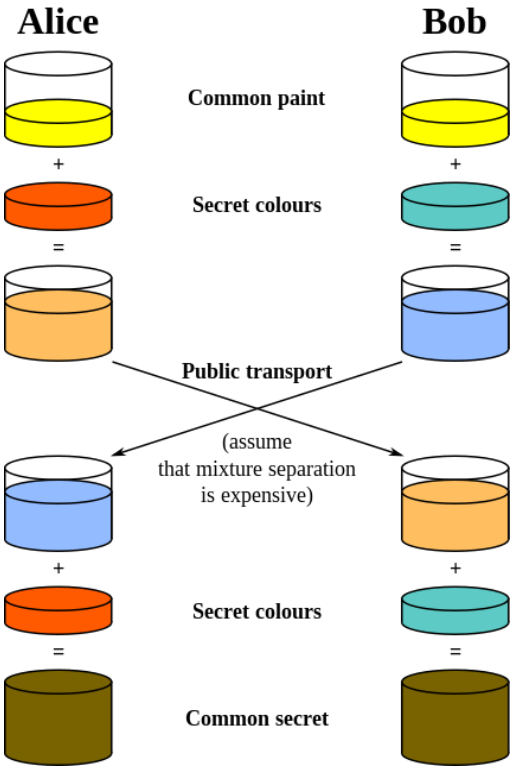
Mid-Sem Exam debrief (some questions explained…)

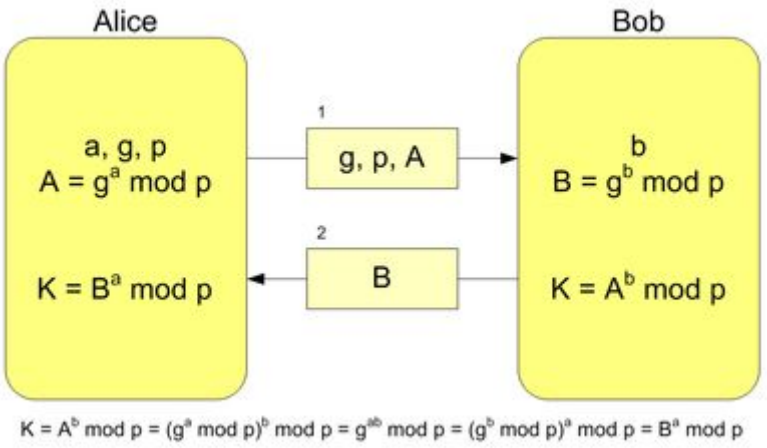
- Q5 - president with nuclear launch codes
  - Type 1/Type 2 Errors - what if you want to launch but you can't (because of the single point of failure of someone holding the pin code). Conversely, what if you don't want to launch and you do (because of a psychopathic president)
- Q10 - Merkle puzzle
  - You need a code that can be broken, but can't be broken too quickly. Work Factor of 5,000 - 500,000
  - Vignere + Caesar are too easy - RSA 64 bit is the only one within reason that can be cracked although If you crack 1 then you have the key, so you've cracked everything.
  - **There is no answer!**

In the exam when he mentions proof of liveness it is to proof that there is a person on the other end and not a replay attack (a "challenge and response" authentication).

- **Diffie Hellman Key Exchange (haven't we already done this?)**

Method of generating a shared secret between two people.
Here's a nice graphic representation... (got in before richard mentioned it)





Where:

- a, b : private key
- g : base
- p : huge prime number
- A,B : public key
- K : shared secret

Diffie Hellman does not give us **authentication** it only gives us **confidentiality** (replay attacks cannot be authenticated). With a numeric example of why this works is because (a^b)^c = (a^c)^b=a^bc, so if a is known and you tell someone a^b and they tell you a^c you cannot work out a^bc if you do not know the secret.

Check out the hall of fame for Dean Wunder's explanation on this.

- Solves the problem of sharing a secret between 2 parties at distance
- Someone can listen to everything, and still not understand anything

- Public knowledge:
  - Use 5 as the base
- Private knowledge
  - P1 chooses number 7
  - P2 chooses number 3
- P1 tells P2 $5^7 = 78125$
  - P2 takes 78125^3
- P2 tells P1 $5^3 = 125$
  - P1 takes 125^7
- The result will be the same
  - This result is known by P1 and P2 but no one else
  - So it can be used to encrypt messages now

- Reality:
  - pick a big prime number for base
  - big number for the index too
  - discrete log problem: it's hard to go backwards if you do this with big numbers
  - Use a publicly known value used as modulus to make the number smaller

    - Could use this to establish an RSA key or an AES key
  - Gives us confidentiality and integrity, not authentication
  - Breaking this:
    - man-in-the-middle
- Diffie-Hellman gives **Perfect Forward Secrecy** (PFS), also known as Forward Secrecy, is an encryption style known for producing temporary private key exchanges between clients and servers. For every individual session initiated by a user, a unique session key is generated. If one of these session keys is compromised, data from any other session will not be affected. Therefore, past sessions and the information within them are protected from any future attacks.

- Krak Des Chevaliers
  - Crusades era castle, held by the Knights Templar, had an outer ring and inner ring. Eventually fell when the Ottomans or whoever forged a letter from the head of the Templars telling them to surrender, and never fell due to weak defences
  - Also an example of defense in depth
  - Fell to social engineering rather than defence flaws

Cyber Literacy - Vulnerability

- Vulnerabilities - a potential weakness in something
- Software Bug - sometimes a vulnerability
- Exploits - attacks a vulnerability to compromise a system
  - Memory Corruption Attacks - the attacker changes what is in memory so that the program behaves differently, the most common type is a buffer overflow:
    - Buffer overflow: Attack on the stack, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.
  - How functions work in C: Functions freeze their information when a new function is called, the instruction pointer saves the location of the parent function (I think?), the stack grows
  - Format String error: Was a problem everywhere, until everyone realised and patched against it within a few months, and it wasn't an issue anymore, but is Starting to come back
    - %x attack - arbitrary read of information below our memory address
      - E.g. `printf("%x\n"); // read the next item on the stack`
    - %n attack - arbitrary write of information below our memory address
      - Value written is the number of bytes in the string so far
      - E.g `printf("1234%n\n") // put 4 in the next position on the stack`
      - When overwriting the return address of the function, it is essentially an arbitrary jump
    - To protect against format string attack
      - Don't pass a variable as the only argument into a printf

        - Provide all arguments that correspond to the format string you have written
    - People who don't take into account buffer overflows anymore are like the anti-vaxxers of security engineering
    - Printf - anything that's not properly formatted/specified just get printed literally.
        - printf("%s \n", "hello world"); - this is the "correct way" but no one does that, instead just write- printf("hello world \n");
        - Take an input example - name ← enter name
          printf(name);
        - → Enter name = "Richard%s"
          printf("Richard%s");
          C will look for an argument for the %s character, if there is no argument in the function then it will just print the next string from the stack
          → Enter "%x%x%x%x%x ….%x"
          Will print the next n addresses in hexadecimal - can get it to print stack information, this can then be made to print the canary if we know where it is on the stack. By doing this we can keep the canary there and overflow the buffer till the canary and then give the known value of the canary and continue overflowing the buffer without the canary realising there has been a buffer overflow.
          → "%n" - writes to memory as well!! - hacker used this to get root access. Can overwrite address in the stack
    - Swiss cheese analysis - sometimes the holes line up 😫

\\

- Shell code - piece of code to grant terminal console on remote device (ultimately used to escalate privileges
- NOP Sled - create a long list of No-Ops (0x90) so you can jump to the sled and then slide down to your malicious code i.e. add a bunch of NOPs to your malicious payload which will pad said payload so at any point, when the program jumps to ANY of the memory addresses of your NOPs, it will execute the NOPs and reach your malicious code.
    - There's more sophisticated ways to do that now that look like the program is doing something when its not. Flipping backwards and forwards, adding and subtracting a constant etc.
- Vulnerabilities stored on regulated database - CVN
- Responsible disclosure
    - Consult the Vendor first
    - Then escalate to CERT (or relevant authority)
    - Still up in the air

- Cold War thinking: It's more important to win/destroy the Russians than to have a habitable planet 1000 years down the line
    - Ozymandias: Poem excerpt
      "My name is Ozymandias, King of Kings;
      Look on my Works, ye Mighty, and despair!
      Nothing beside remains. Round the decay
      Of that colossal Wreck, boundless and bare

> The lone and level sands stretch far away”
> https://www.poetryfoundation.org/poems/46565/ozymandias

Canary - buffer overflow protection modifies the organization of data in the stackframe of a function call to include a "canary" value that, when overwritten, shows that a buffer preceding it in memory has overflowed into the canary’s address. With the canary it is not going to prevent a malicious buffer overflow. You can find where the canary is in the stack and just fill an array with the canary so that it does not get triggered and alert the computer that a buffer overflow has happened.

NOP sled is a technique used to circumvent stack randomisation in buffer overflow attacks. As stack randomisation and other runtime differences change where the program will jump, the attacker places a NOP sled in a big range of memory. This will slide the code execution down into the payload code, right next to the sled. No-operation is available in most architectures and it does nothing other than occupying memory and runtime.

Responsible disclosure -> Vendor -> CERT (e.g. Cert Australia)

Swiss Cheese Safety

- If you slice up a block of swisse cheese and realign, the holes don't normally line up and we g
- But every now and then, the holes line up and we are vulnerable

OWASP Top 10

- **iirc** he says that “we should know this”, so we should probably have the list here for 2017:
    - Injection, broken auth, sensitive data exposure, XML external entities, broken access control, security misconfiguration, XSS, insecure deserialization, using components with known vulnerabilities, insufficient logging and monitoring

T10 OWASP Top 10 Application Security Risks – 2017

A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Security Engineering - Assets

- Security is made to protect our assets
- Sometimes we protect the wrong assets
    - e.g. if you have a front door well protected but no protection on the glass windows or the back door

Identifying assets – what is that you're trying to protect

- It is easy to identify the wrong thing
    - Richard car-burglar and AIDS story...
        - § Don't protect your car over yourself...
        - § Not getting aids from a syringe > $5 wallets
    - Richard window smashing story...
        - § Are you protecting money or window? leave your window open.

Coke

- Pretend the asset is the secret recipe
- But the real asset is the reputation/brand
- People prefer to drink coke even though they prefer the taste of Pepsi

- It's important to identify the right things to protect
- It's easy to protect the wrong thing
- Uni's asset review
    - Machines + Computers were mainly identified but didn't considered...
    - Reputation also needs to be considered
    - Students + Staff
    - Private user data

Strategies for Asset Protection

- Multiple pairs of eyes - ask lots of people
- Standards and protocols tend to miss more than they catch
    - A good starting point - shouldn't be completely ignored
    - Examples of standards are what NIST recommends, Information Security Forum (ISF) came up with some best practice rules. None of these are mandatory per say just a good starting point.
- Don't think you've done everything
    - There is always a blind spot or new category to consider
- Regularly surveying the values of people involved in what you are protecting
- Develop a sensible plan - people suck at knowing what they want. Need to ask the right questions to tease the information
- Constantly re-evaluate current list of assets

Categorising Types of Assets:

- Tangible assets: Those that are easily given a value (e.g. jewellery)
- Intangible assets: These cannot be easily and objectively valued (Customer info, company secrets)

- Monetary + psychological/emotional costs
- Difficult != Don't do

Strategies for assigning values to assets:

- Survey what many people think
    - No single person/group should be solely evaluating the assets
    - Examples of the info. that should be gathered are as follows:
        - "How much money would you lose were this data centre to go down for 24 hours?"
        - "How much will you lose if your company is disconnected from the internet for 3 hours?"
- Examples:
    - In assessing the value of a park
    - Picasso

Diffie Hellman works to establish confidentiality and integrity but how do we get authentication? If we use public and private keys on their own we are susceptible to Man In The Middle attacks. So what can we do?

- We could use Public Key Infrastructure (PKI). It is like a passport in that it links the public key to a domain name which is certified by a 'trusted' group (signer - certificate authority). It needs to know the signers public key which is often installed in your computer automatically.
- This PKI system is not foolproof you can get variations of legitimate sites (Gooogle instead of Google for example) approved because the signers have an incentive to agree to sign things because they get money that way.
- From Threatpost article on weakness of PKI "Researchers said certificate abuse boils down to three types of weaknesses in the code signing PKI: inadequate client-side protections of certificates, publisher-side key mismanagement, and certificate authority-side verification failures." https://threatpost.com/assessing-weaknesses-in-public-key-infrastructure/128793/
- PKI can let malware through (Struxnet and Flame)
- Pretty Good Privacy (PGP)/Web of Trust is another way to deal with the problem of authentication.
- SSL / TLS diagram of it

PKI Weaknesses

- Single point of failure
    - CA can verify someone who is malicious or is impersonating another entity - e.g. InfoWorld article
    - A Root CA certifies a CA that then issues a fraudulent certificate
    - LetsEncrypt provides free SSL certificates. Lowers the barriers for individual web developers but also for attackers
- Padlock - can provide a false sense of security now that many sites (legitimate and phishing) use HTTPS

- Users can easily proceed if a browser presents an SSL (Secure Sockets Layer) warning for an invalid/expired certificate etc

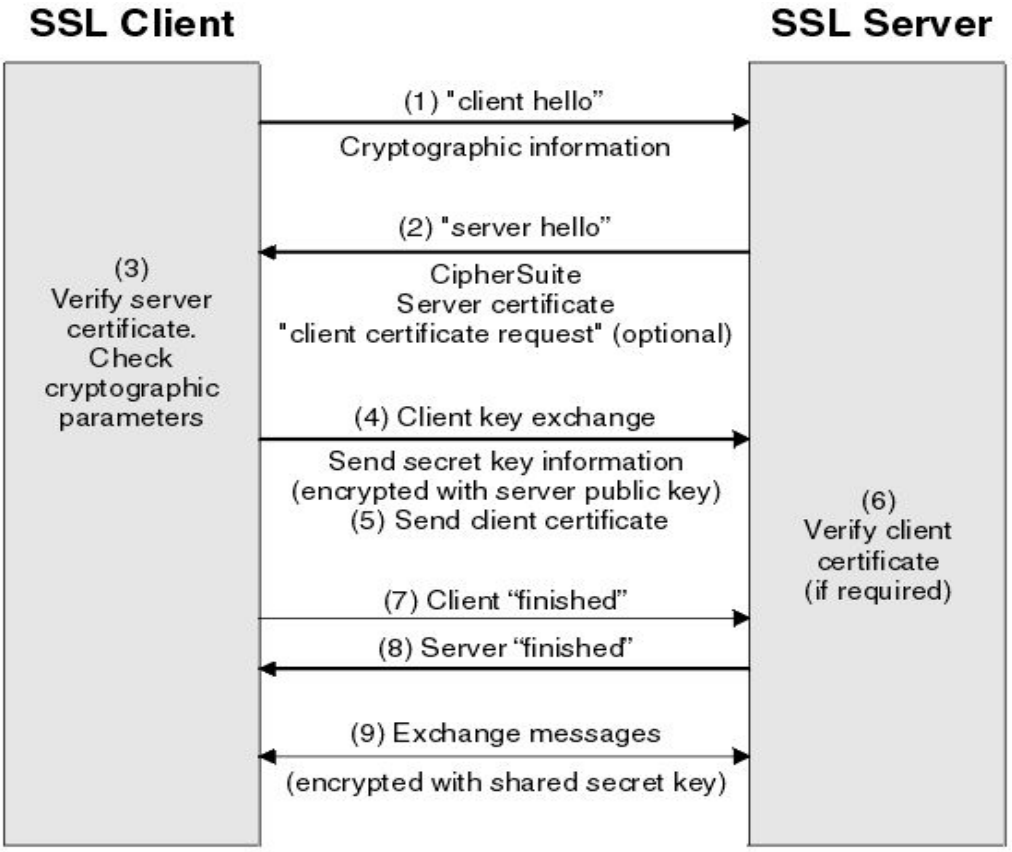
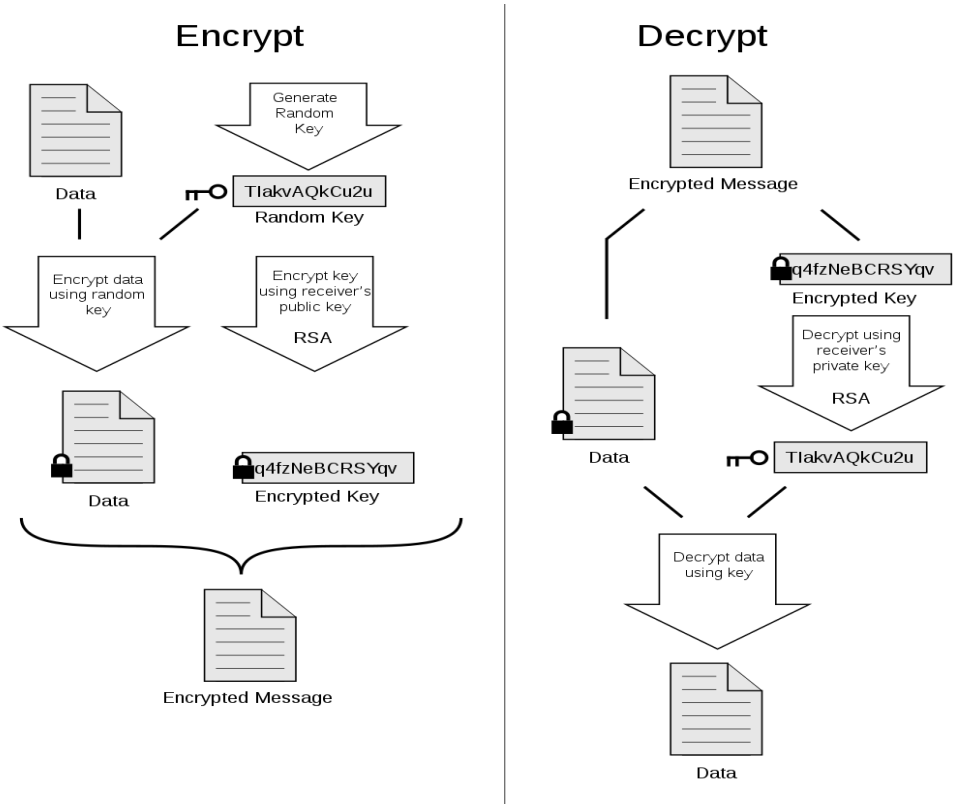


DIAGRAM OF PGP



A diagram of PGP

TLS (Transport Layer Security) Handbook Example:

1. A client contacts the server
2. The client and the server exchange information about the communications they intend to perform, such as the ciphers to use (SSL handshake)
3. The server transmits its certificate to the client
4. The client checks that it trusts the certification authority that issued the certificate. If it does not recognise the CA and does not get an override, the communication ends
5. The client checks for revocation information on the certificate. If the certificate is revoked or revocation information is unavailable, then the client might attempt to obtain an override. Implementations vary on how they deal with null or unreachable CRL information, but almost all will refuse to communicate with any entity using a revoked certificate
6. Both client and server send each other random data, which they can use to make calculations separately and then derive the same session keys. Three kinds of randomly generated data are sent from one side to another.
    - The “client random”: This is a random string of bytes that the client sends to the server
    - The “server random”: This is similar to the client random, except that the server sends it to the client
    - The “premaster secret”: This is yet another string of data. In some versions of the TLS handshake, the client generates this and sends it to the server encrypted with the public key; in other versions, the client and the server generate the premaster secret on their own, using agreed upon algorithm parameters to arrive at the same result.
7. Both parties generate 4 session keys using this data
8. All communications in the same conversation are encrypted with that set of keys

Bug Bounties

- Crowd-Sourced Bug Bounty Websites
    - Public: Hackerone, bugcrowd
    - Private: Synack
- Often have criteria of whats in/out of scope, as well as what kind of bugs they won’t accept. For example websites that they don’t want you touch
- Tips
    - Learn web apps
    - Use a wide scope → bigger net = more bugs
        - Stay in the scope
    - Look for software updates, or assets that have recently changed
    - Look for publicly disclosed reports → Can see prior bugs that have been found/exposed. If a bug has occurred once, there's a chance it will occur again
- Process:

    1. Find a program
    2. Review scope
    3. Find target via Recon
    4. Hit and find vulnerability
    5. Write a report
    6. Submit it
- Fuzzing
    - **fuzzing** is the usually automated process of finding hackable software bugs by randomly feeding different permutations of data into a target program until one of those permutations reveals a vulnerability
    - Automate process - a program that continually adds input
    - Some fuzzers are aware of input structure, and some even are aware of program structure
    - Fuzzers aren't precise, but can test a large amount of inputs
    - Ideally you would use both the Fuzzing and human-written tests, but that often doesn't happen.
    - Fuzzing software - afl (the way to go apparently)
- Mutation strategies - bit flips, byte flips, arithmetic, havoc (combination
- Use fuzzing to test your own software
- **Homework: Do the fuzzing tutorial**

Penetration Testing

- Why is it important?
    - It allows us to discover vulnerabilities in our system before attackers do
    - It can test you security controls(Firewalls, IPS,IDS)
    - Sometimes thinking like an attacker is the best way to expose weaknesses
    - Digital security in today's age is everything
- Authorized simulated attack on a computer system to evaluate security risks
    - Performed to identify strengths and weaknesses
- Steps of pentesting
    - 1. Recon
        - Intelligence is gathered
        - Pentester finds potential vulnerabilities in the system
    - 2. Planning
        - Plan how you are going to execute exploits
        - That payloads are going to be used
    - 3. Exploitation
        - Put gathered intel to use
        - Exploit vulnerabilities
    - 4. Post exploitation
        - Establishing persistence
- Certification
    - Offensive security

        - OSCP (Certified professional)
        - OSWE (Web expert)
        - OSEE (Exploitation Expert)
    - Kali Linux training

- Pentesting tools
    - Metasploit
        - Provides information about vulnerabilities and exploits of many different systems
        - Your antivirus will go nuts with the installation
    - Burp
        - Scans websites for vulnerabilities
    - Wireshark
        - Network protocol analyzer
        - See what's happening on the network
    - Kali
        - Linux OS containing many security tools
    - Nmap
        - DANGEROUS
        - Perform security scans
        - Network discovery and security auditing
            - Discover hosts and services on a computer network by sending packets and analyzing the responses
        - Finding open ports on remote machine
    - Gobuster
        - Brute force
            - URLs in websites
                - Directories and files
            - DNS subdomains
- CTF Websites
    - Pwnable.kr
    - Hackthebox.eu
    - Root-me.org
    - Overthewire.org