CO331 - Network and Web Security (60015)

Week 1

Vulnerabilities

We define **vulnerabilities** as bugs or design flaws in software that can be exploited by attackers to compromise computers. These are taken advantage of by **exploits**, which are pieces of software. If it is unknown to the software vendor, it is referred to as a **zero day** (has not been disclosed to the public domain).

Advisories are used to publicly disclose new vulnerabilities, issued by vendors or security companies. These are important for developers, sysadmins as well as regular users of the software, in order to keep up to date or patch systems.

The **vulnerability reports** often vary in format. Bugs and systems can differ from each other, as well as researchers putting in varying levels of effort. Generally, the key information consists of the affected systems, descriptions, impact, proof of concept code, as well as proposed fixes.

There are a number of approaches for when vulnerabilities are discovered;

- **non-disclosure** keep the vulnerability secret

  This is preferred by vendors who choose not to use resources to fix bugs (based on 'security by obscurity'), or by parties intending to exploit it.

  An issue with hoarding vulnerabilities is the accidental release. For example, *WannaCry* used two exploits hoarded by the NSA - if this was disclosed, many more systems may have been patched.

- **responsible disclosure** affected vendor decides when and what to release

  This approach is preferred by software vendors, motivated by the idea that end users will not develop their own fixes. However this can lead to a long duration between a discovery and fix.

- **full disclosure** make details public

  Eliminates any asymmetric information advantage attackers may have. This method is preferred by security researchers, as well as the open source community. However, it may affect users to attacks.

  The current approach, spearheaded by *Google Project Zero*, is to give a window of time to vendors to fix vulnerabilities before it is publicly disclosed.

Malware

Malicious software can be characterised by infection vector;

- **virus** malicious code copying into existing programs
- **worm** replicates program over network or removable devices
- **trojan / spoofed software** provides (or pretends to) useful service to act legitimate
- **drive-by download** code executed by visiting malicious website

Another way to characterise malware is by purpose (malware often has multiple of these working together);

- **rootkit** — strongest, works at OS level (can hide itself)
- **backdoor** — allows attackers to connect over network
- **RAT (remote access tool)** — remote control
- **botnet** — recruit machine into botnet
- **keylogger** — logs keystrokes
- **spyware** — steals sensitive documents
- **ransomware** — blocks access to machine or data until ransom is paid
- **cryptominer** — uses system resources to mine cryptocurrency
- **adware** — displays advertisements

Malware can exist in several formats;

- injected code added to a legitimate program
- library loaded by a legitimate program
- scripts run by application (such as macros in *Microsoft Office*)
- standalone executable run by the user
- code loaded in volatile memory (fileless malware) - without a file, detection can be difficult

Viruses can propagate in a number of ways, either by the attacker in the case of self-replication, or drive-by downloads, or installed by the user, either through social engineering or compromised certificates (in fake software updates).

A virus can have varying privileges, either from the lowest level (in a rootkit, where it owns the machine), or have user privileges which can do limited damage.

APTs (Advanced Persistent Threats) are used to reach high-value victims. These attacks are specific to the victim, often driven by a human. Decisions are made, depending on the specific configurations, and can involve compromising intermediate systems to reach the victim. Detection is avoided, with the use of rootkits to hide presence, as well as large datasets being exfiltrated over a long period of time. Avoiding detection is important as these attacks are often done over a long period of time, waiting for information to enter the system, as well as retaining access for later use.

On the other hand, **botnets** are generic attacks, which aim to infect as many machines as possible. The idea is to infect many machines (bots) to allow an attacker (botmaster) to control them through a command-and-control server. The botnet can be used for the following;

- **data theft** — steal credit card numbers or passwords
- **spam** — less likely to be shut down, compared to single server
- **DDoS** — flood servers with requests
- **brute-force** — similar reasoning to spamming, passwords / credit card credentials
- **network scanning** — probing other hosts
- **click fraud** — generate advertising revenue from different sources
- **cryptojacking** — see above
- **rental** — botnets can also be rented out for use by others

Analysis can be performed on captured samples (to aid in detection or removal), obtained from cleaning up an infection or running **honeypots** (by willingly installing malware). Effects on storage, system settings and network traffic are often analysed in a virtual machine sandbox. However, it may be difficult to trigger malicious behaviour (since it may behave differently in a virtual environment).

Detection can be performed by extracting signatures from analysed samples. **Static** signatures are sequences of bytes, typical of malware, and can be detected quite simply and quickly. However, this method is also easy to evade, where samples are artificially made different from each other, with **metamorphic** malware, or by the use of **crypting** services, which encrypt and obfuscate malware until it is no longer detected (FUD).

On the other hand, **dynamic** signatures or behavioural analysis can be performed, where the host is monitored for patterns of actions typically performed by malware (such as reading data then sending data over a network). A way for this to be evaded is for the malware to mix malicious behaviour with legitimate behaviour.

Current defences for malware include standard antivirus software, which scan existing and downloaded files for static signatures, as well as **end-point protection (EPP)**, which monitors the host for dynamic signatures. Browsers also now include blacklists which prevent access to pages known to be hosting phishing sites and malware. Network based protection can also be used.

However, signatures and blacklists are both based on observed malware, therefore attackers have a window of opportunity before detection. As such, prevention is often the best strategy, such as educating humans to avoid direct installs. Software should also be updated and patched in response to disclosures; it is rare that zero-days are used in attacks, as they are difficult to find and expensive.

Threat Modelling

Threat modelling can be used to guide decision making, by considering who the attackers are and their goals. We should also consider what attacks are likely to occur, and what assumptions the system relies on.

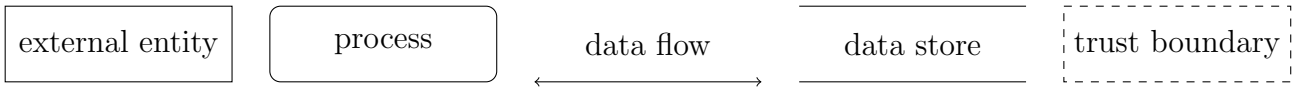
Rather than performing the modelling on the code of the system itself, it's done on the model of the system, thus being free from implementation and deployment details. This allows us to identify better design implementations before the system is built, or can be used to guide the security review of a system after deployment.

There are three key steps;

1. model the system

   This uses consistent visual syntax, to allow for multiple researchers to understand, as well as to build experience. In this course, we focus on **system architecture**, rather than focusing on assets like passwords, credit card numbers, or focusing on attackers.

   Data-flow diagrams (DFD)s are used to depict the flow of information across components. **Trust boundaries** help establish what principal controls what, and attacks tend to cross these boundaries.

   For example, if two processes exist inside the same trust boundary, we generally don't need to be worried about attacks from one process to the other. However, we do need to be concerned about any data flow arrows that cross the boundaries.

2. identify threats (STRIDE / attack trees)

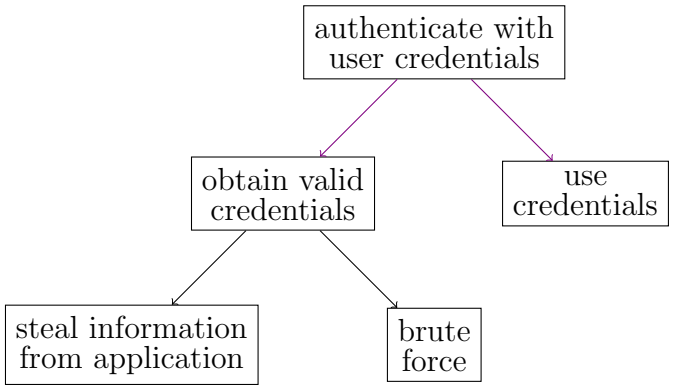
   For STRIDE, we ask what may go wrong in each element of a DFD;

   - **s**poofing — pretend to be something else
   - **t**ampering — modifying without permission
   - **r**epudiation — denying to have performed an action
   - **i**nformation disclosure — revealing information without permission

- **d**enial of service — prevent system from providing a timely service
- **e**levation of privilege — achieve more than what is intended

Threats may belong to more than one of these categories, and threats should be document by writing risk-based security tests when possible.

Another approach is to create an attack tree, where the root node represents the goal of the attack, or the target asset. Children are the steps to achieve the goal, and the leaves are concrete attacks; by default, sibling nodes represent **sufficient** steps (only one needs to be satisfied), but special notation is used to represent **necessary steps** (where all need to be satisfied). Note that the course uses lines between the arrows to denote necessary steps, however I will be using matching colours. For example;



This can also be represented in a textual format, where the root is a bullet point, and the necessary steps are '+', with the sufficient points being '-'.

Attack trees are an alternative to STRIDE, for each element in a DFD, if the goal of an attack tree is relevant, the tree can be traversed to identify possible attacks. Similarly, we can look at previously seen attack trees.

It's important to focus on realistic threats. The threats that should be considered depend on the system being modelled, the budget, and the value of what is being protected.

3. evaluate and address threads (DREAD / META)

The two main approaches for evaluating threats are qualitative (based on insight, experience, and expectations) and quantitative (based on some numerical score). However, quantifying risk is difficult (and realistic parameters are hard to estimate), rare events are also hard to predict (and therefore hard to quantify).

The DREAD methodology is a ranking from 5 to 15, developed by Microsoft;

	rating	high (3)	medium (2)	low (1)
D	damage potential	attacker can subvert full security system, get full trust authorisation, run as administrator, upload content	leaking sensitive information	leaking trivial information
R	reproducibility	attack can be reproduced every time and does not require a timing window	attack can be reproduced but only with a timing window and particular race situation	attack is difficult to reproduce, even with knowledge

E	exploitability	novice programmer could make the attack in a short time	skilled programmer could make the attack	extremely skilled person and in-depth knowledge to exploit every time
A	affected users	all users, default configuration, key customers	some users, non-default configuration	very small percentage of users, obscure feature
D	discoverability	published information explains the attack, vulnerability in most commonly used feature and is noticeable	vulnerability in seldom-used part of product, would take thinking to see malicious use	obscure bug, and users unlikely to work out damage potential

After a thread is addressed, a response should be recommended;

- **m**itigate — make threat harder to exploit

  For example, if the threat was password brute-forcing, mitigations could require better passwords or locking accounts after some number of failed attempts.
- **e**liminate — remove feature exposed to threat
- **t**ransfer — let another party assume the risk

  Continuing with the login scenario, we can use a third party login system. The cost is that the third party has information about customers, and that legal responsibility may still remain (despite technological risk being transferred)
- **a**ccept — when other options are impossible or impractical

  If someone was to guess the password on the first try, nothing can prevent it. It's important to keep track that the threat remains active.

Responses should be documented, such as in a project issue tracker.