CS2107 NOTES

AY19/20 Semester 1

Hanming Zhu

CS2107 Notes
Contents

1. Security Requirements
    a. Introduction
    b. CIA
    c. Others
    d. Breaches in Recent Security News
2. Key Concepts & Basic Mechanisms of Principal Protection Mechanisms
    a. Cryptography & Encryption
        i. Cryptography?
        ii. Classical Ciphers
            1. Substitution Cipher
            2. Caesar Cipher
            3. Permutation Cipher
            4. One-time Pad
        iii. Modern Ciphers
        iv. Properties of Ciphers
        v. Block Ciphers vs Stream Ciphers
        vi. Attacks on Cryptosystem Implementations
        vii. Kerckhoff's Principle vs Security through Obscurity
        viii. Historical Facts
    b. Authentication (Entity Authentication)
        i. Password
            1. Stages
            2. Attacks
            3. Preventive Measures
            4. ATM and ATM Attacks
        ii. Biometrics
        iii. Multi-factor Authentication
    c. Authentication and Cryptography (Data Origin Authentication)
        i. Public Key Cryptography
            1. RSA
        ii. Cryptographic Hash
        iii. Data Integrity
            1. Unkeyed Hash
        iv. Data-Origin Authenticity
            1. MAC
            2. Signature
        v. Attacks and Pitfalls
            1. Birthday Attack on Hash
            2. Using Encryption for Authenticity
            3. Hashed and Salted Passwords
    d. Cryptography Part 2
        i. Public Key Distribution and Public Key Infrastructure
        ii. Certificates
            1. Certificate Authority
            2. Certificate
            3. X.509 Digital Certificate Standard
            4. How to Get a Certificate
        iii. Certificate Authority and Trust Relationship
        iv. Limitations and Attacks
        v. Strong Authentication
            1. Secret Key
            2. Public Key

        vi. Session Key
    e. Putting It All Together
        i. Securing a Communication Channel
        ii. HTTPS
3. Network Security
    a. Network Layers
        i. Network Models
        ii. Addressing Schemes
        iii. Hub, Switch and Router
    b. Network Attacks
        i. Name Resolution and Attacks
        ii. Denial of Service Attacks
        iii. Distributed Denial of Service Attacks
            1. Botnet
    c. Useful Tools
    d. Network Protection
        i. Cryptography
            1. SSL/TLS
            2. WPA2
            3. IPsec
        ii. Firewall
            1. Demilitarized Zone
        iii. Network Security Management
4. Key Concepts Part II
    a. Access Control
        i. Layering Model in Computer System Design
        ii. Access Control Model
        iii. Specifying Access Rights
            1. Access Control Matrix
            2. Access Control List
            3. Capabilities
        iv. Intermediate Control
            1. Privileges
            2. Role-based Access Control
            3. Bell-LaPadula Model
            4. Biba Model
        v. Access Control in UNIX/Linux
            1. Terminology
            2. Password File Protection
            3. File System Permission
            4. Search Path Issues
        vi. UNIX/Linux: Privilege Escalation (Controlled Invocation)
5. Software Security
    a. Overview of Software Security
    b. Computer Architecture
        i. Code vs Data
        ii. Control Flow and Program Counter
        iii. Stack
        iv. Control Flow Integrity
    c. Attacks and Vulnerabilities
        i. Printf() and Format String Vulnerability
        ii. Data Representation & Security
        iii. Buffer Overflow
        iv. Integer Overflow
        v. Code/Script Injection

        vi. Undocumented Access Points
        vii. TOCTOU Race Condition
    d. Defense and Preventive Measures
        i. Filtering / Input Validation
        ii. Safer Functions
        iii. Bounds Checking and Type Safety
        iv. Memory Protection
            1. Randomization
            2. Canary
        v. Code Inspection / Taint Analysis
        vi. Testing
        vii. Principle of Least Privilege
        viii. Patching
        ix. Summary
6. Web Security
    a. Background
    b. Security Issues and Threat Models
        i. Attackers as Another End Systems
        ii. Attackers as a Man-in-the-Middle
    c. Attacks on SSL/TLS
    d. URL and Address Bar Insecurities
    e. Cookies and Same-Origin Policy
    f. Cross-Site Scripting (XSS)
    g. Cross-Site Request Forgery (CSRF)
    h. Other Web Attacks and Terminologies

Adversary Model is a rigorous way of evaluate and describe the security of a system
- describing the class of attacks that it can prevent
    - class of attacks = attacker's goals/resources
- comparing 2 systems (S2 is more secured than S1 w.r.t the attack model)

Security Requirements

Introduction

A system can fail due to various reasons:

- Operator mistakes
- Hardware failures
- Poor implementation
- Deliberate human actions designed to cause failure

Cyber security is concerned with such **intentional failures**. We are concerned with the following:

Threat Vulnerability Control

Assets

- Hardware
- Software
- Data and information
- Reputation, which is intangible

Threat

- A set of circumstances that has the potential to cause loss or harm

leads to

Attack with control of the workstation in the lecture theatre could maliciously gather sensitive info

Vulnerability

- A weakness in the system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm

anyone can reboot the workstation from USB or Disk to gain control

Control

- A control, countermeasure, security mechanism is a mean to counter threats
- It is an action, device, procedure, or technique that removes or reduces a vulnerability

Restricting physical access to the workstation, disable USB booting

A *threat* is blocked by *control* of a *vulnerability*

There are a few other terminologies, but it sums up to:
*There is a **threat agent** that gives rise to a **threat** that exploits a **vulnerability** that leads to a **risk** that can damage an **asset** and cause an **exposure**, all of which can be counter measured by a **safeguard** that directly affects the **threat agent**.*

Within **control**, there are a few kinds.

- Level 3: Physical Controls
    - Facility protection
    - Security guards
    - Locks
    - Monitoring
    - Environmental control
    - Intrusion detection
- Level 2: Technical Controls
    - Logical access controls
    - Encryption
    - Security devices
    - Identification and authentication
- Level 1: Administrative Controls
    - Policies
    - Standards
    - Procedures

can employ a layered defense to secure system

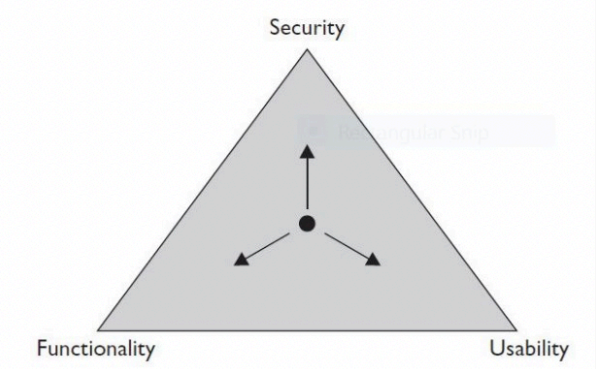
    - Guidelines
    - Screening personnel
    - Security awareness training
- Level 0: Company data and assets

Administrative controls are often the hardest and the weakest link, since they involve people.

It can be difficult to achieve security due to:
- Not considering security during early design stage of a system
- Difficult to formulate security requirements
- Various design constraints
- Difficult to verify that a design achieves the intended security requirements
- Even if the design is secure, the implementation may be wrong
- There will always be a weakest point
- The humans involved in operating the system can be exploited
    - Configuration errors
    - Mismanagement of credentials/patches/etc.

There is always a **trade-off** between security and:
- Ease-of-use: Security mechanisms interfere with working patterns users are originally familiar with
- Performance: Security mechanisms consume more computing resources
- Cost: Security mechanisms are difficult to develop

Security, Functionality and Ease-Of-Use Triangle:
The more secure something is, the less usable and functional it becomes:



We can consider potential harm to assets in two ways.
- What bad things can happen to assets
- Who or what can cause or allow those bad things to happen

CIA

Confidentiality

- The ability to ensure that an asset is viewed only by authorized parties
- Prevention of unauthorized disclosure of information

Related to: Anonymity, Privacy, Covert Channel, Plausible Deniability

Integrity

- The ability to ensure that an asset is modified only by authorized parties
- Prevention of unauthorized modification of information or processes

Changing process: eg(keylogger compromises the integrity of the application & confidentiality of password compromised)

Availability

- The ability to ensure that an asset can be used by any authorized parties
- Prevention of unauthorized withholding of information or resources

DDOS

The above are the C-I-A triad or security triad.

Others

ISO 7498-2 [ISO89] adds to them two more properties that are desirable, particularly in communication networks:

Authenticity/Authentication

- The ability of a system to confirm the identity of a sender

Non-repudiation/Accountability

- The ability of a system to confirm that a sender cannot convincingly deny having sent something

The US Department of Defense adds:

Auditability

- The ability of a system to trace all actions related to a given asset.

Looking at the CIA-triad At Another Angle

We can view it in terms of the nature of harm caused to assets, characterized by four acts:

- Interception
  - Confidentiality suffers if someone intercepts the data
- Interruption
  - Availability is lost is someone or something interrupts a flow of data or access to a computer
- Modification and Fabrication
  - Integrity can fail

Breaches in Recent Security News

Examples of Advanced Persistent Threats, which are often by an organized, well-financed and patient assailants:

- In 2012 and 2013, a series of attacks, apparently organized and supported by the Chinese government, was used to obtain product designs from aerospace companies in the United States. The stub of the attack code was loaded into the victim machines long in advance of the attack. The more complex code was then installed and helped to extract the data.
- In 2014, a series of attacks against J.P. Morgan Chase bank and up to a dozen similar financial institutions allowed the assailants access to 76 million names, phone numbers and email addresses. The attackers – and even their country of origin – remain unknown, as does the motive.

Key Concepts & Basic Mechanisms of Principal Protection Mechanisms

To manage security, we need to have a policy that specifies *who* (which subjects) can access *what* (which objects) and *how* (by which means).

Authentication

- If only Adam can access something, and someone else can impersonate Adam to do the same thing, then security fails
- Thus, we need ways to determine beyond a reasonable doubt that a subject's identity is accurate
- This property of accurate identification is called authentication

Access Control

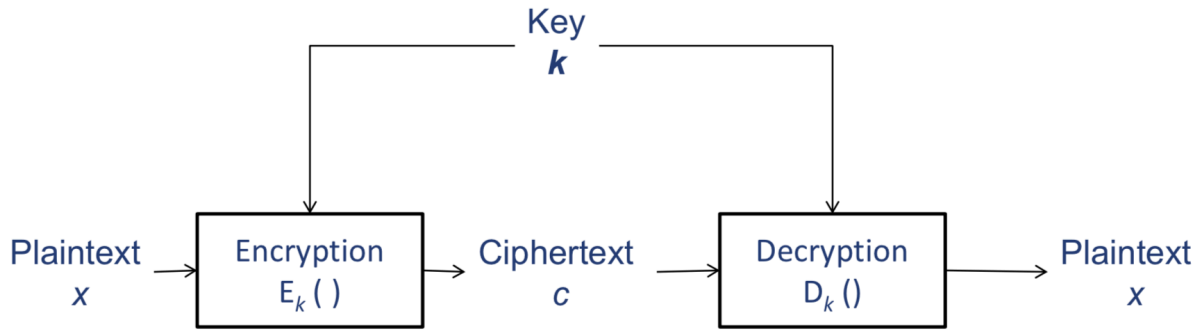
- The converse of the above must also be true: unless authorized, all other subjects must not be allowed to access the object
- After all, there's no meaning to saying that only Adam can access it if everyone else can anyways
- Thus, there needs to be a way to restrict access to only those subjects authorized
- This is thus called access control

Encryption

- Lastly, if the objects to be accessed can only be viewed and deduced by the subjects, then the chances of there being security issues are significantly lower
- The way to ensure non-intended receivers are unable to deduce the concealed objects is called encryption.

Cryptography & Encryption

An encryption scheme (also known as cipher) consists of two algorithms: encryption and decryption.



The ciphertext is transferred via insecure channel

There are two requirements: Cipher requirement

- Correctness message in == message out
  - For any plaintext x and key k, $D_k(E_k(x)) = x$
- Security
  - Given the ciphertext, it should be computationally difficult to derive useful information about the key k and plaintext x. The ciphertext should resemble a random sequence of bytes. Informal definition: Indistinguishably

There is also a performance requirement: the encryption and decryption processes need to be able to be efficiently computed (Not really tested).

Example

I can encrypt a file using Winzip and a password, and send the encrypted file to someone via an insecure channel. I must, however, send the password via a secure channel. The file then can be decrypted using the password.

NOTE: Winzip is not an encryption scheme, but merely employs standard encryptions schemes such as AES.

Cryptography?

Cryptography is the study of techniques in securing communication in the presence of **adversaries** who have access to the communication.

Although cryptography is commonly associated with encryption, encryption is merely one of the primitives of cryptography.

primitives of crypto: includes many others like
cryptographic hash
digital signature

Characters in Cryptography:

- Alice
- Bob
- Eve
- Mallory

Refer to the appendix for more information.

Classical Ciphers

Substitution Cipher

In substitution cipher, every symbol/character is substituted by another symbol/character.
Let us define some terms:

- Plaintext and Ciphertext
  - A string containing elements of a set of symbols U
  - For example: U = {'a', 'b', 'c', …, 'z', '_'}
  - The plaintext may then be "hello_world"
- Key
  - A substitution table S that represents a 1-1 mapping function from U to U

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	_
g	v	w	b	n	e	f	h	d	a	t	l	u	c	q	m	z	i	r	s	j	x	o	y	k	_	p

$S(a) = g,\ S(b) = v,\ \ldots$

The inverse of S:
$S^{-1}(g)=a,\ S^{-1}(v) = b$

- Key Space
  - The set of all possible keys
  - Substitution Cipher: it is a set of all permutations of ABC/U
- Key Space Size
  - Total number of possible keys
  - Substitution Cipher: for U with size 27, it will be 27!
  - If size of U is 50, then the key space size will be 50!
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Substitution Cipher: Key size = $\log_2(27!) = 94$ bits
  - Why? We can think of the key size as the number of possible "states" for the key, i.e. it's a specific combination of n bits of 1s and 0s to form this key out of all possible keys. As such, it's not 27, which is the number of elements, but is rather based on the key space. This will be more relevant when it comes to more modern ciphers that work more with bits.

For a cipher to be secured, exhaustive search must be computationally infeasible

Attacking Substitution Cipher

The goal for attackers is to figure out the key. If the key can be found, then the plaintext can be obtained. [redacted]

There are thus two levels of access to information:

- **ciphertext only** – a large number of ciphertexts all encrypted using the same key
- **known plaintext** – pairs of ciphertext and the corresponding plaintext

Known Plaintext Attack: Substitution Cipher

This attack occurs when the attacker has access to pairs of ciphertexts and their corresponding plaintexts. The attacker can figure out the entries in the key that are used in the ciphertext and plaintext. With a sufficiently long ciphertext, the entire key can be determined.

If the adversary can derive the key, we call the scheme **"insecure under known plaintext attack"** or **"broken under known plaintext attack"**. Hence, substitution cipher is not secure under known plaintext attack.

First Few Bytes
It is not unreasonable for the attacker to obtain at least one pair of ciphertext and plaintext, as only a small number of bytes is required. In fact, only the first few bytes may be required of the plaintext. The rest can be deduced.

These first few bytes can sometime be guessed:

- Email data: Certain words in the headers are fixed, such as “From”, “Subject” etc.
- Many network protocols have fixed headers, or only a few choices in their first few bytes of data packets
- During WWII, cryptologists exploited commonly-used words like “weather” and “nothing to report” as the known plaintext to guess the secret keys

Ciphertext Only Attack: Substitution Cipher
Method #1: Exhaustive Search
One way to find the key is to use exhaustive search, or also known as the brute force search. The idea is to simply examine all possible keys one by one until a logical plaintext is obtained. For most schemes, brute force attacks are not feasible.

However, for some modern ciphers, it may actually be possible to use exhaustive search. More sophisticated attacks exploit characteristics of the encryption scheme to speed up the process.

Consider a table size of 27 for the key. In the worst case, the exhaustive search needs to carry out $27! \approx 2^{94}$ loops, and on average $\approx 2^{93}$ loops. This is infeasible using current computing power.

Method #2: Frequency Analysis
Suppose the plaintexts are English sentences. The **letter frequency distribution** in English text is not uniform, for example, “e” is more commonly used than “z”.

If the adversary is aware that the plaintexts are English sentences, given a sufficiently long ciphertext (say around 50 characters), then an adversary may be able to guess the plaintext by:

- Mapping the frequently-occurring letters in the ciphertext to the frequently-occurring letters of English.
- Frequency analysis can be successfully carried out!

Hence, substitution cipher is **not secure under ciphertext-only attack** either, when the plaintexts are English sentences. In fact, the attack is effective on any language.

Heuristics for Frequency Analysis
The most frequently occurring character in English is either e or a space. There are also single letter words such as I and a, digraphs such as to, it, is, do, on, in, at etc., and trigraphs such as can, and, get, not, for, the, has, one, man, men etc.

Caesar Cipher (Type of Substitution Cipher)
It's a type of substitution cipher where every letter in the table is "shifted" down the alphabet/symbols by a certain number of spaces. For example, with a shift of 1, A will be represented by B in the ciphertext, B will be replaced by C, so on until Z is replaced by _.

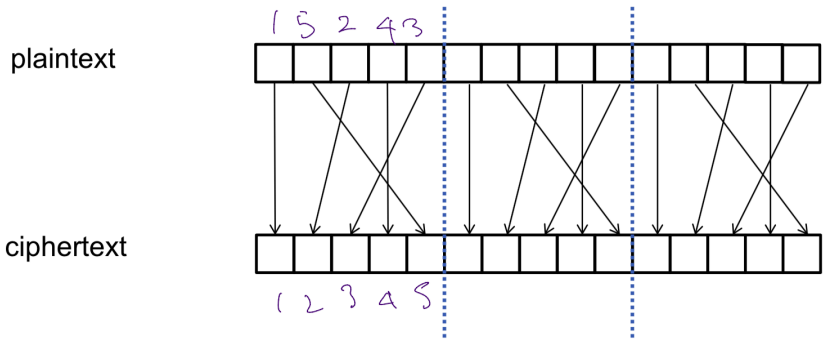
For Caesar Cipher:

- Key Space
  - The set of all possible keys
  - Caesar Cipher: Set of all possible shifts until the original table is obtained.
- Key Space Size
  - Total number of possible keys
  - Caesar Cipher: 26 or 27, depending on whether there's a delimiter character
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Caesar Cipher: Key size = $\log_2(27) = 5$ bits

Permutation Cipher/Transposition Cipher
This cipher first groups the plaintext into blocks of *t* characters, then shuffles the character within each individual block to obtain a fixed and secret permutation. It is a one-to-one function that basically maps characters within a block to another position in the resultant block.

We can express the permutation p as a sequence: $p = (p_1, p_2, p_3, ..., p_t)$. The block size *t* can also be kept secret and be part of the key, as this reduces the information attackers have access to.

Given the plaintext and the key ***t***=5, ***p***=(1,5,2,4,3):



Known Plaintext Attack: Permutation Cipher
Permutation cipher fails miserably under known-plaintext attack, as it is very easy to determine the key given a plaintext and ciphertext.

Ciphertext Only Attack: Permutation Cipher
Permutation cipher can also be easily broken under ciphertext only attack if the plaintext is an English text.

$(X \oplus k) \oplus k = X \oplus (k \oplus k) = X \oplus 0 = X$

correctness in decryption

One-Time-Pad

Given a n-bit plaintext ($x_1x_2\ldots x_n$) and a n-bit key ($k_1k_2\ldots k_n$), we can output the ciphertext, C:

$$C = (x_1 \oplus k_1)(x_2 \oplus k_2)(x_3 \oplus k_3)\ldots(x_n \oplus k_n)$$

The condition is that the key and the plaintext must be of the same length.

We can decrypt the ciphertext by XORing the ciphertext with the key once more to get the plaintext, X:

$$X = (c_1 \oplus k_1)(c_2 \oplus k_2)(c_3 \oplus k_3)\ldots(c_n \oplus k_n)$$

For this to work, we need to be able to transfer the key securely. However, if we can securely transfer the key, we might as well securely transfer the plaintext!

What is ⊕ or XOR?

The XOR or ⊕ operation between A and B is basically: (A + B) mod 2. Basically, if both A and B are the same bit i.e. both 1s or both 0s, then the result from the XOR operation will be 0. If they are different, the result will be 1.

Some interesting properties of XOR:

- Commutative: A ⊕ B = B ⊕ A
- Associative: A ⊕ (B ⊕ C) = (A ⊕ B) ⊕ C
- Identity element: A ⊕ 0 = A, where 0 is the identity element
- Self-inverse: A ⊕ A = 0

For One-Time-Pad:

- Key Space
  - One-Time-Pad: Set of all possible keys of the same length as the plaintext
- Key Space Size
  - Total number of possible keys
  - One-Time-Pad: 2^n
- Key Size or Key Length
  - Number of bits to represent a particular key
  - Caesar Cipher: Key size = $\log_2(2^n)$ = n bits

Security of One-Time-Pad

It is easy to derive the key from knowing the ciphertext and plaintext, but this key will be useless, as the key will not be used again. It is also very difficult to perform exhaustive search on it.

Furthermore, even if you know part of the key, you cannot figure out the rest i.e. it **leaks no information** of the plaintext, even if the attacker has an arbitrary running time. Hence, the one-time-pad is also called “unbreakable”, provided that a good “random” key is used.

Perfect secrecy = the chances that an attacker correctly predicts plaintext before and after knowing ciphertext are the same

Modern Ciphers

Designs of modern ciphers take into consideration of known-plaintext attack, frequency analysis and other known attacks. Some examples of modern ciphers include:

- DES (Data Encryption Standard, 1977)
  - Key length is too short
- RC4 (Rivest's Cipher 4, 1987)
  - Broken in some adoptions
- A5/1 (used in GSM, 1987)
  - Vulnerable
- A5/3 or KASUMI (3G) or SNOW 3E (LTE)
- AES (Advanced Encryption Standard, 2001)
  - Believed to be secure, classified as Type 1

Exhaustive Search, Key Length and Work Factor

If the key length is 32 bits, there are 2^{32} possible keys. As such, the exhaustive search will take:

- 2^{32} searches in the worst case
- 2^{31} searches on average (i.e. half of the possible cases)

We can thus quantify the security of an encryption scheme by the length of the key. For example, comparing scheme A with 64-bit keys and scheme B with 54-bit keys, scheme A is more secure with respect to exhaustive search.

Of course, there are attacks that are more efficient than exhaustive search. In those cases, we still quantify the security by the **equivalent exhaustive search**.

- For example, the best known attack on a 2048-bit RSA requires roughly 2^{112} searches
- Hence, we treat its security as equivalent to 112 bits, and we say that it has a key strength of 112 bits.

Work Factor is the amount of effort needed to break an encryption or mount a successful attack. For example, a 25 character message with 26 possible symbols has 26^{25} possible decipherments. If your computer could perform on the order of 10^{10} operations per second, finding this decipherment would require 10^{25} seconds, or roughly 10^{17} years.

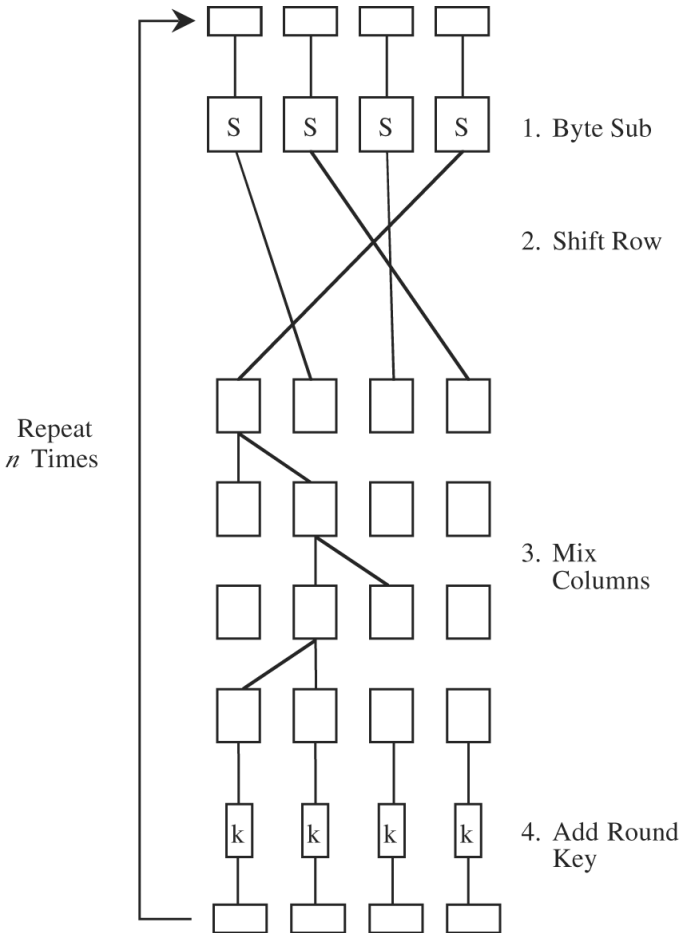
Data Encryption Standard

The key length of DES is 56 bits. While using exhaustive search to crack it did not seem feasible in the 70s, it soon became possible using distributed computing or a specialised chip.

Advanced Encryption Standard

AES is symmetric block cipher developed in 1999 by independent Dutch cryptographers. It involves:

- Byte Sub
- Shift Row
- Mix Columns
- Add Round Key
- The above repeated n times



Comparison between AES and DES

	DES	AES
Date designed	1976	1999
Block size	64 bits	128 bits
Key length	56 bits (effective length); up to 112 bits with multiple keys	128, 192, 256 (and possibly more) bits
Operations	16 rounds	10, 12, 14 (depending on key length); can be increased
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	Secret, but open public comments and criticisms invited
Source	IBM, enhanced by NSA	Independent Dutch Cryptographers

Properties of Ciphers

Confusion

An attacker should not be able to predict what will happen to the ciphertext when one character in the plaintext changes. In other words, the input (plaintext and key pair) undergoes complex transformations during encryption.

A cipher with good confusion has a complex functional relationship between the plaintext/key pair and the ciphertext.

Diffusion

A change in the plaintext will affect many parts of the ciphertext. In other words, information from the plaintext is spread over the entire ciphertext. The transformations depend equally on all bits of the input.

A cipher with good diffusion requires an attack to access much of the ciphertext in order to infer the encryption algorithm.

Block Ciphers vs Stream Ciphers

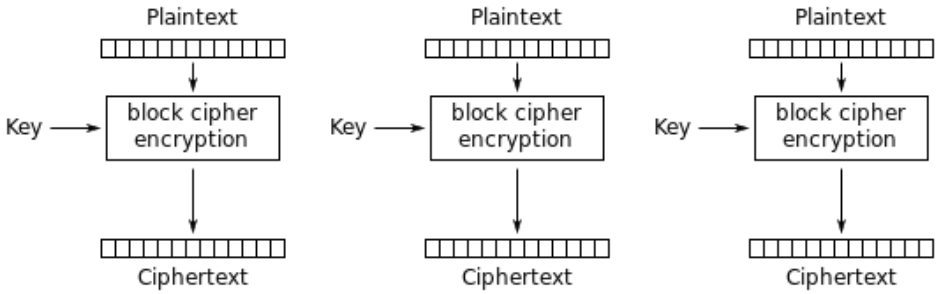
Block Cipher

The concept behind block cipher is to basically break up the plaintext into multiple blocks of some fixed-length, then encrypt each block separately. This encryption process between blocks may be independent or related, depending on the mode used.

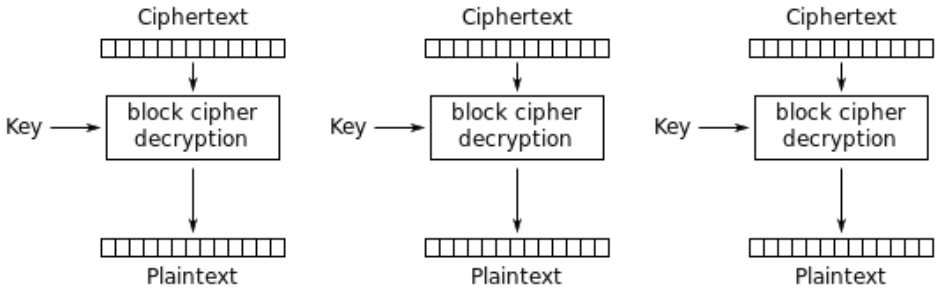
Most modes require a separate initialisation vector, which is a block of bits to randomise the encryption and produce distinct ciphertexts for the same plaintext under the same key.

Electronic Codebook (ECB)

The simplest of the encryption modes is the Electronic Codebook (ECB) mode. The message is divided into blocks, and each block is encrypted separately.



Electronic Codebook (ECB) mode encryption



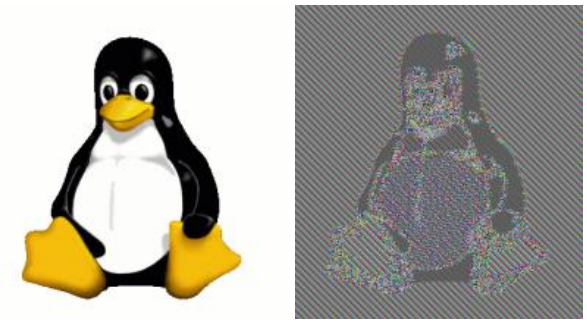
Electronic Codebook (ECB) mode decryption

The disadvantage of this method is the lack of diffusion. Because ECB encrypts identical plaintext blocks into identical ciphertext blocks, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.

A striking example of the degree to which ECB can leave plaintext data patterns in the ciphertext can be seen when ECB mode is used to encrypt a bitmap image which uses large areas of uniform colour. While the colour of each individual pixel is encrypted, the overall image may still be discerned, as the pattern of identically coloured pixels in the original remains in the encrypted version.

An encryption scheme is “deterministic” in that it always produces the same output when given the same input. In contrast, a “probabilistic” encryption scheme produces different ciphertexts even with the same input. AES is deterministic, hence we cannot employ AES with randomly-chosen IVs for each block.

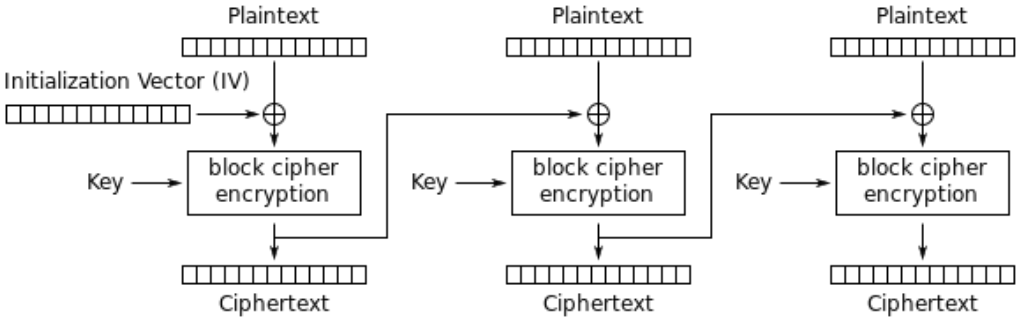
Thus, we have a mode-of-operation that describes how the blocks are to be “linked” such that different blocks at different locations would give different ciphertexts, even if they have the same content. We unfortunately cannot just choose a random IV for each block and store it somewhere, as it will significantly increase the size of the final ciphertext.



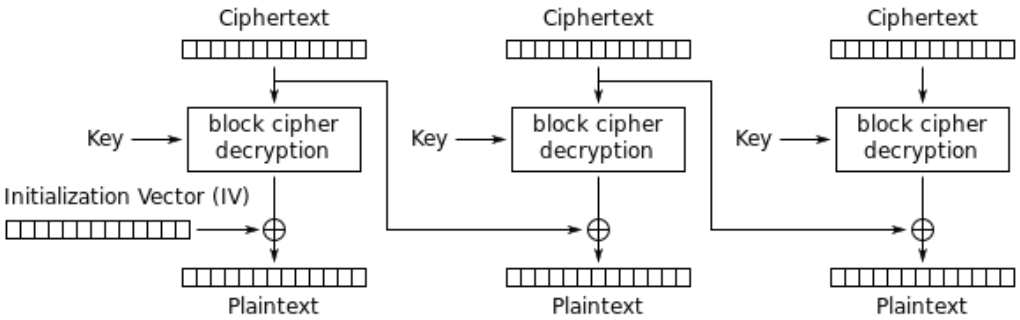
ECB mode can also make protocols without integrity protection even more susceptible to replay attacks, since each block gets decrypted in exactly the same way.

Cipher Block Chaining (CBC)

Ehrsam, Meyer, Smith and Tuchman invented the Cipher Block Chaining (CBC) mode of operation in 1976. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

If the first block has index 0, the mathematical formula for CBC encryption is

$$C_0 = E_K(P_0 \oplus IV)$$
$$C_i = E_K(P_i \oplus C_{i-1})$$

And the mathematical formula for CBC decryption is

$$P_0 = D_K(C_0) \oplus IV$$
$$P_i = D_K(C_i) \oplus C_{i-1}$$

Block Cipher thus can ensure both confusion and diffusion.

Why must we use IV for CBC (specifically)?
If no IV is used in AES-CBC, while two identical blocks within one single plaintext will be encrypted into two different ciphertext blocks, two separate but identical plaintexts will always been encrypted into the same ciphertext. This may not be desirable.

Furthermore, this makes the encryption susceptible to chosen-plaintext attacks.

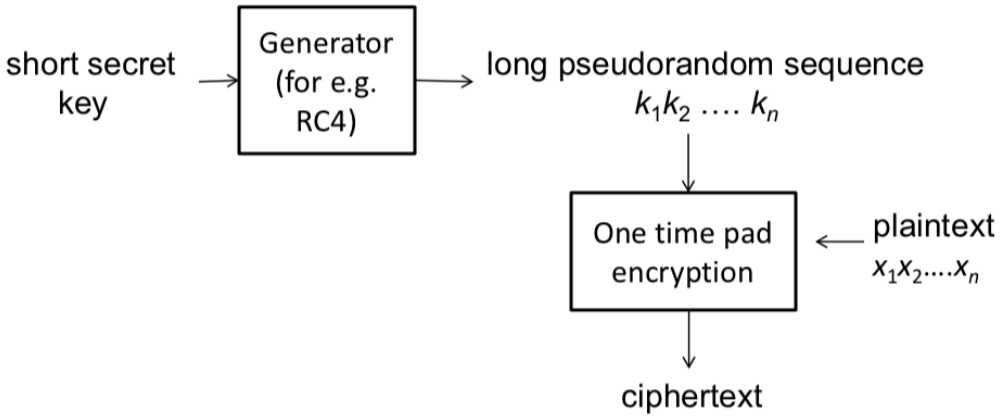
Localised Reordering Attacks
Integrity can be compromised with CBC without the receiver noticing. This can be done via reordering or modifying ciphertext blocks. Notice that in CBC decryption, an altered or corrupted ciphertext block affects the corresponding plaintext block and the following one, but the rest of the blocks remain intact. As such, the effect of a block-ordering attack will be localised.

If the plaintext has some embedded extra information about the block order, or if the out-of-order blocks can be observed "semantically" by the receiver, then this block-reordering attack can be easily detected. Yet, the reordering attack can go undetected if the plaintext represents an image or video, where a small localized modification may not be sufficiently noticeable by the receiver.

A MAC may be used on the whole file to ensure its authenticity and integrity. However, if MAC is applied on a block by block basis, then it is also possible to reorder the MAC along with the blocks to remove any initial suspicion of compromise of authenticity and integrity. Basically, don't MAC block by block.

Stream Cipher
Stream cipher is inspired by the one-time-pad. Suppose the plaintext is 2^{20} bits long, but the secret key is only 256 bits. Stream cipher will generate a 2^{20}-bit sequence from the key, and takes the generated sequence as the "secret key" in one-time-pad.

This generator will need to be carefully designed so as to give a cryptographically-secure pseudorandom sequence.

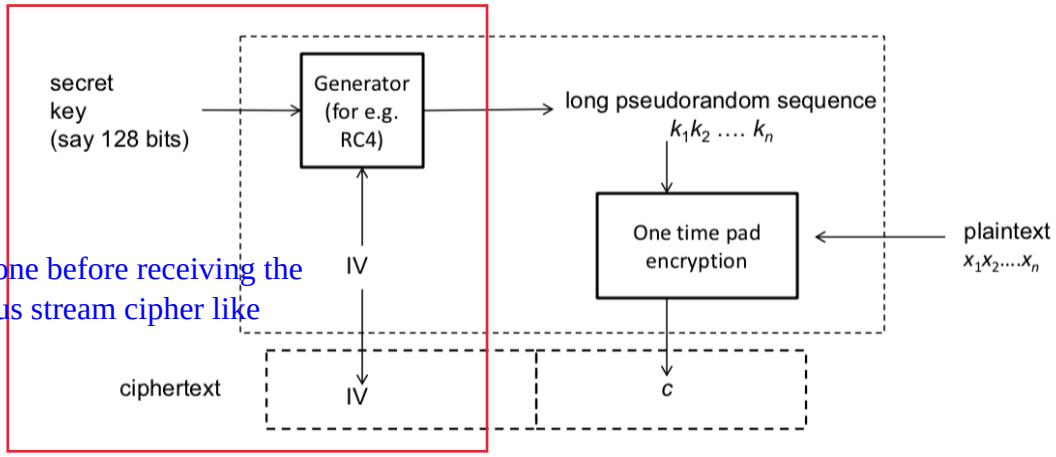


The stream cipher is thus a symmetric cipher, where the same key (or rather, same secret key) is used for both encryption and decryption.

Initial Value / Initialisation Vector (IV)
Stream cipher also has an initial value or initialisation vector, much like block cipher. The IV can be randomly chosen or obtained from a counter.

The IV is then used with the secret key (usually via some process of XOR) to generate a long pseudorandom sequence that is ultimately used to encrypt the plaintext. The IV is then appended to the front of the ciphertext in clear (i.e. non-encrypted).

For decryption, the IV is then extracted from the ciphertext and used with the key to obtain the pseudorandom sequence to decrypt the ciphertext.



This part can be done before receiving the data to encrypt, thus stream cipher like video streaming

Stream Cipher ensures only confusion and not diffusion.

Why must we use IV for Stream Cipher?
Without the IV, if an attacker can obtain two ciphertexts, they can simply XOR the two ciphertexts together.

Let's say we have ciphertext U and V, which are obtained from plaintexts X and Y respectively without using IV.

$$U \oplus V = (X \oplus K) \oplus (Y \oplus K) = (X \oplus Y) \oplus (K \oplus K) = X \oplus Y$$

The above is obtained after doing some manipulation as a result of the associative and commutative properties of XOR. We can get $X \oplus Y$.

So what if we have $X \oplus Y$?
It doesn't sound like it's anything big but let us take X and Y as black and white images, where every pixel corresponds to a bit.

Image X

Image Y

Image X ⊕ Image Y (without IV)

Image X ⊕ Image Y ⊕ K_1 ⊕ K_2

Why is it so different? This is because as K_1 and K_2 are completely different due to being produced with completely different IVs, they cannot cancel the effect of the pseudorandom sequences out. Any attempts to XOR with obtained ciphertexts would not return anything useful.

IV must be different in 2 different processes of encryption, so that the 2 pseudorandom sequences will be different. So xor both ciphertext will not cancel out the random sequences

IV makes an encryption probabilistic

Attacks on Cryptosystem Implementations

Reusing IV

Some applications overlook the importance of the IV generation, and thus can end up reusing the same IV for different plaintexts. For example, one might derive the IV from a filename to encrypt a file F. However, it is not uncommon for different files to have the same filename.

The below is the RC4 flaw from Microsoft:
In this report, we point out a serious security flaw in Microsoft Word and Excel. The stream cipher RC4 with key length up to 128 bits is used in Microsoft Word and Excel to protect the documents. But when an encrypted document gets modified and saved, the initialization vector remains the same and thus the same keystream generated from RC4 is applied to encrypt the different versions of that document. The consequence is disastrous since a lot of information of the document could be recovered easily.

For the Cipher Block Chaining (CBC) mode of AES, the IV needs to be unpredictable to prevent a certain type of attack. The Browser Exploit Against SSL/TLS Attack (BEAST) exploits this, as when encrypting multiple packets, the IV of a packet would be the last ciphertext block of the previous packet, which is visible to anyone.

Reusing One-Time-Pad Key

Similar to reusing IV, if the one-time-pad key is reused and the attacker notices it, it is possible to easily decrypt entire messages. For example, the Venona project saw the decryption of encrypted messages from the Soviet Union by the US.

Predictable Secret Key Generation

When coding, we may use packages such as java.util.Random despite the fact that it is not actually secure. java.security.SecureRandom is preferred. This is because:

1. **Size:** A Random class has only 48 bits whereas SecureRandom can have up to 128 bits. So the chances of repeating in SecureRandom are smaller.

2. **Seed Generation:** Random uses the system clock as the seed/or to generate the seed. So they can be reproduced easily if the attacker knows the time at which the seed was generated. But SecureRandom takes Random Data from your OS (they can be interval between keystrokes etc – most OS collect these data and store them in files – /dev/random and /dev/urandom in case of linux/solaris) and use that as the seed.

3. **Breaking the code:** In case of random, just 2^48 attempts are required, with today's advanced cpu's it is possible to break it in practical time. But for SecureRandom 2^128 attempts will be required, which will take years and years to break even with today's advanced machines.

4. **Generating Function:** The standard Oracle JDK 7 implementation uses what's called a Linear Congruential Generator to produce random values in java.util.Random. Whereas Secure Random implements SHA1PRNG algorithm, which uses SHA1 to generate pseudo-random numbers. The algorithm computes the SHA-1 hash over a true random number(uses an entropy source) and then concatenates it with a 64-bit counter which increments by 1 on each operation.

5. **Security:** Consequently, the java.util.random class must not be used either for security-critical applications or for protecting sensitive data.

Designing Your Own Cipher

You're asking for trouble unless you're really an expert in this field.

Kerckhoff's Principle
"A system should be secure even if everything about the system, *except the secret key*, is public knowledge"

vs

Security through Obscurity
To hide the design of the system in order to achieve security.

Which one is better?
Against Obscurity
RC4

- Was introduced in 1987 and its algorithm was a trade secret
- In 1994, a description of its algorithm was anonymously posted in a mailing group.

MIFARE Classic

- A contactless smartcard widely used in Europe employed a set of proprietary protocols/algorithms
- However, they were **reverse-engineered** in 2007
- It turned out that the encryption algorithms were already known to be weak (using only 48bits) and breakable

For Obscurity
Usernames

- They are not secrets
- However, it is not advisable to publish all the usernames

Computer Network Structure & Settings

- E.g. location of firewall and the firewall rules
- These are not secrets, and many users within the organization may already know the settings
- Still, it is not advisable to reveal them

The actual program used in a smart-card

- It is not advisable to publish it
- If the program is published, an adversary may be able to identity vulnerabilities that were previously unknown, or carry out side-channel attacks
- A sophisticated advisory may be able to reverse-engineer the code nevertheless

So should we use obscurity?
In general, obscurity can be used as one layer in a "defence in depth" strategy. It could deter or discourage novice attackers, but is ineffective against attackers with high skill and motivation. The system must remain secure even if everything about it, except its secret key, becomes known.

Historical Facts

Cryptography

- Cryptography is closely related to warfare and can be traced back to ancient Greece
- Its role became significant when information is sent over the air
- WWII: Famous encryption machines include the Enigma, and the Bombe (that helped to break Enigma)

Modern Ciphers

DES (Data Encryption Standard):

- 1977: DES, 56 bits
- During cold war, cryptography, in particular DES was considered as “munition”, and subjected to export control. Currently, export of certain cryptography products is still controlled by US.
- 1998: A DES key broken in 56 hours
- Triple DES (112 bits) is still in used

$E_{k_1}(E_{k_2}(x)) \neq E_{k_3}(x)$

AES (Advanced Encryption Standard):

- 2001: NIST. 128, 192, 256 bits

RC4

- 1987: Designed by Ron Rivest (RSA Security), initially a trade secret
- 1994: Algorithm leaked
- 1999: Used in widely popular WEP (for WiFi); WEP implementation has 40 or 104-bit key
- 2001: A weakness in how WEP adopts RC4 is published by Fluhrer, Mantin, Shamir
- 2005: A group from FBI demonstrated the attack
- Afterward: Industry switched to WPA2 (with WPA as an intermediate solution)

Meet in the middle attack: this is a known plaintext attack

- compute 2 sets of ciphertext and plaintext
- 1 = from plaintext then brute all cipher
- 2 = from cipher then brute all plain
- compare the middle version to find the similar element

This works because you take the cipher from the 1st part then encrypt it again

2^{k+1} operation 2^{k+1} space

Padding Oracle Attack = using the PKCS#7 padding standard (Padding bytes are just a repeat [x x x x 03 03 03])
- can be used to break AES CBC (Ciphertext only attack)

With the (IV || c), which in CBC will be block i & i-1
to find plaintext pt = c XOR iv
01 = c XOR iv XOR correct value
repeat continuously

The idea that the oracle leaks 'bits' information by showing a different error message should be avoided

Tutorial Learnings

Clock Cycles

If 512 clock cycles are needed to test whether a 64-bit cryptographic key is correct, when given a 64-bit plaintext and its corresponding ciphertext, and a 4GHz single-core processor has 2^{32} cycles per second, then it takes 2^{41} seconds to check all 2^{64} keys, which is approximately 2^{16} years, which is approximately 64K years, where $1K = 2^{10}$.

If we have 1024 servers, each with a quad-core processor, we have 2^{12} processors, and the time needed is now 16 years.

Time-Space Trade-off

If the question asks about a certain ciphertext that is transmitted with a lot of 0s at the front and 1s at the back, and the key used is short, we can actually construct a table to allow for quick lookup, in exchange for large amount of storage space used.

In the tutorial question, we construct a table of $Enc_k(000…000)$ (64 bits of 0s) for all possible 2^{32} values of (k XOR IV), which is 32-bit long. Thus the derived table will take 2^{32} x 64 bits = 32GB. If (k XOR IV) is also stored, then another 16GB is needed. Upon receiving the first 64 bits, we look it up and determine the employed (k XOR IV).

We don't even need to know the original k, as we just need the (k XOR IV) to decipher the ciphertext.

Compression

Compressing an encrypted file will yield very little or no compression gain. This is since the encrypted file will resemble a random sequence, which is a requirement of a good encryption scheme. A compression algorithm, which takes advantages of repeating patterns, therefore will not work well on an encrypted file.

Password Length

Even if we use a 256-bit AES key, if the key is generated from hashing a password of known length, known keyspace and with a known hash function, then an attacker can easily reverse engineer the password to decrypt the file, without needing to try all 2^{256} possible keys. For example, if the password is made of all digits and is 6 digit long, then there are only 10^6 possible passwords.

Ciphertext XOR-ing Attack

If two plaintexts were encrypted using stream cipher with the same secret key and IV, by XOR-ing the two ciphertexts, we actually get the same result as when we XOR the two plaintexts, as the secret key and IV XORs itself and cancels out.

Authentication (Entity Authentication)

Authentication

Authentication is the process of assuring that the communicating entity or the origin of a piece of information is what it claims to be.

There are thus two types of authentication:

Entity Authentication

This is for connection-oriented communication, where the communicating entity is an entity involved in a connection. Some common mechanisms for this type of authentication are thus password, challenge and response.

This may be required to ensure authenticity over various communication channels:

- Phone call from the Police Department: Is this authentic?
- Logging in to LumiNUS via my account: Is the server that I'm interacting with authentic? Why would the LumiNUS' server be convinced that the entity logged in is the authentic me?
- Connecting to the "NUS" WiFi: Was that WiFi access point authentic?

Data-origin Authentication

This is for connectionless communication, where the communicating entity is the origin of a piece of information. Data-origin authenticity implies data integrity. Some common mechanisms are MAC and digital signature.

This may be required to ensure authenticity of digital data and physical documents:

- Submitted a MC for a test: Is this MC authentic?
- Obama's birth certificate: Is it authentic?
- Email from lecturer saying that the quiz is cancelled: Is the email authentic?

Authenticity vs Integrity

Authentic is an adjective to say that the claimed entity/origin is assured by supporting evidence. Authenticity is the condition of being authentic.

Authenticity and integrity are thus related. In the context of an insecure channel, we can say that a message that has been modified in transit means that it no longer comes from its original source.

In other words, a message whose integrity is compromised also means that its authenticity is compromised. As such, **data-origin authenticity implies data integrity.** But **data integrity does not imply data-origin authenticity.** Authenticity is thus a stronger requirement than integrity.

Protocol = A set of steps where the both parties need to follow these rules

Password

A password is part of an authentication system that usually consists of:

- Identity (Identification)
  - The identity need not to be kept secret
  - It can be a username in a system, bank account number, customer ID etc.
- Password (Authentication)
  - The password is kept secret
  - Only the authentic user and the server knows it
  - The fact that an entity knows the password **implies that it is either the server or the authentic user**

The process can be broken down into:

Process	Provided by	To Answer	Attributes	Uniqueness Requirement
Identification	Principal	"Who are you"	Public assertion	Yes (locally)
Authentication	Principal	"How can you prove that it is you?"	Secret response	No
Authorization	System	"What can I do?"	Token/Ticket, Access Control	-

Authorization goes more into access control, which basically has the system determining what the user can or cannot do.

There are two stages to the password authentication system:

Stage 1: Bootstrapping

During bootstrapping, the server and the user establishes a common password. The server then keeps track of a file recording the identity (i.e. userid, username) and the corresponding password.

A quick and painless, or even no bootstrapping process can be good, but care must be taken so that the resultant password is still effective.

The password establishing can be done by either:

- The server / user chooses a password and sends it to the user / server through another communication channel. For example, NUSNET sends the password via physical mail.
- A default password is set

Might lead to some sort of issue when most users do not change - so need to force users to change

Stage 2: Password-based Authentication

This is the authentication that most of us are used to.
User → Server: Hanming is my username.
Server → User: Ok, Hanming. What is your password?
User → Server: hellohello is my password.
Server: Ok. You are indeed Hanming.

similar to a handshake

Alternatively, it could be through a SMS to a server:
userid: hanming@nus.edu.sg password: hellohello instruction: unsubscribe

What the system does is to check against its password file to find the username and password.

Weak Authentication System
The password system is classified as a weak authentication system. A weak authentication system is one that is vulnerable to **replay attack**: information sniffed from the communication channel can be used to impersonate the user at a later time.

This is in contrast with strong authentication, where information sniffed during the process cannot be used to impersonate the user.

Attacks on the Password System

1. Attack the Bootstrapping

It is possible for an attacker to attack the bootstrapping. They can target the use of default passwords (real example: IP Security Cameras).

They can also aim to intercept the password in the process of being sent via an alternative channel (e.g. stealing the postal mail containing the NUSNET password).

Why do manufacturers not use individual passwords?
For the device manufacturer – cost will increase since there is a need to:

- print the password on each equipment/manual/case
- record the passwords in a password database
- have a customer service line or Web site to help users retrieve their device's default password.

For device users – usability will decrease if a user is unable to find or retrieve the default password. This may lead to a bad product review, and also affect manufacturer's sales.

2. Searching for the Password

Guessing
It is not uncommon for people to set passwords that are based off certain information about themselves and their loved ones, for the greater ease of remembering. An attacker can thus gather information about the user and attempt to guess the password.

There are two types of password guessing:

Online Password Guessing
In an online version of the password guessing attack, an adversary tries to guess a password by logging to a server. This version of the attack is less powerful than the offline version since the adversary is limited by maximum allowed login attempts, whereby no such limitations exist in the offline version.

Offline Password Guessing
In the offline version an adversary gets in the possession of some password-related data of a user (e.g. hashed password) and thus iteratively tries to guess a password and verify its hashed version with the intercepted one. In this version of the password guessing attack, the adversary is only limited with the processing power of its own machine, meaning there exists no other lock to stop him/her from trying to guess the password, since the attack is done locally on the adversary's machine. Because hardware is getting ever more powerful according to Moore's Law, the adversaries can use such enhanced power for more guessing attempts per second.

Exhaustive Attacks
Exhaustive search is as previously introduced. The attacker can try all possible combinations and see if any of them works. However, this can take a great amount of time, as there are passwords of different lengths and character sets.

This makes exhaustive search on passwords not very feasible. There is also no reason to try some combinations, given the low probability of the common user using certain characters.

Table 3-1. Possible Keyspaces by Password Length and Character Set Size

Char. Set Size	Character Types				Password Length				
	Digits	Letters	Symbols	Other	4	8	12	16	20
10	Decimal				$1*10^4$	$1*10^8$	$1*10^{12}$	$1*10^{16}$	$1*10^{20}$
16	Hexa-decimal				$7*10^4$	$4*10^9$	$3*10^{14}$	$2*10^{19}$	$1*10^{24}$
26		Case-insensitive			$5*10^5$	$2*10^{11}$	$1*10^{17}$	$4*10^{22}$	$2*10^{28}$
36	Decimal	Case-insensitive			$2*10^6$	$3*10^{12}$	$5*10^{18}$	$8*10^{24}$	$1*10^{31}$
46	Decimal	Case-insensitive	10 common[7]		$4*10^6$	$2*10^{13}$	$9*10^{19}$	$4*10^{26}$	$2*10^{33}$
52		Upper and lower			$7*10^6$	$5*10^{13}$	$4*10^{20}$	$3*10^{27}$	$2*10^{34}$
62	Decimal	Upper and lower			$1*10^7$	$2*10^{14}$	$3*10^{21}$	$5*10^{28}$	$7*10^{35}$
72	Decimal	Upper and lower	10 common		$3*10^7$	$7*10^{14}$	$2*10^{22}$	$5*10^{29}$	$1*10^{37}$
95	Decimal	Upper and lower	All symbols on standard keyboard		$8*10^7$	$7*10^{15}$	$5*10^{23}$	$4*10^{31}$	$4*10^{39}$
222	Decimal	Upper and lower	All symbols on standard keyboard	All other ASCII characters	$2*10^9$	$6*10^{18}$	$1*10^{28}$	$3*10^{37}$	$8*10^{46}$

Dictionary Attacks

This is a form of exhaustive search where the attacker can restrict the search space to a large collection of probable passwords. These words tend to be words from the English dictionary, known compromised passwords, dictionaries from other languages etc.

It is thus possible to carry out a hybrid attack, i.e. a mix of exhaustive search and dictionary attack. For example, we could try all combinations of 2 words from the dictionary, and exhaustively try all possible capitalizations of each word, substituting ‘a’ with ‘@’, etc.

3. Stealing the Password

Eavesdropping: Sniffing and Keylogging

The most primitive method is the shoulder surfing, also known as the look-over-the-shoulder attack. Other ways include the interception or sniffing of communication channels, as some systems and protocols simply send the password over a public network in clear (i.e. unencrypted). For example, FTP, Telnet and HTTP do that.

It is also possible to sniff a wireless keyboard by picking up on the signals transmitted, or even via listening to the sound of the keyboard! side channel attack

A keylogger captures/records the keystrokes and sends the information back to the attacker via a “covert channel”. This can be done via both software and hardware. For example, some computer viruses are designed to capture keyboard inputs. There are also hardware keyloggers that are plugged in and intercepts the transmissions by the keyboard.

Phishing

Phishing attacks ask for the user’s password under some false pretense. The user is then tricked to voluntarily send the password to the attacker over the network. This is typically done through emails, but can also be carried out over phone calls.

Spear phishing is the process of targeting a particular small group of users and is an example of targeted attacks. This can make the phishing attempt more realistic and believable, and much more effective.

Phishing attacks are basically social engineering attacks, which, in the context of information security, refers to the psychological manipulation of people into performing actions or divulging confidential information.

Some prevention means are to educate users via phishing drills. There are also phishing repository sites out there. However, it can be very tricky to accurately determine if an unsolicited email is a phishing attempt.

Login Spoofing
Attackers can also display “spoofed” login screens to trick users. To prevent this from happening, some systems have a **secure attention key** or **secure attention sequence** (e.g. Ctrl + Alt + Del for Windows). When this sequence is pressed, the system starts the trusted login processing.

Password Caching
When using a shared workstation, information keyed in may be cache. The next user can then see the cache. To prevent this, clear the browser’s cache and close the browser when using a shared workstation.

Insider Attacks
An example of this would be a malicious system admin who steals the password file. Another possibility is the compromising of a system admin’s account, leading to the loss of the password file.

So far, online needs 29 bits of entropy = alpha numeric characters
offline = 128 bits

Preventive Measures

1. User Side

Use a Strong Password

The most direct method to reduce the chances for an attacker is to improve the strength of the password. One way is to make sure the password is randomly chosen, perhaps using an automated password generator.

There is, however, always a trade-off between security and convenience. A password with really "high entropy" is often very difficult to remember.

Some methods to strengthen passwords:

- Mnemonic Method: Pbmbval! (Please be my best valentine!)
- Altered Passphrases: Dressed*2*tge*9z
- Combining and Altering Words: B@nkC@mera

Password Ageing

It is often recommended for users to regularly change passwords. However, many believe that frequent changes of passwords actually reduce security, due to people following some standard transformations e.g. increasing numbers, adding extra digits etc.

2. Admin/Server Side

Limited Login Attempts

The admin can add delays in the login process, security questions, and can also lock the account after a few failed attempts. All these reduce the efficiency of the attacks.

Password Checker

Check for weak passwords during registration or password changes.

Password Metering

Indicate weak, average and strong passwords.

Password Usage Policy

The organization can set rules to ensure that users use strong passwords, such as a minimum character count etc.

Protect the Password File

This is the important prevention measure for the admin/server side. As the password file stores usernames and passwords, any leakage due to insider attack, accidental leakage, system hacking etc. can be disastrous.

There have been many cases where weakly protected password files are leaked, leading to a large number of passwords being compromised. Hence, we need to add an additional layer of protection to the password file.

Password Hashing

The way to protect passwords is generally to hash passwords before storing them in the password files. This is different from encryption, as there is a way to recover the password from the ciphertext. However, for a cryptographically secure hash, it is not feasible to recover the password from its hashed value.

During authentication, the password entered by the entity is hashed, and compared with the value stored in the password file.

We also cannot have the same password being hashed to the same value for two different usernames. This allows attackers to obtain all un-hashed passwords of the same value

simply by comparing hashed values. The way to prevent this is to add salt, i.e. a random string of characters to the front of the password before hashing it. This salt is randomly generated for all users and is also stored in the password file.

Security Questions
Security questions can be viewed as a mechanism for fallback mechanism or a self-service password reset. It enhances usability, as users can still login even if they have lost their passwords. It reduces costs, as it reduces the work of a helpdesk. However, this weakens security, since attackers now have another mean of obtaining access.

However, it can be difficult to provide a perfect security question. The criteria are as follows:

- Memorable
  - If users cannot remember their answers, then there is no point to having security questions
- Consistent
  - The user's answers should not change over time
  - For example, the answer to "What is your current job?" may change over time
- Nearly universal
  - The question should be applicable to a wide audience if possible
- Safe
  - The answer should not be something easily guessed or researched i.e. not publicly available information

An open-ended question can make a good question. It is also good to highlight that the user does not have to faithfully give the actual or right answer.

Nowadays, a second factor (e.g. SMS) or secondary email address is preferred over security questions for resetting passwords.

Security - Cost - Usability tradeoff

ATM and ATM Attacks
An ATM card and the PIN actually work very similarly to how a username and password would. To get authenticated, a user has to present a card and the PIN. The ATM card contains a magnetic stripe, which stores the user account ID. It essentially simplifies the process of inputting the account ID; instead of having to key it in, the user just has to insert the card. The PIN then plays the role of the password.

Data is encoded into the magnetic stripe using **well-known standards**. Given a valid card, an attacker can "copy" the card by reading the info from the card and write it to a spoofed card.

ATM Skimming
An ATM skimmer steals the victim's account ID and PIN. A skimmer usually consists of a card-reader attached on top of an existing ATM reader, a camera overlooking the keypad or a spoofed keypad on top of the existing keypad, and some means to record and transmit the information back to the attacker.

With this method, the attacker can spoof the card and obtain the PIN.

Preventive Measures
To prevent the above from happening, we can:

- Install anti-skimming devices that prevent external card readers from being attached onto the ATM
- Shield the keypad
- Increase awareness of users
- Use newer chip-based (EMV) cards, which use encryption

Biometrics

Biometrics use unique physical characteristics of a person for authentication.

During enrollment: a reference template of a user's biometric data is constructed and stored, similar to bootstrapping in the password system.

During verification: the biometric sample data of the person-in-question is captured and compared with the template using a matching algorithm. The algorithm decides whether to accept or reject the authentication.

Biometrics can be used for both:
Verification: 1 to 1 comparison to determine whether the person is the claimed person
Identification: 1 to many comparison to identify the person from a database of many persons

Differences between Biometric and Password

- Biometrics cannot be changed, while a password can
- There is no need to remember anything for biometrics, while we need to remember our passwords
- There is a probability of error for biometrics, while there is zero non-matched rate for passwords
- It is not possible for a person to "transfer" the biometrics to another (in general), while users can pass their password to others

Matching Algorithm: Similarity / Inexact Matching

As there are inevitable noises in capturing biometric data, there will be errors in the matching decision. Based on these error rates, we get the following:

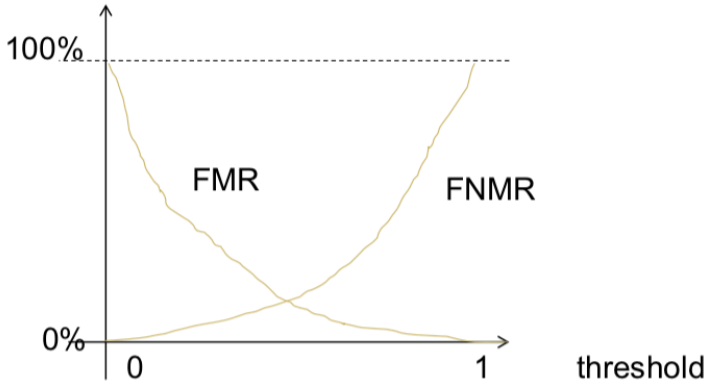
$$False\ Match\ Rate = \frac{Number\ of\ successful\ false\ matches\ (B)}{Number\ of\ attempted\ false\ matches\ (B+D)}$$

$$False\ Non-Match\ Rate = \frac{Number\ of\ rejected\ genuine\ matches\ (C)}{Number\ of\ attempted\ genuine\ matches\ (A+C)}$$

where A, B, C, D refer to the following:

	Accepted / Match	Rejected / Non-Match
Genuine Attempt	**A**	**C**
False Attempt	**B**	**D**

There is thus a threshold for the matching algorithm, which when adjusted, would adjust the FMR and FNMR as well. By having a lower threshold, our matching algorithm is more relaxed, while a higher threshold would result in a more stringent matching algorithm.



Other Types of Error and Rates
Equal Error Rate (EER)
The rate when FMR = FNMR

Failure-to-enroll Rate (FER)
Some users' biometric data may not be able to be captured for enrollment, due to reasons such as injury

Failure-to-capture Rate (FTC)
A user's biometric data may fail to be captured during authentication. For example, the finger of the user may be too dirty or too dry etc.

How Does Fingerprint Scanning Work?
What the scanner looks out for are feature points, which are basically edges or branches in the fingerprint. For a fingerprint, feature points are also known as minutiae.

The performance of the fingerprint as a biometric depends on the quality of the scanner, as EER can range from 0.5% to 5%.

Other Forms of Biometrics
Some other forms are palm prints, palm veins, hand geometry, face, iris, retina, DNA.

Security of a Biometric System
We assume the scanner to be secure, with no tampering possible. Even then, it may still be possible to spoof biometric data, similar to how movies show it. As such, some systems actually have "liveness detection" to verify if the entity scanned by the scanner is indeed "live" instead of being a spoofed material, e.g. photograph.

Multi-Factor Authentication

Multi-factor authentication, or n-factor authentication, requires at least 2 different authentication “factors”.

Some commonly used factors are:

1. What you know
    a. Password
    b. PIN
2. What you have
    a. Smart Card
    b. ATM Card
    c. Mobile Phone
    d. Security / OTP Token
3. Who you are
    a. Biometrics

Other possible factors are:

4. Where you are
5. What you do

It is called 2-factor authentication if 2 factors are required. The Monetary Authority of Singapore expects all banks in Singapore to provide 2-factor authentication for e-banking.

OTP Token

The One-Time Password (OTP) token is a hardware that generates a one-time password (i.e. a password that can only be used once). The server usually issues a OTP token to the user, which contains a secret key that the server knows. The user will also set a password to their account.

There are two ways for the OTP to be generated:

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

The authentication process is as such:

- User “presses” the token, which then computes and displays a one-time-password
- User sends username, password and OTP to server
- Since the server has the “secret key”, the server can also compute the OTP
- Server verifies that both the OTP and the password are correct

Authenticator Application

There is also a rising trend of using a mobile application as a “soft” OTP token. This can be a custom app or an authenticator app.

SMS – Is it secure?

It is also possible to register 2FA using SMS. To register, the user gives the server his mobile phone number and password.

The authentication process is as such:

- User sends their password and username to the server
- Server verifies that the password is correct and sends a one-time-password (OTP) to the user via SMS
- User receives the SMS and enters the OTP

- Server verifies that the OTP is correct

However, SMS OTP is **not secure**. The National Institute of Standards and Technology's (NIST's) "Digital Authentication Guideline" also recommended the deprecation of SMS OTP.

Some possible security threats are:
- Interception of cellular networks' channel
- SMS messages are stored as plaintext by the Short Message Service Center (SMSC)
- Malware / Trojan on smartphones
- Singapore government has access to all of our messages

However, it is acknowledged that doing SMS OTP is still better than just using username and password.

Smartcard + Fingerprint (Door Access System)
This is usually set up by having the server issue a smartcard to the user, with the smartcard containing a secret key K. The user then enrolls their fingerprint.

The authentication process is as such:
- User inserts smartcard into the reader.
- The reader obtains the user identity and verifies if the smartcard is authentic. If so, continue.
- User presents fingerprint to the reader.
- The reader performs matching to verify that the fingerprint is authentic. If so, grant access.

Very often, the information on the user identity, the secret key K, and the fingerprint template are not stored in the reader. The reader has a secure communication channel to a server that stores these information. We thus assume that the reader and server are secure, i.e. attackers are unable to access them.

A smart card has this security feature where even if an attacker has physical access to the card, it is extremely difficult, if not impossible, to extract a secret stored in the card.

Tutorial Learnings

Keeping UserID Secret

A system that checks for the existence of a userid and prompts the user specifically about it makes it easier to be attacked. Suppose an attacker needs x guesses for the userid, and y guesses for the password, in order to log in as a valid user. The attacker will thus need x + y guesses to get in, whereas he will need xy guesses on a system that prompts for both userid and password if the combination is incorrect.

Notice that, in practice, userid is not a secret. But for a layered defense, it is good to hide it as much as possible. Ultimately, the security still relies on the password.

Using Social Information for Authentication

This can be good in that there is little to no bootstrap process, and the information needed to login are easy to remember.

However, there is a potential privacy leak. If an attacker knows the information (which is not difficult), he can access the system. If he knows all but one piece of information, he can probe the login screen to figure out the last piece.

Ultimately, it is a trade-off between usability and security.

SMS – No Information or Complete Information?

M1: No information, e.g. “Enter OTP 132373 to complete your transaction.”
M2: Complete information, e.g. “You have requested to transfer $10,000 from account no 1388293-43-23 to account no 12398-234-A2. Enter OTP 132373 to complete your transaction.”

Firstly, a SMS is not encrypted in an “end-to-end” fashion, and there could be multiple telco entities handling the SMS. You can also consider the scenario where the PC is in an Internet cafe, and the cafe owner could be malicious, or honest but curious.

Secondly, notice that arguing “a system M1 is more secure than M2” requires us to find an attack that M1 can prevent, but not M2.

M1 is more secure: if the mobile phone is lost or somehow an adversary can get hold of the mobile phone, and the previous messages are not deleted, then the adversary can see the undeleted transaction details. This will lead to a privacy leakage. Alternatively, if the adversary can tap into the SMS communication channel, then he will able to capture the transaction history.

M2 is more secure: if a malware resides, or is purposely planted in the PC, then the message displayed to the user can be different from the transaction actually sent to the server. M2 can help the user verify if his/her money is transferred correctly.

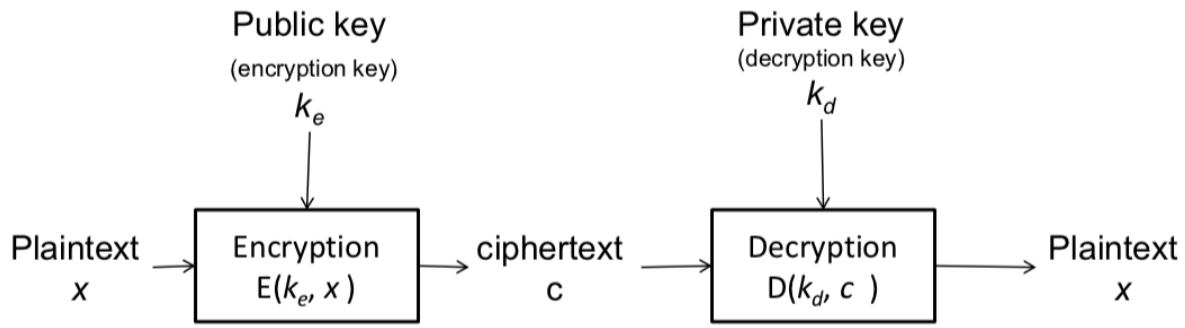
But if we can decide on the message format, we can show partial account information instead, which is what the industry is doing today.

Authentication (Data Origin)

Public Key Cryptography

Symmetric Key vs Public Key

A symmetric key encryption scheme uses the same key for both encryption and decryption, while a public key or asymmetric key scheme uses two different keys for encryption and decryption.



In truth, there is no need to clearly indicate which key is the encryption key and which is the decryption key, as both keys are able to perform either decryption or encryption, just not both. As such, the following two statements are both valid:

$$P = D\left(k_{PRIV}, E(k_{PUB}, P)\right)$$
$$P = D\left(k_{PUB}, E(k_{PRIV}, P)\right)$$

Since attacker can encrypt to test, PKC must be secured against a chosen plaintext attack

In fact, decryption can even be done before encryption. All these will work, as it is effectively impossible to deduce one key from the other. It is also important for the plaintext to be difficult to determine without the private key, even with the public key and ciphertext.

Implied requirement is that difficult to get private key from public key

As such, typically, the public key is the one shared with everyone, while the private key is kept secret by the user. It is thus possible for a person sending a message to me to encrypt it with my public key and send it to me for me to decrypt with my private key. The reverse also works – I can "decrypt" it with my private key for people to "encrypt" with my public key. However, generally, if I were to send something to somebody, I would encrypt it with the other party's public key.

Why use Public-Key Scheme (PKS)?

Without PKS, every individual would need to keep a symmetric secret key pair with everyone else. If the number of entities is large, the number of keys to keep track would be very huge as well. Any new parties joining in would result in a lot of secret key establishing as well, which is high difficult.

The total number of keys needed without using PKS and using symmetric-key setting is n(n-1)/2 = $O(n^2)$.

With public-key setting, every entity just needs its own public and private key. The total number of keys is 2n = O(n).

Rivest-Shamir-Adleman Algorithm

The RSA Algorithm was proposed in 1977, after Diffie and Hellman published the concept of the public-private key cryptosystem in 1976.

Textbook RSA

Keys for the RSA Algorithm are generated by:

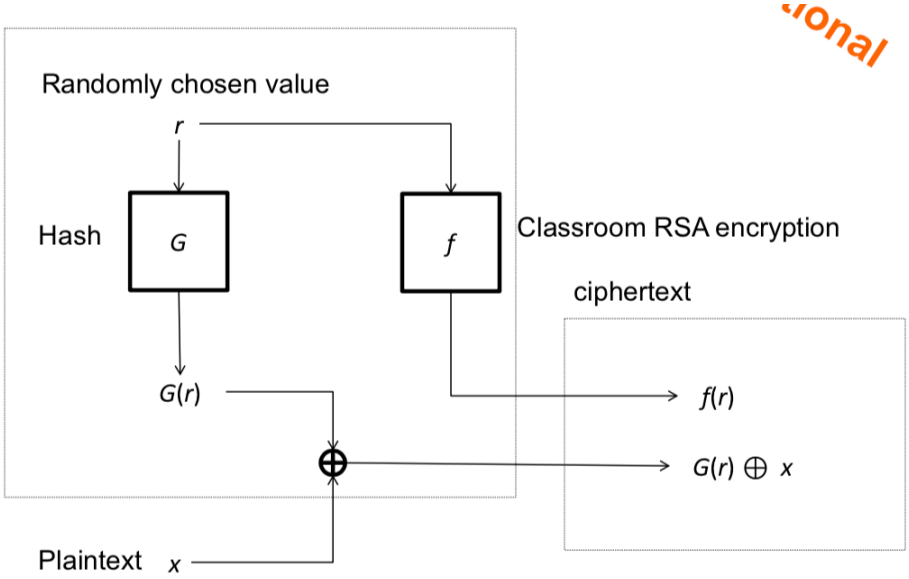
1. Choose two distinct prime numbers p and q
    a. They should be chosen at random and should be similar in terms of magnitude, but differ in length by a few digits
    b. Can be found efficiently via the primality test
    c. They are to be kept secret
2. Compute n = pq
    a. n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length
    b. n is released as part of the public key
3. Compute λ(n). After some calculations, we get λ(n) = lcm(p − 1, q − 1).
    a. λ(n) is kept secret
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and λ(n) are coprime.
    a. e is released as part of the public key
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo λ(n).
    a. This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$.
    b. d is kept secret as the private key exponent.

For encryption and decryption:
(e,n) is the public key, (d,n) the private one.

- To encrypt a message m, compute $c \equiv m^e \bmod n$.
- To decrypt a ciphertext c, compute $m \equiv c^d \bmod n$.

However, RSA has the same problem that symmetric-key encryption faces, which is that the encryption of the same plaintext at different times will give the same ciphertexts. Thus, some form of IV or padding is required to introduce an element of randomness.

The standard Public-Key Cryptography Standards (PKCS) #1 adds “optimal padding” to achieve the above.



To make RSA non-deterministic

Issues that RSA faces

Poor Efficiency and Performance

- RSA is significantly slower than AES (10,000x slower)
- A 128-bit AES has the same key strength as a 3072-bit RSA

To overcome this issue:
When a large file F is to be encrypted under the public-key setting, for efficiency, the following steps can be carried out:

1. Randomly choose an AES key k
2. Encrypt F using AES with k as the key to produce the ciphertext C
3. Encrypt k using RSA to produce the ciphertext q
4. The final ciphertext consists of two components: (q, C)

The reverse is to be done for decryption:

1. Decrypt q using RSA to produce the key k
2. Decrypt C using AES with k as the key to produce plaintext F

Security of RSA

RSA is not necessarily "more secure" than AES. It can be shown that getting the private key from the public key is as difficult as factorization. But it is not known whether the problem of getting the plaintext from the ciphertext and public key is as difficult as factorization.

As mentioned before, the "textbook" RSA has to be modified so that different encryptions of the same plaintext lead to different ciphertexts, and such modifications are not straightforward (e.g. PKCS#1).

Strengths of PKC

The main strength is the "public-key" setting, which allows an entity in the public to perform an encryption without a pre-established pair-wise secret key. This "secret-key-less" feature is also very useful for providing authentication. In practice, PKC is rarely used to encrypt a large data file.

The homomorphic property or RSA is useful for blind signature (authentication)

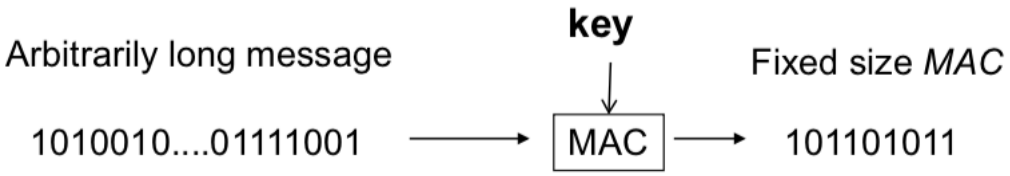
- but vulnerable because the Jacobi symbol of plaintext and ciphertext is the same, thus will leak 1 bit of information of plaintext if attacker knows modulo(n) and ciphertext(c)

Cryptographic Hash
A hash is a function that takes in an arbitrarily long message as input and outputs a fixed-size **digest**.

This is basically the hash function that we have learnt before.

Security Requirements
- Preimage Resistant or One-way
  - Given a digest d, it is difficult to find a m such that h(m) = d
  - In other words, it is difficult to reverse-engineer
- Second-preimage resistant
  - Given m_1, it is difficult to find a second preimage $m_1 \neq m_2$ such that $h(m_1) = h(m_2)$
- Collision-resistant
  - It is difficult to find two different messages $m_1 \neq m_2$ that "hashes" into the same digest, i.e. $h(m_1) = h(m_2)$

Message Authentication Code (MAC) aka Keyed-Hash
A keyed-hash is a function that takes an arbitrarily long message and a secret key as input, and outputs a fixed-size MAC.



Security Requirements
- Without knowing the key, it is difficult to forge the MAC

should be unforgeable: After seen multiple valid pairs of messages and corresponding mac, difficult for attack to forge the mac of a message not seen before

Popular Hashes
SHA-0, SHA-1, SHA-2, SHA-3

History:
1. **SHA-0** was published by NIST in **1993**.
   a. It produces a 160-bits digest.
   b. It was withdrawn shortly after publication and superseded by the revised version SHA-1 in 1995.
   c. In 1998, an attack that finds collision of SHA-0 in 261 operations was discovered. (Simply using the straight forward birthday attack, a collision can be found in 2160/2 = 280 operations).
   d. In 2004, a collision was found, using 80,000 CPU hours.
   e. In 2005, Wang Xiaoyun et al. (Shandong University) gave an attack that can find collisions in 239 operations.
2. **SHA-1** is a popular standard. It produces 160-bits message digest. It is employed in SSL, SSH, etc.
   a. In 2005, Xiaoyun Wang et al. gave a method of finding collision in SHA-1 using 269 operations, which was later improved to 263.
   b. In Feb 2017, researchers from CWI and Google announced the first successful SHA1 collision (https://shattered.io) – see the next 2 slides
   c. In 2001, NIST published SHA-224, SHA-256, SHA-384, SHA-512, collectively known as SHA-2.
   d. The number in the name indicates the digest length.

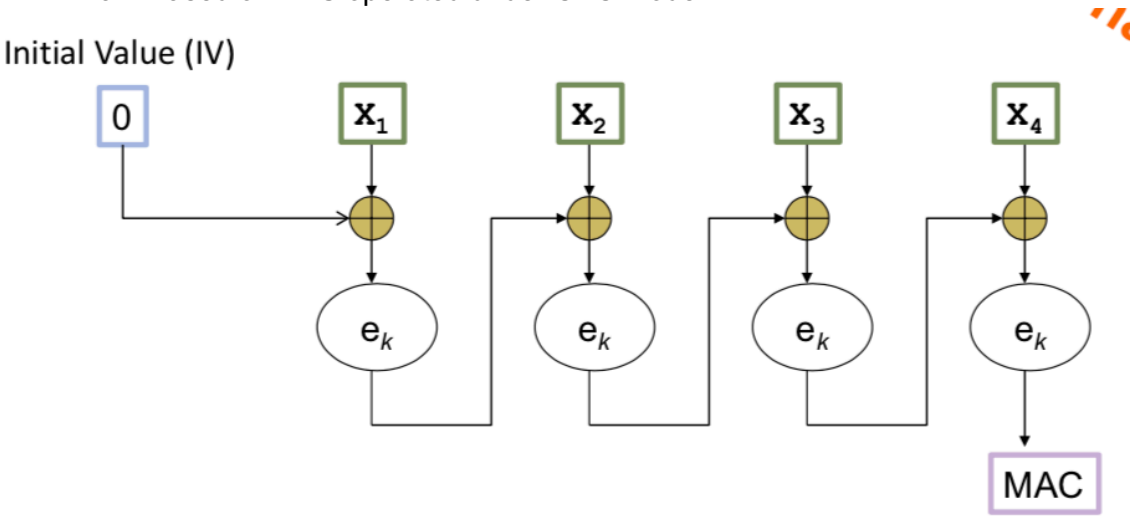
e. No known attack on full SHA-2 but there are known attacks on "partial" SHA-2, for e.g. attack on a 41-round SHA-256 (whereas the full SHA-256 takes 64 rounds).
f. In Nov 2007, NIST called for proposal of SHA-3. In Oct 2012, NIST announced the winner, Keccak (pronounced "catch-ack").

Popular but Obsolete Hashes

- **MD5**
  a. Designed by Rivest, who also invented MD, MD2, MD3, MD4, MD6.
  b. MD6 was submitted to NIST SHA-3 competition, but did not advance to the second round of the competition.
  c. MD5 was widely used. It produces 128-bit digest.
  d. In 1996, Dobbertin announced a collision of the compress function of MD5.
  e. In 2004, collision was announced by Xiaoyu Wang et al. The attack was reported to take one hour.
  f. In 2006, Klima give an algorithm that can find collision within one minute on a single notebook!
  g. Security implication: *Do not* use MD5!

Popular Keyed-Hash (MAC)

1. CBC-MAC
   a. Based on AES operated under CBC mode



2. HMAC
   a. Based on any iterative cryptographic hash function (e.g. SHA)
   b. Hashed-based MAC
   c. Standardized under RFC 2104
   d. Still deterministic, however, despite its more implicated operations

$$HMAC_k(x) = \textbf{SHA-1}\,((K \oplus opad) \,||\, \textbf{SHA-1}\,((K \oplus ipad) \,||\, x))$$

where:

opad =	3636...36	(outer pad)
ipad =	5c5c...5c	(inner pad)

(Note: the values above are in hexadecimal)

|| *means concatenation*

Data Integrity

How do we know if an email we received or a software we downloaded is authentic? Or that a file F that we need is authentic?

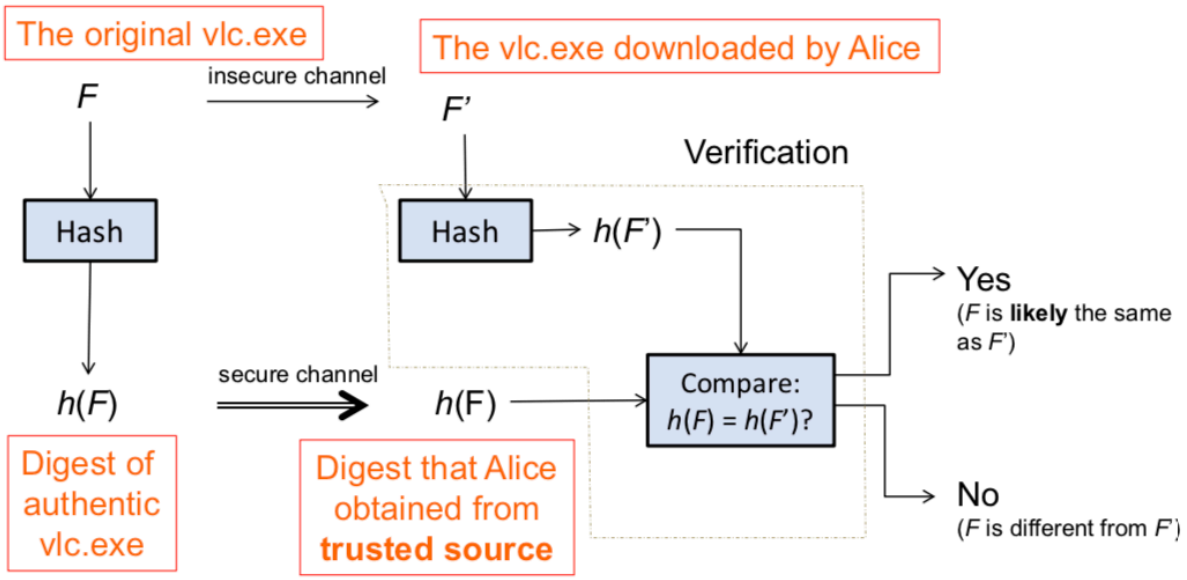
Unkeyed Hash for Integrity Protection

Let us assume that there is a secure channel to send a short piece of information. We can then carry out the following steps:

- Let F be the original file
- We obtain the digest h(F) from the secure channel
- We then obtain the file, F', whose origin claims that it is F
- We can compute and compare the two digests h(F) and h(F')
    a. If h(F) = h'(F), then F = F' (with a very high confidence)
    b. Else if h(F) ≠ h'(F), then the file integrity is compromised

We may even argue that with the digest, the verifier can be assured that the data is authentic, and thus the authenticity of the data origin is achieved. Nonetheless, in many literature and documents, when there is no secret key involved, the hash function only provides integrity, not authenticity. There are even sources that argue that even integrity is not ensured by unkeyed hash.

This is because a man-in-the-middle can potentially edit the original message before rehashing it using the same hash function, then sending the new digest over. This is why there is a need for the digest to be sent separately via some secure channel e.g. digest posted on a webpage which is sent over HTTPS, which many would argue defeats the purpose of ensuring integrity already.



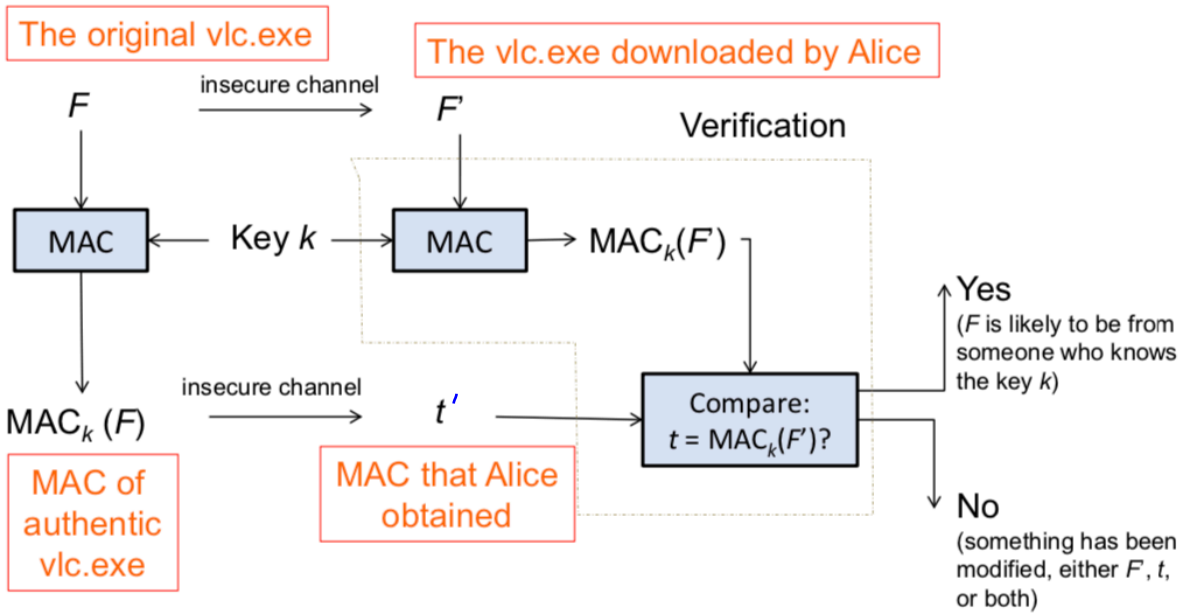
As an attacker, the aim to create a file that will collide with the original file hash, and then upload

Data-Origin Authenticity

The previous example of using an unkeyed hash can have us obtaining a digest that is faked. So what can we do when we do not have a secure channel to deliver the digest?

Message Authentication Code

MAC can help to ensure integrity and authenticity as only the sender will know of the secret key used for the MAC. By comparing $MAC_k(F)$ (which can be sent through an insecure channel) and $MAC_k(F')$, we can easily determine if the file is from the right person. If the MAC matches, F is likely to be from someone who knows the key k. Else, something has been modified, either F', the $MAC_k(F)$ sent through the insecure channel, or both.



Note that there are actually no concerns about confidentiality for this situation. The file F can be sent in clear. Usually, the MAC is appended to F and the two are stored in a single file or transmitted together through a communication channel. Hence, the MAC is also called the **authentication tag**. Later, an entity who wants to verify the authenticity of F can carry out the verification process using the secret key.

Digital Signature

Digital Signature is essentially the asymmetric-key version of MAC. The steps are as follows:

- The original file, F, is first hashed using a hash function
- The hash value is then passed through the signature function together with the private or signature key k_{PRIV} to produce signature s, the signature of the authentic file
- The signature is then appended to the original file F and sent to the recipient via an insecure channel
- The recipient then passes the received signature s' and the public or verification key k_{PUB} into the verification function
- The verification function will produce a hash value
- The file F' is then pass through a hash function to return a hash value
- The hash values are compared to see if the file has been modified, or if the digital signature is valid
- Since digital signature is created by 'private' key of signer and no one else can have this key; the signer cannot repudiate signing the data in future

Signature scheme achieves non-repudiation = since only 1 person (private key) can sign and create the signature, everyone else only has public key can only verify

Symmetric key version cannot achieve since can blame the verifier
to be the one who signed it since both will have the same key

The computed signature is typically appended to F and stored as a single file. Subsequently, the authenticity of F can be verified by anyone who knows the signer's public key. This is because the valid signature can be computed only be someone knowing the private key.

In addition to authenticity, signature scheme also achieves non-repudiation: assurance that someone cannot deny his/her previous commitments or actions.

Why hash the file before signing? (Not tested)
This is due to the greater efficiency. As the hash value is a unique representation of the data, it is sufficient to sign the hash in place of data. Let us assume RSA is used as the signing algorithm. As discussed in public key encryption chapter, the encryption/signing process using RSA involves modular exponentiation.

Signing large data through modular exponentiation is computationally expensive and time consuming. The hash of the data is a relatively small digest of the data, hence signing a hash is more efficient than signing the entire data.

Attacks and Pitfalls

Birthday Attack on Hash

The birthday paradox has us seeing 23 people being the minimum number of people required before the probability of any pair of students sharing the same birthday exceeds 50%. The reason for this unintuitive number is due to us not looking at comparing one individual's birthday with the rest, but about comparing between every possible pair of students.

This can be applied to the hash function. Suppose we have M messages, and each message is tagged with a value randomly chosen from {1, 2, 3, 4, ...,T}.

$$M > 1.17\sqrt{T}$$

If M >. 1.17 $T^{0.5}$, then there is >50% chance of a pair of messages being tagged to the same vzlue. This result is the mathematics behind a **birthday attack** on a hash function, as the arobability of finding a collision is rather high. Generally, the probability that a collision occurs can be approximated with 1 – exp(-M^2 / 2T).

$$1 - e^{-M^2/2T}$$

For the birthday attack variant, we are looking at calculating the probability of a random set of elements sharing at least one element with another set of distinct elements, with both sets being subsets of an even larger set. Let the second set be S, with k distinct elements, where each element is a n-bit binary string. We now select m random n-bit binary strings. The probability that at least one of the randomly chosen strings is in S is larger than:

Approx: $p = 0.63$ $\quad k \cdot m = 2^n \Rightarrow m = \frac{2^n}{k}$ $\qquad 1 - 2.7^{-km2^{-n}}$ $\qquad m = \frac{\log(0.5)}{-k \times 2^{-n} \log(2.7)}$

This has a serious consequence on the digest length required by a hash function to be collision resistant. When the key length of a symmetric key is 112, the corresponding recommended length for digest is at least 224.

Using Encryption for Authenticity

It is common for people to claim that their communication channel is secure as they use a certain encryption scheme that provides a high level of security. This is in fact a false sense of security, as encryption schemes merely provide confidentiality. However, it does not provide the integrity and authenticity required for communication channels.

For example, a server that communicates instructions via SMS does not provide integrity and authenticity if it simply encrypts its messages. A secure design could use MAC instead of encryption.

Even then, there are opportunities for "replay attacks". For example, if I intercept one of the messages, even with a MAC appended, I can simply repeatedly send the message I intercepted, which will result in some command or instruction being repeatedly executed. To prevent replay attacks, a cryptographic nonce is required, which is a random or pseudo-random number issued that prevents old communication from being reused.

Hashed and Salted Passwords

Covered before under passwords.

Time - space / Time - Memory tradeoff:

- If allowed the time to precompute a table (using space, we can reduce the amount of time required to break)
- Can use rainbow table
- Or Hash chain (Space requires about n/3) since linked list

(Time: Increase from normal table)

(Accuracy: approx half since there will be collisions (due to reduce function) within the chains since cannot be unique)

Tutorial Learnings

Pseudo Randomness of Hash Functions

Cryptographic hash functions, such as SHA-1, are often employed to generate "pseudo-random" numbers. The SHA hash function family is thus deterministic if the seed is known. Thus any form of key or IV generation with hash functions and a known seed is not secure, as any attacker can simply repeat the same process to obtain the exact same key and IV.

Using a Random Function that Depends on Time

Assume

```
srand(time(NULL));
int s = rand();
```

If an adversary knows the time, which is possible in practice, then he/she can derive the key. Otherwise, if the adversary knows only the approximate time, still he/she can exhaustively search all possible times.

Even if the adversary knows nothing about the time, it is still possible to brute force the variable s. This is since int data type in C is only either 2-byte (16-bit) or 4-byte (32-bit) long depending on the platforms used.

More can be found in the section on Encryption.

Hashing IVs to Get Keys

Highly insecure, since IVs are known by the attacker. Let's say we generate a random number in a secure manner, then we hash it to get a 160-bit string. We use the first 128-bits as the IV, then we hash the 160-bit string again to get another 160-bit string, which we use the first 128-bits of as the key.

Now it seems like it may be secure, but with knowledge of the IV, we just need to generate all possibilities of the remaining 32-bits of the first 160-bit string i.e. 2^{32} possibilities. Then we hash it and check if the 128-bit key can decrypt the ciphertext.

Requirements of Public Keys

Two requirements must be met:

1. It is difficult to get the private key from the public key.
2. It is difficult to get the plaintext from the public key and the ciphertext.

Many people argue that RSA is secure just because of requirement 1, but that is an incomplete justification.

Using HMAC with the same key for plaintexts

This allows attackers to actually observe the MACs of two plaintext blocks to infer about the plaintext blocks. If the MACs are the same, then the plaintext must be identical.

Collision-resistant(h) → one-way(h)?

To prove this, we can prove the contrapositive, i.e. ¬one-way(h) → ¬collision-resistant(h). To show this, suppose that there is a fast algorithm A that, when given x, can easily find a m such that h(m) = x, i.e. h is not one-way since it is not computationally difficult to get a m from x.

Then we can show that we can easily derive another fast algorithm B that can find a collision with high probability, i.e. B can find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$. We would then have proven that if h is not one-way, it is also not collision resistant.

B is as such:

- Randomly pick a m
- Compute x = h(m)
- Compute the inverse m' = A(x)
- If m' ≠ m, then return (m, m'), else repeat.

There is a good probability that algorithm B will soon stop, since the set of hash values is finite, and the set of messages is much larger than the set of hash values. Therefore, many different messages would be hashed into a single hash value, i.e. collisions should occur frequently.

Example Question for Birthday Attack Variant
Consider this scenario. There are $2^7 = 128$ students in the class. Each student is assigned a secret 16-bit ID, which is known only by the student and the lecturer. The probability of correctly guessing the ID of a particular student is thus 2^{-16}, which is very small. One day, the lecturer posted a multiple choice question during the lecture, and asked each student to write down the answer on a piece of paper together with his/her 16-bit ID, and insert it into a box in the lecture hall.

Suppose you know the correct answer, and want to generously share it with your classmates. You quickly write down the correct answer on 32 pieces of paper, each with a randomly chosen ID, and covertly insert them into the box.

What is the probability that at least one student benefits from your attempted good deed?
Let n = 16, k = 2 , and m = 2. By applying the given formula,

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

we have the probability ≈ 0.06 (which is low).

How many pieces of paper do you need to submit so that the probability is more than 0.5?
We need to find m such that the above expression equals to 0.5. We thus get m ≈ 362. Hence, you need to submit more than 362 pieces of paper.

Note: There is also a quick approximation to find m simply by setting $km = 2^n$, so that the probability becomes $1 - 2.7^{-1} = 0.63$. We thus can find the approximate $m = 2^{16-7} = 2^9 = 512$.

This approximation technique is useful in case you don't have a calculator with you, or you just want to have an approximate m.

Cryptography Part 2

Public Key Distribution

After we have decided on using public and private keys, one issue remains – there needs to be a way to transfer the public key securely. A secure channel is needed.

If a public key is distributed insecurely, we may face a man-in-the-middle attack, where Mallory may intercept communications between Alice and Bob, intercepting one party's public key and providing his own public key to the other party instead. The attacker can then read all messages between Alice and Bob and modify them as required.

Only when a public key is securely distributed can we use it for encryption (confidentiality) and signature verification (authenticity). As such, there is seemingly no difference between symmetric-key setting and public-key setting, since there is still a need for a secure channel to send a key from Alice to Bob (secret key for symmetric-key setting, public key for public-key setting).

Nevertheless, the public-key setting is arguably easier to handle:

Requirement Aspect	Symmetric-Key Setting	Public-Key Setting
No. of times a secure channel is required	For every pair of entities: $n(n-1)/2$ $O(n^2)$	Each entity just needs to securely broadcast its public key: n $O(n)$
Item to be transmitted	Shared secret key	(Publicly-published) public key
Secure channel timing requirement (e.g. when a new entity needs to securely talk to another entity)	A secure channel is needed to deliver both parties' newly-set secret key	Previously announced public key(s) just need to be made accessible to a party requiring the key(s)

Three Different Methods for Public Key Distribution

1. Public Announcement

The owner of the public key can broadcast their public key. Many list their Pretty Good Privacy (PGP) public key on their websites. However, this method is not very standardized and there is no systematic way to find or verify the public key when needed.

Furthermore, there is a need to trust the entity distributing the public key. For example, we need to trust that the website holding someone's key can be trusted.

2. Publicly Available Directory

We can potentially list all names/email addresses and public keys in a public-key directory server. By querying the name or email address, we can access the public key.

However, it is not easy to have a "secure" public directory. How does the server verify that the information provided in a request to post a public key is correct? Eventually, some entity will need to be trusted.

There is also a possibility of the server that the user is accessing being a non-authentic one i.e. the user is actually visiting a spoofed server.

3. Public Key Infrastructure

Public Key Infrastructure (PKI) is a standardized system that distributes public keys. The objectives of PKI are:

- To make public-key cryptography deployable on a large scale
- To make public keys verifiable without requiring any two communicating parties to directly trust each other
- To manage public and private key pairs throughout their entire key lifecycle

There is thus a concept of trust that needs to be managed.

Trust

A central issue of any digital exchange is trust. How do you know that a certain web page really belongs to the company that it claims to be from? We may try to determine whether something online can be trusted based on its appearance of authenticity and outside information.

Public keys are insufficient on its own as we do not know whether the public key we obtained is authentic. We thus need a trustworthy means to bind a public key with an identity. The best way to do so is to have a common respected individual or authority that our trust is based on.

Public Key Infrastructure is centered around two important components – Certificate and Certificate/Certification Authority (CA). PKI provides a mechanism for trust to be extended in a distributed manner, starting from the root CA. It is an arrangement that binds public keys with the respective identities of entities e.g. people and organizations.

Certificates and Trust

Certificate Authority

A certificate authority (CA) issues and signs digital certificates. The cryptography involved is that of digital signatures. It keeps a directory of public keys, and it also has its own public-private key pair. We assume that the CA's public key has been securely distributed to all entities involved.

Most operating systems and browsers have a few pre-loaded CA's public keys. They are known as the "root" CAs. There are stringent operational requirements for a CA – for example, it must pass the WebTrust audit.

Certificate

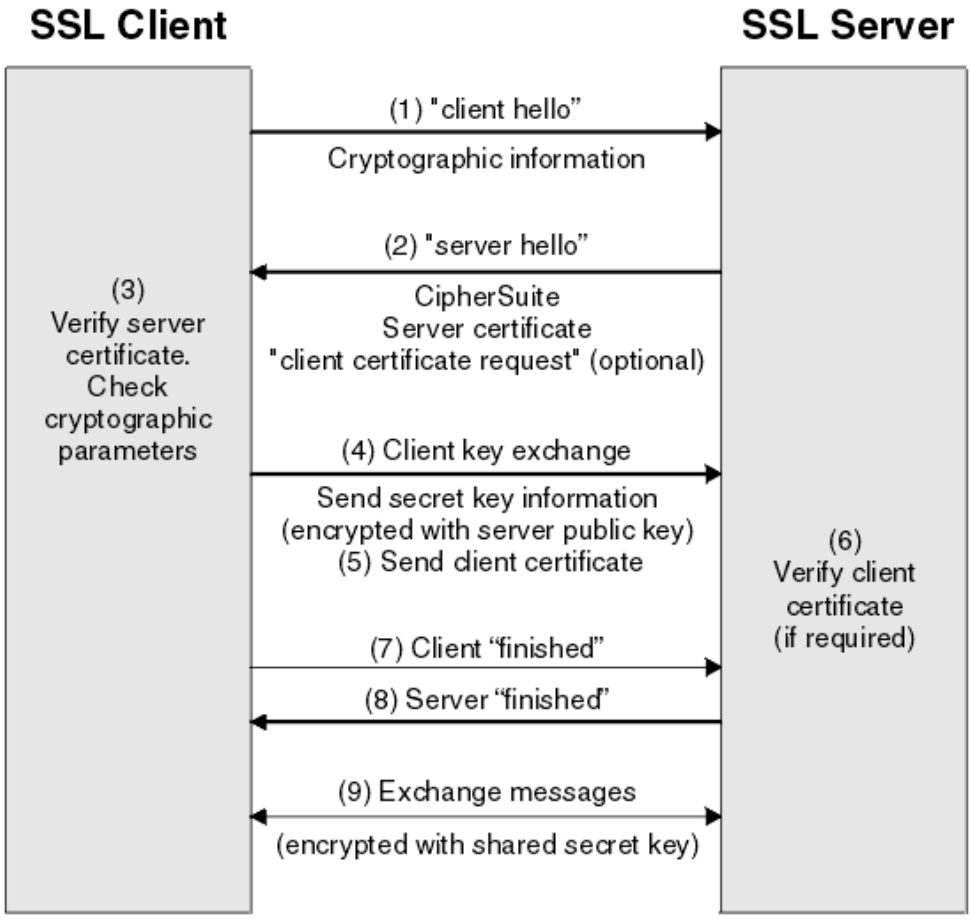
A certificate is a digital document that contains at least the following main items:

1. Identity of the owner
2. Public key of the owner
3. Valid time window of this certificate
4. Digital Signature of the Certificate Authority (entity that issued the certificate)
    a. This Digital Signature is computed from the CA's private key.

There is other additional information based on the intended purpose of the certificate.

A certificate is widely used by Internet applications – Secure Sockets Layer (SSL) / Transport Layer Security (TLS), Secure/Multipurpose Internet Mail Extensions (S/MIME), Secure Shell (SSH), etc.

This is how the SSL or TLS handshake looks like:



In general, these are the steps:
- ClientHello from client to server
- ServerHello from server to client
- Certificate from server to client
- ServerHelloDone from server to client

Advantages of Certificate-based PKI
In a certificate-based PKI, the CA acts as the directory server. Instead of constantly needing to query for the public key from the server, the CA issues a certificate to entities to bind their public keys with them. This overcomes two problems:
- The directory server is a bottleneck
- The verifier needs to have online access to the directory server at the verification point

Previously, if I were to send an email signed with my private key to someone, that someone would need to query the directory server to ask for my public key.

With certificates, I will send an email signed with my private key along with my certificate. The recipient would verify that the signature in the certificate is indeed signed by the CA, and since no one except the CA can produce the valid signature, the authenticity of the information in the certificate is as good as coming from the CA.

This certificate that I would send is obtained from the "offline" CA beforehand. There is still a need to check that the certificate has not been revoked, which is done with the Online CRL Distribution Point or OCSP Responder.

<u>X.509 Digital Certificate Standard</u>
The bodies that can decide the X.509 Standard are:
- International Telecommunications Union's Standardisation Sector (ITU-T)
    - Was here before the Internet era
    - Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
- Public-Key Infrastructure Working Group (PKIX)
    - IETF working group that creates Internet standards on issues related PKI based on X.509 certificates

The structure of an X.509 v3 digital certificate is as such:
- Certificate
    - Version Number
    - Serial Number
    - Signature Algorithm ID (Note Signature Algorithm below too)
    - Issuer Name
    - Validity period
        - Not Before
        - Not After
    - Subject Name
    - Subject Public Key Info
        - Public Key Algorithm
        - Subject Public Key
    - Issuer Unique Identifier (optional)
    - Subject Unique Identifier (optional)
    - Extensions (optional)
- Certificate Signature Algorithm
- Certificate Signature

This certificate is bound to a particular Distinguished Name (DN). A DN has these common attribute types:

- Country (C)
- State (S)
- Locale (L)
- Organization name (O)
- Organizational unit name (OU)
- Common name (CN) - Common name can be an individual user or any other entity, e.g. a web server

How do I get a certificate?

Get a Root CA to issue you one:

- Paid ones: $10 - $50 / year (not costly)

“Let's Encrypt” provides (basic) TLS certs at no charge:

- Launched in April 2016
- A certificate is valid for 90 days
- Its renewal can take place at anytime
- Automated process of cert creation, validation, signing, installation, and renewal
- No of issued certs: 1M (March 2016) to 380M (Sept 2018)

Firefox Telemetry:

- 77% of all page loads via Firefox are now encrypted
- It is predicted that it will reach 90% by the end of 2019

Certificate Authority and Trust Relationship

The CA is also responsible of verifying that the information in a certificate is correct. For example, when issuing a certificate for a certain website domain, the CA should check that the applicant owns the domain name.

This may involve some manual checking and can be costly, especially for **Extended Validation SSL** (EV SSL) certificates. Those certificates are the highest class of certificates with a lot of stringent checks done, and they activate both the padlock and green address bar in major browsers.

Different SSL Certificates

1. Domain Validation (DV) SSL certificate
    a. Issued if the purchaser can demonstrate the right to administratively manage a domain name
    b. For example, replying to an email sent to the email contact in the domain's whois details, publishing a DNS TXT record
2. Organization Validation (OV) SSL certificate
    a. Issued if the purchaser additionally has an organization's actual existence as a legal entity
3. Extended Validation (EV) SSL certificate
    a. Issued if the purchaser can persuade the certificate provider of its legal identity
    b. This process involves manual verification checks by a human

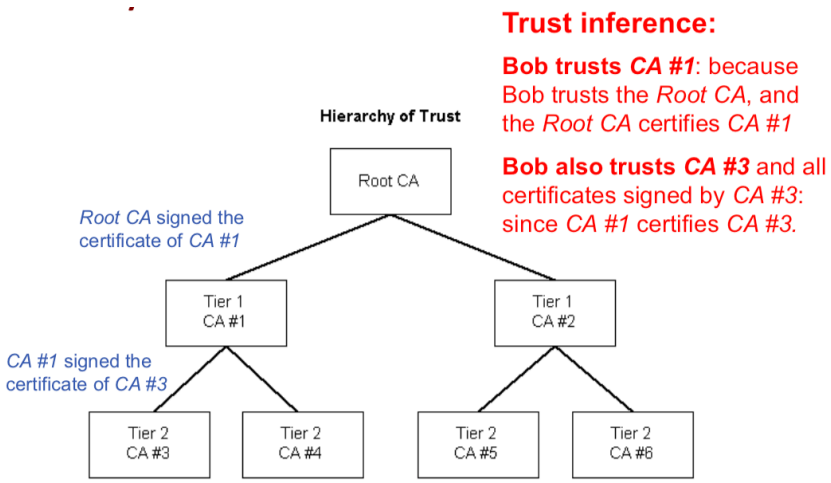
TLS Certificate Level Summaries

Certificate type	HTTPS encrypted?	Padlock displayed?	Domain validated?	Address validated?	Identity validation	Green address bar?
DV	Yes	Yes	Yes	No	None	No
OV	Yes	Yes	Yes	Yes	Good	No
EV	Yes	Yes	Yes	Yes	Strong	Yes

Types of Certificate Authority

There are three different types of CA:

- Root CA
    - Self-signed Certificate
- Subordinate / Intermediate CA
    - Tier 1, 2 etc.
- Leaf CA



There is a hierarchy of trust when it comes to the CAs. The Root CA signs the certificates of Tier 1 CAs, and the Tier 1 CAs sign the certificates of Tier 2 CAs. Suppose Alice's certificate is issued and signed by a Tier 1 CA, CA_1 and she sends a message, signed, with her certificate to Bob. But Bob does not have the public key of CA_1. What happens?

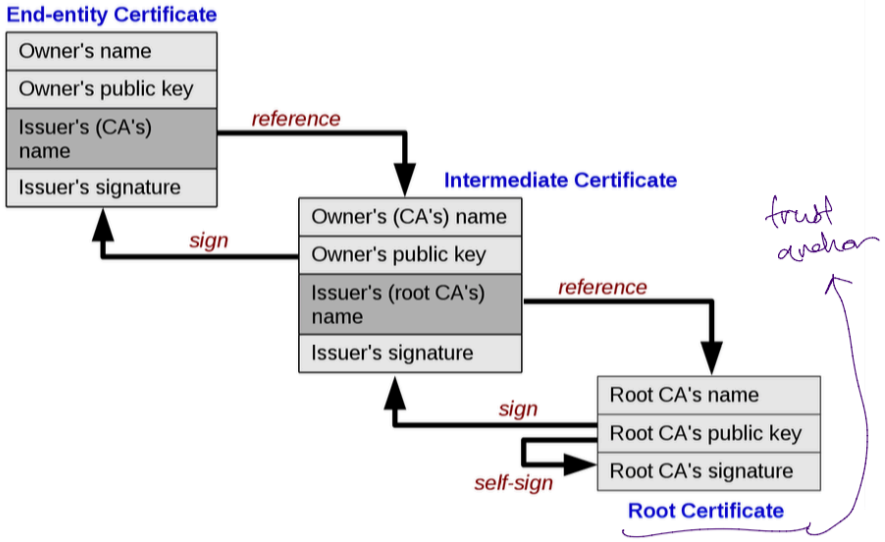
This is the **certification chain or path verification.** Alice will actually send CA_1's certificate along with her own (along with the Root CA's certificate as well). Bob will then verify CA_1's certificate using the Root CA's public key, Alice's certificate with the verified CA_1's public key, then verify Alice's message using Alice's verified public key.

Certification Chain
This is a list of certificates starting with an end-entity certificate followed by one or more CA certificates (with the last one being a self-signed root certificate).

For each certificate (except the last one):
- The issuer matches the subject of the next certificate in the list
- It is signed by the secret key of the next certificate in the list

The last certificate in the list, i.e. the root CA's, is the trust anchor**.**

There must also be a note in the certificate issued that states that the certificate owner is able to issue certificates to others, before non-root CAs are able to issue certificates.



Certificate Revocation
When certificates expire, they will naturally be revoked. But non-expired certificates can also be revoked for various reasons:
- Private key was compromised
- Issuing CA was compromised
- Entity left an organisation
- Business entity closed

Thus, a verifier needs to check if a certificate is still valid, even if it is not expired yet. There are different approaches to certificate revocation:
- Certificate Revocation List (CRL)
  - CA periodically signs and publishes a revocation list
- Online Certificate Status Protocol (OCSP)
  - OCSP Responder validates a certificate in question

An online CRL Distribution Point or OCSP Responder is thus needed.

However, there are issues with OCSP:

- Privacy
    - The OCSP Responder knows which certificates that you are validating
- Soft-fail Validation
    - Some browsers proceed in the event of no reply to an OCSP request (no reply *is* a “good” reply)

Solutions

- OCSP Stapling
    - Allows a certificate to be accompanied or “stapled” by a (time-stamped) OCSP response signed by CA
- Part of TLS handshake
    - Clients do not need to contact CA or OCSP Responder
- Drawback of the above
    - Increased network cost

Limitations and Attacks

There are numerous security issues when it comes to certificate authorities:

1. Compromise of CAs

As the CAs ultimately rely on the Root CA, if the Root CA is compromised, a lot of issues will occur.

Examples:

DigiNotar (Netherlands)

- Resulted in 500+ fraudulent certificates, including for *.google.com, *.mozilla.com, *.windowsupdate.com, *.torproject.org, in Sept 2011
- Immediately removed by major browsers
- Declared bankrupt within the same month

Turktrust (Turkey)

- Its subordinate CA, *.EGO.GOV.TR, issued *.gmail.com certificates
- Fraudulent certificates were used against Google Web properties

2. Abuse by CAs

There are so many CAs; some of them may be malicious. A rogue CA can practically forge any certificate. For example, Trustwave issued a "subordinate root certificate", which can then issue other certificates, one of which went to an organization that monitored the network. With this certificate, the organization can "spoof" X.509 certificates, and hence is able to act as the man-in-the-middle of any SSL/TLS connection.

3. MITM Attack by Rogue CA

A rogue CA can perform a man-in-the-middle attack by doing proxy re-encryption. The requirement is that the rogue CA's self-signed certificate is already accepted by the victim. These are the steps that will happen:

a) First, Mallory intercepts Alice's ClientHello to Bob.
b) Mallory pretends to be Bob. As a CA, Mallory is able to issue a certificate with Bob's details but with Mallory's own public key, and the digital signature is signed with Mallory's private key.
c) Alice will accept the information listed in the certificate issued by Mallory, because Alice has already accepted Mallory's certificate.
d) Alice and Mallory will establish communication, coming up with session keys k_1 to encrypt communications and t_1 for authentication e.g. MAC.
e) Mallory then sends ClientHello to Bob. Bob will send over his certificate, which Mallory will accept, thus coming up with session keys k_2 and t_2.
f) Mallory now can see all traffic and also modify all data.
    a. Alice – k_{e1} – k_{d1} – Mallory – k_{e2} – k_{d2} → Bob
    b. Bob – k_{e2} – k_{d2} – Mallory – k_{e1} – k_{d1} → Alice

4. Weak Browser Trust Model

Certification is as strong as the weakest root CA. The Browser Trust Model is a pre-loaded list of widely-used root CAs compiled by web browser developers. This is a form of Certificate Trust List (CTL) approach, where a list of CAs' certificates are compiled by a "trusted" authority.

The trust anchor in this situation would be the **union** of all root CAs. But there is always the question of which root CA should we use?

Implementation Bugs

There are also limitations in implementations that lead to severe vulnerability. These are not so much related with Certificates and CAs, but is related to web security. Below are some examples:

Due to different ways browsers and certificates interpret strings

1. Null-byte Injection Attack

Some browsers ignore the substrings in the entity's identity/name field after null characters when displaying it in the address bar, but include them when verifying the certificate.

For example, www.comp.nus.edu.sg\0.hacker.com will be displayed as www.comp.nus.edu.sg.

2. Social Engineering

A social engineering attack is typosquatting. Basically, hackers will register for domain names that look visually similar to an actual URL e.g. luminus.nvs.edu.sg, get a valid certificate for that domain, then employ a phishing attack to trick a victim to click on the above link. The victim does not notice the incorrect URL and ends up giving their credentials to the site.

Strong Authentication

As mentioned before, the password is a weak authentication system as an eavesdropper can easily replay the transmission to get authenticated. There is another way for the user to prove their identity i.e. she knows the secret, without revealing the secret.

Secret-Key-Cryptography-Based (SKC-Based) Challenge-Response

Suppose Alice and Bob have a shared secret k, and they agree on an encryption scheme e.g. AES.

The following steps will happen:

1. Alice sends to Bob a hello message
    a. "Hi, I'm Alice"
2. (Challenge) Bob randomly picks a plaintext m.
    a. Bob computes $y = E_k(m)$
    b. and sends y to Alice
3. (Response) Alice decrypts y to get m and then sends m to Bob.
4. Bob verifies that the message received is indeed m. If not, any further communication is rejected.

or can use mac, does not have to be encryption

Even if the eavesdropper Eve can obtain all communication between Alice and Bob, Eve still cannot get the secret key k. Replay is also impossible as the challenge m is randomly chosen and will likely be different in the next authentication session. The m ensures the freshness of the authentication process.

The protocol only authenticates Alice; it is called a unilateral authentication. There are also protocols to verify both parties, which are called mutual authentication.

Public-Key-Cryptography-Based (PKC-Based) Challenge-Response

Suppose Alice wishes to authenticate Bob.

The following steps will happen:

1. (Challenge) Alice chooses a random number r and sends to Bob
    a. "Bob, here is your challenge, r"
2. (Response) Bob uses his private key to sign r
    a. Bob also attaches his certificate sign(r)
3. Alice verifies Bob's certificate and extracts Bob's public key from the certificate before verifying that the signature is correct

An eavesdropper can't derive Bob's private key and replay the response because the challenge is likely to be different. The value r is also known as the cryptographic nonce (or simply nonce).

By security property of signature, the eavesdropper cant derive Bob's private key and reply response

Nonce purpose: to achieve freshness & prevent replay attacks

Session Keys

Strong authentication by itself is unable to ensure that Mallory cannot interrupt the session. This is because Mallory can still interrupt and take over the channel after strong authentication is carried out i.e. Mallory can pretend to be Alice.

We thus need something more. The outcome of the authentication process is a new secret key i.e. session key established between Alice and Bob. This process is called **key exchange** or **key agreement.**

Authenticated Key Exchange

If the process is incorporated with authentication, then this process is called authenticated key exchange or station-to-station protocol (if the key-agreement used is Diffie-Hellman). To carry out the protocol, Alice and Bob needs to know each other's public key, such as using PKI. After the protocol is completed, a set of session keys would be established: e.g. 1 for encryption and 1 for MAC, 1 key for each direction of encryption etc.

Tutorial Learnings

What is a self-signed certificate? Who typically uses one?
A “self-signed certificate” is a certificate that is signed by the stated entity’s private key.

It is used by a root CA. It is also quite commonly used by developers during the early stage of software development period when a valid certificate of a relevant host is not available yet.

Is it feasible to have browsers forward to a monitoring gateway all session keys they share?
Highly difficult to implement, since it requires a browser change. This requirement poses a serious problem since the school needs to make the required modifications on various types of popular Web browsers on different OS platforms, and their numerous available versions. Also, browsers do get updated very frequently by their developers, including for security reasons. The solution is therefore not feasible.

Putting It All Together

Securing a Communication Channel

The definition of a secure channel is one that establishes, between two programs, a data channel that has confidentiality, integrity and authenticity against a computationally-bounded network attacker.

This will involve the use of all concepts learnt so far. Let us see how we can secure a communication channel between Alice and Bob.com.

Step 1: Unilateral Authenticated Key Exchange

Alice and Bob.com will carry out unilateral authenticated key exchange using Bob's private and public key.

After authentication, both Bob and Alice know two randomly selected session keys (k, t), where k is the secret key of a symmetric-key encryption e.g. AES, and t is the secret key of a MAC.

Step 2: Authenticated Encryption

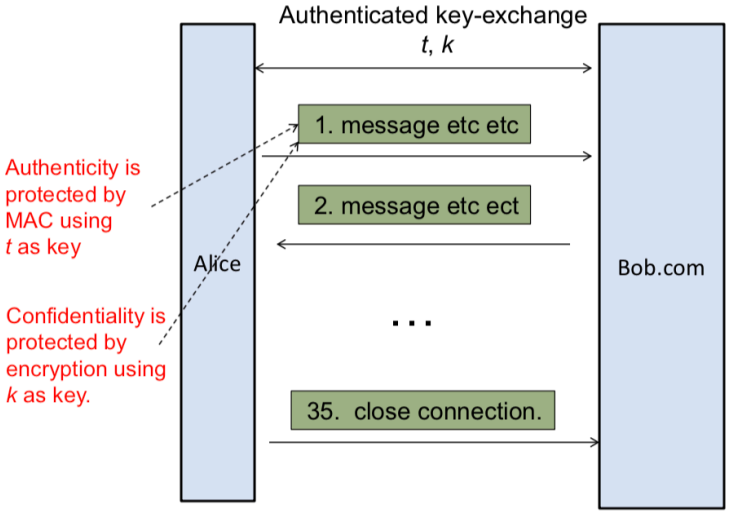
Subsequent communication between Alice and Bob.com will be protected by k, t and a sequence number i.

Suppose m_1, m_2, m_3, … are the sequence of messages exchanged, the actual data to be sent for m_i will be:

$$E_k (i \,||\, m_i) \,||\, \text{MAC} (E_k (i \,||\, m_i))$$

where i is the sequence number and || refers to concatenation.

The above is known as "encrypt-then-MAC", while there are other variants of authenticated encryption such as "MAC-then-encrypt" and "MAC-and-encrypt".



There is a need for the sequence number as it helps both sides to detect duplicates, dropped records and out-of-order records.

Use of PKI

For the above, PKI is often employed to distribute the public key. The authenticated key exchange is thus likely to involve certificates. After all, Alice needs to verify that the entity she is communicating with is indeed Bob.com.

HTTPS

HTTPS (Hypertext Transfer Protocol Secure) is widely used to secure Web traffic. It is built on top of SSL (Secure Sockets Layer) / TLS (Transfer Layer Security), i.e. HTTPS = HTTP + SSL. Hence HTTPS is also called HTTP over SSL or HTTP over TLS.

SSL / TLS

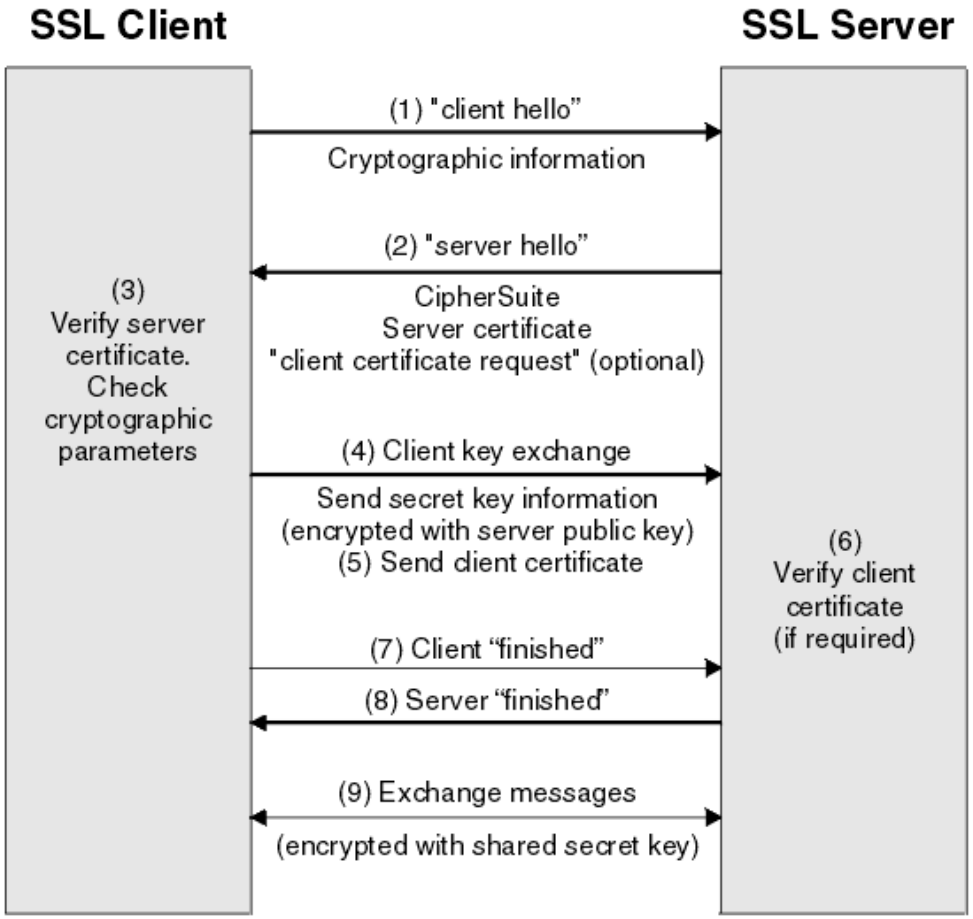
Transport Layer Security (TLS) is a protocol to secure communication using cryptographic means. SSL is the predecessor of TLS: Netscape SSL 2.0. They adopt a similar framework to secure a communication channel, and works in a similar manner to what was described above.

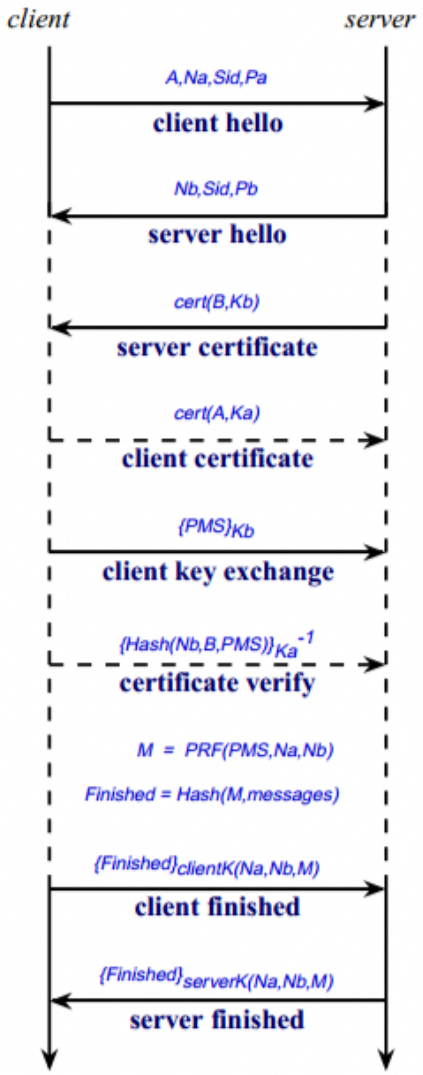
HTTPS

Overview of how HTTPS works:

- Ciphers Negotiation
- Authenticated Key Exchange (AKE)
    - Exchange of session key, which also authenticates the identities of parties involved
- Symmetric-key based secure communication
- Re-negotiation (if necessary)

TLS Handshake (Ciphers Negotiation & Authenticated Key Exchange)





To summarise, these are the steps

1. User asks for certificate — handshake
2. Server / Site sends the certificate — handshake
3. User verifies certificate and gets the public key Ke
4. User will generate a session key pair (k, t) - k is for encryption, t is for authentication
5. User will encrypt his session key pair using public key : E(Ke, (k, t)) and send to Server

This session key pair will expire after a while, and renegotiation is needed. What happens is that steps 1-3 will be skipped, since we already have the certificate and the public key. We just need to redo steps 4 and 5.

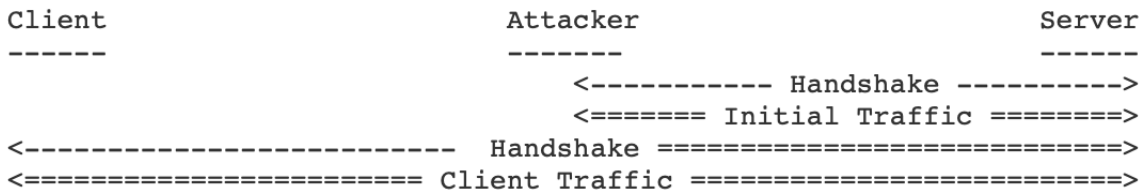
Protocol

In computer networking, a protocol is a set of rules for exchanging information between multiple entities. A protocol is often described as steps of actions to be carried out by the entities, and the data to be transmitted.

TLS Renegotiation Attack

We must first understand the TLS renegotiation feature. This feature allows a client and server who already have a TLS connection to negotiate new parameters, generate new keys, and more. This renegotiation is carried out **in the existing TLS connection**.

This renegotiation feature thus creates a problem – the encrypted connection before renegotiation and that after renegotiation can actually be controlled by different parties! To make things worse, web servers will combine the data they receive prior to renegotiation (which is coming from an attacker) with the data they receive after renegotiation (which is coming from a victim).



How does this work?

1. To mount this attack, the attacker first connects to the TLS server. A first handshake is done, and session keys are established. He can communicate with the server as much as he wants.
2. When he's ready, he hijacks the client's connection with the server, and when the client tries to establish their connection with the server, the attacker will intercept and proxy the client's traffic over his encrypted channel.
    a. This client hello is in clear, since no public key has been obtained yet.
    b. The attacker then encrypts this client hello using his session keys.
3. To the server, this seems like a renegotiation request, and the session keys generated by the client but redirected to the server by the attacker would seem like they were from the attacker. The server has no way to determine whose keys these are.
4. The attacker gets the acknowledgement and replays it to the client. The client will thus think that everything is okay.
5. The result is that the attacker's initial traffic and the client's subsequent traffic would be combined, with the web server thinking that the initial traffic was also from the client, or that the subsequent traffic is also from the attacker. Usually it is the former that would be more beneficial.

Possible Applications

This can be exploited in a way that is somewhat similar to an injection attack. The attacker can send a partial HTTP request that requests for some resource, but requires certain cookies, which are secret tokens that are sent with any request and are usually used as proof of identity.

Afterwards, the "renegotiation" is handled by the client, and the client's real request is concatenated to the end of the attacker's request, and the client's cookies are used instead.

For example, the attacker may send:

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

And leave the last line empty without a carriage return line feed. When the client makes his own request:

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

The two requests are merged into:

```
GET /pizza?toppings=pepperoni;address=attackersaddress HTTP/1.1
X-Ignore-This: GET /pizza?toppings=sausage;address=victimssaddress HTTP/1.1
Cookie: victimscookie
```

The server then uses the victim's account to send a pizza to the attacker.

This is another example:

```
GET /path/to/resource.jsp HTTP/1.0
Dummy-Header: GET /index.jsp HTTP/1.0
Cookie: sessionCookie=Token
```

where the red highlighted part is by the attacker, and the green is by the victim.

Notes on the attack

The attacker does not know the session keys that the client has established with the server, nor does he get to see any sensitive information that the client sends directly. It's all encrypted. However, he can exploit side effects of the exchange, such as getting a pizza out of it.

In other words, the renegotiation attack on TLS does not compromise confidentiality. However, it may breach the integrity of the client-server communication.

The second handshake has its key partially encrypted. It contains the session keys encrypted by the client using the server's public key, which is further encrypted by the attacker's current session key pair.

Some applications actually prompt for renegotiation when certain orders are being performed. Those are also possible for attackers to exploit.

Naïve Solutions that Do Not Work

1. Change the handshake message **from client to server**:
    a. Original: "Hello, I want to connect"
    b. New: "Hello, and I want to connect and this is the x handshake"
        i. Where x can be first, second, etc.
    c. Does not work as the attacker can simply change the first to second. This is because the original client hello is in clear to the attacker. It is unsure if the attacker would forward this as well, since technically this step can be skipped for the attacker.
2. Change the handshake message **from server to client**:
    a. Original: "Ok, I am happy to connect. Here is my certificate and other information."
    b. New: "Ok, I am happy to connect. Here is my certificate and other information. This is our x handshake."
        i. Where x can be first, second etc.
    c. Does not work as the attacker can change second to first easily. Unsure if the attacker would forward the initial client hello:
        i. If he forwards it, the reply would be encrypted, and the attacker can easily decrypt it and change x to the correct value before sending it back in clear to the client.
        ii. If he does not forward it, he can just reply the client with a standard response with x being first.

Solutions from Transport Layer that May Work

1. Server can sign the message “Ok, I am happy to connect… This is our x handshake.” using its private key, thus preventing the attacker from modifying it.
2. It can apply MAC using a secret key previously established with the client.

Solutions from Application Layer that May Work

1. Send two GET commands instead of one, where the first GET command is just a dummy operation.
2. The GET command includes (part of) the cookie for the server’s verification.
3. The GET command includes a value that is derived from the cookie, e.g. MAC of the GET command and the secret cookie, for the server’s verification.

However, any remedial solution from the web developer would be messy, as even it if works, the application itself now has to take care of communication security. This is not desirable since it is against the modularity principle of a good system design. That is, communication security is supposed to be handled by the layer below the application layer.

Can’t we just disable renegotiation?
It will affect availability since some applications may need the renegotiation feature of the TLS protocol.

Authentication in Applications
Most web applications perform an initial client authentication via a client’s username / password pair, and then subsequently persist that authentication state with HTTP / eb cookies (i.e. secret server-generated tokens that are automatically sent with subsequent client’s requests to the server).

More will be covered in subsequent topics.

Security Reduction:

There exists an attack on S the breaks A => There exists an attack on S that breaks B
There does not exists an attack on S that breaks A <= There does not exists an attack on S that breaks B
The system S achieves requirement S <= The system S achieves requirement B

For any hash function H
There exists an attack that invert H => There exists an attack that finds collision of H
There does not exists an efficient way to invert <= There does not exists an efficient way to find collision
H is one-way < = H is collision resistant

Network Security

Network Layers

Open Systems Interconnection (OSI) Model
The Open Systems Interconnection (OSI) model is a conceptual/reference model that standardizes the communication functions of a telecommunication or computing system.

7 – Application
6 – Presentation
5 – Session
4 – Transport
3 – Network
2 – Data Link
1 – Physical

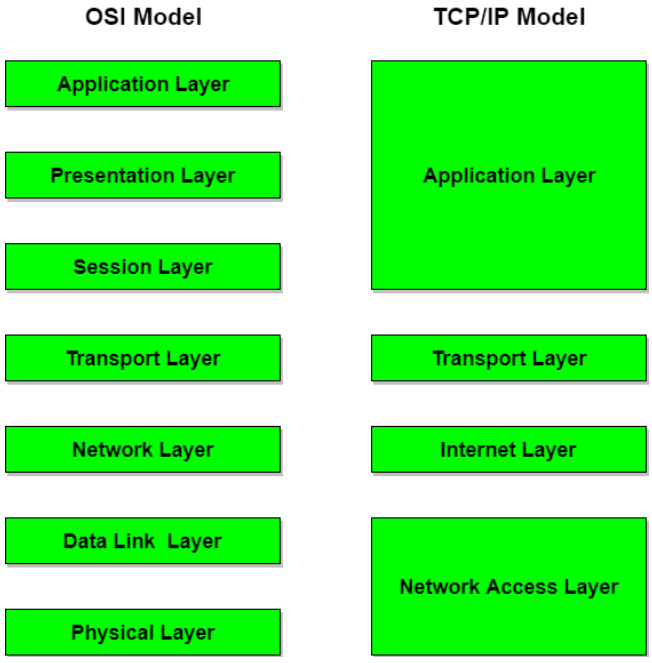
Transmission Control Protocol / Internet Protocol (TCP/IP) Reference Model

Application (e.g. HTTP)
Transport (e.g. TCP, UDP)
Layer 3 – IP / Internet Protocol
Layer 2 – Data Link
Layer 1 - Physical

UDP is unreliable

TCP/IP is reliable because needs to verify is the destination has received the message in the correct order

where Layers 2 and 1 are the Network Access/Interface. In some models, these two layers are summarized into a single Network Interface layer.



Reliability does not imply security = Malicious intermediate node can:
- spoof an IP packet to inform one node to close the connection, while still communicating with the other node
- change information on the packets ordering, so that destination reconstruct a scrambled message

OSI vs TCP/IP

So why are there two models when it comes to Network Layers? There are actually differences between them.

OSI (Open System Interconnection)	TCP/IP (Transmission Control Protocol / Internet Protocol)
OSI is a generic, protocol independent standard, acting as a communication gateway between the network and end user.	TCP/IP model is based on standard protocols around which the Internet has developed. It is a communication protocol, which allows connection of hosts over a network.
In OSI model the transport layer guarantees the delivery of packets.	In TCP/IP model the transport layer does not guarantees delivery of packets. Still the TCP/IP model is more reliable.
Follows vertical approach.	Follows horizontal approach.
OSI model has a separate Presentation layer and Session layer.	TCP/IP does not have a separate Presentation layer or Session layer.
Transport Layer is Connection Oriented.	Transport Layer is both Connection Oriented and Connection less.
Network Layer is both Connection Oriented and Connection less.	Network Layer is Connection less.
OSI is a reference model around which the networks are built. Generally it is used as a guidance tool.	TCP/IP model is, in a way implementation of the OSI model.
Network layer of OSI model provides both connection oriented and connectionless service.	The Network layer in TCP/IP model provides connectionless service.
OSI model has a problem of fitting the protocols into the model.	TCP/IP model does not fit any protocol
Protocols are hidden in OSI model and are easily replaced as the technology changes.	In TCP/IP replacing protocol is not easy.
OSI model defines services, interfaces and protocols very clearly and makes clear distinction between them. It is protocol independent.	In TCP/IP, services, interfaces and protocols are not clearly separated. It is also protocol dependent.
It has 7 layers	It has 4 layers

But in general, the differences do not really matter for this module.

Why do we use network layering?

It partitions a complex communication system into several abstraction layers. For example, we can view the layer-N protocol to be built upon a virtual connection at layer N-1. In other words, it just passes down the message, and let the level below do its job.

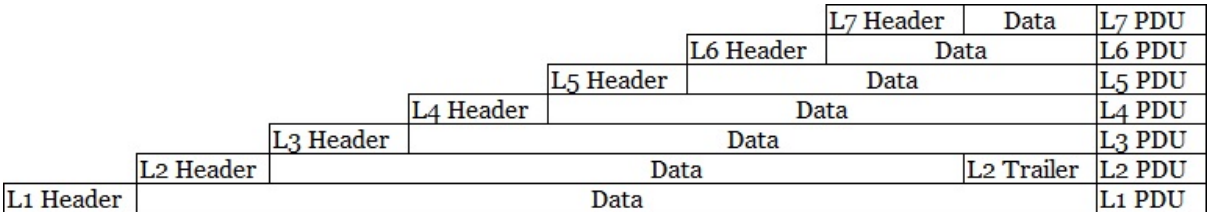
This is the concept called encapsulation in networking.

Encapsulation and Protocol Data Unit (PDU)
In networking, encapsulation is the method of designing **modular** communication protocols in which logically separate functions in the network are abstracted from their underlying structures via inclusion or information hiding within higher level objects, i.e. the lower layers are unable to discern which part of the data came from which higher level layer.

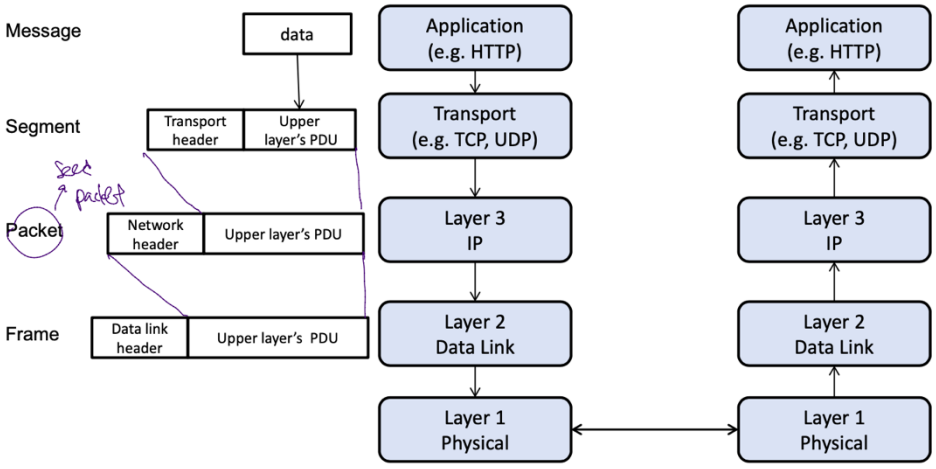
The roles and responsibilities are clear:
- Physical Layer: Physical transmission of the data
- Link Encapsulation Layer: Local area networking
- Internet Protocol Layer: Global addressing of local computers
- Transmission Control Protocol (Transport) Layer: Select the process or application i.e. the port which specifies the service such as a Web or TFTP server

During encapsulation, each layer builds a protocol data unit (PDU) by adding a header (and sometimes trailer) containing control information to the PDU from the layer above. For example, in the Internet Protocol suite, the contents of a web page are encapsulated with an HTTP header, then by a TCP header, an IP header, and, finally, by a frame header and trailer. The frame is forwarded to the destination node as a stream of bits, where it is decapsulated (or de-encapsulated) into the respective PDUs and interpreted at each layer by the receiving node.

The result of encapsulation is that each lower layer provides a service to the layer or layers above it, while at the same time each layer communicates with its **corresponding layer on the receiving node.** These are known as adjacent-layer interaction and same-layer interaction, respectively.



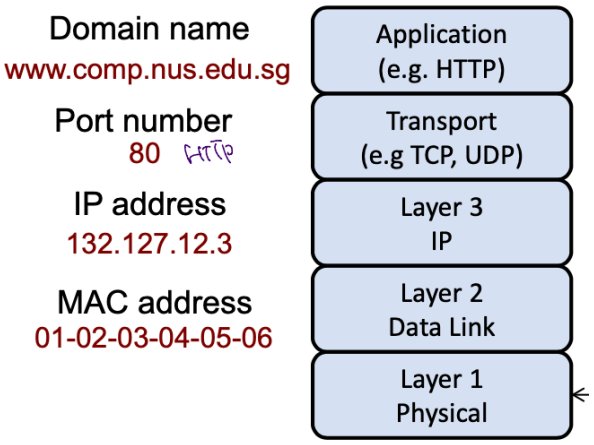
*Graphical representation of the PDUs in the **OSI model***



*Graphical representation of the PDUs in the **IP/TCP model***

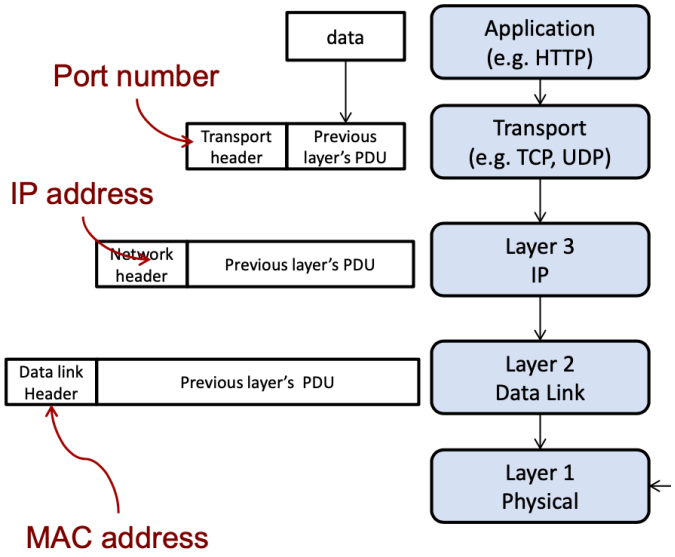
Addressing Schemes

As mentioned above, each layer has its own function in addressing. Refer to the diagrams below:



Different Addressing Schemes at Different Layers

Note: MAC here stands for Medium Access Control, not the same MAC in cryptography



Different Addressing Schemes in Headers

Hops

A hop happens when a packet is passed from one network segment to the next, usually through routers, from the source to the destination. The hop count refers to the number of intermediate devices through which data must pass between source and destination.

On a layer 3 network such as Internet Protocol (IP), each router along the data path constitutes a hop. By itself, this metric is, however, not useful for determining the optimum network path, as it does not take into consideration the speed, load, reliability, or latency of any particular hop, but merely the total count. Nevertheless, some routing protocols, such as Routing Information Protocol (RIP), use hop count as their sole metric.

Each time a router receives a packet, it modifies the packet, decrementing the time to live (TTL). The router discards any packets received with a zero TTL value. This prevents packets from endlessly bouncing around the network in the event of routing errors. Routers

are capable of managing hop counts, but other types of network devices (e.g. Ethernet hubs and bridges) are not.

What are the differences between a hub, a switch and a router?

Hub

Hub is commonly used to connect segments of a LAN (Local Area Network). A hub contains multiple ports. When a packet arrives at one port, it is copied to the other ports so that all segments of the LAN can see all packets. Hub acts as a common connection point for devices in a network.

A hub is the least expensive, least intelligent, and least complicated of the three. Its job is very simple: anything that comes in one port is sent out to the others. That's it.

Switch

A switch operates at the data link layer (layer 2) and sometimes the network layer (layer 3) of the OSI (Open Systems Interconnection) Reference Model and therefore support any packet protocol. LANs that use switches to join segments are called switched LANs or, in the case of Ethernet networks, switched Ethernet LANs. In networks, the switch is the device that filters and forwards packets between LAN segments.

A switch is slightly smarter than a hub in that it learns where the sender of a message is, such that any subsequent messages destinated for that sender need only be sent to that single port.

Router

A router is connected to at least two networks, commonly two LANs or WANs (Wide Area Networks) or a LAN and its ISP's (Internet Service Provider's) network. The router is generally located at gateways, the places where two or more networks connect. Using headers and forwarding tables, router determines the best path to forward the packets. In addition, router uses protocols such as ICMP (Internet Control Message Protocol) to communicate with each other and configures the best route between any two hosts. In a word, router forwards data packets along with networks.

Consumer-grade routers perform (at minimum) two additional and important tasks: DHCP and NAT.

DHCP (Dynamic Host Configuration Protocol) is how dynamic IP addresses are assigned. When it first connects to the network, a device asks for an IP address to be assigned to it, and a DHCP server responds with an IP address assignment. A router connected to your ISP-provided internet connection will ask your ISP's server for an IP address; this will be your IP address on the internet. Your local computers, on the other hand, will ask the router for an IP address, and these addresses are local to your network.

NAT – Network Address Translation – is the way the router *translates* the IP addresses of packets that cross the internet/local network boundary. When computer "A" sends a packet, the IP address that it's "from" is that of computer "A" – 192.168.0.1, for example. When the router passes that on to the internet, it replaces the local IP address with the internet IP address assigned by the ISP – 1.2.3.4, for example. It also keeps track, so if there's a response the router knows to do the translation in reverse, replacing the internet IP address with the local IP address for machine "A", and then sending that response packet on to machine "A".

A side effect of NAT is that machines on the internet cannot initiate communications to local machines; they can only respond to communications initiated by them. This means that the router also acts as an effective firewall.

What can a router's web admin do?

A router's web admin can do a lot of things, given that the admin is well-versed in this area. The admin can do the following:

1. Set another computer to intercept all traffic
2. If insecure email is used, email account details can be intercepted in clear whenever the email software checks the mail automatically
    a. Further incoming email can be blocked
    b. Individual messages can be deleted
3. With details in email, any password resets can be intercepted and used to gain access to secure sites
4. Cookies can be stolen if they are not uniformly secure, allowing access into an account without the password
5. See the end points of encrypted web traffic, though not the content itself
6. Change DNS settings to point to a domain under their control, as your computer would use the router as a DNS server when looking up what IP address a certain domain corresponds to.
7. If the router is part of an entire network of routers, compromising one router is enough to take over all of them, as they trust each other and there are routing protocols that can be subverted.

Challenges in Network Security:

[Complexity]

- Nodes are owned by many different semi-trusted parties
- Intermediate node has access to both the header and the payload, and can modify (authenticity), read (confidentiality), drop (availability)

Attacker at a certain layer can access all information in and below the layer

[Security Requirements]

- Many different security requirements
- In particular, availability cannot be handled by crypto alone
- Other requirements: anonymity, accountability, routing integrity, etc

[Legacy & Security tradeoff]

- Initial design of many networking protocols did not consider intentional attacks
- For eg: source ip address in the IP header. Without additional protection mechanisms, a malicious sender can send packets with spoofed source ip address
- For better performance and usability, many services do not employ strong protection mechanisms (DNS, network printer)

[Management]

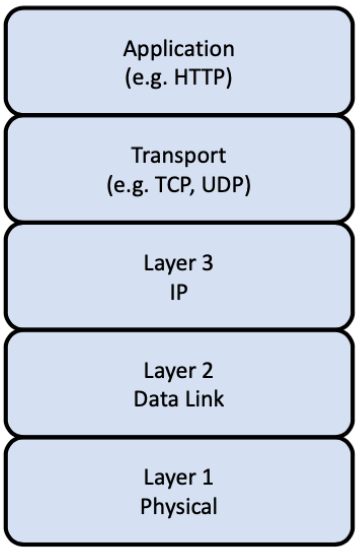
- There is a need to isolate and control information flow
- Ensuring that the information goes to the correct party, only, and does not go anywhere else

Network Attacks

Name Resolution and Attacks

Each peer entity (computer systems connected to each other via the internet) has a name. A single node may have different name at a different layer. For example:

www.comp.nus.edu.sg	Domain name
132.127.12.3	IP address
01-02-03-04-05-06	MAC address



When a peer entity uses the virtual connection in the layer below, it needs to find the corresponding name mapping. For example, finding the IP address of a domain name. Protocols that perform name mappings are known as resolution protocols.

Many initial design of resolution protocols did not take security into account, and thus it was easy for attackers to manipulate the outcome.

One such resolution protocol is the Domain Name System (DNS), which maps domain names to their respective IP addresses. It uses a hierarchical decentralized naming system. An attacker can thus target the association of the domain name with the IP address.

Another resolution protocol is the Address Resolution Protocol (ARP), which associates or maps IP addresses (logical addresses) with MAC addresses (physical addresses). It uses a broadcast mechanism on a local network. An attacker on the local network can target the association.

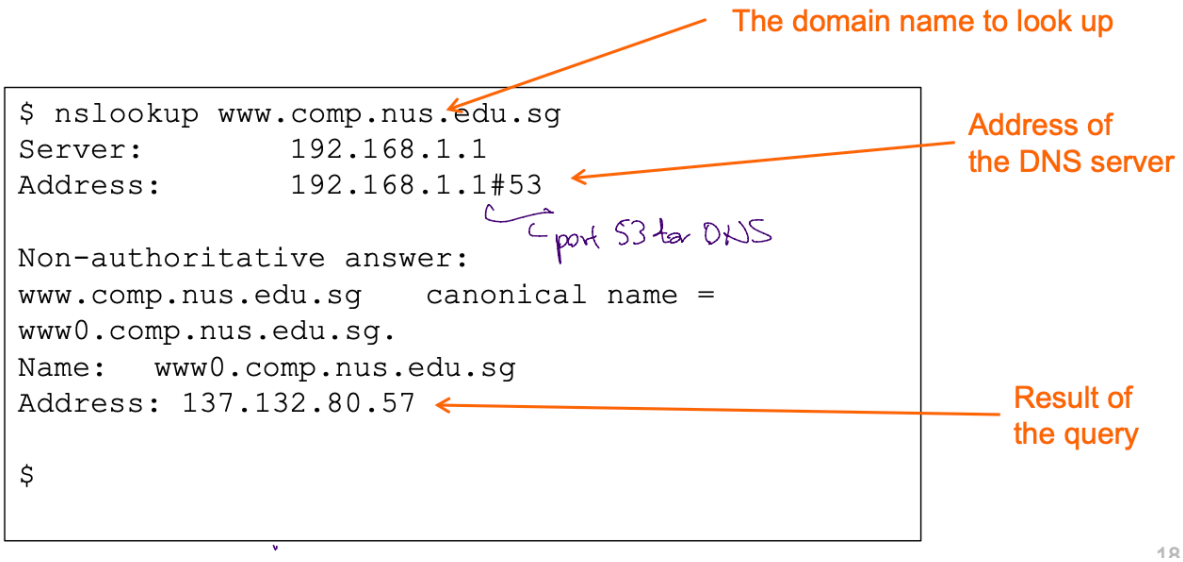
Domain Name System (DNS)

Given a domain name (e.g. www.comp.nus.edu.sg), its IP address can be found by either looking up a locally stored host table, or by querying a DNS server. The process is known as **name resolution**.

The entity (aka client) that initiates the query is called the resolver. If the address is found, we say that the domain name is resolved.

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

Note that no encryption or MAC is involved, as in the original design consideration, the QID is probably not meant for authentication, but as an efficient way to match multiple queries.



Local DNS Attack

Let us consider the case of Alice, who is at a café using an unprotected WIFI connection to surf the web. She visits www.comp.nus.edu.sg, and types the domain name into the browser's address bar.

The browser makes a query to a DNS server to determine the IP address, then connects to the IP address obtained.

However, if there is an attacker at the physical layer, i.e. in the café accessing the same WIFI, the attacker can sniff data from the unprotected communication channel and spoof data into it as well. The attacker cannot modify or remove data already sent by Alice.

Should the attacker own a web server with a spoofed SoC website, the attacker can spoof a reply with the same QID as Alice's query with the attacker's own IP address as the message. Since the attacker is closer to Alice, the attacker's reply will reach Alice first, before the reply from the DNS server does. Alice takes the first reply as the answer and connects to the fake IP address.

Timing attack = where the first reply with the correct QID will be taken as the "real" one, local attacker will be faster than the actual DNS reply

More about DNS

- DNS operates at the **application layer.**
- Although the attacker is at the physical layer, for ease of analysis, we can assume that the attacker is just below the application layer. That is, there exists some virtual connection that can send the message across.
- Hence, the previous portion doesn't mention about the MAC and IP addresses, etc., of the DNS server.
- The DNS is an important component as it resolves the domain name. Hence, an DNS server can be the "single-point-of-failure" for the network.
- A DoS attacks, instead of attacking a Web server, could attack the DNS server instead.

ARP poisoning = essentially just a MITM attack
- attacker will first try to poison the ARP table to get specific port to send to him, and then he will send out to the destination port
- All in layer 2

Denial of Service Attacks

DoS attacks target availability, preventing some service from being accessible and usable upon demand by an authorised entity, delaying time-critical operations.

Types of DoS Attacks

	Stopping Service	Exhausting Resources
Local Attack (Easily detected and punished/mitigated)	Process killing Process crashing System reconfiguring	Spawning processes Filling up the file system
Remote Attack (over the Internet)	Sending malformed packet attacks Requires vulnerabilities, which are easily patched up	Packet flooding, as the system cannot tell if the request is valid or invalid

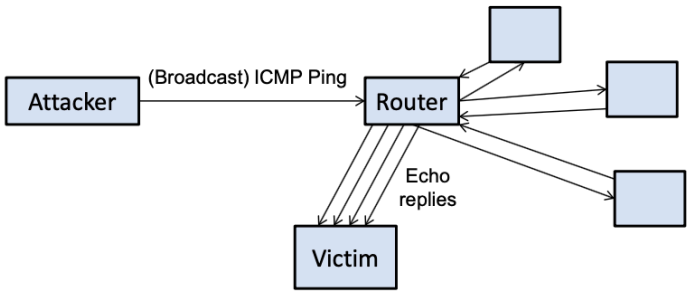
More on the above:

- Local Attacks
  - Can be more easily tracked than remote attacks
- Malformed Packet Attacks
  - Sending malformed packets remotely does not usually work on updated operating systems, since it requires vulnerabilities
- Packet Flooding Attacks
  - Many effective DOS attacks simply remotely flood the victims with an overwhelming amount of requests or data
  - The attacker can amplify small traffic to obtain a large amount of traffic, typically done by using available public servers (Internet infrastructure), such as DNS, NTP and CharGen.

DoS Example 1: ICMP/Smurf Flood Attack
Why "smurf"? This is because this attack can bring down big targets.

This is an attack that makes use of public servers to target a specific victim IP address. This is done via:

1. An attacker sends the **"ICMP PING"** request to a router, instructing the router to broadcast this request.
2. The request's source IP address is spoofed and replaced with the victim IP address.
3. The router thus broadcasts this request.
4. Every entity that receives this request will reply to it by sending an **"Echo reply"** to the source, which is the victim
5. The victim is overwhelmed with "Echo reply"s from the entire network.
6. The attacker thus takes advantage of the **amplification effect** to attack the victim.

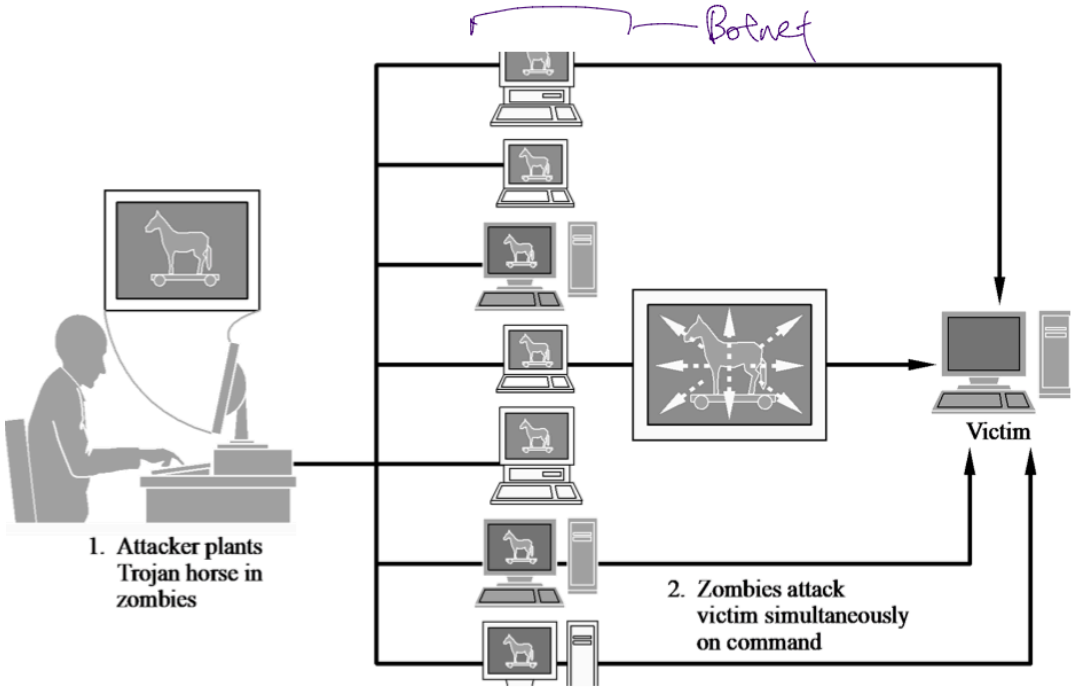


Is this attack still effective? Fortunately not, as most routers are now configured to not broadcast the requests. To prevent this attack, the measure is to simply disable a feature that was previously thought to be useful.

DoS Example 2: Application-Layer DoS Attack (HTTP Get)
The trick is to simply flood a web server with HTTP requests. For this attack to be effective, a large number of attackers are required, since each attacker can only send requests at a low rate.

When DoS is carried out by a large number of attackers, this is called **Distributed Denial of Service (DDoS).**

Distributed Denial of Service (DDoS)



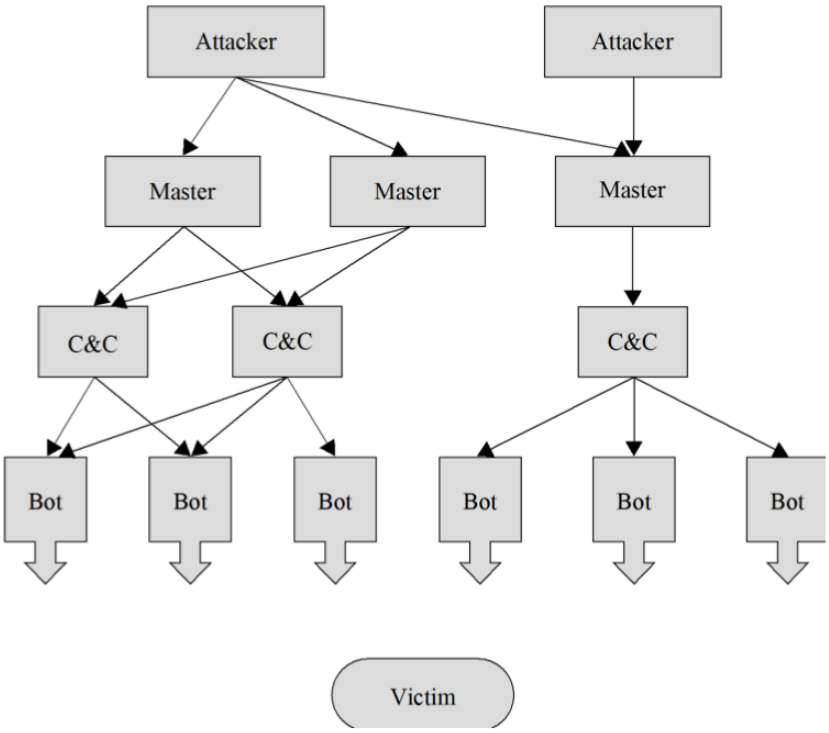
As shown above, DDoS is normally achieved via a botnet.

A bot, or zombie, is a compromised machine, and a botnet, or zombie army, is a large collection of connected bots, communicating via covert channels.

Why via covert channels? This is to prevent the owner of the zombie computer from noticing. Most owners of zombie computers are unaware that their system is being used in this way. Because the owner tends to be unaware, these computers are metaphorically compared to zombies.

A botnet has a **command-and-control** mechanism and can thus be controlled by a single individual to carry out a Distributed Denial of Service attack.

Some other possible usages of a botnet includes:
- Vulnerability Scanning
- Anonymising HTTP Proxy
- Email Address Harvesting
- Cipher Breaking



How a botnet may look like

Reflection and amplification attack

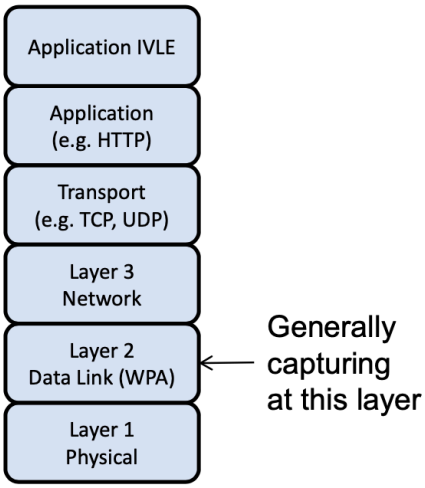
- Reflection attack is type of DOS where attackers send requests to intermediate nodes, which in turn send overwhelming traffic to victim
- Amplification attack, where a small request will lead to a huge reply, so can send many in a short time but will take the server much longer to send a reply back

Mitigation is to just increase the size - like in the cloud services, expand size to handle more incoming services

- or to see incoming packets and determine how to handle them based on determining purpose
  -- drop or put on lower priority

Useful Tools

Wireshark (Packet Analyser)
Wireshark is a popular, free and open-source network packet analyser. It generally performs capturing at the **link layer**. This depends on the operating system and hardware, however. It essentially captures "interactions" between the operating system and the network card driver.



Some things that Wireshark can do:
- View list of packets
- View packet details
- View packet bytes
- Filter for packets
- Follow TCP stream

Nmap (Port Scanner)
What is a port? Previously, we saw that the port number was assigned at the **Transport Layer** of the TCP/IP model.

A port helps a server decide which application process to handle an incoming packet. By saying that a process or service is "listening" to a particular port, we mean that the process is running and ready to process packets with that particular port number. We also say a port is "open" when there exists such a process running in the server.

Well-known port numbers:
- 1: TCP Port Service Multiplexer
- 7: Echo Protocol
- 17: Quote of the Day
- 19: Character Generator Protocol (CHARGEN)
- 20: File Transfer Protocol (FTP) data transfer
- 21: File Transfer Protocol (FTP) control
- 22: Secure Shell (SSH), secure logins, file transfers (scp, sftp) and port forwarding
- **25: Simple Mail Transfer Protocol (SMTP), used for email routing between mail servers**
- 43: WHOIS Protocol
- 53: Domain Name System (DNS)
- **80: Hypertext Transfer Protocol (HTTP)**
- 220: Internet Message Access Protocol (IMAP), version 3
- **443: Hypertext Transfer Protocol over TLS/SSL (HTTPS)**
- 465: Authenticated SMTP over TLS/SSL (SMTPS)
- **515: Line Printer Daemon (LDP), print service**
- 666: Doom, first online first-person shooter

Port scanning is thus the process of determining which ports are open on hosts in a network. Ports are somewhat like the "doors" into each machine, hence port scanning is somewhat like knocking on the doors.

A port scanner is thus a tool to perform port scanning. It is useful for both attackers and network administrators to scan for vulnerabilities. Nmap is a very popular port scanner.

Nmap is a full featured port-scanning tool:
- Command-line tool with a GUI frontend
- Installation: sudo apt-get install nmap, zenmap

- Usage: nmap [ScanType(s)] [Options] {target specification}
- Examples:
    - TCP ACK scan (a stealthier scan): nmap -sA
    - OS fingerprinting: nmap -O
    - Service/version detection: nmap -sV

Network Protection

Cryptography

There are several cryptographic techniques that can help us achieve **confidentiality** (via encryption) and **authenticity** (via MAC, PKI, and Strong Authentication) over a public communication channel, even if the adversary can sniff and spoof the data.

There are various security protocols that essentially achieve that, but operates at different layers. Some prominent protocols are:

- TLS/SSL
- WIFI Protected Access II (WPA2)
- Internet Protocol Security (IPsec)

Issues faced when we discuss security protocols and attacks:

- Often, when we discuss a security protocol, we indicate the layer that the protocol aims to protect.
    - Complication: Some protections span across multiple layers, or do not provide full protection of the targeted layer.
- When analysing an attack, it is also insightful to figure out at which layer the attacker resides.
    - Complication: Likewise, some attacks span across multiple layers. In such situations, trying hard to pinpoint the layer could sometimes be very confusing.

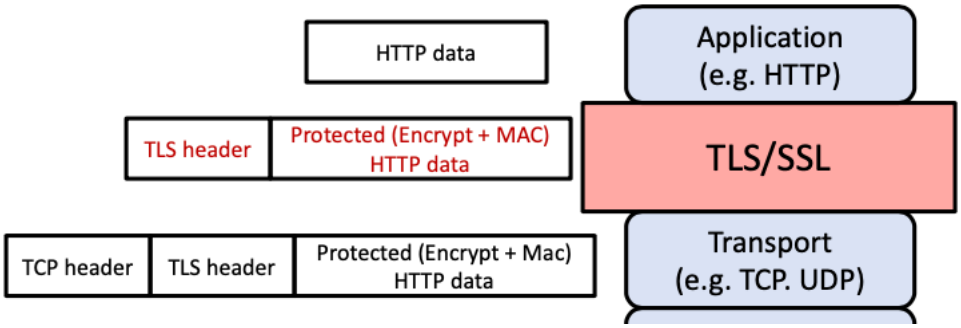
Thus, we have the following general guideline:

A security protocol that protects layer k would protect information from that layer and above against an attacker sitting at layer k-1 and below

For example, if an attacker resides at layer 1 and there is a security protocol that protects layer 3, what is protected by the security protocol is the information generated in layer 3 and above, but what is not protected is the information generated in layer 2.

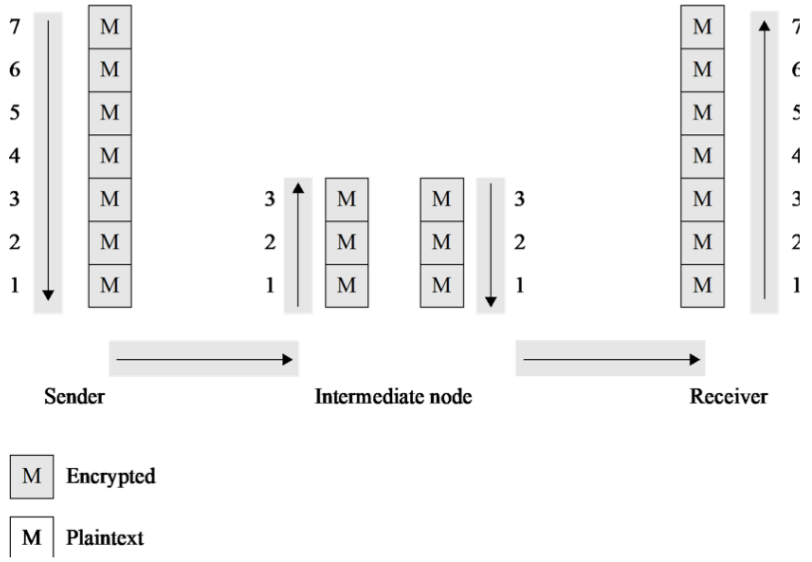
Secure Sockets Layer / Transport Layer Security (SSL/TLS)
The SSL/TLS sits **on top of the transport layer.** In other words, when an application, such as a browser or an email agent, wants to send data to the other end point, it first passes the data and the destination IP address to the SSL/TLS.

Next, SSL/TLS protects the data using encryption (for confidentiality) and MAC (for authenticity), then it instructs the transport layer to send the protected data. An end-to-end encryption is performed.

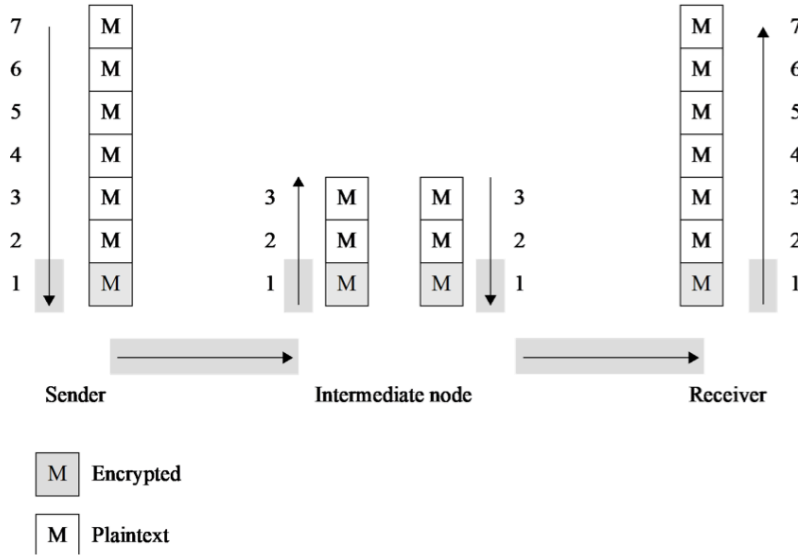


The receiver end-point decrypts the received data **at the corresponding layer**.

What is end-to-end encryption?
End-to-end encryption is called as such because data stays encrypted from one end of its journey to the other. In this case, it is actually encrypted once it leaves the application layer, and remains encrypted until it reaches above the transport layer of the receiver side. Below is a diagram to illustrate the point.



This is in contrast with Link Encryption, of Hop-by-Hop Encryption. In Link Encryption, the encryption occurs at the Data Link and Physical layers, and the packet is decrypted at every device between the two ends. Below is a diagram to illustrate the point.



Examples
Let us see some examples of how SSL/TLS works.

1. Alice accesses the LumiNUS web application to upload her report, a.pdf, to the LumiNUS server.
    a. Note that LumiNUS uses HTTPS, which in turn employs SSL/TLS.
2. Alice's machine does the following:
    a. The "LumiNUS client" passes the file a.pdf to HTTPS, which in turns passes it to TLS
    b. TLS protects the data by encryption and MAC
    c. TLS passes the protected data to the transport layer
3. The LumiNUS server carries out the following:
    a. The transport layer passes the protected layer to TLS
    b. TLS decrypts the data and verify the MAC for integrity
    c. TLS passes the decrypted data to the LumiNUS application.

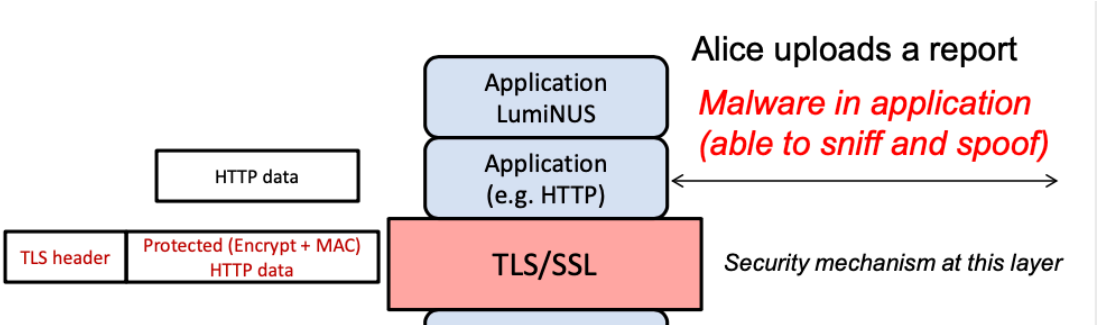
Note that in the process, "handshaking" occurs, where the two parties establish their session keys.

Attack Scenario 1: Attacker at the Physical Layer
Suppose there is an attacker at the physical layer who can sniff and spoof the message at that layer. Alice then uploads her report in that café using their free and open WIFI, that has no WPA protection. Hence, anyone in the café has access to the physical layer, and can sniff and spoof messages in that layer.

Can the attacker learn:
- Alice's uploaded report?
    - No, as it is protected by SSL/TLS, which protects the application layer, and information from that layer would be protected from an attacker below it, i.e. physical layer.
- The fact that Alice is visiting the LumiNUS website i.e. can the attacker lean the website's IP address?
    - Yes, as the information is contained in the IP headers from the network layer, which is not protected by TLS/SSL that sits on top of the transport layer from the attacker below it at the data link or physical layer.

Attack Scenario 2: Attacker at the Application Layer
Suppose that there is an adversary at the application layer. For example, a malicious JavaScript code is injected into LumiNUS and is being executed by Alice's computer. Can the malicious script learn:

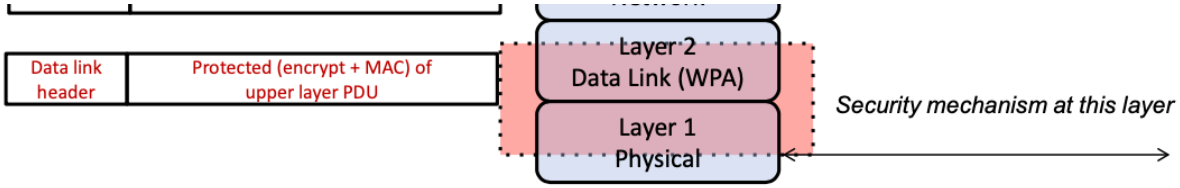
- Alice's report?
    - Yes, as the SSL/TLS does not protect information from its layer and above from attackers who are also on its layer or above. The malware in the application is still able to sniff and spoof.
- Alice's MAC address?
    - No, as the MAC address is defined at layer 2. An attacker at the application layer is unable to attack "downwards", and can only attack upwards.



WIFI Protected Access II (WPA2)

WPA2 is a popular protocol employed in home WIFI access points, and is more secure than Wired Equivalent Privacy (WEP), which is broken, and WPA.

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



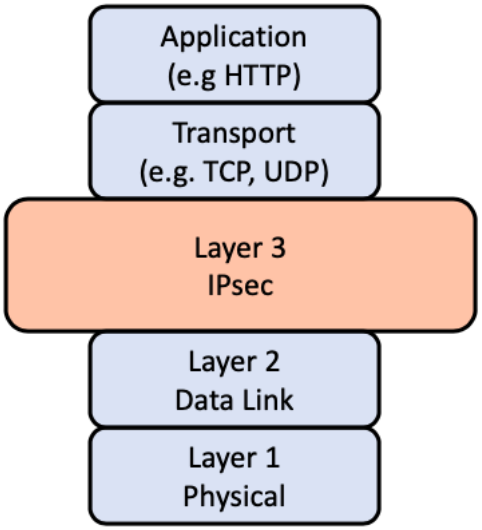
Attack Scenario: Attacker at the Physical Layer
Suppose there is an attacker at the physical layer who is able to sniff and spoof information, and Alice uploads a report. Can the attacker learn:

- Alice's report?
  - No, as data from the upper layers have been encrypted with AES.
- The fact that Alice is visiting LumiNUS website?
  - No
- The MAC address (which is assigned in the link layer)?
  - The MAC address is never encrypted, as the MAC itself is required to enable the packet to reach the router and enable the router to send packets back.

Anyone within the range of the network might be able to see the traffic, but it will be scrambled with the most up-to-date encryption standards. WPA2 uses AES and keys that are 64 hexadecimal digits long.

WPA3 was announced in January 2018.

Internet Protocol Security (IPsec)



IPsec provides integrity and authenticity protection of IP addresses, but not confidentiality. Hence, attackers are unable to “spoof” the source IP addresses, but they can still learn the source and destination IP addresses of sniffed packets.

It is a mechanism whose goal is to protect the IP layer. The following is the detailed description:

IPsec is a is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing **mutual authentication between agents at the beginning of the session** and **negotiation of cryptographic keys to be used during the session**. IPsec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway.

Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPsec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.

IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at Application layer. Hence, **only IPsec protects any application traffic over an IP network**. Applications can be automatically secured by IPsec at the IP layer.

Firewall

Having SSL/TLS and WPA2 is still insufficient when it comes to protecting the network. There are concerns with Denial of Service attacks, which they cannot prevent. SSL/TLS and WPA2 does not protect us when we interact with many services and applications, such as the DNS server. It is no practical, due to efficiency, to establish a SSL/TLS to the DNS server for a DNS query, hence we are susceptible to DNS spoofing.

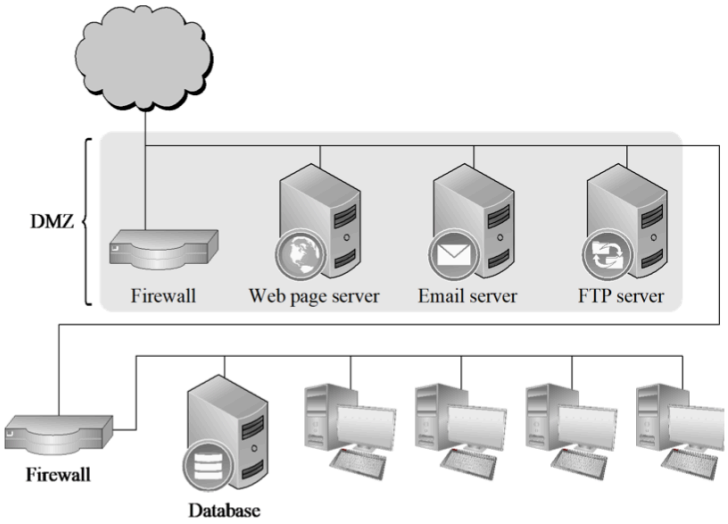
There is a need to control the flow of traffic between networks, especially between the untrusted public network (Internet) and the trusted internal network.

What is a Firewall?
A firewall is a device or program that controls the flow of network traffic between networks or hosts that employ **differing security postures.** It sits at the border between networks and looks at addresses, services and other characteristics of traffic. It then controls what traffic is allowed to enter the network (ingress filtering), or leave the network (egress filtering).

Demilitarized Zone (DMZ)
There is thus a concept of a DMZ – a small sub-network that exposes the organisation's external service to the (untrusted) Internet.

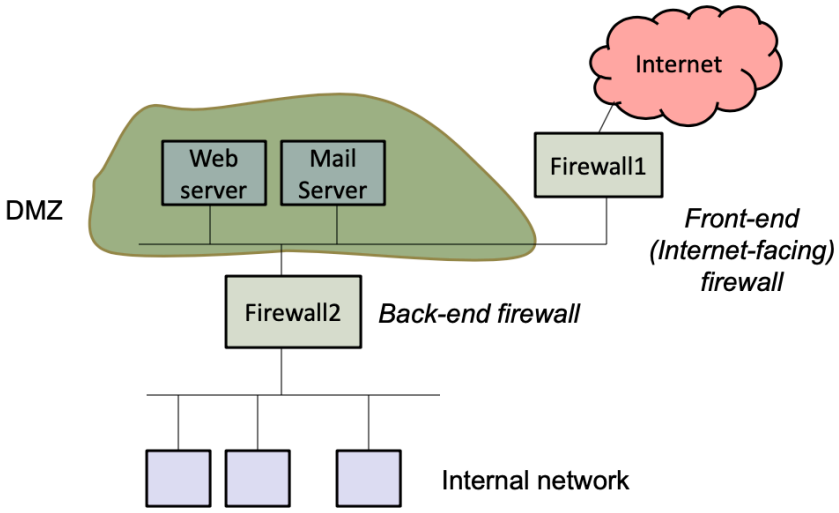
The purpose of a DMZ is to add an additional layer of security to an organization's local area network (LAN): an external network node can access only what is exposed in the DMZ, while the rest of the organization's network is firewalled. The DMZ functions as a small, isolated network positioned between the Internet and the private network and, if its design is effective, allows the organization extra time to detect and address breaches before they would further penetrate into the internal networks.

In this case, the hosts most vulnerable to attack are those that provide services to users outside of the local area network, such as e-mail, Web and Domain Name System (DNS) servers. Because of the increased potential of these hosts suffering an attack, they are placed into this specific subnetwork in order to protect the rest of the network should any of them become compromised.



As seen above, this is a dual firewall architecture or 2-firewall setting. It is the most secure approach, according to Corlton Fralick – to use two firewalls to create a DMZ. The first firewall (also called the "front-end" or "perimeter" firewall) must be configured to allow traffic

destined to the DMZ only. The second firewall (also called "back-end" or "internal" firewall) only allows traffic to the DMZ from the internal network.



This setup is considered more secure since two devices would need to be compromised. There is even more protection if the two firewalls are provided by two different vendors, because it makes it less likely that both devices suffer from the same security vulnerabilities.

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

An example of rules for Firewall-2 (back-end firewall) would be:
- Block HTTP
- Allow from Internal Network to Mail Server: SMTP, POP3

An example of rules for Firewall-1 (front-end firewall) would be:
- Allow from anywhere to Mail Server: SMTP only

How the rules are to be specified thus differs based on the devices and software. Here is a sample firewall configuration.

Rule No	Protocol Type	Source Address	Destination Address	Designation Port	Action
1	TCP	*	192.168.1.*	25	Permit
2	TCP	*	192.168.1.*	69	Permit
3	TCP	192.168.1.*	*	80	Permit
4	TCP	*	192.168.1.18	80	Permit
5	TCP	*	192.168.1.*	*	Deny
6	UDP	*	192.168.1.*	*	Deny

*(any) matches any value

The table is processed in a top-down manner, and the first matching rule determines the action taken. Hence, the most specific rule is on top, and the most general rule is last.

if payload is inspected, then its called deep packet inspection

Types of Firewall
There are 6 types of firewalls in the textbook, but they are usually grouped into 3 types:

1. **(Traditional) Packet Filters**

Filters packets based on information in packet headers.

2. **Stateful-Inspection (Packet Filters):**

Maintains a state table of all active connections, and filters packets based on active connection states.

3. **Application Proxy**

Understands application logic and acts as a relay of application-level traffic.

Below are optional information on the various types of firewalls:

Packet Filter	Stateful Inspection	Application Proxy	Circuit Gateway	Guard	Personal Firewall
Simplest decision-making rules, packet by packet	Correlates data across packets	Simulates effect of an application program	Joins two subnetworks	Implements any conditions that can be programmed	Similar to packet filter, but getting more complex
Sees only addresses and service protocol type	Can see addresses and data	Sees and analyzes full data portion of pack	Sees addresses and data	Sees and analyzes full content of data	Can see full data portion
Auditing limited because of speed limitations	Auditing possible	Auditing likely	Auditing likely	Auditing likely	Auditing likely
Screens based on connection rules	Screens based on information across multiple packets—in either headers or data	Screens based on behavior of application	Screens based on address	Screens based on interpretation of content	Typically, screens based on content of each packet individually, based on address or content
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex decision rules, but proxies must be aware of application's behavior	Relatively simple addressing rules; make configuration straightforward	Complex guard functionality; can be difficult to define and program accurately	Usually starts in mode to deny all inbound traffic; adds addresses and functions to trust as they arise

Intrusion Detection System (IDS)
- Sensors that gather data by:

1) Attack signature Detection = attack uses certain ports/ ip address
2) Anomaly Detection = sudden surge of packets with a certain port number
3) Behavior based IDS = System keeps a profile of users, then tries to detect if users deviates from profile (anomaly detection but for each user)

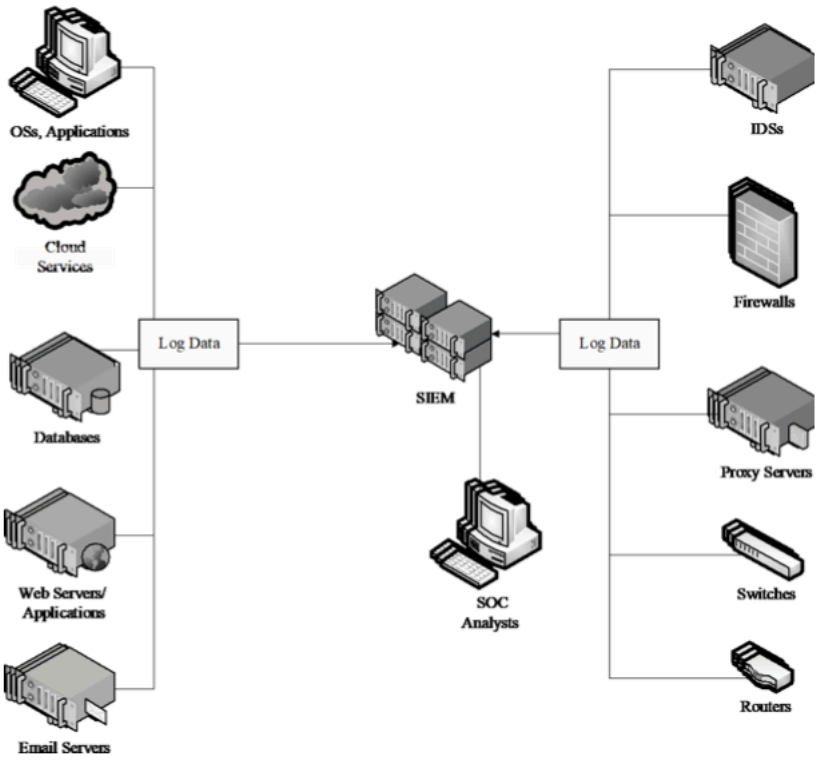
Network Security Management

There is a need to continuously monitor and adjust network characteristics. Hence, some best practices have been adopted to do so:

Security Operations Center (SOC)
A centralised unit in an organisation that monitors the IT systems and deals with security issues.

Security Information and Event Management (SIEM)
Pronounced as “SIM”, SIEM is an approach that aims to provide real-time analysis of security alerts generated by network hardware and network applications. This may include the following capabilities:

- Data aggregation and correlation
- Event alerting
- Compliance report generation
- Forensic analysis



Firewall Design Case Study (from Tutorial 6)
Assume that we have two firewalls and the following machines:

1. Lab
    a. Total of 100 machines in labs for students to:
        i. prepare for reports
        ii. search for materials on the web
    b. There are also network printers in the labs
2. Teachers
    a. Every teacher has a PC in the teacher room
    b. They use the PCs to:
        i. enter students' grades
        ii. send/receive emails
        iii. prepare teaching materials
        iv. print exam questions
        v. perform web searches
    c. There are also network printers in the teacher rooms
3. Web-server
    a. School's web server
4. Email-server
    a. School's SMTP email server
5. SQL-server
    a. Stores the student database
    b. Some information can be accessed via a web-based application hosted in the Web-server
    c. For example, the app can allow students to update their mobile phone numbers
    d. Some other information can be accessed only by the teachers

These are the more precise requirements:

1. Prevent cases where exam questions get mistakenly printed in the Lab
2. Protect the SQL server
3. Block outbound packets that do not have legitimate source IP addresses, as some students may be running hacking tools that generate spoofed source IP addresses
4. We ignore the detailed issue of routing, i.e. we do not consider the internet gateway and Network Address Translation (NAT). For simplicity, we just assume that all machines use "public" IP addresses.

What we can do is the following:

Internal ← (IN) F_2 (OUT) → DMZ ← (IN) F_1 (OUT) → Internet

With the following setup:

- DMZ
    - Web-server
    - Email-server
    - Lab
        - We place it here since Lab PCs do not contain any important data and we want to segregate Lab and Teachers as required
    - Lab-printers
- Internal
    - Teachers
    - Teacher-printers
    - SQL-server

We then configure firewalls F_1 and F_2 as such:

Source IP	Dest IP	Source Port	Dest Port	Direction	Action
Web-server	*	HTTP	*	OUT	Allow
*	Web-server	*	HTTP	IN	Allow
Email-server	*	SMTP	*	OUT	Allow
*	Email-server	*	SMTP	IN	Allow
Lab	*	*	HTTP	OUT	Allow
*	Lab	HTTP	*	IN	Allow
Teachers	*	*	HTTP	OUT	Allow
*	Teachers	HTTP	*	IN	Allow
*	*	*	*	*	Block

Table 1: Firewall rules for the front-end / outer firewall F_1

Source IP	Dest IP	Source Port	Dest Port	Direction	Action
SQL-server	Web-server	SQL	*	OUT	Allow
Web-server	SQL-server	*	SQL	IN	Allow
Teachers	*	*	HTTP	OUT	Allow
*	Teachers	HTTP	*	IN	Allow
Teachers	Email-server	*	SMTP	OUT	Allow
Email-server	Teachers	SMTP	*	IN	Allow
*	*	*	*	*	Block

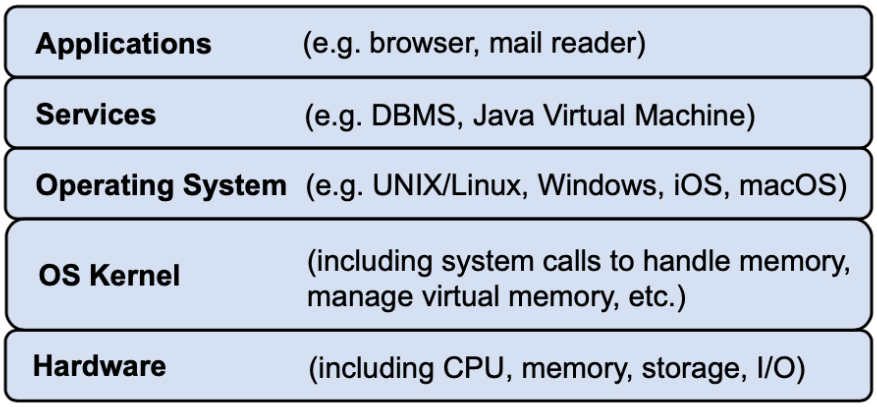
Table 2: Firewall rules for the back-end / inner firewall F_2

Note that requirement 3, which aims to block outgoing packets with illegitimate source IP addresses (egress filtering), is automatically met by the given rule sets. Any other firewall rules can be added as necessary, such as those needed to allow DNS and HTTPS traffic.

Type of Firewall: The firewalls above inspect only a few important fields of network packets. Thus they are packet filtering firewalls, which operate at the IP layer.

Access Control

Layering Model in Computer System Design
This is a model of the layers in a computer system:



We can actually view the OS Kernel as part of the OS. These layers are used as a guideline, and actual systems typically don't have distinct layers. For example, a windowing system may span multiple "layers".

How does this compare against Network Layers?

Network Layers	System Layers
The boundary is more well-defined	The boundary is less well-defined
Information and data flows from the topmost layer down to the lowest layer, and is transmitted from the lowest layer to the topmost layer	Every layer has its own "processes" and "data", although ultimately, the raw processes and data are handled by the hardware
A concern of data confidentiality and integrity	The main concern is about access to the processes and memory/storage (both volatile and non-volatile memory). Hence, besides data confidentiality and integrity (e.g. password file), there is also a concern of process "integrity" – which is whether it deviates from its original execution path

Using System Layers in Security

Let us assume there to be a system with Layers 2, 1 and 0, with 0 being the most important or more privileged layer. Suppose an attacker access Layer 1. He can then access data in Layer 1 i.e. Layer 1 is compromised.

What we need to thus ensure is that the attacker **must not** be able to directly manipulate objects and processes in Layer 0. This is very difficult to ensure, due to possible implementation errors, overlooked design errors, etc.

In the case of the System Layers, the “least important” layer is the application layer, while the “most important” layer is the hardware layer. But this is not really always true. The principle is to ensure that attackers should not be able to go anywhere that he should not be able to go.

It is thus insightful to figure out which layer a security mechanism or attack resides at. A (layer-based) security mechanism should have a **well-defined security perimeter or boundary**, where the parts of the system that can malfunction without compromising the protection mechanism lie outside this perimeter. The parts of the system that can be used to disable the protection mechanism lie within this perimeter.

Basically, leave the vulnerable parts within the perimeter. However, quite often, it is difficult to determine this boundary or perimeter. An important design consideration of the security mechanism is how to prevent an attacker from getting access to a layer inside the boundary or perimeter.

For example, a Structured Query Language (SQL) Injection attack targets at the SQL Database Management System. The OS Password Management, which is in a layer below, should still remain intact even if the injection attack has been successfully carried out.

It is also possible that an application takes care of its own security i.e. self-secure itself. For example, if an application always encrypts its data before writing them to the file system, even if the access control of the file system is compromised e.g. a malicious user reads the files, the confidentiality of the data will still be preserved.

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

Access Control Model

Why Access Control?

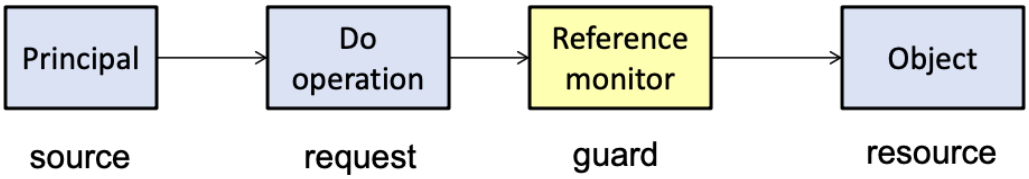
Access control is about the selective restriction of access to a place or other resource, and is required in computer systems, information systems, and physical systems, to prevent unauthorized people from accessing the system. The access control model gives a way to specify such restrictions on the subjects, objects and actions.

Different application domains have different interpretations of access control, and they have their own requirements as well. Some examples of application domains are:

- Operating system
- Social media (e.g. Facebook)
- Documents in an organization (which can be classified as “restricted”, “confidential”, “secret”, etc.)
- Physical access to different parts of a building

There is a concept called “Separation between policy and mechanism”. This design principle basically states that mechanisms should not dictate, or overly restrict, the policies. Decoupling the mechanism implementations from the policy specifications makes it possible for different applications to use the same mechanism implementations with different policies. Also, hardwiring policy and mechanism together makes policies rigid and harder to change, as trying to change the policy is likely to destabilize the mechanisms.

Principal/Subject, Operation, Object



As seen in the diagram above, a principal or subject wants to access an object with some operation. The reference monitor either grants or denies the access.

A principal is different from a subject in that the principal refers to the human user, while the subject refers to the entity in the system that operates on behalf of the principal, e.g. processes and requests.

Types of Accesses

Accesses can be classified into the following:

- Observe
  - To read a file or some data
  - For example, downloading a file from LumiNUS
- Alter
  - To write or delete a file, or change file properties
  - For example, uploading a file to the Workbin on LumiNUS
- Action
  - Execute a program

Who decides the access rights?

There are two approaches:

1. **Discretionary Access Control (DAC):** The owner of the object decides the rights
2. **Mandatory Access Control (MAC):** A system-wide policy that decides the rights, which must be followed by everyone in the system. It is stricter as it’s a fixed policy.

Specifying Access Rights

Access Control Matrix
We can use a table called the access control matrix to specify the access rights of a particular principal to a particular object. This is similar to an adjacency matrix.

For example:

	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}	{}

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

Although the above access control matrix can specify the access rights for **all possible pairs** of principals and objects, the table can be very large and is thus difficult to manage (i.e. low time complexity but high space complexity). Hence, it is often treated as an abstract concept only, and seldom explicitly deployed.

Much like an adjacency matrix can be expressed as an adjacency list or an edge list, the access control matrix can also be represented in two different ways: **Access Control List** or **Capabilities**.

Access Control List (ACL)
The access control list stores the access rights **to a particular object** as a list, i.e. it is object centered. This is an example:

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})	

Similar to how an adjacency list is implemented, the access control list may be implemented using a linked list concept.

Strength
The strength of an access control list is that it is easy to know all the subjects that can access a specific objects. This is because it is object centered.

Weakness
It is very difficult to find out all the objects that a specific subject can access. This will require O(os) searching through all o objects and their lists of subjects to find that individual subject.

Capabilities
Capabilities is the reverse of the access control list. It is subject centered, and every subject has a list of **capabilities,** where each capability is the access rights to an object.

A capability is defined as an unforgeable token that gives the possessor certain rights to an object.

Here is an example of an implementation of capabilities:

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})		

Strength
The strength of capabilities is that we can easily find out all the objects that a certain subject has access to, and the subject's access rights to that object.

Weakness
It is now very difficult to find out all the subjects that have access rights to a particular object.

Overall Drawback of both ACL and Capabilities
The sizes of the lists are still too large to manage. Hence, we need some ways to simplify the representation.

One such way is to group the subjects and objects and define the access rights on the defined groups. We thus need intermediate control.

Intermediate Control

In UNIX file permission, an ACL is used, but unlike the above implementation of the ACL, this ACL only specifies the rights for three parties:

- Owner
- Group
- World (others)

Subjects in the same group have the same access rights. Some systems thus demand that a subject is in a single group, but some systems do not put such a restriction.

Trivia: It is possible that an owner does not have read access, but others do.

There are many ways to perform this intermediate control.

Users and Groups

Groups are simply ways to classify users. In LumiNUS, project groups can be created, where objects created in that group can only be read by members of the group and the lecturers.

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

Privileges

We often use the term privilege for the access right to **execute** a process. Privilege can also be viewed as an intermediate control.

For example, privilege 1 is the access right to execute process `mysh.sh`, while privilege 2 is the access right to execute `sudo`. One user may be assigned privilege 1, while another user may be assigned privileges 1 and 2.

This is however a bit troublesome to assign these privileges individually.

Role-based Access Control (RBAC)

The grouping can be determined by the role of the subjects, and the role itself is associated with a collection of procedures. In order to carry out these procedures, access rights to certain objects are then required.

If a subject is assigned a particular role, then we can use the **least privilege principle** to determine the role's access rights. The least privilege principle states that only access rights that are required to complete the role will be assigned, i.e. it's a need-to-know principle.

For example, a teaching assistant's role is to enter the grades for his students. He should thus have a "write" access on the grades. However, he should not have the write access to the students' names, since it is **not required** for the TA to complete his tasks.

Protection Rings

We can think of it as rings within rings, where the innermost ring has the lowest number, 0, and outer rings have increasingly larger numbers. This is somewhat similar to the example raised when discussing System Layers.

If a process is assigned a number i, we say that it runs in ring i. Objects with smaller numbers are thus more important. Often, we call processes with a lower ring number as having a "higher privilege".

Whether a subject can access an object is determined by its assigned number. A subject cannot access (i.e. both read and write) an object with a smaller ring number. It can only do so if its privilege gets "escalated" – something we will cover later.

UNIX only has two rings – superuser and user. We can also view this as a special case of RBAC in the sense that the ring number is the "role" itself.

In the Protection Rings model, subjects can thus access objects that are classified with the same or lower privilege. There are, however, reasonable alternatives.

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

In both models, objects and subjects are divided into linear models e.g. level 0, level 1, level 2, and a higher level corresponds to higher "security". For example, we may have a tiering system such as Unclassified, Confidential, Secret, Top Secret, etc. This is thus called multilevel security, which is when we have information at different security levels and thus have incompatible classifications.

Bell-LaPadula Model
The Bell-LaPadula Model focuses on confidentiality, and has the following restrictions:

1. **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.

2. **No write down**

A subject has **write access** to objects whose information level is **above** its current clearance level.

This prevents a subject from passing information to levels lower than its current level.

For example, to prevent information leakage, a clerk working in the highly-classified department should not be gossiping with staff from departments of lower security levels.

Biba Model
The Biba Model focuses on integrity, and has the following restrictions:

1. **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.

2. **No read down**

A subject only has **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 confidentiality, a subject that can append to objects at higher security levels is able to distort its original content (renegotiation attack - still confidential but integrity compromised)

In a model that imposes both the Biba and Bell-LaPadula models, subjects then can only read and write to objects **in the same level.**

Summary of the two models:

Bell-LaPadula Model (Confidentiality)
- A subject at a given security level cannot read an object at a higher security level
- A subject at a given security level cannot write to any object at a lower security level
- Both of these:
    - Prevents people from reading stuff that's too secure for them
    - Prevents leakage of information downwards
- Deals with secure "states", and classifies objects by security

Biba Model (Integrity)
- A subject at a given level of integrity must not read data at a lower integrity level
- A subject at a given level of integrity must not write to data at a higher level of integrity
- Both of these:
    - Prevents corruption of data of higher integrity levels by unauthorized parties
    - Prevents corruption of subjects by data from lower integrity levels
- Deals with integrity, and groups data by ordered levels of integrity

Example of an Access Control Policy (From Tutorial 6)
Let's consider the access control policy of LumiNUS forums.

The users (principals) are the lecturer, teaching assistants, students and guests.

Information associated to a post include author's name, title, content, rating of 1 to 5 stars, and number of users who have read the post.

Naturally, mandatory access control applies, as the rights of the principals on the available objects are set by the LumiNUS system. Let us consider Teaching Assistants as part of Lecturer.

Object > Principal v	Whole post (with no child post)	Author's name	Title	Content	Viewed	Rating
Lecturer (post-owner)	delete, create*	r	rw	rw	r	rw
Lecturer (non-post-owner)	delete	r	rw	rw	r	rw
Students (post-owner)	delete, create*	r	rw	rw	r	r
Students (non-post-owner)	-	r	r	r	r	r
Guests	-	r	r	r	r	r

*A whole post is created with default values set on its fields

Alternatively, instead of splitting the principals into post-owners and non-post-owners, another way is to split the objects into post-owners and non-post-owners. However, the above representation is more compact and hence more visually appealing.

Access Control in UNIX/Linux

The following section contains various technical information about users, groups and processes.

Terminology

Objects

In UNIX, objects of access control include:

- Files
- Directories
- Memory Devices
- I/O Devices

All of these resources are treated as files.

Users and Groups

Each user:

- Has a unique user/login name
- Has a numeric **user identifier (UID)** stored in `/etc/passwd`
- Can belong to one or more groups:
    - The first group is stored in `/etc/passwd`
    - Any additional groups are stored in `/etc/group`

Each group:

- Has a unique group name
- Has a numeric **group identifier (GID)**

What is the purpose of UIDs and GIDs?

- To determine the **ownership** of various system resources
- To determine the **credentials** of running processes
- To control the **permissions granted** to processes that wish to access certain resources

Therefore, in UNIX, we can think of it as:
User (UID) + Group (GID) → Principal (UID + GID) → Subject (Process ID (PID)) → Object

Principals are made up of user identities (UID) and group identities (GID). The information on these principals or the user accounts are stored in the password file `/etc/passwd`

Example of how the user account may look like in that file:
`root:*:0:0:System Administrator:/var/root:/bin/sh`
In the above, `0:0` means that it has UID 0 and GID 0 respectively.

A special user is the **superuser**, with UID 0, and it usually has the username `root`. All security checks are turned off for `root`, as we mentioned earlier, UNIX's protection rings consists of only 2 rings: superuser and users.

Subjects are the processes. Each process has a process ID (PID). We can use the command `ps aux` or `ps -ef` to display a list of running processes.

Password File Protection

The passwd file is made world-readable because some information in `/etc/passwd` are needed by non-root processes. This file contains a list of user accounts in the format shown below.

```
jsmith:x:1001:1000:Joe Smith,Room 1007,(234)555-8910,(234)555-
0044,email:/home/jsmith:/bin/sh
```

The fields, in order from left to right, are:

1. **Username**
    a. This is the string that a user would type in when logging into the operating system, i.e. the logname.
    b. Must be unique across all users listed in the file.
2. **Information to validate a user's password**
    a. In most modern uses, this field is usually set to "x" or "*", or some other indicator.
    b. The actual password information is stored in a separate shadow password file.
    c. In older versions of UNIX, the location of "*" was the hashed password H(pw), allowing all users to have access to this hashed-password field.
        i. The availability of the hashed password allowed hackers to do offline password guessing.
        ii. Since many passwords are typically short, exhaustive search is able to obtain many passwords.
    d. Thus now the actual password is stored in `/etc/shadow`, which is not world-readable.
3. **User Identifier Number**
    a. Used by the OS for internal purposes
    b. Needs not be unique
4. **Group Identifier Number**
    a. Identifies the primary group of the user
    b. All files created by this user may initially be accessible to this group
5. **Gecos field**
    a. Commentary that describes this person or account
    b. Typically, it is a set of comma-separated values that includes the user's full name and contact details
6. **Path to the user's home directory**
7. **Program that is started every time the user logs into the system**
    a. For an interactive user, this is usually one of the system's command line interpreters (shells)

Shadow Password File

This is the file that actually contains the hashed passwords. The fields of the entry are:

- Login Name
- Hashed Password
- Date Of Last Password Change
- Minimum Password Age
- Maximum Password Age
- Password Warning Period
- Password Inactivity Period
- Account Expiration Date
- Reserved Field

The following is an example:

```
user1:$6$yonrs//S$bUdht9fglwJW0LduAxEJpcExtMfKokFMJoT8tGkKLx5xFGJk22/trPstOHXr4PdB
lD0AV1xko5LfFVDwW.aJS.:17275:0:99999:7:::
```

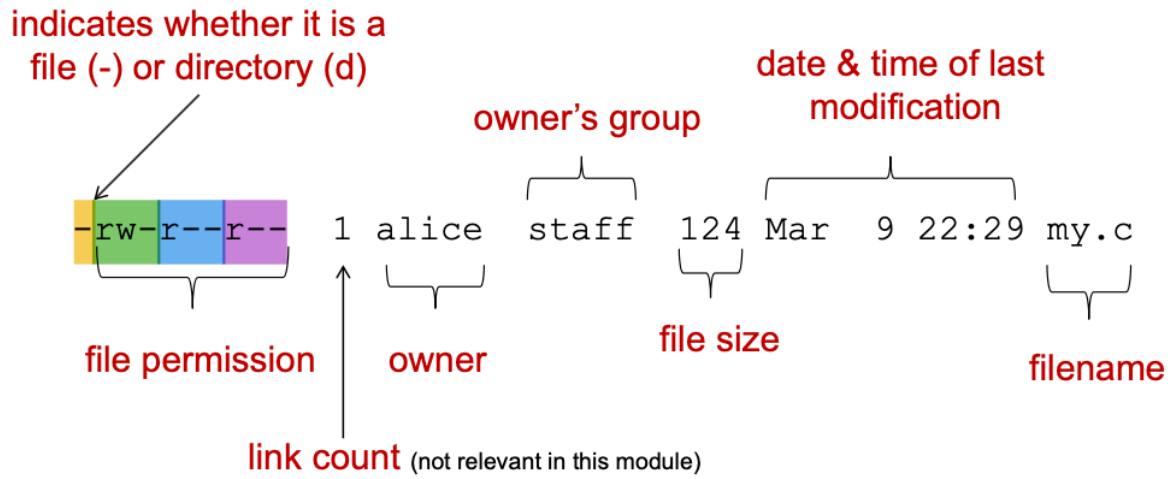
The second field, hashed password, is shown in red and has the following format: idsalt$hashed-key

- **id:** ID of the hash method used
  - 1 – MD5
  - 2a – Blowfish
  - 2y – Blowfish
  - 5 – SHA-256
  - 6 – SHA-512
- **salt:** up to 16 characters drawn from the set [a-zA-Z0-9./]
- **hashed-key:** hash of the password
  - 43 characters for SHA-256
  - 86 characters for SHA-512

Thus, in the example above, the id is 6 i.e. SHA512 is used, the salt is `yonrs//S`, and the hashed-key is the long 86 character string.

File System Permission

When we use `ls -la`, we see a list of all content within that certain directory, with the following information:



In the above image, we see that the **file permission** component is made up of 3 triples, that define the read, write and execute access. The breakdown is as follows:

- First triple: Owner ("user")
- Second triple: Group
- Third triple: Others (the "world")

Within each triple, the characters are:

- First character:
  - '-': access is not granted
  - 'r': read access is granted
- Second character:
  - '-': access is not granted
  - 'w': write access is granted (including delete)
- Third character:
  - '-': access is not granted
  - 'x': execute access is granted
  - 's': setuid/setgid and execute access is granted
  - 't': sticky and execute access is granted
  - 'S': setuid/setgid and execute access is **not** granted
  - 'T': sticky and execute access is **not** granted

The above is called the symbolic notation.

Changing File Permission Bits

You can use the chmod command to change the file permission bits. The command is:

```
chmod [options] mode[,mode] file1 [file2 ...]
```

The useful options available are:

- -R: Recursive, i.e. to include objects in subdirectories
- -f: force the processing to continue even if errors occur
- -v: verbose, i.e. show the objects changed

There are two notations for mode:

- Symbolic mode notation

    - Syntax: [references][operator][modes]
    - References:
        - u (user)
        - g (group)
        - o (others)
        - a (all)
    - Operators:
        - + (add)
        - - (remove)
        - = (set)
    - Mode:
        - r (read)
        - w (write)
        - x (execute)
        - s (setuid/gid)
        - t (sticky)
    - Examples:
        - `chmod g+w shared_dir`
            - Add write access to the shared directory for the group
        - `chmod ug=rw groupAgreements.txt`
            - Set read and write access for user and group to groupAgreements.txt
- Octal mode notation
    - This notation contains 3 or 4 octal digits.
    - The 3 rightmost digits refer to the permissions for the file user, the group, and others respectively.
    - There is an optional leading digit, allowing 4 digits to be given. This digit specifies the special file permissions:
        - setuid
        - setgid
        - sticky bit

#	Permission	rwx	Binary
7	read, write and execute	rwx	111
6	read and write	rw-	110
5	read and execute	r-x	101
4	read only	r--	100
3	write and execute	-wx	011
2	write only	-w-	010
1	execute only	--x	001
0	none	---	000

#	Permission	rwx	Binary
4	setuid	s/S	100
2	setgid	s/S	010
1	sticky	t/T	001
0	none	-	000

To some extent, these special file permissions “piggyback” on the third bit of the first three bits, i.e. using s/S/t/T instead of x or – to represent the leading digit.

- Examples
  - `chmod 664 sharedFile`
    - User can read and write
    - Group can read and write
    - Others can read only
  - `chmod 4755 setCtrls.sh`
    - setuid is enabled
    - setgid disabled
    - sticky bit disabled
    - User can read, write and execute
    - Group can read and execute
    - Others can read and execute

Directory Permissions
Directory permissions are slightly unique:
- **r:** Read access allows a user to view the directory’s contents
- **w:** Write access allows a user to create new files to delete files in the directory
- **x:** Execute access determines if a user can enter (cd) into the directory or run a program or script

If you want all group members to be able to write, edit or delete files within this directory, you can chmod g+w for that directory. However, this means that a user with write privileges in the directory can actually delete a file even if they do not have write permissions for the file.

Special File Permissions
Set-UID
- The process’ effective user ID is that of the **owner** of the executable file (which is usually root), rather than the user running the executable.
- For example: `-r-sr-sr-x 3 root sys 104580 Sep 16 12:02 /usr/bin/passwd` will run with root as the effective user ID.

Set-GID
- The process’ effective group ID is the **owner’s group**.
- For example, `-r-sr-sr-x 3 root sys 104580 Sep 16 12:02 /usr/bin/passwd` will run with sys as the effective group ID.

Sticky bit
- If a directory has the sticky bit set, the files within the directory can only be deleted by the owner of the file, the owner of the directory, or by root.
- This prevents a user from deleting files of other users from public directories such as `/tmp`.

Back to Access Control
In UNIX, objects are files, and each file is owned by a user and a group, and is associated with a 9-bit permission.

When a non-root user (subject) wishes to access a file (object), the following are checked in order:
1. If the user is the owner, the permission bits for the **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 right

3. If the user is not the owner nor a member of the group that owns the file, then the permission bits for **other** decide the access rights.

In general, the owner of a file and the superuser can change the permission bits.

Search Path Issues

When a user types in the command to execute a program, e.g. "`su`", without specifying the full path, what happens is that there will be a search for this program through the directories specified in the **search path.**

Use the command `echo $PATH` to see the search path. The search path is a list of directories, and the search will go through them one by one to find the program. Once a program with the name is found in a directory, the search stops and the program will be executed.

Suppose an attacker manages to store a malicious program at the directory that appears in the beginning of the search path, and the malicious program has a common name, e.g. "`su`". When a user exerts "`su`", the malicious program will be invoked instead.

To prevent such an attack, specify the full path always. Also avoid putting the current directory ("`.`") within the search path, e.g. `./a.out`. Say we have all these little scripts all over our filesystem; one day we will run the wrong one for sure. So, having our path as a predefined list of static paths is all about order and saving ourselves from a potential problem.

However, if you're going to add "`.`" to your `PATH`, it is recommended to append it to the end of the list (`export PATH=$PATH:.`). At least you won't override system-wide binaries this way.

However, if you're a root on the system and have your system exposed to other users' accounts, having "`.`" in PATH is a huge security risk – you can cd to some user's directory, and unintentionally run a malicious script there only because you mistyped a thing or there's a script that has the same name as a system-wide binary.

A low privilege user will have to invoke a program created by root/higher priv that will help to do operation for her

similar to a public function to help to edit a private member

UNIX/Linux: Privilege Escalation (Controlled Invocation)

Certain resources in the UNIX/Linus system can only be accessed by superuser, for example, listening at the trusted port 0-1023, accessing `/etc/shadow` file etc. Sometimes, a user needs those resources for certain operations such as changing their password. But it is not advisable for user to elevate their user status to superuser.

A solution is **controlled invocation**: the OS provides a predefined set of operations (programs) in superuser mode, and the user can then invoke those operations with the superuser mode.

Processes and Set-UID

A process is a subject and has a Process ID as identification. A new process can be created by executing a file or due to a fork in an existing process.

The process is associated with process credentials:

- Real UID
- Effective UID

The real UID is inherited from the user who invokes the process. It identifies the real owner of the process. For example, if the user who invokes it the process is Alice, then the process' real UID is that of Alice.

Let's say we invoked the process by executing a file. Then:

- If set-UID is disabled i.e. the permission bit is "`x`", then the process' effective UID is the same as the real UID
  - real UID is `alice`
  - effective UID is `alice`
- If set-UID is enabled i.e. the permission bit is "`s`", then the process' effective UID is inherited from the UID of the file's owner.
  - real UID is `alice`
  - effective UID is `root`
    - If the owner is Bob, then the effective UID would be `bob`

Process (Subject) Accessing Files (Objects)

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

For example:

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

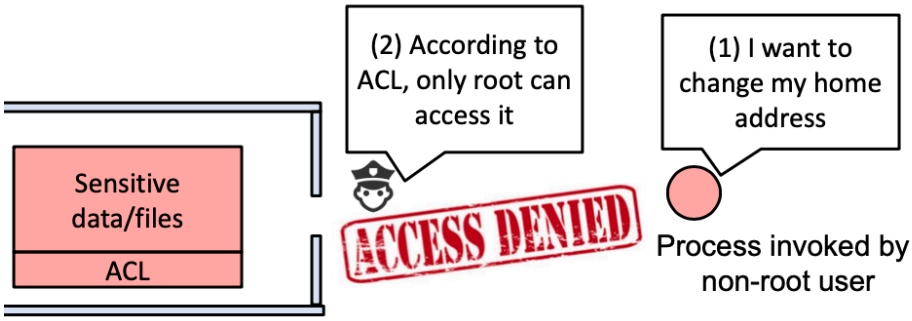
If the effective UID of a process is `alice`, then the process will be denied from reading the file. However, if the effective UID of a process is `root`, then the process will be allowed access.

Use Case of Set-UID

Consider a scenario where the file `employee.txt` contains personal information of the users. This is sensitive information, hence the system administrator sets it as non-readable except by `root`:

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

However, users should be allowed to self-view and even self-edit some fields 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 or write to it.



The solution would be to create an executable file `editprofile` owned by `root`:

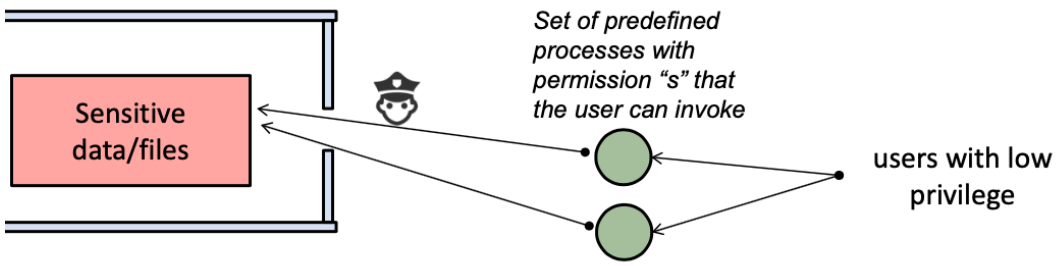
```
-r-sr-xr-x 1 root staff 6 Mar 18 08:00 editprofile
```

The program is made world-executable, such that any user can execute it. Furthermore, the set-UID bit is set, thus when it is executed by users other than `root`, the effective UID of the process will be `root`.

This is an example of a set-UID-root program or executable. Now, if `alice` executes the file, the process' real UID is `alice`, but its effective UID is `root`. This process can now read and write to the file `employee.txt`.

In the above example, the process `editprofile` is temporarily elevated to the superuser status i.e. `root`, such that it can access the sensitive data.

We can view the elevated process as the interfaces where a user can access the sensitive information. These processes are predefined "bridges" for the user to access the data. Note that the "bridge" can only be built by the `root`.
Bridges are just the programs that is created to help low privilege user do a high privilege operation
Due to the capability of the "bridges", it is important that they are correctly implemented and do not leak more information than required.



Privilege Escalation Attacks
Suppose that one of these processes is not implemented correctly, and contains an exploitable vulnerability. An attacker, by feeding this process with carefully-crafted input, can actually cause it to perform malicious operations, compromising confidentiality and integrity.

This can have serious implications, since the process is now running with an elevated privilege. An attack of such form is also known as a privilege escalation attack.

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.

This thus leads us to **secure programming** and **software security.**

Even More Complications – Real UID, Effective UID, Saved UID
The OS actually maintains three IDs for a process:

- Real UID
- Effective UID
- Saved UID

Effective (uid) is usually ruid, only when the file has 's' permission then will be owner uid

The saved UID is like a “temp” container and is useful for a process running with elevated privileges to drop its privilege temporarily – a good set-UID programming practice.

A process would remove its privileged user ID from its effective UID, but stores it in its saved UID. Later on, the process may restore the privilege by restoring the saved privileged UID back as its effective UID.

The details may easily confuse many programmers, not to mention that different UNIX versions may have different behaviours when it comes to this.

All this complexity is actually bad for security!

Backdoor Programs and Insider Threats (from Tutorial 7)

Some background info:

Shell program

Consider a UNIX-based OS and an executable program named `mysh`, which essentially is an existing command `sh` known as "shell". Running the program `mysh` with parameter `file1` will read and execute all commands contained in the file `file1`. Example:

```
$ mysh file1
```

Backdoor program

An attacker, who is assumed to already have a local access, can run the following commands using your account to plant a backdoor in your system in seconds:

```
$ cp /bin/sh /tmp/mysh
$ chmod u+s /tmp/mysh
```

Notice that you are the owner of the planted shell program. Therefore, when later the attacker run the planted set-UID program, the program's effective UID will be your UID.

Seeing set-UID-root in action

A set-UID-root program is an executable program owned by the root, and has its set-UID bit enabled. When it is invoked by a local non-root user, the process will run with its effective user ID (euid) set to the root, thus running with an elevated/escalated privilege.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
  printf("Real user id = %d, Effective user id = %d\n", getuid(), geteuid());
  return 0;
}
```

When you compile the above code as root, enable its set-UID bit and run it as a local non-root user, the output differs from when its set-UID bit is disabled.

Backdoor Creation as an Insider/System Administrator

Suppose a system administrator had root access on a system and was about to leave a company, and wanted to plant a backdoor into the target system. The assumptions are:

- He would still have local access, through
  - Given local user access
  - Knew the password of an existing local user
  - Created a local user that would go unnoticed
- Root password would be changed after he left
- His home directory would be erased
- The new system admin would compare the system-level directories, such as `/usr/bin`, to make sure that no malicious changes had been made

The above-mentioned method of using set-UID probably can't work in your favourite modern UNIX-based OS. This is because, due to the above security concerns, many OSes ignore the elevated effective UID when running shell scripts. This can be viewed as an "ad-hoc" patch to the security concern.

Nonetheless, there are methods to get around it, and there is fundamentally no way of stopping the system administrator. The “ad-hoc” patch only makes it slightly more difficult for the admin to create a backdoor.

Failed Attempt to Plant a Copied Shell
Back to the given scenario, when the admin still has root access, as the root, he can attempt to plant a copied shell by doing:

```
# mkdir -p /games/pokemon/scores
# cp /bin/sh /games/pokemon/scores/removescores
# chmod u+s /games/pokemon/scores/removescores
```

Unfortunately, when the admin later logs in as a non-root local user and executes `removescores`, the invoked shell won’t run with root privilege. You can check this by using the `id` command

This is because in Ubuntu, `/bin/sh` actually points to `/bin/dash` (Debian Almquist shell), which is a lightweight `bash` variant. As mentioned above, as a security protection measure, `bash` automatically downgrades its privilege. When `bash` detects that it is executed in a set-UID process, it will immediately change its effective user ID to the process’ real user ID, thus essentially dropping the escalated privilege. (Note that bash in older versions of Ubuntu, e.g. version 12.04, does not have this protection measure.)

However, there are ways to bypass this protection measure. The real root can still invoke `bash` without its privilege gets dropped. Hence, if we can make `bash` run with its real user ID set to 0, then the root privilege will be retained.

Successful Attempt
Instead of copying `/bin/sh`, as the root, the admin can just compile `getroot.c` to create an executable named `/games/pokemon/scores/removescores`, and enable its set-UID bit. As you can see, `getroot.c` first turns the current set-UID process into a real root process by invoking, among others, `setuid(0)`.

What `setuid(0)` does is, if the effective user ID is 0, then it will set the real user ID to 0. Else, an unprivileged program can use `setuid()` to change its effective user ID to the real user ID (in the case that it was started as a set-UID non-root e.g. Bob running a program with Alice’s user ID as his effective user ID). Only a privileged programme i.e. effective user ID of 0 can call `setuid()` to set both user IDs to a value of its own choice.

If the calling process is privileged, setuid(0) sets both the real and effective user IDs of the process to 0. In this case, `getroot.c` is privileged. Once both real and effective user IDs are set to 0, a shell `/bin/sh` is subsequently invoked. As a result, when the admin executes the planted executable as a non-root local user, he will obtain a shell running with a root privilege as planned.

The technique above is just one way of creating a backdoor. There are other ways, such as:

- setting a non-root local user to have UID 0
- including a non-root local user into the `sudo` group

Software Security

Overview of Software Security

Requirements of a program:

- Correct
- Efficient
- Secure

But oftentimes, a program behaves beyond its intended functionality!

Security Problems, Examples and Root Causes

Security problems faced:

- Insecure Implementation
  - Many programs are not implemented properly, allowing the attacker i.e. the person who is invoking the process, to deviate from the programmer's intents
- Unanticipated Input
  - The attacker may supply input in a form that is not anticipated by the programmer, which can unintendedly cause the process to:
    - Access sensitive resources
    - Execute some injected codes
    - Deviate from the original intended execution path
  - Regardless, the attacker is using the program to **elevate their privilege.**

Examples:

- Buffer Overflow
  - Morris worm (1988)
    - Exploited a UNIX finger service to propagate itself over the Internet
  - Code Red worm (2001)
    - Exploited Microsoft's IIS 5.0
  - SQL Slammer worm (2003)
    - Compromised machines running Microsoft SQL Server 2000
  - Various attacks on game consoles such that unlicensed software can run without the need for hardware modifications
    - Xbox
    - PlayStation 2 (PS2 Independence Exploit)
    - Wii (Twilight Hack)
- SQL Injection
  - Yahoo! (2012)
    - 450,000 login credentials claimed to be stolen using a "union-based SQL injection technique"
  - British Telco TalkTalk (2015)
    - A vulnerability in a legacy web portal was exploited to steal the personal details of 156959 customers
- Integer Overflow
  - European Space Agency's Ariane 5 Rocket (1996)
    - Unhandled arithmetic overflow in the engine steering software caused its crash, costing $7 billion
  - Resorts World Casino (2016)
    - A casino machine printed a prize ticket of $42,949,672.76.

Root causes:

- Functionality still remains the primary concern during design and implementation
  - Security is the secondary goal
  - Features pay the bills

- Unavoidable human mistakes
    - Lack of awareness of security problems
    - Careless programmers
- Complex modern computing systems
    - Many of the bugs are simple and seem easy to prevent
    - But programs for complex systems are large
        - Windows XP has 45 million source line of codes
    - This results in a large attack surface
- Learn Programming Fast! guide books
    - Guide books may be enough for learning basic functionality, but never enough for learning the subtle implications of the various functionalities
    - Result: Programs can do more than you expect!

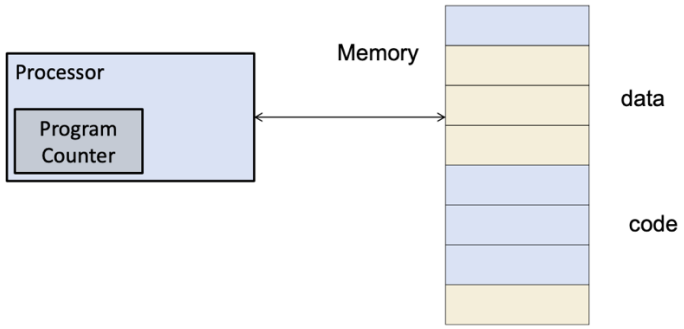
Computer Architecture

Code vs Data

Modern computers are based on the Von Neumann computer architecture. This means that:

- Code and data are stored together in the memory
- There is no clear distinction between code and data
- This is in contrast to the Harvard architecture, which has hardware components that separately store code and data
- **Implication: Programs may be tricked into treating input data as code!**
    - Basis for all code-injection attacks

Control Flow and Program Counter



The processor, as the name suggests, is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions.

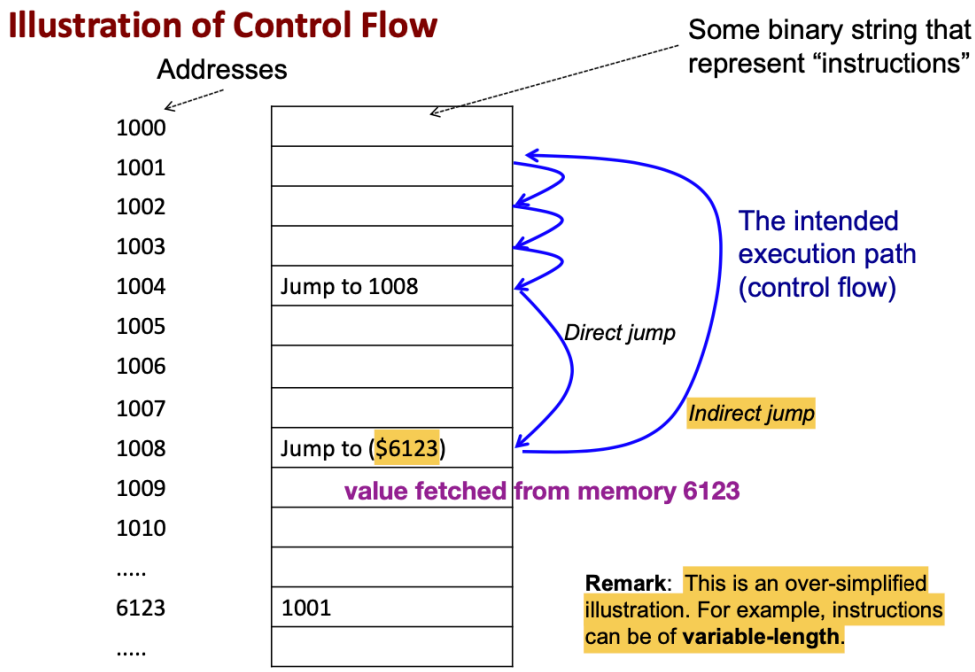
These instructions are read with the help of a program counter, or instruction pointer/counter. The program counter is a processor register that indicates where a computer is in its program sequence.

Usually, it holds the memory address of (or “points to”) the next instruction that would be executed. After an instruction is completed, the processor fetches the next instruction from the address stored in the program counter. Once this fetching is done, the program counter automatically increases by 1 (assuming a system with fixed length-instructions).

Changes to Program Counter

Besides getting incremented, the program counter can also be changed, for example by:

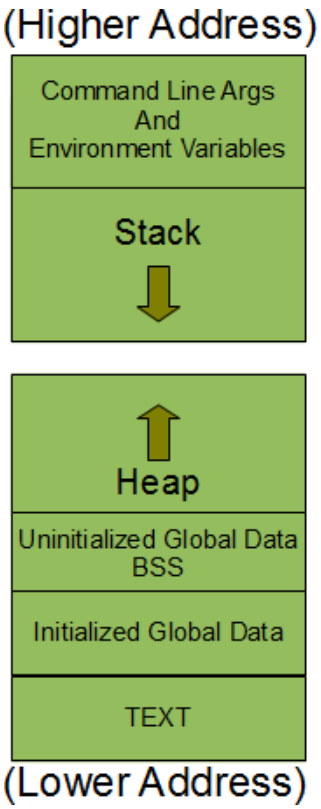
- Direct Jump
    - Replaced with a constant value specified in the instruction
- Indirect Jump
    - Replaced with a value fetched from memory
    - There are many different forms of indirect jump



Stack
Also called Execution Stack or Call Stack

Functions
Functions break code into smaller pieces and facilitate modular design and code reuse. A function can be called in many different locations, and can be called many times e.g. recursive function. How does the program know where it should continue from after the function finishes, and where are the function's arguments and local variables stored?

Process Memory Layout



As we can see in the image to the right, the Linux process memory contains the following:

- Command Line Arguments and Environment Variables
  - Stored at the top of the process memory layout at the higher addresses
- Stack
  - Memory area used by the process to store local variables of functions and other information that is saved every time a function is called. More about this later.
  - Grows downwards
- Heap
  - Used for dynamic memory allocation
  - Shared amongst all threads of a single process, but not between different processes running in the system
  - Memory from this segment should be used cautiously and should be deallocated as soon as the process is done using that memory
  - Grows upwards
- Uninitialized Global Variables
  - They are initialized with the value zero

    - BSS stands for “Block Started by Symbol”.
- Initialized Global Variables
- Text
    - Memory area that contains the machine instructions that the CPU executes
    - Shared across different instances of the same program being executed
    - Since there is no point in changing the CPU instructions, this segment has read-only privileges.

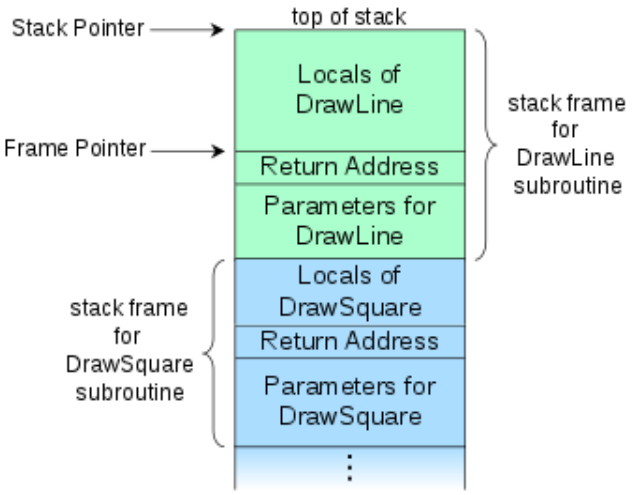
Call Stack
The Call Stack is a data structure in the memory that stores important information of a running process. As with a stack, it is LIFO or FILO, with push(), pop() and top() operations.

The location of the top element is referred to by the stack pointer. Generally, in this mod, we talk about stacks growing upwards, but from higher addresses to lower addresses. Some diagrams have the higher addresses above, and have the stack growing “downwards”, but it’s still the same.

Architectures do differ as to whether call stacks grow towards higher addresses or towards lower addresses. What is important in drawing a stack diagram is that the placement of the top, and so the direction of stack growth, can be easily understood. The logic of the diagram is independent of the addressing choice.

The stack helps to keep track of:
- Control flow information, such as return addresses
- Parameters passed to functions
- Local variables of functions

Each call of a function pushes an activation record, or **stack frame**, to the stack.



The stack frame at the top of the stack is for the currently executing routine. The stack frame usually includes at least the following items, in push order:
- Passed parameters
- Return address back to the routine’s caller e.g. in the DrawLine stack frame, an address into DrawSquare’s code
- Previous frame pointer or pointer to the previous stack frame
- Space for the local variables of the routine, if any

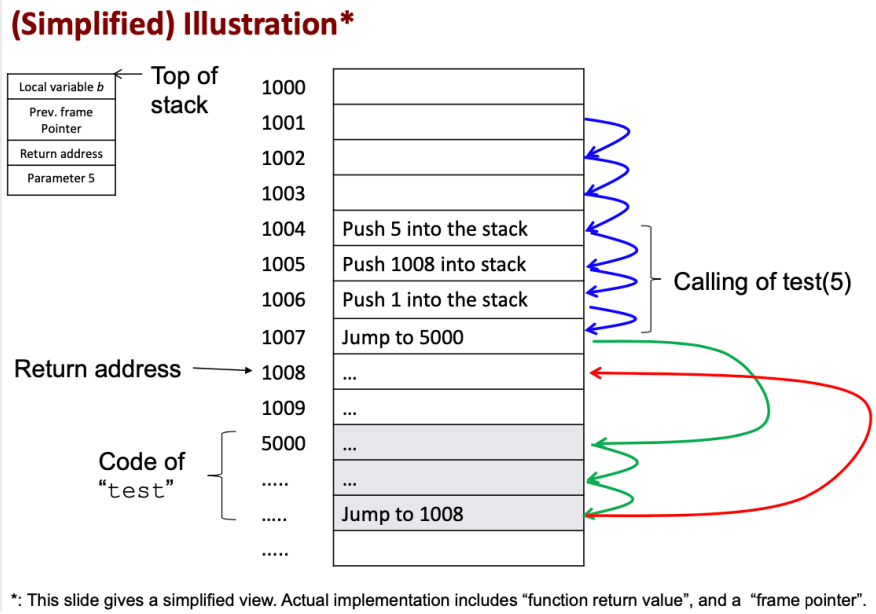
What is the frame pointer? Frame pointer is a pointer within the stack frame that is unchanging. Unlike the stack pointer and the top of the stack that changes as local values

are pushed on to and popped off of the stack, the frame pointer can allow for easier access to variables via an offset. This is also called the base address or ebp. The frame pointer will need to be pushed into the stack (as the pointer to the previous stack frame) and restored when the current function calls other functions as well.

This is very confusing stuff, but for now, this understanding is sufficient.

Example:
Consider a function test that takes in parameter int a and declares an int b = 1 within. When test(5) is invoked, the following happens:

- Stack pushing:
  - Parameter 5 is pushed into the stack
  - Return address is pushed into the stack
  - Previous frame pointer is pushed into the stack
  - Local variable b and its value 1 is pushed into the stack
- Control flow jumps into the code of test
- Execute test
- After test is completed, the stack frame is popped from the stack
- Control flow jumps into the restored return address.



*: This slide gives a simplified view. Actual implementation includes "function return value", and a "frame pointer".

Control Flow Integrity

The call stack stores a return address, the location of a to-be-executed instruction, as data in the memory. In fact, the instruction itself is also stored as data in the memory. This allows for greater flexibility, but also many security issues.

An attacker could compromise a process' execution integrity by either modifying the code or the control flow. It is difficult for the system to distinguish these malicious pieces of code from benign data.

Memory Integrity

In general, it is not easy for an attacker to compromise memory integrity, i.e. modify data in the memory. One way an attacker can do that is to exploit some vulnerabilities so as to "trick" the victim process to write to some of its memory locations, e.g. via a buffer overflow attack. The above mechanisms typically have some restrictions:

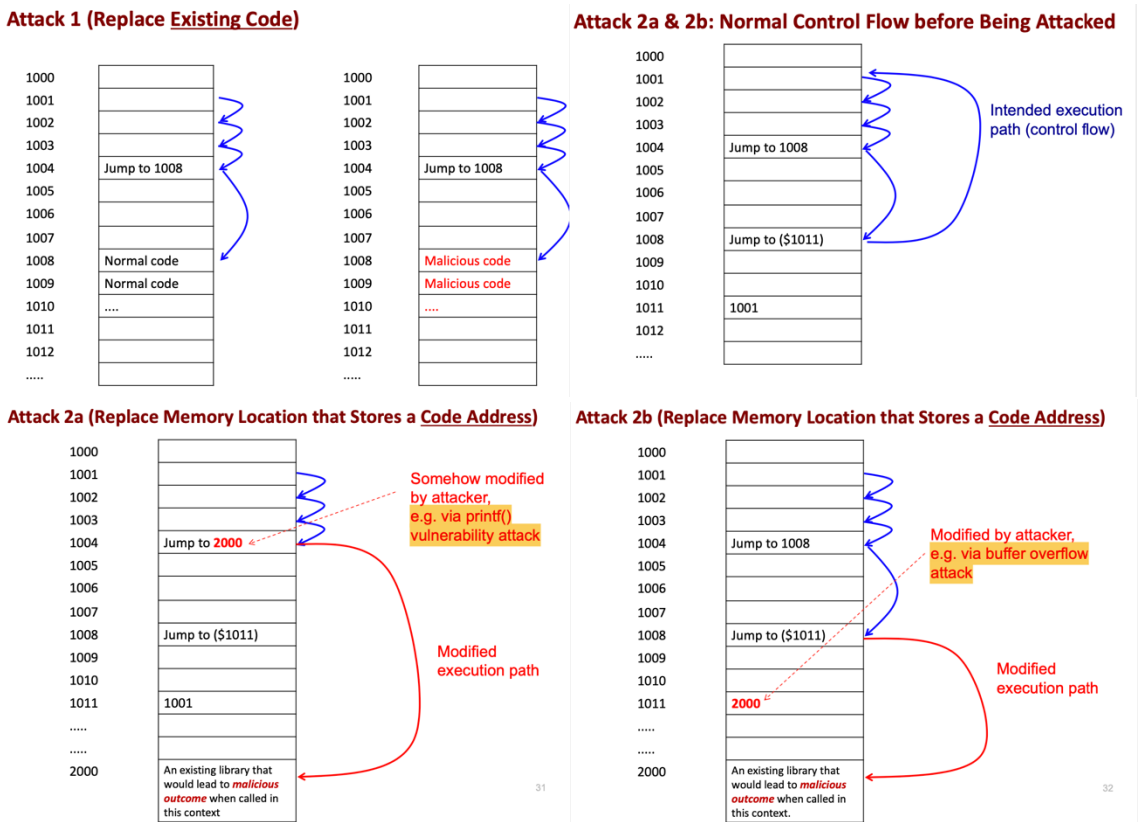
- The attacker can only write to a small amount of memory
- Or they can only write a sequence of consecutive bytes

Hence the attack needs to be very precise and surgical.

Possible Attack Mechanisms

Assuming that the attacker can now write to some memory locations, the attacker could:

1. Overwrite existing execution code portion with malicious code
2. Overwrite a piece of control-flow information:
    a. Replace a memory location storing a code address that is used by a **direct jump**
    b. Replace a memory location storing a code address that is used by an **indirect jump**



Attacks and Vulnerabilities

Printf() and Format String Vulnerability

printf() is the C function for formatting output. It is special in that it can take in **any** number of arguments.

The general format is as such:

```
int printf(const char* format, …)
```

where … is any amount of arguments that would be printed based on how the format string is written.

Format Specifiers

In the format string, there would be format specifiers to indicate the type of the variable to be printed. For example,

```
printf("1st string is %s, 2nd string is %s", s1, s2);
```

would cause printf() to print "1st string is ", look up for the second parameter in the call stack (which is s1) and print it, then "2nd string is ", and look up for the third parameter in the call stack and print it.

Note that the first parameter is the format string itself.

Some common format specifiers:

- %d: decimal (int)
- %u: unsigned decimal (unsigned int)
- %x: hexadecimal (unsigned int)
- %s: string ((const) (unsigned) char *)
- %n: number of bytes written so far, (* int)

Note that %s and %n are passed as a reference.

Missing Arguments

When only one parameter is supplied, only that parameter will be displayed.

```
printf("hello world");
```

That is fine if the parameter is a string by itself, but if it has a format specifier within, printf() will actually search for the second parameter in the stack to be displayed.

This is especially the case when the format string variable to be printed is supplied by the user. The attacker can actually get information by carefully designing the string to be printed.

For example, the attacker can:

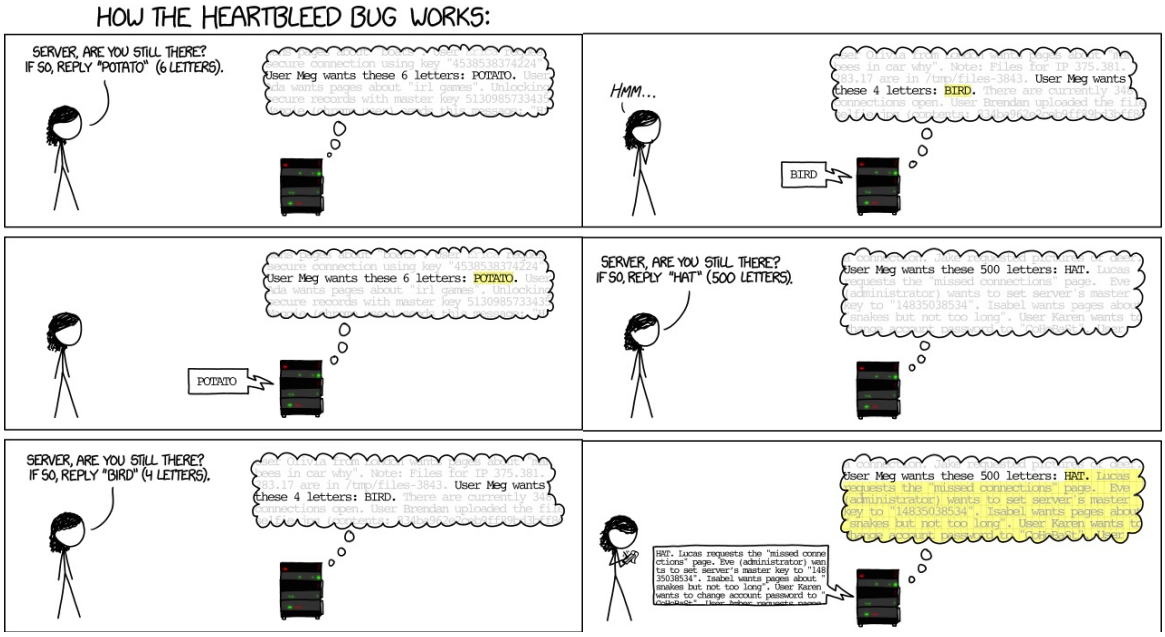
- Obtain more information of the program's call stack
  - Using "%d.%d.%d" to fetch values from the top of the stack and print them out as decimal values
    - Sample output: 73896.0.269401708
  - Using "%08x.%08x.%08x" to fetch values from the top of the stack and print them out as lower-case hexadecimal, with 8 digits shown (from the 8), and with padding/prefixes of 0s (from the 0)
    - Sample output: 000120a8.00000000.08309e6c
    - The 000120a8 and 00000000 actually corresponds to the first two decimals printed before
- Cause the program to crash
  - "%s" will fetch a number from the stack and treat this number as an **address**.
  - The printf() function will try to print out the memory contents pointed by this address as a string, until a null character is read.

    - Typically, a randomly fetched number is not an address, thus the memory pointed at by this number does not exist.
    - The program will thus crash
- Modify the program's memory content, such as by using %n. This is not covered in this module.

Big Picture Exploitation

This vulnerability can be exploited:

- In a multi-user setting
    - If the program has an elevated privilege i.e. via set-UID, an attacker may be able to obtain **system-level information**
- In a client-server setting
    - If the server program is vulnerable, the attacker may be able to submit a request and obtain sensitive information e.g. a secret key
    - This is how the **Heartbleed** bug works, where an Over-Read Request allows the client to receive information that's on the server



Preventive Measures

Avoid taking a user input as a format string. We can read the input into a separate variable, then print out that variable via our own format string. For example:

- printf(t)
    - Where t is supplied by the user, thus it is potentially insecure
- printf(f, t)
    - Where f is not supplied by the user, thus it is generally okay

Many modern compilers also statically check format strings and produce warnings for dangerous or suspect formats.

In GNU Compiler Collection, the relevant compiler flags are:

- `-Wall`
    - This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning).
- `-Wformat`

    - Check calls to printf and scanf, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense.
- `-Wno-format-extra-args`
    - If -Wformat is specified, do not warn about excess arguments to a printf or scanf format function. The C standard specifies that such arguments are ignored.
- `-Wformat-security`
    - If -Wformat is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to printf and scanf functions where the format string is not a string literal and there are no format arguments, as in `printf(foo);`.
- `-Wformat-nonliteral`
    - If -Wformat is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a va_list.
- `-Wformat=2`
    - Enable -Wformat plus additional format checks. Currently equivalent to -Wformat -Wformat-nonliteral -Wformat-security -Wformat-y2k.

Data Representation & Security

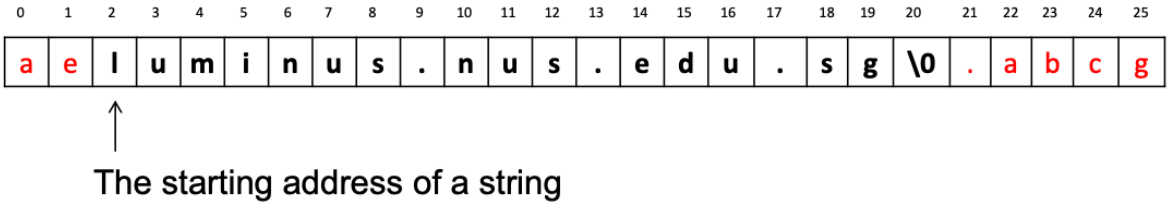
Different parts of a program or system may adopt different data representations. Such inconsistencies could lead to vulnerabilities.

For example, CVE-2013-4073: “Ruby’s SSL client implements hostname identity check, but it does not properly handle hostnames in the certificate that contain null bytes.”

One important data representation type is string.

String

In C, printf() adopts a efficient representation, where the length is not explicitly stored, and the first occurrence of the null character i.e. byte with value 0, indicates the end of the string, thus implicitly providing the length.



However, not all systems adopt this convention. There are two types:

- NULL-termination representation
- Non-NULL-termination representation

Exploitable Vulnerability 1: NULL-Byte Injection

A Certificate Authority **may accept** a hostname containing a null character, e.g. luminus.nus.edu.sg\0.attacker.com

A verifier who uses both of the above representation conventions to verify the certificate **could be** vulnerable. Consider a browser that does this:

- It verifies a certificate based on a non-NULL-termination representation
- It compares the name in the certificate and the name entered by user based on the NULL-termination representation

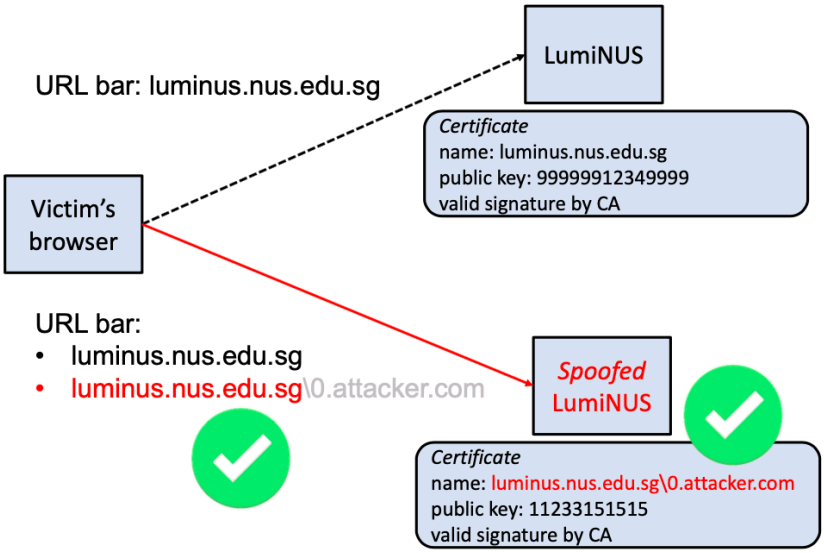
We can thus have this scenario:

1. Let's say an attacker registers the following domain name and purchases a valid certificate with this domain name from some CA: luminus.nus.edu.sg\0.attacker.com
2. The attacker then sets up a spoofed LumiNUS website on another web server. The attacker directs the victim to the spoofed web server by controlling the physical later or via social engineering.
3. When visiting the spoofed site, the victim's browser would then:
    a. Find that the web server in the certificate is valid based on the non-NULL representation i.e. luminus.nus.edu.sg\0.attacker.com
    b. Compares and displays the address as luminus.nus.edu.sg based on the NULL-termination representation

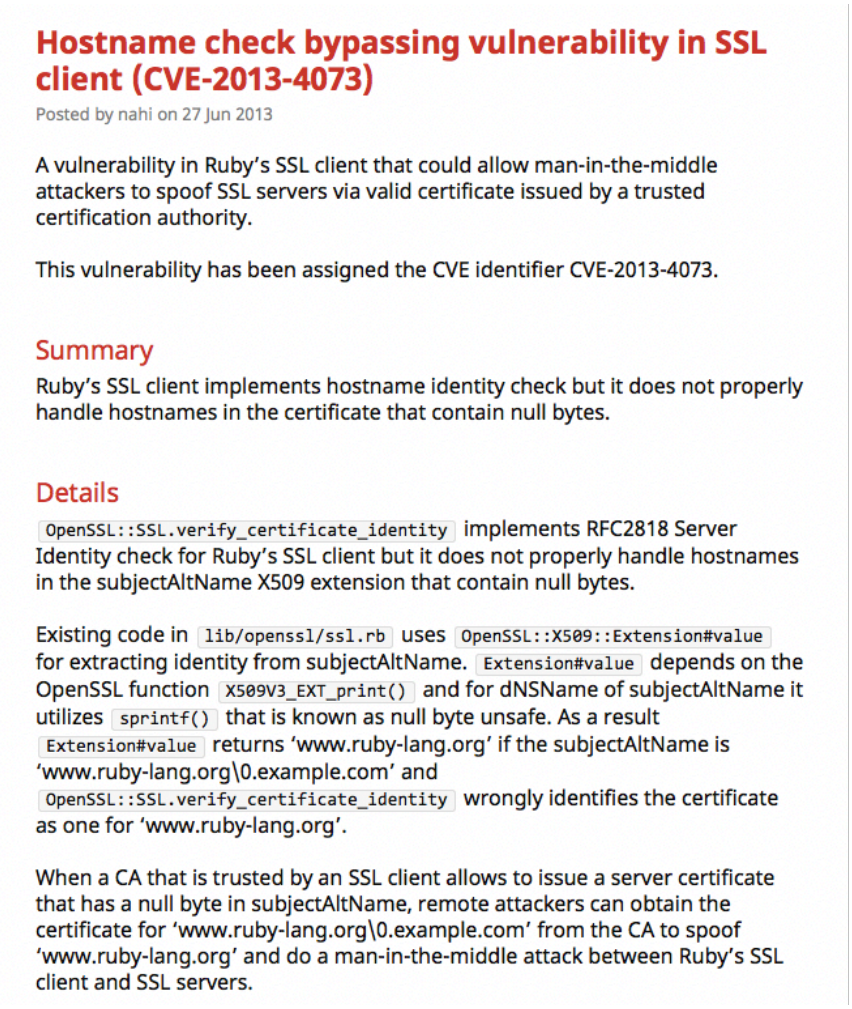
This is more effective than the normal web-spoofing attack, as a careful user would notice that the address displayed in the address bar is not LumiNUS, or that the address bar displays luminus.nus.edu.sg but the TLS/SSL authentication protocol rejects the connection i.e. certificate is not trusted.

Thus this attack is much more dangerous.

Below is a slide showing a summary:



Below are some details on the CVE-2013-4073. CVE stands for Common Vulnerabilities and Exploit.



Hostname check bypassing vulnerability in SSL client (CVE-2013-4073)

Posted by nahi on 27 Jun 2013

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.

This vulnerability has been assigned the CVE identifier CVE-2013-4073.

Summary

Ruby's SSL client implements hostname identity check but it does not properly handle hostnames in the certificate that contain null bytes.

Details

`OpenSSL::SSL.verify_certificate_identity` implements RFC2818 Server Identity check for Ruby's SSL client but it does not properly handle hostnames in the subjectAltName X509 extension that contain null bytes.

Existing code in `lib/openssl/ssl.rb` uses `OpenSSL::X509::Extension#value` for extracting identity from subjectAltName. `Extension#value` depends on the OpenSSL function `X509V3_EXT_print()` and for dNSName of subjectAltName it utilizes `sprintf()` that is known as null byte unsafe. As a result `Extension#value` returns 'www.ruby-lang.org' if the subjectAltName is 'www.ruby-lang.org\0.example.com' and `OpenSSL::SSL.verify_certificate_identity` wrongly identifies the certificate as one for 'www.ruby-lang.org'.

When a CA that is trusted by an SSL client allows to issue a server certificate that has a null byte in subjectAltName, remote attackers can obtain the certificate for 'www.ruby-lang.org\0.example.com' from the CA to spoof 'www.ruby-lang.org' and do a man-in-the-middle attack between Ruby's SSL client and SSL servers.

Encoding: ASCII and UTF-8

American Standard Code for Information Interchange (ASCII)

ASCII character encoding is a standard for electronic communication. It encodes 128 characters into 7-bit integers, with 95 printable characters (digits, letters, punctuation symbols) and 33 non-printing (control) characters.

There is also an Extended ASCII, EASCII or high ASCII character encoding, which comprises of:

- The standard 7-bit ASCII characters

- Additional characters

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Standard ASCII characters

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	|	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

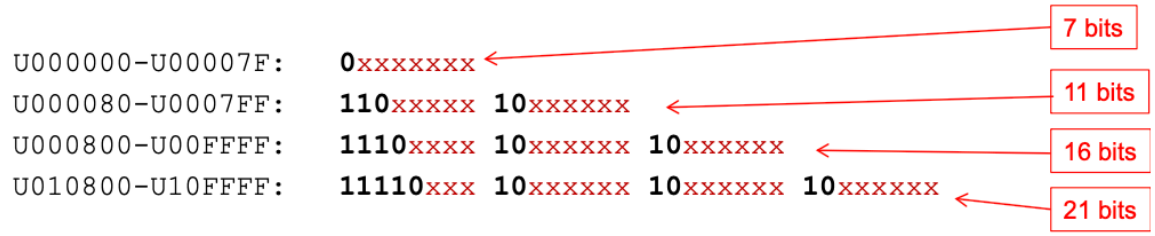
Source: www.LookupTables.com

Extended ASCII Codes

Unicode Transformation Format 8-bit (UTF-8)
UTF-8 is a character encoding that is capable of encoding all 1,112,064 valid code points in Unicode using one to four 8-bit bytes. It is a variable-length encoding, where code points with higher frequency of occurring are encoded with lower numerical values, thus using fewer byes.

The first 128 characters of Unicode correspond 1-to-1 with ASCII, and is encoded using a single octet with the same binary value as ASCII. Each byte thus starts with the bit 0 followed by the 7 bits of the original ASCII bits. Hence ASCII characters remain unchanged in UTF-8. There is backward compatibility with ASCII, as UTF-8 encoding was defined for Unicode on systems that were designed for ASCII.

Exploitable Vulnerability 2: UTF-8 "Variant" Encoding Issues
A Unicode character is referred to by "U+" and its hexadecimal digits. The following are byte representations of Unicode characters, with the left-hand side being the Unicode representation, and the right-hand side being the byte representation:



Notice that the prefix bits in the first byte changes based on the overall length, and that there are prefix bits as well in the continuation bytes or following bytes. The xxx bits are replaced by the significant bits of the **code point of the respective Unicode character**. By the rules above, the byte representation of a UTF-8 character is unique.

However, many implementations also accept multiple and longer "variants" of a character! In other words, there is more than one way to represent a single character using UTF-8. The reason is that different interpreters of UTF-8 interpret differently, and there is some room to accommodate the differences.

This leads to certain issues. Consider the following example.

Example 1: '/' (Representation using UTF-8 encoding)
Consider the ASCII character '/', whose ASCII code is: 0010 1111 = 0x2F

Under the UTF-8 definition, a 1-byte 2F is a unique representation. However, in many implementations, the following longer variants are also decoded to be '/':
- (2-byte) 110**00000** 10**101111**
- (3-byte) 1110**0000** 10**000000** 10**101111**
- (4-byte) 11110**000** 10**000000** 10**000000** 10**101111**

There is potential inconsistency when doing character verification before any operations that use this character.

Scenario:
In a typical file system, files are organised inside a directory. Suppose there is a server-side program that receives a string <file-name> from a client and carries out the following steps:
- **Step 1:** Append <file-name> to the prefix (directory) string /home/student/alice/public_html/ and take the concatenated string as string F

- **Step 2:** Invoke a system call to open the file F and send the file content to the client

In the above example, the client can be any remote public user, i.e. similar to a HTTP client. The original intention is the limit the files that the client can retrieve to only those under the directory public_html. This is called **file-access containment.**

However, an attacker may send in this string: ../cs2107report.pdf

The server would then try to read /home/student/alice/public_html/../cs2107report.pdf, which violates the intended file-access containment. To prevent this, the server may add an input validation step, making sure that the substring "../" does not appear within the input string.

In other words, there is now a:
- **Step 1.5:** Check that the <file-name> does not contain the substring "../", else quit.

Does this work?
Let us assume that the system call in Step 2 above uses a convention that can process "%" followed by two hexadecimal digits as a single byte, similar to URL encoding, e.g. "/home/student/%61lice/" will have the %61 replaced by a to give "/home/student/alice/".

We also assume that the system call uses UTF-8.

Then the following strings will actually all be equivalent to the string "../cs2107report.pdf":
- "%2Fcs2107report.pdf"
- "%C0%AFcs2107report.pdf"
- "%E0%80%AFcs2107report.pdf"
- "%F0%E0%80%AFcs2107report.pdf"

All these inputs, when decoded, will give the same system call as before.

In general, a blacklisting-based filtering system can be incomplete due to the "flexibility" of character encoding.

Example 2: IP Address (Representation as Strings and Integers)
The 4-byte IP address is typically written as a string, e.g. 132.127.8.16. Consider a blacklist that contains a list of banned IP addresses, where each IP address is represented as 4 bytes.

Assume that there is a function BL() (blacklist) that takes in 4 integers of type int (i.e. 32-bits) and checks whether the IP address represented by these 4 integers is in the blacklist:
int BL(int a, int b, int c, int d)

There are thus 4 arrays of integers, named A, B, C and D, and it simply tries to find i such that A[i]==a, B[i]==b, C[i]==c, and D[i]==d.

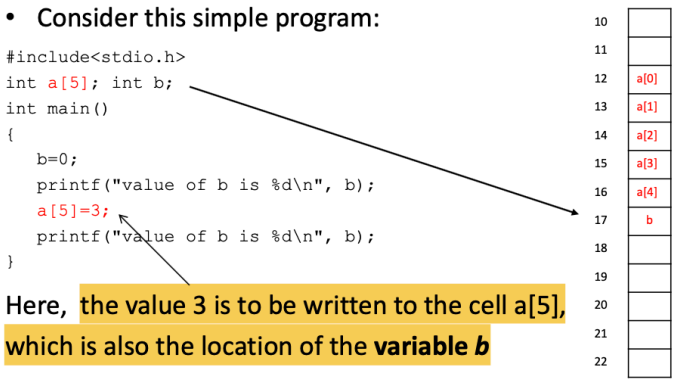
The overall program thus does the following:
1. Get string s from user
2. Check if the s is of the correct format i.e. 4 integers separated by ".". If not, quit, else extract a, b, c and d.
3. Call BL() to check, if in blacklist, quit.
4. Let ip = $a*2^{24} + b*2^{16} + c*2^{8} + d$, where ip is a 32-bit **integer**
5. Continue the remaining processes with filtered address ip.

What can happen now is that this process can still be exploited, as integers can go negative. Unexpected and undesired results may occur.

How to deal with issues with Data Representation: Use Canonical Representation
The important lesson is that we cannot trust input from the user, and we can never directly use input from them. Always convert the input to a standard i.e. canonical representation immediately.

Preferably, do not rely on the verification check done in the application i.e. do not rely on the application developers to write the verification. Rather, try to make use of the underlying **system access control mechanism.**

Buffer Overflow
In languages such as C and C++, programmers themselves are able to manage the memory via pointer arithmetic. There is no array-bound checking. Such flexibility is useful but very prone to bugs, leading to vulnerabilities.



The image above illustrates buffer overflow, or buffer overrun.

What is a buffer?
A data buffer, or just buffer, is a contiguous region of memory used to temporarily store data, while it is being moved from one place to another.

In general, a buffer overflow refers to a situation where data is written beyond a buffer's boundary. In the previous example, the array a is of size 5, and the location a[5] is beyond its boundary. Hence, writing to it causes a buffer overflow.

A well-known function in C that is prone to buffer overflow is a string copying function: strcpy().

strcpy()
Consider this code segment:

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

In the above, the length of s2 can be potentially more than 10, since the length is determined by the first occurrence of null. The strcpy() may copy the whole string of s2 to s1, even if the length of s2 is more than 10. Since the buffer size of s1 is only 10, the extra values will be **overflowed** and written to other parts of the memory.

If s2 is supplied by a **malicious** user, a well-crafted input can overwrite important memory and modify the computation!

As such, avoid using strcpy(), and use strncpy() instead. The function is as such:

`strncpy(s1, s2, n)`

where at most n characters are copied. There are still vulnerabilities if used improperly, it is merely **more secure.**

Stack Smashing

Stack smashing is a special case of buffer overflow that targets a process' call stack. As we've covered before, when a function is invoked, the return address will be pushed into the stack. If the stack is being overflowed such that the return address is modified, then the execution's control flow is changed.

A well-designed overflow could also inject the attacker's **shellcode** into the process' memory, then execute the shellcode. If the target executable is set-UID-root, then the injected shellcode runs with root privilege.

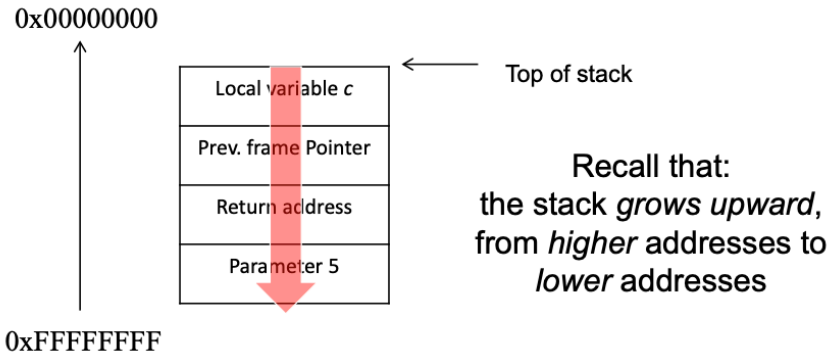
Some defences such as canary are available, which will be covered later.

Example of Stack Smashing

Consider the following segment of a C program:

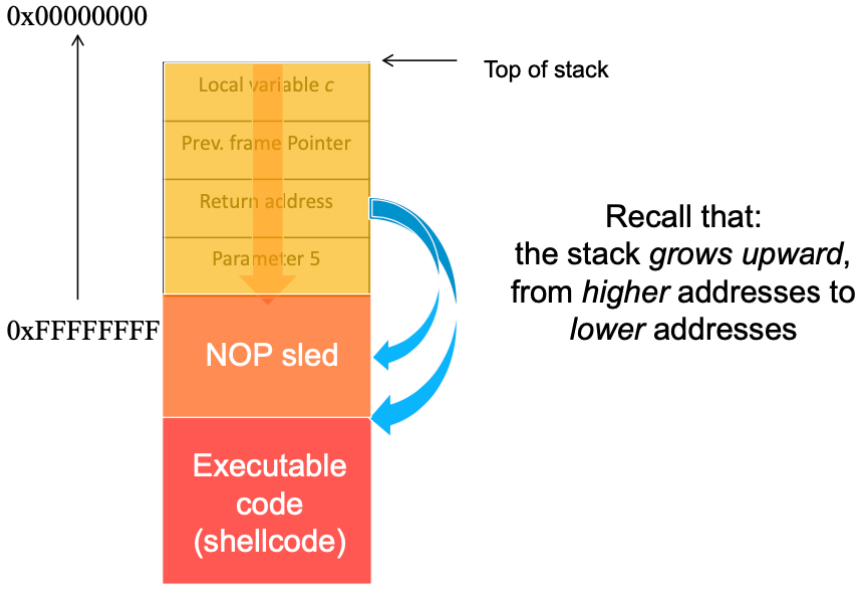
```
int foo(int a)
{
   char c[12];
   ......
   strcpy(c, bar); /* bar is a string input by user */
}

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

After foo(5) is invoked, a few values are pushed into the stack.



It is important to observe that the buffer c actually **grows towards** the return address! If an attacker manages to modify the return address, the control flow will jump to the address indicated by the attacker.

More can be seen from the diagram below:



What is the NOP Sled?
The NOP sled or NOP ramp is a sequence of NOP (no-operation) instructions meant to "slide" the CPU's instruction execution flow to its final, desired destination whenever the program branches to a memory address anywhere on the slide.

This is because sometimes attackers may not be able to have the return address point to the exact location or when the target address that the control flow is looking for is not known precisely. It creates a greater area for attackers to strike and still ensure that their shellcode will run.

While a NOP slide will function if it consists of a list of canonical NOP instructions, the presence of such code is suspicious and easy to automatically detect. For this reason, practical NOP slides are often composed of non-canonical NOP instructions such as a moving a program counter/register to itself or adding zero i.e. instructions that affect program state inconsequentially, which makes them a lot more difficult to identify.

Integer Overflow
The integer arithmetic in many programming languages is actually **modulo arithmetic**. Suppose a is a single byte (i.e. 8-bit) unsigned integer.

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

Its value would now be 0, since the addition is done in the form of modulo 256. Hence, the predicate (a < a+1) is not always true! Many programmers do not realise this, leading to possible vulnerabilities.

Code/Script Injection

As mentioned earlier, a key concept in computer architecture is how code is treated as data. This is actually unsafe, since many attacks inject malicious code as data, which then gets executed by the target system.

Scripting languages are programming languages that can be interpreted by another program during runtime, instead of during compilation. Some well-known examples are JavaScript, Perl, PHP and SQL. Many scripting languages also allow the script to be modified while being interpreted, opening up the possibility of malicious code being injected into the script.

We will consider a well-known attack: Structured Query Language (SQL) Injection (SQLI) Attacks.

Structured Query Language (SQL)

SQL is a database query language. Consider a database, which can be viewed as a table, with each column being associated with an attribute.

name	account	weight
bob12367	12333	56
alice153315	4314	75
eve3141451	111	45
peter341614	312341	86

Table name: client

This query script

```
SELECT * FROM client WHERE name = ‘bob’
```

searches and returns the **rows** where the name matches ‘bob’.

Sometimes, the script may also include variables. A script may first get the user’s input and stores it in the variable `$userinput`, and subsequently runs:

```
SELECT * FROM client WHERE name = ‘$userinput’
```

SQL Injection

In the above example, the database is designed such that the username is a secret, hence only the authentic entity who knows the name can get the record. However, if the attacker passes the following as the input:

```
Bob’ OR 1=1 --
```

The interpreter will execute the following:

```
SELECT * FROM client WHERE name = ‘Bob’ OR 1=1 --'
```

Since 1=1 for all, the interpreter actually returns all the records! The -- causes the interpreter to ignore everything behind, including the trailing single colon.

Code Injection + Buffer Overflow

Code injection is not limited to SQL injection. It is possible to exploit buffer overflow by injecting malicious code and then transferring the process execution to the malicious code.

Undocumented Access Points

For debugging purposes, many programmers insert undocumented access points to allow them to better inspect states. For example:

- Pressing a certain combination of keys will allow the values of certain variables to be displayed
- Certain input strings would cause the program to branch to some debugging mode

Many of these access points are left in the final production system, providing backdoors to attackers. A backdoor is a covert method of bypassing normal authentication.

Such access points are also known as Easter eggs. Some of these are benign and are intentionally planted by the developer for fun or publicity. However, there are also known cases where unhappy and disgruntled programmers planted the backdoors on purpose.

The backdoor can be accessed by the programmer and any other user who knows or discovers them.

Time-of-Check Time-of-Use (TOCTOU) Race Condition

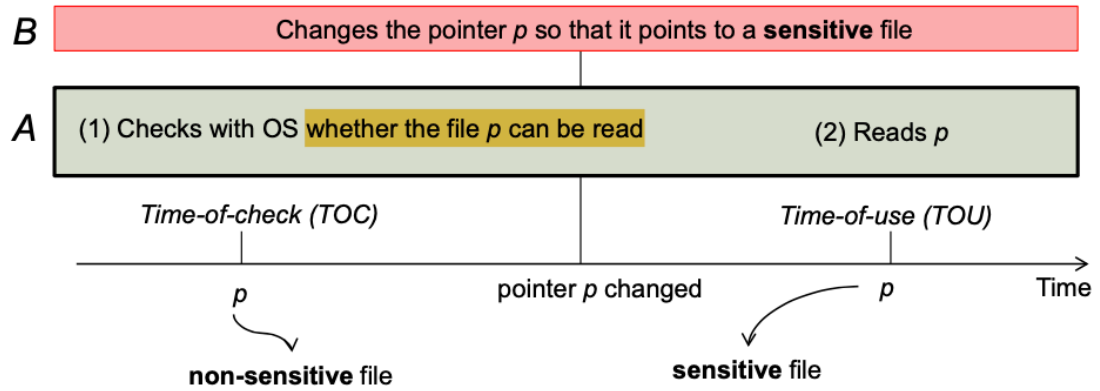
In general, a race condition occurs when multiple processes access a piece of shared data in a way that the final outcome depends on the **sequence of the accesses.**

In security, the multiple processes usually refer to:

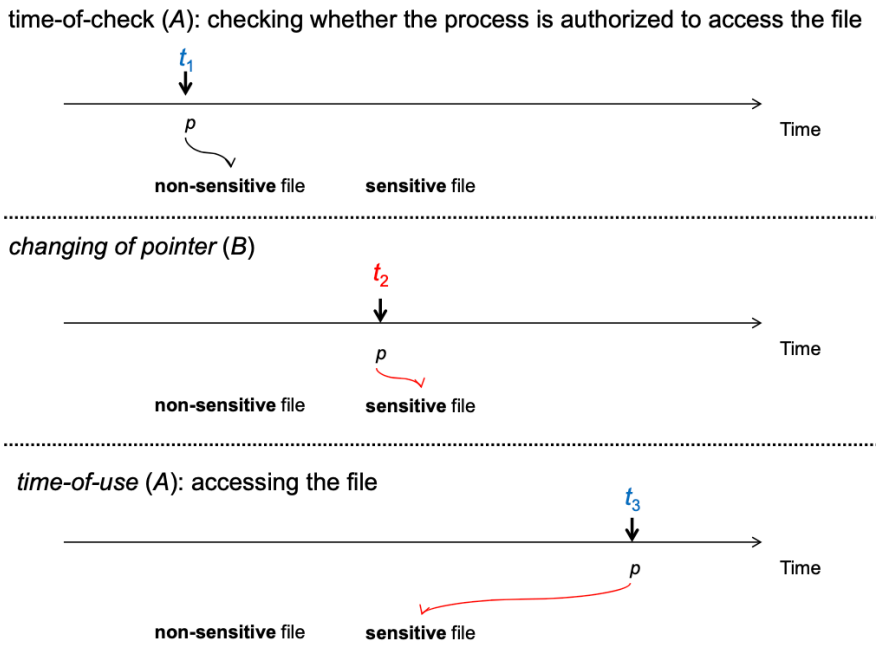
- A vulnerable process A that checks for permission to access a shared data, and subsequently accesses the data
- A malicious process B that swaps the data

These two processes can be run by a single malicious user in the system. There is thus a race between processes A and B. If B is able to complete the swapping after A checks for permission but before A accesses the data, then the attack as succeeded.

Refer to the diagram below:



Overview of TOCTOU



Step by step breakdown

Example CWE-367 (Common Weakness Enumeration)
This program needs to be run with elevated privilege, i.e. set-UID-root. Inside, there is a invocation of function access(f, W_OK).

This function checks whether the user executing the program has the permission to write to the specified filename f based on the process' **real UID and GID**, and returns 0 if the process **has** the permission. Since it checks real UID, a malicious user needs to find a way around it in order to get a sensitive target file.

TOC

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

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

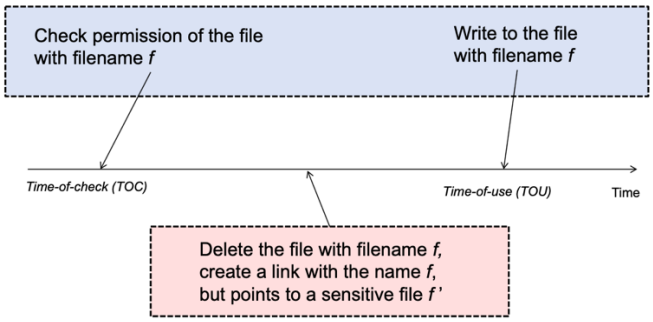
TOU

One key note about access() is that access() checks whether the calling process can access the file pathname. If pathname is a symbolic link, it is dereferenced, i.e. the contents that is linked is accessed. R_OK, W_OK, and X_OK test whether the file exists and grants read, write, and execute permissions, respectively.

This thus allows a malicious user Bob (real UID: bob, effective UID: root) to do the following:

1. Let us suppose that the filename f is `/usr/year1/bob/list.txt`, which is a file owned by Bob. Bob has the permission to write to it.
2. After Bob starts the process, he immediately replaces the file `/usr/year1/bob/list.txt` with a **symbolic link** that points to a sensitive system file.

a. He does this by running a script that does the following:
    i. Delete `/usr/year1/bob/list.txt`
    ii. Create a symbolic name with the filename f and point it to a system file via this UNIX command:
        1. `ln -s /usr/course/cs2107/grade.txt /usr/year1/bob/list.txt`
3. With some probability, Bob wins the race, and the process operates on the system file `/usr/course/cs2107/grade.txt`
4. Bob himself does not have write permission to the sensitive system file, but he is able to change the file due to his effective UID.



Defence Approach 1 for TOCTOU

In C programming, avoid using separate system calls that take the same filename as input. Instead, try to open the file only once, effectively locking it to block any further changes by other processes, and use the file handle/descriptor.

In terms of code, avoid using the access() system call; rather, open the file once using open() and use fstat() system call to check the permission.

```
//  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 );
}
```

TOC

TOU

Defence Approach 2 for TOCTOU

A better practice would be to avoid writing your own access-control on files, and leave the checking to the **underlying OS** after appropriately setting your process credentials. For instance, we can set the process' effective UID to the user real UID (normal/non-root) before accessing the file. This way, the OS checks the permission, and decides whether to grant or deny the access. Afterwards, reset the effective UID back to root if required.

```
//  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

…
```

Elevated Privilege

Lowered privilege

Elevated (root) privilege restored

15

Defence and Preventive Measures

Now that we've covered all the various attacks, it's time for us to look at what we can do against them. Many bugs and vulnerabilities are due to a programmer's ignorance. It is difficult to ensure that a program is bug-free. However, there are various useful countermeasures available.

Input Validation using Filtering

In almost all examples we have seen (except TOCTOU), attacks are carried out by feeding carefully-crafted input to the system. Those inputs do not follow the expected format, e.g. input is too long, contains control or meta characters, contains negative numbers etc.

Hence, one preventive measure is to perform an input validation or filtering whenever an input is required from the user. If it is not of the expected format, reject it.

Difficulties

It is difficult to ensure that the filtering is perfect and complete, as seen in the case of UTF-8. There may be representations that a programmer is not aware of, and a filter that completely blocks all bad input while still accepting all legitimate inputs is **very difficult to design**.

Approaches

There are generally two approaches:

1. White List
    a. A list of items that are known to be benign and are allowed to pass
    b. Could be expressed using regular expression
    c. However, some legitimate inputs may be blocked
2. Black List
    a. A list of items that are known to be bad and to be rejected
    b. For example, to prevent SQL injection, if the input contains meta characters, reject it
    c. However, some malicious input may be passed

The White List approach is more secure, but may not be more desirable. It is a trade-off.

Using "Safer" Functions

We can completely avoid functions that are known to be insecure, and use the **safer** versions of these functions.

For example, we can use strncpy() instead of strcpy(). However, even then, there could still be vulnerabilities. These functions are merely safer, it does not mean that they cannot be exploited still.

Bounds Checking and Type Safety

Bounds Checking

Some programming languages such as Java and Pascal perform bounds checking at runtime. In other words, when an array is declared, its upper and lower bounds have to be declared.

At runtime, whenever a reference to an array location is made, the index or subscript is checked against the upper and lower bounds. In other words, when doing a[i] = 5;, the following happen:

- Check if i >= lower bound
- Check if i <= upper bound
- Then assign 5 to the memory location

If the checks fail, the process will be halted, and an exception may be thrown. The added 2 steps reduce efficiency, but it prevents buffer overflow.

Many of the known vulnerabilities are actually due to buffer overflow, which can be prevented by this simple bounds checking.

The infamous C and C++ do not perform bounds checking, yet so many pieces of software are written in C or C++! Here's a sharing by a Turing Award winner:

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. For example, trying to do a = b; when a is an 8-bit integer and b is a 64-bit integer would throw an error.

This can be done at runtime i.e. dynamic type checking, or when the program is being compiled i.e. static type checking.

Bounds checking can also be considered as one mechanism that ensures "type safety". For example in Pascal:

```
Type    indexrange = 1..10;
Var     A: array [indexrange] of integer;
Begin
    A[0] = 5;  <-------------  wrong type
```

Memory Protection
Randomization
It is to the attacker's advantage when data and codes are always stored in the same locations in memory. As such, we have **address space layout randomization (ASLR)** as a prevention technique that can help decrease the attacker's chances.

ASLR randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.

We can turn disable ASLR in Linux via:

```
sudo sysctl -w kernel.randomize_va_space=0
```

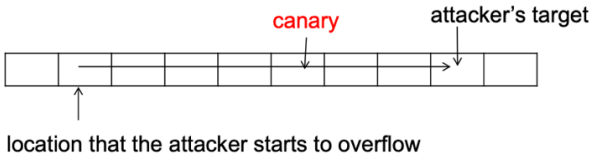
Canary
Canaries are secret values inserted at carefully selected memory locations during runtime. These secret values are not to be modified; checks are carried out at runtime to ensure that the values remain the same. If they have been changed, the program will halt.

Canaries can help to **detect (but not prevent)** buffer overflow, especially stack overflow. In a typical buffer overflow situation, consecutive memory locations have to be ran over. The canaries would naturally be modified.

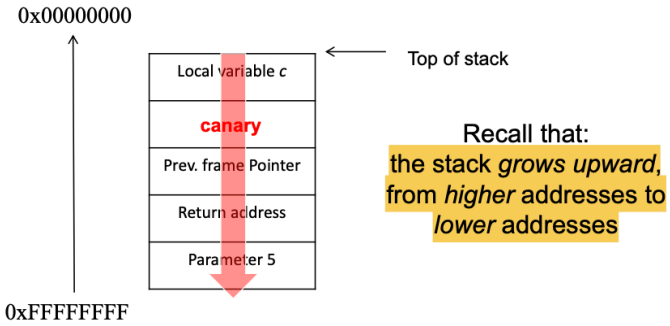
It is also important to keep the values secret. If the attacker knows the secret value, they can simply write the value to the canary while overrunning it, making it impossible to detect.

In Linux, we can turn off gcc canary-based stack protection by supplying this flag when compiling: `-fno-stack-protector`

This is the model of the memory:



Rough representation



Canary is located below the local variables

Let us look at some examples of how canaries can help detect stack smashing. Let us look at the following program:

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

int main(int argc, char *argv[])
{
    char text[15];

    strcpy(text, argv[1]);
    printf("Your supplied argument is :%s\n", text);

    printf("main() is exiting\n");
    return 0;
}
```

When compiled with stack protector:

```
./program-wsp aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
main() is exiting
*** stack smashing detected ***: ./program-wsp
terminated
Aborted (core dumped)
```

Canary is checked only when the function exits, hence the program completes.

When compiled without stack protector:

```
./program-wosp aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
main() is exiting
Segmentation fault (core dumped)
```

The segmentation fault is due to the return address pointing to an invalid instruction. If the attacker is smart, the return address would actually still point to a valid address, hence stack smashing would not be detected.

Similar events would occur if the program calls another function with stack smashing. Once the function completes, canary check would terminate the program, while segmentation fault may occur depending on the return address.

Code Inspection

One common sense solution is to check our code. We can do it manually, but this is tedious and often not very reliable.

The other way is to do automated checking using some automations and tools. An example would be taint analysis, or taint propagation analysis.

In taint analysis:

- Variables that contain input from users are labelled as **sources**.
- Critical functions are labelled as **sinks**.
- Taint analysis checks if any of the sinks' arguments could potentially be affected or tainted by a source
- For example
    - Source is user input
    - Sink is the opening of a system file or the evaluation of a SQL command
- If so, a special check i.e. manual inspection would be carried out

Taint analysis can be both static, i.e. checking the code without running or tracing it, or dynamic, i.e. running the code with some inputs. However, code coverage is an issue for dynamic analysis, since you do not know what are all the possible inputs that can result in issues.

Testing

Vulnerabilities can be discovered via testing. There are three types of testing:

- White-box testing
    - The tester has access to the application's source code
- Black-box testing
    - The tester does not have access to the source code
- Grey-box testing
    - A combination of the above, where the tester has some reverse-engineered binary or executable

Security testing attempts to discover areas susceptible to intentional attack, and hence would test for inputs that rarely occur under normal circumstances, e.g. very long names, names with numeric values, strings containing meta characters, etc.

Fuzzing is a technique that sends **malformed inputs** to discover vulnerabilities. Fuzzing can be both automated or semi-automated, and is more effective than simply sending in random inputs.

Principle of Least Privilege

We have covered this principle before. Basically, when writing a program, be conservative in elevating privileges. When deploying a software system, do not give users more access rights than necessary, and do not activate unnecessary options.

For example, 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 or off.

Should all features 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 switching off all unnecessary features. Your clients might not be aware of the implications and thus are at a higher risk.

Hardening refers to the process of securing a system by reducing its surface of vulnerability. This is done by reducing the number of functions in a system.

Patching

This is the lifecycle of a vulnerability:

1. Vulnerability is discovered
2. Affected code is fixed
3. Revised version is tested
4. Patch is made public
5. Patch is applied

In some cases, the vulnerability may be announced without any technical details before a patch is released. This vulnerability is likely to be known to only a small number of attackers, or even none, before it is announced.

When a patch is released, the patch may actually help attackers to derive the vulnerability, as now they can inspect the patch. Hence, interestingly, the number of successful attacks can go up after the vulnerability or patch is announced, since more attackers would be aware of the exploit.

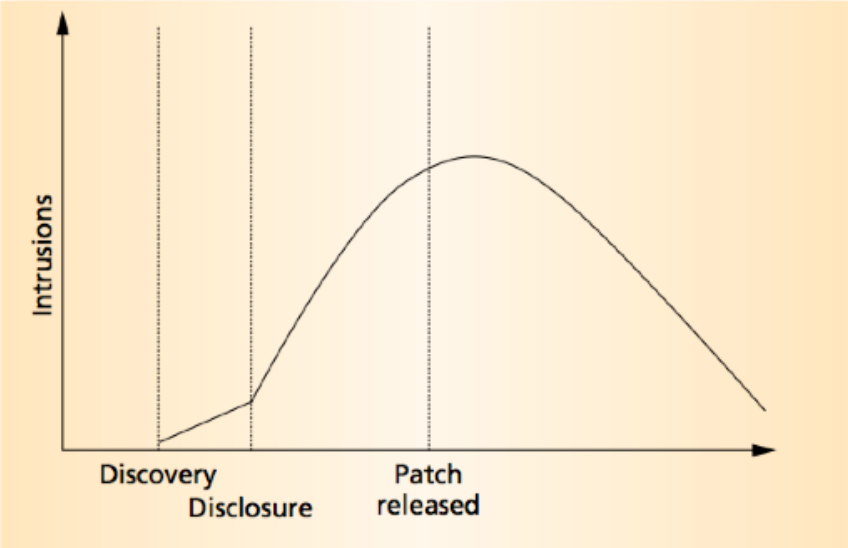


Figure 1. Intuitive life cycle of a system-security vulnerability. Intrusions increase once users discover a vulnerability, and the rate continues to increase until the system administrator releases a patch or workaround.

It is thus crucial to apply a patch timely. However, it is much more complex than it seems:

- For critical systems, it is not wise to apply the patch immediately before rigorous testing
    - For example, a patch may cause the train scheduling software to crash
- Patches may affect the applications, thus affecting the operations of organisations
    - For example, after a student applied a patch on his Adobe Flash, he could not upload a report to LumiNUS and thus missed a project deadline.

It is so complex that Patch Management is an actual field of study!

Summary

We thus have the following defence mechanisms and countermeasures, to be adopted at different stages of the Software Development Lifecycle (SDLC):

- Development Stage
    - Using Safer Functions
    - Bounds Checking and Type Safety
    - Filtering (Input validation)
    - Code Inspection (Taint analysis)
    - Principle of Least Privilege*
    - Executable Generation with Enabled Memory Protection*
- Testing Stage
    - Testing: Fuzzing and Penetration Testing
- Deployment (including Software Execution) Stage
    - Runtime Memory Protection*: Randomization
    - Principle of Least Privilege*: Disable Unnecessary Features
    - Patching

* applies to multiple stages

Tutorial Learnings

strcat(dest, source);
Unsafe. The buffer dest can get overflowed since there is no limit to the number of characters concatenated into it.

strncat(dest, source, n);
Safe if n < the remaining characters (bytes) available on the dest buffer. Note that the dest buffer must be large enough to contain the concatenated resulting string and also an additional null character.

memcpy(dest, source, n);
Safe if n ≤ the sizes of the dest and src buffers. Note that memcpy performs a binary-copying operation.

strncpy(dest, source, strlen(source))
Unsafe. The buffer dest can get overflowed since strlen(source) can be greater than dest's length.

printf(f, str)
Potentially unsafe if f comes from a user input.

printf("hello my name is %s", str);
Safe.

sprint(str, f)
Unsafe. The buffer str can get overflowed since the formatted string output can be longer than str's length.

printf("Please key in your name: "); gets(str);
Unsafe. The buffer str can get overflowed since there is no limit to the number of characters read and stored into it.

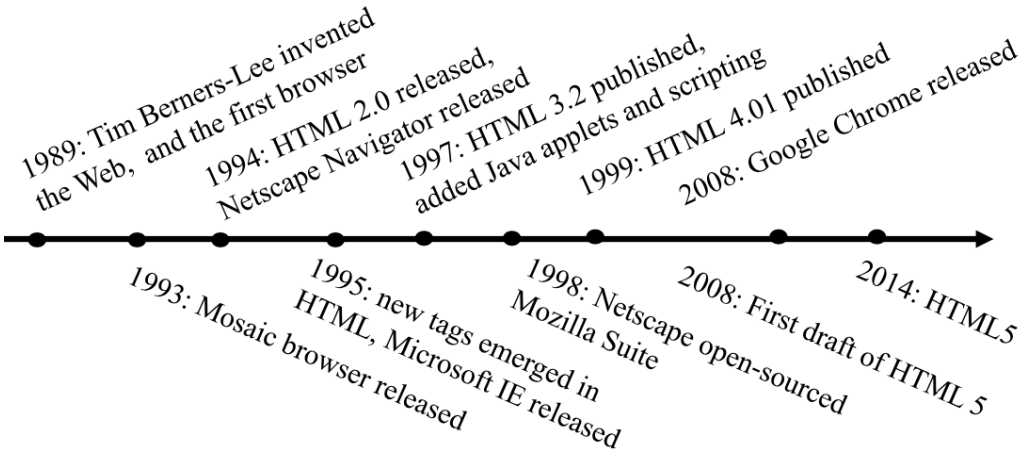
scanf("%s", str);
Unsafe. The buffer str can get overflowed since there is no limit to the number of characters read and stored into it.

scanf("%20s", str);
Safe if the size of str > 20 since at most 20 characters are stored by scanf() into str. Note that a terminating null character is added at the end of str. The specified maximum input length (i.e. 20) does not include this additional terminator character. As such, str must be at least one character longer than the specified maximum input length.

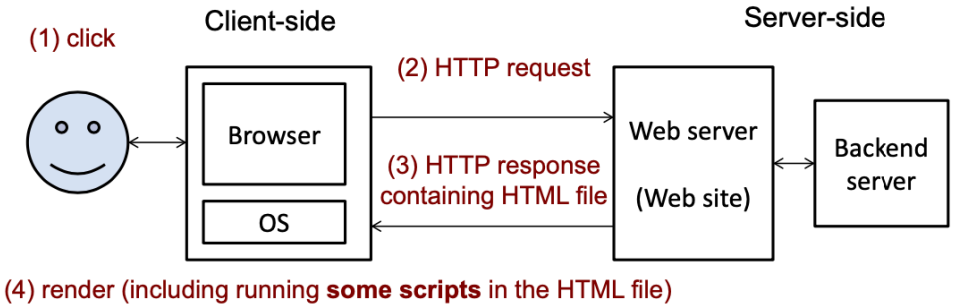
Accessing Uninitialized Variables
In C, the value of an uninitialized local variable is indeterminate/undefined, which basically can be anything. Accessing such an uninitialized variable leads to an "undefined behaviour". A program which prints out an uninitialized array poses a security risk since the printed data could be sensitive. This is the case since memory chunks in a running process' memory are basically "recycled", both between different process executions and during the process execution. If the array happens to contain a sensitive piece of information, the information will then get leaked out to the external user.

Web Security

Background



Overview of HTTP



1. A user clicks on a URL link e.g. luminus.nus.edu.sg/
2. A HTTP request is sent to the server, with cookies if any
3. Server constructs and include a HTML file inside its HTTP response to the browser, likely with cookies
4. The browser renders the HTML file, which describes the layout to be rendered and presented to the user, and the cookies are stored in the browser.

Sub-Resources of a Web Page

A HTML page may contain sub-resources such as images, multimedia files, CSS and scripts, including ones from external or third-party websites. When parsing a page with sub-resources, the browser also contacts the respective server for each sub-resource. A separate HTTP request for every single file on a page is thus made, since each file requires its own.

Interactive Query-Page

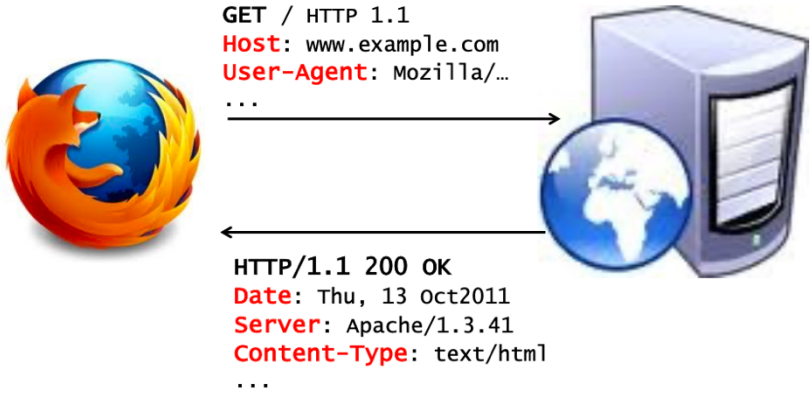
Let us take a look at Google.

1. When we visit their site, a HTML file is sent by the server to the browser. The browser renders the site.
2. The user enters the search keywords “CS2107 NUS”.
3. The browser, by running the HTML file, constructs a query, for instance: https://www.google.com/search?client=safari&source=hp&ei=MJHSXdiAEoHLvgTY64qAAw&q=**CS2107+NUS**&oq=**CS2107+NUS**&gs_l=... as URL parameters. These information are useful for the server, and could even be **in the form of a script.**

4. The server constructs a reply. In some cases, the reply contains substrings sent in Step 3.

HTTP Request and Response Messages
For every message, both request and response, headers are used. There are many headers used.

The key ones we would need to be aware of are:



HTTP Request

- Request line
  - GET /test.html HTTP/1.1
- Request headers
  - Accept
    - image/gif
    - image/jpeg
    - */*
  - Accept-Language
    - us-en
    - fr
    - cn
  - Cookie
    - theme=light;
    - sessionToken=abc123;
- Empty/blank line
- Optional message body

HTTP Response

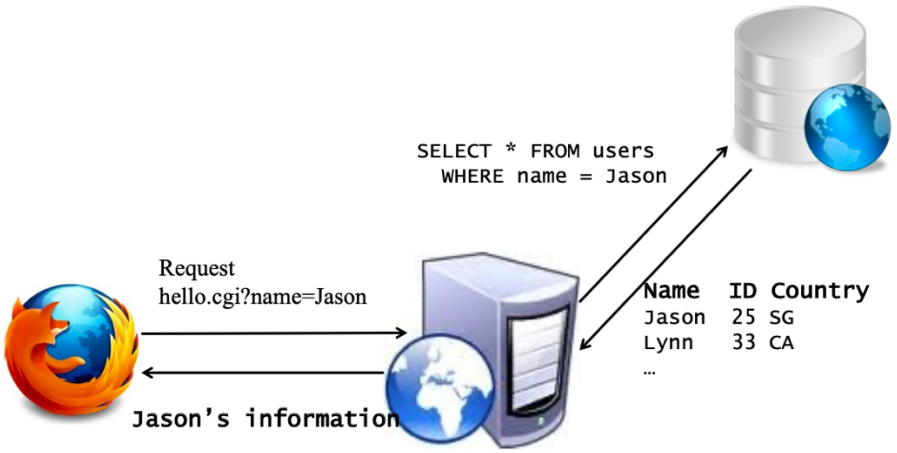
- Status line containing status code & reason phrase
  - HTTP/1.1 200 OK
  - HTTP/1.0 404 Not Found
- Response headers
  - Content-Type
    - text/html
  - Content-Length
    - 35
  - Set-Cookie
    - theme=light;
    - sessionToken=abc123;
    - Expires=Wed, 09 Jun 2021 10:18:14 GMT

Web Client and Server Components
Client-side Components

- Hypertext Markup Language (HTML): Webpage Content
- Cascading Style Sheets (CSS): Webpage Presentation
- Javascript: Webpage Behaviour – making pages interactive and responsive

Server-side Components

- Web server: nowadays a scripting language is typically used as well, e.g. PHP
- Database server



Understanding Javascript
Example of Javascript in HTML:
<script type="text/javascript"> document.write('Hello World!"); </script>

What can Javascript do in a browser?

- Write a variable or text into a HTML page
  - document.write("<h1>" + studentname + "</h1>")
- Read and change HTML elements
  - var doc = document.childNodes[0];
- React to events, such as when a page has finished loading or when a user clicks on a HTML element
  - <a href="someURL" onclick="alert('User just clicked me!')">
- Validate user data, such as form inputs
- Access cookies
  - var doccookie = document.cookie;
- Interact with the server, such as using AJAX (Asynchronous Javascript And XML)

Understanding Hypertext Preprocessor (PHP)
PHP is a widely used, free server scripting language for making dynamic web pages.

A sample PHP page:

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
?>
</body>
</html>
```

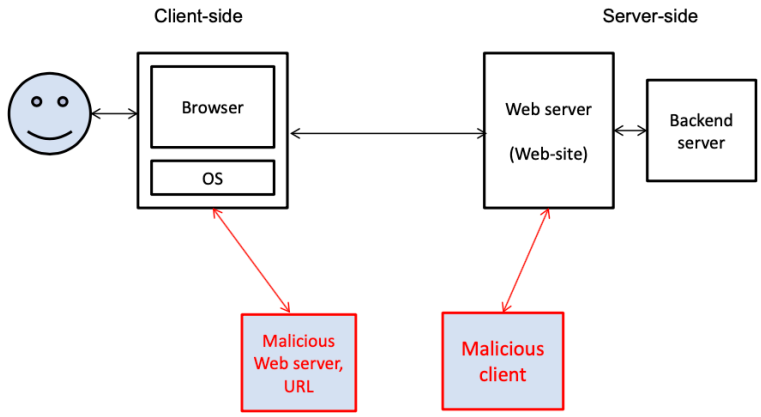
Security Issues and Threat Models

There are many reasons why Web Security is tough.

- Browsers run with the same privileges as the user – they can thus access the user's files
- At any instance, multiple servers with different domain names could provide the content. Access isolation among sites is thus required.
- Browsers support a rich command set and controls for content providers to render the content
- Browsers keep user's information and secrets. For example, some are stored in cookies.
- For enhanced functionality, many browsers support plugins, add-ons and extensions by third parties. The exact definitions and differences between these three may not be clear.
- Users can update content in the server, and more and more sensitive data is stored in the Web or on the cloud
- For the PC, the browser is becoming the main/super application. To some extent, the browser **IS** the OS.

Threat Model 1: Attackers as Another End Systems

In networking jargon, the computers that are connected to a computer network are sometimes referred to as **end systems**, because they sit as the edge of the network. End systems are the devices that provide information or services.



In Threat Model 1, the attackers are just another end systems. For example, it could be a malicious web server that lures the victim to visit it, or a malicious web client who has access to the targeted server.

Attacker Types in Threat Model 1

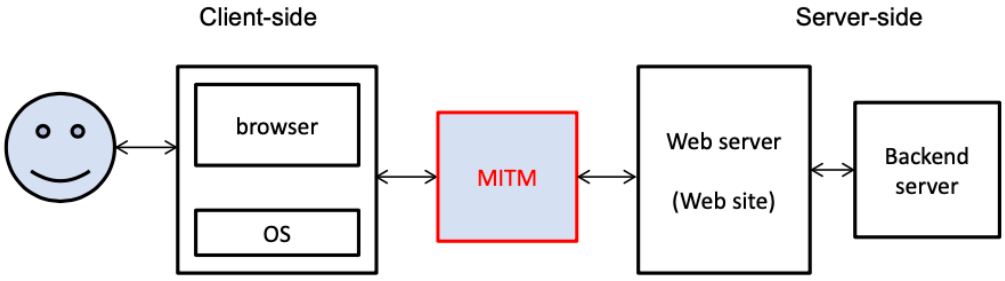
These are the different types of attackers:

1. Forum Poster
    a. The weakest attacker type who simply baits you to click something
        i. Link may direct you to a spoofed site or perform stored XSS (covered later)
    b. A user of an existing web app e.g. spreading on existing forums
    c. Does not register domains or host his own application content
2. Web Attacker
    a. Owns a valid domain and web server with an SSL certificate
    b. Entices a victim to visit his site
        i. "Click here to get a free iPad"
        ii. Or via an advertisement, thus no clicks needed
    c. Cannot intercept or read traffic for other sites
    d. It is also the most commonly-assumed attacker type

        i. It is quite effortless these days to get one's own domain and become a Web Attacker
3. Related-domain Attacker
    a. A web attacker who is able to host content in a **related domain** of the target web application
    b. It may be a sibling or a child domain of the target application.
        i. attacker.target.com
        ii. attacker.app.target.com
        iii. In an attempt to target app.target.com
4. Related-path Attacker
    a. A web attacker who is able to host an application on a **different path** than the target application, but **within the same origin**
        i. www.comp.nus.edu.sg/~attacker
        ii. In an attempt to target www.comp.nus.edu.sg/~target
    b. Very high level, difficult

Threat Model 2: Attackers as a Man-in-the-Middle

A man-in-the-middle attack is an attack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.



In Threat Model 2, the attacker is a Man-in-the-Middle (at the IP layer). For example, an attacker can be a malicious café-owner who provides the free WIFI services in our previous examples.

Attacker Types in Threat Model 2

These are the different types of attackers:

1. Passive Network Attacker
    a. An attacker who can passively eavesdrop on network traffic, but cannot manipulate or spoof traffic (Eve)
    b. Can additionally act as a web attacker
2. Active Network Attacker
    a. An attacker who can launch active attacks on a network (Mallory)
    b. Can additionally act as a web attacker
    c. This is the most powerful attacker type
    d. Yet, the attacker is generally considered to be incapable of presenting valid certificates for HTTPS sites that are not under his control.
        i. Unless the user himself clicks through i.e. accepts the invalid certificate
        ii. Or the attacker is a rouge CA, which we have touched upon before
        iii. Both conditions are generally hard to fulfil

Ultimately, it's still difficult to clearly classify web attacks, since many use a combination of various other attacks. We will now look at some of these web attacks, and the relevant and common protection mechanisms.

Attacks on the Secure Communication Channel (SSL/TLS)

As we know, HTTPS protocol stands for HTTP over TLS/SSL, where the latter includes Netscape SSL 2.0 in 1993 to TLS 1.3 in 2018.

It provides a data channel that has confidentiality, integrity and authenticity between two programs, providing security against a computationally-bounded "network attacker".

HTTPS works via:

- Ciphers negotiation
- Authenticated key exchange (AKE)
- Symmetric key encryption and MAC

Attack on a Secure Channel by a MITM

There are two conditions of a Man-in-the-Middle attack:

- The attacker is a MITM between the browser and the web server
- The attacker is able to sniff and spoof packets at the TCP/IP layers

If the connection is through HTTPS, the MITM is unable to compromise both confidentiality and authenticity **unless** the web user accepts a forged certificate or a rouge CA (which is the MITM in this context).

Yet, this might not always be the case as there exist vulnerabilities in the protocol or its implementation. Some examples are:

- FREAK Attack
    - Factoring RSA Export Keys is a security exploit of a cryptographic weakness in the SSL/TLS protocols long ago. There were limitations on exportable software to use only public key pairs with RSA moduli of 512 bits or less so that the NSA can crack it but not other organisations with lesser computing resources.
    - However, in the early 2010s, increases in computing power meant that anyone could break it. Combined with the ability of a MITM attack to manipulate the initial cipher suite negotiation between the endpoints in the connection and the fact that the finished hash only depended on the master secret, this meant that a MITM attack with only a modest amount of computation could break the security of any website that allowed the use of 512-bit export-grade keys.
    - Discovered in 2015, but existed since 1990s.
- Superfish Attack
    - It's a preinstalled MITM on Lenovo machines since 2014 which sits between the machine and the web server. The installation included a universal self-signed certificate authority, allowing the Superfish Visual Search software to intercept your HTTPS communications and inject advertisements via proxy re-encryption.
    - However, this CA had the same private key across laptops, allowing third-party eavesdroppers to intercept or modify HTTPS secure communications without triggering browser warnings.
    - Its HTTPS-decrypting and interception software is basically an SSL hijacker.
- Heartbleed Attack
    - Heartbleed is a security bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014.
    - It may be exploited regardless of whether the vulnerable OpenSSL instance is running as a TLS server or client. It results from improper input validation due to a missing bounds check in the implementation of the TLS Heartbeat Extension.

    - The Heartbeat Extension provides a way to test and keep alive secure communication links without the need to renegotiate the connection each time.
    - The vulnerability is classified as a buffer over-read, where more data can be read than should be allowed.
- Re-negotiation Attack
    - Covered in the section on HTTPS, under Tutorial 5.
    - Basically, there was a lack of continuity between communications before and after re-negotiation, allowing attackers to "combine" their communications with a victim's.
- BEAST Attack
    - The Browser Exploit Against SSL/TLS Attack attacked the fact that the Cipher Block Chaining mode of the AES used the last ciphertext block (which is visible) of one packet as the initial IV of the next packet.

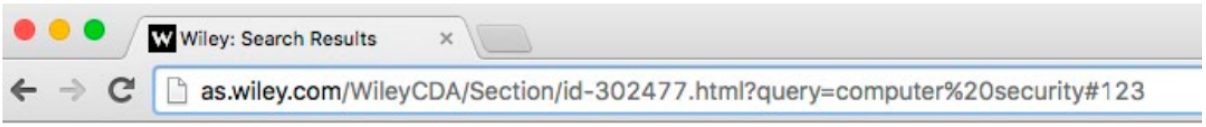
URL and Address Bar Insecurities (i.e. Mislead the User)

The Uniform Resource Locator (URL) consist of a few components:

- Scheme
- Authority (aka the hostname)
- Path
- Query
- Fragment

For example:
http://www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security#12
Scheme: http
Authority: www.wiley.com
Path: WileyCDA/Section/id-302477.html
Query (?): query=computer%20security
Fragment (#): 12

A browser usually helps with this by displaying the authority and everything else with different levels of intensity i.e. different font colour.



URL Confusion

Suppose that there is no clear visual distinction between the hostname and the path of a URL. The delimiter that separates the hostname and the path can be a character **in the hostname or path** itself. For example, in the previous example, the character is "/", which is also found in the path.

Thus, a malicious website whose hostname contains the targeted hostname followed by some character that looks like the delimiter "/" may confuse users:
www.wiley.com.lwiley.in/Section/id-302477.html

The actual domain is lwiley.in. Another example is nuslogin.789greeting.co.uk, where 7 supposedly looks like a "/".

For all these, however, the displayed different intensities can help the user spot the attack. But what if the address bar itself is spoofed?

Address Bar Spoofing
The address bar is an important component to protect, as it is the only indicator of what URL the page is actually rendering. What if it can be spoofed? An attacker could trick someone to visit malicious URL X, all the while making the user believe that the URL is Y. This may be possible with a poorly-designed browser.

An example would be how in the early design of browsers, a web page could render objects or pop-ups in an arbitrary location. This allowed a malicious page to overlay a spoofed address bar on top of the actual address bar.

Current versions of popular browsers have mechanisms to prevent this issue. However, a recent attack, CVE-2015-3830: Android Browser All Versions – Address Bar Spoofing Vulnerability, occurred.

Cookies and Same-Origin Policy
Note that the same-origin policy is not an attack, but a protection mechanism to protect cookies.

A HTTP cookie is a piece of textual data sent by a Web server and stored on the user's web browser while the user is browsing. It is sent in a HTTP response's "Set-Cookie" header field.

A cookie consists of a name-value pair, and can be used to indicate a user preference, shopping cart content, server-based session identifier, etc. Whenever a client revisits the site i.e. submit another HTTP request, the browser automatically sends in all "in-scope" cookies back to the server in its HTTP request's "Cookie" header.

What is "in-scope"? Cookies are only sent back to the same cookie origin, i.e. to the server that is the "origin" of the cookies. Scheme and protocol checking **may be** optional.

Viewing Cookies
On Firefox
- Right-click → View Page Info → Security → View Cookies; or
- Tools → Web developer → Developer toolbar → Storage

On Chrome
- chrome://settings/content/cookies

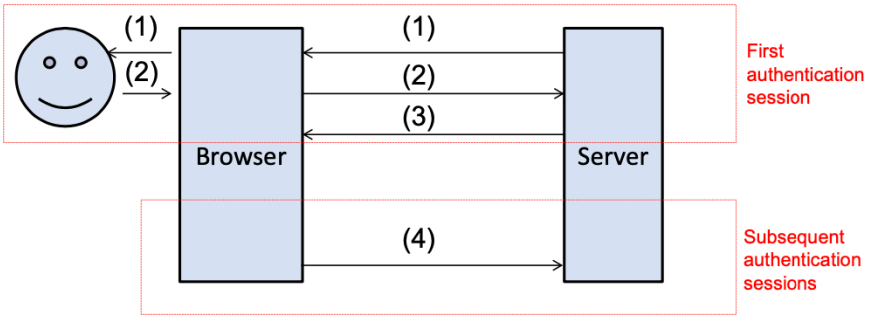
Usage
There are a few types of cookies, such as:
- Session cookie
  - Deleted after the browsing session ends
- Persistent cookie
  - Expires at a specific date or after a specific length of time
- Secure cookie
  - Can only be transmitted over HTTPS

The checking of scheme when doing the "same origin" check for cookies is optional, except for secure cookies that strictly require HTTPS.

Since HTTP is stateless (and thus HTTPS as well), there is a need to keep track of a web session. Cookies are thus commonly used to set and indicate a session ID. It is a better approach than attaching the session ID as a URL-encoded parameter in the HTTP request or as a form field, but it does have its own issues.

Token-based Authentication
To ease a web user's tedious task of repeated logins, many websites use "token-based" authentication. After a user A is authenticated e.g. entering the right username and password, the server sends a value t, known as the token, to A. In subsequent connections, whoever presents the token t is thus accepted as the authentic user A.

This token t typically has an expiry date, and it can identify a session, thus it's also called a session ID (SID). In web applications, a token is often stored as a cookie.



1. Authentication challenge e.g. asking for password
2. Authentication response that involves the user e.g. providing the password
3. Server sends a token t and browser keeps the token t.
4. Browser presents the token t with every HTTP request, and the server verifies it

We assume that the communication channel is secure – done over HTTPS (with server being authenticated) and the HTTPS being free from vulnerabilities.

Choice of Token
A token t needs to be random and sufficiently long. However, if the token t is a randomly chosen number, then the server would need a table to store all issued tokens, which can take up a lot of space.

To avoid storing the table, one could use:
- Insecure method
    - The cookie is some meaningful information concatenated with a predictable sequence number
    - Example: t = "alicetan:16/04/2015:128829"
    - Insecure as an attacker, who knows how the token is generated by observing their own token, can forge it
    - This is the weakness of security by obscurity
        - We assume that the attackers do not know the format, which is false
- Secure method
    - The cookie consists of two parts
        - Randomly chosen value or meaningful information like the expiry date
        - MAC computed with the server's secret key
    - Example: t = "alicetan:16/04/2015:adc8213kjd891067ad9993a"
    - Secure as it relies on the security of MAC

Both for methods, when the server finds out that the provided token t is not in the correct format or does not contain the correct content, the server rejects the token.

Scripts
A script that runs in a browser can access cookies. The scripts must then be limited in the cookies they are able to access. Due to security concerns, browsers employ the access control mechanism called Same-Origin Policy

Same-Origin Policy (SOP)
SOP states that a script in web page A (identified by its URL) can access cookies stored by another web page B (identified by its URL) only if A and B have the same origin. The origin is defined as the combination of protocol, hostname and port number.

Sounds safe and simple, but unfortunately there are complications.

Let us assume we have this URL: http://www.example.com. Then what we have is as such:

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.

There are many exceptions, resulting in confusion and this being error-prone. For example, unlike other browsers, the Microsoft IE does not include the port in the calculation of the origin, using the Security Zone in its place.

Cross Site Scripting (XSS) Attacks
There are two types of XSS Attacks:

1. **Reflected (Non-Persistent) XSS Attack**

In many sites, the client can enter a string into the browser, which is to be sent to the server. The server then responds with a HTML containing s, which is then rendered and displayed by the client's browser. If the string contains a script, then the page will actually execute the script!

Note that this won't work if the server performs HTML (entity) encoding, which replaces the special characters with safe versions of the character e.g. replaces "<" with < or %3c.

The steps are as follows:
1. The attacker tricks the user to click on a URL
    a. The URL contains the target website and a malicious script s
    b. The link could be sent via email, href as "click me", or is a link in a malicious window
2. The request is sent to the server
3. The server constructs a response HTML
    a. The server does not check the request carefully
    b. The response contains s
4. The browser renders the HTML page and runs the script s

A script may be benign, but a malicious one could deface the original webpage or steal the user's cookies. Due to the same-origin policy, as the script now comes from the target web server, it can access cookies previously sent by the web server.

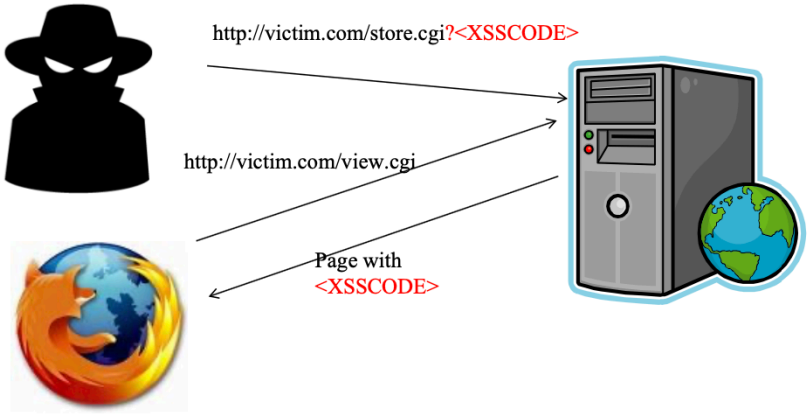
This is an example of privilege escalation – the malicious script from the attacker now has the privileges of the web server and can read the cookies. The attack exploits the client's trust of the server, as the browser believes that the injected script is from the server.

2. **Stored (Persistent) XSS Attack**

The script s is **stored** in the target web server. For example, it may be posted onto the forums, thus stored in a forum page. The attacker is a malicious forum poster.

An example would be the Samy XSS works on MySpace.com, where Samy became a friend of 1 million users in less than 20 hours.

This is more dangerous than reflected XSS attacks as the malicious script is rendered automatically without the need to lure target victims to a third-party website. The victim to script ratio is also many to one.



What is XSS in short?

It is a type of injection attack on web apps, whereby a forum poster or web attacker attacks another web user by causing the latter run a (malicious) script from the former in the execution context of a page from an involved web server, thus subverting the Same Origin Policy.

The attack works by exploiting the victim's trust of the involved server:

- In reflected XSS, the web server that **returns** a page reflecting the injected script
- In persistent XSS, the web server that **stores** a page containing the injected script

Defence Mechanisms

Most defence rely on mechanisms carried out on the server-side:

- Filter and remove any malicious script in a HTTP request while constructing its response page
- Filter and remove any malicious script in a user's post before it is saved into the forum database

Some example techniques are Script Filtering and Noscript Region. A Noscript Region is basically a region of a web page where JavaScript is not allowed to appear.

However, this is not fool-proof. To additionally detected reflected XSS attacks, some browsers employ a client-side detection mechanism e.g. XSS auditor.

Cross Site Request Forgery (CSRF) Attacks
There are also two types of CSRF attacks:

1. **Victim clicks on a URL**

This is also known as “sea-surf”, cross-site **reference** forgery or session riding. The attack goes as such:

- Suppose a client Alice is already authenticated by a target website such as www.bank.com and this site accepts an authentication token cookie
- The attacker Bob tricks Alice into clicking a URL of this site, which maliciously requests for a service. For example, transfer $1000 to Bob:
    - www.bank.com/transfer?account=Alice&amount=1000&to=Bob
- Alice’s cookie will also be automatically sent to the site, indicating that the request comes from the already-authenticated Alice
- Hence, the transaction will be carried out

2. **Victim doesn’t click on a URL**

A web attacker can also perform a CSRF attack without UI actions from the victim. An example would be:

- Again, suppose Alice is already authenticated by a target website www.bank.com and the site accepts an authentication token cookie
- Alice visits the attacker’s site, whose page contains the following:
    -
- Alice’s browser issues another HTTP request to obtain the image
- Alice’s cookie will also be automatically sent to the website
- Hence, the transaction will be carried out

What is CSRF in short?
It is a type of authorization attack on web apps, whereby a web attacker attacks a web user by issuing a forged request to a vulnerable web server ‘on behalf’ of the victim user.

The attacker disrupts the integrity of the target user’s session. This is, in a way, the reverse of XSS – it exploits the server’s trust of the client. The server believes that the request is from the client.

Defence Mechanisms
This is relatively easier to prevent compared to XSS. The SID/authentication-token cookie that is automatically sent by the browser is deemed as insufficient. The server must issue and require extra information i.e. anti-CSRF token. For example, the server may include a dynamic anti-CSRF token in its money transfer request page.

- The anti-CSRF token can then be included in a URL
    - www.bank.com/transfer?account=Alice&amount=1000&to=Bob&Token=xxk34n890ad7casdf897e324
- It is also possible to include the anti-CSRF token inside a HTTP request header or a hidden form field.

Other Web Attacks and Terminologies
Here is a list:

- Drive-by download
    - Unintended download of computer software from the Internet
        - May be authorised by user but without understanding the consequences e.g. downloads which install some unknown executable automatically
        - Any download that happens without a person’s knowledge, often a virus, spyware, malware or crimeware

    - May happen when visiting a website, opening an email attachment, clicking a link, or clicking on a deceptive pop-up window
- Web beacon / Web bug / Tracking bug / Tag / Page tag
    - A technique used on web pages and emails to unobtrusively, usually invisibly, allow checking that a user has accessed some content.
    - They are typically used by third parties to monitor the activity of users at a website for purposes of web analytics or page tagging.
    - May also be used for email tracking.
- Clickjacking (User Interface Redress Attack)
    - A malicious technique of tricking a user into clicking something different from what the user perceives, thus potentially revealing confidential information or allowing others to take control of their computer while clicking on seemingly innocuous objects, including web pages.
    - It is usually embedded code or script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function
- Click fraud
    - A type of fraud that occurs on the Internet in pay-per-click online advertising. As owners of sites that post these ads receive money based on how many visitors click on the ads, they may use a person, automated script or computer program that imitates a legitimate user of a web browser to click on such an ad without having an actual interest in the target of the ad's link.
    - This may even be in the form of employing low-wage workers to repeatedly click on each AdSense ad on their website, thereby generating money to be paid by the advertiser to the publisher and to Google.
- Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)
    - A challenge-response test used in computing to determine whether or not the user is human.
    - The most common type requires someone to correctly evaluate and enter a sequence of letters or numbers perceptible in a distorted image displayed on their screen.
    - Because the test is administered by a computer, in contrast to a standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.