

11. Defence Techniques



What you need today

- ❖ Nothing fancy, just need your Kali
- ❖ Things we do today
 - ❖ Web vulnerability scanning
 - ❖ MTD

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

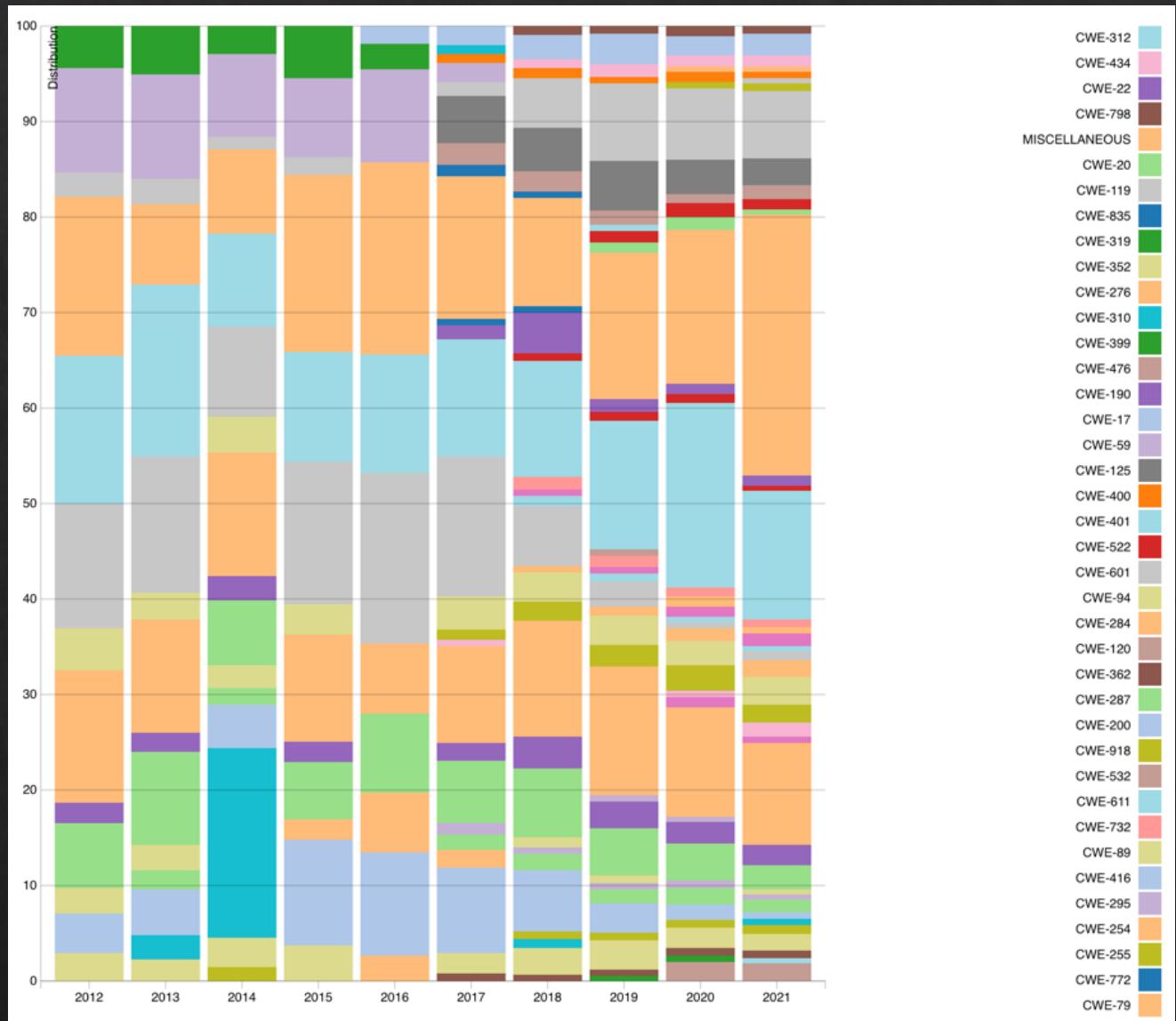
- ❖ Vulnerability scanning can
 - ❖ Identify **flaws** in the system applications and designs
 - ❖ Find what **services** are running
 - ❖ Ensure vulnerabilities are **patched** correctly
 - ❖ Part of a pre-emptive step to **remove** vulnerabilities before attackers exploit them
- ❖ “Scanning” is means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

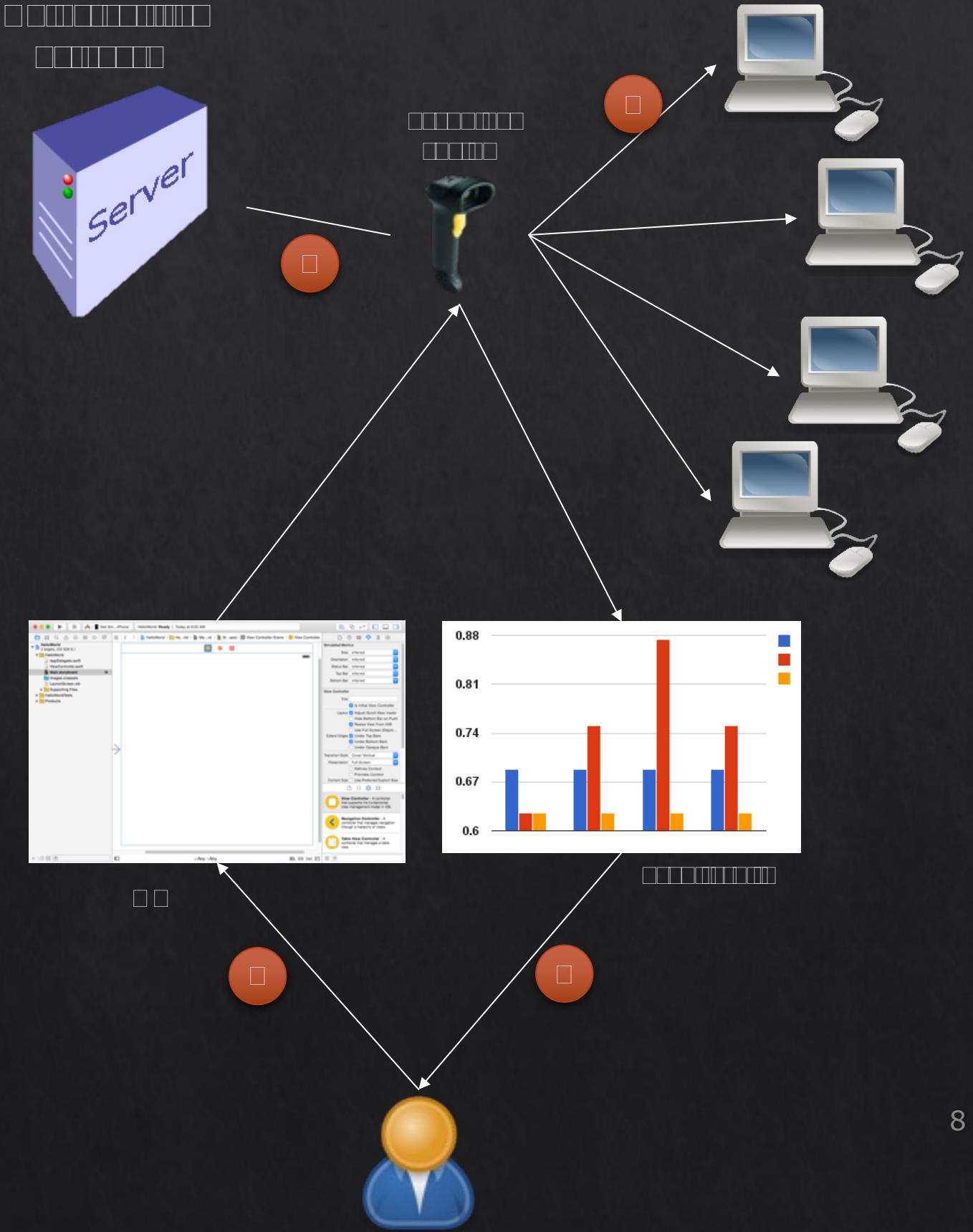
Vulnerability Scanning



Vulnerability Scanning

- ❖ Some widely used scanning 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.

Vulnerability Scanning



Vulnerability Scanning

REPORT

Website Scanner (Light)

ASSET

<https://www.uwa.edu.au>

Scan summary

Overall risk level

Medium

Risk ratings

High

Scan status

Finished

Start time

01/10/2022, 16:55:15

Medium

3

Low

6

Info

10

Finish time

01/10/2022, 16:56:16

Scan duration

1 minute, 1 second

Tests performed

19/19

Dashboard

Attack Surface Summary

UWA

All

Automatically Collected : IP 668 Domain 667

Manually Collected : IP 64679 Domain 1

2022-09-24 ~ 2022-10-01



Azure
9

▲

Google
0

▼

AWS
80

▲ 80

Host
65,647

▲ 65647

IDC
65,549

▲ 65549

Other Hostings
9

▲ 9

RISK
196

▲ 196

Domain
668

▲ 668

Certificate
640

▲ 640

Software
470

▲ 470

Unique Port
88

▲ 88

Country



Australia

99.9%

United States

0.07%

Singapore

0.01%

Netherlands

0.01%

Canada

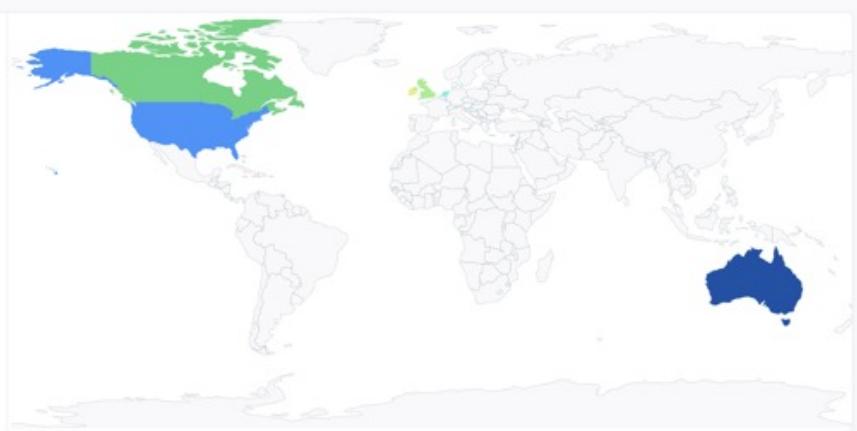
0%

United Kingdom

0%

Ireland

0%



9

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
 - ❖ Currently \$3729USD per annum for the professional version (as of 2022)



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

128.143.13.168			
Summary			
Critical	High	Medium	Low
0	0	4	1
Details			
Severity	Plugin Id	Name	
Medium (5.1)	18405	Microsoft Windows Remote Desktop Protocol	
Medium (5.0)	26920	Microsoft Windows SMB NULL Session Authentication	
Medium (5.0)	57608	SMB Signing Disabled	
Medium (4.3)	57690	Terminal Services Encryption Level is Medium	
Low (2.6)	30218	Terminal Services Encryption Level is not FIPS Enabled	
Info	10150	Windows NetBIOS / SMB Remote Host Information	
Info	10287	Traceroute Information	
Info	10394	Microsoft Windows SMB Log In Possible	
Info	10785	Microsoft Windows SMB NativeLanManager	11
Info	10940	Windows Terminal Services Enabled	

Vulnerability database

- ❖ Widely used vulnerability database



NVD

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



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



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!
-
- ❖ **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



- ❖ 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)
- ❖ Partition process memory into segments



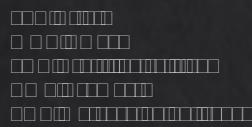
- ❖ Locate unsafe instructions (e.g., jmp, load, store)
 - ❖ Add safety instructions for memory access
 - ❖ Add guards before unsafe instructions at compile time
 - ❖ Validate guards when loading them

- ❖ 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

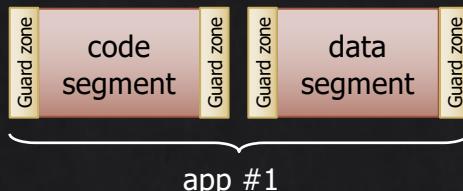
```
STR R0, R1 ; write R1 to Mem[R0]
```

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
```

- ❖ 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





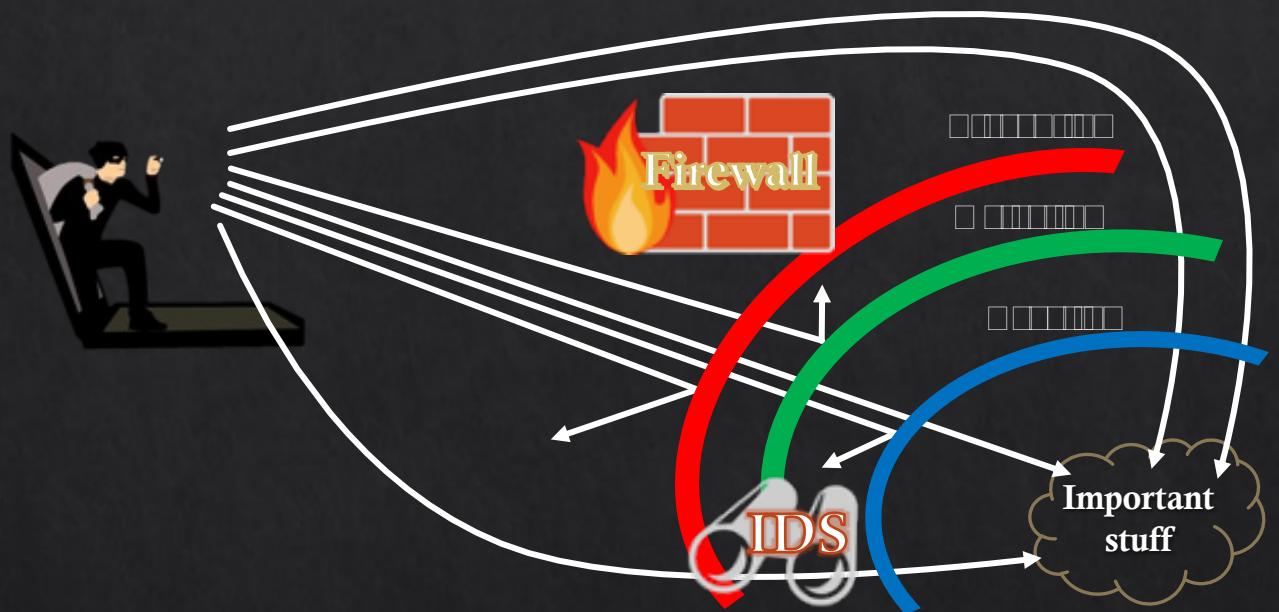
- ❖ Verifier ensures **all** instructions are safe
- ❖ Verifier checks **no privileged** instructions are in the code
- ❖ Verifier also checks relevant instructions are **within** the code/data segments
- ❖ If the sandboxed code **fails** any checks, verifier *rejects* the code

- ❖ 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 affect 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

Tools



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

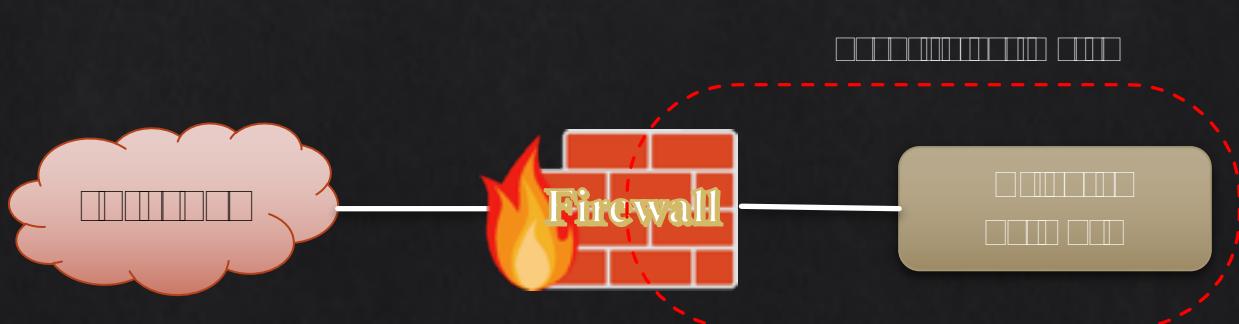


Types of Firewalls

- ❖ 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
- ❖ 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)
- ❖ What happens when there is no matching rule?

Gen1: Packet filtering

□ □□□□	□ □□□□□	□□□□	□□□□□□□□□	□□□□	□□□ □ □□
□□□□□	□	□	□	□	□ □□□□□
□ □□□□	□	□□	□	□	□□□□□□□□□□□□ □□

□ □□□□	□ □□□□□	□□□□	□□□□□□□□□	□□□□	□□□ □ □□
□□□□□	□	□	□	□□	□□□□□□□□□□□□ □□□

Gen1: Packet filtering

□ □ □ □ □	□ □ □ □ □ □	□ □ □ □	□ □ □ □ □ □ □	□ □ □ □	□ □ □ □	□ □ □ □ □ □
□ □ □ □	□	□	□	□ □		□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- ❖ Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side
- ❖ The ACK signifies that the packet is part of an ongoing conversation

Gen1: Packet filtering

❖ Advantages:

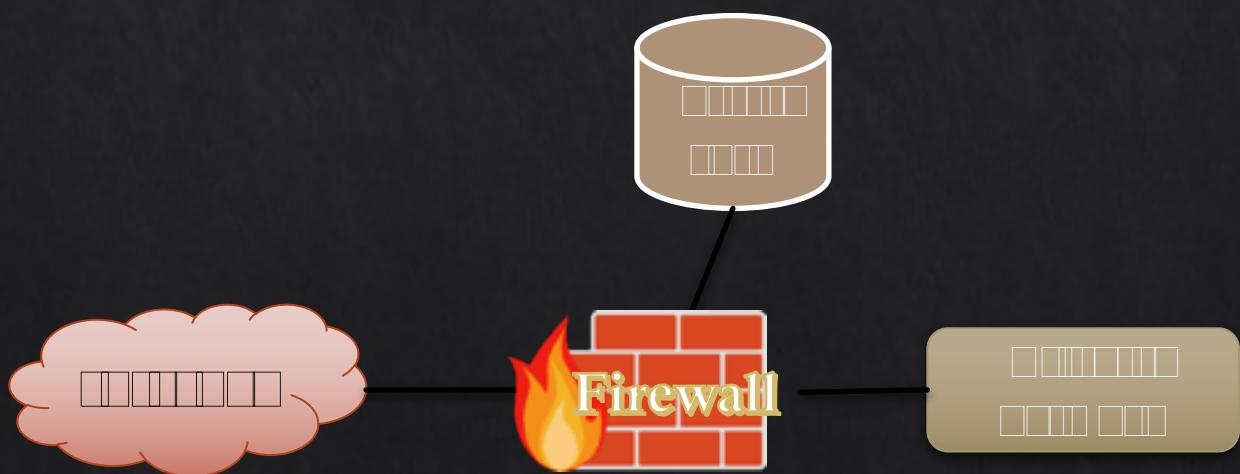
- ❖ Simple
- ❖ Typically transparent to users and are very fast

❖ 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

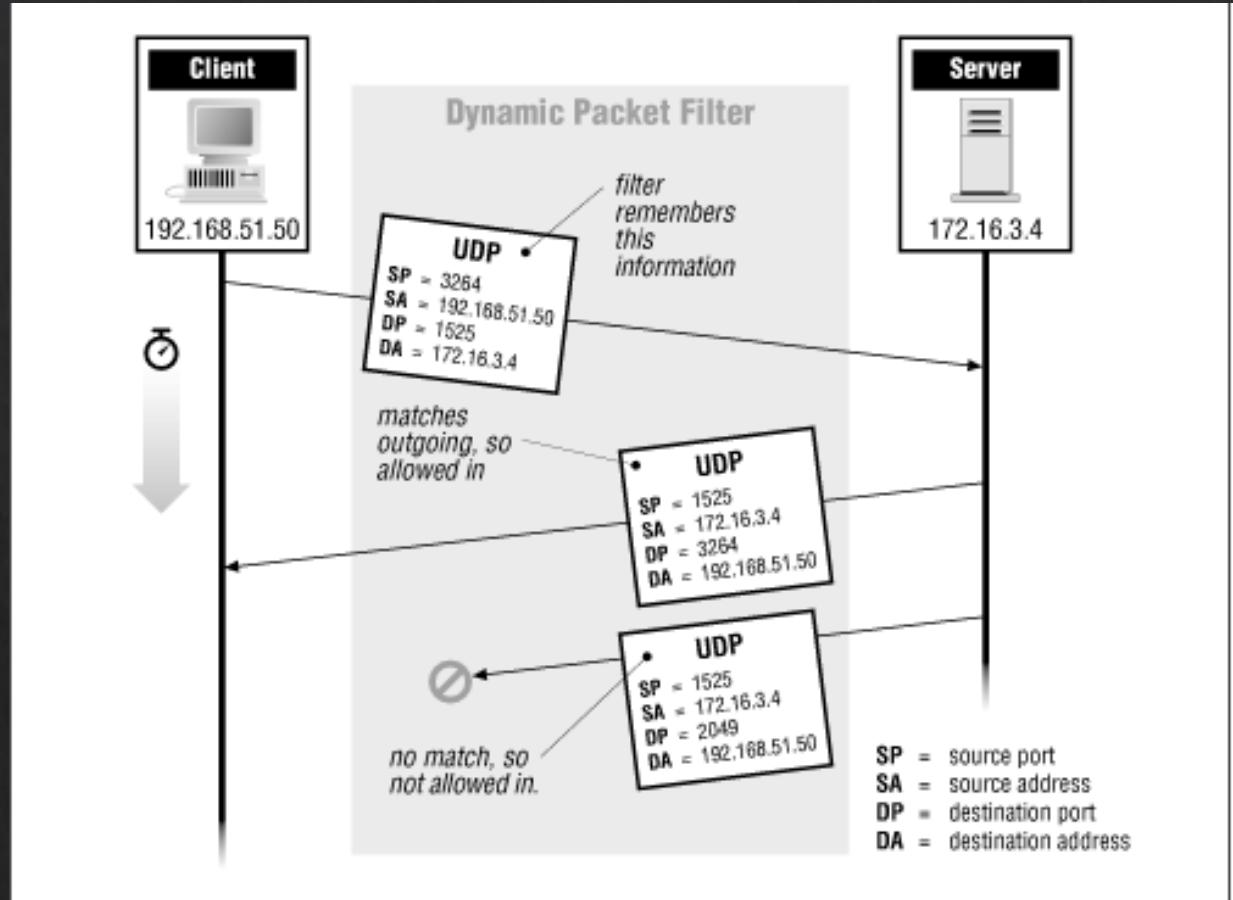
- ❖ Generation 1 firewalls do not examine higher layer context
- ❖ Generation 2 firewalls address this problem by examining each IP packet in context
 - ❖ Keeps the track of client-server sessions
 - ❖ Check each packet validity – e.g., does it belong to someone?
- ❖ Provides better capabilities to detect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



Gen2: Stateful inspection

- ❖ 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



- ◆ The firewall keeps track of currently established connections

Source IP	Source Port	Dest. IP	Dest. Port	Con. State
192.168.51.50	3264	172.16.3.4	1525	Established
172.16.3.4	1525	192.168.51.50	3264	Established
192.168.51.50	3264	172.16.3.4	2049	Established
172.16.3.4	1525	192.168.51.50	2049	Established

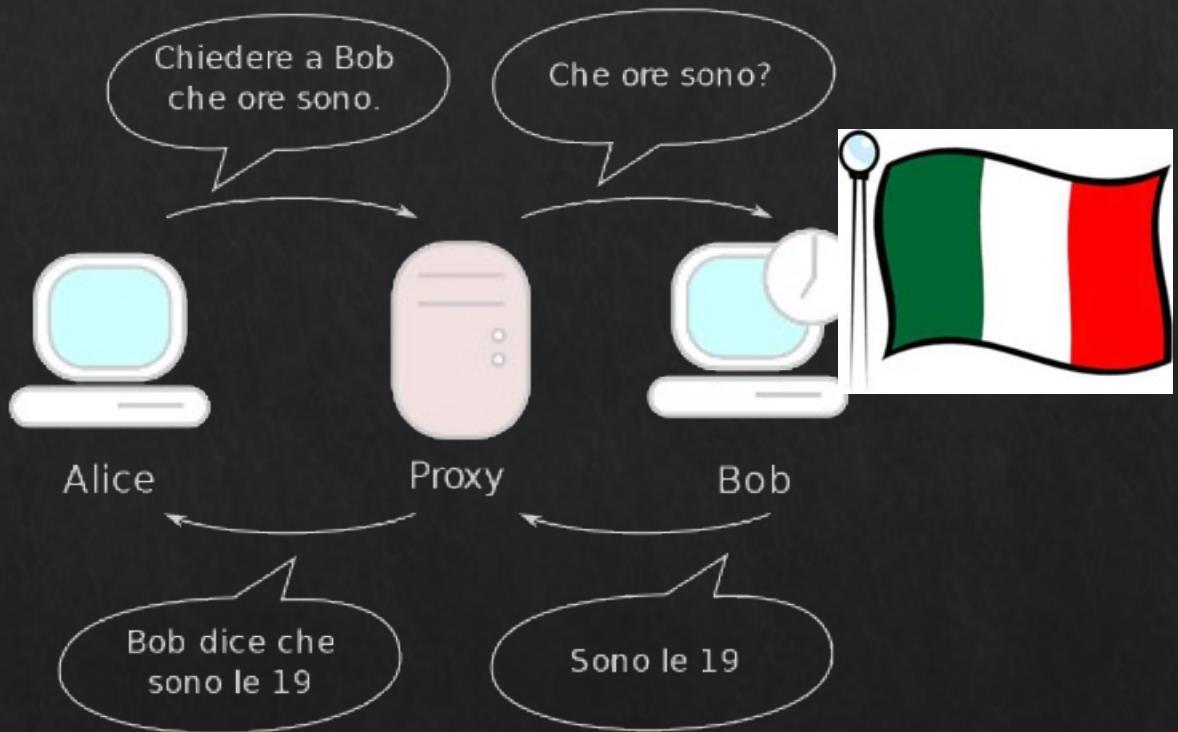
- ◆ The packet filter allows incoming traffic to high-numbered ports only for those packets that fit the profile

Gen2: Stateful inspection

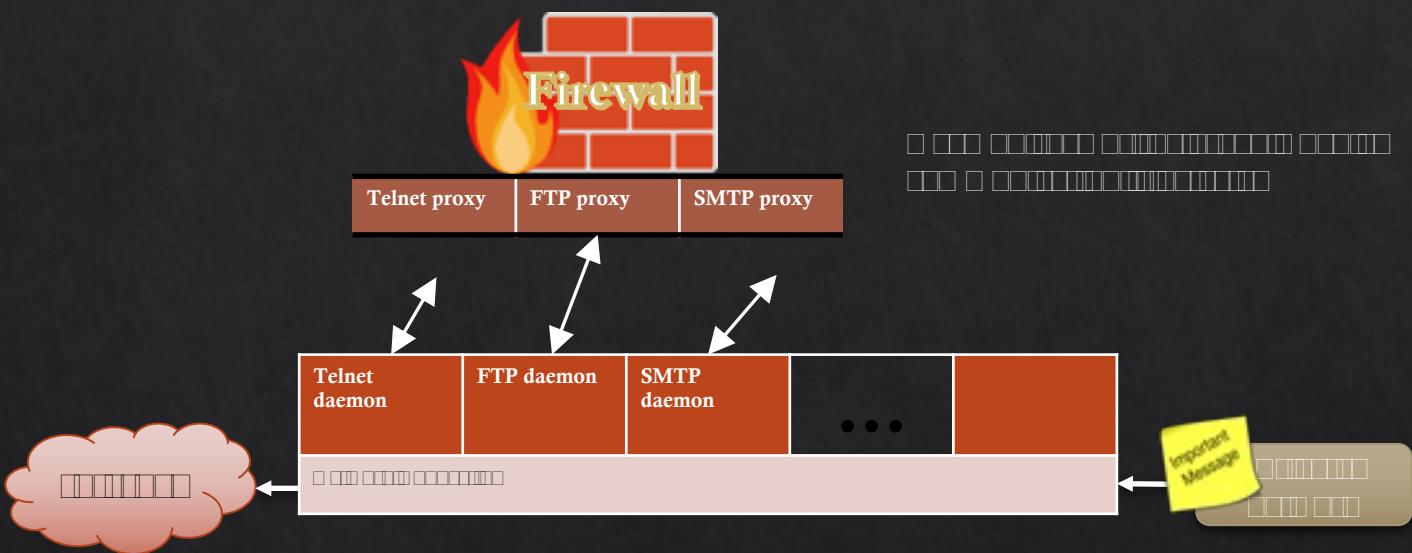
- ❖ Options to keep track of the TCP sequence numbers
 - ❖ To prevent attacks that depend on the sequence number
 - ❖ Q: What attack was it?
 - ❖ E.g., session hijacking
- ❖ Some may also inspect application data for well known protocols
 - ❖ e.g., FTP

Gen3: Application proxy

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



Gen3: Application proxy



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

- ❖ 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

The Goods

- □ □□□□□□ □ □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

The Bads

- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
• □□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
• □□□□□□□□□□□□□□□□□□

Other cool tools

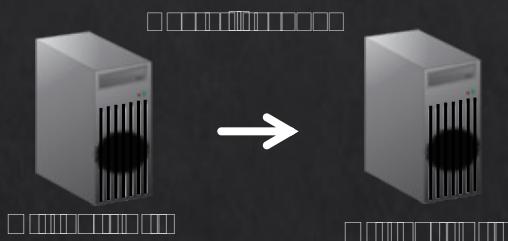
- ❖ There are many other tools that are useful for defence (as well as attacks), such as
 - ❖ Wireshark
 - ❖ Metasploit
 - ❖ Nessus
 - ❖ Snort
 - ❖ Burp
 - ❖ OpenVAS
 - ❖ Etc.

Moving Target Defense

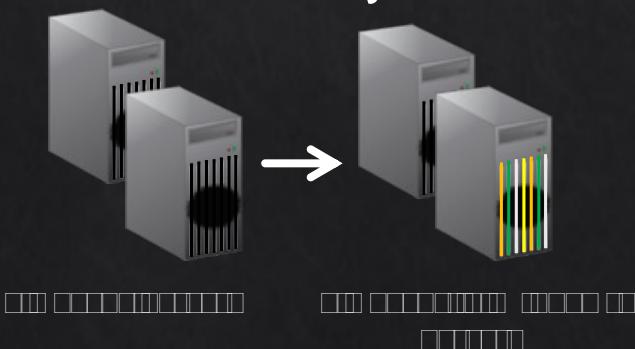


- ❖ Simply MTD, is a defense mechanism that continuously changes the attack surface to thwart cyberattacks
- ❖ Different types of changes can be made in the system, and they can be categorised into three

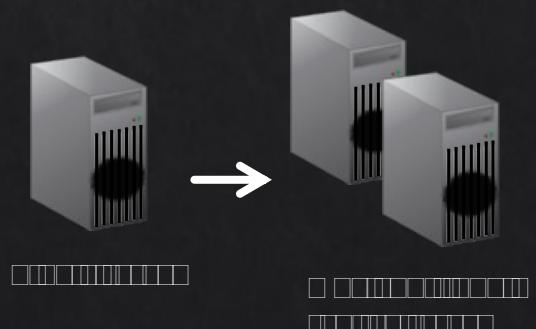
Shuffle



Diversity



Redundancy



MTD

❖ MTD can also be implemented in different layers of the system

❖ For example:

- ❖ Application layer – code regeneration/diversification
- ❖ Transport layer – routing node redundancy
- ❖ Internet layer – IP shuffling and virtualisation

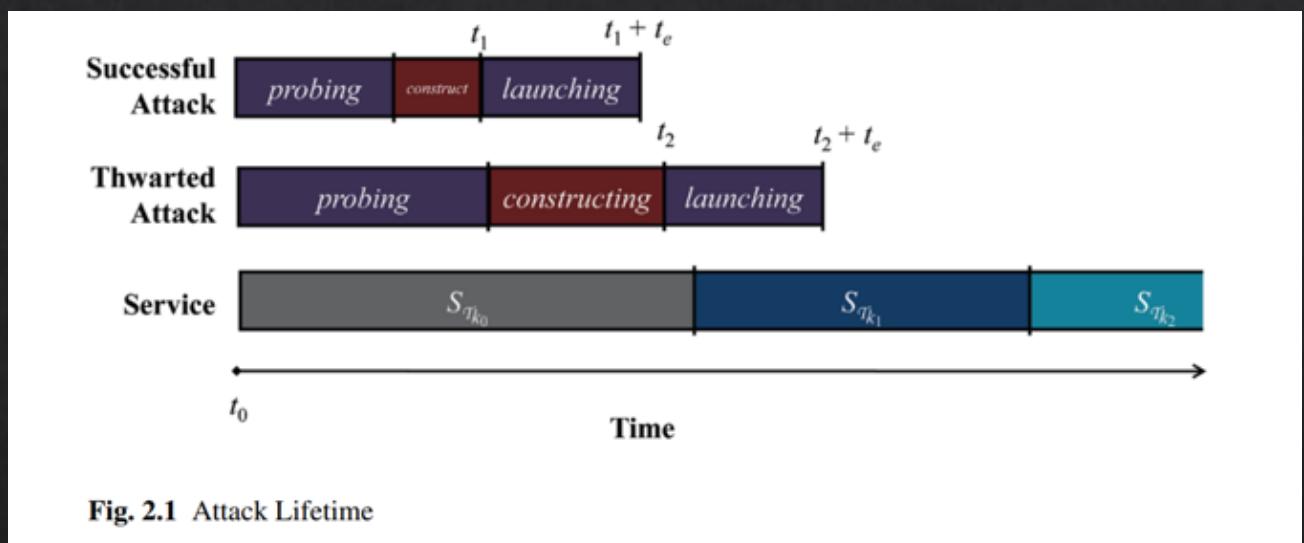


Fig. 2.1 Attack Lifetime

MTD Shuffle

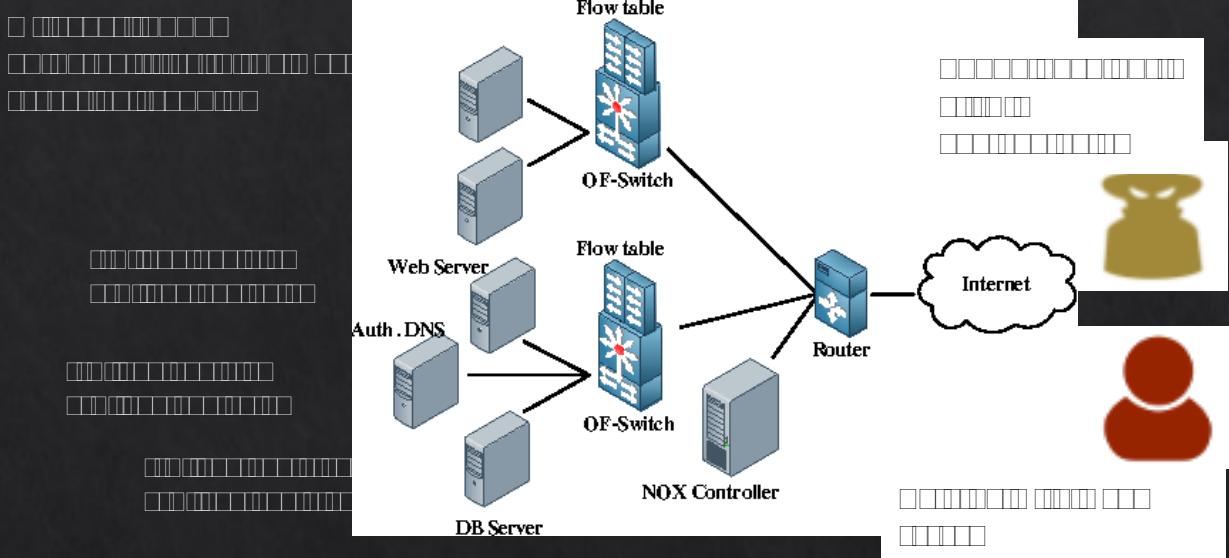
- ❖ The MTD shuffle technique **changes** the configuration of the system such that connections/dependencies are altered but the operations are maintained
 - ❖ E.g., changing the routing table in the SDN
- ❖ Existing vulnerabilities are **not removed**, but it redirects the attack sequence to be taken
 - ❖ Hence, delaying the reconnaissance process, as well as breaking an ongoing attack

MTD Shuffle

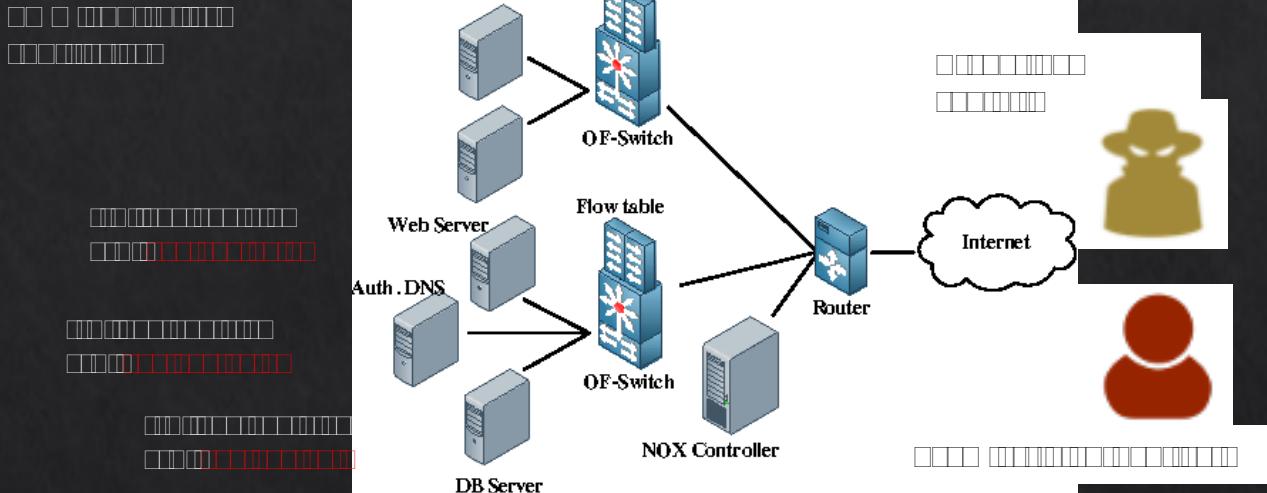
❖ Example: Openflow Random Host Mutation

- ❖ Typically, a host has one (physically) IP address
- ❖ In HF-RHM scheme, each host has a physical IP address and a virtual IP address
- ❖ Only the SDN controller knows the physical IP address of each host
- ❖ Virtual IP (vIP) addresses are used for establishing communication channels
- ❖ vIP addresses are shuffled every x seconds, OF switches are updated to redirect flows and update the existing communications
- ❖ Attackers probing for IP addresses loses the identified victim host as vIP shuffles

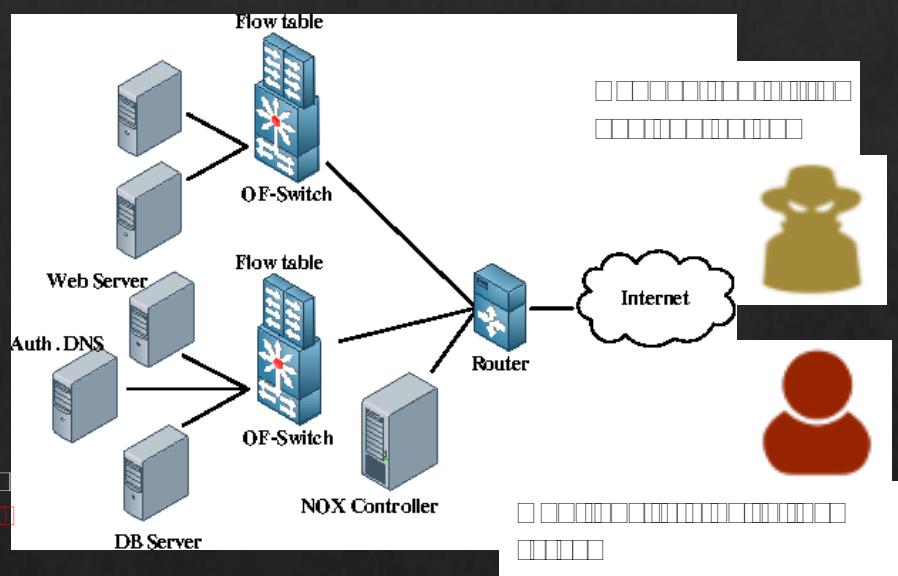
MTD Shuffle



MTD Shuffle



MTD Shuffle



MTD Shuffle

- ❖ What are some issues to consider using shuffle?
 - ❖ Shuffled system state may not be secure
 - ❖ Shuffle may violate security requirements
 - ❖ Shuffle may not be feasible
 - ❖ E.g., cannot satisfy performance requirements
 - ❖ Shuffle may not be applicable
 - ❖ E.g., homogeneous network/system
- ❖ How frequently should we shuffle?
 - ❖ Proactively, reactively, etc.

MTD Diversity

- ❖ Diversity technique provides replacement of an existing component/service, with **different implementation/configuration** that provides the **same functionalities**
- ❖ Existing vulnerabilities may be **replaced** with a new set of vulnerabilities
 - ❖ E.g., replaced the Windows-based computer to Mac

MTD Diversity

◆ Example: Compiler generated software diversity

- ◆ A version of software has a specific signature/binary
- ◆ There are number of instructions to carry out the software task
- ◆ These instructions can be remapped with other (single or combinations) instructions to do the same operations
 - ◆ Simply, $3+5$ is the same as $14-6$

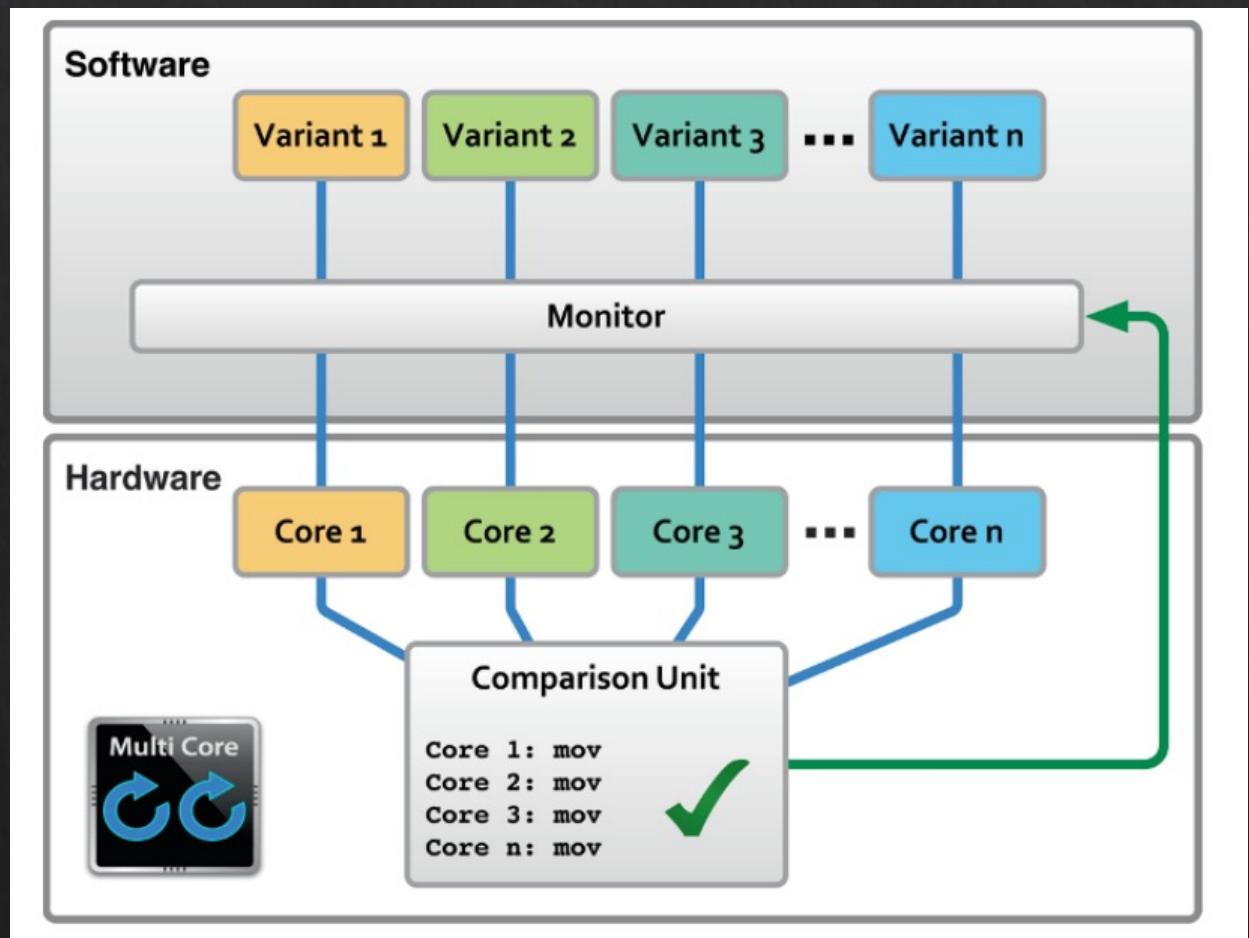
```
movl %edx, %eax  
xchgl %edx, %eax
```

can be replaced with

```
leal (%edx),  
%eax  
xchgl %edx, %eax
```

MTD Diversity

□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□
□□□ □□□□□□□□□□□□□□□□

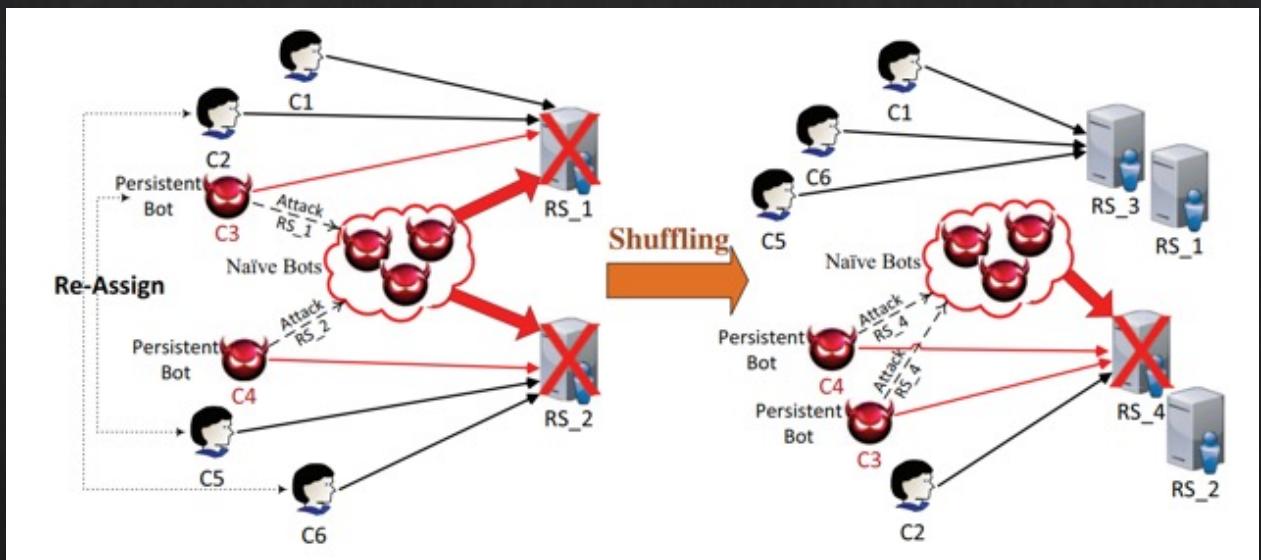


MTD Diversity

- ❖ What are some issues to consider using diversity?
 - ❖ New variant(s) may be more vulnerable
 - ❖ Difficult to generate variants
 - ❖ Cost to implement diversity technique
 - ❖ E.g., OS diversity in servers
 - ❖ Downtime is more significant than shuffle
 - ❖ E.g., rebooting vs SDN flow switching

MTD Redundancy

- ❖ Redundancy aims to ensure the availability of the system
- ❖ Replicas provide resources to handle high usage of the system
- ❖ It can be used in conjunction with other MTD techniques to provide defense against attackers
- ❖ For example DDoS attacks



MTD Redundancy

- ❖ By adding redundant servers and shuffling identified bots to the same server, we can ensure the **availability** of the cloud resources to legitimate users
- ❖ Problems with redundancy techniques include
 - ❖ Resource constraints
 - ❖ Ineffective against attacks violating confidentiality and integrity (but could be used for tolerance)
 - ❖ Degrading performance
 - ❖ Increasing redundancy does not proportionally increase availability

❖ Visualisation demo

MTD summary

- ❖ Different MTD techniques provide **different** approaches to ensure the **security objectives** of the system
- ❖ Still work to do in order to
 - ❖ Assess the effectiveness of MTD techniques
 - ❖ Models, metrics, methods, etc.
 - ❖ Combinations of MTD techniques
 - ❖ Applicable layers, compatibility etc.