the one you think you're communicating with
Intercepts the communication from the user and then passes it on to the intended party

Keystroke logging
The user typing data is stored to keep record of all passwords typed and things sent.

- Kinds: Application-specific, system keyboard, hardware keyboard

Interface illusions

- We think we are doing one thing but really we are doing another (dragging a program into our computer instead of what we want)
- **Phishing**: the fraudulent practice of sending emails pretending to be from reputable companies in order to induce individuals to reveal personal information, such as passwords and credit card number

INTENTIONAL (non-malicious)

Covert channels
An attacker creates a capability to transfer sensitive information through a channel that is not supposed to transmit that information (by transferring objects using the structure of the existing medium to convey data in small parts)

Side channels
Sort of like a back door in which the attacker can gain information about the victim without the victim being aware of this (listening for sensitive information)

SOFTWARE LIFECYCLE

- Specification
- Design
  - Modularity (easier to check smaller pieces for flaws), Encapsulation (reducing sharing of information), information hiding (hide internal states from developers on other modules), mutual suspicion (checking input), confinement (limit untrustworthy external sources)
- Implementation
  - Don't use C, static code analysis, etc.…
- Version management (git)
- Code review
- Testing
- Documentation
- Maintenance

MODULE 3 – OPERATING SYSTEM SECURITY

An **operating system** allows different users to access different resources in a shared way
- Identification and Authentication required

Separation (for protected objects) - PTLC
- Physical (use different physical resources for different users)
- Temporal (execute at different times)
- Logical (impression that no other users exist)
- Cryptographic (encrypt data and make it unintelligible to outsiders)

- OS should allow **flexible sharing**
- Memory protection is part of translation from virtual to physical address

PROTECTION TECHNIQUES

Fence register
- Creates boundary between the operating system and the user programs by giving an exception if memory is accessed below the address in the fence register
- Single user OS only!

Base/bounds register pair
- Defines position for a segment in memory and gives an exception if memory is accessed below/above the address in those registers
- It is maintained by OS during context switch
- Has limited flexibility

Tagged architecture
- Each memory bit has one or more extra bits (tag) that identify access rights to that word (sort of describes the type of data)
- Very flexible
- Large overhead

Segmentation
- Each program has multiple address spaces (**segments** – for code, data, stack)

  < segment_name, offsite_within_segment >
- The OS maps the segment name to its base physical address in the Segment Table
- The OS transparently relocates/resizes segments and shares them between processes
- PROS:
  - Each address reference is checked for protection by hardware
  - Many different classes of data items can be assigned different levels of protection
  - Users can share access to a segment, with potentially different access rights
  - Users cannot access an unpermitted segment

- CONS:
    - External fragmentation (memory separated into blocks)
    - Dynamic length of segments requires costly out-of-bounds check for generated physical addresses
    - Segment names are difficult to implement efficiently

Paging

- Program (virtual address) is divided into equal-sized chunks (**pages**)
- Physical memory is divided into equal-sized chunks (**frames**)

  < page #, offset within page >

- OS maps page # to its base physical address in the Page Table
- PROS:
    - Each address reference is checked for protection by hardware
    - Users can share access to a page, with potentially different access rights
    - Users cannot access an unpermitted page
    - Unpopular pages can be moved to disk to free memory
- CONS:
    - Internal fragmentation (allocating more memory than needed and leaving it unused)
    - Assignment different levels of protection to different classes of data

X86 architecture

- Uses <u>segmentation</u> and <u>paging</u> (Windows and Linux)
- Memory protection bits: no access || read/write access || read-only access
- NX (no execute) bit for forbidding instructions in page

ACCESS CONTROL

<u>Goals:</u>

1. Check every access -> else OS might fail to notice that access has been revoked
2. Enforce least privilege -> grant access only to smallest number of objects required to perform task
3. Verify acceptable use -> limit types of activity that can be performed on ab object

Access control matrix:

- O (protected objects), S (subjects), R (rights)
- a[s, o] = R
- rarely a matrix -> set of control lists/capabilities (2D array)

Access Control Lists

- each object has list of subjects with their access rights (orwx)
- implemented in Windows File System (NTFS)
- classic UNIX file systems has simple ACLs

Capabilities

- A capability is an unforgeable token that gives its owner some access rights to an object
- Ex: { Alice: File 1:orw, File 2:rx, File 3:o }
- This unforgeability is enforced by having the OS sore and maintain tokens or by cryptographic mechanisms (digital signatures)

*We can also combine ACL and capabilities.

Role-based Access Control (RBAC)

- Users assigned rights depending on their roles
- **Hierarchical roles**: reduces number of role/access rights assignments
- **Separation of duty**: 'an order need to be signed by two different roles, but the people in those roles cannot be the same person'

USER AUTHENTICATION

Identification: who you are

Authentication: proof

Authentication factors:

- Something the user **knows**
- Something the user **has**
- Something the user **is**
- Something about the user's **context**

We can combine factors from different classes for a more solid authentication

Passwords

Attacks:

- Shoulder surfing
- Keystroke logging
- Interface illusion / phishing
- Password re-use across sites
- Password guessing
    - Brute force *offline*: attacker has little to no communication with the system
    - Brute force *online*: attacker has system working/participating -> detectable

Password file:

- Store only a digital fingerprint (cryptographic hash) of password
- When logging in, system computes fingerprint of entered password and compares it with user's stored fingerprint
- Will still allow offline attacks

Defending against guessing attacks:

- UNIX: includes user-specific salt (derived from time of date and process ID of /bin/passwd) in the password fingerprint -> makes guessing harder

- Shouldn't use standard cryptographic hash (SHA-1 or SHA-512)
- Instead use iterated hash function that is expensive to compute and maybe that uses a lot of memory
- Can also use MAC instead of cryptographic hash
    - Uses secret key to computer password fingerprint

Password recovery

- cannot be recovered from just a hash value, if that is necessary, need to store an encrypted version of the password in password file

The Adobe Password Hack

- November 2013 – 130 million encrypted passwords leaked

Interception attacks

- Attackers can intercept passwords while it is in transmission from client to server
- **Challenge-response protocol**
- There are cryptographic protocols (SRP) that make intercepted information useless to an attacker

* systems authenticate users with the help of a password, but users should also authenticate servers to make sure it is not an attacker (phishing)

Biometrics:

- Remove vs Local Authentication
- Biometrics only work on local (making sure things are not being faked)
- Problems: privacy, accuracy, and secrecy

SECURITY POLICIES AND MODELS

A trusted operating system builds on 4 factors:

1. **Policy**: a set of rules outlining what is secured and why
2. **Model**: a model that implements the policy and that can be used for reasoning about the policy
3. **Design**: a specification of how the OS implements the model
4. **Trust**: assurance that the OS is implemented according to design

Trusted software

We expect the software to do what it is expected to do and nothing more

- Functional correctness
- Enforcement of integrity
- Limited privilege
- Appropriate confidence level

Security Policies

- **Multilevel Security (MLS)** policies
    - Each object/subject has a security/clearance level
    - Each object/subject might also be assigned to one or more compartments

- Clark-Wilson security policy
- Well-informed transactions
- Conflicts of interest
- **Chinese wall security policy** -> once you have been granted access to information about a particular kind of company, you will no longer be able to access information about other companies of the same kind
    - **ss-property**: subject s can access object o iff each object previously accessed by s either belongs to the same company as o or belongs to a different kind of company than o does
        - ('no read up')
    - ***-property**: for a write access to o by s, we also need to ensure that all objects readable by s either belong to the same company as o or have been sanitized
        - ('no write down')

Bell-La Padula <u>Confidentiality</u> Model

- regulates information flow in MLS policies
- "information can only go up"

Biba <u>Integrity</u> Model

- prevents inappropriate modification of data
- 'no read down' – unreliable information cannot contaminate subject
- 'no write up' – unreliable person cannot modify file containing high integrity info
- "information flows down"