



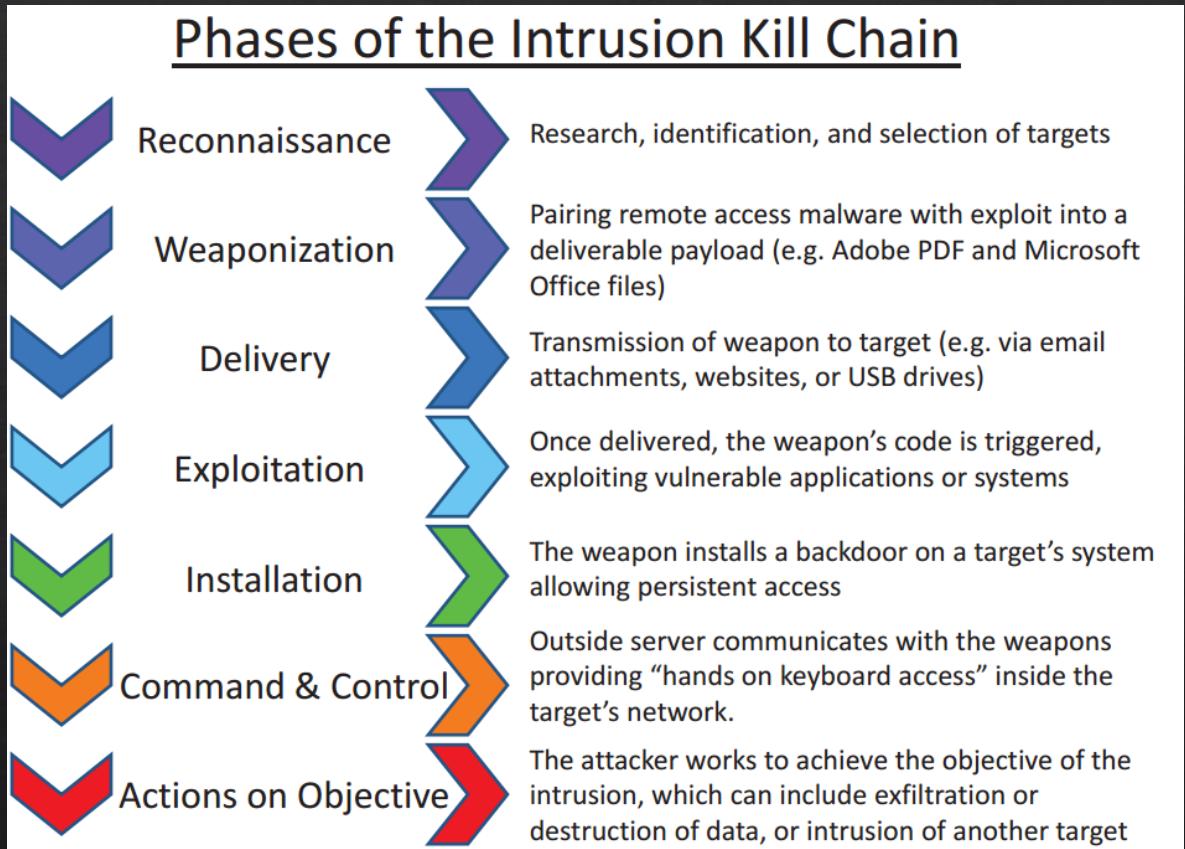
2. Reconnaissance

Jin Hong
jin.hong@uwa.edu.au

Cyber Kill Chain

- ❖ Before we talk about any exploits, we must first understand how attacks (usually) happen and progress.
- ❖ There are many approaches, but one of the most widely adopted approaches is *Cyber Kill Chain*
- ❖ It was developed by Lockheed Martin (Company) in 2011, specifying 7 high level goals.
- ❖ Naturally, *breaking* the kill chain and its methods became defense against such attacks.

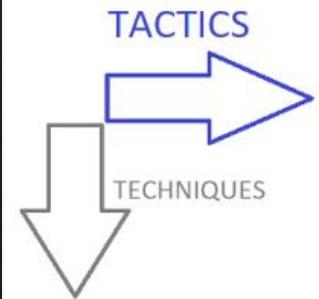
Cyber Kill Chain



MITRE ATT&CK

- ❖ MITRE ATT&CK is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations.
- ❖ This is the current industry standard and most used framework for understanding and communicating how attacks work.
- ❖ It goes a step further than the Cyber Kill Chain by expanding the attackers' high-level goals to 14 different tactics.

MITRE ATT&CK



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
9 techniques	4 techniques	9 techniques	4 techniques	19 techniques	11 techniques	9 techniques	2 techniques	17 techniques	8 techniques	4 techniques	10 techniques
Deliver Malicious App via Authorized App Store	Broadcast Receivers	Broadcast Receivers	Code Injection	Application Discovery	Access Notifications	Application Discovery	Attack PC via USB Connection	Access Calendar	Alternate Network Mediums	Alternate Network Mediums	Carrier Billing Fraud
Deliver Malicious App via Other Means	Command-Line Interface	Code Injection	Device Administrator Permissions	Code Injection	Access Sensitive Data in Device Logs	Evasion Analysis Environment	Exploit Enterprise Resources	Access Call Log	Commonly Used Port	Commonly Used Port	Clipboard Modification
Drive-by Compromise	Native Code	Comromise Application Executable	Exploit OS Vulnerability	Delete Device Data	Access Stored Application Data	File and Directory Discovery		Access Contact List	Domain Generation Algorithms	Data Encrypted	Data Encrypted for Impact
Exploit via Charging Station or PC	Scheduled Task/Job	Foreground Persistence	Exploit TEE Vulnerability	Device Lockout	Disguise Root/Jailbreak Indicators	Capture Clipboard Data	Location Tracking	Access Notifications	Remote File Copy	Standard Application Layer Protocol	Delete Device Data
Exploit via Radio Interfaces		Modify cached Executable Code		Download New Code at Runtime	Capture SMS Messages	Network Service Scanning		Access Sensitive Data in Device Logs	Standard Application Layer Protocol	Standard Cryptographic Protocol	Device Lockout
Install Insecure or Malicious Configuration		Modify OS Kernel or Boot Partition		Evasion Analysis Environment	Exploit TEE Vulnerability	Process Discovery		Access Stored Application Data	Uncommonly Used Port	Generate Fraudulent Advertising Revenue	Input Injection
Lockscreen Bypass		Modify System Partition		Geofencing	Input Capture	System Information Discovery		Capture Audio	Web Service	Manipulate App Store Rankings or Ratings	Manipulate System Partition
Masquerade as Legitimate Application		Modify Trusted Execution Environment		Input Injection	Input Prompt	Input Capture		Capture Camera		SMS Control	SMS Control
Supply Chain Compromise		Scheduled Task/Job		Install Insecure or Malicious Configuration	Keychain	System Network Configuration Discovery		Capture Clipboard Data			
				Masquerade as Legitimate Application	Network Traffic Capture or Redirection	System Network Connections Discovery		Capture SMS Messages			
				Modify OS Kernel or Boot Partition	URI Hijacking			Data from Local System			
				Modify System Partition				Foreground Persistence			
				Modify Trusted Execution Environment				Input Capture			
				Native Code				Location Tracking			
				Obfuscated Files or Information				Network Information Discovery			
				Proxy Through Victim				Network Traffic Capture or Redirection			
				Suppress Application Icon				Screen Capture			
				Uninstall Malicious Application							

CKC & MITRE ATT&CK

- ❖ Both frameworks capture how attacks progress.
- ❖ Lots of common approaches in the framework.
- ❖ We will start from the beginning - Reconnaissance

Reconnaissance

“Reconnaissance consists of techniques that involve adversaries actively or passively **gathering information** that can be used to support targeting. Such information may include details of the victim organization, infrastructure, or staff/personnel.”

- MITRE Corp

Reconnaissance

- ◊ Active scanning
 - ◊ Scan IP
 - ◊ Scan Vulnerability
 - ◊ Crawling
- ◊ Gather Target Information
 - ◊ Hardware
 - ◊ Software
 - ◊ Firmware
 - ◊ Configurations
- ◊ Gather Target Identity Information
 - ◊ Credentials
 - ◊ Emails
 - ◊ Names
- ◊ Gather Network Information
 - ◊ Domain
 - ◊ DNS
 - ◊ Trust dependencies
 - ◊ Topology
 - ◊ IP addresses
 - ◊ Security appliances
- ◊ Gather organization information
 - ◊ Location
 - ◊ Business relationships
 - ◊ Business tempo
 - ◊ Roles
- ◊ Phishing
 - ◊ Service
 - ◊ Attachments
 - ◊ Links

Etc.

See more details at <https://attack.mitre.org/tactics/TA0043/>

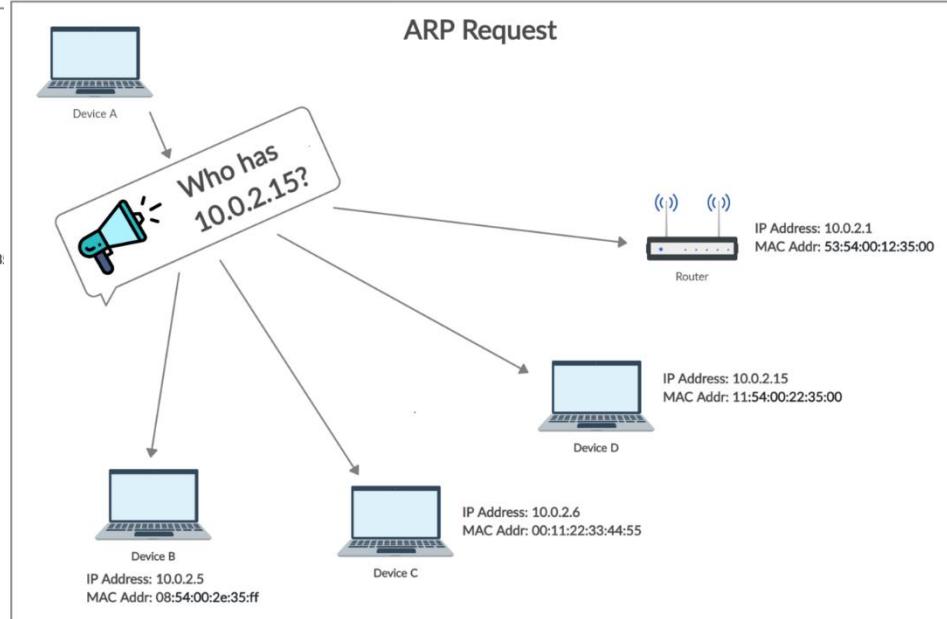
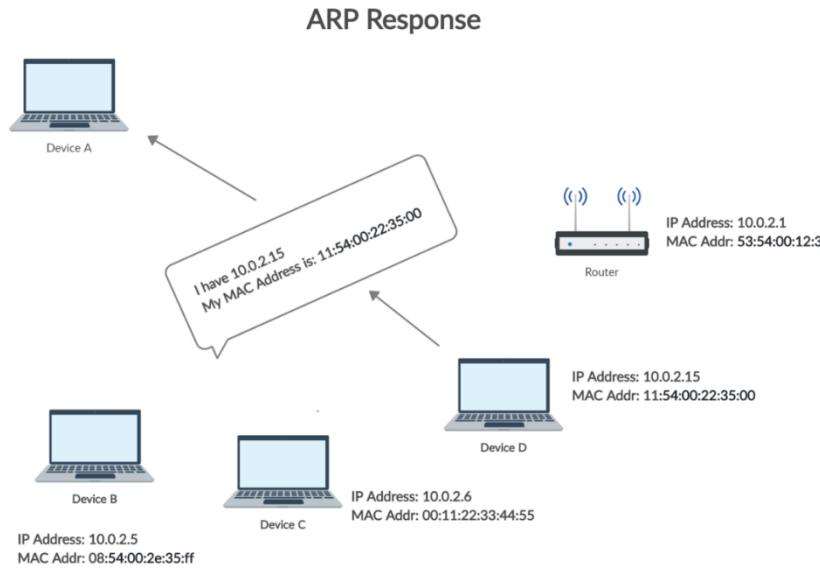
Reconnaissance

- ❖ The very first thing we will try is to discover network hosts.
 - ❖ We will look at IP scanning and Port scanning (configuration)
- ❖ We will write the code to do this.
 - ❖ Don't worry, you are provided with the code.

Network Scanner

- ❖ We can use ARP (Address Resolution Protocol) to discover hosts in the network, which translates IP to MAC addresses.
- ❖ We are accustomed to use IP addresses to communicate, but computers use MAC address to communicate – we can use this fact to discover live IP addresses using ARP.
- ❖ By checking IP via ARP, we know the host at the IP address exists if we get a reply.

Network Scanner





READ: Any knowledge and techniques presented here are for your learning purposes only. It is **ABSOLUTELY ILLEGAL** to apply the learned knowledge to others without proper consent/permission, and even then, you must check and comply with any regulatory restrictions and laws.



Network Scanner

Demo

Network Scanner

- ❖ It really is this simple!

```
1 import scapy.all as scapy
2 import argparse
3
4 def get_args():
5     parser = argparse.ArgumentParser()
6     parser.add_argument('-t', '--target', dest='target', help='Target IP Address/Adresses')
7     options = parser.parse_args()
8
9     #Check for errors i.e if the user does not specify the target IP Address
10    #Quit the program if the argument is missing
11    #While quitting also display an error message
12    if not options.target:
13        #Code to handle if interface is not specified
14        parser.error("[-] Please specify an IP Address or Addresses, use --help for more information")
15    return options
16
17 def scan(ip):
18    arp_req_frame = scapy.ARP(pdst = ip)
19    broadcast_ether_frame = scapy.Ether(dst = "ff:ff:ff:ff:ff:ff")
20    broadcast_ether_arp_req_frame = broadcast_ether_frame / arp_req_frame
21
22    answered_list = scapy.srp(broadcast_ether_arp_req_frame, timeout = 1, verbose = False)
23    result = []
24    for i in range(0,len(answered_list)):
25        client_dict = {"ip" : answered_list[i][1].psrc, "mac" : answered_list[i][1].hwsrc}
26        result.append(client_dict)
27    return result
28
29 def display_result(result):
30    print("-----\nIP Address\tMAC Address\n-----")
31    for i in result:
32        print("{}\t{}".format(i["ip"], i["mac"]))
33
34
35 options = get_args()
36 scanned_output = scan(options.target)
37 display_result(scanned_output)
```

Network Scanner

- ❖ We can scan and view other hosts in the network!

```
(jin㉿kali)-[~/cits3006/lecture1]
$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:73:0d:97:14 txqueuelen 0 (Ethernet)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 0 bytes 0 (0.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.67.3 netmask 255.255.255.0 broadcast 192.168.67.255
        ether 2e:06:ec:ba:8f:92 txqueuelen 1000 (Ethernet)
          RX packets 7 bytes 1326 (1.2 KiB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 22 bytes 2551 (2.4 KiB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 0 bytes 0 (0.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
(jin㉿kali)-[~/cits3006/lecture1]
$ sudo python3 network_scanner.py -t 192.168.67.0/24
[sudo] password for jin:
```

IP Address	MAC Address
192.168.67.1	be:d0:74:b2:00:64
192.168.67.6	e6:04:a6:2f:3c:54

Port Scanner

- ❖ Once we figure out the host we would like to exploit, the next step is to discover what services it has running
- ❖ Services expose selected port(s) to provide connections to others
- ❖ We can check whether the port is open or not using *socket* programming



Port Scanner

Demo

Port Scanner

- ❖ The socket function call takes the IP address to initialize the socket connection
- ❖ Once the port is specified, the connection attempt is made
- ❖ A response 0 indicates the port is open (i.e., can connect)

```
1 from socket import *
2 import argparse
3
4 def get_args():
5     parser = argparse.ArgumentParser()
6     parser.add_argument('-t', '--target', dest='target', help='Target IP Address')
7     options = parser.parse_args()
8
9     #Check for errors i.e if the user does not specify the target IP Address
10    #Quit the program if the argument is missing
11    #While quitting also display an error message
12    if not options.target:
13        #Code to handle if interface is not specified
14        parser.error("[-] Please specify an IP Address or Addresses, use --help")
15    return options
16
17 def main():
18    if __name__ == '__main__':
19        target = get_args().target
20        t_IP = gethostbyname(target)
21        print ('Starting scan on host: ', t_IP)
22
23        for i in range(1, 1024):
24            s = socket(AF_INET, SOCK_STREAM)
25
26            conn = s.connect_ex((t_IP, i))
27            if(conn == 0) :
28                print ('Port %d: OPEN' % (i,))
29            s.close()
30
31 main()
```

Port Scanner

- ◊ We see that bunch of ports are open!
- ◊ We see familiar ports open:
 - ◊ FTP (21)
 - ◊ SSH (22)
 - ◊ Telnet (23)
 - ◊ SMTP (25)
 - ◊ DNS (53)
 - ◊ HTTP (80)

```
(jin㉿kali)-[~/cits3006/lect1]
$ ls
network_scanner.py  port_scanner.py
```

```
(jin㉿kali)-[~/cits3006/lect1]
$ sudo python3 port_scanner.py -t 192.168.67.6
Starting scan on host: 192.168.67.6
Port 21: OPEN
Port 22: OPEN
Port 23: OPEN
Port 25: OPEN
Port 53: OPEN
Port 80: OPEN
Port 111: OPEN
Port 139: OPEN
Port 445: OPEN
Port 512: OPEN
Port 513: OPEN
Port 514: OPEN
```

```
(jin㉿kali)-[~/cits3006/lect1]
$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
          inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
              ether 02:42:88:15:98:61  txqueuelen 0  (Ethernet)
                  RX packets 0  bytes 0 (0.0 B)
                  RX errors 0  dropped 0  overruns 0  frame 0
                  TX packets 0  bytes 0 (0.0 B)
                  TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
     inet 192.168.67.7  netmask 255.255.255.0  broadcast 192.168.67.255
          ether fe80::b0d2:91ff:fe43:86e5  txqueuelen 0  (link-layer)
              link-layer MTU: 1500  queueing discipline: pfifo_fast
              brd 192.168.67.255  scopeid 0x20<link>
              inet 192.168.67.7  netmask 255.255.255.0  broadcast 192.168.67.255
                  brd 192.168.67.255  scopeid 0x10<inet>
                  link-laddr fe80::b0d2:91ff:fe43:86e5  brd fe80::ff:fe43:86e5  scopeid 0x20<inet>
                  inet6 fe80::b0d2:91ff:fe43:86e5  prefixlen 64  scopeid 0x20<inet>
                      link-laddr fe80::b0d2:91ff:fe43:86e5  brd fe80::ff:fe43:86e5  scopeid 0x20<inet>
```

More will be
explored in
Lab 1!