# Operating System Security

Asst Prof. Tianwei Zhang

12 November 2020

# Self-Introduction

## ❑ Experience

- 2019–present: Assistant Professor @ SCSE, NTU
- 2017–2019: Software Engineer @ Amazon Web Services
  - **Cloud Computing** and **Robotics** platforms
- 2011–2017: PhD @ EE, Princeton University
  - Thesis topic: **cloud computing security**

## ❑ Research interest

- Design and develop **secure** computer systems
- Principle: enhance system **security** without compromising **performance**, **cost** and **usability**.

# Questions Before Our Course

- If you download and run a video player app on your computer, can it delete your photos?
- If you install a maps app on your smartphone, can it steal your privacy information?
- If you click the attachment of an unknown email, can it destroy your data on the disk?

Yes, if your OS does not have appropriate security protection, e.g., *app isolation*, *permission management*

# Outline

- System security basis
  - Definition
  - Security properties
- Security Protection Stages employed by OS
  - Authentication
  - Access Control
  - Audit
- Security Threats in OS
- Hardware Protection
  - Integrity verification
  - Trusted Execution Environment

# Outline

- System security basis
  - Definition
  - Security properties
- Security Protection Stages employed by OS
  - Authentication
  - Access Control
  - Audit
- Security Threats in OS
- Hardware Protection
  - Integrity verification
  - Trusted Execution Environment

# OS Becomes more Complex

- From single-user to multi-user
  - DOS is truly single user
  - MacOS, Linux, NT-based Windows are multi-user, but typically only 1 user in PCs.
  - Cloud computing allows multiple users all over the world to run on the same OS, and they do not know each other.
  - Tradeoff: efficiency versus security
- From trusted apps to untrusted apps
  - Simple real-time systems: only run one specific app
  - Runs verified apps from trusted parties
  - Modern PCs and smartphones: run apps from third-party developers
  - Tradeoff: functionality versus security

# Complex OS brings More Challenges

- **Protecting a single computer with one user is easy**
  - Prevent everybody else from having access
  - Encrypt all data with a key only one person knows

- **Sharing resources safely is hard**
  - Preventing some people from reading private data (e.g. grades)
  - Prevent some people from using too many resources (e.g. disk space, CPU core)
  - Prevent some people from interfering with other programs (e.g. inserting keystrokes / modifying displays)

# OS Responsibility

**Functionalities**

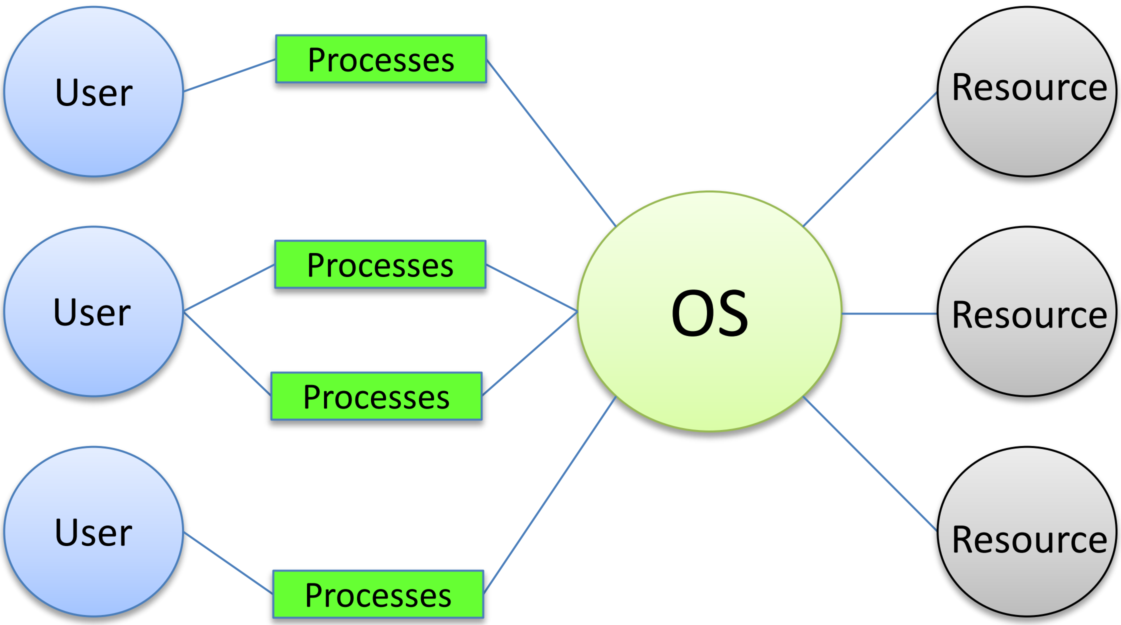
- Support multiple users concurrently
- Manage multiple apps concurrently
- Connect to the network
- Sharing data with different domains

**Security goals**

- Protect users from each other
- Protect apps from each other
- Protect the system from the network
- Secure the data sharing

# What's being protected? Resources

- System is secure if resources used and accessed as intended under all circumstances



# Security Properties

- Confidentiality (C)
  - Prevent unauthorized disclosure of information
  - Sensitive information should not be leaked to unauthorized parties
- Integrity (I)
  - Prevent unauthorized modification of information
  - Critical system state and code cannot be altered by malicious parties
- Availability (A)
  - Prevent unauthorized withholding of information or resources
  - The resources should be always available for authorized users
- Other properties
  - Accountability: actions of an entity can be traced and identified
  - Non-repudiation: unforgeable evidence that specific actions occur

# Outline

- System security basis
  - Definition
  - Security properties
- Security Protection Stages employed by OS
  - Authentication
  - Access Control
  - Audit
- Security Threats in OS
- Hardware Protection
  - Integrity verification
  - Trusted Execution Environment

# Security Protection from OS

- OS is responsible for protecting the apps running on it
  - OS controls what users/processes can do

# Authentication

- How does a computer know if I am a correct user?
  - What you know? password, PIN, public/private keys…
  - What you have? smartcard, hardware tokens…
  - Who you are? biometrics, face recognition, voice recognition…
- How does the system conduct authentication?
  - Compare the input credential with the stored one
    - Password file: **/etc/passwd** for UNIX
  - Allow entry when the credential matches
    - Assign the user an identifier: 32bit for UNIX

# Hash Function

## A one-way function f

- Takes an input $x$ of arbitrary length, and produces an output $f(x)$ of fixed length.

## Pre-image resistant

- Given an input $x$ it is easy to compute $f(x)$, but given an output $y$ it is hard to find $x$ so that $y = f(x)$

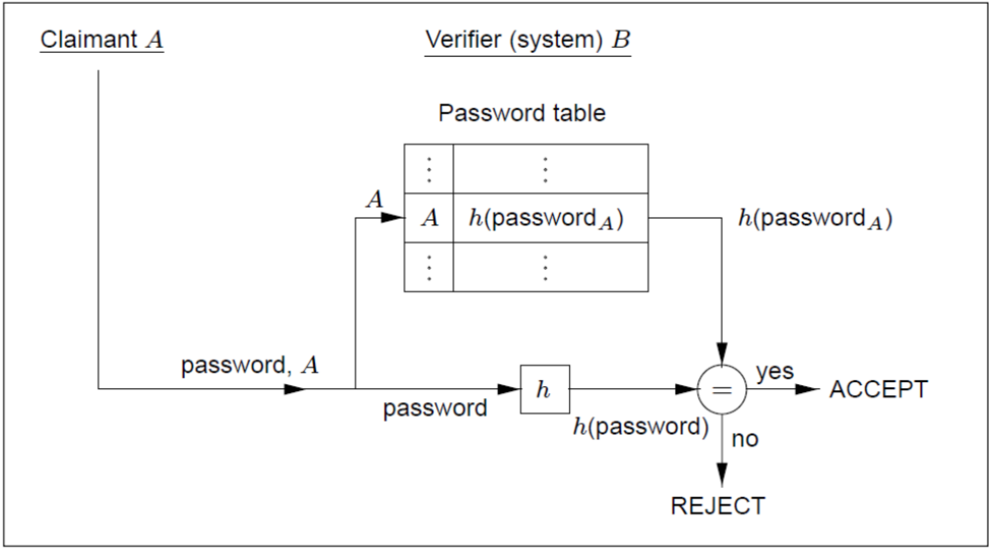
## Collision resistant

- It is computationally infeasible to find a pair $(y_1, y_2)$, such that $y_1 \neq y_2$ and $f(y_1) = f(y_2)$

# Password Storage

## ❑ Hashed passwords

- Passwords are hashed and stored in a password table
- When a user inputs a password, its hash value is computed and checked against the password table.



# Password Security

## ❑ Why hashed passwords?

- Insider attack: even the attacker can access the password table, he is not able to recover the password from the hash values.

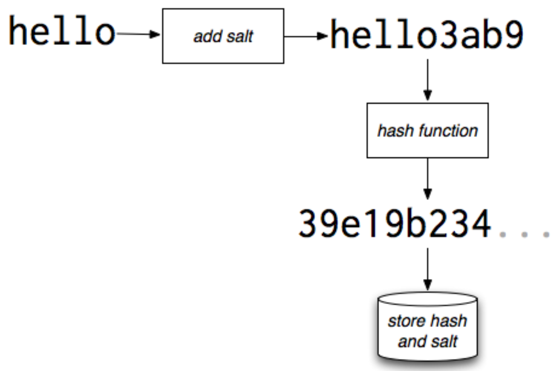
## ❑ Dictionary attack

- Hashed passwords, especially for human-generated passwords, are still vulnerable to dictionary attack.
- This exploits weakness in human-chosen passwords, which tend to derive from words in natural languages.
  - Guess some commonly used passwords
  - Compute their hash values
  - Look for the same hash values in the password table

# Preventing Dictionary Attack

## ❑ Password salting

- A salt is added to a password before applying the hash function
- A salt is a random string
- Each password has its own unique salt. So even the same password will have different hash values
- The salt value is stored along with the hash of the password + salt
- The attacker needs more hash computation to recover passwords

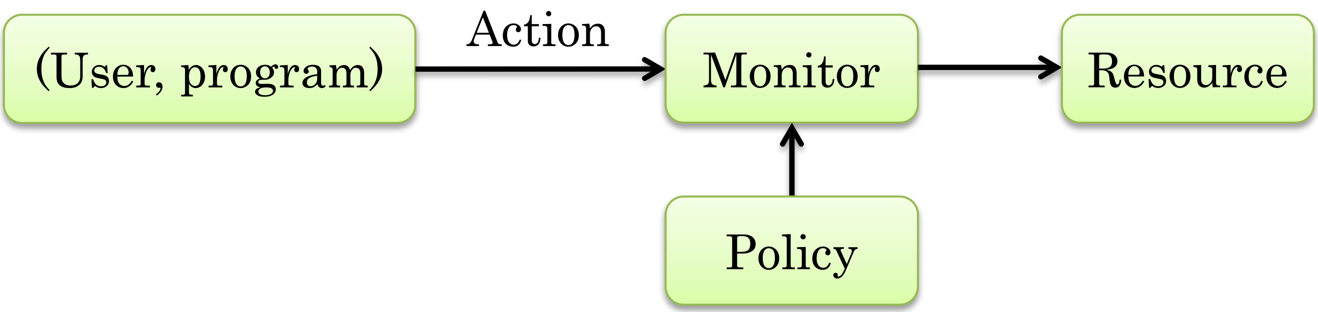


# Password Complexity

- Set up higher requirements for the password
  - Larger space (lower case, upper case, numbers, special symbols…)
  - Length
  - No consecutive repeated characters; Not in a dictionary
- Pros:
  - Increase the difficulty of password guessing attack
- Cons:
  - Hard to remember, and easy to type wrong
  - People may try to choose passwords that are easy to remember: attacker can guess them easily as well
  - People may reuse the old passwords
  - People write down the passwords.

# Access Control

- **Security policy**
  - Specifies (subject, verb, object) triples
  - Subject = (user, program) pair
  - Verb = action
  - Object = resources
- **Monitor**
  - Checks whether the action should be allowed



# Access Control Policy

- **Who sets policy?**
  - Users, with some system restrictions

- **How is access control list stored?**
  - Sparse matrix (default deny), store as list

- **How is policy enforced?**
  - OS exposes API to apps, with privileged operations
  - Checks ACL when API functions are called

# Authorization

## ❑ Access Control Matrix

- Each column represents an object
- Each row represents a subject
- The entry shows the allowed verbs.

| | /etc | /homes | /usr |
|---|---|---|---|
| Alice | Read | Read | Read<br>Write |
| Bob | Read<br>Write | Read<br>Write | Read<br>Write |
| Carl | None | None | Read |

# Update Access Matrix

## ❑ Access Control Changes

- Grant capabilities: the owner of the object can grant rights to other users.
- Revoke capabilities: subjects can revoke the rights from others

## ❑ Six Commands to Alter the Access Matrix

- **create subject *s***: creates a new subject s.
- **create object *o***: creates a new object o.
- **enter *r* into *Ms,o***: adds right $r$ to cell $Ms,o$.
- **delete *r* from *Ms,o***: deletes right $r$ from cell $Ms,o$.
- **destroy subject *s***: deletes subject s. The column and row for s in $M$ are also deleted.
- **destroy object *o***: deletes object $o$. The column for $o$ in $M$ is also deleted.

# More Representations

- Access Control List (ACLs)
  - For one object, which subject has accesses to it? (check the column in the Access Matrix)
- Capability:
  - For one subject, which objects it has capability to access? (check rows in the Access Matrix)
- Most systems use both
  - ACLs for opening an object (e.g. fopen())
  - Capabilities for performing operations (e.g. read())

# Data Sharing

- ❑ Problem: multiple users want to access the same file or data
  - Give each user the corresponding permissions.
  - When a new user joins, the permissions have to be granted again.
  - When permissions are changed, need to alter each user.

- ❑ Solution: group
  - Set permissions for the group instead of the user
  - A user joining the group will have the corresponding permissions.
  - A user quitting the group will loss the corresponding permissions.
  - Easier to manage and update.

# Audit Logs

## ❑ Audit trail

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

| | |
|---|---|
| **/usr/adm/lastlog** | Records the last time a user has logged in; displayed with **finger** |
| **/var/adm/utmp** | Records accounting information used by the **who** command. |
| **/var/adm/wtmp** | Records every time a user logs in or logs out; displayed with the **last** command. |
| **/var/adm/acct** | Records all executed commands; displayed with **lastcomm** |
| **/var/log/** | In modern Linux systems, log files are located in there |

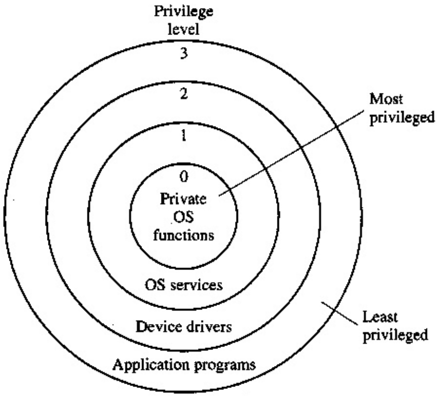
# Outline

- System security basis
  - Definition
  - Security properties
- Security Protection Stages employed by OS
  - Authentication
  - Access Control
  - Audit
- **Security Threats in OS**
- Hardware Protection
  - Integrity verification
  - Trusted Execution Environment

# Privileged Rings

## ❑ Operating modes

- Kernel mode has the highest privileges, running the critical functions and services
- Entities with the higher privilege levels cannot call the functions and access the objects in the lower privilege levels directly.
  - System call, interrupt, etc.
- Status flag allows system to work in different modes (context switching)



# Malware

- Software code that maliciously subvert the computer system
  - Virus: malicious program that causes copies of itself to be created when trigged by the user
  - Worms: malicious program that causes copies of itself to be created without any user intervention
  - Trojan horses: appears to do something useful, but masks some hidden malicious activities
  - Rootkit: hides other malware from detection and maintains root-level access to the computer.
  - Backdoor: allow a remote party to gain access to the computer
  - Bot: inserted into a computer and lies dormant until invoked by remotely to perform a function
  - Spyware: inserted into the computer to capture users' data
  - Ransomware: locks up data via encryption, demanding payment to unlock it

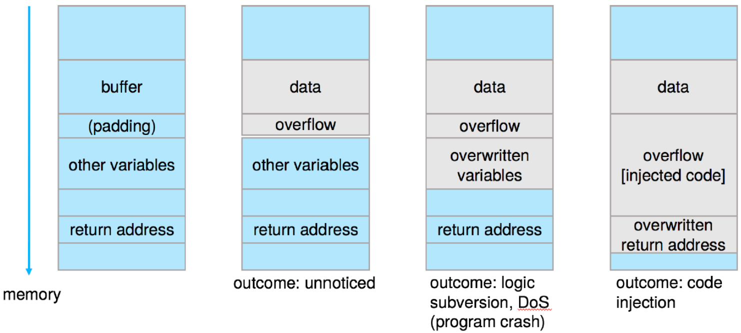
# How to Inject Malware?

## ❑ Code-injection attack

- The system code is not malicious but has bugs allowing executable code to be added or modified

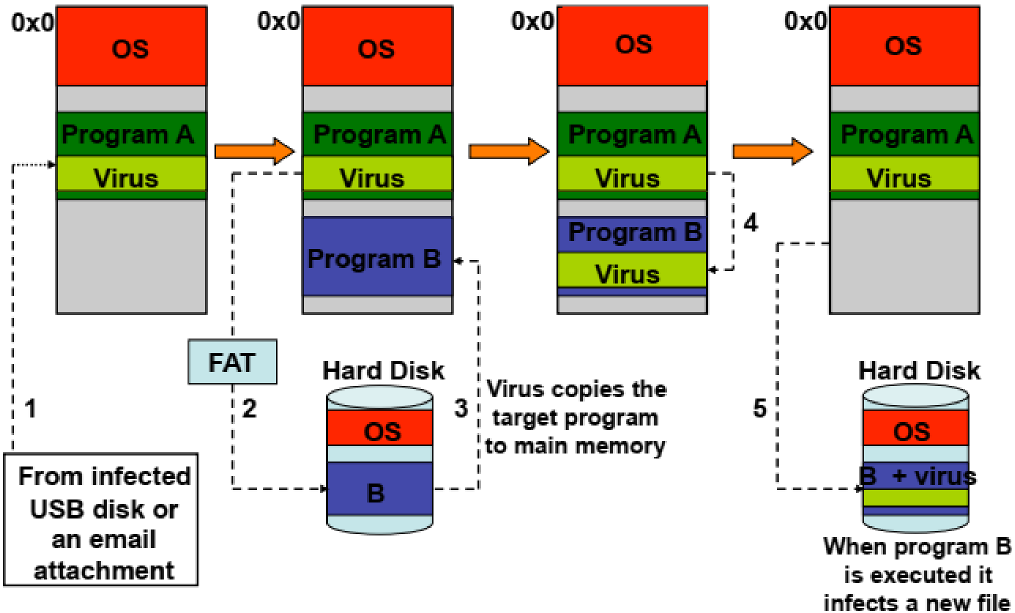
## ❑ Buffer overflow

```
#include <stdio.h>
#define BUFFER SIZE 256
int main(int argc, char *argv[])
{
    char buffer[BUFFER SIZE];
    if (argc < 2)
          return -1;
    else {
          strcpy(buffer,argv[1]);
          return 0;
    }
}
```



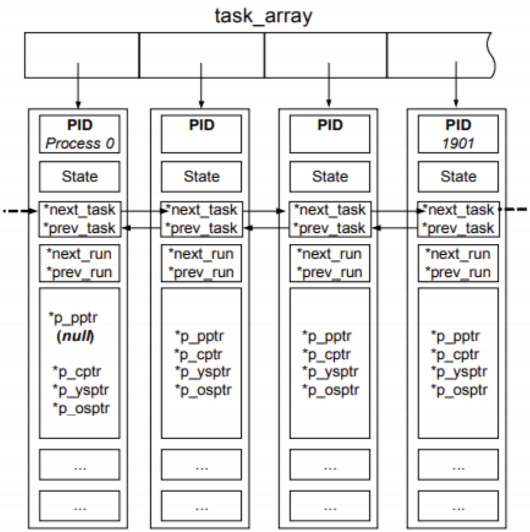
# How to Propagate Malware?

- Virus tries to copy itself to other programs and propagate it to other systems



# How to Hide Malware?

- Rootkits hook the function called by the anti-malware, and remove their existences
  - Windows:
    - NtOpenProcess
    - NtQuerySystemInformation
    - PsActiveProcessLinkHead
  - Linux:
    - Proc filesystem (procfs)
    - task_struct



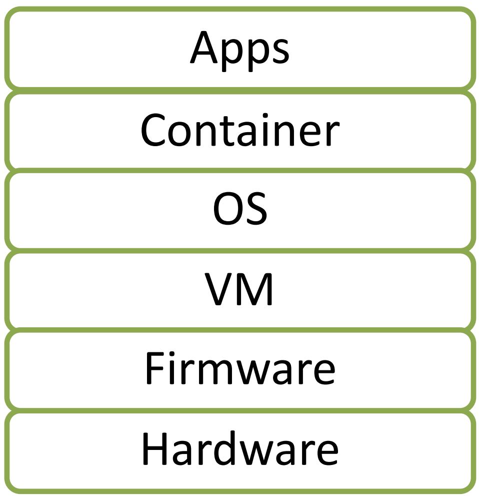
# Outline

- System security basis
  - Definition
  - Security properties
- Security Protection Stages employed by OS
  - Authentication
  - Access Control
  - Audit
- Security Threats in OS
- Hardware Protection
  - Integrity verification
  - Trusted Execution Environment

# Computer System: A Hierarchic View

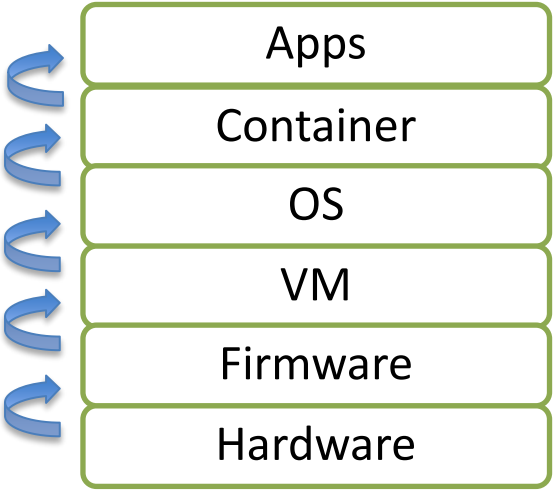
## ❑ System layers

- Different scenarios may have different layers
- Lower layers have higher privileges and can protect higher layers.
- Lower layers need to be better protected



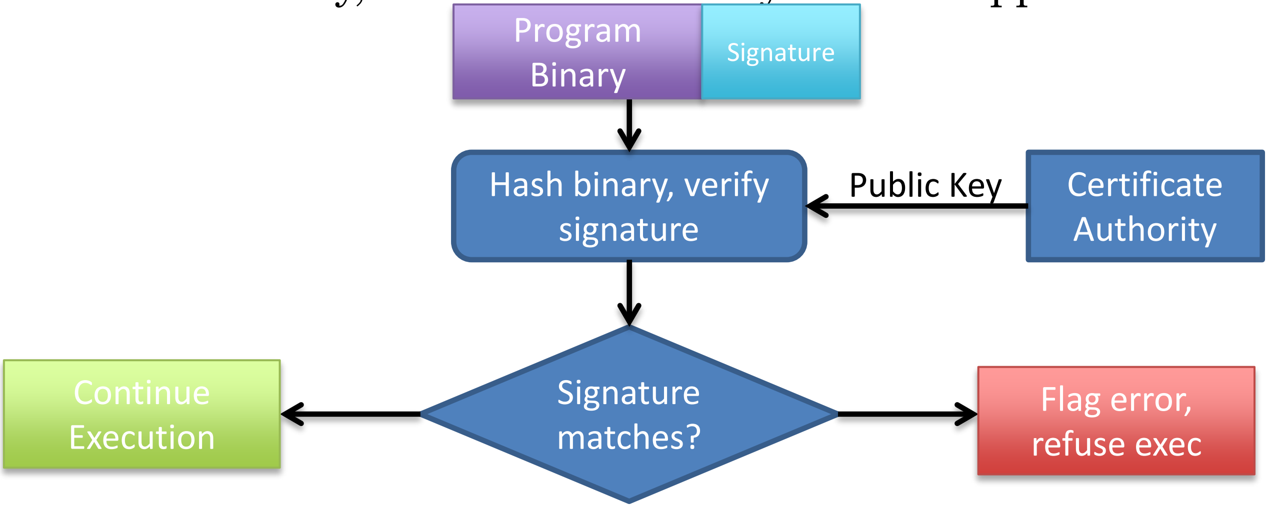
# Chains of Trust

- Establish verified systems from bottom to top
  - The bottom layers validate the integrity of the top layers
  - If the verification passes, then it is safe to launch it.
  - Each layer is vulnerable to attack from below if the lower layers are not secured appropriately



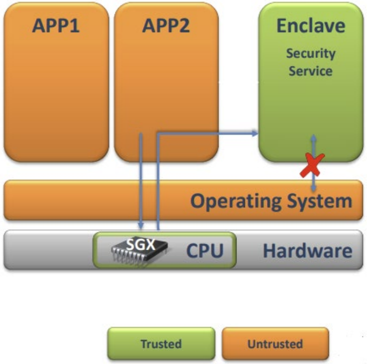
# Integrity Verification

- Only execute code signed by an entity we trust
  - Load the bootloader in the firmware
  - Reads and verifies the kernel
  - Only loads kernel if the signature is verified
  - Similarly, kernel can run only verified applications



# Protect Applications from Untrusted OS

- Building a secure OS is difficult
  - Large code base size and complex functionalities
  - An untrusted OS can compromise all applications
  - Can we protect the security of apps even when the OS is malicious?
- Solution: Intel Secure Guard Extension (SGX)
  - Security critical code isolated in enclave
  - Only CPU is trusted
  - Memory is encrypted
  - Support remote attestation



# Thank You!

tianwei.zhang@ntu.edu.sg