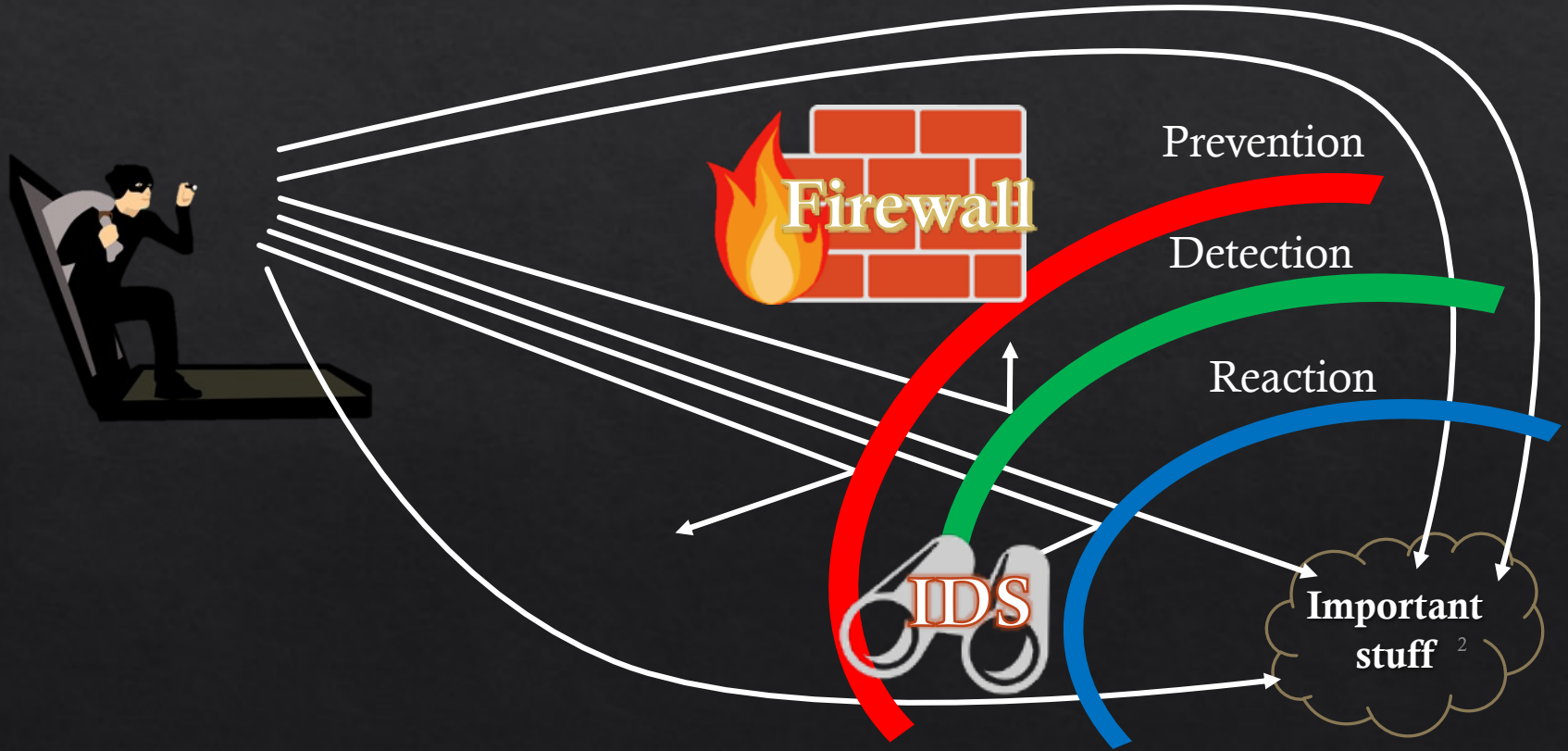
6. Security Tools

Jin Hong

jin.hong@uwa.edu.au

Tools


Firewall
Prevention
Detection
Reaction
IDS
Important stuff
2

Firewall

- We create networks and subnets **everywhere**
  - Need to ensure that bad things don't come into our networks
- Forms the **first** barrier for protecting LANs
- Sits **between** the premises network and the Internet
- Typically, **all traffic** from outside to inside passes through the firewall
- Unauthorised traffic will be **filtered** by the firewall

Firewall

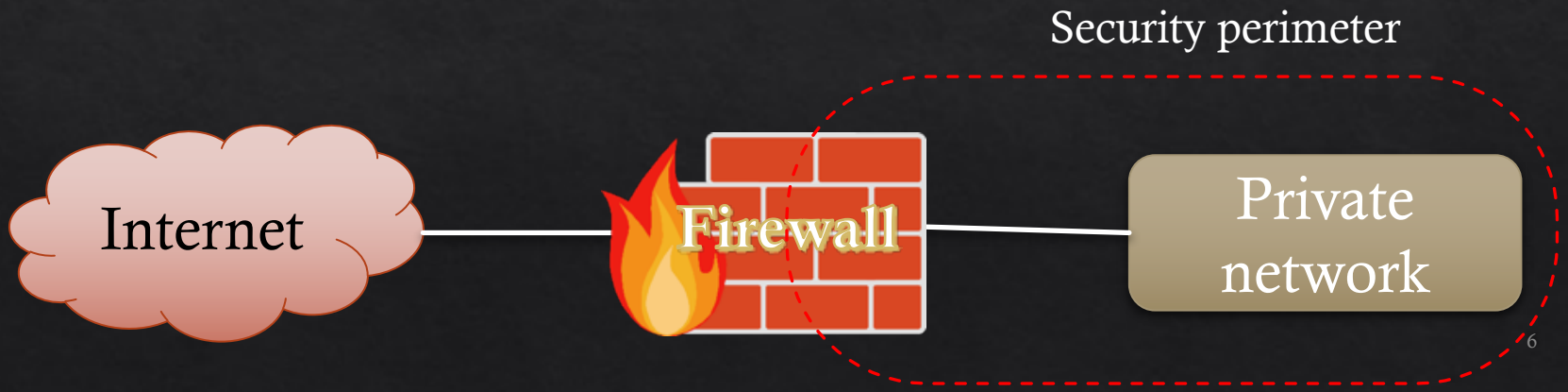
- What are the things examined by a firewall?
  - IP address
  - Protocol headers, payload, or port number
  - Track of client-server sessions
  - Application level protocols

Types of Firewalls

- Packet filtering firewall (Gen 1)
- Stateful inspection firewall (Gen 2)
- Application proxy firewall (Gen 3)
- Circuit-level proxy firewall (Gen 3)

Gen1: Packet filtering

- Defines a set of rules for each IP packet
- Apply the rules to determine the packet is forwarded or dropped



Gen1: Packet filtering

- Filtering rules are based on the following
  - Source IP address
  - Destination IP address
  - Transport level address (e.g., TCP or UDP port number)
  - IP protocol field
  - Interfaces
    - A firewall with 3 or more ports, which interface the packet came from or is going to

Gen1: Packet filtering

- A list of rules are established using the information
- If the incoming (or outgoing) packet matches to one of the rules, that rule is **invoked** (forward/discard)

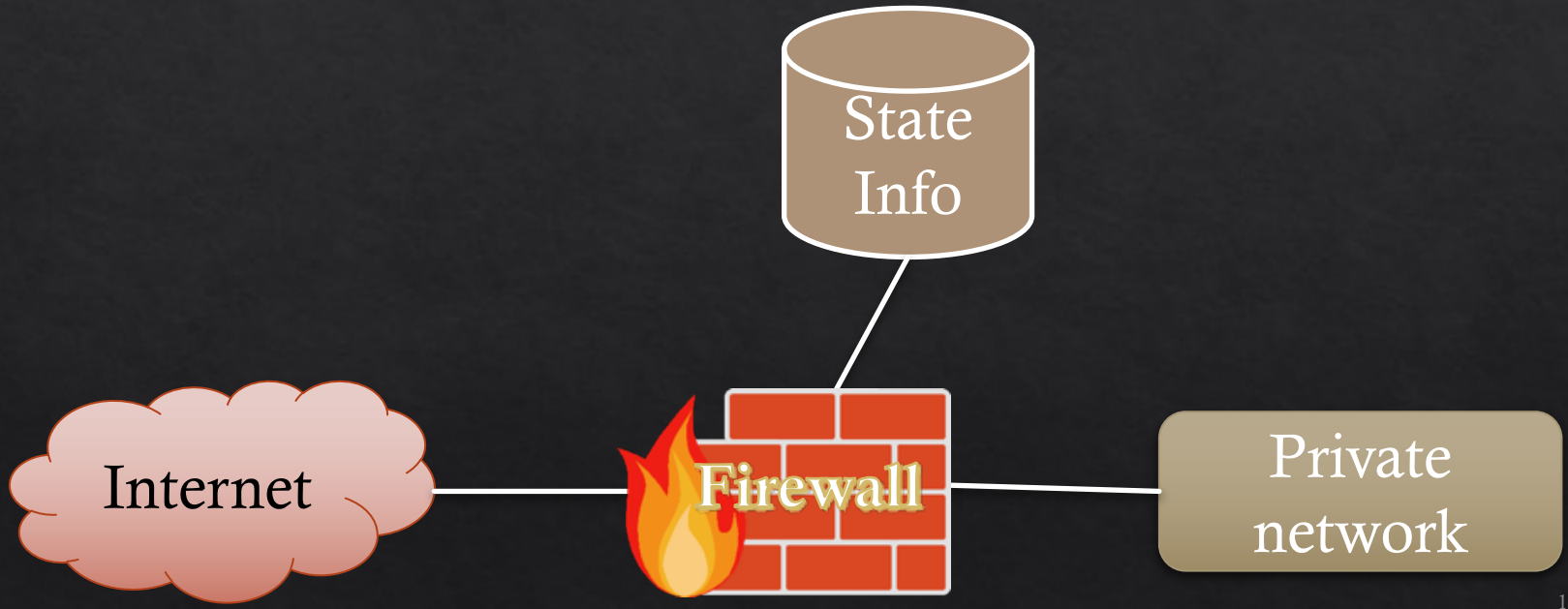
Gen1: Packet filtering

- Advantages:
  - Simple
  - Typically transparent to users and are very fast
- Disadvantages:
  - Cannot prevent attacks at the application layer
  - Limited logging capabilities
  - No advanced user authentication
  - TCP/IP protocol bugs can be exploited – e.g., IP spoofing

Gen2: Stateful inspection

- Generation 1 firewalls do not examine higher layer context
- Generation 2 firewalls address this problem by examining each IP packet in context
  - Keeps the track of client-server sessions
  - Check each packet validity – e.g., does it belong to someone?
- Provides better capabilities to detect bogus packets

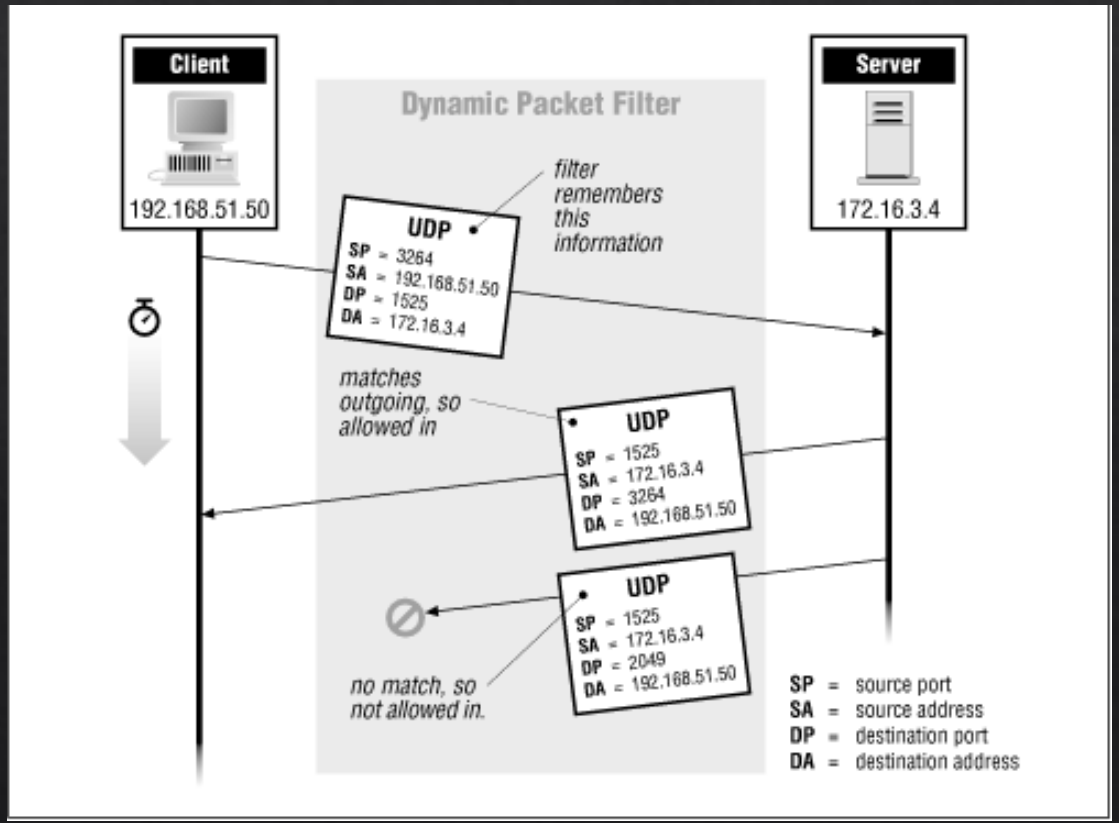
Gen2: Stateful inspection



Gen2: Stateful inspection

- TCP connections (typically)
  - Server port number < 1024
  - Client port number between 1024 and 16383
- Permanent assignments
  - (20, 21 -> FTP), (23 -> Telnet), (25 -> SMTP), (80 -> HTTP)
- If client wants to use port 2048, firewall must allow incoming traffic on this port
- Sol: To keep track of the outgoing requests

Gen2: Stateful inspection



Gen2: Stateful inspection

- The firewall keeps track of currently established connections

Source IP	Source Port	Dest. IP	Dest. Port	Con. State
192.168.1.100	1030	210.9.34.11	80	Established
192.168.1.103	1875	173.46.34.101	25	Established
192.43.1.101	2248	192.168.1.6	80	Established
210.168.1.113	1056	192.168.1.6	80	Established
222.99.1.73	1301	168.39.45.67	79	Established

- the profile

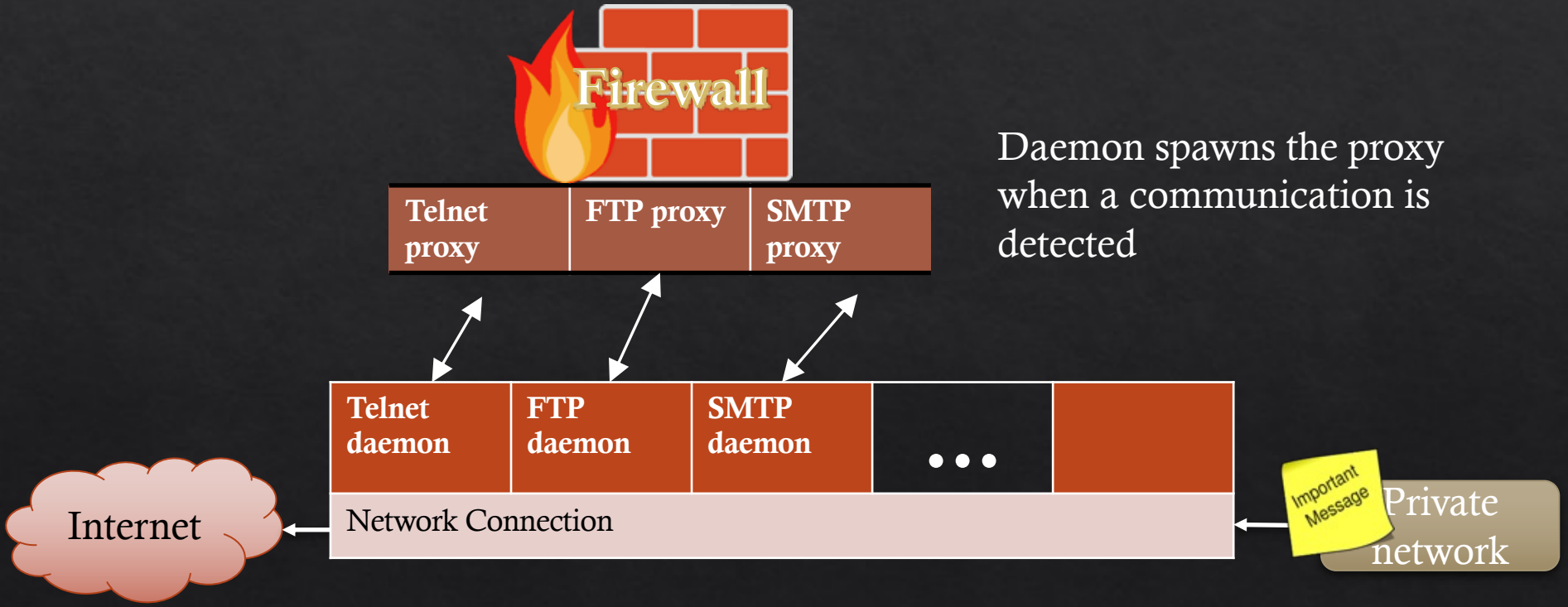
Gen2: Stateful inspection

- Options to keep track of the TCP sequence numbers
  - To prevent attacks that depend on the sequence number
  - Q: What attack was it?
  - E.g.,
- Some may also inspect application data for well known protocols
  - e.g.,

Gen3: Application proxy

- Application proxy (application level gateway)
- Relays application-level traffic

Gen3: Application proxy



Daemon spawns the proxy when a communication is detected

Gen3: Circuit-level proxy

- Proxy at the transport layer (TCP/IP model)
  - Establishes two TCP connections
- Typically used for **trusted** inside users
- Monitors TCP data packet handshaking and session fulfilment of firewall rules and policies
- Does **not filter** individual packets
- Relays TCP segments **without examining** contents
- Enforce a security function
  - That determines which connections will be allowed

Gen3 and more

- Application level
  - Considers commands in the application protocols
- Circuit level
  - Does not consider application protocols, but
  - Provides service for a wide variety of different protocols
- Multi-level firewalls

Firewalls summary

The Goods	The Bads

• All communication goes through a single point
• Specific location(s) for monitoring events
• Provides a platform for various Internet functions not security related
  • e.g., serve as the platform for IPSec
	• Attackers can bypass firewall
	• Cannot protect from internal threats
	• Improper network design can be penetrated from outside
	  • e.g., not so secure wireless LAN
	• Infected devices can be brought into the internal network
	  • e.g., laptop, tablets, mobile phones, USB

Defense Mechanisms 2: outline

- Techniques and methods
  - Vulnerability scanning
  - Sandboxing
  - Steganography
  - Moving Target Defense

Vulnerability

- What is a vulnerability?
  - Software flaws or configuration error(s) that allow attackers to violate the security objectives (CIA) of the system
- Vulnerabilities arise from bugs in applications or design flaws in the system
- Vulnerabilities can be found in all layers of the system

Vulnerability Scanning

- Vulnerability scanning can
  - Identify **flaws** in the system applications and designs
  - Find what **services** are running
  - Ensure vulnerabilities are **patched** correctly
  - Part of a pre-emptive step to **remove** vulnerabilities before attackers exploit them

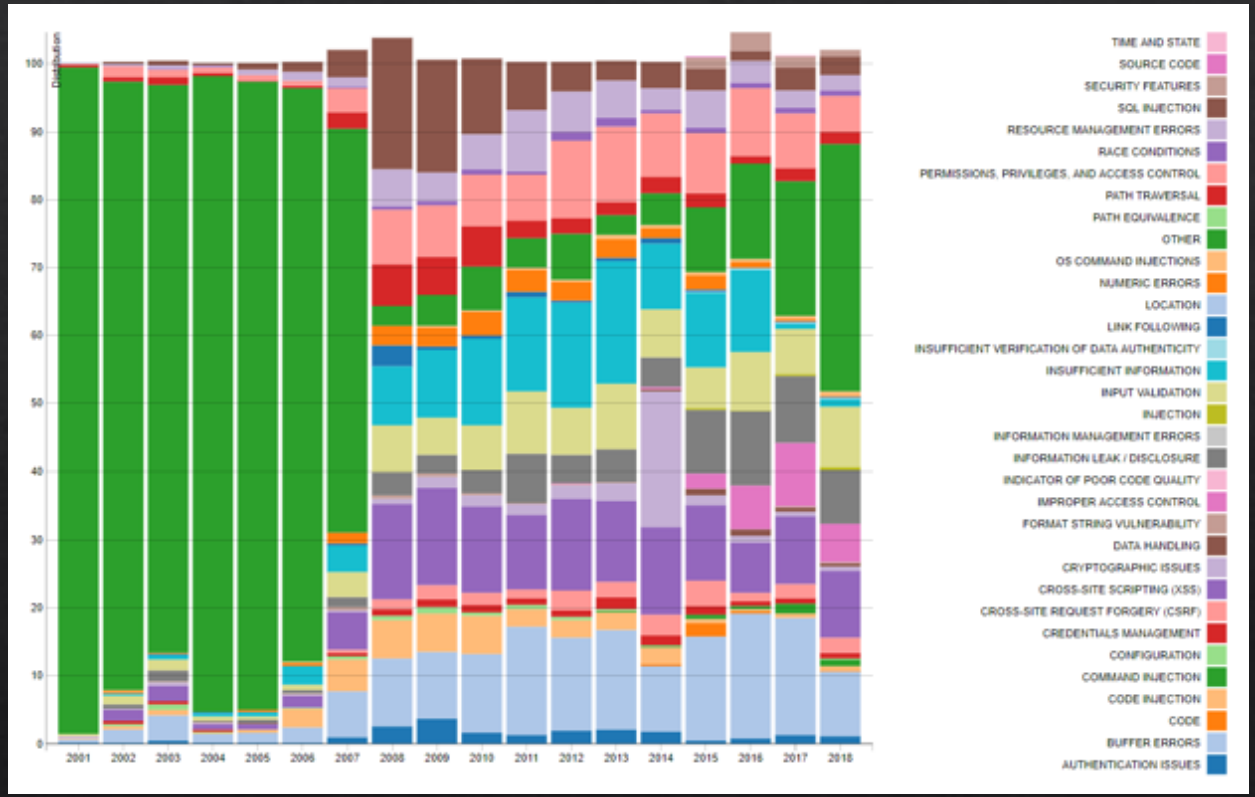
Vulnerability Scanning

- "Scanning" is means of **collecting information** about computer systems and the entities they belong to
- Attackers also conduct "scanning" in the pre-attack phase, or more commonly known as **reconnaissance**

Vulnerability Scanning

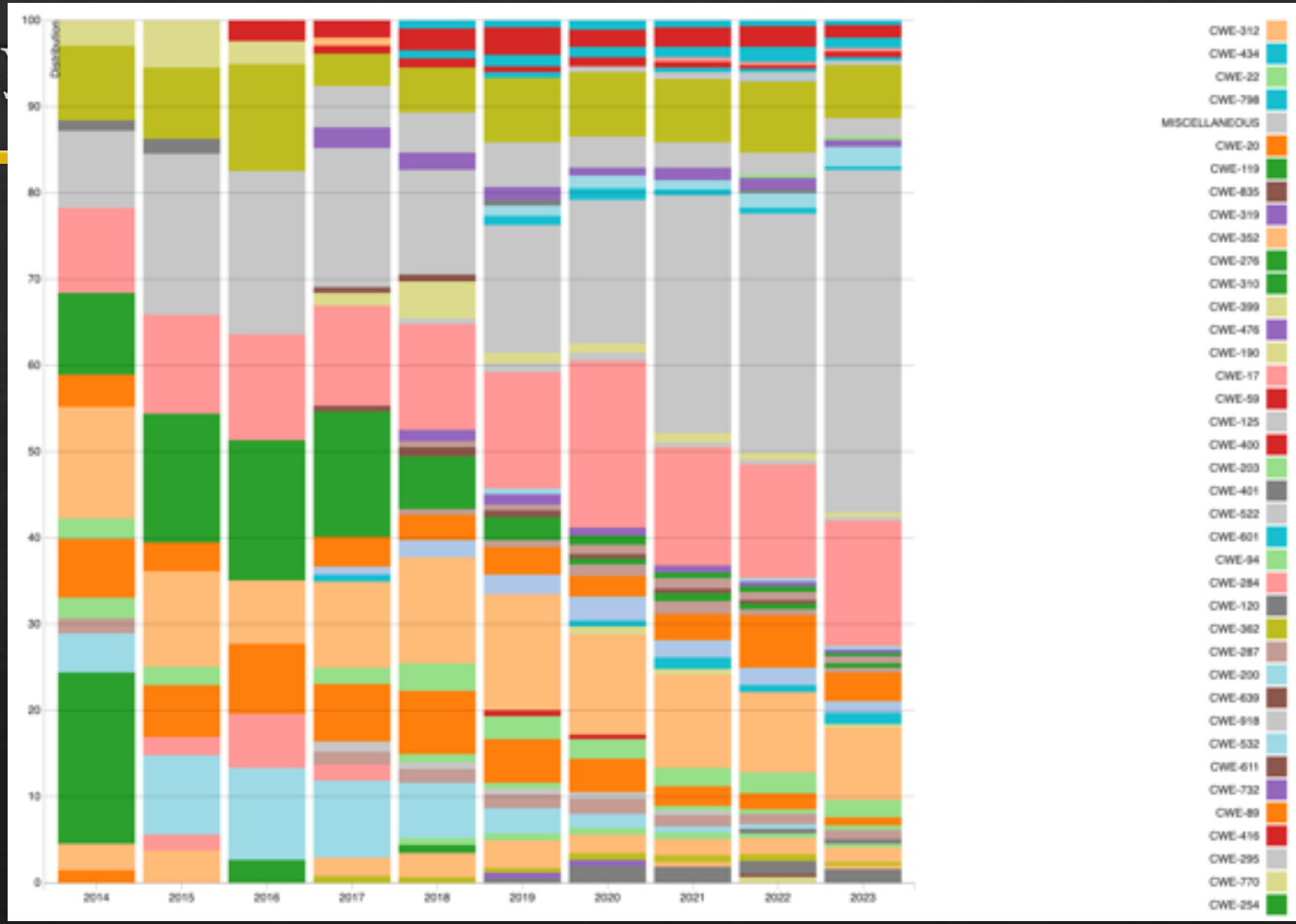
- How are vulnerabilities discovered?
  - Known vulnerabilities using tools
  - New vulnerabilities by humans
    - Hacker groups
    - Security companies or
    - Researchers
  - E.g., a group may discover a vulnerability related to a software (could be by accident or through a directed research)
  - Vulnerability is disclosed to the public
    - **When?**

Vulnerability Scanning



https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cwe-over-time#vuln-type-total-by-year-title

Vulnerabilit



https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cwe-over-time#vuln-type-total-by-year-title

Vulnerability Scanning

- Some widely used canning categories are
  - Port scanners
    - Nmap – https://nmap.org/
  - Network vulnerability scanners
    - Nessus - https://www.tenable.com/products/nessus/nessus-professional
    - OpenVAS - http://www.openvas.org/
    - Qualys, SAINT, etc.
  - Web application vulnerability scanning tools
    - List from OWASP - https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
- There are other types as well
  - E.g., database security scanner, host-based vulnerability scanner etc.

Basics of scanning

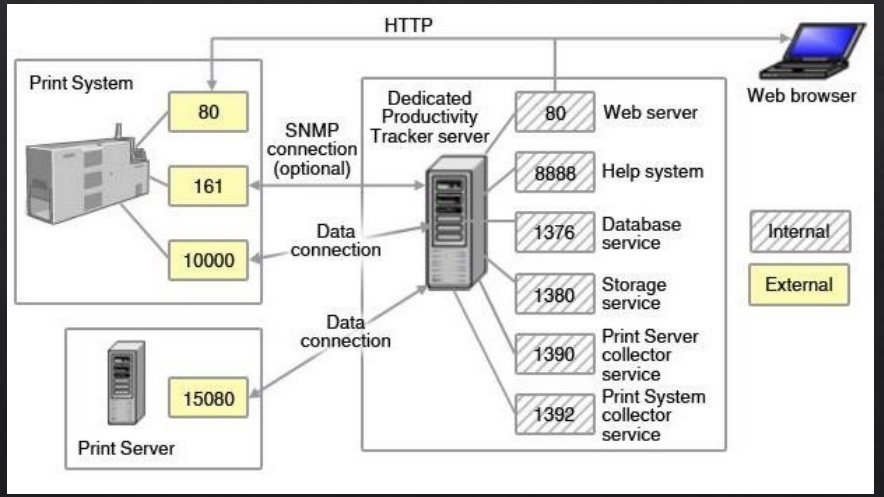
- Ping and ICMP (Internet Control Message Protocol) scan
  - Typically for server scanning
- TCP and UDP scan
  - Typically for port scanning

Ping in Windows

```
C:\>ping www.uwa.edu.au

Pinging www.uwa.edu.au.cdn.cloudflare.net [104.20.11.164] with 32 bytes of data:
Reply from 104.20.11.164: bytes=32 time=2ms TTL=56
Reply from 104.20.11.164: bytes=32 time=2ms TTL=56
Reply from 104.20.11.164: bytes=32 time=2ms TTL=56
Reply from 104.20.11.164: bytes=32 time=2ms TTL=56

Ping statistics for 104.20.11.164:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 2ms, Average = 2ms

C:\>_
```

Port scanning

Perth Air“port”



Well known ports

Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP, TCP	DNS
67, 68	UDP	DHCP
69	UDP	TFTP
80	TCP	HTTP (WWW)
110	TCP	POP3
161	UDP	SNMP
443	TCP	SSL

TCP/UDP scanning

- UDP scanning
  - Attacker sends an UDP packet
    - No response → the port is open
    - ICMP Unreachable packet received → the port is closed
- TCP scanning
  - Attacker sends SYN packet
    - SYN+ACK packet received → the port is open
    - RST+ACK packet received → the port is closed

TCP scanning

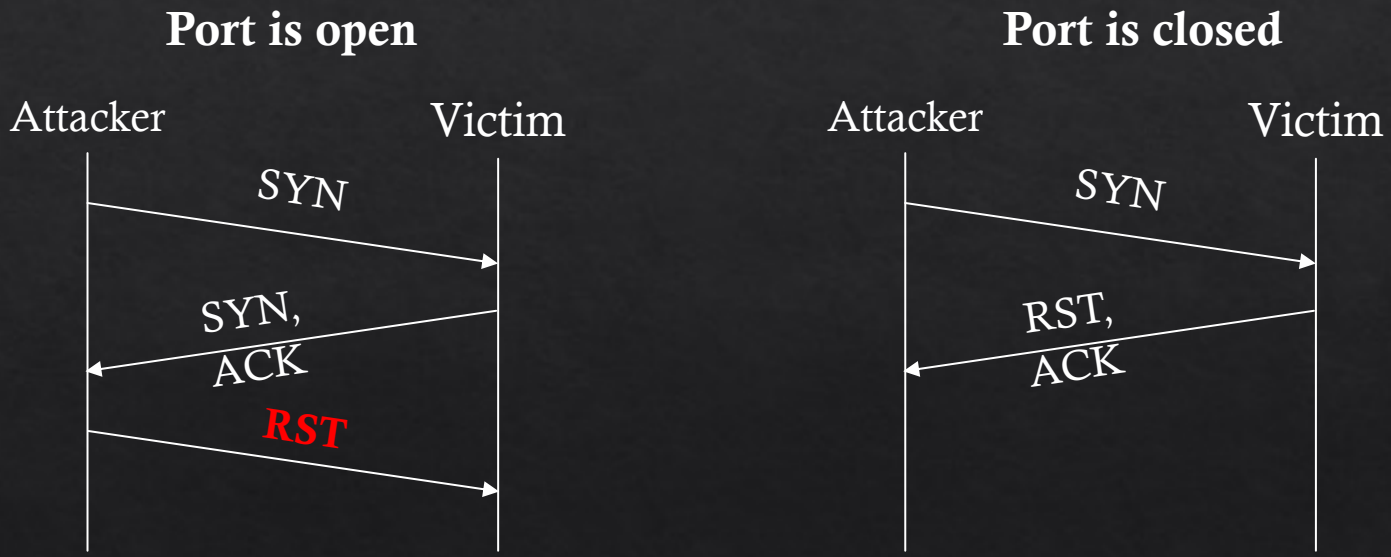
- Initiate the 3-way handshake to check the port
- Provides **fast** port scanning and requires **no privilege**

- However, this is easily **detectable**
  - And so likely to be **blocked**

  - Sol: use ***stealth*** scanning methods

Stealth TCP scanning 1

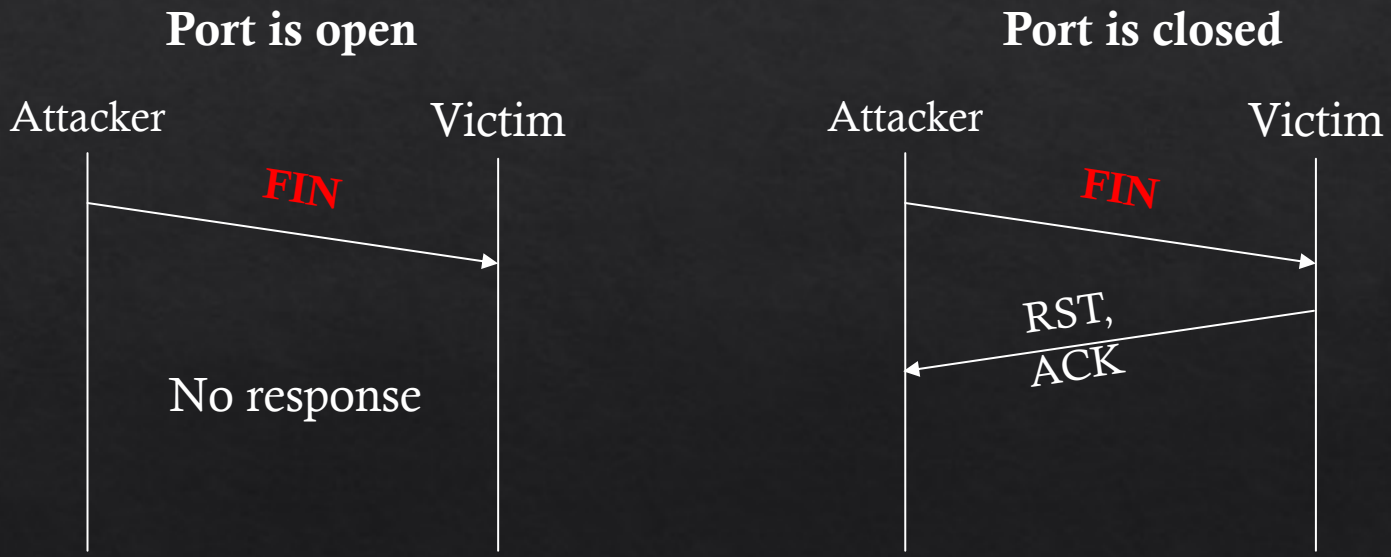
- TCP SYN scanning (aka half-open scanning)



Note: many systems will not log this connection attempt

Stealth TCP scanning 2

- TCP FIN scanning

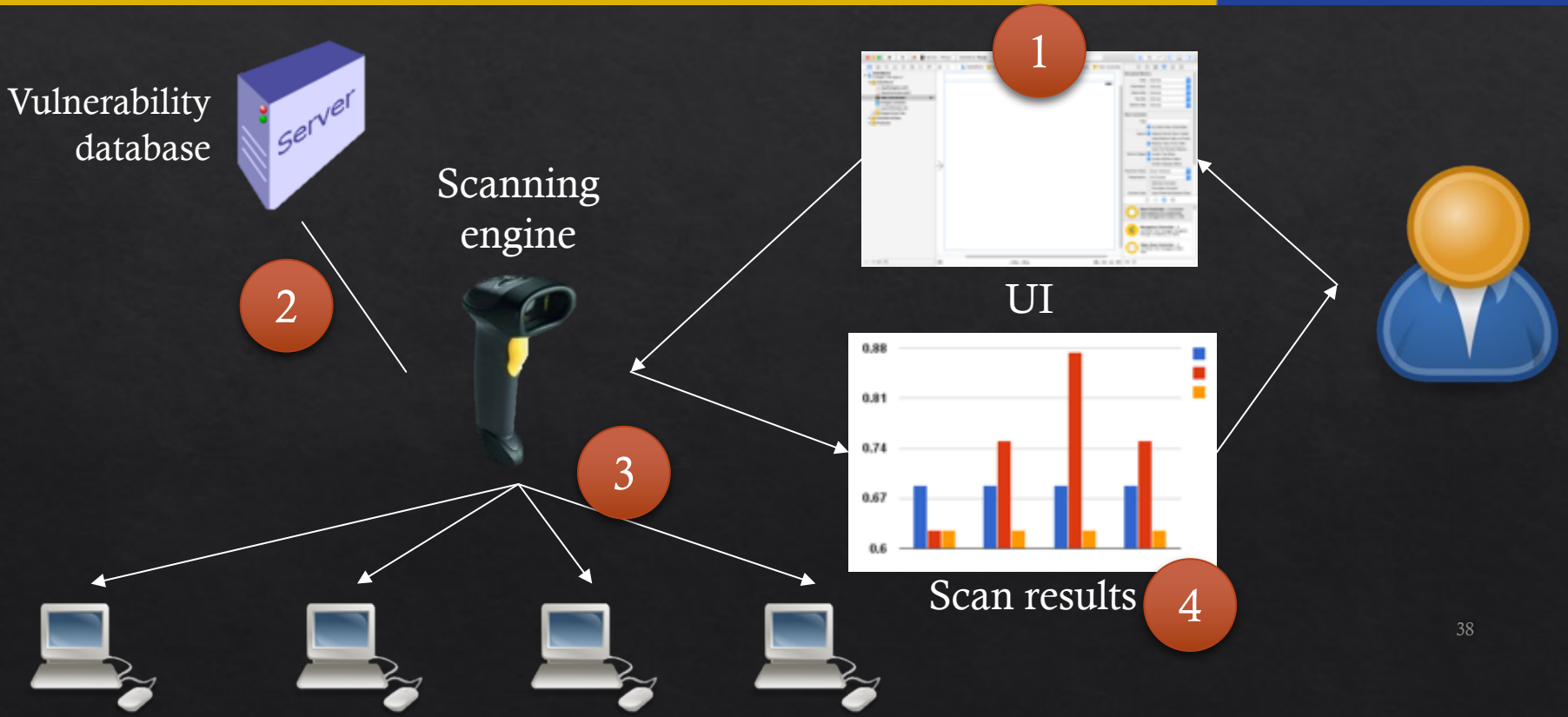


Note: Some systems may sent RST packets regardless

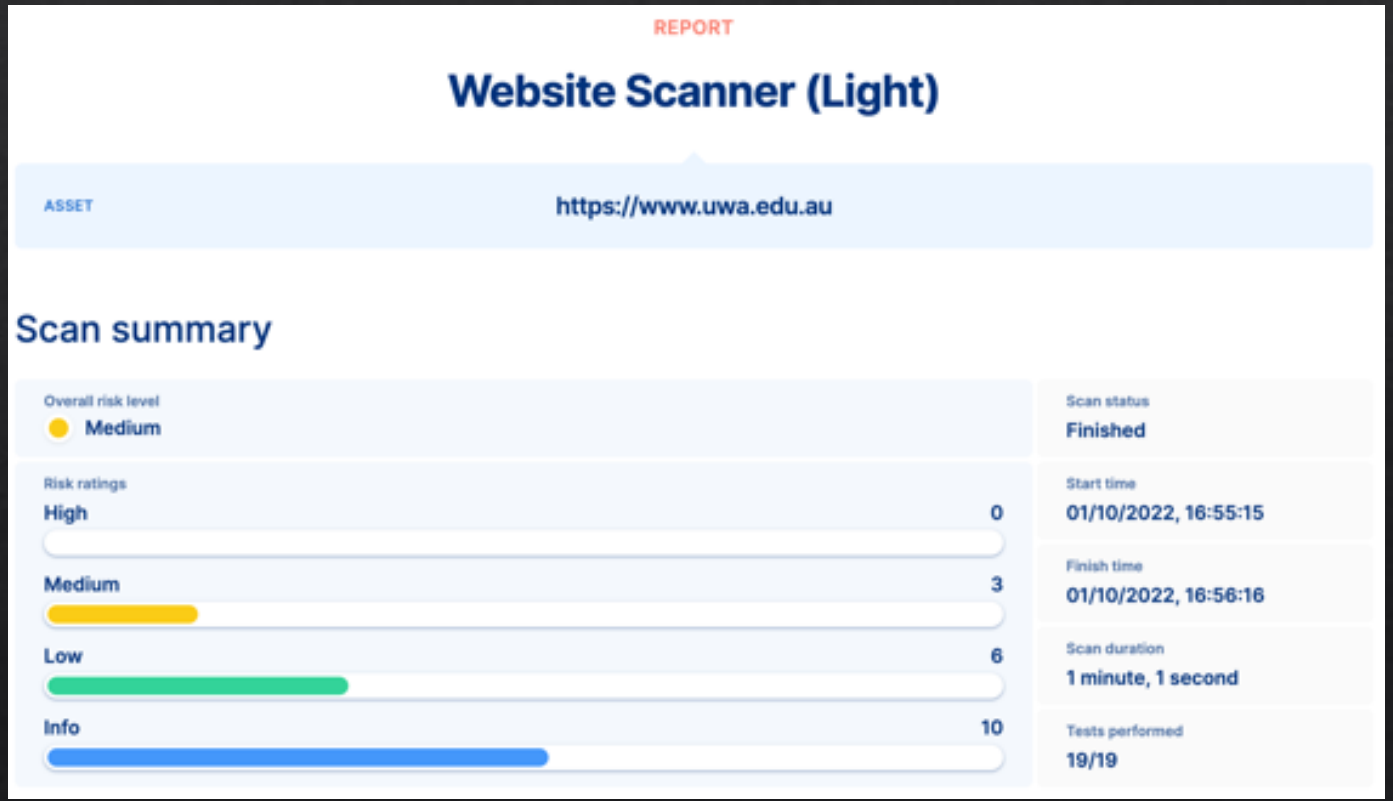
Stealth TCP scanning +

- Other techniques include
  - Fragmentation scanning
    - Chopping the existing scanning packets into smaller fragments
  - TCP reverse ident scanning
    - Ident protocol to extract connection information
  - FTP bounce attack
    - Manipulating the FTP server protocol interpreter to redirect files
  - Etc… see *additional items* for more.

Vulnerability Scanning



Vulnerability Scanning



REPORT

Website Scanner (Light)

ASSET https://www.uwa.edu.au

Scan summary

Overall risk level
Medium

Risk ratings

High	0
Medium	3
Low	6
Info	10

Scan status
Finished

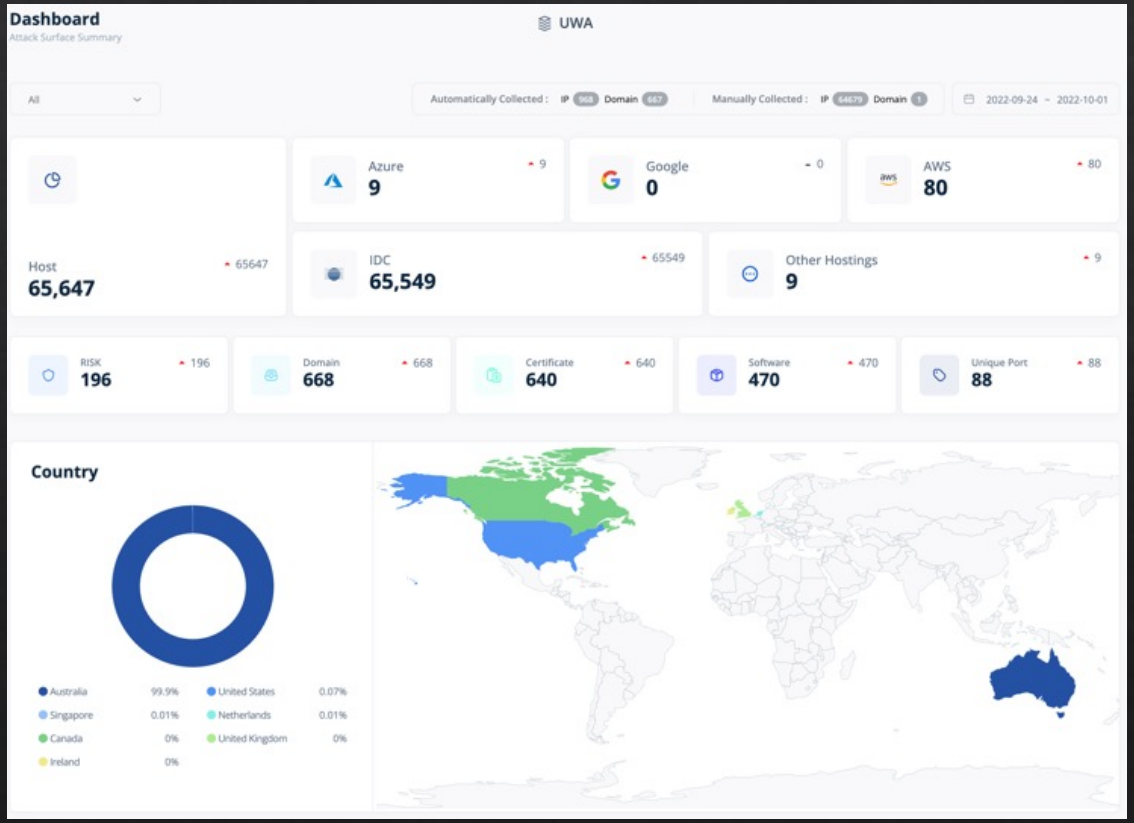
Start time
01/10/2022, 16:55:15

Finish time
01/10/2022, 16:56:16

Scan duration
1 minute, 1 second

Tests performed
19/19

Vulnerability Scanning



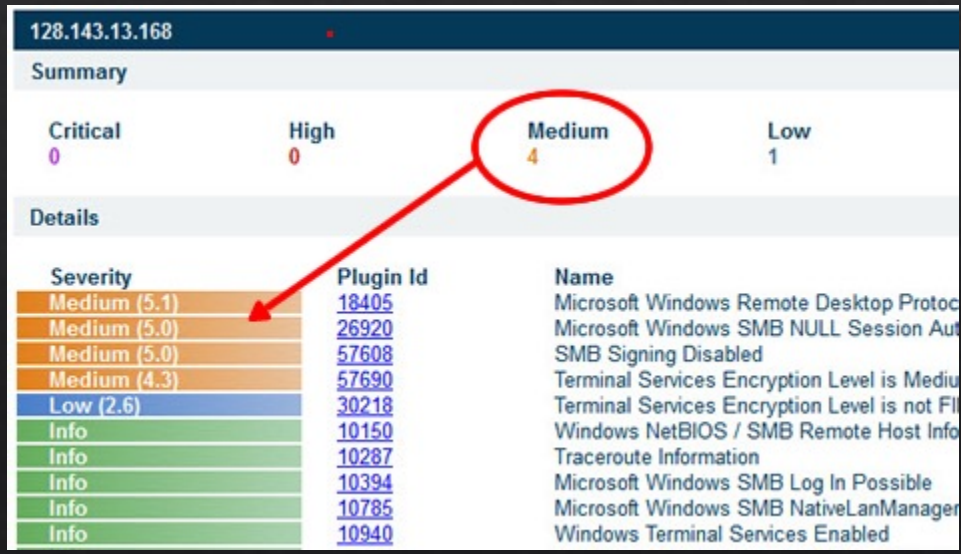
Example: Nessus

- From Tenable©
- Runs on various Oses
  - Linux, Unix-based, Windows etc
- Keeps their own vulnerability database
  - Good or bad?
- Uses Nessus Attack Scripting Language
  - So you can write your own scripts and plugins
- There are various (and many free) plugins available to use
  - https://www.tenable.com/plugins/nessus/families
- Vulnerabilities found are classified based on their risk-factor
- Is a commercial product
  - In 2018: $2300USD per annum for the professional version
  - In 2019: $2800USD per annum
  - In 2024: $4389USD per annum!!

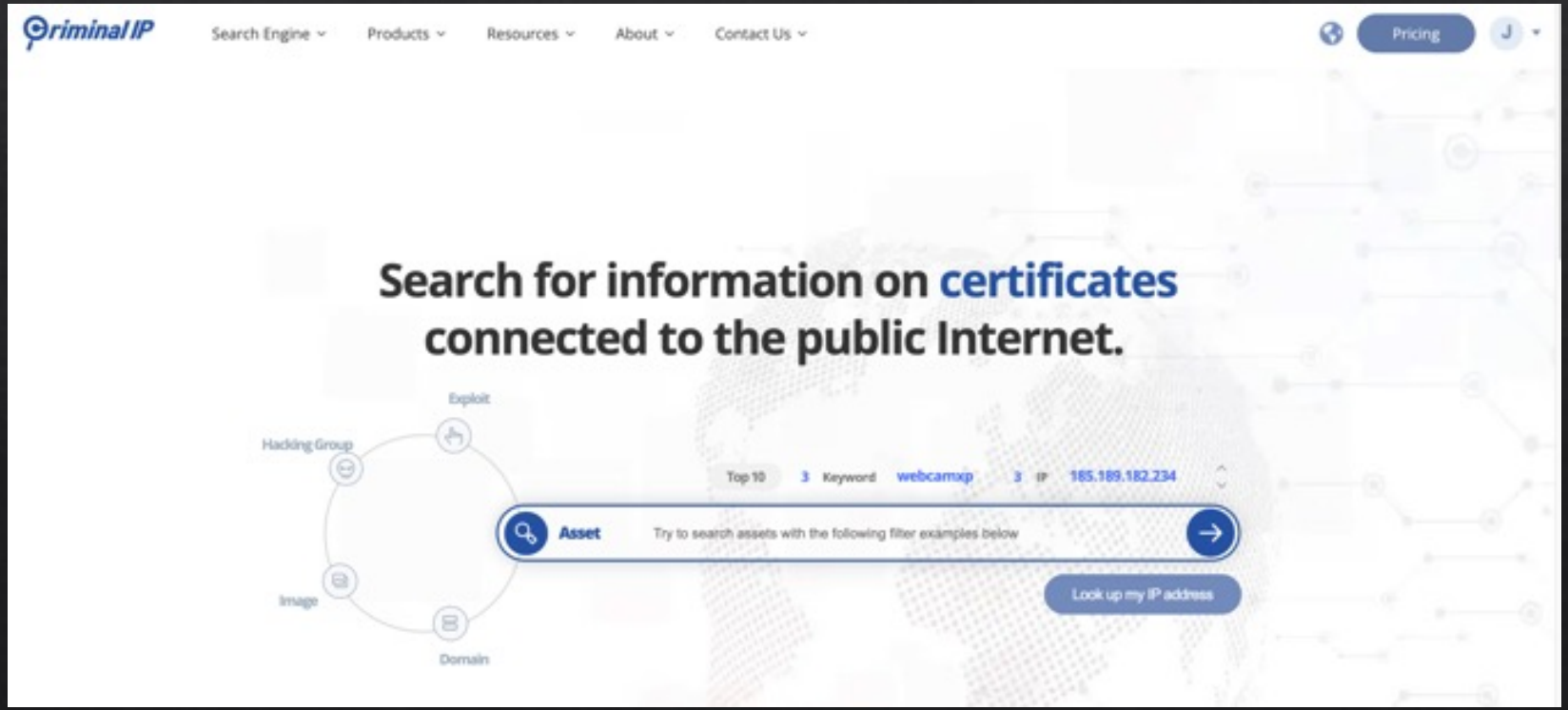
https://www.tenable.com/buy

Example: Nessus

- Nessus can scan various types of vulnerabilities
  - Windows vulnerabilities
  - Insecure scripts
    - E.g., Common Gateway Interface
  - RPC (remote procedure call) program vulnerabilities
  - Firewall misconfigurations
  - FTP insecure implementations
  - Etc.
- Provides ranking of them



Example: CriminalIP



Example: CriminalIP

- IP-based threat detection tool
- Scans the whole internet regularly to update its data
- Using AI, determine the vulnerabilities and threats detected based on network traffic

Vulnerability database

- Widely used vulnerability database

Vulnerability database

- CVE provides a list of standardised names for vulnerabilities
  - These vulnerabilities are publicly known
  - Security experts form editorial board and provide the description
  - E.g., CVE-2014-0160
    The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the **Heartbleed** bug.
- MITRE Corporation maintains CVE and moderates the editorial board discussions

http://cve.mitre.org

Vulnerability database

- NVD is a comprehensive cybersecurity vulnerability database
  - Maintained by NIST, an US organisation
  - Synchronised with CVE vulnerability naming standard
  - Provides some security metrics about the vulnerability
    - E.g., CVSS base score, vectors, etc.
    - CVE-2014-0160 has the CVSS BS of 5.0

http://nvd.nist.gov

Vulnerability Scanning: Summary

- Requires continuous efforts to discover and mitigate vulnerabilities
- Difficult to address problems with new and unknown vulnerabilities
  - E.g., zero-day vulnerabilities
- Global efforts are made to address issues associated with each steps of vulnerability scanning
  - E.g., improve anomaly detection using AI and machine learning

Sandboxing

- Often we have untrusted code that we wish to run
  - E.g., program from Internet including toolbars, viewers, codecs etc
- We should contain that code in a zone such that if it misbehaves, we can easily kill it
- Create the zone through confinement – sandboxing!

Sandboxing

- **Confinement**: To contain the application in an isolation to ensure it does not carry out unapproved actions
- Can be implemented at many levels:
  - **Hardware**: run application on isolated hw (air gap)
  - **Virtual machines**: isolate OS's on single hardware
  - **System call interposition**: Isolates a process in a single operating system
  - **Software Fault Isolation** (SFI): Isolating threads sharing same address space
  - **Application specific**: e.g. browser-based confinement

chroot

- Example: using *chroot*
- Often used for guest accounts

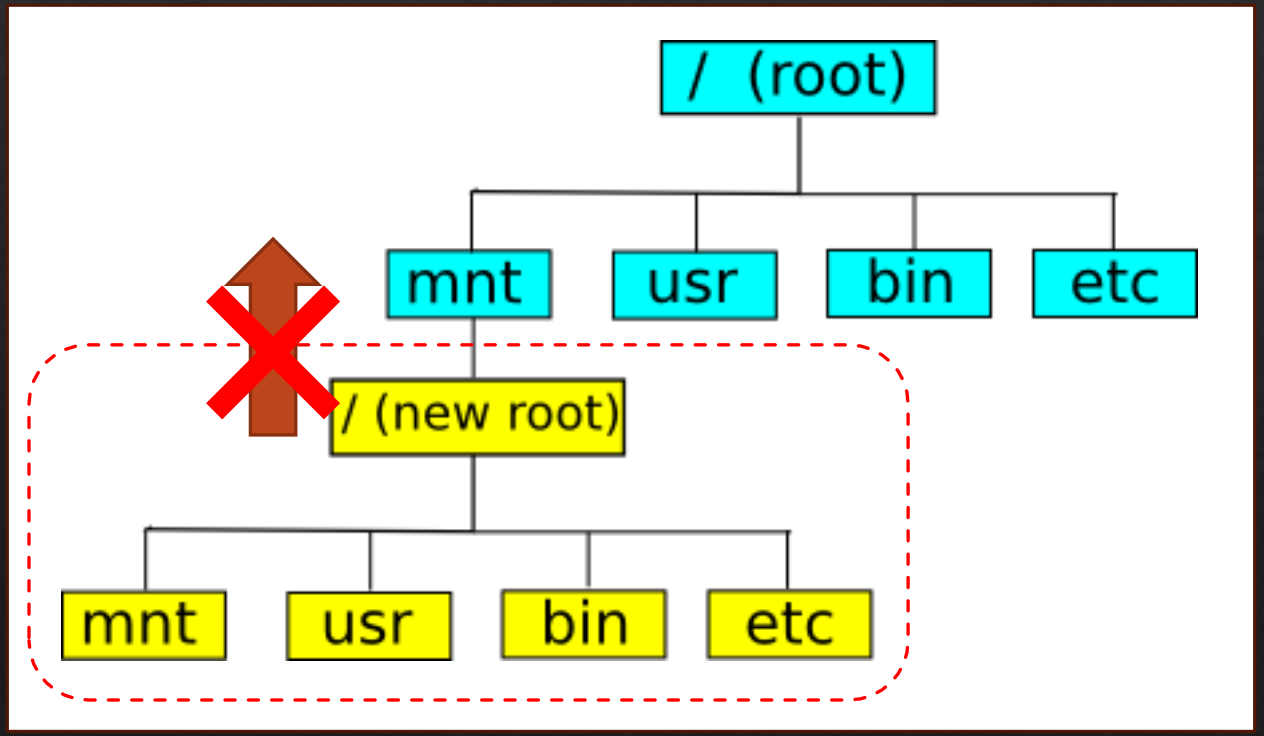
Requires root privilege

chroot /tmp/guest	root dir "/" is now "/temp/guest"
su guest	EUID is set to "guest"

- The path "/tmp/guest" is added to the file system accesses for applications in jail

Problem: utility programs also need to be available in the jail (e.g., ls, ps, vi)

chroot



Access is restricted within the boundaries

chroot

- Application cannot access files outside of jail
- Jailkit is used to address this problem
- auto builds files, libs, and directories needed in the jail environment
  - **jk_init**: creates jail environment
  - **jk_check:** checks jail env for security problems
    - checks for any modified programs,
    - checks for world writable directories, etc.
  - **jk_lsh**: restricted shell to be used inside jail

chroot

- Of course, people will try to break out of jails…

Scenario 1:

```
open("../../etc/password", "r")
```

Scenario 2:

```
mkdir "/asdf/etc/passwd"
chroot "/asdf"
chroot("../../../../../../../../../..")
```

If the process has a root privilege, you can escape the jail!

chroot

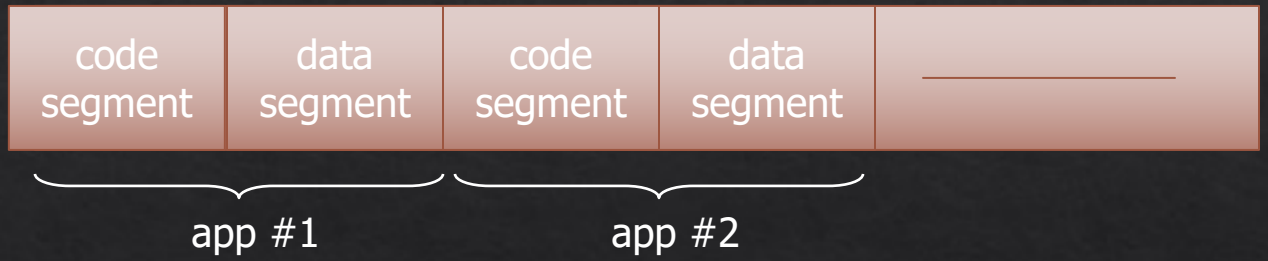
- chroot **lacks** many capabilities
  - All or nothing access to the file system
    - Not so useful for web applications etc
  - Need to provide access to files outside the jail
  - Malicious applications can access the network
    - And do something with other internally connected machines
- Suggestion: **do not** use chroot as a security solution
  - Use better approaches and combinations with other approaches

SFI

- Software Fault Isolation
- **Problem**: legitimate and malicious applications running in the same address space
  - E.g., device drivers trying to corrupt kernel
- **Solution**
  - Separate applications into different address spaces
  - Instructions are added before memory operations and verify behaviour (e.g., illegal memory access)

SFI

- Partition process memory into segments



- Locate unsafe instructions (e.g., jmp, load, store)
  - Add safety instructions for memory access
  - Add guards before unsafe instructions at compile time
  - Validate guards when loading them

SFI

- Reconstruct code instructions for safety
- Untrusted code (applications) should only be able to
  - Jump within its domain's code segment
  - Write within its domain's data segment

Unsafe instruction

```
STR R0, R1          ; write R1 to Mem[R0]
```

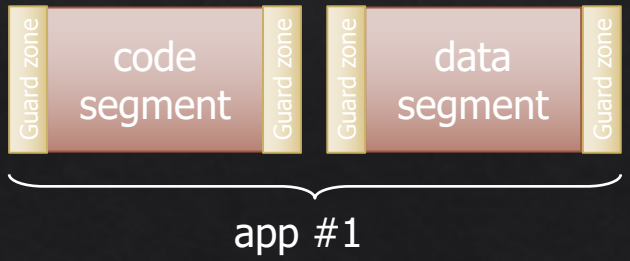
Rewritten instruction by sandbox

```
MOV  Ra, R0         ; copy R0 into Ra
SHR  Rb, Ra, Rc     ; Rb = Ra >> Rc, get the segment ID
CMP  Rb, Rd         ; Rd holds the data segment ID
BNE  fault          ; wrong data segment ID
STR  Ra, R1         ; Ra in data segment, so write
```

STR – store
MOV – move
SHR – shift logical right
CMP – compare
BNE – branch on not equal

SFI

- Some instructions use register and offset addressing
  - i.e., needs extra space for the instruction
- Add guard zone to each segment to avoid calculating the offset



SFI

- Verifier ensures **all** instructions are safe
- Verifier checks **no privileged** instructs are in the code
- Verifier also checks relevant instructions are **within** the code/data segments
- If the sandboxed code **fails** any checks, verifier *rejects* the code

SFI

- Provides a **good** performance with a little overhead
  - Typically around 4% processing overhead
- Confines writes and control transfers in extension's data and code segments, respectively
- Prevent execution of **privileged** instructions
- However, more **difficult** to implement on x86 architectures
  - It was not designed for x86 in the first place
  - Variable length instructions: hard to place guards
  - Many instructions that affects the memory: need more guards

Sandboxing summary

- Various techniques for sandboxing in different layers of the system
- It is difficult to fully isolate malicious code, due to dependencies
- Defining sandboxing policy is one of the top key challenges

Other cool tools

- There are many other tools that are useful for defence (as well as attacks), such as
  - Wireshark
  - Metasploit
  - Snort
  - Burp
  - OpenVAS
  - Etc.

Summary

- We looked at security related tools and methods
  - Firewall
  - vulnerability scanning
  - sandboxing

- There are **many other** security methods and techniques to be explored!
- All security methods and techniques provide means to **ensure the security objectives** of the system
- Should be carefully **designed and implemented** to maximise the security

Additional Items

- Firewall
  - https://www.digitalocean.com/community/tutorials/what-is-a-firewall-and-how-does-it-work
- List of security tools
  - https://sectools.org/
- Port scanning
  - https://nmap.org/nmap_doc.html
- Vulnerability database
  - CVE – https://cve.mitre.org
  - NVD – https://nvd.nist.gov
- Sandboxing
  - chroot jail escape – http://www.ouah.org/chroot-break.html