Advanced Netfilter: Content Replacement (ala Snort_inline) and Combining Port Knocking with p0f

Michael Rash

DEFCON 12

07/31/2004

http://www.enterasys.com
http://www.cipherdyne.org

Introduction

- Port knocking
- Passive OS fingerprinting with p0f
- fwknop

- Iptables string match extension
- String replacement patch
- Netperf benchmarks

Port Knocking

- Information hiding within sequences of connections to closed (or open) ports
- Access control modification
- Can be encrypted or shared
- Multiple protocols
- Relative timings
- Third party IP access
- Server side is totally passive

http://www.portknocking.org
Martin Krzywinski

Shared Sequence (1)

- Knock client (187.30.30.2):
  - tcp/1001
  - tcp/1002
  - tcp/1003
  - tcp/1004
  - tcp/22
- Knock server (204.10.10.1):
  - opens tcp/22

Shared Sequence (2)

- Knock client (187.30.30.2):
  - tcp/64531
  - udp/63533
  - tcp/5001
  - tcp/5002
  - icmp echo request
  - icmp echo request
  - (wait at least 10 seconds)
  - udp/5003

- Knock server (204.10.10.1):
  - opens tcp/22 and udp/5000 for five minutes

Encrypted Sequence (1)

- Knock client (187.30.30.2):
  - IP: four 8 bit values (174.4.3.2)
  - Port: concatenation of two 8 bit values (5500)
  - Protocol: one 8 bit value (1, 6, 17)
  - Rijndael block size = 16 bytes
  - plain text: 174 4 3 2 21 124 6
  - cipher text: 186 242 110 108 160 231 219 80 142 ...
- Knock server (204.10.10.1):
  - Special range of ports, 8 bits wide (e.g. 60000-60255)

Encrypted Sequence (2)

- Knock client (187.30.30.2):
  - tcp/60186
  - tcp/60242
  - tcp/60110
  - tcp/60108
  - tcp/60160
  - tcp/60231
  - tcp/60219
  - ...
- Knock server (204.10.10.1):
  - Give access through firewall to tcp/5500 for IP 174.4.3.2

p0f

- OS fingerprinting via tcp SYN packets
- SYN+ACK fingerprinting
- RST fingerprinting

- libpcap

http://lcamtuf.coredump.cx/p0f.shtml,
Michal Zalewski

Header Fields

- tcp SYN flag
- tcp window size
- maximum segment size
- selective acknowledgement OK?
- NOP
- tcp window scale
- timestamp

- ttl
- fragment bits
- packet size

p0f Signatures

```
FreeBSD:4.4::FreeBSD 4.4:
      1024:64:1:60:M*,N,W0,N,N,T

Linux:2.4::Linux 2.4/2.6:
      S4:64:1:60:M*,S,T,N,W0
```

- Window size
- Initial TTL
- Don't fragment bit
- Packet size
- TCP options

Iptables Logs

```
iptables -A INPUT -p tcp -i eth0 -j LOG
--log-prefix “DROP “ --log-tcp-options

(iptables -A INPUT -p tcp -i eth0 -j
DROP)
```

Iptables Logs; Decoded IP header fields

- Source and destination IP addresses
- IP datagram length
- Type of service
- TTL
- IP ID
- Fragment bits
- Protocol

```
Jul  8 03:06:12 orthanc kernel: DROP IN=eth0
OUT=
MAC=00:a0:cc:28:42:5a:00:00:00:22:2d:42:00:00
SRC:192.168.10.3 DST:192.168.10.1 LEN=60
TOS=0x10 PREC=0x00 TTL=64 ID=6854 DF
PROTO=TCP
```

Iptables Logs; Decoded TCP header fields

- Source and destination ports
- TCP window size
- TCP flags

```
Jul  8 03:06:12 orthanc kernel: DROP IN=eth0
OUT=
MAC=00:a0:cc:28:42:5a:00:00:00:22:2d:42:00:00
SRC:192.168.10.3 DST:192.168.10.1 LEN=60
TOS=0x10 PREC=0x00 TTL=64 ID=6854 DF
PROTO=TCP SPT=32788 DPT=5500 WINDOW=5840
RES=0x00 SYN URGP=0
```

Iptables Logs; Encoded TCP header fields

- **TCP options!!! p0f depends on this.**

```
Jul  8 03:06:12 orthanc kernel: DROP IN=eth0
OUT=
MAC=00:a0:cc:28:42:5a:00:00:00:22:2d:42:00:00
SRC:192.168.10.3 DST:192.168.10.1 LEN=60
TOS=0x10 PREC=0x00 TTL=64 ID=6854 DF
PROTO=TCP SPT=32788 DPT=5500 WINDOW=5840
RES=0x00 SYN URGP=0 OPT
(020405B40402080A006F1D8E000000001030300)
```

TCP options encoding

- Two formats:
  - type (8 bits)
  - type (8 bits) / length (8 bits) / value (n-16 bits)

e.g. 020405b4 = MSS / 4 bytes / 1460

01 = NOP
00 = End of options

Decoded TCP options

- OPT (020405B40402080A00749E8600000000001030300)
  - MSS = 1460
  - Selective Acknowledgement permitted
  - Timestamp
  - NOP
  - Window Scale = 0

Packet Summary

- Length = 60
- Don't fragment bit
- TTL = 64
- Window size = 5840
- MSS = 1460
- Selective Acknowledgement permitted
- Timestamp
- NOP
- Window Scale = 0

What does p0f have to say?

```
S4:64:1:60:M*,S,T,N,W0

Linux:2.4::Linux 2.4/2.6
```

Other fingerprinting strategies

- IP ID
- Type of Service

"Passive OS Fingerprinting: Details and Techniques", Toby Miller

Xprobe
http://sys-security.com, Ofir Arkin

fwknop

- Iptables log messages
- Shared or encrypted knock sequences
- Multi-protocol (tcp, udp, icmp)
- IP/network restriction
- Relative and absolute port timings
- Local username identification
- Selectable iptables ruleset entry
- Access timeouts
- Exact or regex OS match

Implementation

- “client/server”
- Knock sequences encrypted via Rijndael
- syslog monitor “knopmd”
- named pipe
- sysklogd and syslog-ng
- Process monitoring “knopwatchd”
- OpenBSD /etc/pf.os

http://www.cipherdyne.org/fwknop/

Live Demo...

PART II

Iptables string match extension

- Application layer inspection
- Boyer-Moore algorithm
- BM_MAX_HLEN = 1024

  linux/include/linux/netfilter_ipv4/ipt_string.h

String match interface

```
Snort SID 940: “WEB-FRONTPAGE
shtml.dll”


iptables -I FORWARD 1 -p tcp --dport 80
--tcp-flags ACK ACK -m string  --string
“/_vti_bin/shtml.dll” -j LOG --log-
prefix “SID940 ”
```

String match interface (2)

```
Snort SID 261: “DNS EXPLOIT named
overflow attempt”

iptables -I FORWARD 1 -p tcp --dport 53
--tcp-flags ACK ACK -m string --hex-
string “|CD80 E8D7 FFFF FF|/bin/sh” -j
LOG --log-prefix “SID261 ”
```

Iptables targets

```
-j ACCEPT

-j DROP

-j RETURN

-j REJECT
     --reject-with
        icmp-net-unreachable
        icmp-host-unreachable
        icmp-port-unreachable
        ...
        tcp-reset
```

fwsnort

- Translates Snort rules into “equivalent” iptables rules
- String match extension
- 70% translation rate

http://www.cipherdyne.org/fwsnort/

Evasion

- Packet fragmentation
- Polymorphic shellcode
- URL encoding
- Session splicing (Whisker)

Snort_inline

- Inline IDS/IPS
- Linux bridge
- Netfilter libipq
- libnet

Snort_inline packet decisions

- Alert
- Drop
- Reject
- Replace
- Pass

```
content: “/bin/sh”; replace:
“/ben/sh”;
```

Snort_inline packet journey

- (kernel space) packet in ingress interface
- (kernel space) iptables FORWARD chain
- (kernel space) QUEUE target -> libipq
- (user space) context switch to Snort_inline
- (user space) Snort detection engine
- (user space) libipq and packet verdict
- (kernel space) packet on egress interface

Iptables string match patch

```
linux/net/ipv4/netfilter/ipt_string.c

char * search(char *needle, char
*haystack, int nlen, int hlen)
```

We get a pointer to the data!!!

What about checksums?

Replace string interface

```
Snort SID 940: “WEB-FRONTPAGE
shtml.dll”


iptables -I FORWARD 1 -p tcp --dport 80
--tcp-flags ACK ACK -m string  --string
“/_vti_bin/shtml.dll” --replace-string
“/vti_bin/shtml.dab” -j LOG --log-
prefix “nullify SID940 ”
```

Replace string interface (2)

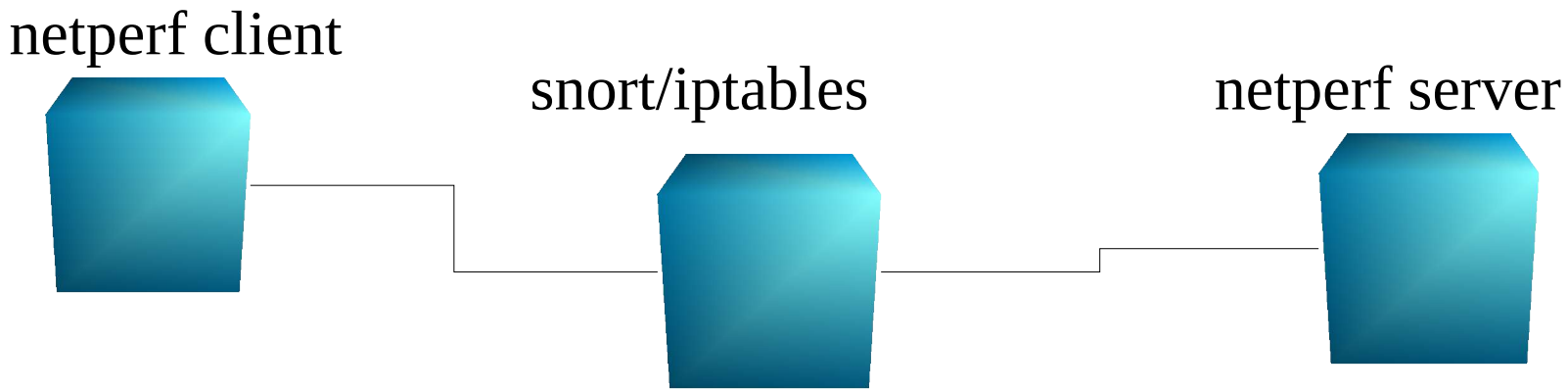
```
Snort SID 261: “DNS EXPLOIT named
overflow attempt”

iptables -I FORWARD 1 -p tcp --dport 53
--tcp-flags ACK ACK -m string --hex-
string “|CD80 E8D7 FFFF FF|/bin/sh” --
replace-hex-string “|4141 4141 4141
41|/ben/sh” -j LOG --log-prefix
“nullify SID261 ”
```

Iptables packet journey

- (kernel space) packet on ingress interface
- (kernel space) packet match in FORWARD chain
- (kernel space) string match function
- (kernel space) data replacement
- (kernel space) packet on egress interface

Netperf bencharks

- Netperf
- Data port patch



Throughput; 100MB Network

- Iptables forwarding: 1.95MB/s
- Snort_inline forwarding: 1.72MB/s
    - 12% faster

- Iptables replace: 1.92MB/s
- Snort_inline replace: 1.72MB/s
    - 10% faster

- Iptables replace and log: 1.53MB/s
- Snort_inline replace and log: 1.02MB/s
    - 43% faster

Applications?

- Well-defined exploits
- Preserve application layer responses

References

fwknop: http://www.cipherdyne.org/fwknop

fwsnort: http://www.cipherdyne.org/fwsnort

snort2iptables: http://www.stearns.org/snort2iptables

Snort_inline: http://snort-inline.sourceforge.net

“The Base Rate Fallacy and its Implications for the Difficulty of Intrusion Detection”: http://www.raid-symposium.org/raid99/PAPERS/Axelsson.pdf