Building an IPS solution for inline usage during Red Teaming

Repurposing defensive technologies for offensive Red Team operations

K. Mladenov A. Zismer
{kmladenov,azismer}@os3.nl

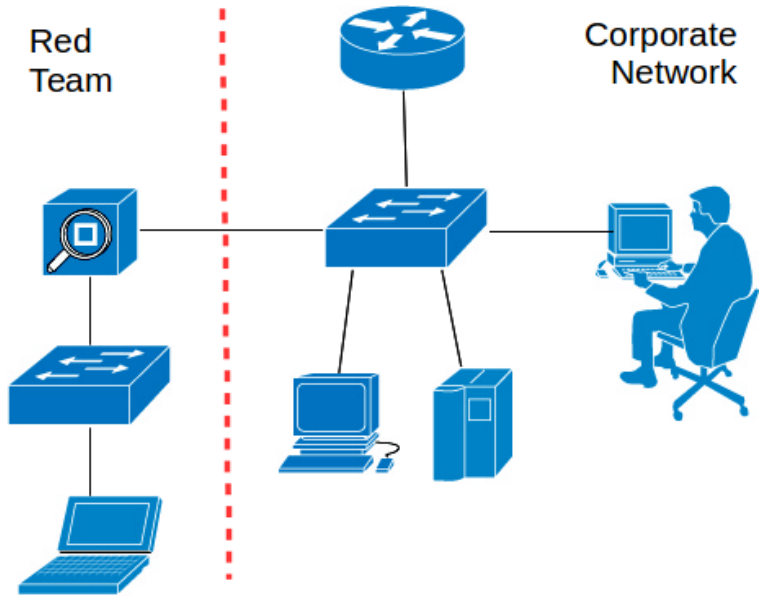
Master Students in
System and Network Engineering
University of Amsterdam

February 2017

Outline

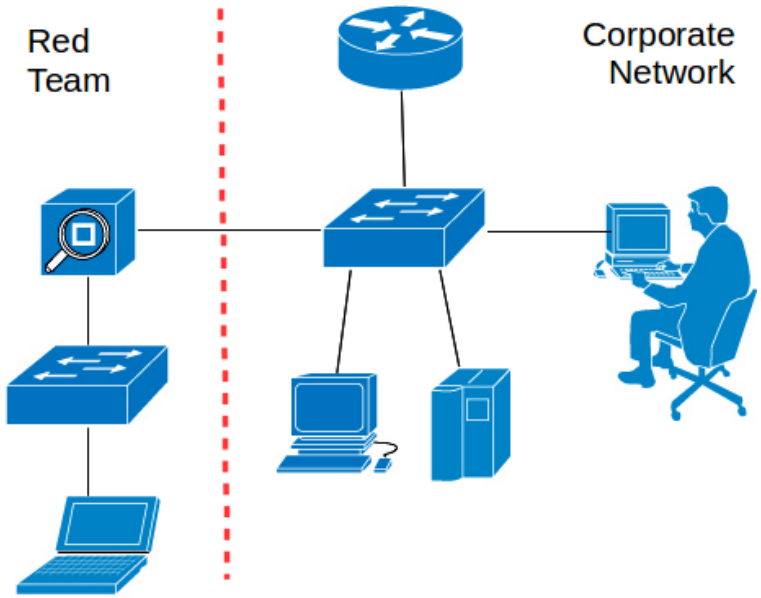
The idea

- Originally from Deloitte.



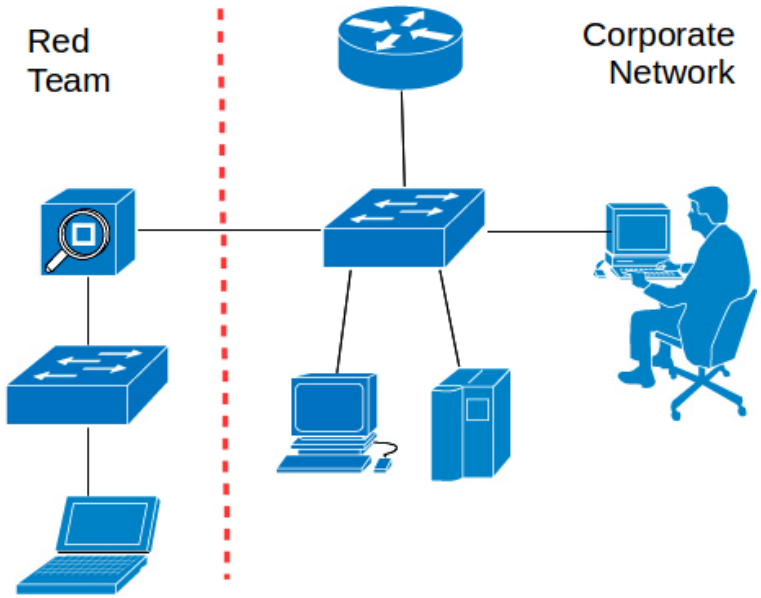
The idea

- Originally from Deloitte.
- For use during penetration tests (Red Teaming)



The idea

- Originally from Deloitte.
- For use during penetration tests (Red Teaming)
- Prevent the attackers from doing detectable mistakes



Research question

In how far is it possible to design a transparent device that disguises an attacker's computer inside a local network?

Research question

In how far is it possible to design a transparent device that disguises an attacker's computer inside a local network?

1. How can outgoing traffic be filtered and sanitised by an IPS?

Research question

In how far is it possible to design a transparent device that disguises an attacker's computer inside a local network?

1. How can outgoing traffic be filtered and sanitised by an IPS?
2. How can incoming traffic be handled to evade investigation and detection?

Types of IDS/IPS engines

Network based

- Deployed either to listen to replica of the traffic or inline.
- Can get visibility over the entire network if properly placed.
- Fail short with encrypted traffic.

Types of IDS/IPS engines

Network based

- Deployed either to listen to replica of the traffic or inline.
- Can get visibility over the entire network if properly placed.
- Fail short with encrypted traffic.

Host based

- Can get full visibility over traffic about to be {en/de}crypted.
- Imposes some difficulty with managing multiple instances on multiple computers.

Types of IDS/IPS engines

Network based

- Deployed either to listen to replica of the traffic or inline.
- Can get visibility over the entire network if properly placed.
- Fail short with encrypted traffic.

Host based

- Can get full visibility over traffic about to be {en/de}crypted.
- Imposes some difficulty with managing multiple instances on multiple computers.

In our case a network-based solution would do the job.
But should it be signature or anomaly based?

How can intruders get detected?

By doing things detectable by an IDS.

How can intruders get detected?

By doing things detectable by an IDS.

But also:

Passively

- Different Operating systems behave in different ways for things not standardised in RFC.
- Some examples include TTL and initial TCP window size.

OS	TTL	TCP window (B)
Windows 7	128	8192
Windows 10	128	8192
Kali Linux	64	29200

How can intruders get detected?

By doing things detectable by an IDS.

But also:

Passively

- Different Operating systems behave in different ways for things not standardised in RFC.
- Some examples include TTL and initial TCP window size.

Actively

- By doing active scans against them.
- More about to follow.

OS	TTL	TCP window (B)
Windows 7	128	8192
Windows 10	128	8192
Kali Linux	64	29200

How can an IPS help?

By using built-in normalizers.

- For IP traffic - handle the TTL.
- For TCP traffic - handle the initial TCP window size.

How can an IPS help?

By using built-in normalizers.

- For IP traffic - handle the TTL.
- For TCP traffic - handle the initial TCP window size.

So how did the selected engines perform?

IPS Engine	TTL handling	TCP window handling
Snort 2.9.9.0	yes	no
Snort 3 alpha	yes	no
Suricata 3.2	no	no

How can an IPS help?

By using built-in normalizers.

- For IP traffic - handle the TTL.
- For TCP traffic - handle the initial TCP window size.

So how did the selected engines perform?

IPS Engine	TTL handling	TCP window handling
Snort 2.9.9.0	yes	no
Snort 3 alpha	yes	no
Suricata 3.2	no	no

But is really Suricata that bad?

Not really. It has LuaJIT support!

But is really Suricata that bad?

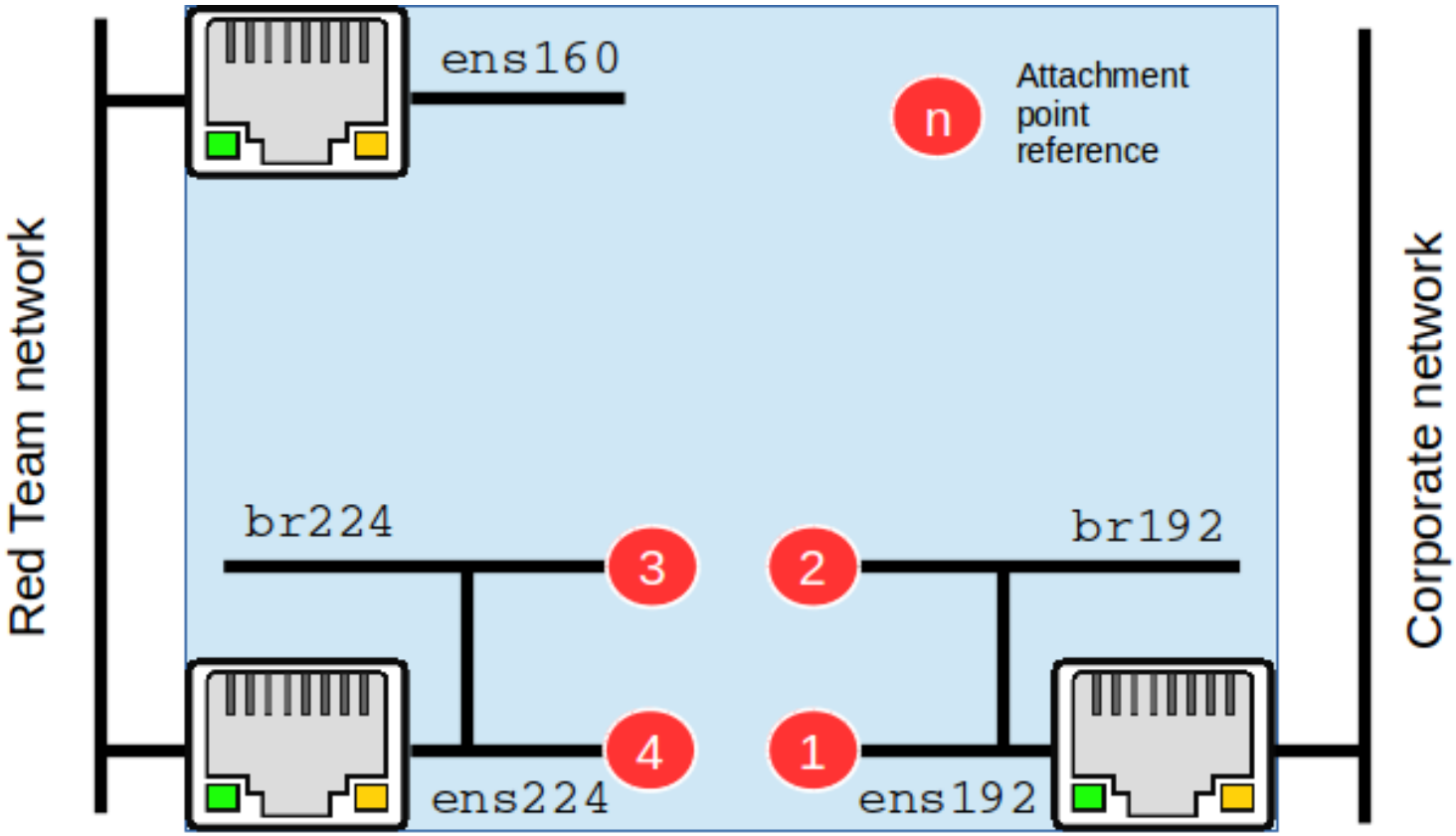
Not really. It has LuaJIT support!

And that means scripting, triggered by a rule!
Including executing commands from the system shell!

```
drop tcp 10.0.0.200 any -> any any (msg:"TCP SYN for
inspection by LUA"; flags:S; sid 1000002; rev:001;
luajit:tcpinspect.lua;)
```

How did the IPS get connected to the network?

There was a need for a solution that did not require scripting... But how did it get attached in this transparent device?



Active detection

- TCP/IP fingerprinting

Active detection

- TCP/IP fingerprinting
- Service and version detection

TCP/IP fingerprinting

- unspecified situations in the RFCs of TCP/IP

TCP/IP fingerprinting

- unspecified situations in the RFCs of TCP/IP

→ different OS specific implementations of the TCP/IP stack

TCP/IP fingerprinting

- unspecified situations in the RFCs of TCP/IP

→ different OS specific implementations of the TCP/IP stack

- `Nmap` sends a variety of probing packets
  - ICMP
  - TCP
  - UDP

TCP/IP fingerprinting

- unspecified situations in the RFCs of TCP/IP

→ different OS specific implementations of the TCP/IP stack

- `Nmap` sends a variety of probing packets
  - ICMP
  - TCP
  - UDP

- results of different tests are combined to create an individual fingerprint

TCP/IP fingerprinting

- unspecified situations in the RFCs of TCP/IP

→ different OS specific implementations of the TCP/IP stack

- `Nmap` sends a variety of probing packets
  - ICMP
  - TCP
  - UDP
- results of different tests are combined to create an individual fingerprint
- known OS/fingerprint mappings are stored in a database

Nmap OS fingerprint format

```
SCAN(V=5.05BETA1%D=8/23%OT=22%CT=1%CU=42341%PV=N%DS=0%DC=L%G=Y%TM=4A91CB90%
     P=i686-pc-linux-gnu)
SEQ(SP=C9%GCD=1%ISR=CF%TI=Z%CI=Z%II=I%TS=A)
OPS(O1=M400CST11NW5%O2=M400CST11NW5%O3=M400CNNT11NW5%
    O4=M400CST11NW5%O5=M400CST11NW5%O6=M400CST11)
WIN(W1=8000%W2=8000%W3=8000%W4=8000%W5=8000%W6=8000)
ECN(R=Y%DF=Y%T=40%W=8018%O=M400CNNSNW5%CC=N%Q=)
T1(R=Y%DF=Y%T=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=Y%T=40%W=8000%S=O%A=S+%F=AS%O=M400CST11NW5%RD=0%Q=)
T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)
IE(R=Y%DFI=N%T=40%CD=S)
```

Nmap OS detection

```
$ sudo nmap -O 10.0.0.220

Starting Nmap 7.01 ( https://nmap.org ) at 2017-02-06 21:47 CET
Nmap scan report for 10.0.0.220
Host is up (0.000063s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 00:0C:29:40:E7:6A (VMware)
Device type: general purpose
Running: Linux 3.X|4.X                          # OS detection correct
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.0
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.20 seconds
```

Listing 1: Inspecting a Ubuntu machine with kernel 4.4.0-59-generic

How can OS detection be evaded?

- `IP Personality`

How can OS detection be evaded?

- `IP Personality`
  - kernel patch that can simulate multiple OS fingerprints
  - needs to be compiled into the kernel
  - only available for 2.4 Linux kernels

How can OS detection be evaded?

- `IP Personality`
  - kernel patch that can simulate multiple OS fingerprints
  - needs to be compiled into the kernel
  - only available for 2.4 Linux kernels
- `honeyD`

How can OS detection be evaded?

- `IP Personality`
  - kernel patch that can simulate multiple OS fingerprints
  - needs to be compiled into the kernel
  - only available for 2.4 Linux kernels
- `honeyD`
  - virtual honeypot framework
  - simulates networks of low-interaction honeypots

How can OS detection be evaded?

- `IP Personality`
  - kernel patch that can simulate multiple OS fingerprints
  - needs to be compiled into the kernel
  - only available for 2.4 Linux kernels
- `honeyD`
  - virtual honeypot framework
  - simulates networks of low-interaction honeypots
  - **Personality engine** to simulate TCP/IP stack

How can OS detection be evaded?

- `IP Personality`
  - kernel patch that can simulate multiple OS fingerprints
  - needs to be compiled into the kernel
  - only available for 2.4 Linux kernels
- `honeyD`
  - virtual honeypot framework
  - simulates networks of low-interaction honeypots
  - **Personality engine** to simulate TCP/IP stack
  - `apt-get install honeyd`

Simulating a Windows XP machine with honeyD

```
1  create winxp
2  set winxp personality "Microsoft Windows XP Professional"
3  set winxp default tcp action reset
4  set winxp default udp action reset
5  set winxp default icmp action closed
6  add winxp udp port 123 open
7  add winxp tcp port 3389 proxy 10.0.0.60:3389
8  add winxp tcp port 22 proxy $ipsrc:22
9  add winxp tcp port 23 "/etc/honeypot/scripts/fake_telnet.sh"
10
11 bind 10.0.0.200 winxp
```

Listing 2: honeyd.conf

Service and version detection

- `Nmap` uses two methods to detect a service

Service and version detection

- `Nmap` uses two methods to detect a service
  1. statically mapping well-known ports to their services

Service and version detection

- `Nmap` uses two methods to detect a service
  1. statically mapping well-known ports to their services
  2. attempting to interact with the services to obtain more details:
     - application name
     - version number

Service and version detection

- `Nmap` uses two methods to detect a service
  1. statically mapping well-known ports to their services
  2. attempting to interact with the services to obtain more details:
     - application name
     - version number
     - **OS family**

Nmap service and version detection

```
$ sudo nmap -sV 10.0.0.205

Starting Nmap 7.01 ( https://nmap.org ) at 2017-02-06 22:39 CET
Nmap scan report for 10.0.0.205
Host is up (0.000047s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.1 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
MAC Address: 00:0C:29:16:3C:76 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/
    submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.10 seconds
```

Listing 3: Inspecting a Ubuntu machine with and ssh service and apache running

How can service detection be evaded?

- Beacons are connecting to CnC server through port 80
- Port 80 is suspicious

How can service detection be evaded?

- Beacons are connecting to CnC server through port 80
- Port 80 is suspicious

→ **Can we detect an `Nmap` scan and temporary close port 80?**

How does Nmap perform a service scan?

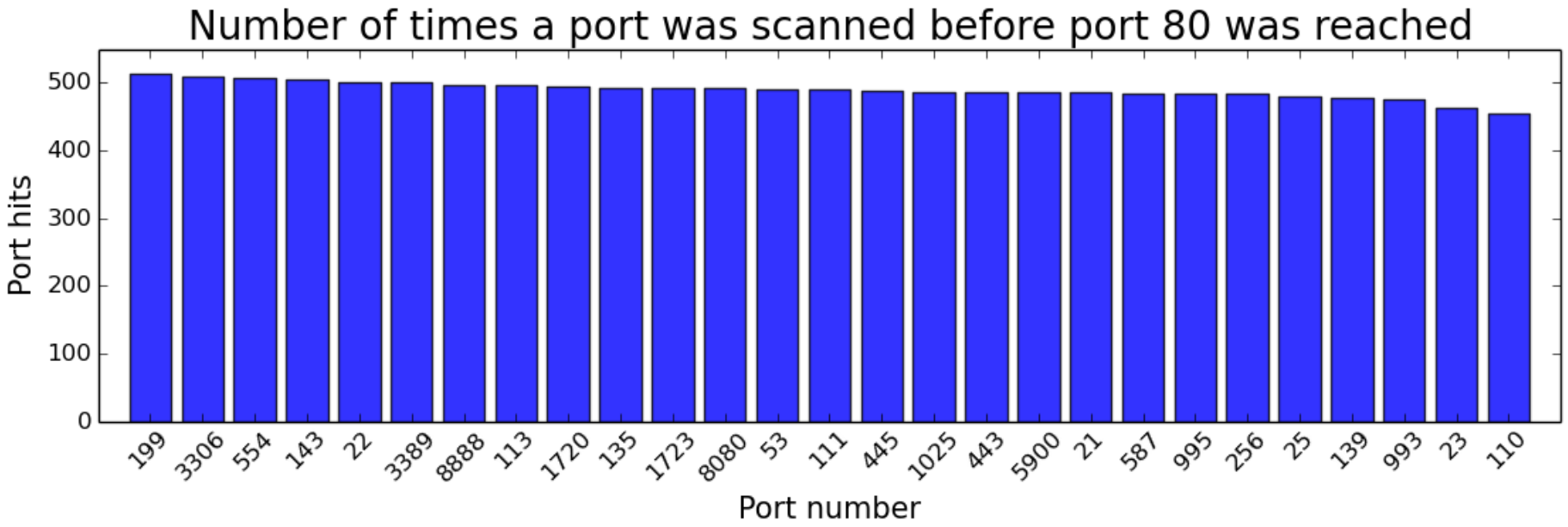


Figure: Result of 1000 Nmap scans

How to react to connection attempts to closed ports?

- Port knocking:
  - listening to secret sequences of port connections

How to react to connection attempts to closed ports?

- Port knocking:
  - listening to secret sequences of port connections
  - opening another port, if a certain sequence is detected

How to react to connection attempts to closed ports?

- Port knocking:
  - listening to secret sequences of port connections
  - opening another port, if a certain sequence is detected
  - `knockD` is a very flexible port knocking daemon

Configuring knockD

```
1 # closing port 22 if 2222, 3333 and 4444 are knocked
2 [opencloseSSH]
3     sequence      = 2222,3333,4444
4     seq_timeout   = 15
5     tcpflags      = syn,ack
6     start_command = iptables -A INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
7     cmd_timeout   = 10
8     stop_command  = iptables -D INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
```

Listing 4: knockd.conf

Configuring knockD

```
1 # close port 80 if either of 199, 3306, 554, ... is knocked
2 [close80]
3     sequence      = 199/3306/554/143/22/3389/8888/...
4     seq_timeout   = 15
5     tcpflags      = syn,ack
6     start_command = iptables -A INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
7     cmd_timeout   = 10
8     stop_command  = iptables -D INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
```

Listing 5: knockd.conf

Configuring knockD

```
1  # separate rule for each port
2  [close80_199]
3      sequence      = 199
4      seq_timeout   = 15
5      tcpflags      = syn,ack
6      start_command = iptables -A INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
7      cmd_timeout   = 10
8      stop_command  = iptables -D INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
9  [close80_3306]
10     sequence      = 3306
11     seq_timeout   = 15
12     tcpflags      = syn,ack
13     start_command = iptables -A INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
14     cmd_timeout   = 10
15     stop_command  = iptables -D INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
16 ...
```

Listing 6: knockd.conf

Configuring knockD

```
1  # rules for all possible permutations
2  [close80_199_3306]
3      sequence      = 199,3306
4      seq_timeout   = 15
5      tcpflags      = syn,ack
6      start_command = iptables -A INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
7      cmd_timeout   = 10
8      stop_command  = iptables -D INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
9  [close80_199_554]
10     sequence      = 199,554
11     seq_timeout   = 15
12     tcpflags      = syn,ack
13     start_command = iptables -A INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
14     cmd_timeout   = 10
15     stop_command  = iptables -D INPUT -s %IP% -p tcp -m multiport --dports 80 -j REJECT
16 ...
```

Listing 7: knockd.conf

Configuring knockD

- running multiple instances of knockD with different rules
- multiple sequences can be detected in parallel
- if Nmap starts scanning with port 80, it cannot be hidden

Reliability

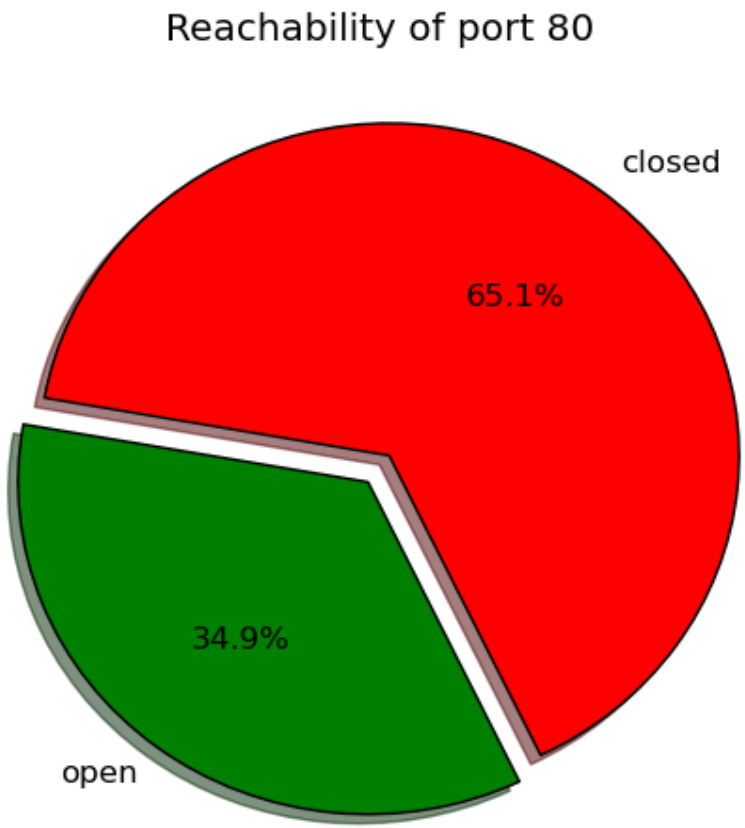


Figure: Result of 1000 `Nmap` scans

Conclusion

- We were able to implement a transparent solution which can protect the attacker from being easily detected.
- At this point it requires additional configuration and has some tradeoffs between different options.

Conclusion

- We were able to implement a transparent solution which can protect the attacker from being easily detected.
- At this point it requires additional configuration and has some tradeoffs between different options.

- Future work
  - Improving reliability of hiding port 80
  - Replace the functionality of external daemons by Lua scripts

Demo

Questions?