Computer Security Notes

Found at: benjaminshaw.uk

1 Basics

1.1 A Definition of Security

Confidentiality

Ensure that assets are accessed only by authorised parties.

Integrity

Assets can only be modified by authorised parties, or by authorised means.

Availability

Assets are only accessible to authorised parties at the appropriate times.

Accountability

Actions are traceable to those responsible

Authentication

User/data origin accurately identifiable

1.2 Security Countermeasures

Prevention

Stop security breaches via system design and defences

Detection

If a breach does occur, detect it.

Response

A plan utilised when a breach is detected.

1.3 Denial of Availability

A user will expect that services be available to them. A common attack is denying users this privilege. Denial of Service (DOS) attacks or malware are two common ways of attacking availability.

2 Cyber Security Essentials

2.1 Secure Configuration

Principles:

Devices on a network should be configured such that they minimise the number of inherent vulnerabilities.

Default settings can *often* be insecure, which includes default passwords.

Actions:

- Remove unnecessary user accounts, such as the *Admin* account found on Windows XP installs.
- Changing the default password
- Removal of unnecessary software
- Firewall software should regulate the incoming/outgoing connections on a device

2.2 Boundary Firewalls & Internet Gateways

Principles:

Devices should be protected against unauthorised access and disclosure.

Firewalls are the first line of defence and can stop attacks before they even reach the network.

Actions:

- Change default passwords
- Rules should be scrutinised before they are applied
- Unapproved services should be blocked by a rule
- Obsolete rules should be purged
- Firewall administration tools should not be accessible from outwith the network

2.3 Access Control and Privilege Management

Principles:

User accounts should have the minimum amount of privileges, with extended privileges awarded upon authorisation.

A compromised account with high levels of access can lead to a lot of damage.

Actions:

- Account creation should be subjected to an approval process
- Administration accounts should only be used for legitimate administration purposes and not activities that can be achieved with a standard account
- Elevated privilege accounts should require password changes periodically
- Users should be authenticated before being granted access to devices and applications
- Elevated accounts should be used when no longer required

2.4 Patch Management

Principles:

Remove unnecessary vulnerabilities by keeping software up-to-date.

Actions:

- Software should be kept up-to-date and fully licenced
- Out-of-date software removed
- Updates when made available should be installed in a timely manner

2.5 Malware Protection

Principles:

Internet-facing devices should make use of malware protection software that continuously monitor for known-malware instances.

Actions:

- Install anti-malware software
- Keep said software up-to-date
- Regularly scan all files
- Prevent connections to known malicious website

3 Network Security Threats

Types of Threat:

- **Interception**

  Unauthorised viewing of information
- **Modification**

  Unauthorised changing of information
- **Fabrication**

  Unauthorised creation of information
- **Interruption**

  Prevention of authorised access

3.1 Man-in-the-Middle

When communications between two devices is intercepted and changed by a third party intruder.

Man-in-the-middle is not necessarily an attack, as VPNs are man-in-the-middle by nature.

3.2 Denial of Service

When valid users are prevented from accessing a service.

3.2.1 SYN Flooding

When a large amount of *SYN requests* are sent to the victim with the intention of overloading them. SYN packets are the first packet sent in a TCP handshake.

The attacker would send many of these packets without acknowledging any of the replies to the point that the receiver can handle no more.

SYN flooding uses a lot of the attacker's bandwidth, and is also traceable if sent from their own IP. An ideal type of attack against a small target but tends not to be affective against larger targets.

SYN flooding in *conjunction with TCP source spoofing* makes it harder to trace the original attacker and ACKs will be directed to some other address.

3.2.2 Distributed Denial of Service

The same concept as a denial of service attack, but multiple systems flood the intended victim.

3.2.3 Smurfing

An example of *distributed* denial of service attack.

Exploits the *Internet Control Message Protocol* (ICMP) specification of *pings*, which allow for hosts to indicate that they are alive.

A forged ping packet is constructed with the source IP address as the victim's IP, and is sent to a *smurf amplifier*, subsequently that will swamp the target with replies.

A smurf amplifier is a poorly configured host that will not anticipate this attack, and will reply to the forged host.

3.3 Domain Name Service Attacks

Utilising the DNS system, where a DNS record will be maliciously altered to set up a man-in-the-middle attack, or prevent access to some URL.

4 Network Defences

4.1 Firewalls

Firewalls *divide* the untrusted exterior of the network and the trusted interior of the network; they are said to operate on the *Network* layer, or the 3rd layer, of the IP stack.

Firewalls either run on dedicated hardware or as a software package such as `iptables`. Hardware firewalls operate faster and act as a physical divider, whereas software is easier to deploy.

Firewalls operate upon a *set of rules*. These rules dictate if it allows or denies any given traffic.

4.1.1 Packet Filtering

The simplest task of a firewall, compares packet headers to the pre-defined rules.

However, this does not detect forged packets and requires a large and well-defined set of rules to function well.

4.1.2 Stateful Inspection

Remembers state between packets received, unlike a packet filter.

This allows it to filter attacks that utilise the scanning of many ports on a host, as it would detect the high number of packets from one source. With this piece of information, it could add to the firewall rules to block this specific host.

4.1.3 Application Proxy

A type of man-in-the-middle attack, as it screens messages received at the *Application* layer of the IP stack.

This allows for the firewall to block application requests, such as emails containing certain words or confidential information from a database leaving the network.

4.2 Intrusion Detection Systems

Designed to detect a potential intrusion *in progress*.

IDSs will monitor the network for behaviour similar to that of known attacks and raise the alert when it does so.

There is a fine line to walk as IDS needs to be sensitive to detect attacks, as they are not prone to being as blatant as possible; however, too many false alarms can soon prove to a be a liability.

4.3 Signature-based

Utilises *pattern matching*, where a pattern is provided in advance.

This method does not anticipate new types of attack but has a high level of accuracy which is effective for well-known and common types of attack.

4.4 Heuristic-based

The method builds a model of what *normal* behaviour looks like. This allows for possible anticipation of new types of attack.

However, it can take a long time to build this model and is susceptible to false positives.

5 Security Amongst Users

A large number of attacks occur due to user error, so it is in the best interests of organisations to make sure that users are able to identify attempts to fool them.

5.1 Phishing Attacks

A type of attack that *baits* the user into interacting with a malicious object. This can be done by impersonating a person or organisation that the user may trust beforehand.

This type of attack is usually delivered by email to try and get the user to go to an illegitimate link that will provide the user with malware.

Spear phishing is used against a small subset of users, but allows for the attacker to craft the attack in a way that is more likely to trick the user.

Whaling is an attempt to use a phishing attack against a financially wealthy victim.

Earmarks of a phishing attack are hyper links to an address that is designed to look like something else, e.g. *www.microsoftrewards.legitwebsite.com*, or emails that come with attachments with titles designed to lure the user to open it.

5.2 SSL/Malware Warnings

Secure Socket Layer is the technology that `HTTPS` is built upon, and allows for secure communications between two parties.

SSL allows for validation that the two parties are who they say they are and prevents a man-in-the-middle from reading packet contents, as they are encrypted.

Most modern web browsers will alert users when there are *certificate mismatches*, which can be an indication that the website is trying to mimic another's identity.

Many modern browsers will also look up the URL that the user is trying to visit, and cross-reference this with a blacklist of *known malicious websites*. If this check comes back as true, then a warning shall be displayed to users indicating that the site they are trying to visit has been knwon to distribute malicious content.

Whilst this can be effective against users visiting *new* websites, studies have shown that users are likely to ignore such warnings if they are trying to visit a website that they have been to before. This may be because users think that the blacklist check is at error, or that they can trust the source of the link that they are following.

Roughly *one and a half percent* of these warnings are false-positives.

5.3 Meaningful Warnings

It is obviously in the best interests of network administrators and developers to have warning messages that are understandable and effective when displayed to a user.

5.3.1 NEAT

- **Necessary**

  Is there a need for this user to see or decide upon the warning?

- **Explained**

  Is all information provided for the user to make an informed decision?

- **Actionable**

  Is there a set of steps that the user can take to make the correct decision?

- **Tested**

  Have you tested upon someone who would be an expected user, both in benign and malicious situations?

5.3.2 SPRUCE

- **Source**

  Ensure the source of who is asking for the decision is made clear

- **Process**

  Provide the user with actionable steps to make a good decision

- **Risk**

  Explain the risks of making the wrong decision to the user

- **Unique User Knowledge**

  Tell the user the information that *they* bring to the decision

- **Choices**

  List available options, *clearly recommending one*

- **Evidence**

  Highligh information the user should include or exclude whilst making the decision

6 Overflow Attacks

6.1 Vulnerabilities in C

Many of the original C functions are susceptible to attacks, as they look for the *null byte*, `'\0'`. The attacker can craft an input that doesn't contain the null byte, resulting in the vulnerable code to keep on reading input in. Such fundamentally vulnerable functions are `strcpy` and `scanf`.

6.2 Buffer Overflows

When executable code is injected into areas of memory earmarked for code execution, by overflowing areas that are, *most likely poorly*, designed to take input.

The malicious code will then be pushed into some other area and possibly change the flow of execution that the program follows.

6.2.1 Shell Injection

The aim of this type of attack is to *spawn a shell*, which would give the attacker general access to the victim's system.

6.3 Integer Overflows

When an attempt is made to store a value greater than the maximum value an integer can hold.

Poorly implemented code is susceptible to an attack using such a method as it may result in length checks failing to terminate.

6.4 Formatted String Exploits

If an attacker is able to ascertain what *format* a program is expecting to receive, they can craft some input that will directly load malicious code into memory.

6.5 Defences Against Overflow Attacks

6.5.1 Stack Canaries

Named after the canaries used in mines to detect carbon monoxide leaks in deep mines, they are used to detect when an overflow has caused some data within the stack to be overwritten.

The *stack canary* will be placed just before a return pointer. Before execution, the canary will be checked if it still present before the pointer, as to overwrite the pointer, the canary must be overwritten too.

If the canary is not present, the code will not be executed.

Canaries can hold random values calculated at the start of the process, or can hold termination values such as the `NULL` byte.

6.5.2 Make Stack and Heap Non-Executable

Prevention of overflowing code being executed, as the range it is within is not executed. However, the code can still be executed by using some of the *system calls* found within `libc`.

6.5.3 Address Space Layout Randomisation

Standard libraries are placed in randomly allocated, non-standard locations. This prevents attackers from directly pointing to where a system library call may usually be.

7 Authentication

To verify a fact about an entity before it can be allowed to execute some action. This can be achieved by requiring something they *know*, *are* or *have*.

7.1 Multifactor Authentication

An authentication process that requires more than one factor; examples being the *chip* and *pin* within your bank card.

7.2 Passwords

The most popular method of authentication, a *password* is a character string known to the user and system. They are easy to store, enter and require no special equipment whilst being easily scalable.

7.3 Password Entropy

The problem is different passwords have differing levels of *safety*.

A bad password would be one that has low *entropy*, or predictable in other words. It is commonly found that users use a common string, such as `'123456789'`, or an easily identifiable trait about their self, such as their name or date of birth.

A good password contains a wider range of characters, such as special characters and mixed cases.

A way to combat this is utilising *machine generated passwords*. A well made program can create passwords that are much harder to guess, but tend to be harder for a human to remember. Algorithms have been created to create passwords that are *pronounceable*, increasing the likelihood of a human being able to remember it.

7.4 Protecting Passwords

Passwords are rendered useless if an attacker is able to compromise the server and learn it that way. There are a number of mechanisms designed to keep passwords safe in the instance that the server has been compromised.

7.4.1 Hashing

A *good* hashing function translates some input to some unique output in a way that is hard to determine the underlying function itself.

This means that the server only stores *hashes*, instead of the plain-text passwords. Even if the hashes were obtained, they would be meaningless to the attacker.

A method called *salting* is used as additional input to further increase the security of hashing. This means that for one string as input but a set of different *salts*, the output would be different for each salt.

This can serve as a protection mechanism against *brute force attacks*. If a server is compromised, the attacker could continuously create a number of passwords with the hashing function to figure out the mapping between inputs and outputs. Salting makes this far more difficult to break the hashing function as it would need to be done for *each* salt alongside the hashing function.

7.4.2 Lockout

Another brute force attack is to utilise make a massive number of guesses. This takes advantage of the fact that many user-defined passwords are predictable. An attacker can guess many times against an unsecured authentication system and correctly guess the password in a minuscule amount of time if the password has a low enough entropy.

The idea of a *lockout* is to limit the amount guesses that the attacker has before they are prevented from another attempt, or requiring another factor that they may not have.

7.5 Keys

Accompanies the concept of using something you *have* as a means of authentication. Like having a key to unlock a physical doors, keys are also used within digital authentication.

7.5.1 Key Fob

A *key fob* has a securely generated number that must be provided alongside the password as a means of two factor authentication.

7.5.2 Public/Private Pair

A pair of keys, one of which is *private*, the other which is *public*. A message that one locks and the other unlocks, and vice versa.

7.6 Biometric & Behaviour

To accommodate the idea of something you *are* as a method of authentication.

7.6.1 Fingerprint Readers

Fingerprints are *nearly unique* to each person, but can sometimes be difficult to read or, more akin to Cold War antics, can be destroyed. They can also never be changed, which is severely detrimental in the event that they are somehow obtained.

7.6.2 User Behaviour

A means of *continuous authentication*, a user's behaviour is monitored. A user's typing patterns is something that is hard to mimic, but this can suffer from real world events such as the user being injured.

8 Access Control

Governs whether or not some user, (*the principal, or subject*) is allowed access to a resources.

Information flow control is a similar concept, but instead scrutinises if an object is allowed *to be known to* a principal.

8.1 Access Operations

Access modes determine that way that an object can be accessed, with *rights* being a *combination of such modes*.

8.1.1 BLP: Bell-LaPadula Model

A model that enforces confidentiality, it has two access modes: *to observe*, that is examine an object, and *to alter*, that is to change an object.

BLP has the following access rights:

	Observe	Alter
Execute		
Read	✓	
Append		✓
Write	✓	✓

8.1.2 Mechanisms for Access Control

Rights are commonly defined by the *access control matrix* that represents *who* has *which* access to *what*.

There is a set S of objects, a set A of operations that create the matrix M.

An example matrix would be:

	file1.txt	code.sh
User1	{}	{ `read` }
User2	{ `read, write, execute`}	{ `read, execute` }

8.1.3 UNIX

`UNIX` has three access rights: `r` for read, `w` for write, and `e` for execute.

There are 3 subjects: the *owner*, the *group* and the *world*.

An example of `UNIX` representing permissions for some file would be:

`rwxr-----`

The first three characters, `rwx`, means that the owner can read, write and execute the file. The second, `r--`, means that the other members of the group can read the file. The final three, `---`, means that nobody else has any access to the object.

8.2 Discretionary and Mandatory AC

Discretionary access control means that the *owner* of a resource dictates who may access it, set on a case-by-case basis.

Mandatory access control is *system-wide* as per a uniform policy.

8.3 Multi-Level Security

A system that has an order of levels, e.g.

`unclassified` $\leq$ `confidential` $\leq$ `secret` $\leq$ `top secret`

This can express policies meaning that a subject with a slight elevated level of privilege cannot write to an object that has a lower amount required to access it, as this may mean that more privileged information is revealed to those with lower privileges.

But a more flexible policy may be required.

8.3.1 Security Lattices

A method of defining a policy of which users can read and write to what levels in a system that utilises multi-level security.

For example, for two subjects `a` and `b`, and `a` $\leq$ `b`, we would say that b **dominates** a: `system low` is the bottom, *dominated* by all others; `system high` is the top, *dominating* all others.

Say we have two objects that are at differing levels; there is a *minimum* security level for a *subject to access* both.

Now, say we have two subjects that are at differing levels. There is a *maximum* security level for an *object to be* readable by both.

9 Security on the Web

The idea is that web-side applications should provide the same guarantee of security as of that of a standalone application.

Visiting a website should not be able to gain local file access, or infect your machine with malware. *Sandboxing* and *privilege separation* act as defences against this.

Nor should one site being accessed compromise connections with other sites; achieved by a *same-origin policy*, isolating one accessed site from others.

9.1 Injection Attacks

When malicious information is sent to an *interpreter* that can be executed; subsequently causing damage to the host, or providing information that should not be accessible to the attacker.

9.1.1 Command Injection

When a script used by the server reads specially crafted input that is then executed as a system command on the victim's side. This can occur when *data* and *code* share the same channel.

For example, a script may read some input from the user then treats it as directly executable code, which is obviously susceptible to abuse.

Input *validation* and *escaping* serve as defences against this type of injection.

9.1.2 SQL Injection

The *Structured Query Language* is the most common language utilised by databases, hence its syntax is well known and used by many, many hosts. A poorly secured host that makes use of a database can be severely damaged if SQL code is injected.

Poorly structured code may take user input and directly feed it to its internal SQL interpreter. The interpreter will execute the code that it is given, which may include more code than the developer intended.

For example, the website may have a field asking for some text input. If this input is not validated, it could include a command for SQL to delete or return some records, which would will be executed when it is passed along to the interpreter.

10 Session Hijacking

10.1 Cookies and HTTP

HTTP is a *stateless* system, but can utilise so that a state can be maintained between client and server. There can be difficulties however, as the server may actually be talking to someone who the cookie does not belong to, if the necessary precautions aren't taken.

A unique identifier between the client and the server can be stolen by an attacker; subsequently allowing the attacker to masquerade as the original client to the server.

There are safeguards against this type of theft.

First of all, cookies *tokens* should not be predictable, and tokens should be refreshed periodically.

10.2 XSS: Cross-Site Scripting

Many websites make use of client-side scripting languages, such as *JavaScript* as it provides more power to web developers than the markdown languages such as HTML and CSS.

Cross-site scripting is a type of injection attack, where malicious code is inserted into websites that would otherwise be trusted. These sites can then start serving malicious code to unsuspecting users. As the browser on the user's end thinks that the website is trusted, it has no way of knowing that some of the script code being sent from the website is malicious and executes it regardless. This can possibly give the attacker access to session tokens and cookies that belong to other users.

10.2.1 Reflected XSS

Where the malicious script is *reflected* off the web server and then delivered over this *presumed to be trusted* route to the victim. The script can be hidden within an error request or search result for example.

10.3 CSRF: Cross-Site Request Forgery

When a user is forced to execute unwanted actions on a web application in which they are currently authenticated, attacking requests to change-state rather than stealing data.

This type tries to take advantage of servers that have predictable structures.

The attacker will craft a URL, and then trick the victim to make a request to the vulnerable server. When the victim accesses the malicious URL, the attacker's website will make the victim's browser access the URL that will execute the unwanted action.

CSRF attacks can be prevented by verifying the origin of the request, and not one that has been referred by another website.

11 Secure Communications

Confidentiality: Information shouldn't be revealed to unauthorised entities.

Authentication: To know with certainty the entity that you are communicating with.

Anonymity: The identity of the author of an action should not be revealed.

Non-Repudiation: The author of an action should not be able to deny doing so

Integrity: Data should not be altered in an unauthorised manner during creation or transmission.

Unlinkability: An attacker should not be able to see which services have been provided to a user.

11.1 Cryptography Primitives

11.1.1 LFSR: Linear Feedback Shift Register

A *shift register* that takes its input as the output of the previous sate.

For example, with a 4 bit stream `1101`, the last two bits would be taken, `01`. We would then `XOR` these two bits, which would produce `1`. This is then placed at the front of the register, shifting the bits currently in the register to the right.

```
0 XOR 1 = 1
-----------
 1 -> 1101
 1110 -> 1   # this bit is popped
             off of the register
```

11.1.2 CSS: Content Scrambling System

Makes use of *LFSR* with a key size of 40 bits, used on *DVD-Video*.

Initialisation:

A 17-bit LSFR is initialised as the concatenation of 1 and the first 16 bits of the key.

A second, 25-bit LSFR is initialised as the concatenation of 1 and the remaining 24 bits of the key,

Action:

We take the sum of the output of two LSFRs plus any remainder from the previous, modulo 256. This produces a byte of output.

Flaws:

As the video encoding used in DVD-Video is *MPEG-2*, of which the first 290 bytes are plain text. This means that we know the first 20 bytes of the keystream.

Through brute force on the 17-bit LSFR, of which there are 2^{17} possibilities, we can deduce the output of the 25-bit LSFR using subtraction.

11.2 Perfect Secrecy

A cipher satisfies *perfect secrecy* if, for all messages of the same length, for all encrypted messages:

$$|Pr(E(k, m_1) = c) - Pr(E(k, m_2) = c)| \leq \epsilon$$

In plain English, absolutely no information about the unencrypted message will be revealed by the encrypted message.

11.3 Symmetric Ciphers

For some message, you have an encryption algorithm, E, and a decryption algorithm, D, such that:

K is some secret key known between the two parties
C is the encrypted message M is the unencrypted message

$$E : K \times M \to C$$
$$D : K \times C \to M$$

Examples include one-time pads, `DES` and `AES`.

11.3.1 OTP: One-Time Pad

Assumes that the message, encrypted message, and key are all of the same length.

Either the message or encrypted message, and the key, will be `XOR`'d to produce the desired output.

Encryption

```
k | 0 1 1 0 1 0 0 1
m | 1 0 0 0 1 0 1 1
-   - - - - - - - -
c | 1 1 1 0 0 0 1 0
```

Decryption

```
k | 0 1 1 0 1 0 0 1
c | 1 1 1 0 0 0 1 0
-   - - - - - - - -
m | 1 0 0 0 1 0 1 1
```

Drawbacks of the OTP is that they key *must* be the same length as that of the message, and that OTPs do not guarantee *true* randomness.

11.3.2 Stream Cipher

Similar to the One-Time Pad, but instead of using a really random key, a *psuedo-random* key is used. That is, a key that *looks* random, but is not random in reality.

The stream cipher functions in a way similar to that of the one-time pad, but can utilise a more convenient key which can differ in length from that of the message.

RC4

A stream cipher used by *HTTPS* and *WEP*.

RC4 consists of two parts: the *Key-Scheduling Algorithm*, and the *Pseudo-Random Generation Algorithm*.

Key-Scheduling Algorithm:

The *KSA* makes use of a *secret key* of a variable length; using it in conjunction with an ordered array of 256 elements, S. The aim is to make the array *look* like it has been randomly ordered. After this has been done, the algorithm moves onto the next step.

Psuedo-Random Generation Algorithm:

The *PRGA* increments over the message and outputs one byte at a time.

```
i := 0; j := 0;
do{
  i := (i + 1) mod 256;
  j := (j + S[i]) mod 256;
  swap(S[i], S[j]);
  K := S[(S[i] + S[j]) mod 256];
  output K;
}while(required);
```

So, we have an element of S, found at index i, which is swapped with another element of S, at index j. These two elements are summed, with the result *modulo 256* which gives us the next element of the *resulting encrypted array*. At each iteration, we recalculate i and j:

$$i = (i + 1) \bmod 256$$

$$j = (S[i] + S[j]) \bmod 256$$

The number of iterations is equal to the length of the *key*.

11.3.3 Block Ciphers

A *deterministic algorithm* that operate on fixed-length *blocks* of bits.

The block cipher will take some size bit of plain text, and output the same size of cipher text. A *key* is used to determine the exact transformation between plain and cipher text.

DES: Data Encryption Standard

An primitive version of the block cipher that found widespread use after being developed in the 1970s.

DES takes a block of size 64 bits from the plaintext, and uses a key of size 64 bits[1]. The output generated is of size 64 bits.

First, the plain text block is taken in and its order of contents changed.

Second, this transformed plaintext is fed through 16 *submodules*; each submodule takes in a different permutation of size 48 bits, from the key.

Third, each submodule has the same combination of operations that permute its input with the given key, which is then `XOR`'d with the key and then fed to the next module.

Finally, the ciphertext is transformed once again, in a fashion similar to the first step, and produces the ciphertext output.

It is the same process to decrypt the ciphertext if in possession of the key. Providing DES with the ciphertext and they key will produce the plaintext as output.

However, DES is susceptible to brute force attacks. An *exhaustive search*, over the key space, can be completed in 2^{56} time.

AES: Advanced Encryption Standard

Intended as the successor to DES.

Makes use of a block size of 128 bits, and the key size is either 128, 192 or 256 bits.

11.3.4 Using Block Ciphers

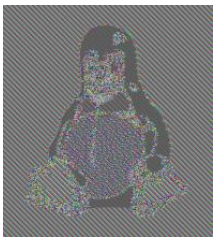
Block ciphers offer different *modes of operation*, which define the fundamental behaviour of the block cipher.

ECB: Electronic Code Book

The message M is divided into blocks, with each block being encrypted with key K separately. No interaction takes place between each block being encrypted.

The main issue with this method is that it doesn't hide *patterns* well at all, as identical plaintext blocks produce identical ciphertext blocks.

Before encryption



After encryption using ECB

[1] DES only makes use of 56 bits of the key, the remaining 8 bits are used to check parity (a check bit)

CBC: Cipher Block Chaining

Introduces the *initialisation vector*.

The first block and the IV are `XOR`'d, then fed to the block cipher with a key K, producing the first block of the ciphertext.

This block of ciphertext is then `XOR`'d with the next block of plain text, and once again fed to the block cipher. This continue is represented until the final block of cipher text is created.

The downside of this is that encryption must be sequential.

CTR: Counter

Turns a block cipher *into a stream cipher*.

CTR makes use of a function that produces *successive values* which are then fed to the cipher, `XOR`'d with the plaintext which produces the ciphertext.

11.4 Asymmetric Ciphers

For some message, you have a key generation algorithm, G, as well as encryption and decryption algorithms, E and D respectively.

K is some secret key known between the two parties
C is the encrypted message M is the unencrypted message

$$E : K \times M \to C$$
$$D : K \times C \to M$$
$$G : K \times K$$

The decryption key is secret, or *private*, whereas the encryption key is *public*.

An example would be `RSA`.

There is also use for *digital signatures*, which is the production of another key, the *verification key*, which is know to everyone but differs from the signing key. This can deduct whether or not some message has been encrypted under some given key.

11.4.1 Establishing a Secret Key

TTP: Trusted Third Party

Users establish their key with a third party that they both trust.

Lets say we have two parties, A and B with the trusted third party S.

A sends their key, encrypted under K_{AS} to B.

B receives the message, attaches their own key encrypted under K_{BS} and sends it to S.

S creates the key K_{AB}, sends it to A under the key K_{AS}. Also within this message is they key required for B, under the key K_{BS}, which A forwards to B.

Public-Private Pair

Lets say A wants to send a message to B

B sends some key that A will encrypt the message under.

A encrypts the message using that key, and then sends it to B.

B receives the message and unlocks it with a decryption key - known *only* to him.

D-H: Diffie-Hellman Protocol

First, we must understand we must understand two concepts to use this protocol.

The *discrete logarithm*, and the *primitive root modulo*.

Primitive Root Modulo
We say a number g is primitive root modulo n if every *coprime* of n is congruent to g modulo n.

Discrete Logarithm
For an equation of the form $b^k = g$, we say k is a discrete logarithm.

So, A and B agree to use a modulus p, and base g (which is a primitive root modulo of p).

A then chooses a secret integer, a, and sends B:

$$A = g^a \bmod p$$

B then chooses a secret integer, b and sends A:

$$B = g^b \bmod 23$$

A and B compute s with the message they were sent.

$$\text{A computes: } s = B^a \bmod p$$
$$\text{B computes: } s = A^b \bmod p$$

A and B will have both arrived at the same value s under mod p.

11.5 Hashes

An algorithm H, that takes a message M and produces a unique hash value D, such that:

$$H : M \to D$$

Hashing algorithms should be *pre-image resistant*, where it should be impossible to take the output of the hashing function and determine the original message fed to the function. There should also be *collision resistance*, where it should be extremely difficult to have two different messages produce the same hash.

Hashing function examples would be `MD5` and `SHA-1`.

11.6 OWF: One-Way Functions

A function f is one-way if there is no efficient algorithm that can compute the function's domain given its co-domain.

The multiplication of two different primes is an OWF, but the successor function is not an OWF.

11.7 CRF: Collision-Resistant Functions

A function f is collision resistant if there is no efficient algorithm that can find two messages, m_1, m_2 where $f(m_1) = f(m_2)$.

11.8 Hashing Function

A *cryptographic hash function* $H : M \to T$ must satisfy four properties:

- $|M| > |T|$
- easy to compute has for any given message
- hard to retrieve a message from its hashed value (OWF)
- hard to find two different messages with the same hash value (CRF)

Hashing functions allow for participants to commit to some *hashed* value without revealing the original message, useful for electronic voting. Also useful for verifying integrity of a file or password.

11.9 MAC: Message Authentication Codes

A short piece of information that is used to *authenticate* a message.

A MAC function take a key and a message as input, and outputs a *MAC tag*. The value protects both *integrity* and *authenticity* of the message, as any changes to the content are detected.

A Representing the Stack

A.1 Register Values

`%ebp`	Base
`%esp`	Stack Frame
`%eip`	Instruction
`%eax`	Return Value

A.2 Example

Code:

```
main{
    f(arg1, arg2);
    b = a;
}

f(arg1, arg2){
    return arg1 + arg2;
}
```

Process:

1. Push arguments onto stack (in reverse)
2. Push return address
3. Jump to function address
4. Push old frame pointer onto the stack: `%ebp`
5. Set frame pointer to where the end of the stack is: `%esp`
6. Push local variables onto the stack
7. Reset previous stack frame
8. Jump back to return address