2. Reconnaissance

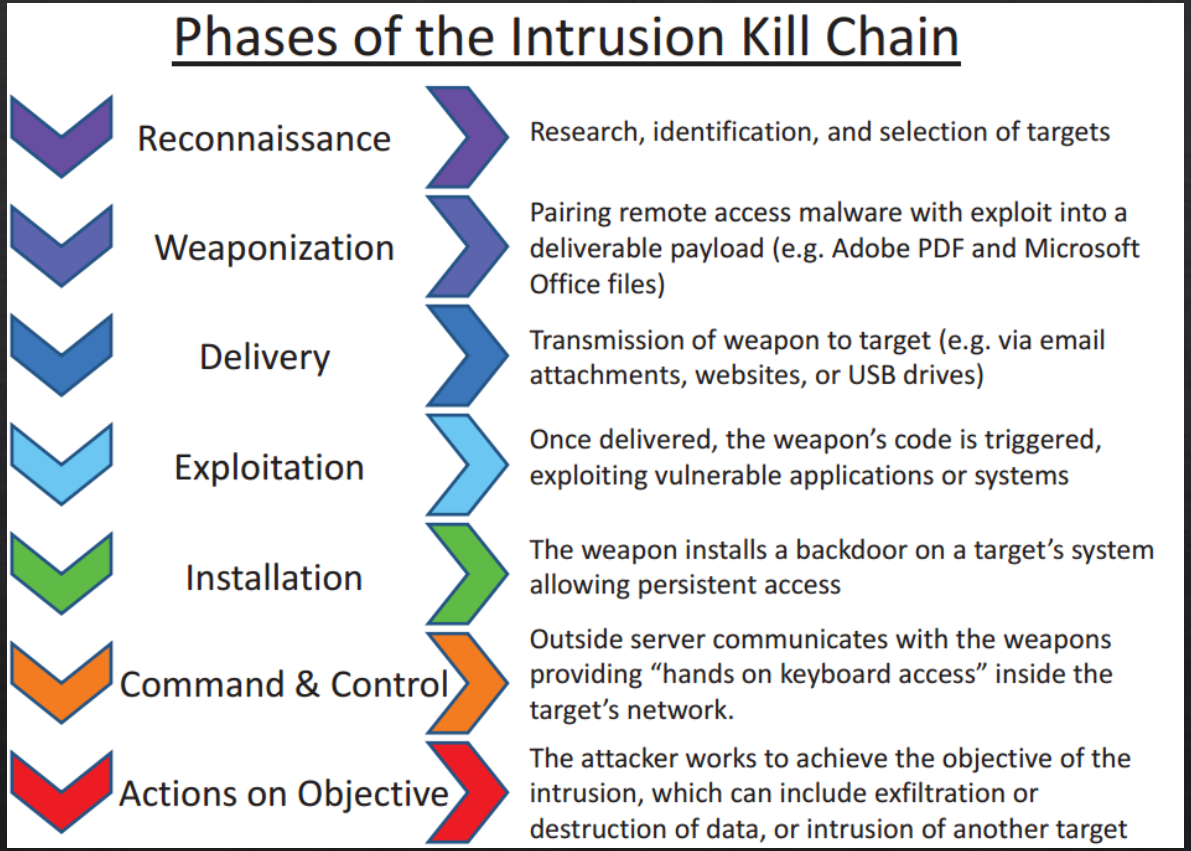
Jin Hong

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.

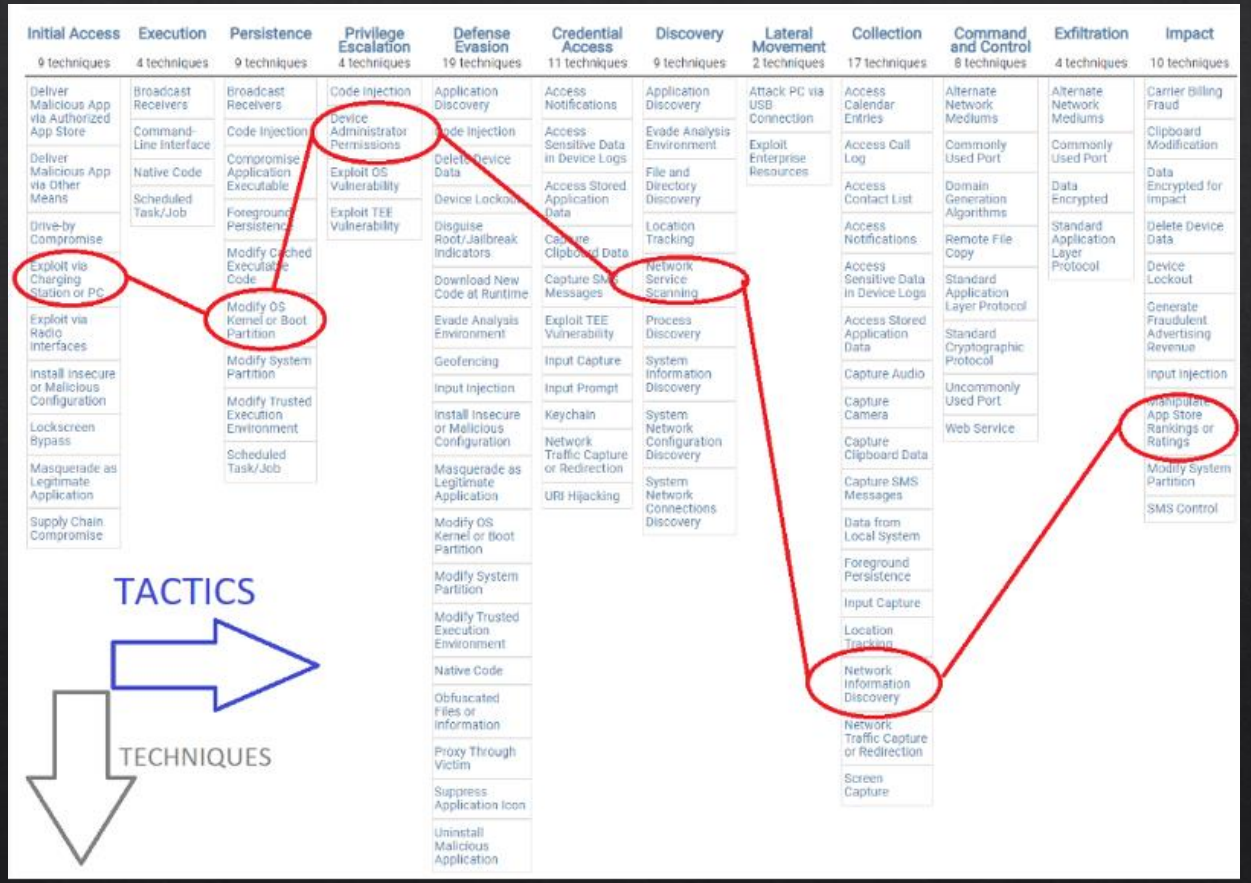
Cyber Kill Chain



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

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

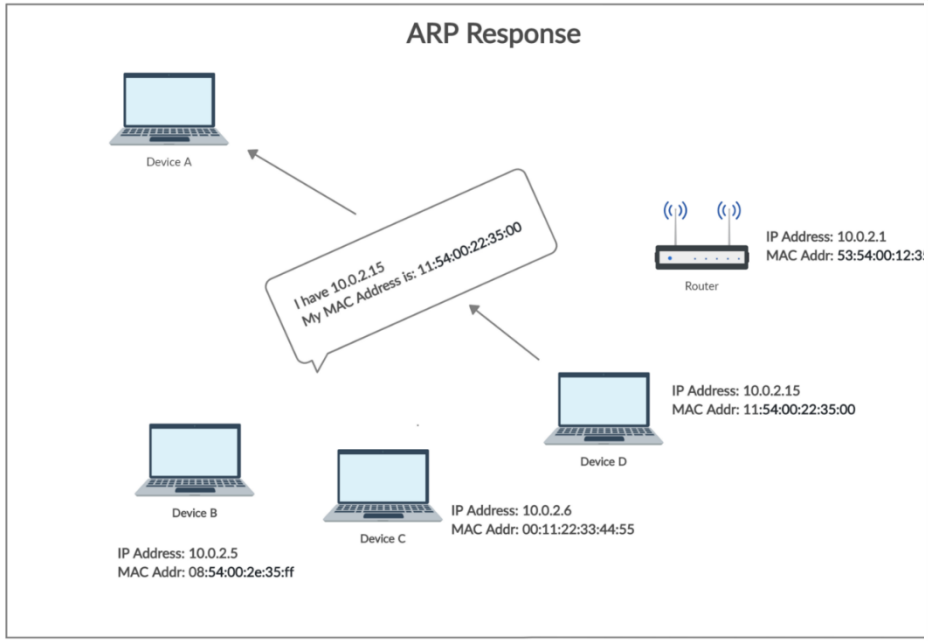
- The very first thing we will try is to discover network hosts.
  - We will look at IP scanning and Port scanning (configuration)
- We will write the code to do this.
  - Don’t worry, you are provided with the code.

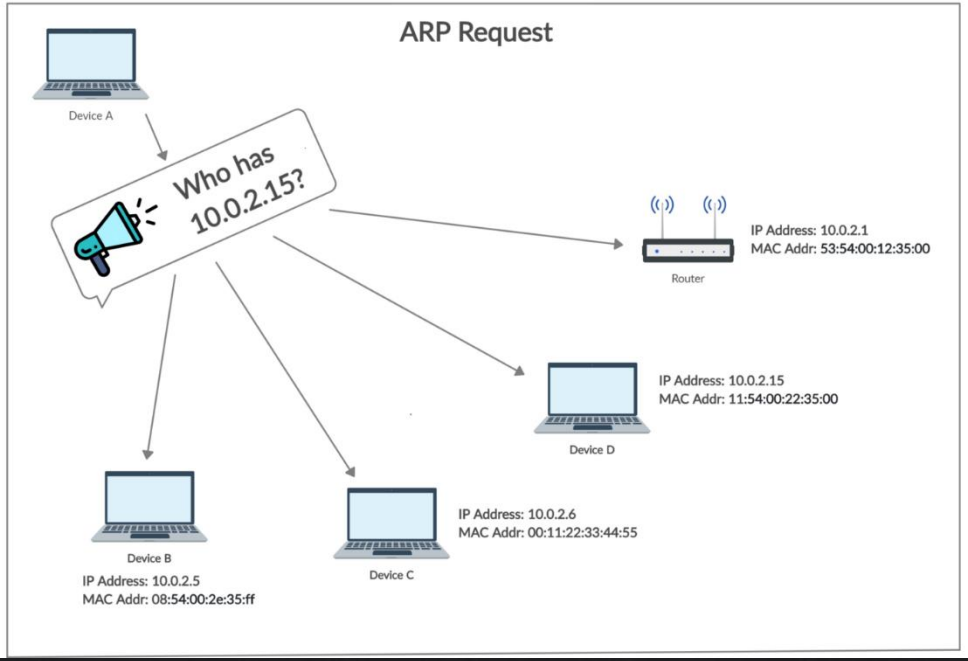
Network Scanner

- 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.

https://raw.githubusercontent.com/uwacyber/cits3006/2022s2/cits3006-labs/files/network_scanner.py

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

- It really is this simple!

```python
import scapy.all as scapy
import argparse

def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--target', dest='target', help='Target IP Address/Adresses'
    options = parser.parse_args()

    #Check for errors i.e if the user does not specify the target IP Address
    #Quit the program if the argument is missing
    #While quitting also display an error message
    if not options.target:
        #Code to handle if interface is not specified
        parser.error("[-] Please specify an IP Address or Addresses, use --help for more i
    return options

def scan(ip):
    arp_req_frame = scapy.ARP(pdst = ip)
    broadcast_ether_frame = scapy.Ether(dst = "ff:ff:ff:ff:ff:ff")
    broadcast_ether_arp_req_frame = broadcast_ether_frame / arp_req_frame

    answered_list = scapy.srp(broadcast_ether_arp_req_frame, timeout = 1, verbose = False)
    result = []
    for i in range(0,len(answered_list)):
        client_dict = {"ip" : answered_list[i][1].psrc, "mac" : answered_list[i][1].hwsrc}
        result.append(client_dict)
    return result

def display_result(result):
    print("-----------------------------------\nIP Address\tMAC Address\n-----------------
    for i in result:
        print("{}\t{}".format(i["ip"], i["mac"]))


options = get_args()
scanned_output = scan(options.target)
display_result(scanned_output)
```

Network Scanner

Demo

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

https://raw.githubusercontent.com/uwacyber/cits3006/2022s2/cits3006-labs/files/port_scanner.py

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 Addres
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

Demo

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 192.168.67.2
        inet6 fe80::b0d2:91ff:fe43:86e5  prefixlen 64  scopeid 0x20<link
```

More will be explored in Lab 1!