COMP 5970/6970: Intrusion Detection

Alex Lewin, Marvin Bell, Lakshmi Krishnaprasad

12/12/19

1 Executive Summary

The purpose of this project was to develop and demonstrate our understanding of the basics of Intrusion Detection Programs (IDS), the different types of IDS, and the process of creating a proprietary IDS.

Our assignment was to pose as member of a company's *"blue team"*, protecting our company against malware. To do this, we designed and tested a proprietary IDS that detects a variety of attacks.

Our script, that utilizes *signature* and *heuristic* based itrusion detection, does successfully detects a variety of attacks.

2 Problem Description

2.1 Problem Overview

Our assignment was to pose as member of a company's *"blue team"*, protecting our company against malware. To do this, we were instructed to design and test a proprietary IDS that detects utilizes at least **two of the following four types of network IDS**.

- **Behavioral**
- **Anomaly**
- **Signature**
- **Heuristic**

Our script must detect the following attack tools being run against the network:

- **NMAP - SYN, ACK, and Christmas Scans**
- **Ettercap**
- **Responder**
- **Metasploit - CVE-2017-010 - ms17_010_psexec**

2.2 Technical Specifications

Attacker machine specifications:

- Operating System: **Kali Linux**
- Server Location: **Shelby 2129**
- IP Address: **192.168.x.10**

3 Information Discovery

In order to create a program that detects specific attacks, we conducted some exploratory tests. For each script, we ran the attack with Wireshark in the background so we could analyze the unique aspects of the attack with the goal of reverse engineering a detection scheme.

3.1 NMAP

To learn how to detect NMAP scans on the network, we first ran the different types of scans with Wireshark performing packet capture in the background. While we approached the detection of each NMAP scan with the same strategy, we had to use slightly different checks for each scan.

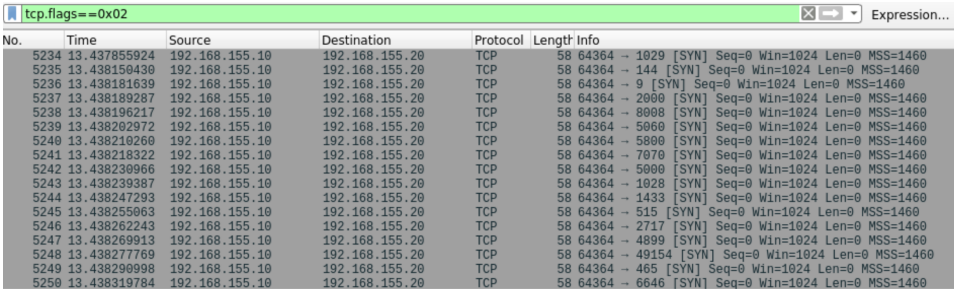
3.1.1 SYN Scan

Because of the frequency of legitimate TCP SYN packets, we had to find a distinguishing factor of NMAP SYN scan packets. From online research, we discovered that NMAP SYN scan packets always have a standard window size: 1024, 2048, 3072, or 4096.

From the information about the window size and given that NMAP SYN scan packets have the FIN flag set, we can distinguish a SYN scan packet *by signature* using the following Wireshark flags:

```
tcp and tcp.flags == 0x02 and (tcp.window_size==1024 or tcp.window_size==2048
or tcp.window_size==3072 or tcp.window_size==4096)
```

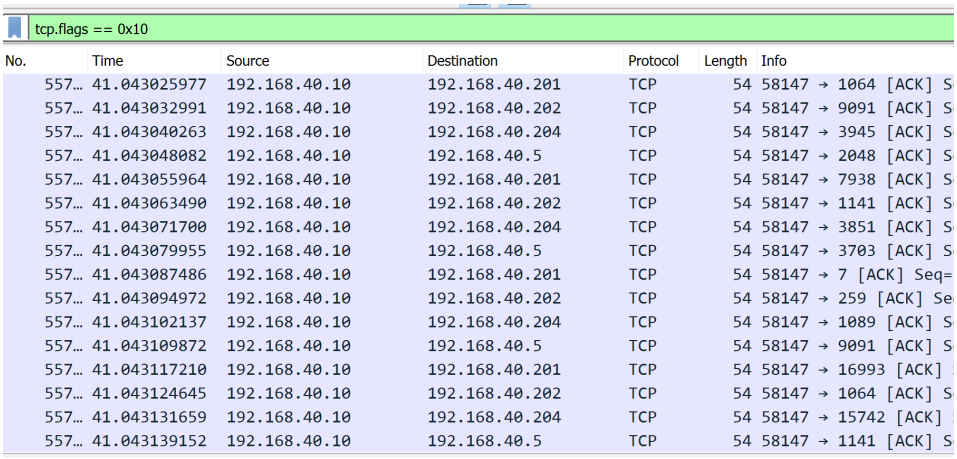
In addition, we utilized a *heuristic* solution to detect NMAP SYN scan traffic. Knowing that an SYN scan generally sends packets to a large number of destination ports, we can detect an SYN scan by keeping track of the source and destination ports of the packets.



3.1.2 ACK Scan

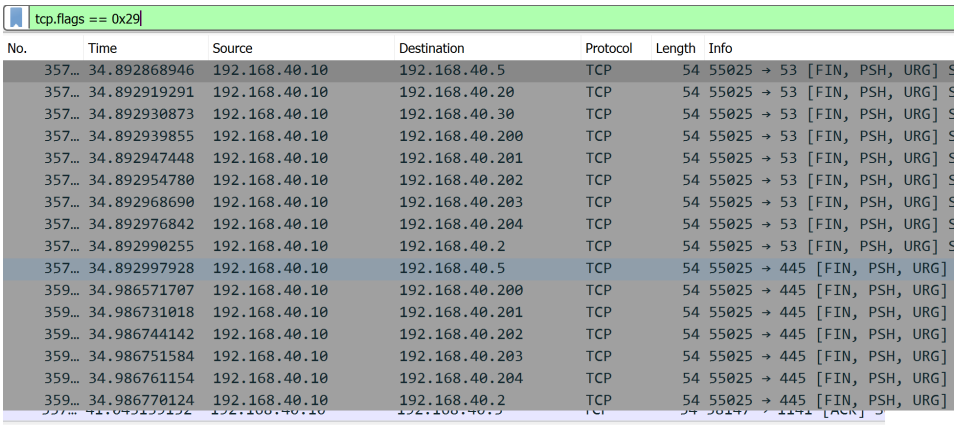
Because of the frequency of legitimate TCP ACK packets, we utilized a *heuristic* solution to detect NMAP ACK scan traffic. Knowing that an ACK scan generally sends packets to a large number of destination ports, we can detect an ACK scan by keeping track of the source and destination ports of the packets. We use this Wireshark filter to detect ACK packets *by signature*:

```
tcp and tcp.flags==0x10
```



3.1.3 XMAS Scan

Because XMAS scans utilize several TCP flags in an otherwise uncommon combination, we can detect XMAS scan packets by simply checking the size of the TCP flags and which flags are currently set (FIN, PSH, URG): `tcp and tcp.flags==0x29`



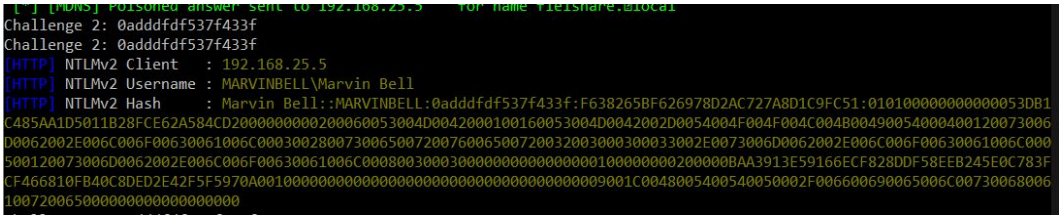
3.2 Ettercap

To learn how to detect a man-in-the-middle attack from Ettercap, we first ran the attack on our network while monitoring the traffic using Wireshark. Since PyShark does not have any support for ARP traffic, we decided to use the python module *Scapy* instead. To detect if an ARP Cache Spoof has occurred, we checked if the reported source mac address in each packet, matched the true mac address.

3.3 Responder

Usually, DNS is used to resolve domain names. When DNS fails, Windows uses LLMNR (Link-Local Multicast Name Resolution) to resolve names. If even LLMNR fails, NBNS (NetBIO Name Service) - which uses UDP - is used for name resolution.

Responder takes advantage of this by listening for NBNS and LLMNR queries. Responder then responds to those queries, claiming to be the domain the victim is searching for. We can detect Responder attacks by listening for a NBNS query followed immediately by a response from a valid IP. This strategy utilizes *signature* detection.



3.4 Metasploit - CVE-2017-010 - ms17_010_psexec

When learning how to detect the metasploit Eternal Blue attack, we first ran an attack on our our network while monitoring and recording the traffic using Wireshark. On closer research of what the Eternal Blue exploit enacts, we monitored the traffic for repeated NT requests/ responses over the network.

The large amount of requests sets up the SMB for a specific packet that will exploit system. The large NT Trans requests lead into Secondary Trans Requests that act as a launcher for the malware on the remote machine. This packet may show as a malformed packet. Successful or in-progress requests and responses for this type of attack will have the Multiplex ID's consisting 82, 81, 65, and 64 as well checking for nt_value for a successful connection.

The following searches for these conditions:

```
smb.mid == 65 and smb.nt_status == 0
```

4 Code Explanation

4.1 NMAP Detection

```
#To detect an ACK and SYN Scan, we store a set of witnessed
# destination ports for each source port.
ack_ports = defaultdict(set()) #src port -> dst port (ack scan)
syn_ports = defaultdict(set()) #src port -> dst port (syn scan)

#Performing a live capture over eth0
capture = pyshark.LiveCapture(interface='eth0')
for packet in capture.sniff_continuously():

  #Check for SYN Scan as described in section 3.1.1
  if packet.tcp and packet.tcp.flags.fin == 1
      and (packet.tcp.window_size==1024 or packet.tcp.window_size==2048 or
          packet.tcp.window_size==3072 or packet.tcp.window_size==4096):

    #Maps seen destination ports to each source port.
    # Key: source port ==> Value: set of destination ports from this source
    syn_ports[packet.tcp.srcport].add(packet.tcp.dstport)


    #Trigger detection if more than 15 destination ports
    # are sent from the same source port.
    if len(syn_ports[packet.tcp.srcport]) > 15:

      #Print source ip
      print("NMAP SYN SCAN DETECTED")
      print("Attacker Machine:", str(packet.ip.src))


  #Check for XMAS Scan as described in section 3.1.3
  elif packet.tcp and packet.tcp.flags==0x2:
    print('NMAP XMAS SCAN DETECTED')

    #Print source ip
    print("Attacker Machine:", str(packet.ip.src))


  #For ACK Scan, we only need to analyze ACK packets
  elif packet.tcp and packet.tcp.flags==0x10:

    #Maps seen destination ports to each source port.
    # Key: source port ==> Value: set of destination ports from this source
    ack_ports[packet.tcp.srcport].add(packet.tcp.dstport)
```

```
    #Trigger detection if more than 15 destination ports
    # are sent from the same source port.
    if len(ack_ports[packet.tcp.srcport]) > 15:
      print('NMAP ACK SCAN DETECTED')
      print("Attacker Machine:", str(packet.ip.src))
```

4.2 Ettercap Detection

Ettercap is a piece of software that can facilitate **Man-in-the-Middle** attacks. In order to accomplish this, Ettercap first performs **ARP Cache Poisoning** which is what we decided to detect.

Since Wireshark does not implement a way to directly filter for ARP traffic, we opted to use the python module, **scapy**, instead.

```
from scapy.all import Ether, ARP, srp, sniff, conf

#Helper function to find mac address
def get_mac(ip):
  #Returns true mac address of ip
  #Throws IndexError if blocked
  p = Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip)

  #Traces true mac address
  result = srp(p, timeout=3, verbose=False)[0]

  #Returns true mac address of ip
  return result[0][1].hwsrc

def checker(pkt):
  #Filters for ARP responses only
  if not pkt.haslayer(ARP) or not pkt[ARP].op == 2:
    return
  try:
    #calls helper function for finding mac
    real_mac = get_mac(pkt[ARP].psrc)
    #gets mac address from packet
    response_mac = pkt[ARP].hwsrc

    #if they don't match, arp
    #cache poisoning is occurring
    if real_mac != response_mac:
      print('ARP Cache Spoof Detected')
      print('Real mac addr: ', str(real_mac.upper()))
      print('Fake mac addr: ', str(response_mac.upper()))
      return
```

```
  except IndexError:
    print('indexerror')
    pass

#Passes every sniffed packed to 'checker'
sniff(iface='Ethernet 10', prn=checker, filter='arp', store=False)
```

4.3 Responder Detection

```
import pyshark


# Start live capture (change interface when needed)
cap = pyshark.LiveCapture(interface='eth0', display_filter='nbns')

# Find when Responder intrudes communication by finding its signature name "RESPPROXYSRV"
for packet in cap.sniff_continuously():
    if packet.nbns.name.find('RESPPROXYSRV'):
        print("RESPONDER ATTACK DETECTED!!")
        break
```

4.4 Metasploit Detection

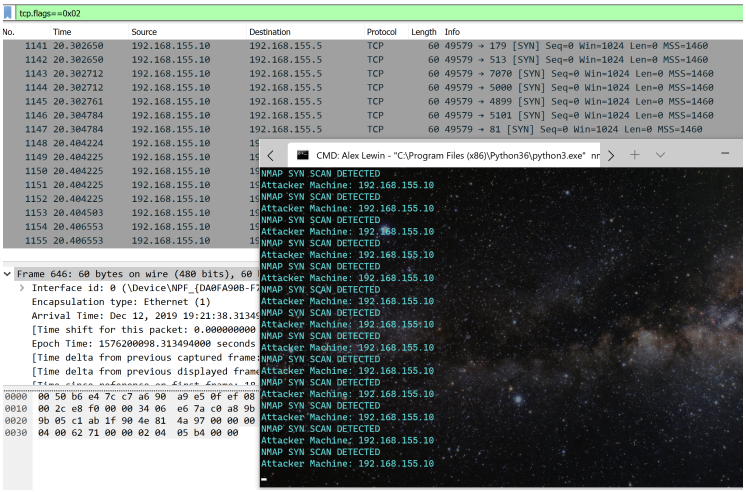
```
#imports pyshark library
import pyshark

#runs live instance of wireshark capture
capture = pyshark.LiveCapture(display_filter='smb')

#continuously loops through packets
for packet in capture:

  #filters wireshark capture for Multiplex ids 65, 64, 81, 82 and nt status
  #alerts and prints attacker ip address
  #the first instance of the correct Multiplex ID due to first request by attacker sent
  if (packet.smb.mid == '65' or packet.smb.mid == '64'
      or packet.smb.mid == '81'
      or packet.smb.mid == '82')
    and (packet.smb.nt_status == '0'):
    print("ETERNAL BLUE DETECTED")
    print("Attacker Machine:", str(packet.ip.src))

capture.sniff()
```

5 Testing and Packet Capture

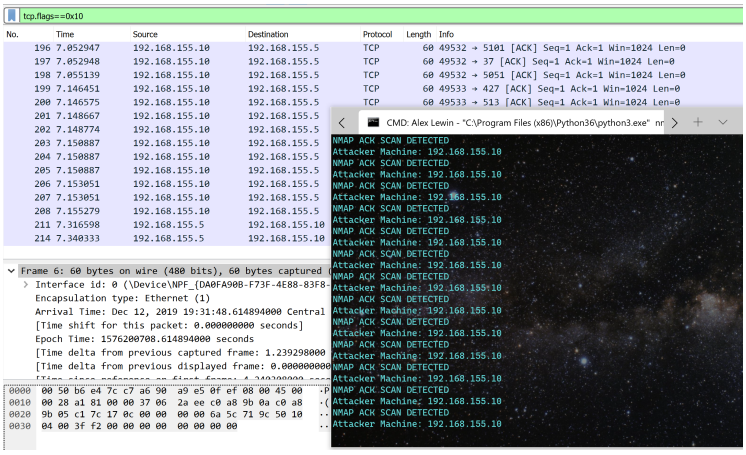
For each attack, we deployed Wireshark and our intrusion detection system in the background.

5.1 NMAP Detection

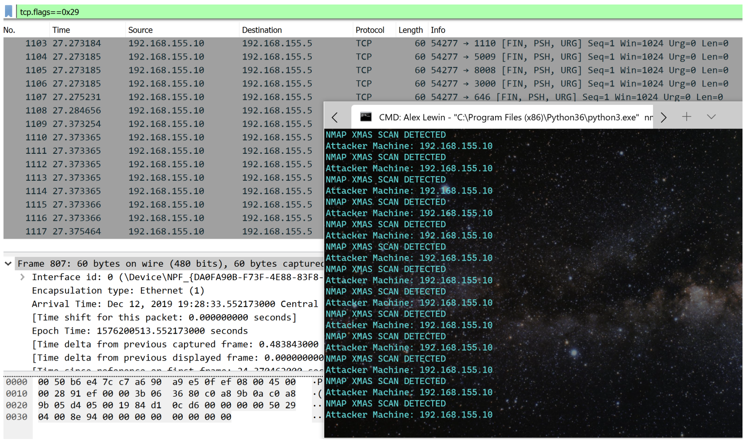
5.1.1 SYN Scan



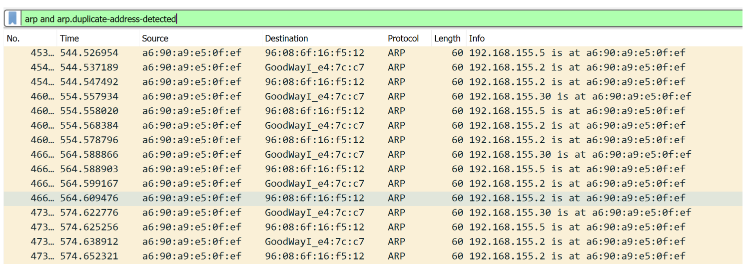
5.1.2 ACK Scan

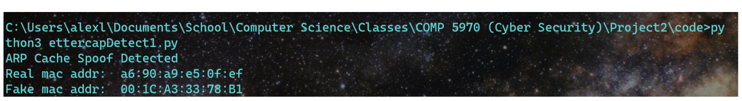


5.1.3 XMAS Scan



5.2 Ettercap Detection



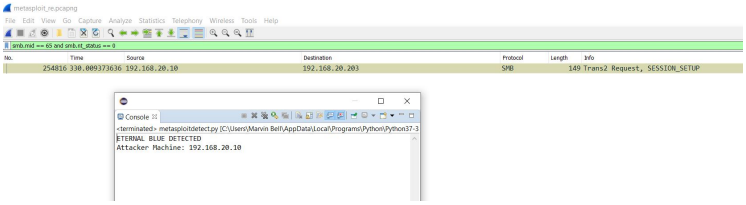


5.3 Responder Detection

```
[+] Poisoning Options:
    Analyze Mode              [OFF]
    Force WPAD auth           [OFF]
    Force Basic Auth          [OFF]
    Force LM downgrade        [OFF]
    Fingerprint hosts         [OFF]

[+] Generic Options:
    Responder NIC             [eth0]
    Responder IP              [192.168.40.10]
    Challenge set             [random]
    Don't Respond To Names    ['ISATAP']



[+] Listening for events...
[*] [LLMNR]  Poisoned answer sent to 192.168.40.5 for name ProxySrv
[*] [LLMNR]  Poisoned answer sent to 192.168.40.5 for name DESKTOP-KNU9HSM
[*] [LLMNR]  Poisoned answer sent to 192.168.40.5 for name DESKTOP-KNU9HSM
---------------------------------------
Exception happened during processing of request from ('192.168.40.5', 55640)
Traceback (most recent call last):
  File "/usr/lib/python2.7/SocketServer.py", line 599, in process_request_thread
    self.finish_request(request, client_address)
  File "/usr/lib/python2.7/SocketServer.py", line 334, in finish_request
    self.RequestHandlerClass(request, client_address, self)
  File "/usr/lib/python2.7/SocketServer.py", line 655, in __init__
    self.handle()
  File "/usr/share/responder/poisoners/LLMNR.py", line 81, in handle
    'AnalyzeMode': '0',
  File "/usr/share/responder/utils.py", line 236, in SavePoisonersToDb
    res = cursor.execute("SELECT COUNT(*) AS count FROM Poisoned WHERE Poisoner=? AND SentToIp=? AND ForName=? AND AnalyzeMode=?", (result['Poisoner'], result[
'SentToIp'], result['ForName'], result['AnalyzeMode']))
OperationalError: database is locked
---------------------------------------
[*] [NBT-NS] Poisoned answer sent to 192.168.40.5 for name WORKGROUP (service: Domain Controller)
[*] [NBT-NS] Poisoned answer sent to 192.168.40.5 for name WORKGROUP (service: Domain Master Browser)
---------------------------------------
Exception happened during processing of request from ('192.168.40.5', 137)
Traceback (most recent call last):
  File "/usr/lib/python2.7/SocketServer.py", line 599, in process_request_thread
    self.finish_request(request, client_address)
  File "/usr/lib/python2.7/SocketServer.py", line 334, in finish_request
```

```
C:\Users\lakum\Documents\GradSchool\Fall2019\ThreatCount_COMP_6970\project2>python responderDetect.py
RESPONDER ATTACK DETECTED!!
```

5.4 Metasploit Detection



6 Conclusions

In completing this assignment, we were successfully able to deploy and detect each of the four attacks using a proprietary intrusion detection system, all while monitoring and analyzing the network traffic. After completing this assignment, we have a much deeper understanding of the nature of intrusion detection systems. In addition, we developed a greater appreciation for the role of a *blue team* in companies, having this project closely reflect their work.

This project showed just how challenging it is to defend against network attacks. While our script detects the vanilla version of each attack, there are certainly a litany of ways to spoof our intrusion detection system. Despite this, we do acknowledge that our program could foreseeably grow into a sophisticated, industrial-grade intrusion detection system. The industrial intrusion detection systems.

7 Recommendations

We would advise any business with an active network to purchase an industrial-grade intrusion detection system or intrusion prevention system. Anti-virus software is the easiest and most efficient way to protect against attackers.

In addition, we advise business to frequently update software to the newest, safe version. It is extremely common for vulnerabilities to be fixed in updates. Having the latest version will protect against most known vulnerabilities.

For example, the vulnerability of Eternal Blue is caused by an outdated version SMB that could be on any given machine. A recommendation to avoid this attack would be to consistently update software to the newest version, such as the MS17-010 security update.

Finally, we advise companies to conduct frequent penetration tests. This will thwart out many of the vulnerabilities in a network.
