Contents

Contents

Lecture 2 – Authentication

Summary and takeaways

• Authenticity: data, communication.
• Password strength
  - Online vs offline. Way to calculate password strength.
• Attacks on password system.
  - Phishing. Bootstrap. Default password. *Phishing* is very effective.
• n-factor.
  - Know how 2-factor is better than 1-factor, especially in online banking.
  - Concrete example of how 2-factor can still fail.

2.1 Overview

Authentication: The process of assuring that the communicating entity, or origin of a piece of information, is the one that it claims to be.
Authentic: The claimed origin/entity is assured by supporting evidences
Authenticity: Condition of being authentic

***Authenticity implies integrity**

Entity Authentication

- For connection-oriented communication
- Verifying authenticity of entities involved in a connection
- Mechanisms: password, challenge and response, cryptographic protocol

Data-Origin Authentication

- For connectionless communication
- Verifying the origin of a piece of information
- Mechanisms: Crypto primitives such as MAC or digital signature

2.2 Password System

1. Bootstrapping

Server and an user established a common password. The server keeps a file recording the identity (aka userid, username) and the corresponding password.

The password is to be establish during **bootstrapping**.

- The server/user chooses a password and sends it to the user/server through another communication channel
- Default password

2. Password Authentication

The server authenticates an entity. If the entity gives the correct password corresponds to the claimed identity, the entity is deemd as authentic.

User → Server: My name is *Alice*
Server → User: What is your password?
User → Server: *OpenSesame*
Server → User: Access Granted

Alternatively, authentication can be carried without interactions.

Example: User sends a sms to a server: *Userid: Alice. Password OpenSesame. Instructions: Unsubscribe*

Weak authentication system and Replay attack

Password system is classified as a "**weak authentication**". A **weak authentication** is one that can be subjected to the simple "**replay attack**"**:** Information sniffed from the communicated channel can be replayed to impersonate the users.

(Under "strong authentication", information sniffed during the process can't be used to impersonate the user)

Terminologies

Sniffing is a process of monitoring and capturing all data packets passing through given network.
Spoofing happens when someone uses deception to appear as another person or source of information.

3. Password Reset

There are many reasons to reset a password. User forgets the password, regular changes as required by password policy, of if a user feels that current password has been stolen.

Resetting password is tricky. How to verify that the entity is authentic? What if the user forgets the password?

Methods to change password:

- Anyone who know the old password can change the password.
- Entity present additional document/information to change the password.

Note: Password is a secret, only the authentic user and the server knows it. The identity does not necessarily be kept secret. If an entity knows the password, it implies the entity is either the server or the authentic user.

2.2.1. Attack the Bootstrapping

Attacker may intercept the password during bootstrapping.

- Example: If the password is sent through postal mail, an attacker could steal the mail to get the password.

- Example: Attacker uses the “default” password (e.g. IP camera, Wifi router). *Mirai* malware used a vulnerability involving weak default passwords.

2.2.2 Searching for the Password

2.1 Guessing the password from social information

- The attacker gathers some social information about the user, and infers the password.
  - E.g. mobile number

2.2 Dictionary attacks

- The attacker tries different passwords during login sessions. The attacker can employ exhaustive search, i.e. tries all combinations.
- The attacker can restrict the search space to a large collection of probable passwords. The collection can include words from English dictionary, known compromised passwords, etc. This is known as **Dictionary attack**
- It is possible to carry out exhaustive search together with dictionary attack. E.g. try all combination of 2 words from the dictionary, exhaustively try all possible captilization of each word, substituting “a” by “@”, etc

Two scenarios in dictionary attacks (there is a crucial implication on the defense mechanism)

Online dictionary attack: an attacker must interact with the authentication system during the searching process. In other words, attacker must be online. (e.g. choose a password and ask the system (oracle whether it is authentic)

Offline dictionary attack: There are two phases.

1. The attacker obtains some information ***D*** about the password from the authentication system, possibly via some interactions. (e.g. steal the password file, or sniffs from interactions)
2. Next, the attacker carries out searching using ***D*** without interacting with the system.

2.2.3. Stealing the Password

1. Sniffing

- **Shoulder surfing:** Look-over-the-shoulder attack.
- **Sniffing** the communication: Some system simply send the password over public network in clear (not encrypted). E.g. (FTP, Telnet, HTTP)

❖ In computer security, a **side-channel attack** is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself.

Virus, Keylogger

A key-logger captures the keystrokes and sends the information back to the attacker via a "**covert channel**".

- Software: Some computer viruses are designed as a **key-logger**
- Hardware: Keyboard hardware, etc.

❖ In computer security, a **covert channel** is a type of attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy.

2. Phishing

Same as login spoofing. The user is tricked to voluntarily send the password to the attacker. Phising attacks ask for password under some false pretense. Phishing attack is a **social engineering** attack.

❖ **Social engineering**, in the context of information security, refers to psychological manipulation of people into performing actions or divulging confidential information.

3. Spear Phishing

Phishing can be targeted to a particular small group of users. Such attack is generally known as **spear phishing** which is an example of **targeted attack. (Spear-phishing is extremely effective)** Phishing of passwords is typically done through fake/spoofed website. They can also be carried out over phone calls.

"Spear phishing is the number one infection vector employed by 71 percent of organized groups in 2017." *Internet Security Threat Report*, Symantec, Vol 23, 2018.

Definitions

❖ **Phishing** is a type of social engineering where an attacker sends a fraudulent message designed to trick a person into revealing sensitive information to the attacker or to deploy malicious software on the victim's infrastructure like ransomware.

❖ **Pharming** is a two-step process that begins with an attacker installing malicious code on a victim's computer or server. That code sends the victim to a spoofed website, where they may be tricked into offering their personal data or login credentials for a website or online service.

❖ **Vishing**, is the use of telephony to conduct phishing attacks.

❖ ***Smishing*** is a phishing cybersecurity attack carried out over mobile text messaging, also known as SMS phishing.

4. Cache

When using a shared workstation (e.g. browser in airport), information keyed in could be cached. The next user can access the cache. (Make sure to close browser when using shared workstation)

5. Insider Attack

The malicious system administrator steals the password file. The system administrator's account is compromised (e.g. pw stolen via phising), leading to lost of password file.

2.2.4 Phishing Prevention

- User Education
- Embedded Phishing Exercise:
    - Authorized entities send out "phishing emails to employees". Effectiveness of such exercise not well studied.
- Phishing repository site:
    - Example: phishtank.com (submit suspected phishes, track status of submissions)
    - Organization actively monitors for phishing site. Take action (push the discovered site to the repository, bring down the website)
- Blocking by mechanism in the browser or firewall (with false positive and negative)
- Neat and clean design of workflow

Password

Random: A password is chosen randomly among all possible keys using an automated password generator. High "entropy" but difficult to remember.

User selection: Mnemonic methods, altered passphrases, combining and altering word

Usability: Strong passwords are difficult to remember. It is difficult to enter alphanumeric passwords into mobile devices. There are alternatives (graphical or gesture-based)

Password Strength

RFC4086 – Password should have atleast **29bits of entropy** to be secure against **online attacks**
NITS Recommendation – Password should have atleast **128bits of entropy** to be secure against **offline attacks**

Entropy per symbol $H = (\log_2 N)$ bits where N = symbol count

Symbol set	Symbol count N	Entropy per symbol H
Arabic numerals (0–9) (e.g. PIN)	10	3.322 bits
Hexadecimal numerals (0–9, A–F) (e.g. WEP keys)	16	4.000 bits
Case insensitive Latin alphabet (a–z or A–Z)	26	4.700 bits
Case insensitive alphanumeric (a–z or A–Z, 0–9)	36	5.170 bits
Case sensitive Latin alphabet (a–z, A–Z)	52	5.700 bits
Case sensitive alphanumeric (a–z, A–Z, 0–9)	62	5.954 bits
All ASCII printable characters except space	94	6.555 bits
All Latin-1 Supplement characters	94	6.555 bits
All ASCII printable characters	95	6.570 bits
All extended ASCII printable characters	218	7.768 bits
Binary (0–255 or 8 bits or 1 byte)	256	8.000 bits
Diceware word list	7776	12.925 bits per word

- Note that typical human generated password of 10 alphanumeric characters do not have entrophy of $10 \log_2(36)$. It is much less than that.
- Recommendation by RFC4086 suggests the password to have atleast 29 bits of entropy to be secure against **online attacks**.
- If cryptographic keys are to be generated from the password, and **offline attacks** are possible, then the password should have at least 128 bits of entropy, based on NITS recommendation of 128 bits for crypto keys.

Online vs Offline Attack

Online: To check whether a password is correct, the attacker needs to communicate with a server not under his control.

- The attacker obtained a list of 1000 valid nusnet IDs. The attacker writes a script that attempts to login to LumiNUS using guessed password for each of these id.
- Attacker obtained the list of 1000 valid nusnet id. The attack script attempts to login to NUS wifi router

Offline: To check whether a password is correct, the attacker can execute some algorithm without connecting to a server.

- Attacker has an AES encrypted file. The key is derived from a password. The attacker wants to find the password.
- In some PW authentication protocol, a "hash" of the PW is sent in clear. The attacker obtains the hash by eavesdropping a valid login session. The attacker then goes offline and search for the password.

Password Entropy

We often encounter this term "entropy" when quantifying strength of password. Entropy is a measurement of randomness.

Suppose a set ***P*** contains ***N*** unique passwords. Alice chooses her passwords by randomly & uniformly picking a password from the set ***P.*** Every password in ***P*** has an equal chance to be chosen (i.e. 1/N). In this case, by definition, the entropy of Alice's password is: $(\log_2 N)\ bits$

What if Alice doesn't choose the passwords uniformly, for e.g., the probability that she picks a word starting with letter "A" is higher than the probability that she picks a word starting with "z"? In such case, the entropy is not ($\log_2$ N). By the definition of entropy, it is

$$-\sum_{i=1}^{N} p_i \log_2 p_i$$

It can be shown that, for the entropy to be highest for a set of *N* items, p_i must be 1/*N*. In other words, uniform choices. So, to increase entropy, we can either make more uniform choices, or increase the size *N*.

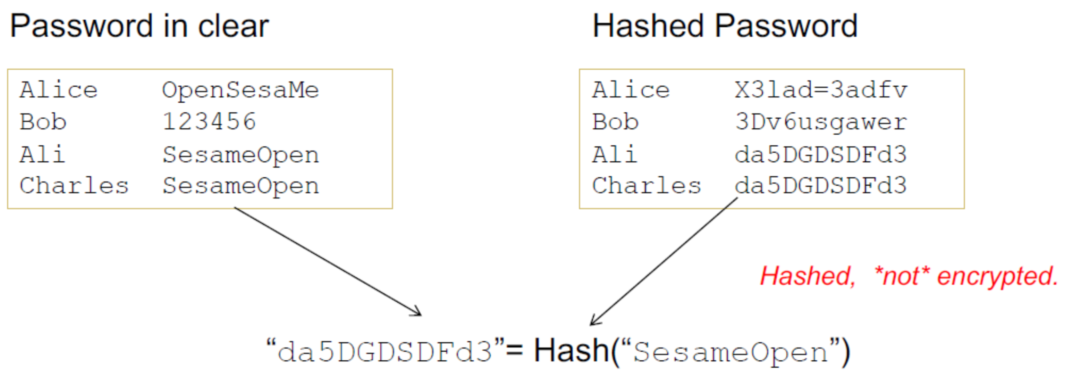
Password Policy

- To make online dictionary attack more difficult, many systems intentionally add delay into login session (have to wait 1 second before next attempt), or lock the account after a few failed attempts.
- System checks for weak password when user register/changes password. (e.g. use password dictionary)
- System require regular changes of passwords, which is controversal. (Many believe frequent changes of PW could lower security)

Password policy: Rules set by the organization to ensure that users use strong password, and to minimise lost of passwords.

- The password file stores the userid and password.
- The file could be leaked, due to insider attack, accidental leakage, system being hacked, etc.
- Many well-known incidents where unprotected or weakly protected password files are leaked, leading to a large number of passwords being compromised (2012 Linkedin)
- It is desired to add an additional layer of protection to the password file. (not all files are equally important)

Passwords should be "hashed" and stored in the password files. During authentication, the password entered by the entity is being hashed, and compared with the the value stored in the password file.



Alice	OpenSesaMe
Bob	123456
Ali	SesameOpen
Charles	SesameOpen

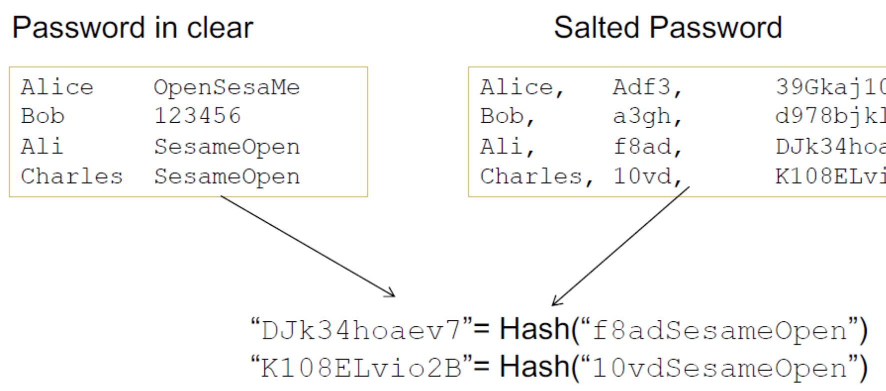
Alice	X3lad=3adfv
Bob	3Dv6usgawer
Ali	da5DGDSDFd3
Charles	da5DGDSDFd3

"da5DGDSDFd3"= Hash("SesameOpen")

To verify whether a password ***P*** belongs to a user ***U***, the following are carried out:

1. Compute *d* = Hash (***P***).
2. If <***U***, *d*> is in the password file, then accept, else reject.

It is desired that the same password would be hashed to two different values for two different userid. Why? (*rainbow table*). This can be achieved using salt.



Alice	OpenSesaMe
Bob	123456
Ali	SesameOpen
Charles	SesameOpen

Alice,	Adf3,	39Gkaj10Dmf
Bob,	a3gh,	d978bjklDFD
Ali,	f8ad,	DJk34hoaev7
Charles,	10vd,	K108ELvio2B

"DJk34hoaev7"= Hash("f8adSesameOpen")
"K108ELvio2B"= Hash("10vdSesameOpen")

2.2.5 Self-help Password Reset

1. Password reset using Security Questions

Security questions can be viewed as a mechanism for **fallback authentication,** or a **self-service password reset.**

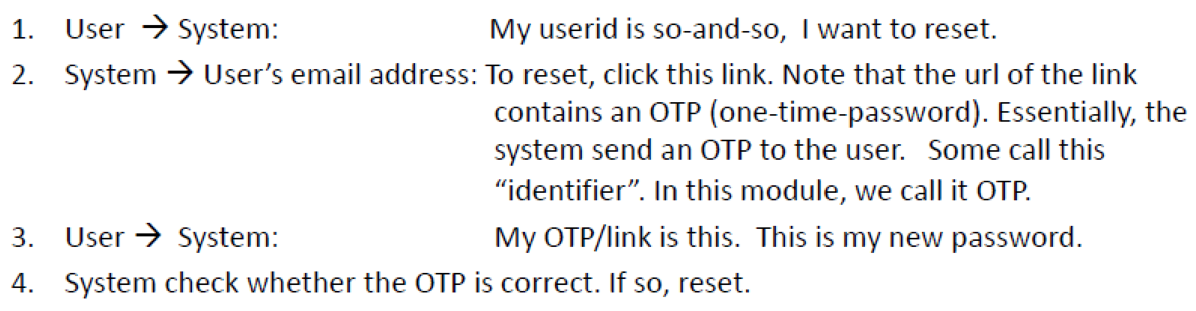
- Enhance **usability:** a user can still login even if password is lost
- Reduce **cost:** reduce operating cost of helpdesk
- Weaken **security:** attackers have another mean to obtain access

Choices of Security Questions

- **Memorable:** If users can't remember their answers, you have achieved nothing
- **Consistent:** The user's answers should not change over time. E.g. (What is the name of your SO?) may have a different answer 5 years from now.
- **Nearly Universal:** The security questions should apply to a wide audience if possible
- **Safe:** The answers to the questions should not be easily guessed or researched (e.g. something that is matter of public record)

2. Password reset using recovery email's account

Many system link an account to a recovery email.



1. User → System: My userid is so-and-so, I want to reset.
2. System → User's email address: To reset, click this link. Note that the url of the link contains an OTP (one-time-password). Essentially, the system send an OTP to the user. Some call this "identifier". In this module, we call it OTP.
3. User → System: My OTP/link is this. This is my new password.
4. System check whether the OTP is correct. If so, reset.

- Ownership of the email address proves that the entity is authentic (viewed as 2FA authentication)
- From attacker's point of view, try to steal the "OTP".
- The protocol can be deployed for account registration. (Change step 1 to "I want to register and account it to this email address "userid")

Social engineering + password reset

The XXXX below is frictional. Nonetheless, it would not be surprising to see variants of the following attack in actual systems.

• Suppose an attacker already knew Bob's password of a social media platform XXXXX . Bob was in a private chat-group ABC in XXXXX.

• The attacker (using Bob's account) posted in the group ABC, mentioning that he received a phishing email who claimed to be from XXXXX, and the email showed an QR code. Bob posted the email with the QR code, with some ha, ha and lol.

• Attacker went to XXXXX's site and attempted to reset ABC members' accounts, prompting XXXXX to automatically send confirmation emails to the members. The emails contained some info embedded in QRcode and asked the members to scan the QRcode using XXXXX's apps.

• A member in ABC replied to Bob's post "I also received this" and posted the QRcode.

• Attacker used the QRcode to confirm the password change.

2.2.6 Example on ATM

ATM Card

To get authenticated, the user presents (1) a card, and (2) a PIN.

- The card contains a magnetic strip, which stores the user account id. Essentially, the magnetic strip simplifies the input of account id into the ATM system: instead of keying it in, just inserting the card.

- The PIN plays the role of password.

Data are encoded into the magnetic strip using well-known standards. Given access to a card, anyone (including attackers) can "copy" the card by reading the info from the card and writing it to the spoofed card.

ATM Skimmer

An ATM skimmer steals the victim's account id (username) and PIN (password).

The skimmer consists of:

1. a card-reader attached on top of existing ATM reader;
2. a camera overlooking the keypad, or a spoofed key-pad on top of existing keypad;
3. some means to record and transmit the information back to the attacker.

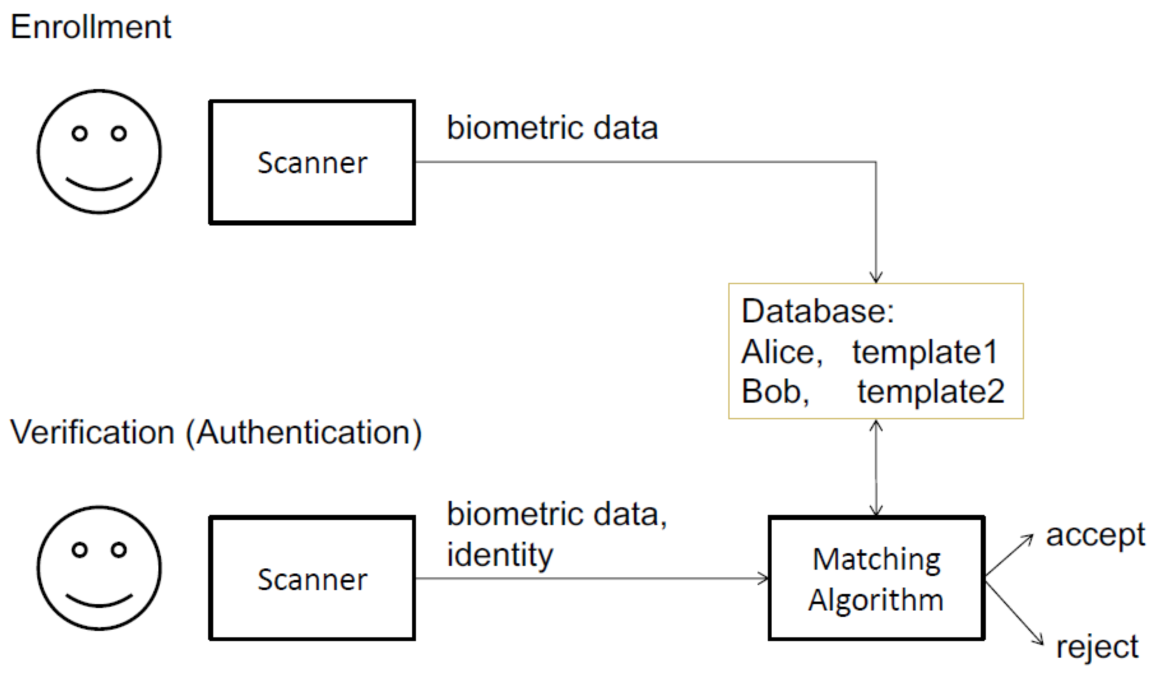
With the information obtained from (1), the attacker can spoof the victim's ATM card. With (2), the attacker obtain the PIN.

Well known incidents in Singapore: DBS in 2012. "$1 million stolen from the bank accounts of 700 DBS and POSB customers."

Measures

- Anti-Skimmer device: A device that prevents external card reader to be attached onto the ATM.
- Shielding the keypad.
- Awareness among users.

2.3 Biometric



Biometric uses unique physical characteristics of a person for authentication. During **enrolment,** a **template** of a user's biometric data is captured and stored (same as **bootstrapping** in password system).

During **verification,** biometric data of the person is captured and compared with the template using a matching algorithm. The algorithm decides whether to accept or reject.

Biometric can be used for ***identification*** (identify the person from a database of many persons), or ***verification*** (verify whether the person is the claimed person). Here, we focus on verification.

Differences between Biometric and Password

Password	Biometric
Can be changed (revoked)	Can't
Need to remember	Don't have to
Zero non-matched rate	*Probability of error*
Users can pass the password to another person	Not possible

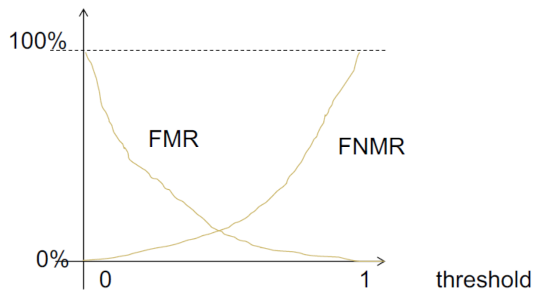
Unlike password, there are inevitable noise in capturing the biometric data, leading to error in making the matching decision. FMR (False match rate) and FNMR (False non-match rate)

$$\text{FMR} = \frac{\text{number of successful false matches (B)}}{\text{number of attempted false matches (B+D)}}$$

$$\text{FNMR} = \frac{\text{number of rejected genuine matches (C)}}{\text{number of attempted genuine matches (A+C)}}$$

	accept	reject
genuine attempt	A	C
false attempt	B	D

The matching algorithm typically makes decision based on some adjustable threshold. (Lower threshold => more relax in accepting, higher threshold => more stringent in accepting).

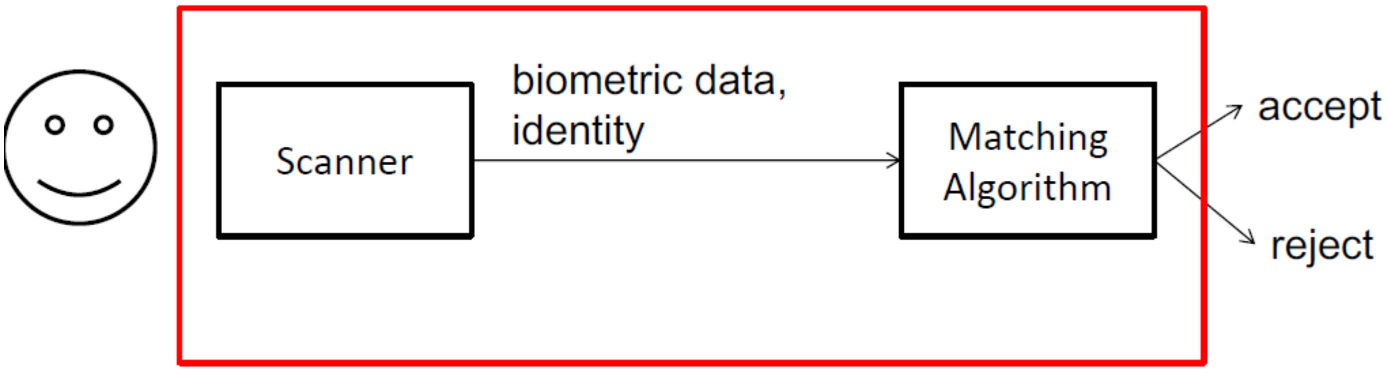


Other type of errors:

- **Equal error rate (EER):** Rate when FNMR = FMR
- **False-to-enroll rate (FER):** Some user's biometric data cant be captured (e.g. injury)
- **Failure-to-capture rate (FTC):** An user's biometric data may fail to be captured during authentication (e.g. fingers too dry, dirty)

How good is fingerprint as a biometric?
Performance depends on the quality of the scanner. EER can range from 0.5 to 5% depending on quality of scanners.

Security of biometric system



The scanner must be secure, that is no tampering of the scanner is possible. (Mobile phones have some hardware+crypto protection to secure the scanner, so as to prevent the attacker, who has access to the phone, in bypassing or tempering the scanner).

Some biometric systems include ***liveness detection*** to verify that the entity scanned by the scanner is indeed "live", instead of spoofed materials, say a photograph. (example, temperature scanner in fingerprint scanner)

2.4 n-Factor Authentication (2FA)

Require **at least two** different authentication "factors".

- Something you **know**:	Password, Pin
- Something you **have**:	Security token, smart card, phone, ATM card
- **Who** you are:	Biometric

It is called 2FA authentication if 2 factors are employed.

*MAS (Monetary Authority of Singapore) expects all banks in Singapore to provide 2-factor authentication for e-banking.

Something you have

Example: ATM Card, Mobile Phone, OTP Token

One Time Password Token: A hardware that generates one time password (that can only be used once). Each token and the server share some secrets.

- **Time-Based**: Based on the shared secret and current time interval, a password K is generated. Now both server and user has a common password K.
- **Sequence-based**: An event (user pressing button) triggers the change of the password

Example of 2FA: Password + Mobile phone(SMS)
Registration: User gives the server his mobile phone number and password.

Authentication:
- User sends PW and userid to server
- Server verifies that PW is correct. Server sends a OTP to user through SMS
- User receives the SMS and enters the OTP
- Server verifies that the OTP is correct

What you know: Password
What you have: The SIM card in the mobile phone

- There are more security enhancement than "what you have, what you know" in previous example.
- Password is long term and can be the same for many months. OTP is only valid for a short period of time, or specific to a transaction.
- After an attacker stole a password, it can be kept for further use, or even traded in darkweb. Not for OTP. This makes OTP+password a huge improvement over password-only authentication since the attacker need to immediately carry out the attack after OTP is stolen.

Example of 2FA (2): Password + OTP Soft token.
Mobile phone can take the role of "hardware token". This is also known as "Soft Token".

Registration:

1. User installs the authentic Soft Token apps. During installation, some form of verification is carried out. After verification, a "secret key" ***k*** is established between the server and the mobile phone.

2. Separately, user registers a password with the server.

Authentication:

(1) The Soft token app establishes connection to the server, using the secret key ***k*** for authentication.

(2) User via another apps or browser, send request for a transaction T to the server. The apps/browser asked for user password. (in many cases, the password are stored in the app/browser, and the app/browser submits the password on behalf of the user.)

(3) The server contact the Soft-Token. The soft token app display the transaction T, and ask the user, are you sure? If user click yes, send confirmation to server.

(4) After received confirmation from Soft-Token, carry out the transaction. Likewise, password is what-you-know. Ownership of the software token (which is the secret key k) is what-you-have.

Likewise, password is what-you-know. Ownership of the software token (which is the secret key k) is what-you-have.

Example of 2FA (3): smartcard + fingerprint (Door access system)

Registration:

The server issues a smartcard to the user (note that the smartcard contains a secret key K). The user enrolls his/her fingerprint.

Authentication:

(1) User insert smartcard to the reader. The reader obtains the user identity and verifies whether the smartcard is authentic. If so, continue.

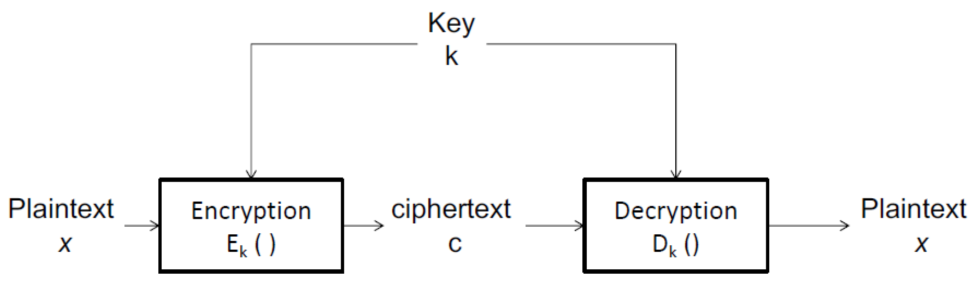
(2) User presents fingerprint to the reader. The reader performs matching to verify that it is authentic. If so, open door.

OCBC 2021: Password+SMS Authentication

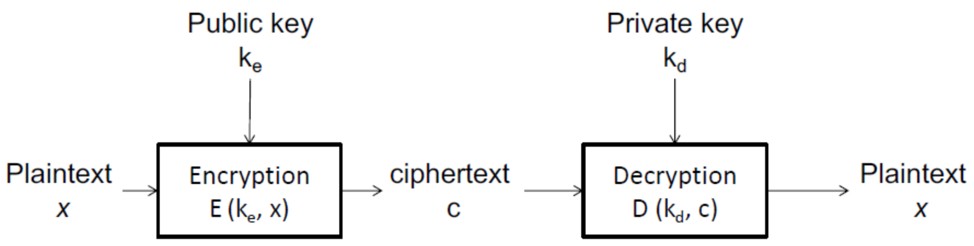
Lecture 3: Authenticity (Data Origin: Mac & Signature)

3.1 Crypto Primitive: Public Key Encryption

Symmetric-key Encryption Scheme: same key is used for encryption and decryption



(Public Key) Asymmetric-key Encryption Scheme: different keys used for encryption and decryption



- The owner Alice keeps the private key as a secret but tells everyone the public key.
- An entity Bob, hash a plaintext for Alice. Bob can encrypt it using the public key and send the ciphertext publicly.
- Another entity Eve obtains the public key and ciphertext. Without the private key, Eve is unable to derive the plaintext.
- Alice with the private key can decrypt and obtain the plaintext.

Security Requirement: Given the public key and ciphertext (but not the private key), it is difficult to determine the plaintext. The requirement implies that it <u>must be difficult to get the private key from the public key</u>.

Encryption Oracle: Anyone can encrypt. So the "encryption oracle" is always available to anyone. So, chosen plaintext attack is a must to be considered.

Advantages in Key Management: Only requires broadcast channel

Suppose we have multiple entities, A1, A2, …, An:

- Each of them can compute a pair of <private key, public key>
- Each broadcast their public key, but keep the private key secret

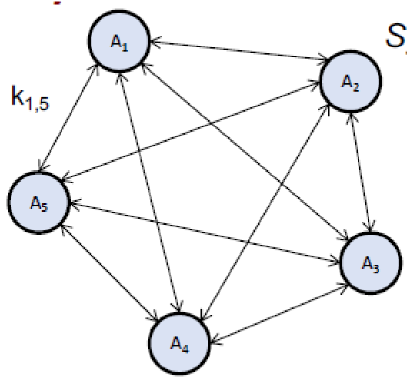
Suppose Ai wants to encrypt a message m to be read only by Aj:

- Ai can use Aj's public key to encrypt m
- By the property of PKC, only Aj can decrypt it

If we don't use the PKC, then any two entities must share a symmetric key via a secure channel. Implications:

- Many keys are required in symmetric keys
- Symmetric key requires both entities to know each other before the actual communication session. In contrast, PKC can handle such scenario.

Keys Distribution



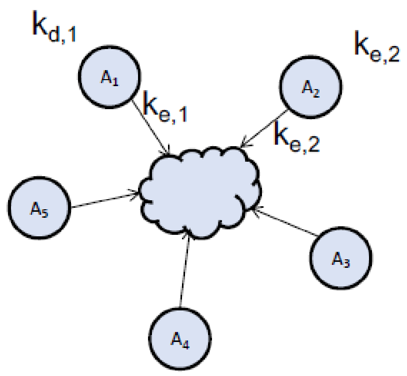
Symmetric key setting: ***Individual secure channel***

Every pair of entities requires one key.

Let $k_{i,j}$ to be the key to be shared by A_i and A_j

$k_{1,2}$, $k_{1,3}$, ...

Total number of keys: n(n-1)/2



Public key setting: ***Secure Broadcast Channel***

Each entity publishes its public key

Entity A_i publish $k_{d,i}$ and keep $k_{e,i}$

Total number of public keys: n
Total number of private keys: n

Many books stress that PKC is "better" because it uses lesser keys. In my opinion, the main advantage is that*:*
the entities do not need to know each other before broadcasting the public keys.

9

- In PKC, we still need a secure broadcast channel to distribute the public key.
- An important application of PKC is in **authentication.**

3.1.1 RSA

Classroom/Textbook RSA is the basic form of RSA. The variant used in practice is different, with padding and special considerations in choosing primes.

Many PKC represent the data as integers, and the algorithm are some arithmetic operations on the integers.

Classroom RSA Setup

- Owner randomly chooses 2 large primes **p, q** and computes **n = pq**.
- Owner randomly chooses an encryption exponent **e** s.t. **gcd(e, (p – 1)(q – 1)) = 1**
- Owner finds the decryption exponent **d** where **d e mod (p – 1)(q – 1) = 1**
- Owner publishes **(n, e)** as public key and safe-keep **(n, d)** as the private key.

e = 65537 is commonly chosen; Compromise between avoiding potential small-exponent attack and still allowing efficient encryption

Given m, the ciphertext c is: $\mathbf{c = m^e \bmod n}$

Given c, the plaintext c is: $\mathbf{m = c^d \bmod n}$

Interchangeable role of encryption and decryption key

Classroom RSA allows us to use the decryption key **d** to encrypt, and then the encryption key **e** to decrypt. i.e. we can swap the **d** and **e.**

This property is special to RSA. It usually does not hold in other PKC schemes. This property is useful in designing signature scheme.

Algorithmic Issues

Given **n, m, e**, there is an efficient algo to compute exponentiation: $\mathbf{m^e \bmod n}$. Likewise, there is an efficient algo to compute $\mathbf{c^d \bmod n}$.

(Step 1 in setup: Primality test): To choose the primes p and q, randomly pick a number and test whether it is a prime. Since there are many primes, there is a high chance a randomly chosen number is a prime. (Some primes are weak and lead to attack)

(Step 3 in setup): The value of d can be efficiently computed from e and n using the extended Euclidean algorithm.

3.1.2 Security of RSA

The problem of getting the RSA private key from the public key is as difficult as the problem of factorizing n. However, it is not known whether the problem of getting the plaintext from the ciphertext is as difficult as factorization.

Post-Quantum Cryptography

A quantum computer can factorize and perform “discrete log” in polynomial time. Hence both RSA and “discrete log based” PKC will be broken with quantum computer.

Post-Quantum Cryptography refers to PKC that are secure against quantum computer.

Padding of RSA

IV is required so that encryption of the same plaintext at different time would give different ciphertext.

Classroom RSA has some interesting properties (e.g. homomorphic property). These properties are useful in applications but lead to attacks and information leakages. Such properties can be destroyed using padding.

The standard PKCS#1 add optimal padding to achieve the above.

RSA vs AES

One common argument that RSA is better than AES is that RSA is provably secure. While it can be prove that getting the private key from the public key is as difficult as factorization, it is not known whether the problem of getting the plaintext from the ciphertext and public key is as difficult as factorization (RSA problem).

Factorization can be efficiently done by Quantum Computer. RSA is broken under quantum computer, but not clear for AES.

Signature vs MAC

- Signature has easier key management compared to MAC
- Signature scheme achieves **non-repudiation**
- Digital signature is like handwritten signature in legal document – no one except the authentic signer can generate the signature

Non-Repudiation: *Assurance that someone canot deny previous commitments or actions*

(Using mac):	Alice sends Bob a message. The MAC is computed with a shared key. Alice can claim Bob generated the MAC.
(Using signature):	Alice sends Bob a message signed using her private key. She cannot deny she sent the message because only she can sign the message using her private key. The signature proofs Alice generated the message.

Design of Signature Scheme

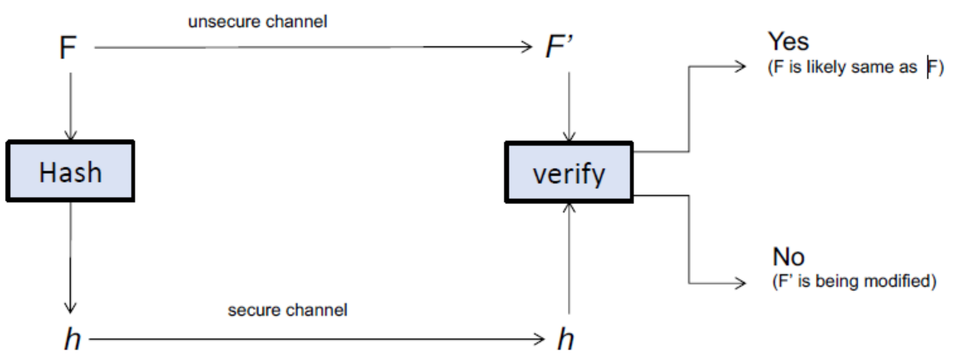
Most signature scheme consists of two components. An unkeyed hash, and the sign/verify algorithm.

RSA-Based Signature

- Use RSA for signing and verification. The signature is the “encrypted” digest. During verification, decrypt to obtain the digest and compare.

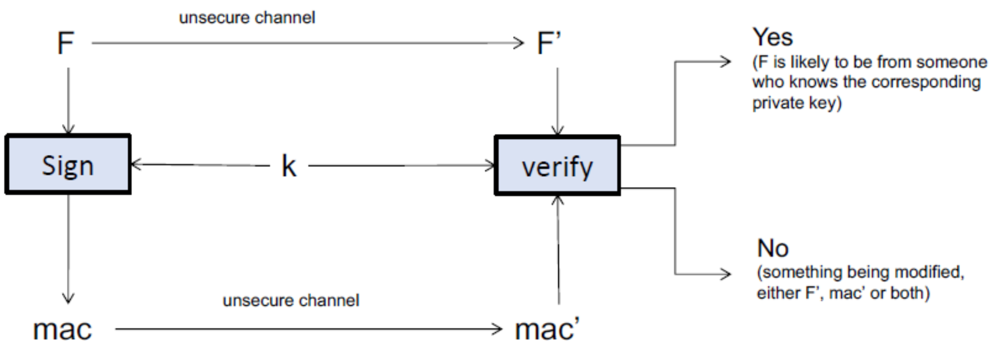
Data Authenticity Summary

Digest (Hash)



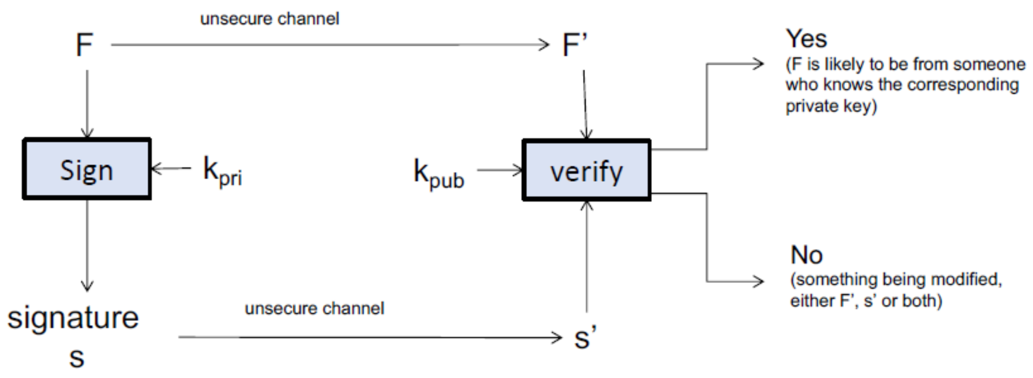
- The digest must be sent through secure channel
- **Security Requirement**: (Collision) Difficult to find a F' with the same digest h
- **Notes**:
  - If the digest is not sent through a secure channel, attacker could send another digest that matches their file (h' for F')
  - If the digest matches, F is likely to be same as F' with high probability, but not guaranteed (hash collision)

MAC (Message Authentication Code)


unsecure channel
F
F'
Sign
k
verify
Yes
(F is likely to be from someone who knows the corresponding private key)
No
(something being modified, either F', mac' or both)
unsecure channel
mac
mac'

- The MAC can be sent through an unsecure channel
- **Security Requirement**: (Forgery) Without knowing *k*, it is difficult to forge a mac

Signature

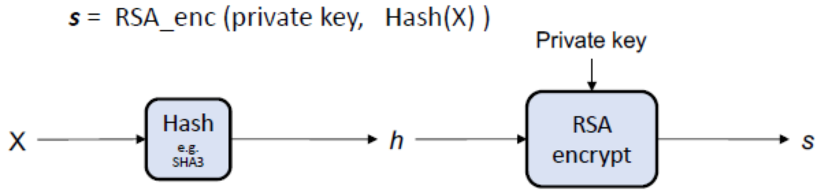

unsecure channel
F
F'
Sign
k_{pri}
k_{pub}
verify
Yes
(F is likely to be from someone who knows the corresponding private key)
No
(something being modified, either F', s' or both)
signature s
unsecure channel
s'

- Verifier and Signer uses different key
- **Security Requirement**: Without knowing the private key, it is difficult to forge a signature

RSA-based Signature

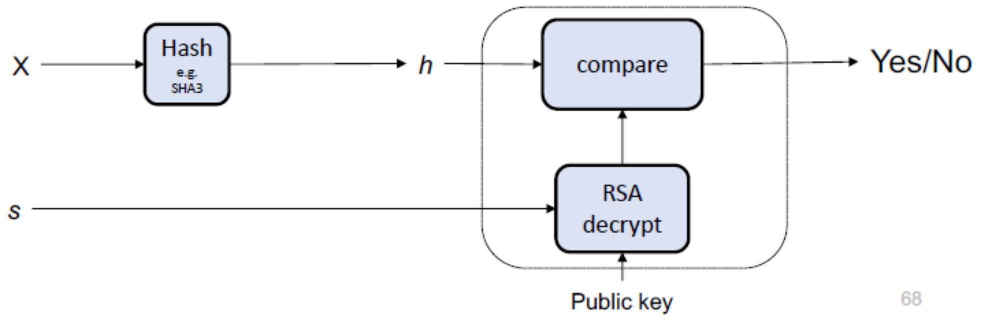
Use RSA for signing and verification. Essentially, the signature is the “encrypted” digest. During verification, decrypt to obtain the digest and compare.

s = RSA_enc (private key, Hash(X))



Verification of (X,***s***) is done by using the public key.

If Hash (X) = RSA_dec (public key, s) then accept, else reject.



3.5.1 Birthday Attacks

All hash functions are subjected to birthday attack. A hash collision occurs when: $h(x_1) = h(x_2) \ \ and \ \ x_1 \neq x_2$.

Length of digest: When key length for symmetric key is 112, the corresponding recommended length for digest is atleast 224 (should be double).

Suppose we have M messages, each message is tagged with a value randomly chosen from {1,2,3,…,T}

If $M > 1.17\ T^{0.5}$, there there is more than 0.5 probability that there is a pair of messages tagged with same value.

In general, the probability is

(From a set of T values, choose M values uniformly)

$$P = 1 - e^{-\frac{M^2}{2T}}$$

Let $\mathcal{S}$ be a set of k distinct elements where each element is an n bits binary string. Now, let us independently and randomly select a set $\mathcal{T}$ of m n-bit binary strings. It can be shown that, the probability that $\mathcal{S}$ has non-empty intersection with $\mathcal{T}$ is more than

$$1 - 2.7^{-km2^{-n}}$$

Lecture 4 – PKI + Channel Security

4.1 Public Key Distribution

1. **Public Announcement**
   The owner broadcasts her public key via email, publishing in social media, or namecard etc.
   Limitations:
   - Not standardized
   - No systematic way to search/verify the public key

2. **Publicly Available Directory**
   If Bob wants to find the public key associated to a name, he can search the public directory by querying the directory server. E.g. https://pgp.mit.edu
   Potential Issues:
   - Anyone can post their public keys in the server. Not clear how to verify the info. How does the server verify that the information is authentic?
   - In terms of performance, the server is the bottleneck and is the “single-point-of-failure”
   - Not everyone trust the server. E.g. Certain entities might not trust mit.edu to host the server.

3. **Public Key Infrastructure (PKI)**
   PKI is a standardized system that distribute public keys. To address limitation of previous 2 methods. **The main objective is to be deployable on a large scale**.
   Centered around two components/notions:
   - Certificate
   - Chain of trust of Certificate Authority (CA)

   Other issues: Revocation

4.2.2 Certificate Authority (CA)

- “Certificates” are useful in distributing public keys. A Certificate Authority (CA) issues certificates.
- **CA is a trusted authority that manages a directory of public keys**. An entity can request adding its public key to the public directory. Anyone can send queries to search the directory. (Certificates facilitate checking/querying to be done in an offline manner)
- The CA also has its own public-private key. We first assume that the CA’s public key has been securely distributed to all entities involved. (A secure channel is needed to distribute CA’s public key)
- Most OSes and browsers have a few pre-loaded CA’s public keys known as the “root” CA. Not all CAs’ public keys are preloaded. Other CA’s public keys can be added through the chain-of-trust.

4.2.1 Certificate

A certificate is a digital document that contains **atleast** the following 4 main items

- The **name** e.g. alice@yahoo.com
- The **public key** of the owner
- The **time window** that this certificate is valid
- The **signature** of the CA

Another important information:

- Usage of certification: E.g. (1) the type of “name”, whether it is an email address or domain name. (2) whether the “name” can take the role of a CA (chain-of-trust)

Other Info:

- Digest (Fingerprint). For verification without using CA’s public key
- Metadata such as the type of algorithm (ECC or RSA, key length etc)

Standard: X509

Standardization bodies:

- ITU-T X.509:
  Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
- The Public-Key Infrastructure (X.509) Working Group (PKIX):
  IETF working group that creates Internet standards on issues related PKI based on X.509 certificates

Self-signed Certificate

- A self-signed certificate is not signed by a CA. It is signed by the 'name' in the certificate. The certificate is to be verified using that 'public key' in the certificate.
- A self-signed certificate by a CA is sometime called a "root-certificate" (always take extra caution in accepting root-certificate)

Certificate Summary

A certificate is simply a document signed by a CA.

- An identity
- The associated public key
- The time window that this certificate is valid
- The signature of the CA

This document "certifies" that the public key is indeed belonged to the claimed identity.
We assume that Bob already has CA's public key "installed" in his machine.

Responsibility of CA

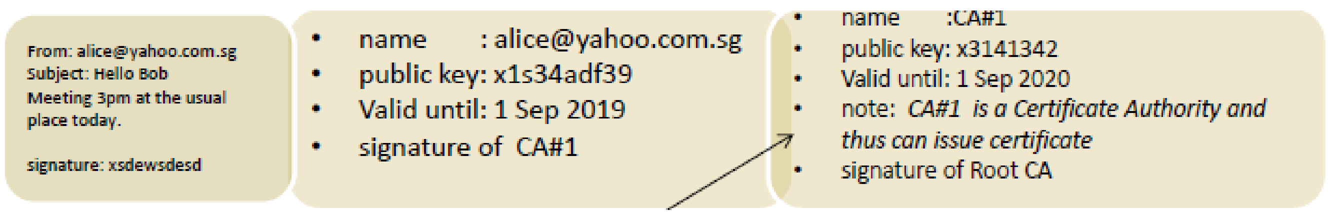
The CA, besides inssuing certificate, is also responsible to verify that the information is correct. The CA should check that the applicant indeed owns the domain name ect. This may involve manual checking and thus it could be costly. Getting a certificate signed by a CA is not free.

Certificate Chain-of-trust

- There are many CA's
- Most OS, browsers already have a few CA's public key pre-loaded. These are the "root CA".

Suppose Alice's certificate is issued (signed) by CA#1, but Bob doesn't have the public key of CA#1. Alice, anticipating that Bob might not have the public key of CA#1, can send her email, her certificate, and CA#1 certificate (issued by the root CA) to Bob. Bob can now:

- Verify CA#1's certificate using root CA's public key
- Verify Alice's certificate using CA#1's public key
- Verify Alice's email using Alice's public key


From: alice@yahoo.com.sg
Subject: Hello Bob
Meeting 3pm at the usual place today.

signature: xsdewsdesd

- name : alice@yahoo.com.sg
- public key: x1s34adf39
- Valid until: 1 Sep 2019
- signature of CA#1

- name :CA#1
- public key: x3141342
- Valid until: 1 Sep 2020
- note: *CA#1 is a Certificate Authority and thus can issue certificate*
- signature of Root CA

If Alice doesn't attach CA#1's certificate, then Bob has to obtain it from other sources.

Certificate Revocation

- **Non-expired certificates to be revoked due to different reasons:**
  - Private key was compromised
  - Entity left an organization
  - Business entity closed
  - Issuing CA was compromised

A verifier needs to check whether a certificate in question is still valid, although the certificate is not expired yet. (Circular/contradicting requirement) Two Different approaches to certificate revocation:

- **Certificate Revocation List (CRL)**
  - CA periodically signs and publishes a revocation list
- **Online Certificate Status Protocol (OCSP)**
  - OSCP Responder validates a cert in question

Either way, either an online CRL Distribution Point or an online OCSP Responder is needed. Recommendation is for a user (e.g. browser) to periodically update its local cache of revocation list. The user does not need to online check whenever the user wants to verify a certificate.

4.3 Limitations/Attacks on PKI

Implementation Bugs

There are numerous well-known implementation bugs leading to severe vulnerability.

- Some browsers ignore substrings in the "name" field after the null characters when displaying it in the address bar, but include them when verifying the certficiate
  - Name appeared in the certificate which is used in verifying the certificate:
    - www.comp.nus.edu.sg\0hacker.13525.com
  - Browser displays it as: www.comp.nus.edu.sg
- Viewers thought they are connecting via https to (b) but infact is connecting to (a)

Abuse by CA

There are many CA's. One of them could be malicious. A rogue CA can practically forge any certificate.

- Trustwave issued a "subordinate root certificate" (i.e. receipt can now issue certificate) to an organization for monitoring the network. With this certificate, the organization can "spoof" X.509 certificates and hence is able to act as the MITM of any SSL/TLS connection.

Social Engineering

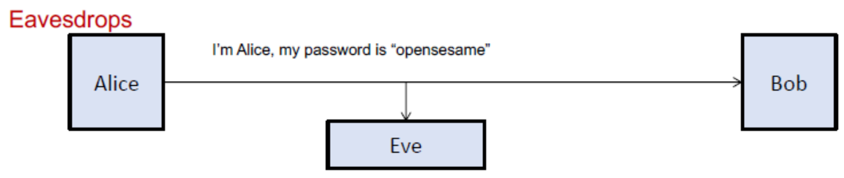
Malicious attacker rightfully register for a domain name that resembles a targeted domain name. Next, use the registered domain to confuse the victim in phishing attack. The techniques include "**Domain typosquatting**", "**Homograph attack**", or using sub-domain name. AKA Domain spoofing, URL spoofing, Fake URL

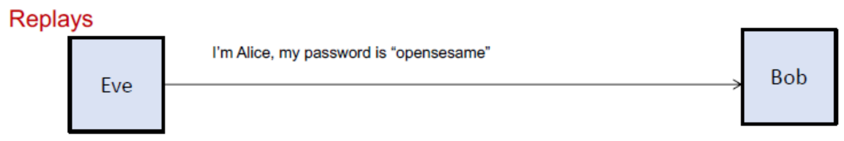
- **Method 1 (typosquatting)**
  - An attacker registered for a domain name: luminus.nus.edv.sg and obtains a valid certificate of the above name.
  - Attacker employs 'phishing attack', tricking the victim to click on the above link which was a spoofed site of luminus.nus.edu.sg
  - The address bar of the victim's browser correctly displayed https://luminus.nus.edv.sg but the victim does not notice and logs in using his/her password
- **Method 2 (sub-domain)**
  - A more commonly deployed method uses subdomain. E.g.
  - Attacker is rightful owner of 134566.com.
  - Attacker creates sub-domain luminus.nus.edu.sg.134566..com and gets valid certificate of *.134566.com

4.4 Authentication

Weak Authentication

An entity who can convince other that it knows a secret is “authentic”. Sending password over is a simple method, but an eavesdropper who get the password can simply replay it.





It is possible to have a mechanism where Alice can prove to Bob that she knows the secret without revealing the secret.

Authentication: Challenge-Resposne

Suppose Alice and Bob have a shared secret **k**, and both have agreed on a message authentication code. An entity who knows **k** is either Alice or Bob. Now an entity **P** wants to convince Alice that is is Bob.

1. P Sends to Alice a hello message: “Hi, Im bob”
2. (Challenge) Alice randomly picks a message m and sends m to P
3. (Response) P computes t = $mac_k(m)$. P sends t to Alice.
4. Alice verifies that the tag received is indeed the mac of m. If so, accepts, otherwise reject.

By property of MAC, even if Eve sniffs the communication between Alice and Bob, and obtain multiple pairs of (m, t), Eve still (1) can’t get the secret key k, and (2) can’t forge the mac for messages that Eve has not seen before.

Eve also can’t replay the response because the challenge is randomly chosen. The challenge m ensures **freshness** of the authentication process. It is known as the **cryptographic nonce** (or simply nonce).

This protocol only authenticates Bob. That is authenticity of Bob is verified. Hence it is called **unilateral authentication.** There are also protocols to verify both parties, which are called **mutual authentication.**

Unilateral Authentication using PKC

We can also have a public key version using signature. Suppose Alice wants to authenticate an entity P who claims to be Bob.

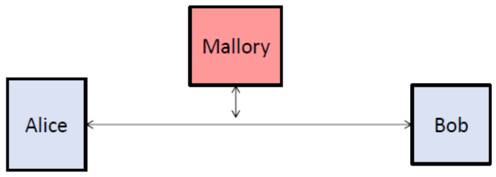
1. (Challenge) Alice chooses a random message r and sends to P:
   (“Bob, here is your challenge”, r)
2. (Response) P uses his private key to sign r. P also attaches a certificate
   (sign(r), certificate)

3. Alice verifies the certificate indeed belong to Bob and is valid. Next extracts the public key from the certificate and verifies that the signature sign(r) is correct. If so, accept.

If Alice already knows Bob’s public key, the certificate can be omitted. Similarly, we assume an attacker can observe multiple interactions between Alice and Bob. By security property of signature, the eavesdropper can’t derive Bob’s private key and can’t forge the response.

The nonce **r** ensure freshness.

Issue with Unilateral Authentication



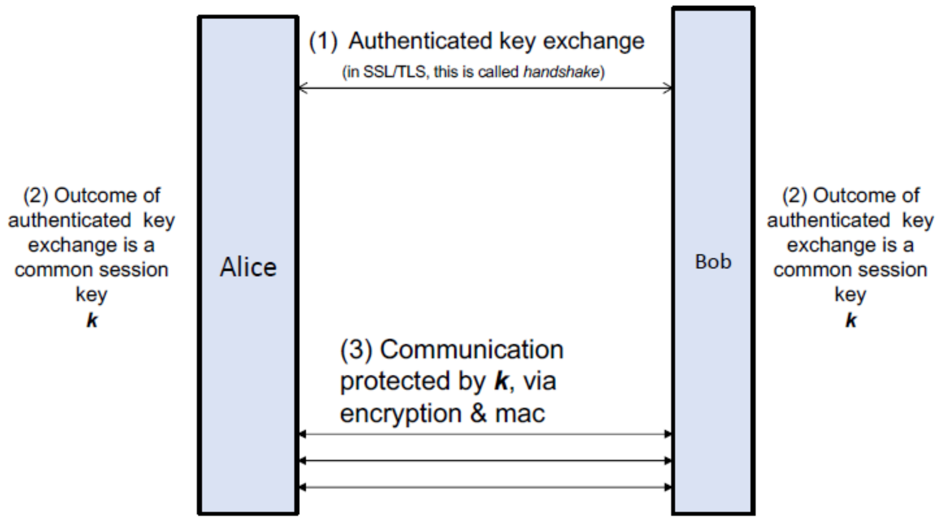
Imagine that Mallory allows Alice and Bob to carry out the strong authentication. After Bob is convinced that he is communicating with Alice, Mallory interrupts and takes over the channel. Later Mallory pretends to be Alice! (Here, Mallory is the man-in-the-middle). So, even if we employ the challenge-response in the previous slide, <u>Mallory can still succeed in this setting</u>.

The previous authentication protocols assume that the adversary is unable to interrupt the session, that is the entity is the same throughout the session.

For applications where Mallory can interrupt the session, we need something more. The outcome of the authentication process must somehow pass to the next phase. This can be done by establishing a new shared secret **k**. In other words, the outcome of the authentication protocol is a shared secret **k**. This shared key is aka **session key.** Subsequent phase will be protected using **k.**

The process of establishing a secret (with or without authentication) between Alice and Bob is called **key-exchange** or **key-agreement**.

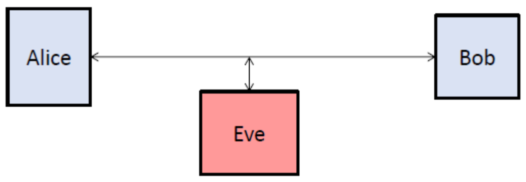
If the process is incorporated with authentication, then it is called **Authenticated key-exchange.** A well-known scheme is called **station-to-station** protocol.


(1) Authenticated key exchange
(in SSL/TLS, this is called *handshake*)
(2) Outcome of authenticated key exchange is a common session key ***k***
Alice
Bob
(2) Outcome of authenticated key exchange is a common session key ***k***
(3) Communication protected by ***k***, via encryption & mac

4.5 Key-exchange, authenticated key-exchange

Key-exchange (Secure against Eve, but not Mallory. No authentication)

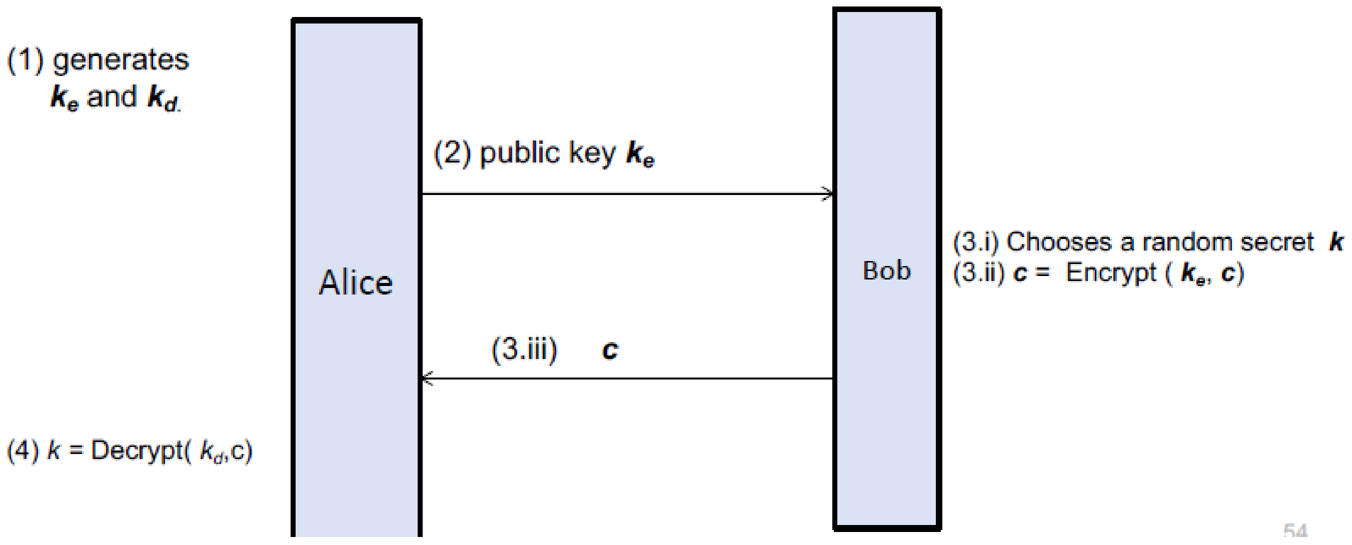
Alice and Bob want to establish a common key. The established key can be used to protect (e.g. via cipher,mac) subsequent communication between Alice and Bob. This interaction could be eavesdropped by Eve. After seen the interactions, Eve wants to guess the established key.



Here, we only consider Eve who can sniff and wants to guess the key. There is no consideration of authenticity here.

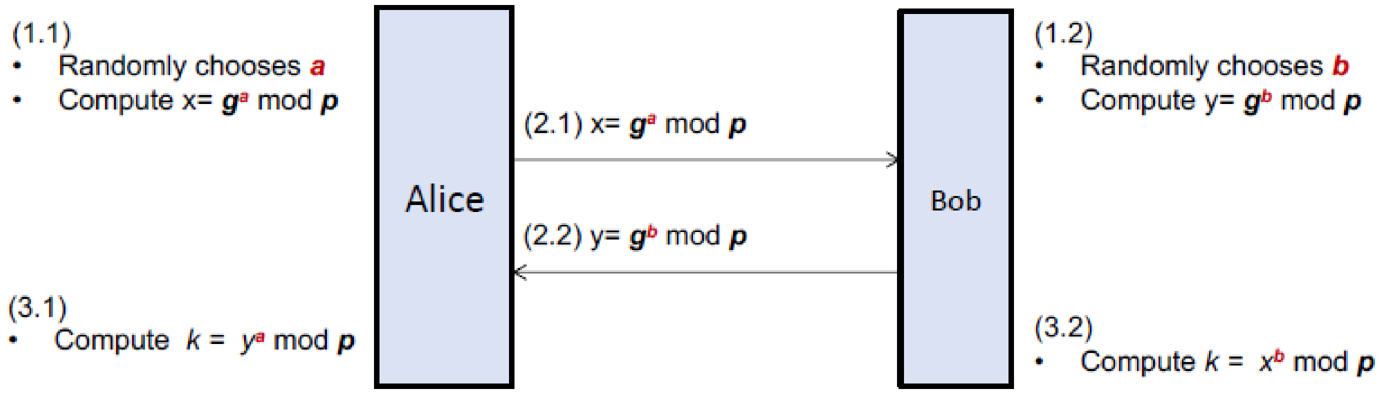
PKC-based Key-exchange (no authentication)

1. Alice generates a pair of private/public key **(k_e, k_d).**
2. Alice sends the public key **k_e** to Bob.
3. Bob carries out the following
   i. Randomly chooses a secret **k**,
   ii. Encrypts **k** using **k_e**
   iii. Sends ciphertext **c** to Alice
4. Alice uses her private key **k_d** to decrypt and obtain **k**.


(1) generates k_e and k_d.
Alice
(2) public key k_e
Bob
(3.i) Chooses a random secret k
(3.ii) c = Encrypt (k_e, c)
(3.iii) c
(4) k = Decrypt(k_d,c)
54

- Attacker (Eve) can obtain the public key k_e and ciphertext c.
- By security of PKC, from the public and ciphertext, attacker can't get any information of the plaintext which is the key k.

Dillie-Hellman key-exchange (no authentication)
We assume both Alice and Bob have agreed on two public parameters, a **generator g**, and a **large prime p**. **Both g and p are not secret** and known to the public.

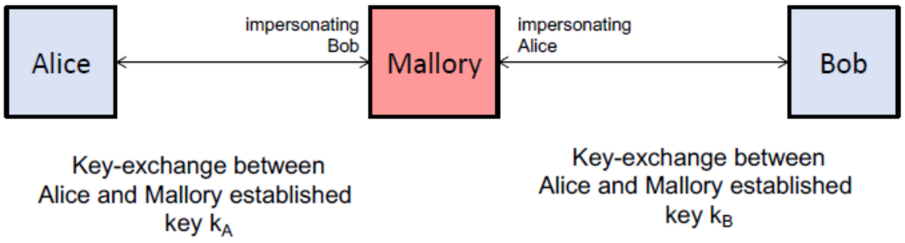


Security relies on the CDH assumption.

Computational Diffie-Hellman CDH assumption:
Given g, p, $x = g^a \bmod p$, $y = g^b \bmod p$, it is computationally infeasible to find $k = g^{ab} \bmod p$.

Remarks:
1. Step (1.1)&(1.2), (2.1)&(2.2), (3.1)&(3.2) can be carried out in parallel.
2. The assumption seems self-fulfilling. Nonetheless, there are many evidences that it holds.
3. The operation of "exponentiation" can be applied to any algebraic group, i.e. not necessary integers. CDH doesn't hold in certain groups. The crypto community actively searches for groups that CDH holds. E.g. Elliptic Curve Cryptography ECC is based on elliptic curve group where CDH believed to hold.

The basic key-exchange can't guard against Mallory.

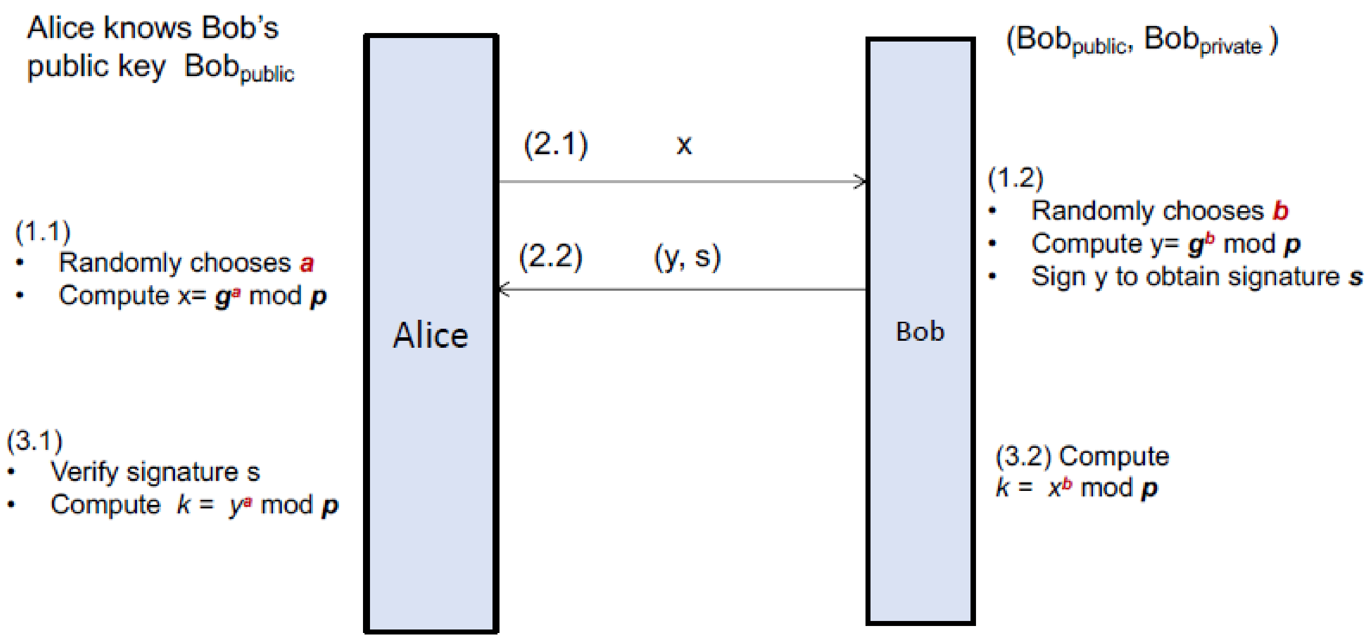

Alice
impersonating Bob
Mallory
impersonating Alice
Bob
Key-exchange between Alice and Mallory established key k_A
Key-exchange between Alice and Mallory established key k_B

In this case, Alice mistaken that Mallory is Bob. Communication from Alice is encrypted using k_A. Mallory can decrypt using k_A and re-encrypt using k_B. Hence, Mallory can see and modify the message.

Authenticated Key-exchange
- A key-exchange protocol assume that the adversary can only sniff, but not malicious.
- To prevent malicious Mallory, we need **authenticated key-exchange.** The authentication can be achieved using PKC. In mutual authentication, Alice and Bob need to know each other public key. In unilateral authentication, only one party need to have public key. After the protocol has completed, a common key (**session key)** is established. Authenticated key-exchange is also applicable in symmetric key setting, although the public key setting is more commonly deployed.
- It turns out th at authenticated key-exchange be be easily obtained from existing key-exchange (either PKC or DH based key-exchange). This is done by simply signing all communication using the private key.
- We have a special name for authenticated key-exchange that uses DH: **Station-To-Station Protocol (STS)**

Station-To-Station Protocol (STS) (Authenticated key-exchange based on DH)

We assume both Alice and Bob have agreed on two public parameters, a **generator g**, and a **large prime p**. **Both g and p are not secret** and known to the public. Here, we consider unilateral authentication. Alice want to authenticate Bob.



Alice knows Bob's public key Bob_{public}
$(Bob_{public}, Bob_{private})$
(2.1) x
(1.2)
• Randomly chooses b
• Compute $y = g^b \bmod p$
• Sign y to obtain signature s
(1.1)
• Randomly chooses a
• Compute $x = g^a \bmod p$
(2.2) (y, s)
Alice
Bob
(3.1)
• Verify signature s
• Compute $k = y^a \bmod p$
(3.2) Compute $k = x^b \bmod p$

Note: This is unilateral authentication. Can extend it to mutual by making Alice sign her messages in step (2.1)

- *Before the protocol*:
  1. Alice has a pair of public, private key (A_{public} , $A_{private}$).
  2. Bob has a pair of public, private key (B_{public} , $B_{private}$).
  3. Alice knows Bob public key and vice versa. These two sets of keys are known as the ***Long-term key*** or ***Master key.***

- They carry out Authenticated key exchange protocol (e.g. STS). If an entity is not authentic, the other will halt.

- *After the protocol*:
  1. Both A and B obtain a shared key **k**, known as the ***Session key.***

- Security Requirement.
  1. (Authenticity) Alice is assured that she is communicating with an entity who knows $B_{private}$.
  2. (Authenticity) Bob is assured that he is communicating with an entity who knows $A_{private}$.
  3. (confidentiality) Attacker unable to get the session key.

4.6 Securing Communication Channel E.g. TLS

Consider a communication channel that is subjected to a Mallory (sniffing, spoofing, modifying), how to secure it using cryptographic tools?

E.g. Alice wants to visit a website Bob.com, how to achieve authenticity (i.e. Alice being assured that Bob.com is authentic) and confidentiality (i.e. no info leakage of the communication) ?

In TLS/SSL: (https uses TLS)

(1) Using **long-term keys (i.e. Bob's public and private key),** carry out authenticated key-exchange (aka handshake in TLS) to establish **session keys.**

(2) Subsequent communication protected by the session keys.

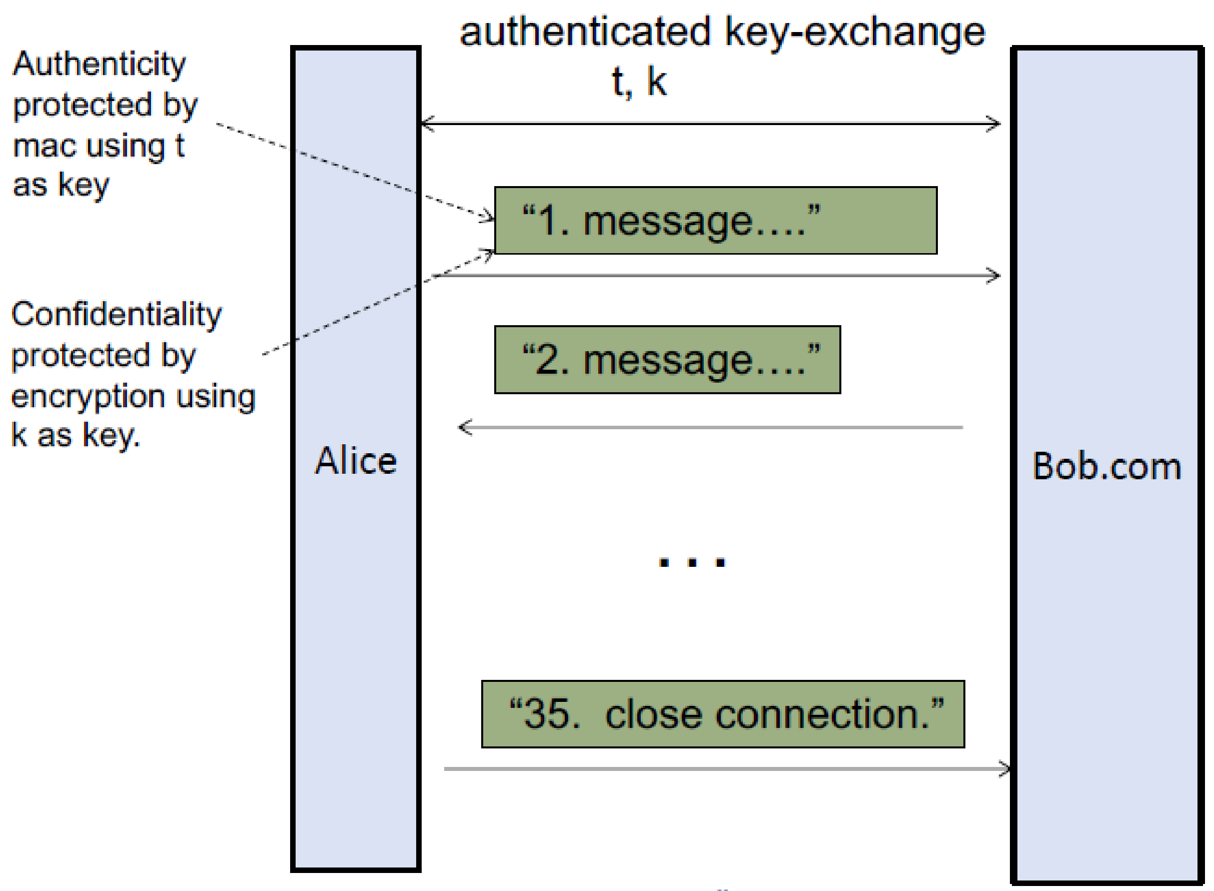
Alice wants to visit Bob.com: How TLS does it.

(Step 0) Alice obtains Bob.com's public key. This is done by having Bob sending his certificate to Alice.

(Step 1) Alice and Bob.com carry out ***unilateral authenticated key exchange*** protocol with Bob's private/public key. After the protocol, both Bob and Alice obtain two shared keys ***t*** , ***k*** where ***t*** is the secret key of the MAC, and ***k*** is the secret key of the symmetric-key encryption, say AES. They are called the ***session keys***. From Alice's point of view, the protocol is secure in the sense that, only an entity who knows Bob's private key is able to complete the protocol. So, Alice is convinced that the entity who now holds ***t***, ***k*** is Bob. Here, Bob doesn't care about Alice's authenticity.

(Step 2) Subsequent interactions between Alice and Bob.com will be protected by ***t***, ***k*** and a sequence number. Suppose m1, m2, m3, … are the sequence of message exchanged, the actual data to be sent for mi is E_k (*i* || m) || mac_t (E_k (*i* || m))
where *i* is the sequence number.

• || refer to string concatenation.


authenticated key-exchange
t, k
Authenticity protected by mac using t as key
"1. message…."
Confidentiality protected by encryption using k as key.
"2. message…."
Alice
Bob.com
. . .
"35. close connection."

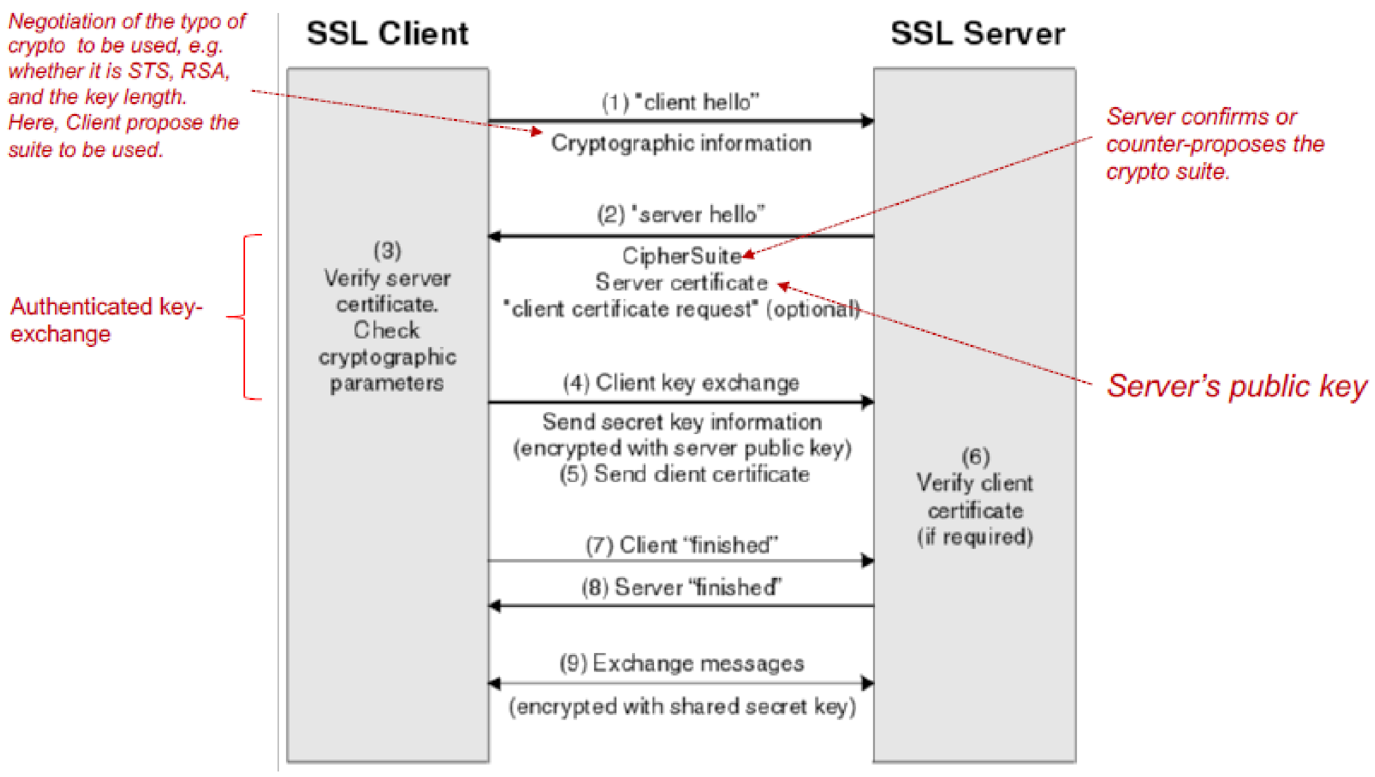
The data eventually sent is E_k ("1. message etc") || mac_t (E_k ("1. message etc"))

Relationship among TLS/SSL/https

- SSL and Transport Layer Security (TLS) are protocols that secure communication using cryptographic mean.
- SSL is predecessor of TLS
- Https is built on top of TLS

Question: Alice is in a café. She uses the free wifi to upload her assignment to LumiNUS (which uses https). The café owner controls the wifi router, and thus can inspect every packet going through the network. Can the café owner see Alice's report? ***No***. We would discuss this later in network security.

TLS handshake (authenticated key exchange)


Negotiation of the typo of crypto to be used, e.g. whether it is STS, RSA, and the key length. Here, Client propose the suite to be used.
SSL Client
SSL Server
(1) "client hello"
Cryptographic information
Server confirms or counter-proposes the crypto suite.
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
Authenticated key-exchange
(3) Verify server certificate. Check cryptographic parameters
Server's public key
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

Lecture 5 –Network Security

Computer Network

Computer Network establishes communicating connection between entities.

To share networking resources and enhance robustness, instead of having dedicated line between any two nodes, **packet switching** is deployed.

1. Messages route via multiple switches and routers
2. Messages are broken into "packets/frames"

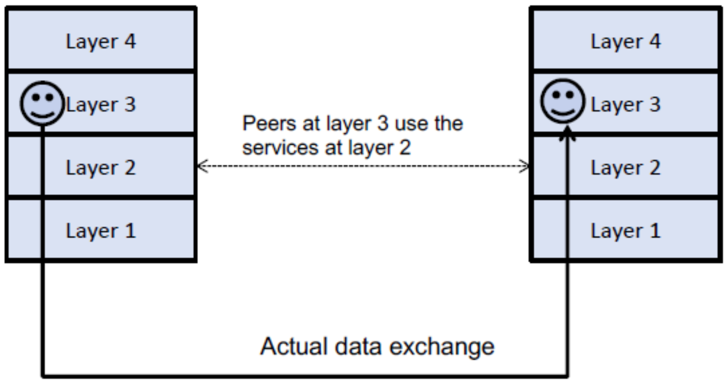
5.1 Network Security Introduction

There is a need to "route" messages among multiple intermediate nodes. In contrast, network security focuses on the effects of attackers among the intermediate nodes. Attacker wants to steal, modify, disrupt as in CIA. In particular, attacker wants to compromise "routing".

Multiple hops & Network layers

To factiliate routing, intermediate nodes need to see & modify some routing information. To handle diffferent type of intermediate nodes, network protocols are abstracted as layers.

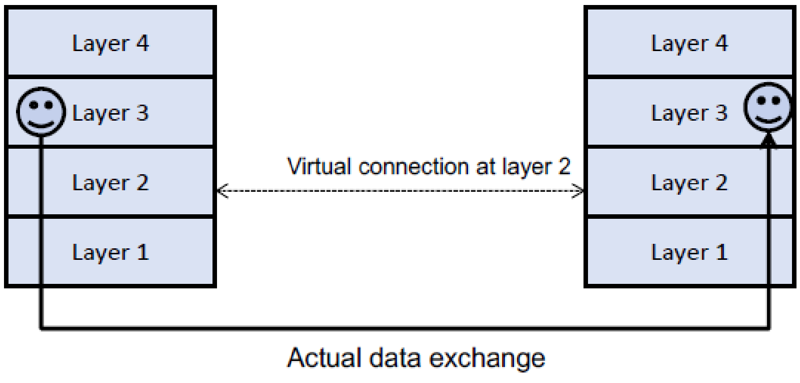
Conceptually, layer (N-1) provides services for entities in layer N. The peer entities in the N layer communicate by executing protocols provided by layer (N-1).



Network Layers

The layer N services/protocols are built on top of protocol provided by layer (N-1). We can view the protocol at (N-1) as a ***virtual connection***.

In turn, the layer N is a virtual connection for layer (N+1).



A hypothetical example:

A Layer 3 protocol (called ***handshake***) is designed as follow,

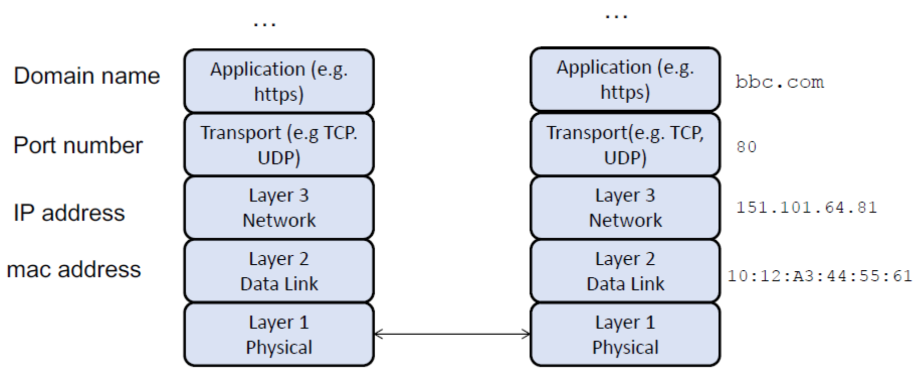
Handshake (A,B):

(1) A → B: "hello", *m*
(2) A ← B: certificate of B, sign (*m*)
If correct, A output accept; else output reject.

The connection "A → B" is provided by Layer 2. The "virtual connection" at layer 2 sends the message "hello" from A to B in step (1) and sends the certificate in step (2).

Now, another entity in Layer 4 can invoke layer 3's protocol *handshake*.

Internet Layers



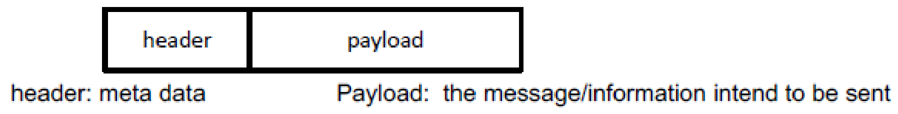
The OSI model actually has 7 layers.

A single node has different name in different layers.

Domain name:	bbc.com
Port Number:	80
IP Address:	151.101.64.81
MAC Address:	10:12:A3:44:55:61

Data flow across layers

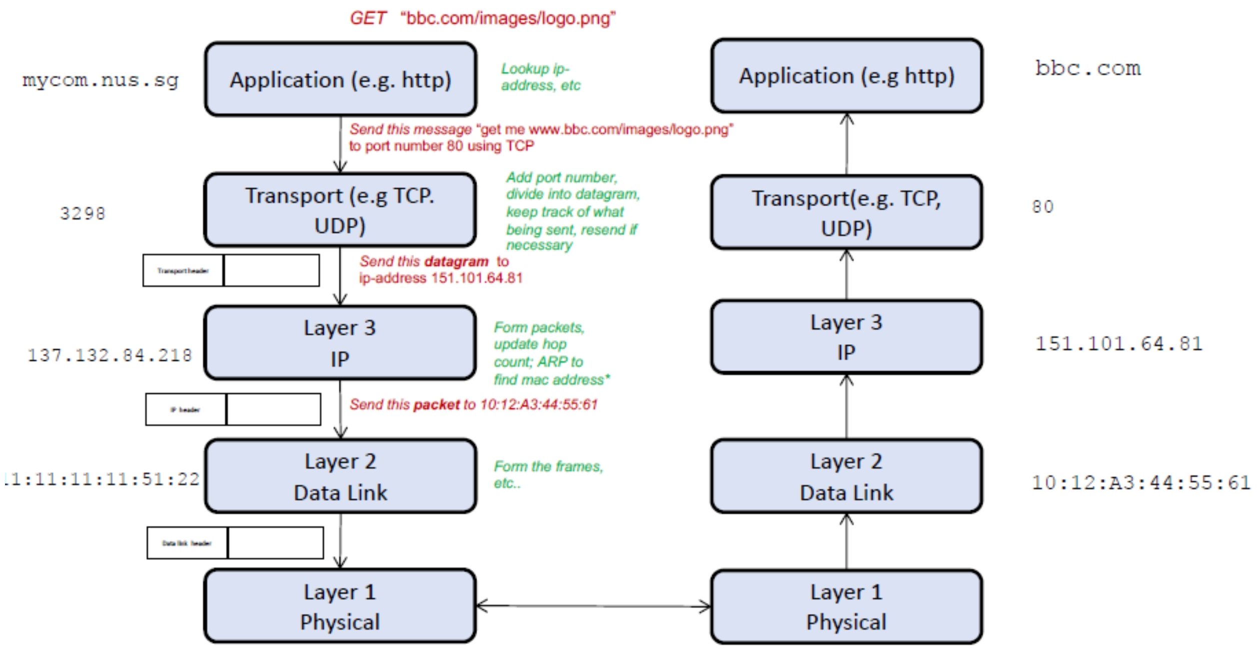
When a layer N service is invoked to send a message m, the protocol in layer N might transform (e.g. encrypt) m into smaller pieces of “payload”. For each payload, the protocol also generates a header. The header and payload form a data unit in layer N to be sent by layer N-1.

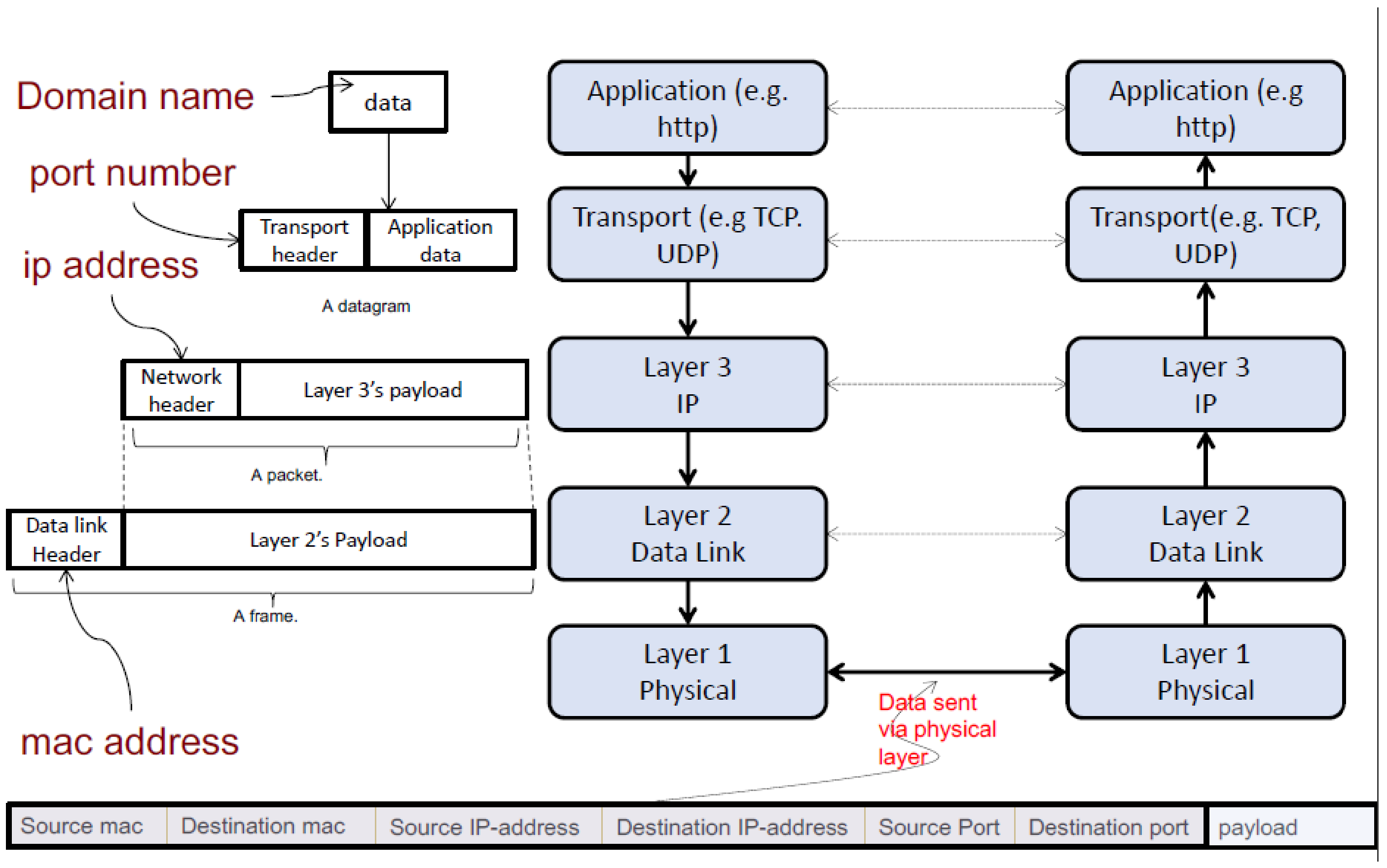


Layer N at the receiver, from a series of received data units reconstruct back to m.

Each header contains atleast two pieces of information

- The source address, i.e. the sender’s address (at layer N)
- The destination adderss, i.e. the receiver’s address (at layer N)

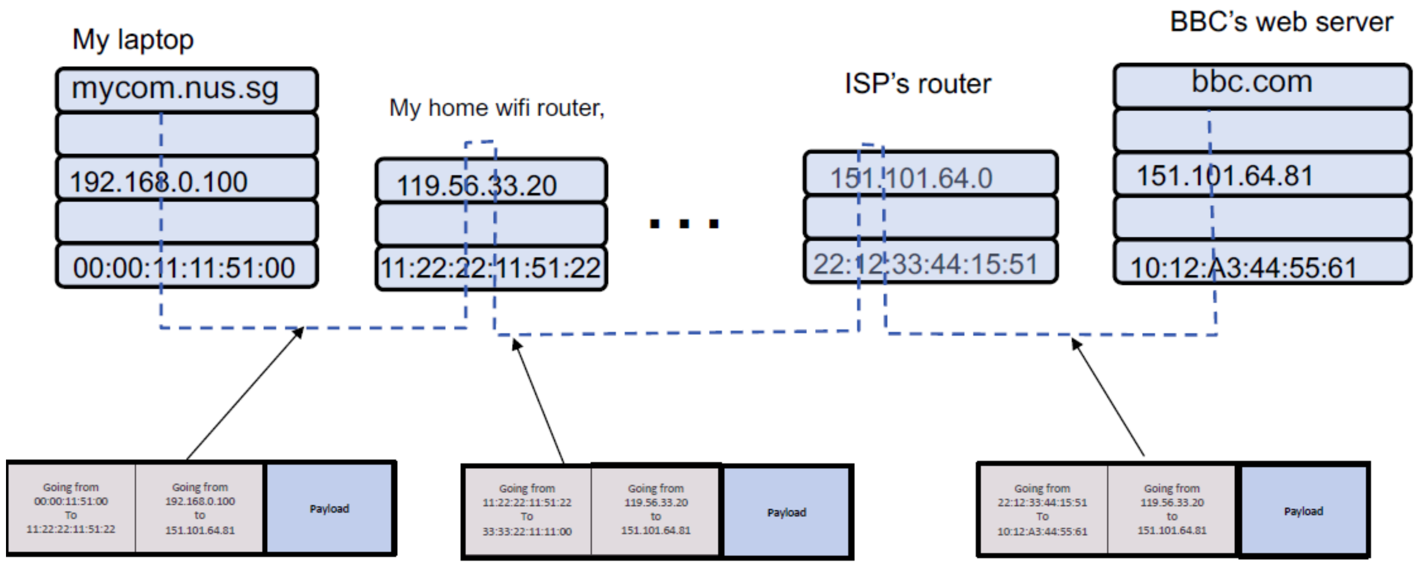




In **transport** layer (layer 4), each data unit called a **datagram**.
In **network** layer, it is called a **packet**.
In **data link** layer, it is called a **frame**.

Intermediate Nodes

Data are routed through multiple hops. Intermediate nodes could be owned by different third parties, e.g. Internet service provider (ISP), company's firewall. To facilitate routing, intermediate nodes see routing/header information, and might also change them.


My laptop
mycom.nus.sg
192.168.0.100
00:00:11:11:51:00
My home wifi router,
119.56.33.20
11:22:22:11:51:22
. . .
ISP's router
151.101.64.0
22:12:33:44:15:51
BBC's web server
bbc.com
151.101.64.81
10:12:A3:44:55:61
Going from 00:00:11:51:00 To 11:22:22:11:51:22
Going from 192.168.0.100 to 151.101.64.81
Payload
Going from 11:22:22:11:51:22 To 33:33:22:11:11:00
Going from 119.56.33.20 to 151.101.64.81
Payload
Going from 22:12:33:44:15:51 To 10:12:A3:44:55:61
Going from 119.56.33.20 to 151.101.64.81
Payload

MITM

A MITM sits in-between two communicating parties. Unless otherwise stated, **the MITM can sniff, spoof, modify, drop the data**.

- The **MITM sits in Layer 3**. We mean that the MITM can see the input to the layer 3 (i.e. Datagram, Transport header), can decide what is the output of Layer 3 (i.e. Packet, Network Header), and know all internal info store in Layer 3, e.g. secret key.
- The **MITM sits in between Layer 2 & 3**, or the **MITM sits just below layer 3**, or **MITM sits just above layer 2**. The MITM can see and modify output of Layer 3, but doesn't have access to the internal data in layer 3.

Challenges in Network Security

(**Intermediate nodes and layers**) There are many intermediate nodes, each handling routing-related information at a different layer.

(**Security Requirements**)
Availability is the main concern in networking. (crypto is insufficient)
Confidentiality and **Integrity** of Routing

- Modifying routing process would break connections (***availability***), redirect traffic to adversary (***confidentiality***). Even if the payload is encrypted, we don't want the adversary to see the ciphertext – side-channel leakage, implementation flaw)
- Leakage of routing information could reveal connectivity information (A is talking to B). **Anonymity** & **Privacy** is concerned.

(Legacy & Security Tradeoff) Initial design of many networking protocols did not consider intention attacks.

- For better performance, many services do not employ strong protection mechanisms (e.g. DNS).

(Management) There is a need to isolate and control data flow. (Firewall)

Transport + IP Layer

Very often, the transport and IP layer are treated as one single layer. In this combined layer, the address of a communicating entity in a particular channel is an IP-address and a port.

- Each node in the network has **65535 ports**
- A communication channel between the two nodes is established by connecting two ports. Ex. Between 11.11.1.1:**2** and 55.55.5.5:**65533**.

UDP/IP

If an application wants to invoke the UDP/IP protocol to send a message, the library call is typically of the form:

```
DatagramSend(2, "33.43.100.2", 65533, message)
DatagramSend(SrcPort, DestIP, DestPort, Message);
*The source IP is implicit and thus no need to specify
```

src ip	dest ip	src port	dest port	message

The library Datagramsend first constructs an **IP** datagram and then an **IP** packet. The packet is then passed to the data link layer.

- There is limit on the size of the array message, at most ~65000 bytes.
- DatagramSend does not return a result indicating whether the destination received the packet. There is a posibility that the packet is lost. Hence UDP is: **unreliable**, and a **connectionless communication**.

TCP/IP

In contrast, TCP/IP is **reliable**.

An application would typically make library calls of the form and in the following order:

```
P = open_connect(2, "33.43.100.2", 65533)
      send(P, out_message)
      read(P, out_message)
close_connection(P)
*There could be multiple rounds of send/read
```

src ip	dest ip	src port	dest port	message

open_connect would carry out some form of handshake protocol (known as TCP 3-way handshake) between the two nodes.

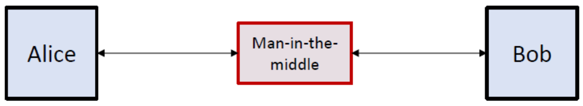
send would construct the **IP packets** of the following form and pass them to the **datalink** layer. In case the message is too long, multiple IP packets will be formed. The protocol also employs a mechanism of *re-sending, reordering, acknowledgement*, etc to ensure that destination indeed receives the message.

Reliability does not imply Security

TCP/IP is reliable, but not secure. Intermediate nodes along the communication can still modify data in the header and payload.

A malicious intermediate node might act as a MITM in the IP layer. It can

- spoof an IP packet to inform one node to close the connection (availablity)
- change information on the packets ordering, so the dest reconstruct a scrambled message



Ports

We can imagine that behind certain ports, there are applications waiting to process data coming via the respective port. In such case, we say that the node/process is "listening" to the oprt, and the port is a "listening port". If the port is not listening, then it is a "closed port".

- Data sent to a closed port will be dropped.
- *There are security implications (to be explored in firewall design).*

5.2 Name resolution and Attacks

Protocols: DNS, ARP

Techniques: Spoofing, Poisoning

Resolution protocol

Imagine you are the sender in layer N, and you know the receiver address in layer N. To use the virtual connection in layer (N-1), you would need to know the receiver name in layer (N-1). The method of finding the corresponding name is called the "resolution" protocol.

- **DNS (Domain Name System)**:	Domain name -> IP Address
- **ARP (Address Resolution Protocol):**	IP Address -> Mac Address

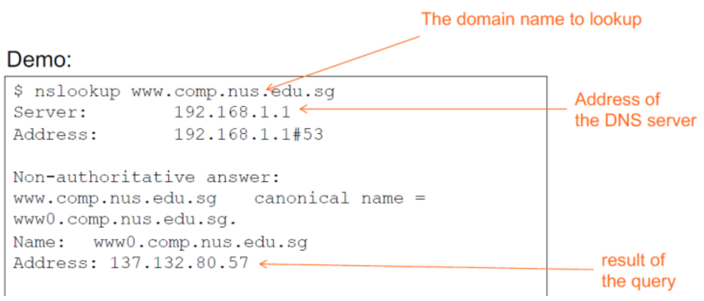
Many initial design of the resolution protocol didn't take security into account.

- DNS attack targets association of domain name with ip address.
- ARP attack targets association of IP address with mac address.

- Spoofing is on forging messages (integrity)
- Poisoning is to modify cache stored in the victim (integrity of internal state)

DNS (Domain Name System)
Given a domain name (e.g. www.comp.nus.edu.sg) , its IP address can be found by querying a remote DNS server. Recap that the process is known as **resolution**. The client who initiates the query is called the **resolver**. If the address is found, we say that the domain name is resolved.

*`nslookup` can be used to resolve the IP of a domain name.

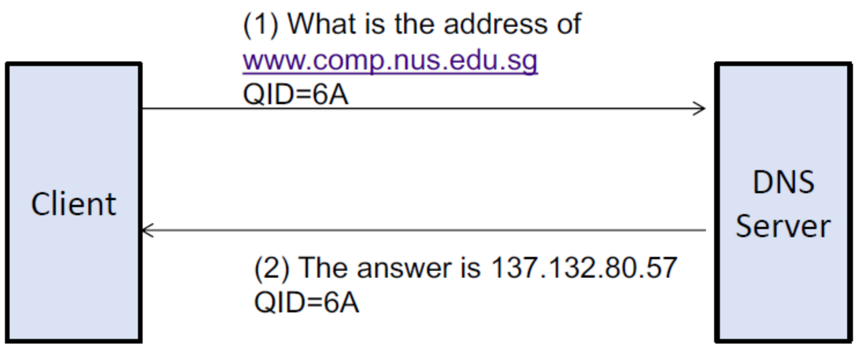


Demo:

```
$ nslookup www.comp.nus.edu.sg
Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
www.comp.nus.edu.sg    canonical name =
www0.comp.nus.edu.sg.
Name:   www0.comp.nus.edu.sg
Address: 137.132.80.57
```

DNS query/answer
DNS Resolution:
(1) Client sends a query to DNS Server (using UDP protocol)
(2) DNS sends the answer back (UDP protocol)



- The query contains a **16-bit number**, known as **QID** (query ID)
- The response from the server must also contain a QID
- If the QID in the response doesn't match the QID in query, the client rejects the answer.

DNS Spoofing

When not under attack:

- Alice is using a café free wifi to surf the web.
- Alice wants to visit nus.edu.sg
- She types the domain name into browser
- The browser makes a query to a DNS server to determine the IP address
- After the browser obtains the IP address, it connects to the IP address

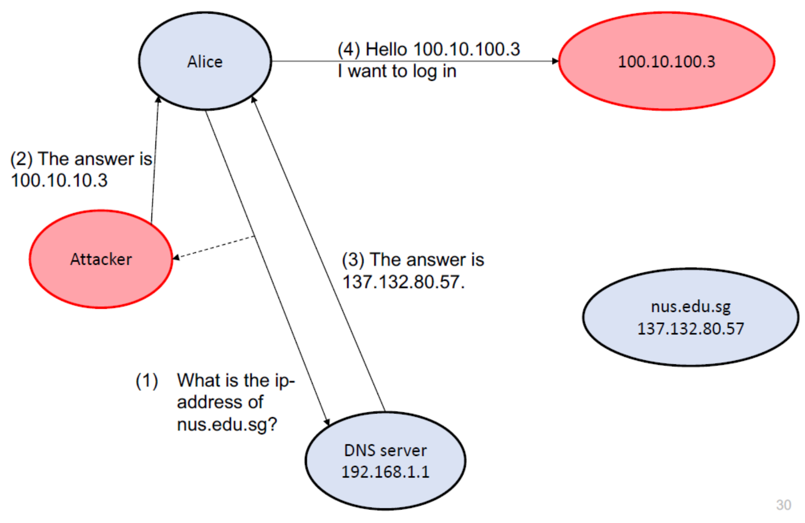
Attack:

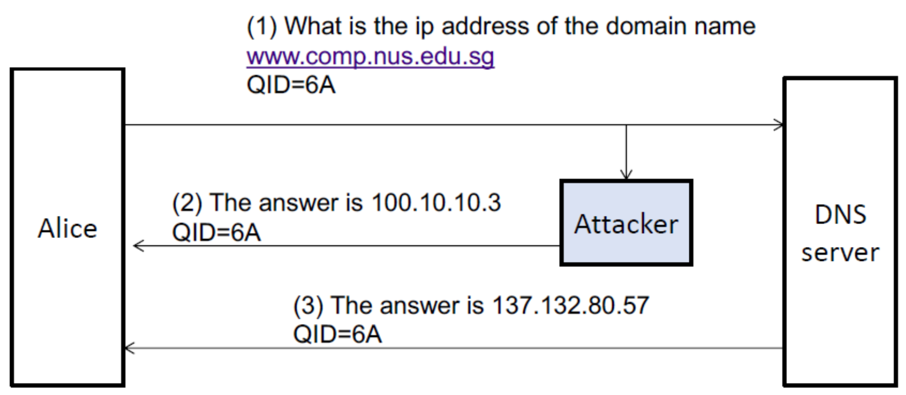
We consider an attack who is also inside the café. Since the wifi is not protected, the attacker can:

- Sniff data from the communication channel
- Inject spoofed data into the communication channel

However, the attacker can't remove/modify data sent by Alice (the attacker sits below the physical layer).

DNS Spoofing



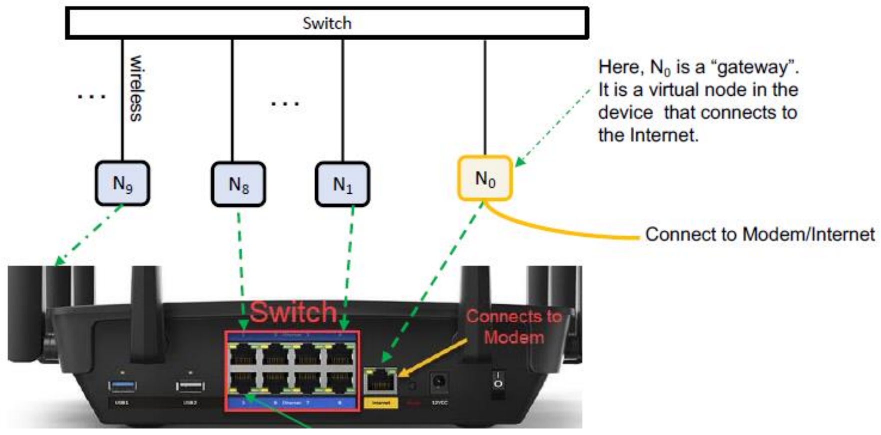

(1) What is the ip address of the domain name
www.comp.nus.edu.sg
QID=6A
Alice
(2) The answer is 100.10.10.3
QID=6A
Attacker
DNS server
(3) The answer is 137.132.80.57
QID=6A

Step (1) Alice asks for the address.
Step (2) Attacker sniffs and knows about it. Attacker quickly spoofs a reply with the same QID.
Step (3) DNS server also sends a reply. Since Attacker is closer to Alice, Attacker's reply likely to reach Alice first.
Alice takes the first reply as answer, and connect to 100.10.10.3

- DNS is an important component as it resolves the domain name. Hence, an DNS server can be a "**single-point-of-failure**" of the network.
- A **denial of service (DOS)** attack on a web service, instead of directly attacking the webserver could conduct DOS to the DNS server instead. When the DNS server is downed, the web service is no longer reachable.

ARP Poisoning

A switch connects a few nodes. (The switch handles mac-addresses, router handles ip-addresses).


Switch
wireless
. . .
. . .
N_9
N_8
N_1
N_0
Here, N_0 is a "gateway". It is a virtual node in the device that connects to the Internet.
Connect to Modem/Internet
Switch
Connects to Modem

*our home wifi "router", although often called as a "router", it also functions as a switch.

- The role of the switch is to connect two "ports".
- Switch deals with mac-addresses. It does not understand IP addresses.
- The switch keeps a table that associates the port to the mac-addresses.
- Resolution of IP address to MAC address is done by the nodes. Each node keeps a table that associate IP-address to MAC-address.
- The nodes update each others using some protocols, regarding info on the ip address and mac addresses.

ARP Poisoning is an attack that modifies (aka poisons) the tables so as to gain MITM access.

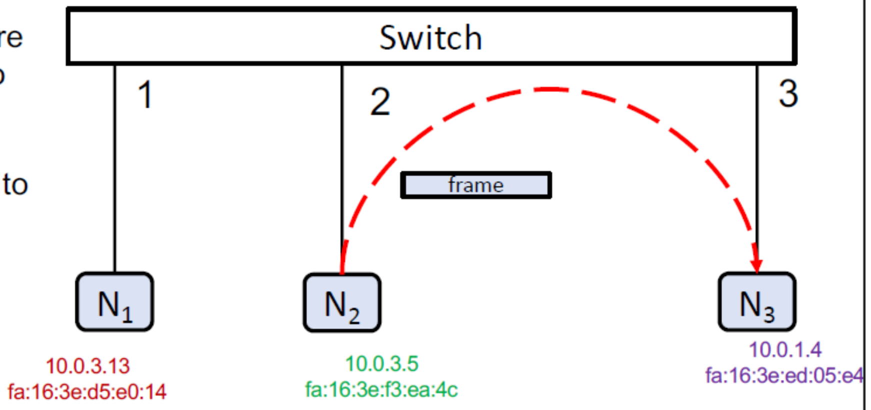
When N_2 want to send to ip address 10.0.1.4

T0

Switch's port	Mac-address
1	fa:16:3e:d5:e0:14
2	fa:16:3e:f3:ea:4c
3	fa:16:3e:ed:05:e4

Under normal circumstances, these are carried out when N_2 sends a packet to 10.0.1.4

1. N_2 looks up the table **T2**. Resolve to fa:16:3e:ed:05:e4
2. N_2 sends the frame to the switch, specifying destination fa:16:3e:ed:05:e4
3. Switch looks up the table **T0**, redirect the frame to port 3.



Ip-address	Mac-address
10.0.3.13	fa:16:3e:d5:e0:14
10.0.1.4	fa:16:3e:ed:05:e4

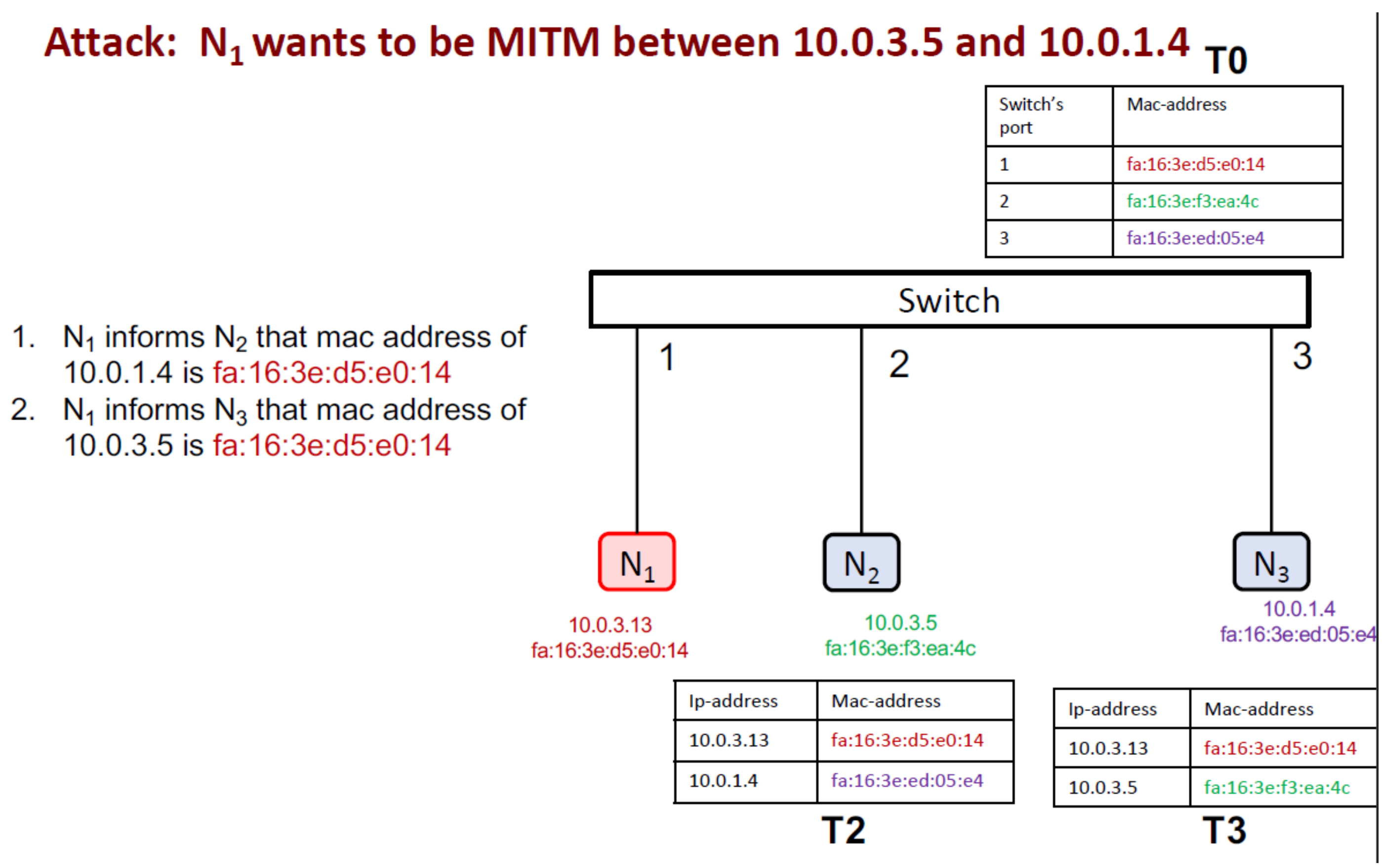
T2

Ip-address	Mac-address
10.0.3.13	fa:16:3e:d5:e0:14
10.0.3.5	fa:16:3e:f3:ea:4c

T3

Note:

- *T0 stored in switch.*
- *T2 stored in N_2*
- *T3 stored in N_3*



Attack: N_1 wants to be MITM between 10.0.3.5 and 10.0.1.4

1. N_1 informs N_2 that mac address of 10.0.1.4 is fa:16:3e:d5:e0:14
2. N_1 informs N_3 that mac address of 10.0.3.5 is fa:16:3e:d5:e0:14

T0

Switch's port	Mac-address
1	fa:16:3e:d5:e0:14
2	fa:16:3e:f3:ea:4c
3	fa:16:3e:ed:05:e4

Ip-address	Mac-address
10.0.3.13	fa:16:3e:d5:e0:14
10.0.1.4	fa:16:3e:ed:05:e4

T2

Ip-address	Mac-address
10.0.3.13	fa:16:3e:d5:e0:14
10.0.3.5	fa:16:3e:f3:ea:4c

T3

- After the tables are poisoned, all frames will be sent to N1.
- N1 can relay the frames, or modify the frames before relaying.
- Hence N1 become the MITM in layer 2.

5.3 Denial of Service (DOS) Attack

DOS is an attack on availability.

Availability: The property of being accessible and usable upon demand by an authorized entity.

Denial of service (DOS): The prevention of authorized access to resources or the delaying of time-critical operations.

Example of DOS attack:

- Simply flood a web-server with http requests.
- Sending large number of DNS requests to DNS server

When DOS is carried out by a large number of attackers, this is called **DDOS: Distributed Denial of Service**.

Reflection and Amplification Attack

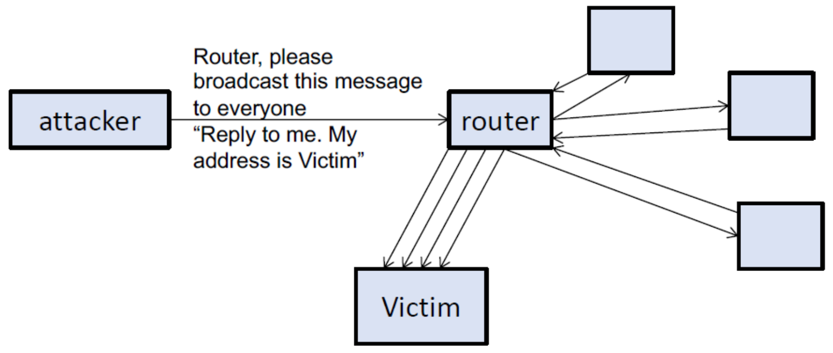
Reflection attack is a type of DOS in which the attackers send requests to intermediate nodes, which in turn send overwhelming traffic to the victim.

- Indirect, and thus more difficult to trace
- Very often, the refelcted traffic could be amplified (a single reqwuest could trigger mutiple responses from the intermedaite nodes). Hence these are also known as Amplification attacks.

Example of Reflection attacks: ICMP/Smurf Flood

- Step 1. An attacker sends the request "ICMP PING" to a router, instructing the router to broadcast this request. The source ip-address of this request is spoofed with the victim ip address.
- Step 2. The router broadcasts this request.
- Step 3. Each entity who has received this request, replies to it by sending an "echo reply" to the source, which is the victim.

The victim's network is overwhelmed with the "echo reply".


attacker
Router, please broadcast this message to everyone
"Reply to me. My address is Victim"
router
Victim

Attack is no longer effective. Most router now configured not to broadcast.

To prevent attack, simply disable the feature.

Other Reflection Attack

• DNS reflection attack.

"During a DNS amplification attack, the perpetrator sends out a DNS query with a forged IP address (the victim's) to an open DNS resolver, prompting it to reply back to that address with a DNS response. With numerous fake queries being sent out, and with several DNS resolvers replying back simultaneously, the victim's network can easily be overwhelmed by the sheer number of DNS responses."

Botnet

- A **bot** (aka zombie) is a compromised machine
- A **botnet** (aka zombie army) is a large collection of connected boths, communication via covert channels.
- A botnet has a command-and-control mechanism, and thus can be controlled by an individual to carry out DDOS.
- Possible usages of a botnet:
    - DDOS flooding, vulnerability scanning, anonymizing HTTP proxy, email address harvesting, cipher breaking

5.4 Useful Tools

Wireshark (Packet Analyzer)

Wireshark listens to "interactions" between the OS and the network card driver. (In other words, it is a MITM between OS and network card).

Hence, header added by the network card, or modification made by the network card, may not be captured by Wireshark. This depends on the
OS and the hardware.

*Typically capture in the Datalink Layer

Nmap (Port Scanning)

• When a port is "***open"***, there exist such a process running in the server. When a port is "***closed***", no process is listening to that port.

• If a port is ***"closed",*** attacker is unable to feed malicious data to that port.

Port scanning: The process of determining which ports are open in a network.
Port scanner: A tool for port scanning. E.g. Nmap.

Port scanner is a useful tool for attacker, and network administrator to scan for vulnerabilities.

5.5 Protection: Securing the communication channel using cryptography

The well-known TLS/SSL, WPA, IPSEC protect different layers.

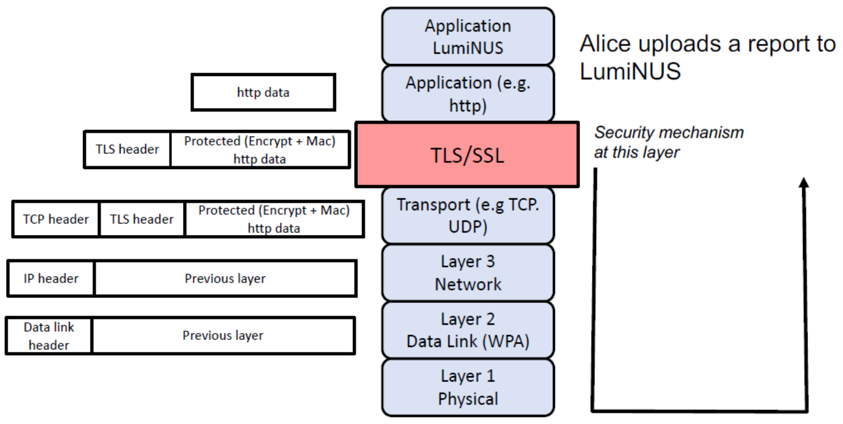
Very often, when referring to a security protocol, we indicate the "layer" the protocol targets to protect. (complication: some protections span across multiple layers, or do not provide full protection of the targeted layer.)

When analyzing an attack, it is also insightful to figure out at what layer the
attacker resides. (complication: likewise, some attacks span across multiple layers. In such situations, trying hard to pin-point the layer could sometime be very confusing).

A security protocol that protects layer k, would protect information in that layer and above.

Hence if an attacker resides at layer 1, and there is a security protocol that protect layer 3, then information generated in layer 3 and above will be protected, but information in layer 2 would not be protected.

SSL/TLS



The receiver end-point decrypts the received data at the corresponding layer.

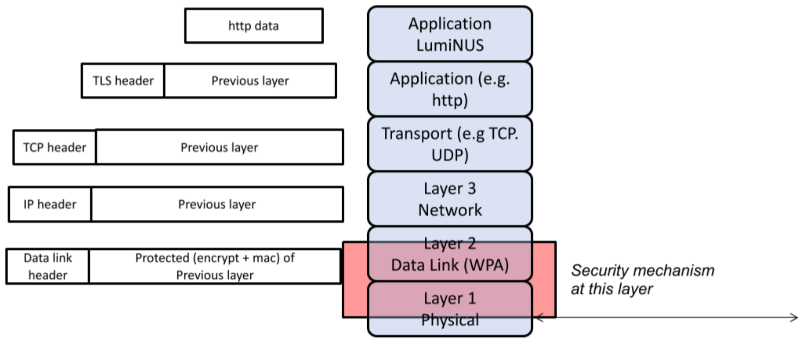
The SSL/TLS sit on top of the **Transport** Layer.

We can imagine that, when an application (say browser or email agent) wants to send data to the other end point, it first passes the data and the address (ip address) to SSL/TLS. Next SSL/TLS first "protects" the data using encryption (confidentiality) and mac (authenticity), and then instructs the transport layer to send the protected data.

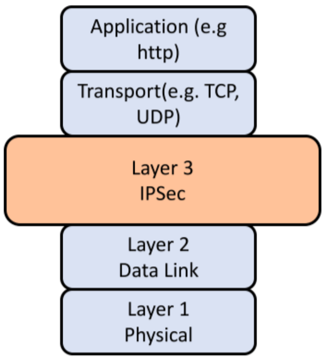
WPA2

Wifi Protected Access II (WPA2)

WPA2 provides protection at layer 2 (Link) and layer 1 (Physical). Not all information in layer 2 are protected.



IPSec



E.g. IPSec provides "integrity/authenticity" protection of ip-address, but not confidentiality. Hence, attackers are unable to "spoof" the source ip-source, but can learn the source and destination ip-address of the sniffed packets.

IPSec is a mechanism whose goal is to protect the IP layer.

5.6 Firewall

Consider the computer network in an organization.

- **Some nodes contain more sensitive information than the other.**
  Ex: Student exam records vs public workstations
- **Some nodes might have unpatched vulnerabilities.**
  Ex: When a patch is available, it might take some time to patch all the system.
- **Certain nodes might be compromised**
- **Certain protocols do not have protection mechanism, or only light-weight protection.**
  Ex: DNS, ARP, Network Printer, etc. Prevent attackers from accessing such protocol

We need to divide the computer network into segments and deny unnecessary access. (**Principle of least privilege, compartmentalization**).

Firewall, Intrusion Detection System (IDS) are tools to control access to the network.

Principle of least privilege

"The principle of least privilege (PoLP, also known as the principle of minimal privilege or the principle of least authority) requires that in a particular abstraction layer of a computing environment, **every module** (such as a process, a user, or a program, depending on the subject**) must be able to access only the information and resources that are necessary for its legitimate purpose**"

Compartmentalization
Generally, it refers to the notion of confining information within compartments.

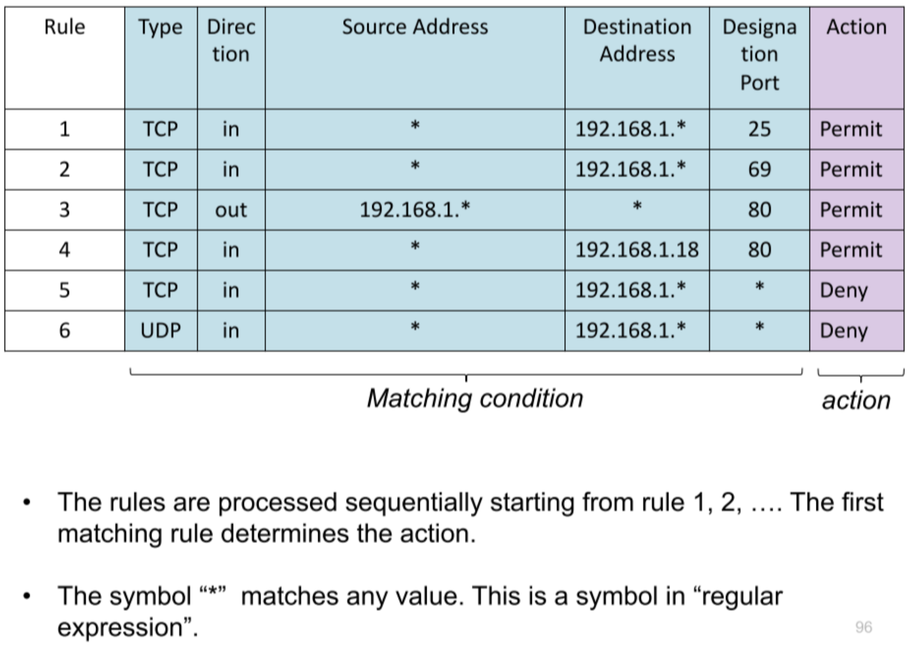
Firewall
A Firewall controls what traffic is allowed to enter the network (ingress filtering) or leave the network (egress filtering). Firewall are devices or programs that control the flow of network traffic between networks or hosts that employ differing security postures.

DMZ: Demilitarized zone
A sub-network that exposes the organization's external service to the (untrusted) Internet.

Packet Filtering/Screening
Firewall's controls are achieved by "packets filtering" (aka screening). Filtering may occur in router, gateway/bridge, host, etc.

Packet filtering inspects every packet, typically only on the TCP/IP packet's header information (network & transport layer). If the payload is inspected, we call it **deep packet inspection (DPI).** Action taken after inspection could be:
- Allow the packet to pass
- Drop the packet
- Reject the packet (drop and inform sender)
- Log info
- Notifys system admin
- Modify the packet (for more advanced device)

Firewall Rules
- Drop packets with "source ip-address" not within the organization's network. (Stop attacks originated within the network)
- Whitelist
  - Drop all packets except those specified in the white-list. (Eg. Drop all except http, email, DNS)
- Blacklist
  - Accept all packets except those in specified in the blacklist (E.g. Allow https, except ip address in the blacklist)



Rule	Type	Direction	Source Address	Destination Address	Designation Port	Action
1	TCP	in	*	192.168.1.*	25	Permit
2	TCP	in	*	192.168.1.*	69	Permit
3	TCP	out	192.168.1.*	*	80	Permit
4	TCP	in	*	192.168.1.18	80	Permit
5	TCP	in	*	192.168.1.*	*	Deny
6	UDP	in	*	192.168.1.*	*	Deny

Matching condition *action*

- The rules are processed sequentially starting from rule 1, 2, …. The first matching rule determines the action.
- The symbol "*" matches any value. This is a symbol in "regular expression".

96

Firewall Design

A Firewall enforces a set of rules provided by the network administrator.

Example on the 2-firewall setting:
Rules for Firewall1

- Block HTTP
- Allow internal to Mail Server: SMTP, POP3

Rules ofr Firewall2

- Allow from anywhere to Mail Server: SMTP

Types of Firewall

NIST's document (NIST 800-41) groups the firewalls into 3 types:

1. Packet filters	(Inspect packet header)
2. Stateful Inspection	(Deep packet inspection)
3. Proxy	(Modify packets)

Intrusion Detection System (IDS)

An IDS system consists of a set of "sensors" who gather data. Sensors could be in the host, or network router. The data are analyzed for intrustion.

Attack Signature Detection

The attack has specific, well-defined signature. For e.g. using certain port number, certain source IP address.

Anomaly Detection

The IDS attempt to detect abnormal pattern. For e.g. a sudden surge of packets with certain port number.

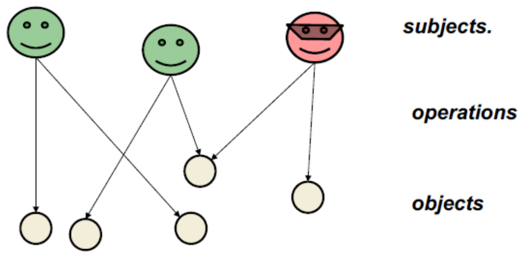
Behavior-based IDS

Can be viewed as a type of anomaly detection that focuses on human behavior. For e.g. The system might keep the profile of each user. It then tries to detect any user who deviates from the profile (e.g. start to download large files).

Popular open-source IDS: Snort
Other Tools: TLS/SSL Scan

Lecture 6 – Access Control

6.1 Access Control model



Access control model is relevant in many systems. E.g.

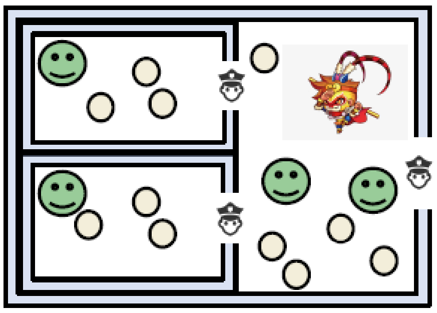
- File system (which files can a user read),
- LumiNUS (who can upload files to the workbin),
- Facebook (which posts can a user read), etc

We want to restrict ***operations*** on ***objects*** by ***subjects.***

• IT and computer system handles resources such as files, printers, network, etc. Certain resources can only be accessed by certain entities. Access control is about controlling such accesses. Different application have different requirements. Generally, it is about "selective restriction of access to a place or other resource" (wiki). An access control system specifies and enforces such restriction on the subject, objects and actions.

• E.g. OS, Social media (e.g. Facebook), documents in an organization (document classified as "restricted", "confidential", "secret" etc), physical access to different part of the building.

• Access control provides security perimeter which in turn facilitates segregation of accesses. Such segregation confines and localize damage caused by attacks.



<u>Security Perimeter</u>

Access control forms security perimeter/boundary.

With the boundary, malicious activities (e.g. malware) outside of the boundary would not affect resources within the perimeter. Furthermore, malicious activities within the boundary stays within the boundary.

Design of the boundary is guided by

• Principle of least privilege
• Compartmentalization
• Defence in depth
• Segregation of duties

<u>Definitions: Principal/Subject, Operation, Object</u>

$$\begin{tikzcd}\boxed{\text{Principal}} \arrow[r] & \boxed{\text{Do operation}} \arrow[r] & \boxed{\text{Reference monitor}} \arrow[r] & \boxed{\text{object}}\end{tikzcd}$$

A ***principal*** (or ***subject***) wants to access an ***object*** with some ***operation***. The ***reference monitor*** either grants or denies the access.

<u>Principals vs Subjects:</u>

- ***Principals***: the human users.
- ***Subjects***: The entities in the system that operate on behalf of the principals.

<u>Accesses to objects can be classified to the following:</u>

- ***Observe***: e.g. Reading a file. (Luminus, downloading a file from workbin)
- ***Alter***: e.g. writing a file, deleting a file, changing properties.(Luminus, uploading a file to the workbin).
- ***Action***: e.g executing a program.

E.g.
LumiNUS:
• a **student(principal)** wants to **submit(action)** a **forum post(object)**.
• a **TA** wants to **read** the **grade of student in another group**.

File system:
• a **user** wants to **delete** a **file**.
• a **user** wants to **change the mode** of a **file** so that it can be read by another user Bob.

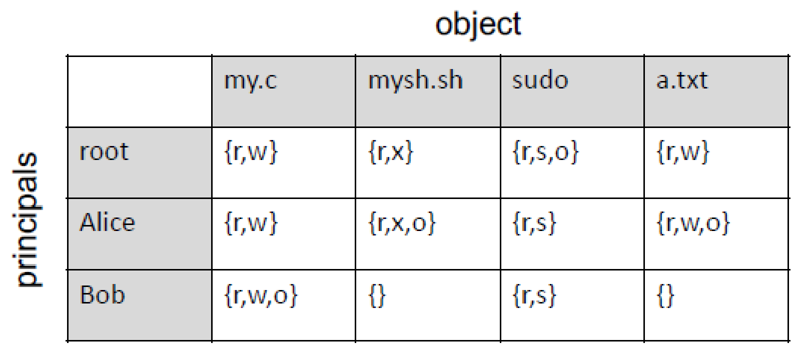
OS:
• An **app** (e.g. touch light) wants **access** to the **phone**.
• An **app** wants to **read files** generated by another app.

<u>Definitions: Ownership</u>
Every object has an "owner". Who decides the access rights to an object?

There are two options:
(1) The *owner* of the object decides the rights. (known as ***discretionary access control***)
(2) A system-wide policy decides. (known as ***mandatory access control)***.

Mandatory access control are strict rules that everyone must follow.

6.1 Access Control Matrix
How do we specific the access right of a particular principal to a particular object? Using a table.



object

principals

	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}	{}

Although the above *access control matrix* can specify the access right for all pairs of principals and objects, the table would be very large, and thus different to manage. Hence, it is seldom explicitly stored.

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

<u>Access Control List (ACL) & Capabilities</u>
The access control matrix can be represented in two different ways: ACL or capabilities.

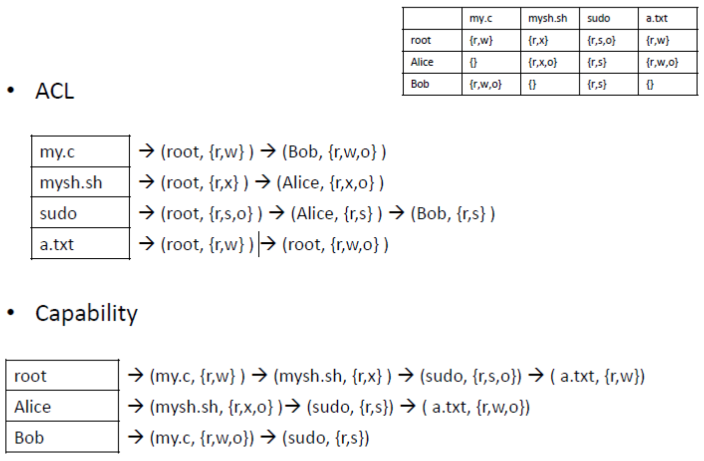
ACL: An ACL stores the access rights to an object as a list.

Capabilities: A subject is given a list of capabilities, where each capability is the access rights to an object.
"a capability is an unforgeable token that gives the possessor certain rights to an object"

Question: Does Unix file system adopt ACL or capabilities?
Answer: **ACL**

For ACL, it is difficult to obtain the list of objects a particular subject has access to. Conversely, for capabilities, it is difficult to get the list of subjects who have access to a particular object.

(to illustrate, in unix, suppose the system admin wants to generate the list of files that user alice0012 has “r” access to. How to quickly generate this list?)



	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

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})

- Capability

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})

6.3 Intermediate Control

We want an intermediate control that is ***fine grain*** (e.g. in Facebook, allow user to specify which friend can view a particular photo) and yet ***easy to manage***.

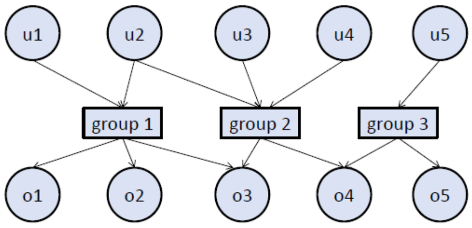
Either ACL or capabilities, the lists can still to be too large to manage. Also, it is not practical for an user to specific each single entries in the access control matrix. So, we need some ways to simplify the representation. One method is to “group” the subjects/objects and define the access rights on the group. This is called intermediate control. There are many ways to group them.

```
--w-r--r--   1 alice  staff       3 Mar 13 00:27 temp
```

In Unix file permission, subjects are divided into groups. Unix file permission uses ACL.
For each object, the owner specific the rights for

- *owner*
- *group*
- *world (everyone)*

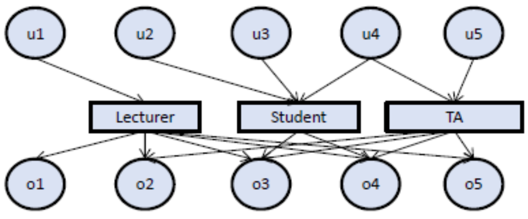
In LumiNUS, project groups can be created by lecturer. Objects created in a group can only be read by members in the group + lecturer.



In Unix, groups can only be created by root. The groups information is stored in the file `/etc/group`

<u>Intermediate control: Role-based access control</u>

The grouping can be determined by the “role” of the subject. A role associates with a collection of procedures. In order to carry out these procedures, access rights to certain objects are required.

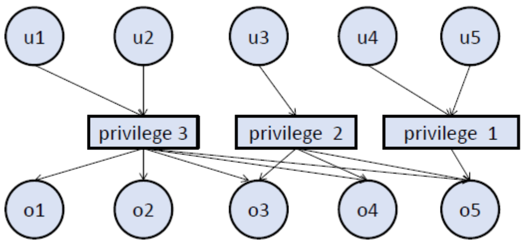


E.g. In LumiNUS, there are predefined rights for different roles: “Lecturer”, “TA” and “Student”.
When Alice enrolled to CS2107 as student, her rights are inherited from the role “Student”. When Alice volunteered as TA in CS1010, her rights are inherited from the role “TA” in CS1010. To design the access right of a role, we should follow the ***least privilege principle,*** i.e. access rights that are not required to complete the role will not be assigned.

e.g. Consider Luminus’s gradebook. The task of a student Teaching Assistance include entering the grade for each students. So we should give the TA “write” access to the gradebook so that the TAs can complete their task. Should we give the TAs the right to delete a gradebook? Since this access right is not required for the TA to complete their task, by the ***least privilege principle,*** the TA should not be given this access right.

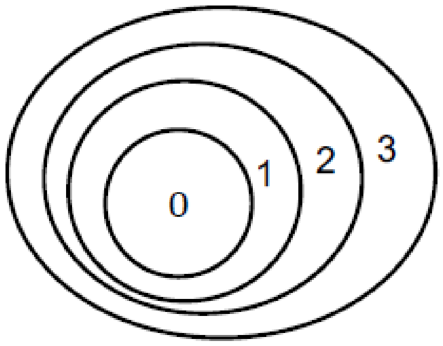
Intermediate Control: Privileges

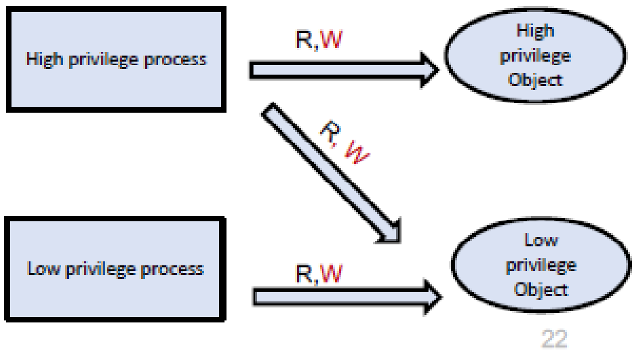
We sometime use the term *privilege* to describe the access right. Privilege can also be viewed as an intermediate control. It can be represented by a number, e.g. 1,2,3. if a subject can access an object, another subject with higher privilege can also access the object.



Intermediate control: Protection rings

e.g.





In OS, "privilege" is often called protection rings. They are the same but with different name. Here, each object (data) and subject (process) is assigned a number. Whether a subject can access an object is determined by their respective assigned number. Object with smaller number are more important. If a process is assigned a number *i*, we say that the process runs in ring *i*. We call processes with lower ring number as having "***higher privilege***". A subject cannot access (both read/write) an object with smaller ring number. **Unix has only 2 rings, *superuser* and *user*.**

Two Examples of intermediate control: Bell-LaPadula vs Biba

In protection rings, the subjects can have read/write access to objects that are classified with the same or lower privilege. Are there reasonable alternatives?

Here are two well-known models: **Bell-LaPadula** and **Biba**. Although they are rarely implemented as-it-is in computer system, they serve as a good guideline.

In both models, objects and subjects are divided into linear levels.
Level 0, level 1, level 2, ... higher level corresponds to higher "security".
(The numbering is opposite of protection ring. This is just a different choice of notations).

Bell-LaPadula Model (for data confidentiality)

Restrictions imposed by the Bell-Lapadula Model:

• **no read up**: A subject has only read access to objects whose security level is below the subject's current clearance level. This prevents a subject from getting access to information available in security levels higher than its current clearance level.

• **no write down**: A subject has append (can add but not delete or modify) access to objects whose security level is higher than its current clearance level. This prevents a subject from passing information to levels lower than its current level.
(e.g. a clerk working in the highly classified department should not gossip with other staff, in order to prevent leakage).

For "Confidentiality"
(A subject can append to objects at higher security level. Is it possible that, by appending to an object, one could distort its original content? Yes. See e.g. in renegotiation attack.)

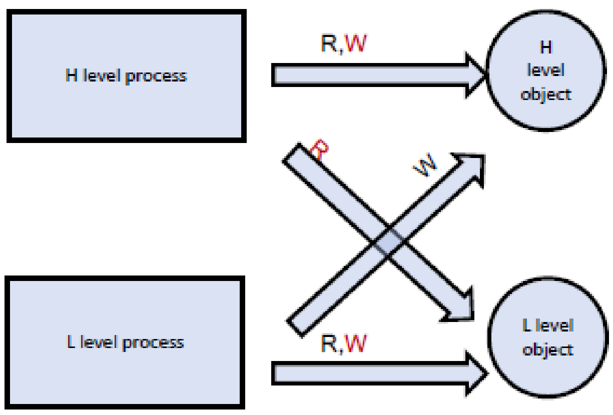
Biba Model (for process integrity)
Restrictions imposed by the Biba Model:

• **no write up**: A subject has only write access to objects whose security level is below the subject's current clearance level. This prevents a subject from compromising the integrity of objects with security levels higher than its current clearance level.

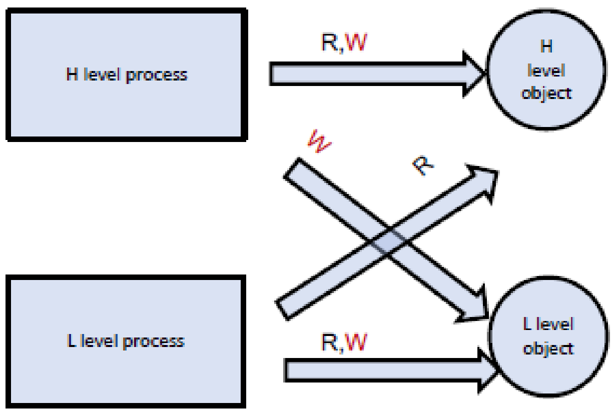
• **no read down**: A subject has only read access to objects whose security level is higher than its current clearance level. This prevents a subject from reading forged information from levels lower than its current level.

For "Integrity"
If a model imposes both Biba and Bell-LaPadula, subjects can only read/write to objects in the same level (not practical).

Direction of Information Flow



Bell-LaPadula (confidentiality)
No information coming down



Biba (Integrity)
No information going up.

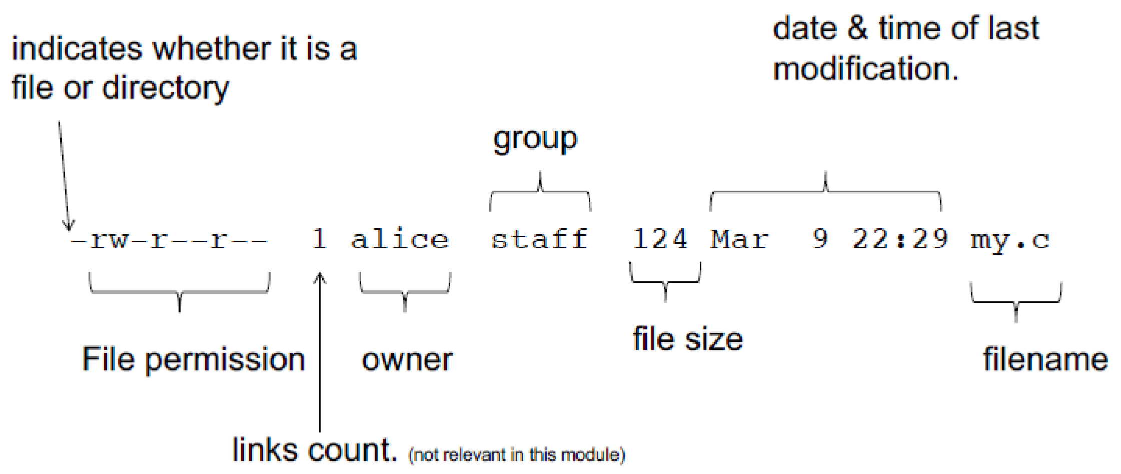
6.4 Unix File System

In Unix, objects includes files, directories, memory devices and I/O devices. All these resources are treated as files.
http://en.wikipedia.org/wiki/File_system_permissions

```
%ls -al
-r-s--x--x 1 root    wheel 164560    Sep 10 2014     sudo
-rwxr-xr-x 2 root    wheel 18608     Nov 7 06:32     sum
-rw-r--r-- 1 alice   staff 124       Mar 9 22:29     myprog.c
lr-xr-xr-x 1 root    wheel 0         Mar 12 16:29    stdin
```

Question: what are the files in the following directories?
/dev
/dev/stdin
/dev/stdout
/dev/urandom

File system permission


indicates whether it is a file or directory
date & time of last modification.
group
-rw-r--r-- 1 alice staff 124 Mar 9 22:29 my.c
File permission
owner
file size
filename
links count. (not relevant in this module)

The file permission are grouped into 3 triples, that define the *read*, *write*, *execute* access for ***owner, group, other (also called the "world").***

A '-' indicates access not granted. Otherwise

r:	read
w:	write (including delete)
x:	execute (s: allow user to execute with the permission of the owner)

Permissions

Unix-like systems implement three specific permissions that apply to each class:

- The *read* permission grants the ability to read a file. When set for a directory, this permission grants the ability to read the **names** of files in the directory, but not to find out any further information about them such as contents, file type, size, ownership, permissions.
- The *write* permission grants the ability to modify a file. When set for a directory, this permission grants the ability to modify entries in the directory, which includes creating files, deleting files, and renaming files. Note that this requires that *execute* is also set; without it, the write permission is meaningless for directories.
- The *execute* permission grants the ability to execute a file. This permission must be set for executable programs, in order to allow the operating system to run them. When set for a directory, the execute permission is interpreted as the *search* permission: it grants the ability to access file contents and meta-information if its name is known, but not list files inside the directory, unless *read* is set also.

The effect of setting the permissions on a directory, rather than a file, is "one of the most frequently misunderstood file permission issues". When a permission is not set, the corresponding rights are denied. Unlike ACL-based systems, permissions on Unix-like systems are not inherited. Files created within a directory do not necessarily have the same permissions as that directory.

Principals, Subjects

• Principals are ***user-identities*** **(UIDs)** and ***group-identities*** **(GIDs)**

• Information of the user accounts are stored in the "password" file

```
/etc/passwd
root:*:0:0:System Administrator:/var/root:/bin/sh
```

• The subjects are processes. Each process has a process ID (PID). (e.g. in Unix, the commend ps –alx display a list of processes).

Remarks on the Password file

• The file is made world-readable because some information in /etc/passwd is needed by non-root program. *In earlier version of Unix*, the "*" in the file was the hashed password H(pw), where H() is some cryptographic hash, and pw the password of the user. Hence, previously all users can see the hashed passwords of others.

• With knowledge of the hashed password, the attackers can carry out offline dictionary attack. To prevent that, it is now replaced as "*", and the actual hashed passwords are stored somewhere else and not world-readable. (You may wonder why can't the we completely remove the field, since putting a * there is redundant. This is for backward compatibility. If the field is completely removed, some existing programs might not work. Artifact of patching and very irritating.)

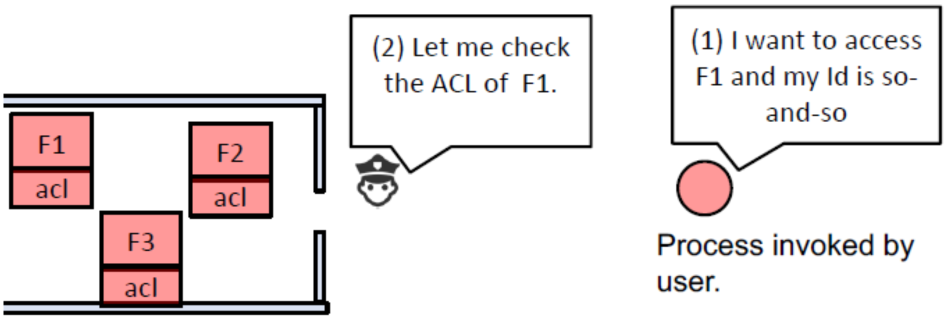
superuser(root)

• A special user is the **superuser**, with **UID 0** and usually with the **username root**. All security checks are turned off for root.

(Unix's protection rings consists of 2 rings: superuser, user)

Unix's Access Control and Reference Monitor

The objects are files. Recall that each file is associated with a 9-bit permission. Each file is owned by a user, and a group.


F1
acl
F2
acl
F3
acl
(2) Let me check the ACL of F1.
(1) I want to access F1 and my Id is so-and-so
Process invoked by user.

Checking Rules

When a user (subject) wants to access a file (object), the following are checked, in the following order:

1. If the user is the owner, the permission bits for ***owner*** decide the access rights.
2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for ***group*** decide the access rights.
3. If the user is not the owner, nor member of the group that own the file, then the permission bits for ***other*** decide. The owner of a file, or superuser can change the permission bits.

6.5 Controlled Invocation & Privilege Elevation

Two separate environments ensure segregation for security. However, to be useful, we need interaction between them. Solution: no free-flow of interactions but "controlled" interactions.

Controlled invocation

Eg in Unix: Some sensitive resources (such as network port 0 to 1023, printer) should be accessible only by the superuser. However, users sometime need those resources.

Eg: Consider a file **F** that contains home addresses of all staffs. Clearly, we cannot grant any user to read **F**. However, we must allow a user to read/modify his/her address, and thus need to make it readable/writeable to that user. The polarized setting where either a process can read or cannot read a file would get stuck!

Solution: ***controlled invocation***.

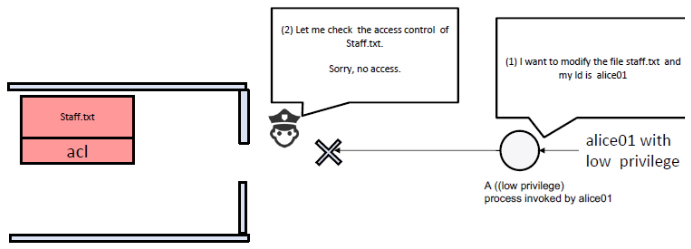
• The system provides a predefined set of applications that have access to **F.**

• These application is granted "elevated privilege" so that they can freely access the file, and any user can invoke the application. Now, any user can access **F** via the application.

• The programmer who write the application bear the responsibility to make sure that the application only performed intended limited operation. In other words, the user stay within the planned boundary when using the application.

Without controlled escalation

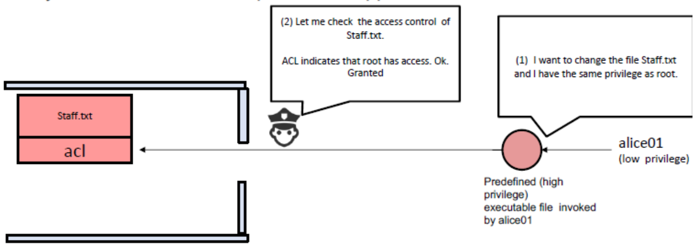
alice01 doesn't has access right to Staff.txt.
Any processes (subject) invoked by alice01 inherit alice01's right


(2) Let me check the access control of Staff.txt.

Sorry, no access.
(1) I want to modify the file staff.txt and my Id is alice01
Staff.txt
acl
alice01 with low privilege
A ((low privilege) process invoked by alice01

With Controlled escalation

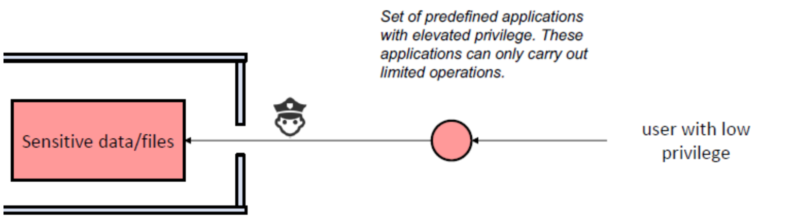
There is a set of predefined applications with "elevated" privilege. A normal user alice01 can't create applications with high privilege. However, any user can invoke these predefined applications.


(2) Let me check the access control of Staff.txt.

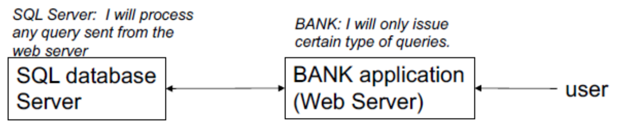
ACL indicates that root has access. Ok. Granted
(1) I want to change the file Staff.txt and I have the same privilege as root.
Staff.txt
acl
alice01 (low privilege)
Predefined (high privilege) executable file invoked by alice01

Bridges with Elevated Privilege

We can view the predefined application as a predefined "bridges" for the user to access sensitive data. (note that the "bridge" can only be built by the system.)


Set of predefined applications with elevated privilege. These applications can only carry out limited operations.
Sensitive data/files
user with low privilege

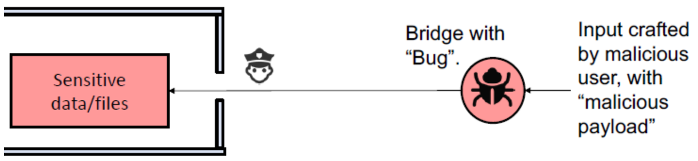
Firewall ensure that users can't directly send query to the SQL database server. Users can indirectly send query to SQL server via website. In this way, only certain types of queries can be sent by the users.

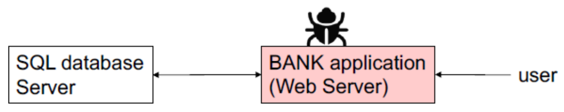

SQL Server: I will process any query sent from the web server
BANK: I will only issue certain type of queries.
SQL database Server
BANK application (Web Server)
user

Suppose a "bridge" is not implemented correctly and contains exploitable vulnerabilities. In some vulnerabilities, an attacker can trick the bridge to perform "illegal" operations not expected by the programmer/designer. This would have serious implication, since the process is now running with "***elevated privilege***".

Attacks of such form is also known as "***privilege escalation***". *"**Privilege escalation** is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to **gain elevated access** to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions."*

Bugs in bridge leads to Privilege escalation.





A bug in BANK that allows the user to send in arbitrary query. (SQL injection)

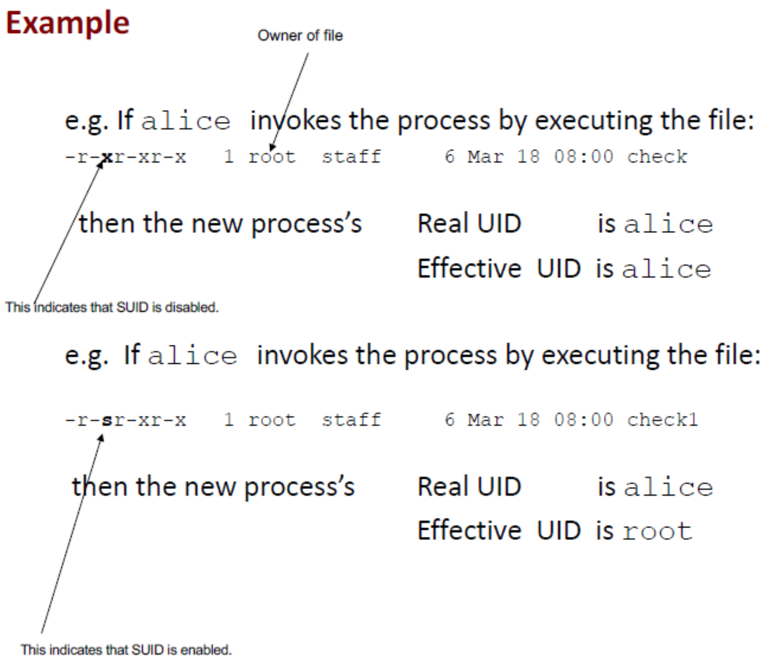
6.6 Controlled invocation in UNIX

Definitions: Process and Set userID (SUID)

- A process is a subject.
- A process has an identification (PID). New process can be created by executing a file or by "forking" an existing process.
- A process is associated with a ***Real UID*** and an ***Effective UID***.
- The ***real UID* is inherited from the user who invokes the process**. For e.g. if the user is alice, then the real UID is alice.

Processes can be created by executing a file. Each executable file has a SUID flag. There are two cases:

- If the **Set User ID (SUID) is disabled** (the permission will be displayed as "x"), then the process' ***effective UID* is same as real UID**.
- If the **Set User ID (SUID) is enabled** (the permission will be displayed as "s"), then the process' ***effective UID* is inherited from the UID of the *file's owner***.


Example
Owner of file
This indicates that SUID is disabled.
This indicates that SUID is enabled.

e.g. If alice invokes the process by executing the file:

```
-r-xr-xr-x    1 root   staff      6 Mar 18 08:00 check
```

then the new process's Real UID is alice

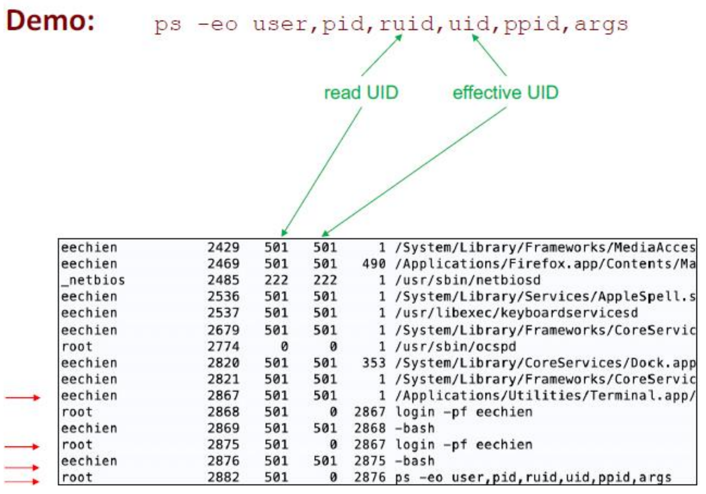
Effective UID is alice

e.g. If alice invokes the process by executing the file:

```
-r-sr-xr-x    1 root   staff      6 Mar 18 08:00 check1
```

then the new process's Real UID is alice

Effective UID is root



Demo:

```
ps -eo user,pid,ruid,uid,ppid,args
```

```
eechien          2429   501   501      1 /System/Library/Frameworks/MediaAcces
eechien          2469   501   501    490 /Applications/Firefox.app/Contents/Ma
_netbios         2485   222   222      1 /usr/sbin/netbiosd
eechien          2536   501   501      1 /System/Library/Services/AppleSpell.s
eechien          2537   501   501      1 /usr/libexec/keyboardservicesd
eechien          2679   501   501      1 /System/Library/Frameworks/CoreServic
root             2774     0     0      1 /usr/sbin/ocspd
eechien          2820   501   501    353 /System/Library/CoreServices/Dock.app
eechien          2821   501   501      1 /System/Library/Frameworks/CoreServic
eechien          2867   501   501      1 /Applications/Utilities/Terminal.app/
root             2868   501     0   2867 login -pf eechien
eechien          2869   501   501   2868 -bash
root             2875   501     0   2867 login -pf eechien
eechien          2876   501   501   2875 -bash
root             2882   501     0   2876 ps -eo user,pid,ruid,uid,ppid,args
```

When a process (subject) wants to read/write/execute a file (object)

When a process wants to access a file, the effective UID of the process is treated as the "subject" and checked against the file permission to decide whether it will be granted or denied access.

E.g consider a file own 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 is denied access to the file.

• If the effective UID of a process is root, then the process is allowed to access the file.

Use Case Scenario of "s" (SUID)

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 selfedit some fields (for e.g. postal address) of their own profile. Since the file permissible 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.

• *What can we do?*

Solution

Create an executable file editprofile owned by root:

```
-r-sr-xr-x 1 root staff 6 Mar 18 08:00 editprofile
```

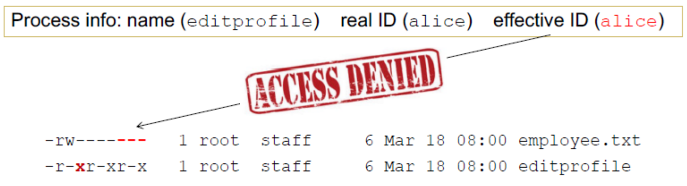
• The program is made world-executable so that *any* user can execute it.

• Furthermore, the permission is set to be “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**. Following the checking rule, this process can now read/write the file employee.txt

Summary: When SUID is Disabled

- If the user `alice` invokes the executable, the process will have its ***effective ID*** as `alice`
- When this process wants to read the file `employee.txt`, the OS (reference monitor) will deny the access

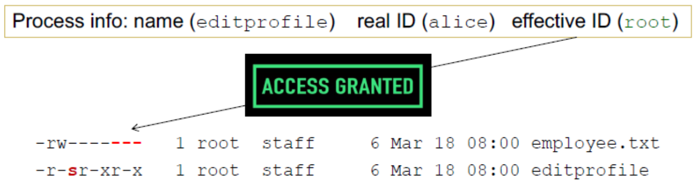


Process info: name (`editprofile`) real ID (`alice`) effective ID (`alice`)

```
-rw-------   1 root  staff    6 Mar 18 08:00 employee.txt
-r-xr-xr-x   1 root  staff    6 Mar 18 08:00 editprofile
```

Summary: When SUID is Enabled

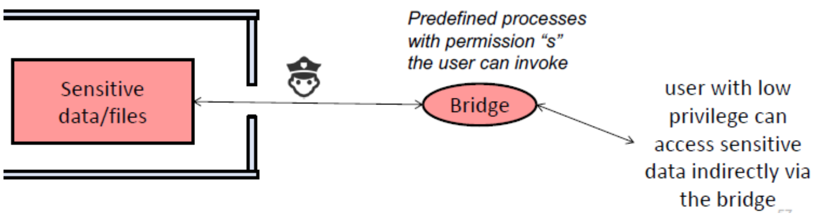
- But if the permission of the executable is “s” instead of “x”, then the invoked process will has `root` as its effective ID
- Hence the OS grants the process to read the file
- Now, the process invoked by `alice` can access `employee.txt`



Process info: name (`editprofile`) real ID (`alice`) effective ID (`root`)

```
-rw-------   1 root  staff    6 Mar 18 08:00 employee.txt
-r-sr-xr-x   1 root  staff    6 Mar 18 08:00 editprofile
```

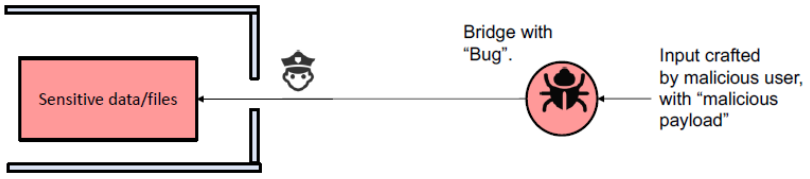
Elevated Privilege


Sensitive data/files
Predefined processes with permission “s” the user can invoke
Bridge
user with low privilege can access sensitive data indirectly via the bridge

In this example, the process editprofile is temporary ***elevated*** to superuser (i.e. root), so that it can access sensitive data. We can view the elevated process as the interfaces where a user can access the “sensitive” information.

- They are the predefined “bridges” for the user to access the data.
- The “bridge” can only be built by the root.

These bridges solve the problem. However, it is important that these “bridges” are correctly implemented and do not leak more than required. If the bridge is not built securely, there could be privilege escalation attack. This leads us to another topic: secure programming and software security.



Lecture 7 – Call Stack

Summary and takeaways

• Demonstrate how process ***integrity can be compromised by modifying values in memory***.
• Call stack facilitates function calls. It keeps track of different instances of function call. Without that, we won't have modular software design.
• An object in the stack is a "stack frame" and each frame contains info of the environment an instance of the function operates. Info include control flow info (return address), local variables, and parameters. Malicious modification of those info would have significant consequence (next lecture on stack smash).

7.2 Stack

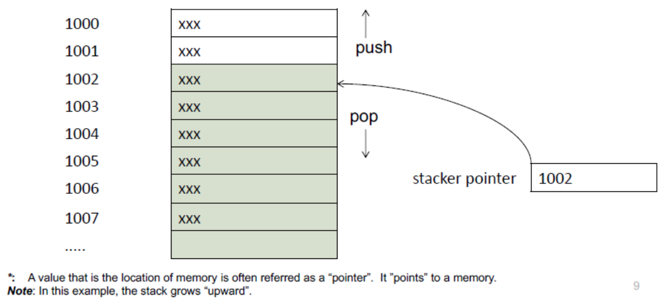
Functions

- Subroutine/function/procedure breaks code into smaller pieces (facilitiate modular design and code reuse)
- A function can be called from different part of the program, even recursively.

***Question* 1**: how does the system knows where it should return to after a function is is completed?
***Question* 2**: where are the function's *arguments* and *local variables* stored?
Answer: These are managed by the "call stack"

Stack

Stack is a data structure. It is not a separate hardware. A **call-stack** is a stack that store important information of the function calls. **Each element of the stack is called a stack frame**. ***Stack pointer*** is a variable that stores location* of the first element. There are two operations in stack: Push and Pop. (Last-In-First-Out).


1000
1001
1002
1003
1004
1005
1006
1007
.....
xxx
push
pop
stacker pointer
1002
*: A value that is the location of memory is often referred as a "pointer". It "points" to a memory.
Note: In this example, the stack grows "upward".
9

Call stack

• Recap that Stack is last-in-first-out. This makes it useful in maintaining function calls.
• During execution, a stack is maintained to keep control flow information and passing of parameters. This stack is known as ***Call stack.***
• For an instance of a function call, the call stack keep tracks of
  - Parameters to be passed
  - Control flow information: return addresses (i.e. who invoke this function)
  - Local variables of functions
• Each call/invocation of a function pushes an activation record (aka ***stack frame***) to the stack, which contains:
  - parameters,
  - return address,
  - local variables, and
  - pointer to the previous stack frame in the stack. *(this is included for efficiency. Conceptually, this piece of info is not required to maintain the control flow.)*

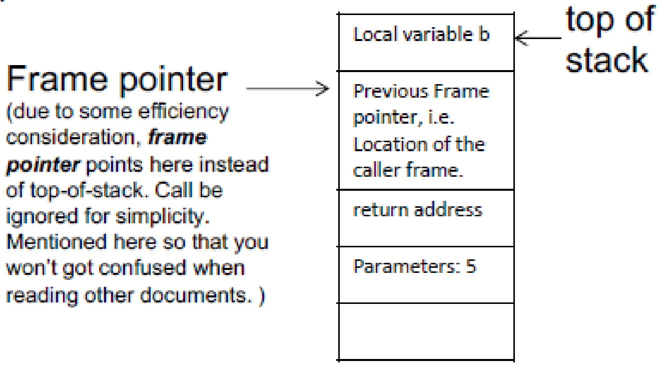
Illustration

When a function is called, the parameters, return address, local variables are "pushed" into the stack.

E.g. for the following segment of C program:

```
int Drawline( int a)
{int b =1;
..
}

int main()
{
   Drawline (5);
}
```


Frame pointer
(due to some efficiency consideration, ***frame pointer*** points here instead of top-of-stack. Call be ignored for simplicity. Mentioned here so that you won't got confused when reading other documents.)
Local variable b
Previous Frame pointer, i.e. Location of the caller frame.
return address
Parameters: 5
top of stack

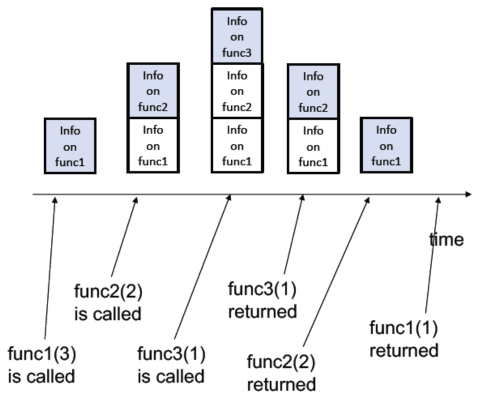
When the function `Drawline(5)` is invoked, the followings steps are carries out:
(1) These data are pushed into the stack in this order:

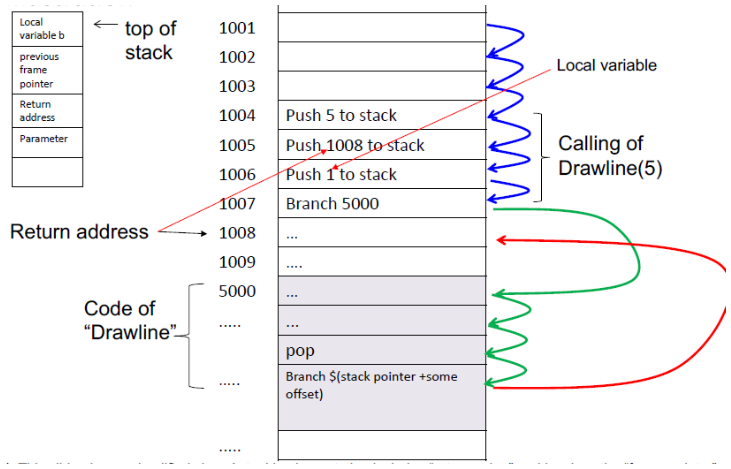
- **parameter** (which is 5),
- **return address** (i.e. value of program counter), and
- **value of the local variable b** (which is 1).

(2) The control flow branches to the code of "Drawline".
(3) Execute "Drawline".
(4) After "Drawline" is completed, pops out the variable, return address and parameter.
(5) Control flow branches to "return address".

```
int func1( int a)
{int b =253;
     func2( 2);
}
int func2( int a)
{int b =15;
     func3( 1);
}
int func3( int a)
{int b =0;
}


int main()
{
   func1(3);
}
```





7.3 Control Flow Integrity (Implications on security)

We have seen how the call stack stores the the *return address* as data in the memory.

- Code is stored as data.
- Attacker could compromise the execution integrity by either modifying the code or modifying the control flow, in particular, the return address in the call stack.

Compromising ***memory integrity -> control flow integrity***

Let's assume that the attacker has the capability to write to some memory (i.e. able to compromise ***memory integrity***). In the next few slide, we show that this can lead to compromise of the ***control flow integrity***.

- E.g., by exploiting buffer overflow (if the program is not carefully written and allow that), the attacker could write to certain memory that the attacker originally doesn't have access to.
- While possible, it is still not so easy for an attacker to compromise memory. In addition, it may come with some restrictions. For e.g. attackers can only write to some particular location, or the attacker can only write a sequence of consecutive bytes, or the attacker can write but not read, etc. In other words, although we assume that attacker can compromise memory, keep in mind that the attackers unlikely to have arbitrary access to any memory.

Possible Attack Mechanisms

Let us assume that the attacker has the capability to *write* to some memory locations and wants to compromise the execution integrity. The attacker could:

(*Attack 1*) Overwrite ***existing execution code portion*** with malicious code; or

(*Attack 2*) Overwrite a piece of control-flow information:

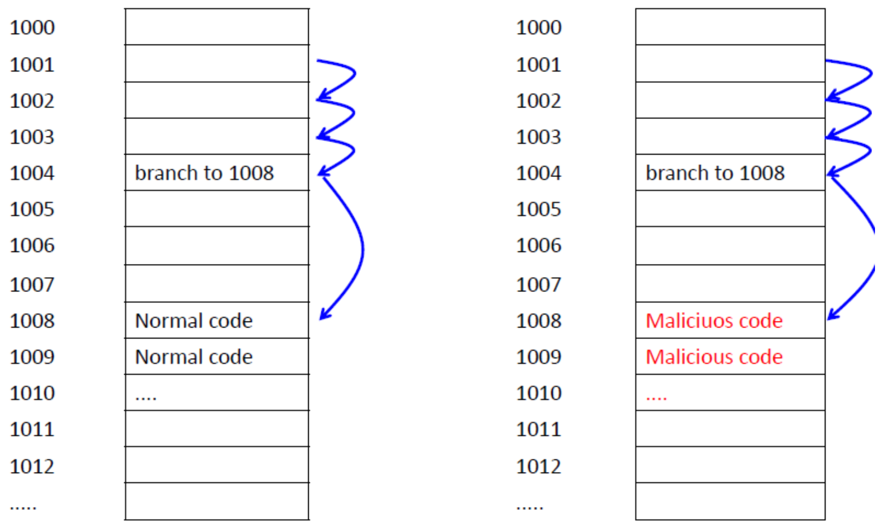
(2a) Replace a ***memory location*** storing a code address that is used by a *direct jump*

(*2b*) Replace a ***memory location*** storing a code address that is used by an *indirect jump*

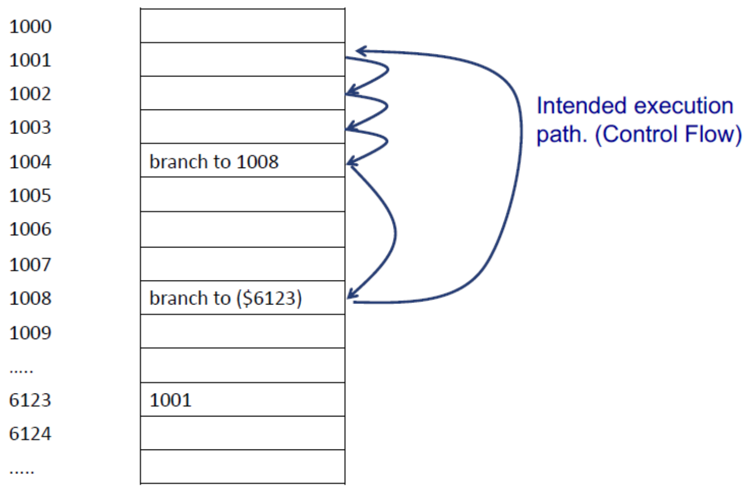
The above three attacks are illustrated in the next few slides.

When Attack (2b) can be carried out on stack, it is aka ***Stack smashing***.

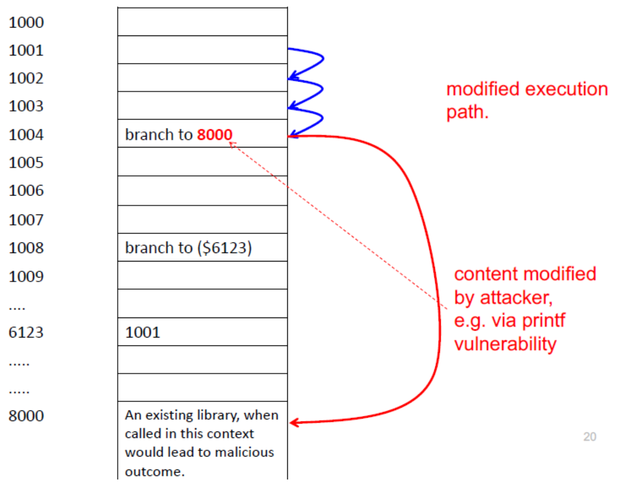
Attack 1 (replace the code)



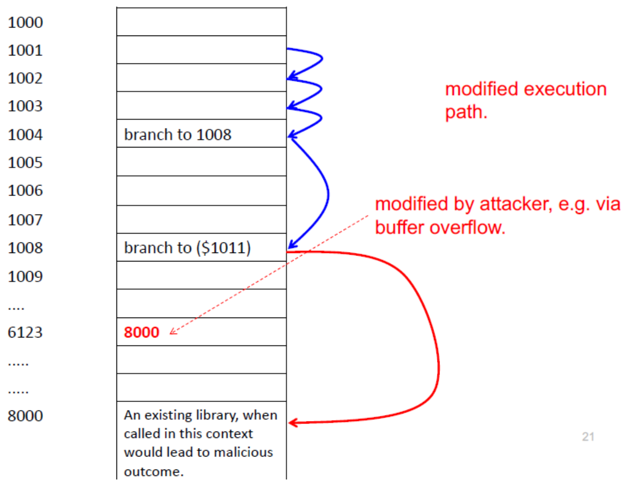
Attack 2a, 2b: Normal control flow before being attacked.



Attack 2a (Memory locations that store the code being modified)


1000
1001
1002
1003
1004
branch to 8000
1005
1006
1007
1008
branch to ($6123)
1009
....
6123
1001
.....
.....
8000
An existing library, when called in this context would lead to malicious outcome.
modified execution path.
content modified by attacker, e.g. via printf vulnerability

Attack 2b (Memory locations that store the addresses being modified)


1000
1001
1002
1003
1004
branch to 1008
1005
1006
1007
1008
branch to ($1011)
1009
....
6123
8000
.....
.....
8000
An existing library, when called in this context would lead to malicious outcome.
modified execution path.
modified by attacker, e.g. via buffer overflow.

Lecture 8 – Secure Programming

Program must be **correct**, **efficient**, **secure**.

• Many programs are not implemented properly, different from the original intents of the programmers. Under typical environment, the program may function normally since the conditions that trigger failure rarely occur. However, a malicious attacker may search for the conditions and intentionally trigger them.

• E.g. It is possible that due to some implementation bugs of a website, if the symbol "<" is entered as the first character of the "name" field, it could crash the webserver. The webserver could be running for very long without failure, since normally names don't have the symbol "<". However, if an attacker knows about it, the attacker can intentionally trigger it. Here, "**execution integrity**" is compromised.

• Many bugs are very simple and easy to correct when found. But program for complex system are large, making detecting such bug challenging. E.g. Window XP has 45 millions SLOC (source line of codes)

• Many bugs are specific to applications and difficult to classified. The book in the next slide classifies them into 24 types.

8.2 Unsafe function. printf()

Resources
http://en.wikipedia.org/wiki/Uncontrolled_format_string
https://www.owasp.org/index.php/Format_string_attack
http://www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf

`printf()` is a function in C for formatting output. It can take in one, or two or more than 2 parameters.

```
int printf ( const char * format, ... );
```

Common usage is printf (format, s) where format specifies the format, and s is the variable to be displayed. E.g.

```
    printf (”the value in temp is %d\n", temp)
```

would display the following if temp contains the value 100 the

```
    the value in temp is 100
```

If there is "%d" in the string, during execution, printf() will still fetch value of the 2nd parameter, from the supposing location of the 2nd parameter, and display it. This is done even if the "printf" in the program does not have the 2nd parameter,

e.g. `printf ( “hello world %d” )`

If the value in the pickup location happened to be 15, then what being displayed will be

```
    hello world 15
```

If it happened to be 148, then we have

```
    hello world 148
```

You may wonder why can't printf() double-check that the program has only one parameter. To facilitate the checking would incur some runtime overhead, i.e. less efficient.

Simple preventive measure

Avoid using the following form where t is a variable:

- `printf(t)`
- `printf(t, a1, a2)`

Safe version

- `printf(”hello”);`
- `printf(”The value of %s is %d”, a1, a2)`

How such printf vulnerability can be exploited

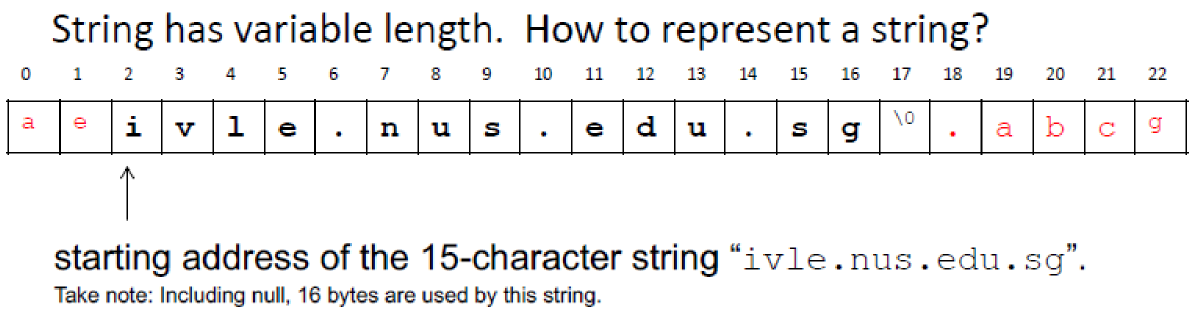
- If a program is vulnerable, the attacker might be able to
  - (1) obtain more information (***confidentiality***)
  - (2) cause the program to crash, e.g. using %s. (***execution integrity***)
  - (3) modifying the memory content using "%n". *(**memory integrity which might lead to execution integrity**)*
- If the program that invokes *printf* has elevated privilege (set UID enabled), a user (the attacker) might be able to obtain information that was previously inaccessible.

  E.g. Suppose the program for a web-server has the unsafe printf. Under normal usage, the server would obtain a request from the client, and then display (via the printf) it for confirmation. Now, a client (the attacker) might able to submit a web request to obtain sensitive information (e.g. the secret key), or to cause the web-server to crash.

8.3 Data Representation

Different parts of a program (or system) adopts different data representations. Such inconsistencies could lead to vulnerability.

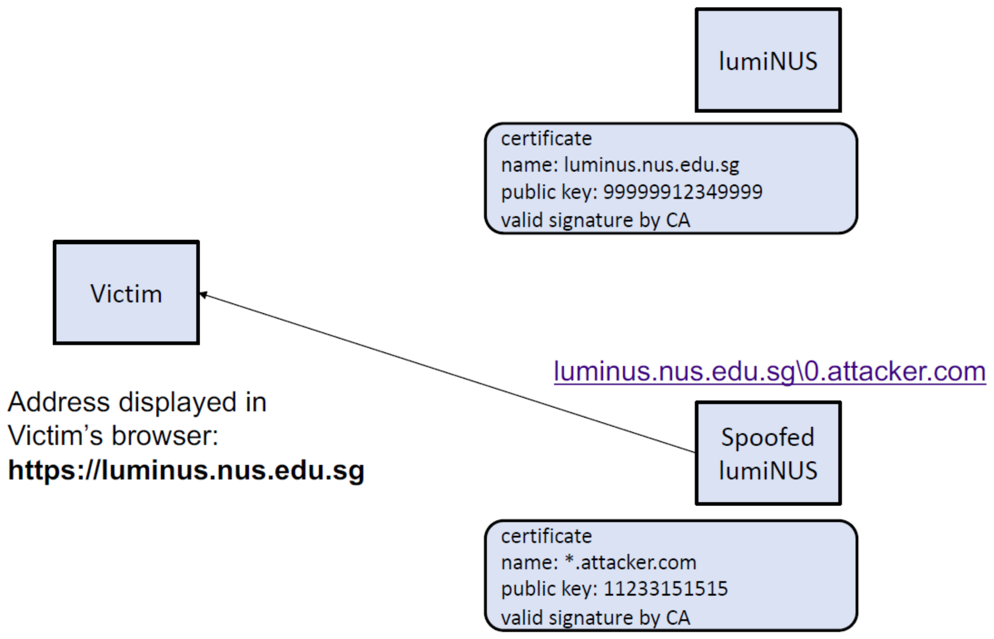
Example: bug in an SSL implementation that is exploitable.
https://www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/
A vulnerability in Ruby's SSL client that could allow man-in-the-middle attackers to spoof SSL servers via valid certificate issued by a trusted certification authority. Ruby's SSL client implements hostname identity check but it does not properly handle hostnames in the certificate that contain null bytes.



String has variable length. How to represent a string?
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
a e i v l e . n u s . e d u . s g \0 . a b c g
starting address of the 15-character string "ivle.nus.edu.sg".
Take note: Including null, 16 bytes are used by this string.

The *printf()* in C adopts an efficient representation. The length is not stored explicitly. The first occurrence of the "null" character (i.e. byte with value 0) indicates end of the string, and thus implicitly gives the string length. Not all systems adopt the above convention. Let's call the above NULL termination, and other non-NULL termination.

*the X509 standard adopt non-null termination to represent Common Name



A system which uses both null and non-null definitions to verify the certificate may get "confused".
Consider a browser implementation that:

(1) verifies a certificate based on non-null termination;

(2) displays the name based on null termination.

1. The attacker registered the following domain name, and purchased a valid certificate C1 with the following name from a CA.

   *.attacker.com

2. The attacker setup a spoofed website of Luminus with the domain name.

   luminus.nus.edu.sg\0.attacker.com

3. The attacker directed a victim to the spoofed webserver (For e.g., (1) via phishing email, the attacker tricked the victim to visit the attacker's webserver, (2) evil café owner).
4. The attacker's webserver presented the certificate C1.
   • The **browser displayed the address (based on null-termination)** on the browser's address bar.
   • The **browser verified the certificate (based on non null-termination)**. Since it was valid, the browser displayed the pad-lock in the address bar.

Example: IP address

An ip address has 4 bytes. There are many ways to represent ip address in a program. An ip address can be represented as:

- A **string**, e.g "132.127.8.16". This is human readable.
- **4 integers**, and each is a 32-bit integer.
- A **single 32-bit integer**. Note that 32 bits = 4 bytes.
- etc

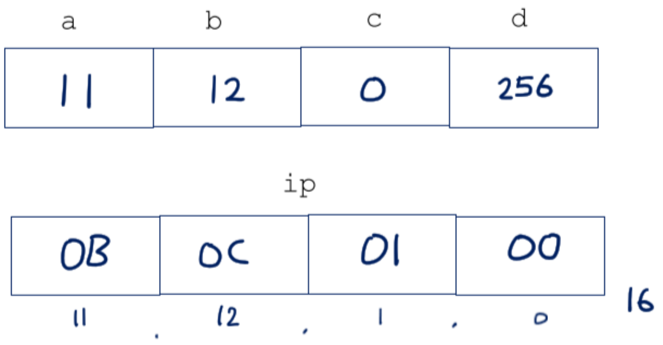
Consider a blacklist containing a lists of ip-addresses. A programmer wrote a sub-routine BL that, takes in 4 integers (each integer is of the type "int", i.e. represented using 32 bits), and check whether the ip-address represented by these 4 integers is in the blacklist. (in C, `int BL(int a, int b, int c, int d)`)

In the routine BL, the blacklist is stored in 4 arrays of integers A, B, C, D. Given the 4 input parameters a, b, c, d, the routine BL simply searches for the index i such that `A[i] ==a, B[i]==b, C[i]==c, and D[i]==d`. If it is in the list, the output of the routine is "TRUE".

Suppose a (vulnerable) program is written using the following flow:

(1) Get a string **s** from user.
(2) Extract four integers (each integer is of type int, i.e. 32-bit) from the string s in this way:
 I. Divide s into 4 substrings. The first substring is the string before the first occurrence of the character "." The 2nd substring is the subsequent substring before the next "." and so on.
 II. Convert each substring into a 32-bit integer. Let them be a, b, c, d. If failure in the conversion or number of substrings is not 4, halt.
(3) Invoke **`BL`** to check that whether (a, b, c, d) is in the blacklist. If return TRUE, quit.
(4) Else, let `ip = a*`2^{24}` + b*`2^{16}` + c*`2^{8}` + d` where `ip` is a 32-bit integer. (i.e. pack the 4 integers into a single 4-byte number)
(5) Continue the rest of processing with the address ip. E.g. call the connect library to connect to ip.

If the input string is "*11.12.0.256*",



The flaw is that the checking in step (2) was not sufficient – the value in each of the 4 components in IP address should be 1 byte / 8 bits / $[0\text{-}255]_{10}$. *11.12.0.256* is not a valid IP address.

Guideline: Use Canonical representation
Do not trust the input from user. Always convert them to a standard (i.e. canonical) representation immediately.

8.4 Buffer Overflow

Memory

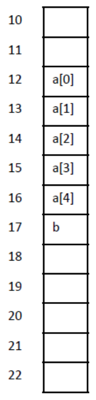
C and C++ do not employ "bound check" during runtime. This achieves efficiency but prone to bugs.

Buffer Overflow/Overruns

The previous example illustrates ***Buffer Overflow*** (aka buffer overrun). In general, buffer overflow refers to a situation where data are written beyond the (buffer's) boundary.

Consider this simple program

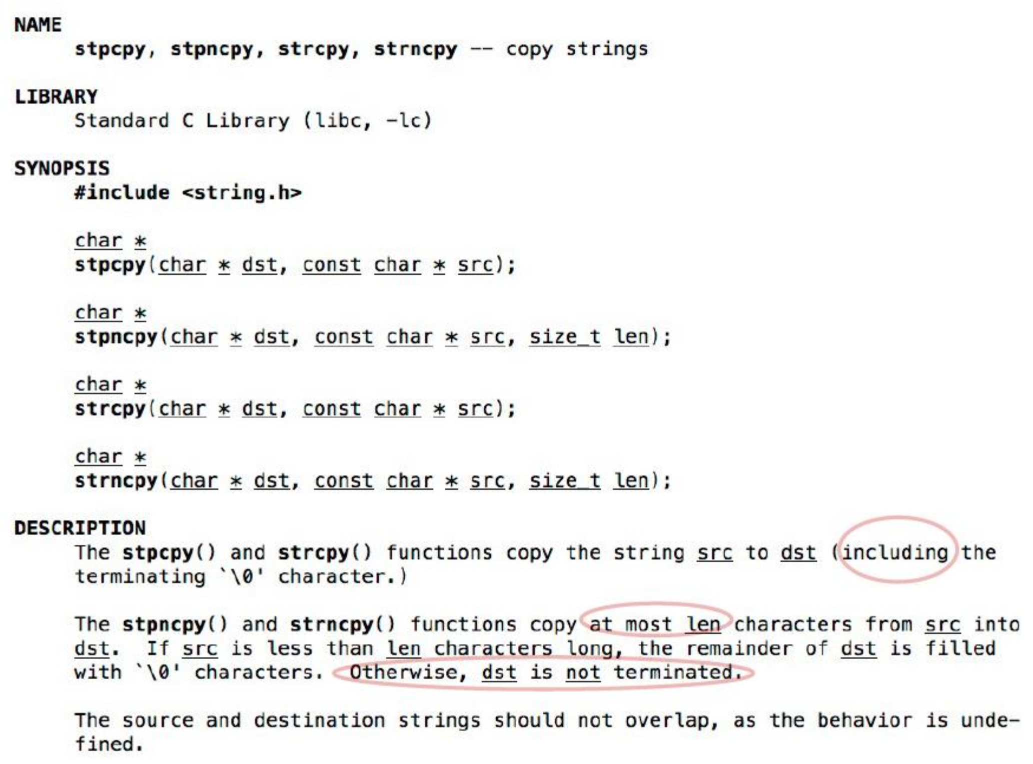
```
#include<stdio.h>
int a[5]; int b;
int main()
{
   b=0;
   printf("value of b is %d\n", b);
   a[5]=3;
   printf("value of b is %d\n", b);
}
```



Here, the value 3 is to be written to the cell `a`[5], which is also the location of the variable `b`.

In the previous example, the array is a buffer of size 5. The location a[5] is beyond its boundary. Hence, writing on it causes "buffer overflow". Well-known function in C that prone to buffer overflow is the string copying function: **strcpy**

strcpy



```
NAME
     stpcpy, stpncpy, strcpy, strncpy -- copy strings

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <string.h>

     char *
     stpcpy(char * dst, const char * src);

     char *
     stpncpy(char * dst, const char * src, size_t len);

     char *
     strcpy(char * dst, const char * src);

     char *
     strncpy(char * dst, const char * src, size_t len);

DESCRIPTION
     The stpcpy() and strcpy() functions copy the string src to dst (including the
     terminating `\0' character.)

     The stpncpy() and strncpy() functions copy at most len characters from src into
     dst.  If src is less than len characters long, the remainder of dst is filled
     with `\0' characters.  Otherwise, dst is not terminated.

     The source and destination strings should not overlap, as the behavior is unde-
     fined.
```

```
char s1 [10];
// .. get some input from user and store in a string s2.
strcpy ( s1, s2 )
```

In the above, potentially the length of s2 can be more than 10. The strcpy function copies the whole string of s2 to s1, even if the length of s2 is more than 10. Note that the "buffer" of s1 is only 10. Thus, the extra values will be overflown and written to other part of the memory. If s2 is supplied by the (malicious) user, a well-crafted input will overwrite beyond the boundary.

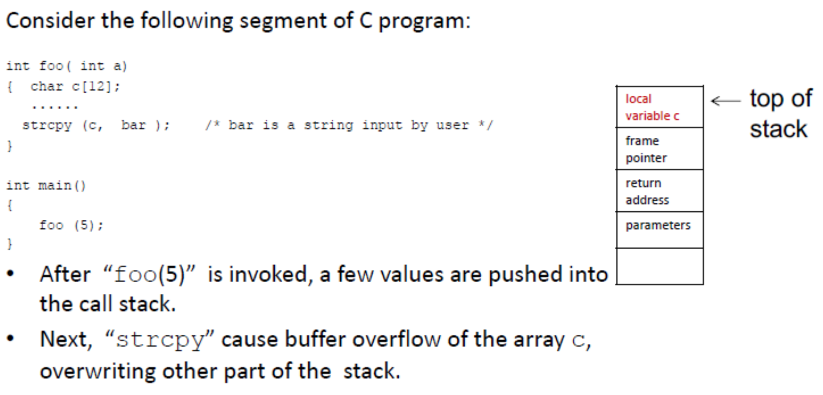
In secure programming practice, use **`strncpy`** instead. The function stcncpy takes in 3 parameters: `strncpy (s1, s2, n)` At **most** *n* characters are copied. Note that improper usage of strncpy could also lead to vulnerability.

Stack Smashing

Stack smash is a special case of buffer overflow that targets stack. Buffer overflow on stack is called stack overflow, stack overrun, or stack smashing.

Recap Call Stack in previous lecture. If the return address (which is stored in stack) is modified, the execution control flow will be changed. A well-designed overflow could "inject" attacker's shell-code into the memory, and then run the shellcode. *(this is an example illustrating how compromise of memory integrity could lead to compromise of execution integrity)* There are effective (but not foolproof) mechanisms (canary) to detect stack overflow.

When a function is called, the parameters, return address, local variables are pushed in a stack. If an attacker manages to modify the return address, the control flow will jump to the address indicated by the attacker.



Consider the following segment of C program:

```
int foo( int a)
{  char c[12];
   ......
  strcpy (c,  bar );     /* bar is a string input by user */
}

int main()
{
    foo (5);
}
```

- After "`foo`(5)" is invoked, a few values are pushed into the call stack.
- Next, "`strcpy`" cause buffer overflow of the array `c`, overwriting other part of the stack.

8.5 Integer Overflow

The integer arithmetic in many programming language are actually "modulo arithmetic". Suppose a is a single byte (i.e. 8-bit) unsigned integer, in the following C and java statements what would be the final value of a?

```
a = 254;
a = a + 2;
```

Its value is 0. The addition is with respect to module 256.
Hence, the following predicate is not necessary always true.

```
(a < (a+1) )
```

Many programmers don't realize that, leading to vulnerability

8.6 Code (Script) injection

Scripting language

• A key concept in computer architecture is the treatment of "code"(i.e. program) as "data". In the context of security, mixing "code" and "data" is potentially unsafe. Many attacks inject malicious codes as data.

• *Scripting languages* are particularly vulnerable to such attack. ***Scripts*** are programs that automates the execution of tasks that could alternatively be executed line-by-line by a human operator. Scripting language are typically "interpreted", instead of being compiled. Well-known examples are PHP, Perl, JavaSript, SQL.

• Many scripting languages allow "script" to be data storing in variables. This opens up the possibility of injecting malicious code into the script.

• Let's consider the well-known ***SQL injection attack***.

SQL Injection

SQL is a database query language.

Consider a database (can be viewed as a table). Each column is associated with an attribute, e.g. "name".

The query script

SELECT * FROM client WHERE name = 'bob'

searches and returns the rows where the name matches 'bob'. The scripting language supports variables. For e.g. a script first gets the user's input and stores it in the variable **$userinput**. Next, it runs the following:

SELECT * FROM client WHERE name = '$userinput'

In this example, let's suppose a web page use the following script.

-- script that get $userinput from user
SELECT * FROM client WHERE name = '$userinput'

The script (1) get **$userinput** from the user, (2) send the sql request to the SQL server, (3) displays the result to the user. The original intention is that, only one record would be displayed at one time, and the user must know the name to get the record. Now, consider a malicious user who doesn't know any name. This user set **$userinput** to be

anything' OR 1=1 --

The interpreter see the following line, which contain the variable **$userinput**

SELECT * FROM client WHERE name = '$userinput'

By design of scripting language, the interpreter simply substitutes in **$userinput** and get

SELECT * FROM client WHERE name = 'anything' OR 1=1 --'

The above is then sent to the SQL server. Outcome? All records will be displayed.

8.7 Undocumented access point (Easter eggs)

• For debugging, many programmers insert "undocumented access point" to inspect the states. For example, by pressing certain combination of keys, value of certain variables would be displayed, or for certain input string, the program would branch to some debugging mode.

• These access points may mistakenly remain in the final production system, providing a "back door" for the attackers.

• Also known as Easter Eggs. Some Easter eggs are benign and intentionally planted by the developer for fun or publicity.

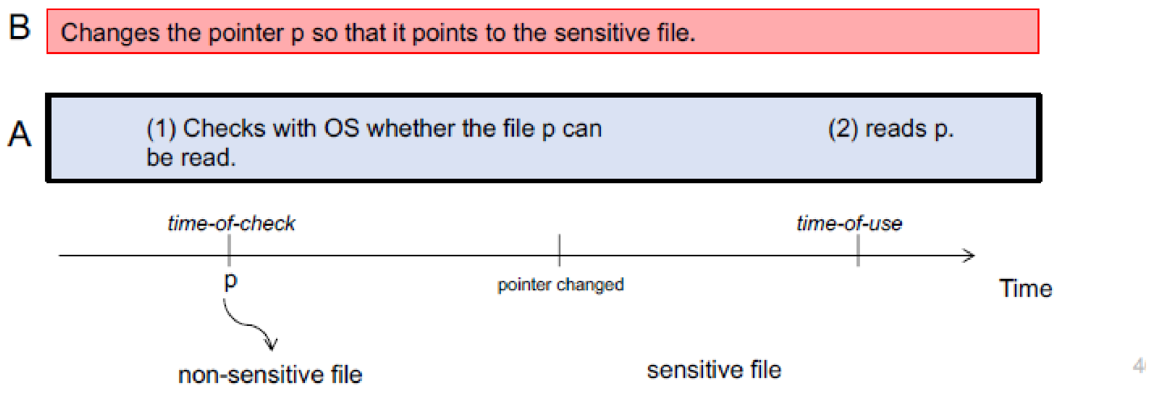
• There are known cases where unhappy programmer planted the backdoors.

Terminologies

- A ***logic bomb*** is a piece of code intentionally inserted into a software system that will set off a malicious function when specified conditions are met.
- In the virtual world, "***Easter eggs***" describes hidden code that appears within an application if a certain action triggers the abnormal application behavior.
- A **backdoor** is a malware type that negates normal authentication procedures to access a system. As a result, remote access is granted to resources within an application, such as databases and file servers.

8.8 Race Condition (TOCTOU)

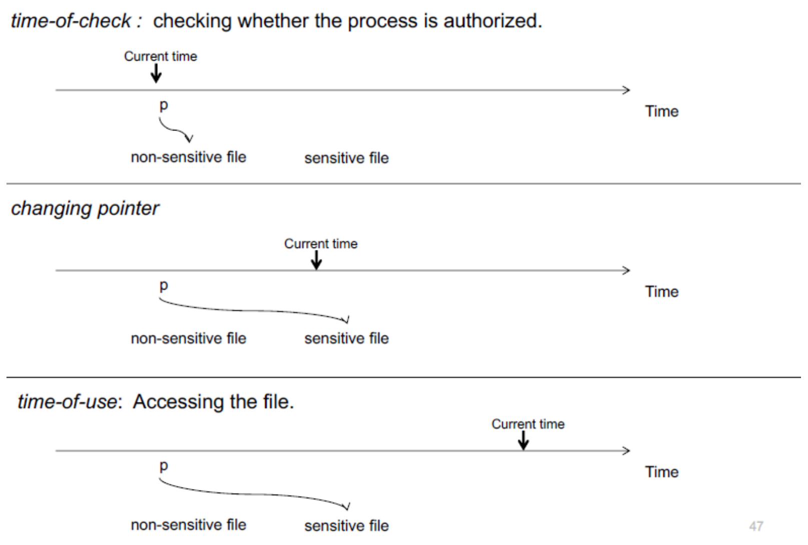
*In the following e.g. B doesn't have permission to access the sensitive file. But B has permission to modify a file pointer **p**.*


B
Changes the pointer p so that it points to the sensitive file.
A
(1) Checks with OS whether the file p can be read.
(2) reads p.
time-of-check
time-of-use
p
pointer changed
Time
non-sensitive file
sensitive file
4

Generally, race condition occurs when multiple processes (running in parallel) access a piece of shared data in such a way that the outcome depends on the sequence of accesses.

In the context of security, the "multiple processes" typically refer to
(1) a process **A** that checks the permission to access the data, follow by accessing the data, and
(2) another process **B** (the malicious one) that "swaps" the data.

There is a "race" among **A** and **B**. If **B** manages to be completed in between the time-of-check and time-of-use in **A,** the attack succeed.



Thus, this type of race condition is also known as ***time-of-check-time-of-use*** **(TOCTOU)**.

TOCTOU Example 1. Implementation of a “bridge” in SetUID e.g.

https://cwe.mitre.org/data/definitions/367.html for another example

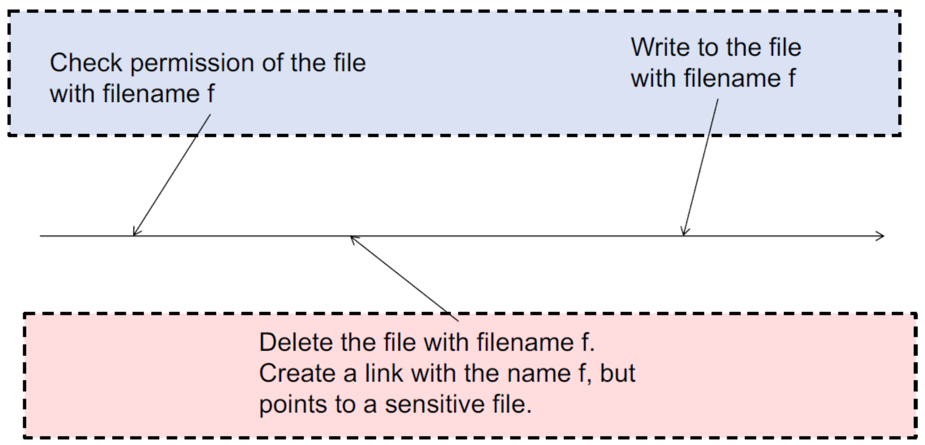
• Here is a program that is to be ran under elevated privilege (i.e setuid enabled, and the owner is root). *Recap that the “bridge” has elevated privilege and it must make sure that the user stay within the pre-defined boundary.*

• The system call `access(f, W_OK)` checks whether the user executing the program has the permission to write to the specified file f. It returns 0 if the process has the permission. The check is done based on the process ***real*** UID.

```
      //  f is a string that contains the name of a file
      //  fd is the file descriptor

TOC   if(!access(f, W_OK))          // check whether the real UID has write permission to f
      {
          fprintf (stderr, “permission to operate %s granted\n”, f );
TOU       fd = open(f,O_RDWR);        // open the file with read and write access
          OP(fd);                      // a routine that operates the file. OP is not a system call
          ....
      }
      else
      {
          fprintf(stderr,"Unable to open file %s.\n", f );
      }
```

• The program has the “setuid enabled” and its owner is “root”. Suppose bob invokes the program. Thus the real UID is bob, but effective UID is root. Furthermore suppose that the filename f is `/usr/year1/bob/list.txt` which is a file owned by bob.


Check permission of the file with filename f
Write to the file with filename f
Delete the file with filename f. Create a link with the name f, but points to a sensitive file.

`/usr/year1/bob/list.txt`

`/usr/course/cz2017/grade.txt`

• After bob starts the process, he immediately replaces the file `/usr/year1/bob/list.txt` by a symbolic link that points to a system file. bob does not has write permission of that system file, but has permission to change `list.txt`

• This can be done by running a script that carries out the following 2 steps:

1. Delete `/usr/year1/bob/list.txt`
2. Create a symbolic link with the filename f, and points to a system file. That is, issuing this Unix command

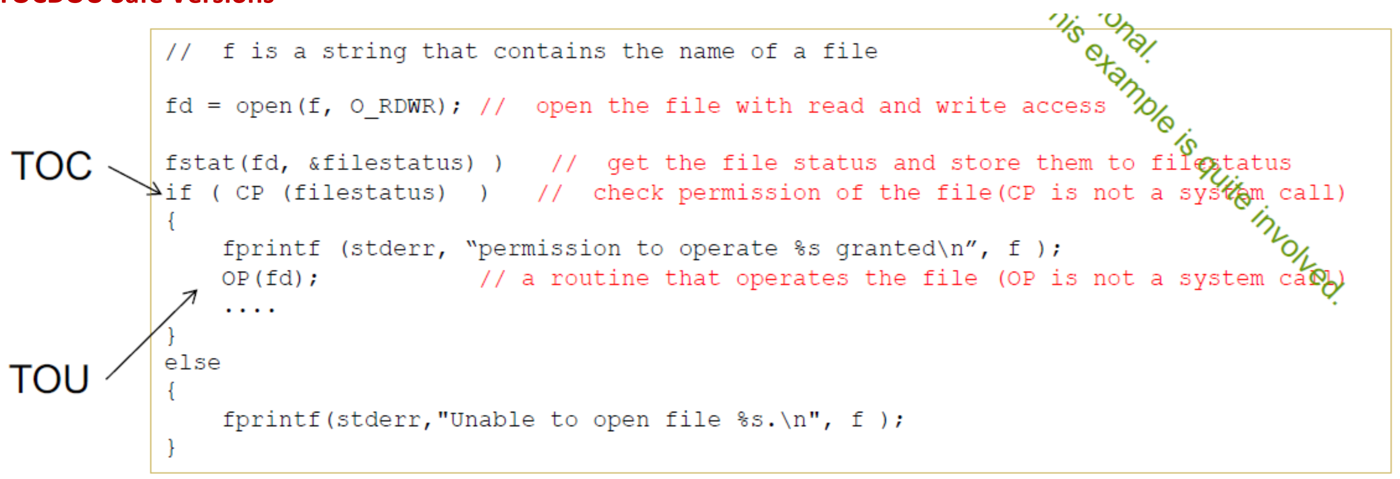
```
ln -s /usr/course/cz2017/grade.txt /usr/year1/bob/list.txt
```

**ln command is used to create hard link/symbolic link to a file*

With some chances, bob wins the race. Hence the process operates on the system file

```
      /usr/course/cz2017/grade.txt instead of /usr/year1/bob/list.txt
```

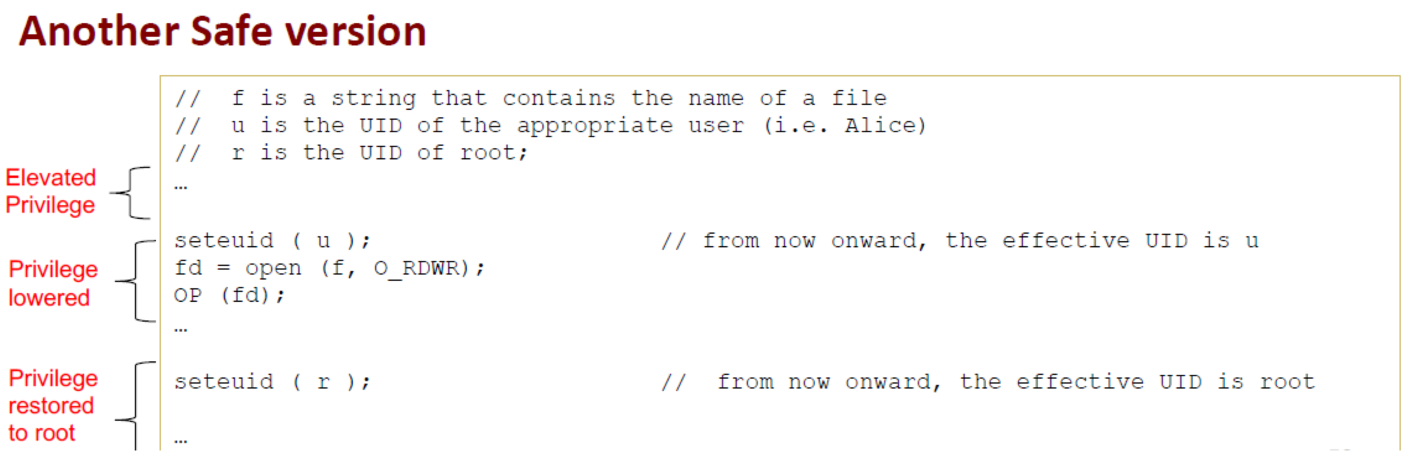
TOCDOU Safe Versions



```
//  f is a string that contains the name of a file

fd = open(f, O_RDWR); //  open the file with read and write access

fstat(fd, &filestatus) )   //  get the file status and store them to filestatus
if ( CP (filestatus)  )   //  check permission of the file(CP is not a system call)
{
    fprintf (stderr, "permission to operate %s granted\n", f );
    OP(fd);              // a routine that operates the file (OP is not a system call)
    ....
}
else
{
    fprintf(stderr,"Unable to open file %s.\n", f );
}
```



Another Safe version

```
//  f is a string that contains the name of a file
//  u is the UID of the appropriate user (i.e. Alice)
//  r is the UID of root;
…

seteuid ( u );                    // from now onward, the effective UID is u
fd = open (f, O_RDWR);
OP (fd);
…

seteuid ( r );                    //  from now onward, the effective UID is root

…
```

8.9 Defense and preventive measures

As illustrated in previous examples, many are bugs due to programmer ignorance. In general, it is difficult to analyze a program to ensure that it is bug-free (the "halting-problem"). There isn't a "fool-proof" method.

8.9.1 Input Validation, Filtering, Parameterized Queries (SQL)

In almost all examples (except TOCTOU) we have seen, the attack is carried out by feeding carefully crafted input to the system. Those malicious input does not follow the "expected" format. For example, the input is too long, contains control characters, contains negative number, etc.

• Hence, a preventive measure is to perform ***input validation*** whenever an input is obtained from the user. If the input is not of the expected format, reject it. There are generally two ways of filtering:

1. **White list**: A list of items that are known to be benign and allowed to pass. The white list could be expressed using regular expression. However, some legitimate input may be blocked. Also, there is still no assurance that all malicious input would be blocked.
2. **Black list**: A list of items that are known to be bad and to be rejected. For example, to prevent SQL injection, if the input contains meta characters, reject it. However, some malicious input may be passed.

It is **difficult to design a filter** that is

• ***complete*** (i.e. doesn't miss out any malicious string); and

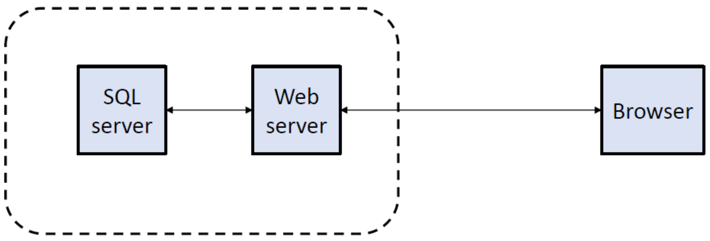
• ***accepts all legitimate*** **inputs**.

Parameterized Queries (SQL)
Parameterized queries are mechanisms introduced in some SQL servers to protect against SQL injection. Here, queries sent to the SQL are explicitly divided to two types: the queries, and the parameters.

The SQL parser will know that the parameters are "data" and are not "script". The SQL parser is designed in such a way that it would never execute any script in the parameters. This check will prevent most injection attack.

Will this stop all scripting attacks? No. E.g. it can't stop XSS.

E.g. Where should input validation apply?
Question: At Browser, Web Server or SQL Server?
Answer: Definitely not browser. Normally In web server instead of SQL server.
(note: parameterized queries in some sense tags substrings given by user as "input". Hence the SQL server can differentiate "data" from "script" and ensure that data are not parsed as script)



8.9.2 Use "safe" function

Completely avoid functions that are known to create problems. Use the "safe" version of the functions.

C and C++ have many of those:

strcpy	strncpy
printf(f)	
access()	

Again, even if they are avoided, there could still be vulnerability.
(e.g. the example that uses a combination of strlen() and strncpy())

8.9.3 Bound checks and type safety

Bound Checks
Some programming languages (e.g. Java, Pascal) perform bound checking during runtime (i.e. while the program is executing). During runtime, when an array is instantiated, its upper and lower bounds will be stored in some memory, and whenever a reference to an array location is made, the index (subscript) is checked against the upper and lower bound. Hence, for a simple assignment like

```
a[i] = 5;
```

what actually being carried out are:

(1) if `i < lower bound`, then halts;
(2) if `i > upper bound`, then halts;
(3) assigns 5 to the location.

If the check fails, the process will be halted (or exception to be thrown, as in Java). The first 2 steps reduce efficiency, but will prevent buffer overflow. The infamous C, C++ do not perform bound check. Many of the known vulnerabilities are due to buffer overflow that can be prevented by the simple bound check.

C. A. R. Hoare (1980 Turing Award winner) on his experience in the design of ALGOL 60, a language that included bounds checking.
"A consequence of this principle is that every occurrence of every subscript of every subscripted variable was on every occasion checked at run time against both the upper and the lower declared bounds of the array. Many years later we asked our customers whether they wished us to provide an option to switch off these checks in the interest of efficiency on production runs. Unanimously, they urged us not to—they already knew how frequently subscript errors occur on production runs where failure to detect them could be disastrous. I note with fear and horror that even in 1980, language designers and users have not learned this lesson. In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law."

Type Safety
Some programming languages carry out "type" checking to ensure that the arguments an operation get during execution are always correct.

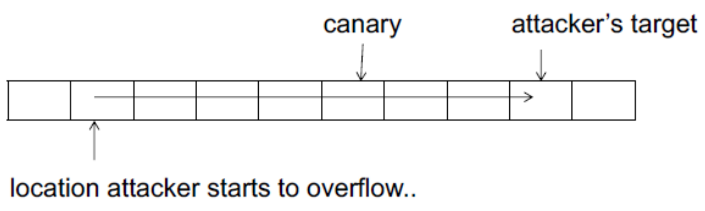
e.g. a = b;
if a is a 8-bit integer, b is a 64-bit integer, then the type is wrong.

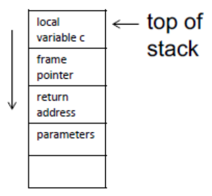
The checking could be done during runtime (i.e. *dynamic* type check), or when the program is being compiled (i.e. *static* type check).

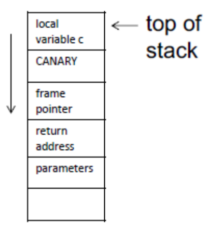
8.9.4 Canaries and Memory protection

Canary
Canaries are secrets inserted at carefully selected memory locations during runtime. Checks are carried out during runtime to make sure that the values are not being modified. If so, halts.

Canaries can help to detect overflow, especially stack overflow. This is because in a typical buffer overflow, consecutive memory locations have to be over-ran. If the attacker wants to write to a particular memory location via buffer overflow, the canaries would be modified. *(It is important to keep the value as "secret". If the attacker happens to know the value, it may able to write the secret value to the canary while over-running it).*







This command turns off canary. (by default it is on)

```
> gcc myprogram.c -fno-stack-protector
```

Question: What is the disadvantage of having the canary protection?

The canary has to be validated on each function return. The performance overhead is only a few percent since a canary is only needed in functions with local arrays.

Question: Consider a C program t.c compiled using gcc and to be run in a particular OS, say ubuntu. Who should implement the canary protection? *programmer of* t.c, ***person who write gcc compiler***, *or the person who wrote OS?*

Memory randomization
It is to the attacker's advantage when the data and codes are always stored in the same locations. *Address space layout randomization (ASLR)* (details omitted) can help to decrease the attackers chance of success.

8.9.5 Code Inspection

Manual Checking: Manually checking the program. Certainly tedious.
Automated Checking: Some automations are possible. For example, *taint analysis:*
Variables that contain input from the (potential malicious) users are labeled as *source.* Critical functions are labeled as *sink.* Taint analysis checks whether the sink's arguments could potentially be affected (i.e. tainted) by the source. If so, special check (for e.g. manual inspection) would be carried out. The taint analysis can be static (i.e. checking the code without "tracing it"), or dynamic (i.e. run the code with some input).

E.g.
Sources: user input
Sink: opening of system files, function that evaluates a SQL command, etc

8.9.6 Testing

White-box testing: The tester has access to the source code.
Black-box testing: The tester does not has access to the source code.
Grey-box testing: A combination of the above.

Vulnerability can be discovered via testing. Security testing attempts to discover intentional attack, and hence would test for inputs that are rarely occurred under normal circumstances. (for e.g. very long names, or names that contain numeric values.) ***Fuzzing*** is a technique that sends malformed inputs to discover vulnerability. There are techniques that are more effective than sending in random input. (detail not required). Fuzzing can be automated or semi-automated.

Terminologies
A ***white hat*** (or a *white hat* hacker) is an ethical security hacker.
A ***black hat*** (or *black-hat* hacker) is a hacker who violates computer *security* out of malice.
Whitelisting denies access to all resources and only the "owner" can allow access.
Blacklisting allows access to all with the provision that only certain items are denied.

8.9.7 Principle of Least Privilege

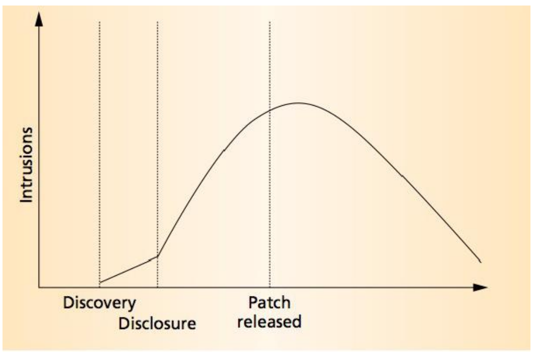
When writing a program, be conservative in elevating the privilege. When deploying software system, do not give the users more access rights than necessary, and do not activate options unnecessary.

E.g.: Software contain many features. (For e.g. a web-cam software could provide many features so that the user can remotely control it.) A user can choose to set which features to be on/off. Suppose you are the developer of the software. Should all features to be switched on by default when the software is shipped to your clients? If so, it is the clients' responsibility to "harden" the system by selectively switch off the unnecessary features. Your clients might not aware of the implications and thus at a higher risk.

Terminology: **hardening**
A process intended to eliminate a means of attack by patching vulnerabilities and turning off nonessential services.

8.9.8 Patching

Life cycle of vulnerability:
Vulnerability is discovered -> Affected code is fixed -> Revised version is tested -> Patch is made public -> Patch is applied.



In some cases, the vulnerability could be announced without the technical details before a patch is released. The vulnerability likely to be known to only a small number of attackers (even none) before it is announced. When a patch is released, the patch can be useful to the attackers. The attackers can inspect the patch and derive the vulnerability. Hence, interestingly, the number of successful attacks goes up after the vulnerability/patch is announced, since more attackers would be aware of the exploit.

A **zero-day vulnerability** is a vulnerability in a system or device that has been disclosed but is not yet patched or unknown to those who should be interested in its mitigation. An exploit that attacks a zero-day vulnerability is called a zero-day exploit.

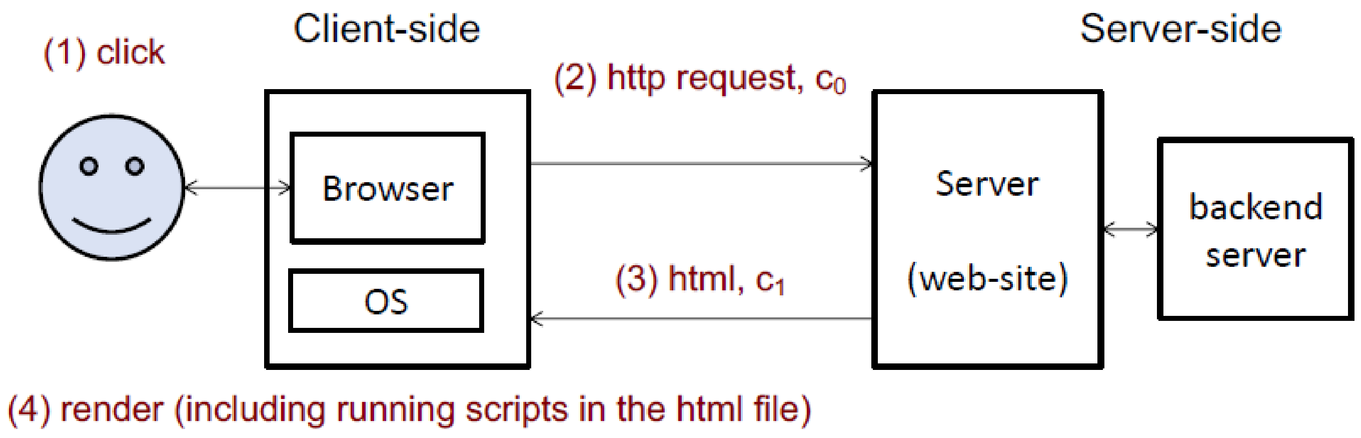
It is crucial to apply the patch timely. Although seems easy, applying patches is not that straightforward. For critical system, it is not wise to apply the patch immediately before rigorous testing. Patches might affect the applications, and thus affect an organization operation. (imagine a scenario where after a patch is applied, a software that manages critical infrastructure crash. Or a scenario where after a student applied a patch on browser, the student can't upload report to IVLE workbin and thus missed the project deadline). "Patch Management" is a field of study.

Lecture 9 – Web Security

Summary & takeaway

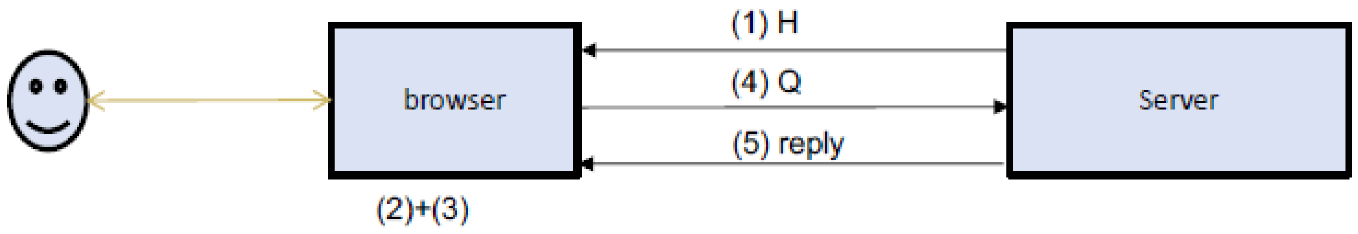
- HTTPS = HTTP on top of TLS.
- Browser interacts with multiple sites. Cookies associate to sites. Weak separation among different sites (same source origin).
- Cookie as authentication token.
- Various attacks.

9.1 Background



1. User clicks on a "link". e.g mysoc.nus.edu.sg/
2. A http request is sent to the server. (with cookie c0 if any)
3. Server constructs and sends a "html" file to the browser, possibly with a cookie c1 (the cookie can be viewed as some information the server wants the browser to keep, which to be passed to the server during the next visit. The cookie can be some secrets).
4. The browser renders the html file. The html file describes the layout to be rendered and presented to the user. The html file also instruct the browser to construct and store cookies. (which could be differ from c1)

Closer look into an example



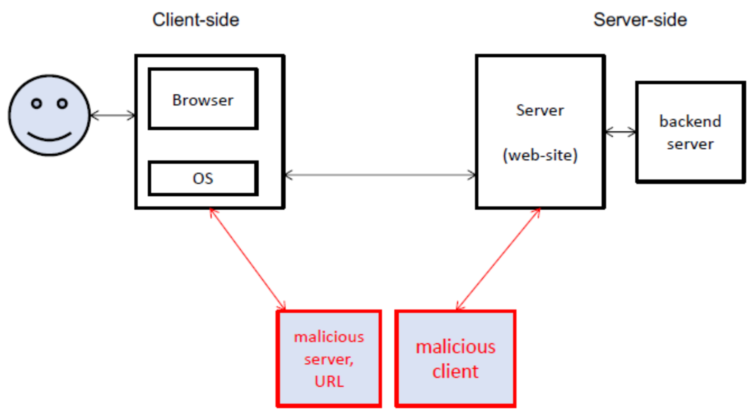
1) Browser visits Google page. A html file H is sent to the browser. The browser renders H.
2) User enters the keywords "CS2017 NUS"
3) The browser, by running H, constructs a query Q.
   https://www.google.com.sg/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=CS2107+NUS
4) Browser sends query Q to Google.
   Note that additional information is added as parameters in Q. These info is useful for the server. The info could even be in the form of script.
5) The server constructs a reply. (in some cases, the reply contain substrings in (3)).

Complications of Web Security

- In many OS'es, browsers run with the same privileges as the user. Hence, the browser can access the user's files. (note: this is not the case in Android).
- Browsers support a rich commands/controls set for content providers to render the content.
- Browsers manage sensitive user's info and secrets (stored in cookies).
- At any one time, there could be many servers providing the content in the browser (*e.g. different advertisements appear at the same time)*
- For enhanced functionality, many browsers support add-ons, plugins, extensions by third parties. *(definitions and differences of add-ons, plugins, extensions are not clear and depends on the browsers.)*
- Users upload info to the server (e.g. forum, names to be displayed).
- More and more sensitive data information is in the web/cloud.

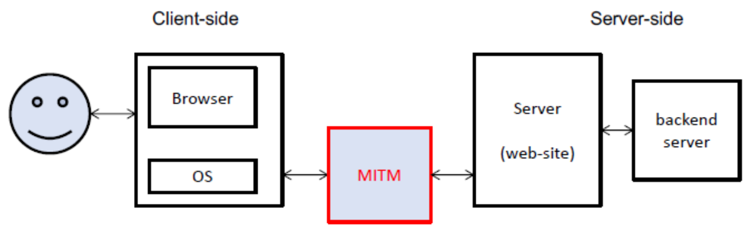
For PC, browser is becoming the main application. (Situation slightly different in mobile os where users also interact with the web using "apps".)

Threat model 1: Attackers as another end systems



Here, the attacker is just another end system. For e.g. a malicious web-server that the victim is lurked to visit, or attackers who has access to the targeted server.

Threat model 2: Attackers as MITM



Here, the attacker is a Man-in-the-middle in the IP or lower layer. The attacker can gain MITM in a few ways, for e.g.,

- As the café-owner who provides the free wifi;
- Via DNS spoofing attack or ARP attack;
- As owner of the VPN server, last hop in the TOR network (not covered in this module).

Because the MITM is in the IP layer, the MITM can attempt to impersonate a server (e.g. phishing attack). So, this threat model also covers some settings of threat model 1 where the attacker is the server.

9.2 Vulnerability in the secure communication channel (SSL/TLS)

HTTPS = HTTP on top of SSL/TLS

Recap: HTTPS considers threat model 2, i.e. a MITM in the IP layer.
HTTPS is "secure" in this model. This is achieved by TLS (securing the channel using combination of crypto primitives and authentication protocol) and PKI (securing distribution of public key).

Recap: From a user's perspective, the presence of padlock and correct domain name in the address bar, assure that the browser is indeed interacting with the authentic server. (hence not under phishing attack).

However, the above assumes that the system is designed and implemented correctly. There are failure.

A number of examples: Superfish (not covered in this year), **Re-negotiation attack** (tutorial), Freak attack (not covered in this year), BEAST attacks (similar to padding oracle) (not covered in this module).

9.3 Misleading user on domain name

URL (Uniform Resource Locator)

A url consists of a few components. (see https://en.wikipedia.org/wiki/Uniform_Resource_Locator)

1. **scheme**,
2. **authority** (a.k.a the **hostname**) ,
3. **path**
4. **query**
5. **fragment**

`http://www.wiley.com/WileyCDA/Section/id302477.html?query=computer%20security#12`

scheme:	http
authority:	wiley
path:	WileyCDA/Section/id-302477.html
query:	query=computer%20security
fragment:	12

Source of Confusion

Visually, no clear distinction of "Hostname" and "path". The delimiter "/" can appear elsewhere, which confuses viewers. A malicious website may register a hostname that contains the targeted hostname with a character that resembles the delimiter "/".

www.wiley.com.66785689.in/Section/id-302477.html

The different intensities could help user to spot the attack.

Example:

The browser Safari chooses to display the hostname *only* in the address bar.

Address bar spoofing

Address bar is an important component to protect. The address bar is the only indicator of which url the page is rendering. If the address bar can be "modified" by the webpage, an attacker could trick the user to visit a malicious url **X**, while making the user to falsely believe that the url is **Y**.

A poorly-designed browser may allow attacker to achieve that. E.g. in early design of some browsers, a web page could render objects/pop-up in arbitrary location, and thus allowing a malicious page to overlay spoofed address bar on top of the actual bar. Current version of popular browser have mechanisms to prevent that.
See a more recent e.g.: **Android Browser All Versions - Address Bar Spoofing Vulnerability - CVE-2015-3830**

9.4 Cookie and Same-origin policy

Cookies

- A http cookie is a piece of data sent by the server in the response of a HTTP request. It is under the "Set-Cookie" header field. The browser permanently stores the cookie.
- Whenever a client revisits the website, the browser automatically sends the cookie back to the server.

Remarks:

- The cookie are sent back only to the "same origin", in the sense that, it is sent to the server which is the "origin" of the cookie. Viewing from access control perspective, the "same origin" set the boundary among different web sites. A script from a web site can only access cookies within its boundary. *(unfortunately, this is not done clearly, leading to many problems…)*
- There are also a few types of cookie, eg. session cookies (deleted after the session ends), secure cookies (can only be transmitted over https), etc.

Same-Origin Policy

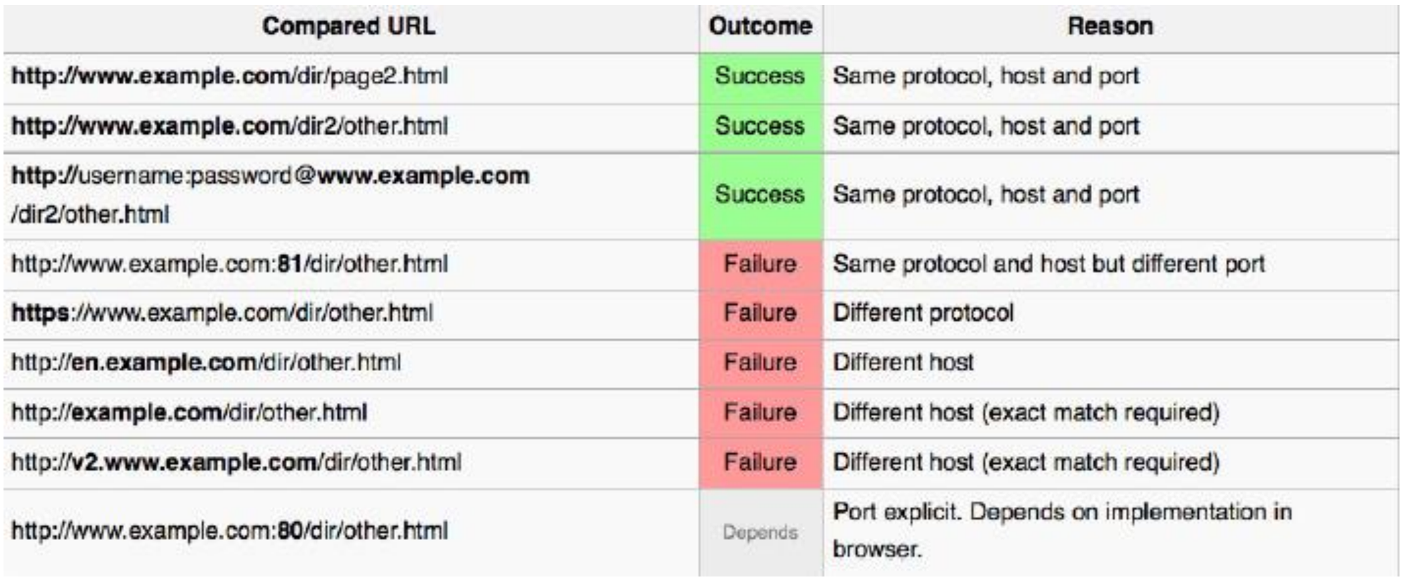
A "script" which runs in the browser could access cookies. Which scripts can access the cookie? Due to security concern, browser employs this access control policy:

The script in a web page A (identified by URL) can access cookies stored by another web page B (identified by URL), only if both A and B have the same ***origin.***

Origin is defined as the combination of **protocol, hostname, and port number**.
The above access policy is simple and thus seeming safe. However, there are a number of complications.

Definition of "same-origin".

Example (from http://en.wikipedia.org/wiki/Same-origin_policy)
URL's with the same origin as: http://www.example.com



Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com** /dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

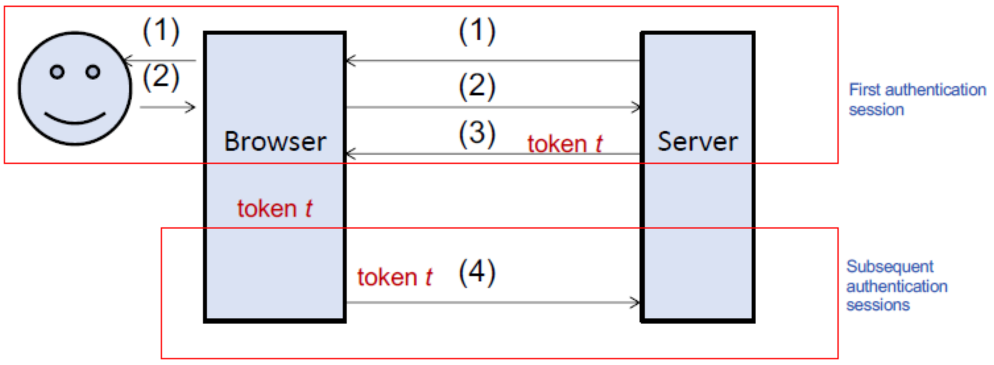
Limitations:

- *Confusing definition*: There are many exceptions, and exceptions of exceptions. Very confusing and thus prone to errors.
- Different browsers may adopt different definitions.

E.g. of Cookie application: Token-based authentication
Cookie is useful. Here is one application for Single-Signed-On. *(This example appeared in the tutorial on renegotiation attack)*

To ease user tedious task of repeated login, many websites use "token-based" authentication. That is,
1. After a user ***A*** is being authenticated (for e.g. password verified), the server sends a value ***t***, known as **token**, to the user.
2. In subsequent connection, whoever presents the correct value is accepted as the authentic user ***A***. (note that user does not has to perform the tedious password authentication again).

Remarks
• Token typically has an expiry date.
• An identification of the session can be used as the token. So the token is also called SID.
• In web applications, the token is often stored in the cookie.


(1)
(2)
Browser
token *t*
(1)
(2)
(3)
token *t*
Server
First authentication session
token *t*
(4)
Subsequent authentication sessions

(1) Authentication challenge (e.g. asking for password).
(2) Authentication Response that involves the user.
(3) Server sends a token *t.* Browser keeps the token
(4) Browser presents the token *t.* Server verifies the token.

We assume that the communication channel is secure. (i.e. all communication over HTTPS (with server authenticated) and the HTTPS is free from vulnerabilities).

<u>Choices of token</u>
Suppose *t* is a randomly chosen number, then the server needs to keep a table storing all the tokens it has issued. To avoid storing the table, one could use

• (secure version) A message authentication code (**MAC**). The token ***t*** consists of two parts: a randomly chosen value, or meaningful information like the expiry date, concatenated with the mac computed using the server secret key.
e.g ***t*** = "alicetan:16/04/2015:adc8213kjd891067ad9993a"

• (insecure version) the cookie is some meaningful information concatenated with a sequence number that can be predicted.
e.g **t**= "alicetan:16/04/2015:128829"

For both methods, when the server finds out that the token is not in the correct format (or not the correct mac), the server rejects. The first method relies on the **security of mac**, while the second method **relies on obscurity** – attackers don't know the format.

The second method is insecure because an attacker, who knows how the token is generated (for e.g. by observing its own token), can forge it. This further illustrates the weakness of security by obscurity.

9.5 Cross Site Scripting (XSS) Attacks

Background

In some websites, if the browser sends a text that contains a substring ***s,*** the replying html sent by the server would also contain the same substring ***s.***

E.g.

• Enter a wrong address.

http://www.comp.nus.edu.sg/nonsense_test

• Search for a book in library

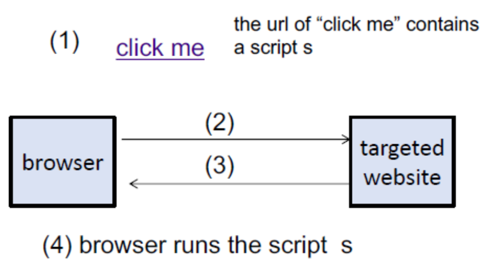
http://nus.preview.summon.serialssolutions.com/#!/search?q=heeheeheee

What if the string ***s*** contains a script?

e.g. http://www.comp.nus.edu.sg/**<script>alert("heehee!");</script>**

(the above attack won't work as the server replaces the special character "<" by &lt)

Attack


(1)
click me
the url of "click me" contains a script s
browser
(2)
(3)
targeted website
(4) browser runs the script s

1. Attacker tricks a user to click on an url, which contains the target website, and a malicious script ***s.*** e.g the link could be sent via email with "click me", or a link in a malicious website.
2. The request is sent to the server.
3. The server constructs a response html. The response contains the script ***s***.
4. The browser renders the html page and runs the script ***s***

Why this is an attack?

The malicious script could

1. Deface the original webpage
2. Steal cookie

The control that protects access to the cookie is the sameorigin policy.

• Now, since the malicious script is coming from the same-origin (the victim clicked on it..), it can now access cookie previously sent by the website.

• This is yet another example of *privilege escalation*. A malicious script has elevated privilege to read the cookie.

This attack exploits the client's trusts of the server.

types of XSS

• Reflection (aka non-persistent). The attack described before.

• Stored XSS (aka persistent). The script ***s*** is stored in the targeted website. For instance, in forum page.

Defense

Most defense rely on mechanisms carried out in the serverside. That is, the server filters and removes malicious script in the request while constructing the response page. However, this is not a fool proof method.

9.6 Cross Site Request Forgery (XSRF)

- Aka “sea surf”, cross-site reference forgery, session riding.
- This is the reverse of XSS. XSRF exploits the server’s trust of the client. Previous attack of XSS exploits the client’s trust of the server.

Example:

- Suppose a client A is already authenticated by a targeted website S, say www.bank.com and the site keeps a cookie as “token”.
- The attacker B tricks A to click on a url of S. The url maliciously requests for a service, say transferring $1000 to the attacker Bob’s. `www.bank.com/transfer?account=Alice&amount=1000&to=Bob`
- The cookie will also be automatically sent to S which is sufficient to convince S that A is already being authenticated. Hence the transaction will be carried out.

Defense

Relatively easier to prevent compared to XSS. Include authentication information in the request. For e.g.
`www.bank.com/transfer?account=Alice&amount=1000&to=Bob&`**`Token=xxk34n890ad7casdf897e324`**

9.7 Other Attacks

- A **drive-by download** refers to the unintentional download of malicious code onto a computer or mobile device that exposes users to different types of threats. What sets this type of attack apart from others is that ***users need not click on anything to initiate the download***. Simply accessing or browsing a website can activate the download.

- A **web beacon/web bug** is a technique used on web pages and email to unobtrusively allow checking that a user has accessed some content. Web beacons are typically used by third parties to monitor the activity of users at a website for the purpose of web analytics or page tagging.

- **Clickjacking** is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online.

- A **CAPTCHA** is a type of challenge–response test used in computing to determine whether the user is human.

- **Click fraud** is the act of illegally clicking on pay-per-click (PPC) ads to increase site revenue or to exhaust a company's advertising budget.

Question: Could merely visiting a malicious web-site (for e.g. wrongly clicked on a phishing email) subjected to attack?
Answer: Yes, by drive-by-download.

Common simple implementation mistakes

- Authentication/filtering at the client side.
- Security credential embedded in the public web pages.
- Server’s secrets stored in cookies.
- Configuration errors.
- URL as secrets, e.g. in password reset link, or zoom link. This is acceptable and commonly used. However, some implementations do not have adequate protection, or unknowingly leak info of the url.

Renegotiation Attack

TLS [RFC5246] allows either the client or the server to initiate renegotiation -- a new handshake that establishes new cryptographic parameters. Unfortunately, although the new handshake is carried out using the cryptographic parameters established by the original handshake, there is no cryptographic binding between the two. This creates the opportunity for an attack in which the attacker who can intercept a client's transport layer connection can inject traffic of his own as a prefix to the client's interaction with the server. One form of this attack [Ray09] proceeds as shown below:

To start the attack, the attacker forms a TLS connection to the server (perhaps in response to an initial intercepted connection from the client). He then sends any traffic of his choice to the server. This may involve multiple requests and responses at the application layer, or may simply be a partial application layer request intended to prefix the client's data. This traffic is shown with == to indicate it is encrypted. He then allows the client's TLS handshake to proceed with the server. The handshake is in the clear to the attacker but encrypted over the attacker's TLS connection to the server. Once the handshake has completed, the client communicates with the server over the newly established security parameters with the server. The **attacker cannot read this traffic**, **but the server believes that the initial traffic to and from the attacker is the same as that to and from the client**.

If certificate-based client authentication is used, the server will see a stream of bytes where the initial bytes are protected but unauthenticated by TLS and subsequent bytes are authenticated by TLS and bound to the client's certificate. In some protocols (notably HTTPS), no distinction is made between pre- and post-authentication stages and the bytes are handled uniformly, resulting in the server believing that the initial traffic corresponds to the authenticated client identity. Even without certificate-based authentication, a variety of attacks may be possible in which the attacker convinces the server to accept data from it as data from the client. For instance, if HTTPS [RFC2818] is in use with HTTP cookies [RFC2965], the attacker may be able to generate a request of his choice validated by the client's cookie.

```
Client                        Attacker                        Server
------                        --------                        ------
                                  <----------- Handshake ---------->
                                  <======= Initial Traffic ========>
<--------------------------  Handshake ============================>
<======================== Client Traffic ==========================>
```

The genuine client sends a Client Hello (GenCH) to the server, but this is intercepted by the MITM.

The MITM keeps hold of it and starts its own handshake by sending a a Client Hello (MitmCH) to the server too. It completes the handshake normally and sends some application data. For example:

Inserted by attacker:
```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
X-Ignore-This:
```

The MITM now forwards the initial Client Hello (GenCH) to the server.

As far as the client is concerned, this is not a re-negotiation, but the initial negotiation. It can't tell the difference. The handshake completes and it sends its own request:

Original request by victim:
```
GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
Cookie: victimscookie
```

As far as the server is concerned, receiving a second `Client Hello` (GenCH) is just a re-negotiation (which either party was able to initiate at any point in time) following the initial handshake initiated with `Client Hello` (MitmCH). The application layer is treated as continuous. This is what the server sees:

The combined request received by pizza shop:
```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
X-Ignore-This: GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
Cookie: victimscookie
```

This way, the attacker is able to alter the request line (and thus query parameters) and insert its own headers, while keeping the cookies of the genuine client. The **MITM doesn't get to see or alter what the client sends, but it's able to add content before**, which is treated as part of the same request on the application layer.

Is the TLS in this attack a unilateral or mutual authentication? If unilateral, which entity, Server or the Client, carries out the verification?

This is a unilateral authentication. The Client carries out the verification.

This attack model assumes that the attacker, before carrying out the re-negotiation, is already a MITM. In which layer does the MITM sit?

IP Layer

How the Client and the Attacker obtain the server's public key?

From the Server’s certificate, which is sent by the Server during the TLS handshake step.

From the Server's point of view, the Server carried out 2 handshakes. Let (k1; t1) and (k2; t2) be the sessions keys established during the first and second handshake respectively. Does the Attacker knows k2?

No, because the second handshake is performed between the Client and the Server. By security of the handshake (authenticated key-exchange), the attacker as a MITM is unable to get the session key.

Refer to introduction in RFC5746. Which key is used to protect the Initial Traffic"?

Key k1, which is derived in the first handshake

The second handshake is carried out between (1) Client and Attacker, and (2) Attacker and Server. Is (1) protected by the session key? Is (2) protected by the session key?

The segment (1) is not protected. The segment (2) is protected by k1.

Which key is used to protect the Client Traffic"?

Key k2, which is derived in the second handshake done before the Client Traffic is communicated.

The vulnerability was discovered in 2009, and RFC 5746 was released in Feb 2010 to update TLS/SSL protocol specification. Suppose that, sometime during late 2009, Alice uploaded her report via Luminus. Later, Alice was very worried that someone would peep into her report. Could the confidentiality of the report be compromised?

No. The renegotiation attack on TLS does not compromise confidentiality.