# Port Knocking and Single Packet Authorization: Practical Deployments

Michael Rash
Security Architect
Enterasys Networks, Inc.

http://www.cipherdyne.org/

The Last HOPE Conference
NYC, 2008.07.19

# Agenda

- Why another talk about PK/SPA?
  - Little consensus in the security community – what are the trends?  Is PK/SPA used in practice?
- Practical security tradeoffs  between PK vs. SPA
  - Built-in protocol deficiencies vs. implementation complexity (previous talks have concentrated on protocol deficiencies of PK)
- Snort rules to detect pre-1.9.6 fwknop SPA messages
- Release of fwknop-1.9.6 + advanced topics
- Real world deployment examples for SSH and HTTP
- Live demo + Questions

# The Basics...

- PK encodes authentication information within packet headers – usually as a series of connections to closed (or just logged) ports
- SPA encodes authentication information within packet application layer data
- Authentication information is collected passively (by log monitoring or sniffing the wire directly)
- Both techniques assume that a service is protected behind a default drop packet filter
- The packet filter is reconfigured to allow temporary access, and sessions are typically kept alive via a connection tracking mechanism
- Scanning for a service with Nmap no longer works

# Why Not Just Look for Brute Force Password Guessing Attempts?

- DenyHosts, fail2ban, custom log parsers, etc...
- “Relay Server Tactic Dupes Auto-Reporting”
  - http://www.theregister.co.uk/2008/07/14/brute_force_ssh_attack/
- Exploits commonly have nothing to do with guessing a weak password (Debian OpenSSL vuln, overflow vulnerabilities from time to time)

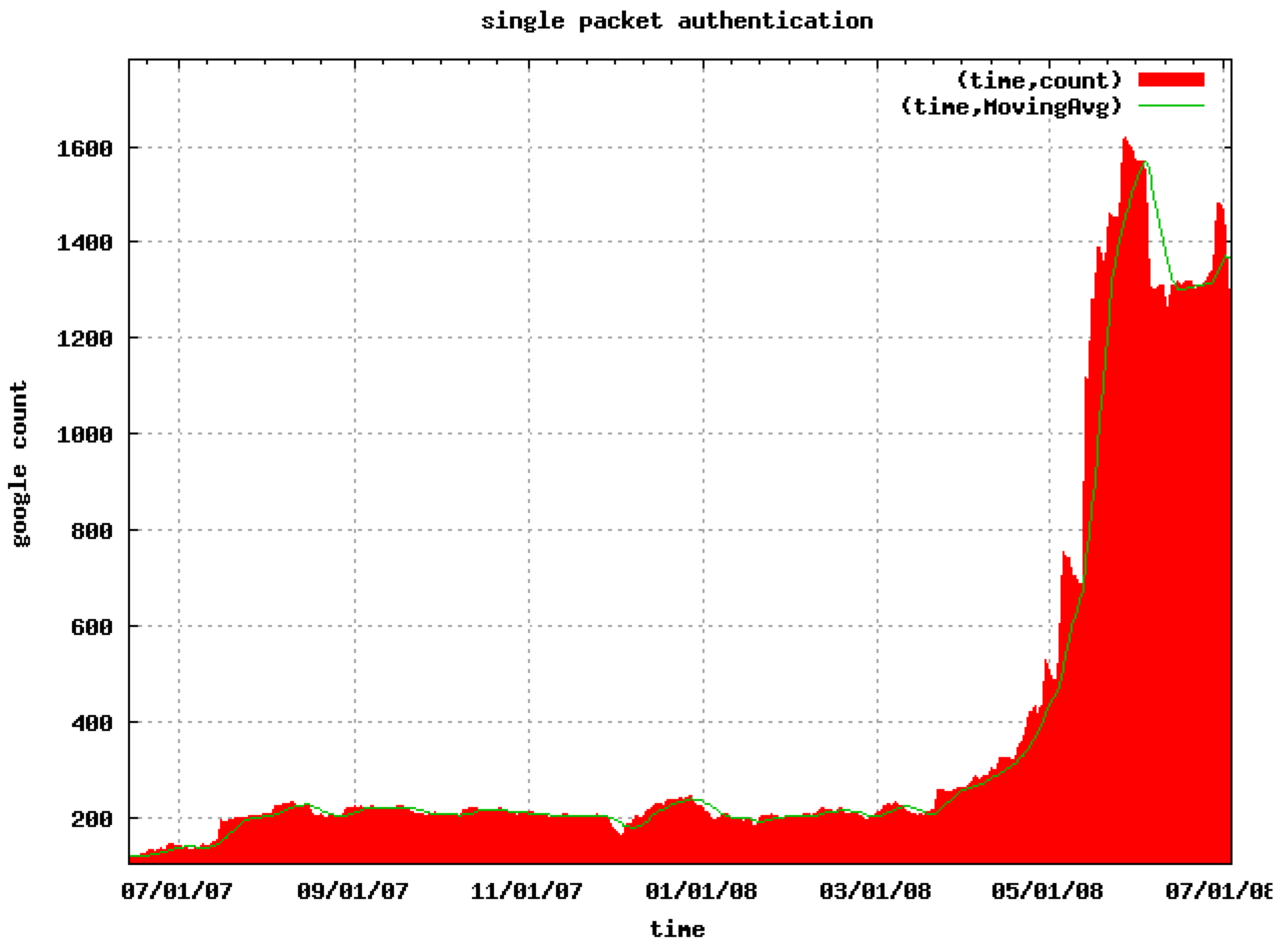
# Is PK/SPA Useful?

- It's difficult to exploit vulnerabilities in services protected by default-drop packet filters.
- PK/SPA is not Security Through Obscurity – it's *concealment* in the same spirit as passwords and encryption keys
- Many competing implementations (~30).
- It is interesting to note that many people still concentrate on PK/SPA detection (more on this later). Why not also propose mechanisms to defeat PK/SPA? (It's easier to defeat PK, whereas SPA *assumes* an attacker can monitor all packets.)

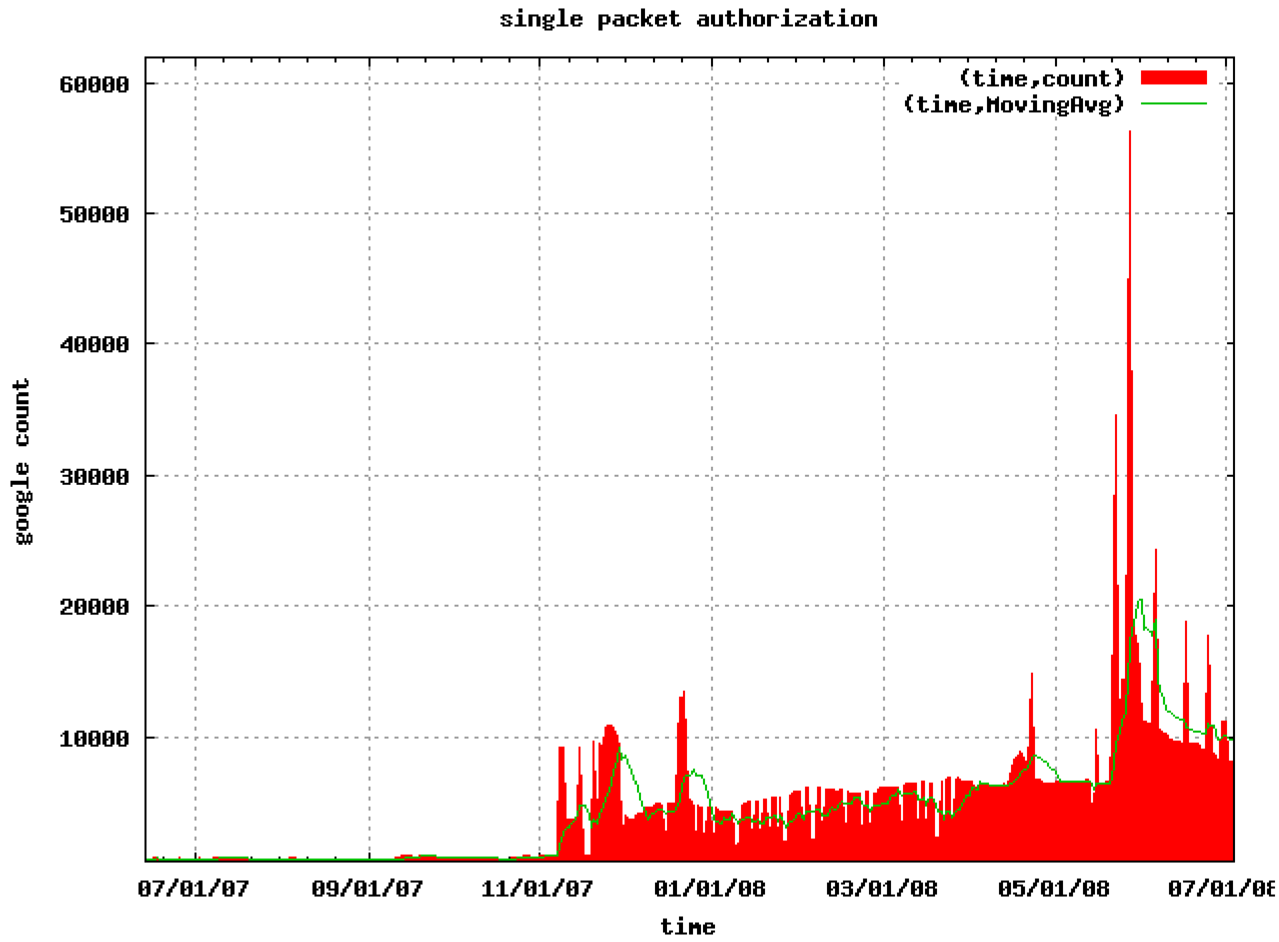
# Is PK/SPA Useful? (con'td)

- fwknop downloads (all versions):
  - 2006: 2768
  - 2007: 6976
  - 2008: 9602 (so far this year)
- Gootrude graphs of trends in search engine results
- Search term results collected once per day from Google since July 2007
- Released under the GPL: http://www.cipherdyne.org/gootrude/

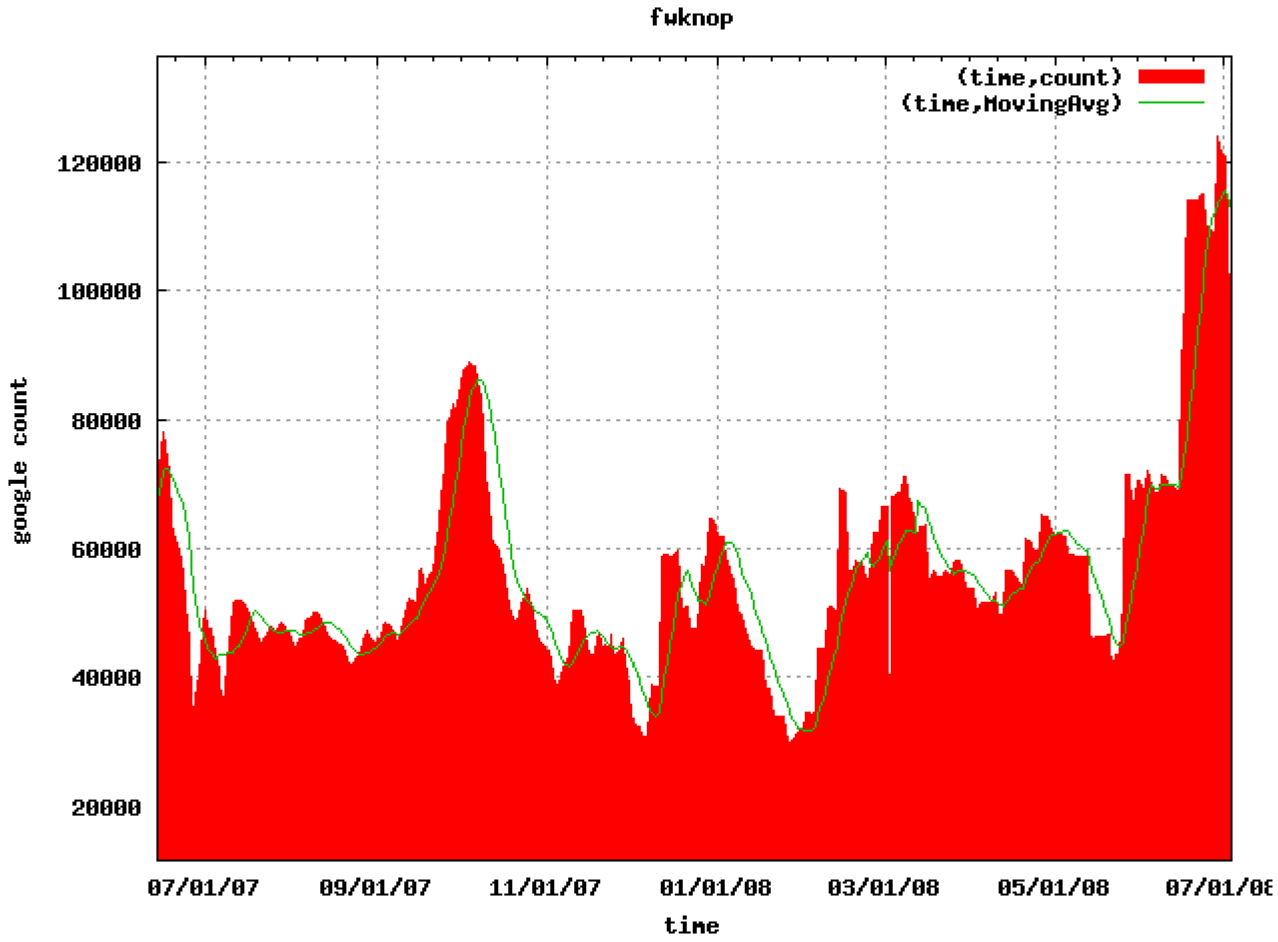
# Gootrude Graphs: “Single Packet Authentication”



# Gootrude Graphs: “Single Packet Authorization”



# Gootrude Graphs: “fwknop”



# Trends?

- SPA usage is up, but widespread deployment has a long way to go.
- A modifier will be efforts to package PK/SPA software for various platforms, and efforts to support different firewalls and/or router ACL's.
- *Open question*: To what extent are PK/SPA techniques used by the blackhat community or in botnets? *...This would make a great topic for a research paper*.

# PK vs. SPA

# Single-Port Shared PK Sequence

"If a (SYN) packet is received on TCP/12345, then grant access to TCP/22 from the source IP"

- Advantages:
  - Simplest possible sequence so complexity of the knock daemon is minimized – only 16 bits of information processed by the daemon for the incoming port
  - libpcap not required – can acquire data from firewall logs
  - PK sequence trivially generated by any client, even a stock web browser
  - Cannot break the sequence with duplicate packets

# Single-Port Shared PK Sequence (cont'd)

- Disadvantages:
  - Why not just Nmap the target for access? (Nmap targets twice and diff the results would expose such PK daemons.)
  - Replay attacks are trivial, and not even necessary to gain access
  - The sequence is basically only effective at stopping automated bots and worms that test basic service availability without scanning other ports

# Multi-Port/Protocol Shared PK Sequence + p0f

- “If the following packets are received, and one of the TCP SYN packets is fingerprinted as from the Linux-2.6 networking stack, then grant access to TCP/22 from the source IP”
  - TCP/12345 (SYN)
  - UDP/100
  - ICMP
  - TCP/54321
  - TCP/1000 (orphaned ACK)

# Multi-Port/Protocol Shared PK Sequence + p0f (cont'd)

- Advantages:
  - Breaking the PK authentication requires eavesdropping
  - Built-in TCP stack characteristics are used as an additional authentication parameter
  - Low complexity of the knock daemon – basic firewall log parsing is sufficient (if the logs contain TCP options – such as iptables logs with –log-tcp-options)

  Jul 19 13:31:26 isengard kernel: [  876.738584] IN=vmnet8 OUT= MAC=00:50:56:c0:00:08:00:0c:29:3e:64:d5:08:00 SRC=192.168.79.128 DST=192.168.79.1 LEN=64 TOS=0x10 PREC=0x00 TTL=64 ID=320 DF PROTO=TCP SPT=56458 ***DPT=12345*** WINDOW=65535 RES=0x00 SYN URGP=0 ***OPT (020405B40103030101080A0013D1B2000000004020000)***

# Multi-Port/Protocol Shared Knock Sequence + p0f (cont'd)

- Disadvantages:
  - Looks like a port scan to any IDS that is watching
  - Sequence replay is trivial whenever eavesdropping is possible
  - PK authentication trivially DoS'd by spoofing a duplicate packet to any port in the sequence
  - Encryption is not used, so cannot vary the access request (strictly a shared sequence)

# Encrypted Port Knocking Sequence + p0f

- Advantages:
  - Can encode the desired access within the encrypted data
  - Breaking the PK authentication requires eavesdropping
  - Built-in TCP stack characteristics are used as an additional authentication parameter
  - Low complexity of the knock daemon – basic firewall log parsing is sufficient

# Encrypted Port Knocking Sequence + p0f (cont'd)

- Disadvantages:
  - Rijndael block size requires a significant number of packets (in the context of a PK sequence)
  - ***Really*** looks like a port scan to any IDS that is watching
  - Sequence replay is trivial whenever eavesdropping is possible
  - PK authentication trivially DoS'd by spoofing a duplicate packet to any port in the sequence (becomes easier as the length of the sequence grows)

# Single Packet Authorization: Rijndael Cipher

- Advantages:
  - It's fast – only a single packet is transmitted
  - Minimal network footprint – unlikely to be flagged by an IDS
  - Can encode the desired access or full commands within the encrypted data
  - Breaking the SPA authentication requires eavesdropping
  - No two SPA packets are the same, so replay attacks are not feasible
  - Easily extensible to new SPA message types (more data to play with whereas transmitting information via PK sequences is comparatively cumbersome)

# Single Packet Authorization: Rijndael Cipher (cont'd)

- Disadvantages:
  - Code complexity is higher (packet sniffing, Rijndael cipher implementation, SHA-256 digest implementation, etc.)
  - Requires a specialized client

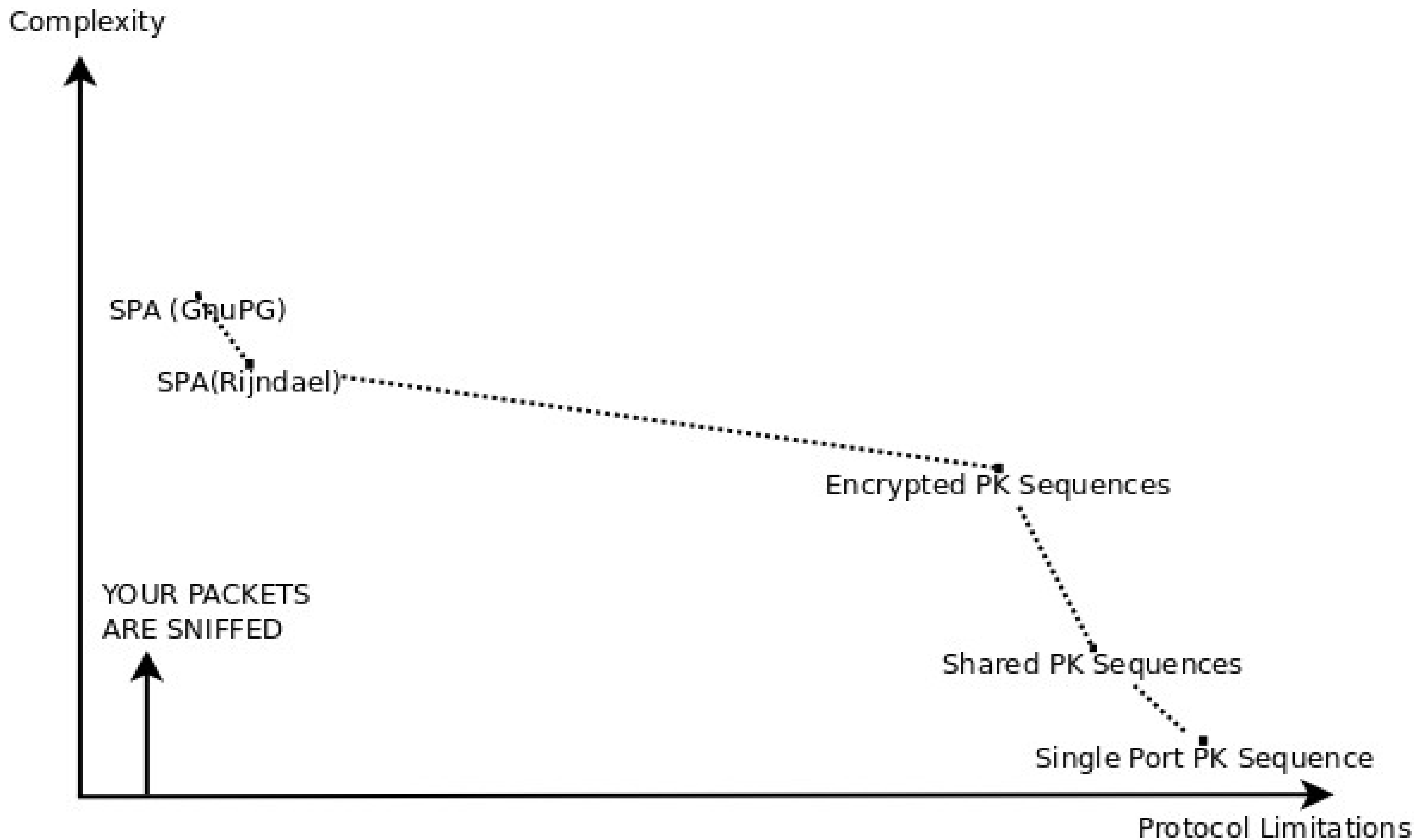
# SPA: GnuPG Encryption

- Advantages (in addition to benefits provided by Rijndael SPA messages):
  - Extremely strong crypto – 2048-bit keys can be used, so cryptanalysis of SPA messages is the most difficult
  - “Important” GnuPG keys (such as for email encryption) are only used for SPA message signing, so new keys only need to be generated for the SPA server side
  - Client-side verification of user password before SPA message is sent (minor)

# SPA: GnuPG Encryption (cont'd)

- Disadvantages:
  - Code complexity is highest
  - Requires a specialized client in addition to a functioning GnuPG installation

# PK vs. SPA Summary



# Detecting SPA Traffic

# Snort Rules for SPA Detection (pre-fwknop-1.9.6)

- Look for base64-encoded data over UDP 62201:

  alert udp any any -> any **62201** (msg:"fwknop SPA traffic"; dsize:>150; **pcre:"/==$/"**; sid:20080001; rev:1;)

  (Unfortunately byte_jump does not apply to the UDP header)

- Look for artifact of Crypt::CBC encryption (“Salted__” prefix):

  alert udp any any -> any **62201** (msg:"fwknop Rijndael SPA traffic"; **content:"U2FsdGVkX1"**; depth:10; dsize:>150; sid:20080002; rev:1;)

# Raw Rijndael SPA Packet

0x0000: ***5361 6c74 6564 5f5f*** 4fa3 9016 66ef f12b ***Salted__***O...f..+

0x0010: 8025 77b8 f454 7feb 5258 3236 5d1d 3616 .%w..T..RX26].6.

0x0020: 48a7 e56d dcb6 5f47 f089 4416 5dbe 9d45 H..m.._G..D.]..E

0x0030: 0878 ed88 cfec c945 06ee 36bd d076 834e .x.....E..6..v.N

0x0040: dbaf ecc2 2960 143f fd6c 09a7 4c1f 138b ....)`.?.l..L...

0x0050: 8789 bc72 faf9 78c4 5506 e226 940a 96d2 ...r..x.U..&....

0x0060: 7a61 e3ff df12 dee0 dd72 63e8 018e cc4c za.......rc....L

0x0070: 8db6 4599 0f98 7460 a03b 3f34 3615 3e12 ..E...t`.;?46.>.

0x0080: 8dab 016f e19c 76f4 aa36 d728 61ad ade6 ...o..v..6.(a...

# Base-64 Encoded Rijndael SPA Packet

- 144 bytes long, so no trailing “=” chars:

***U2FsdGVkX1***9Po5AWZu/xK4Ald7j0VH/rUlgyNl0dNhZIp+Vt3LZfR/CJ
RBZdvp1FCHjtiM/syUUG7ja90HaDTtuv7MIpYBQ//WwJp0wfE4uHibx
y+vl4xFUG4iaUCpbSemHj/98S3uDdcmPoAY7MTI22RZkPmHRgoDs/
NDYVPhKNqwFv4Zx29Ko21yhhra3m

# Snort Rules for SPA Detection (pre-fwknop-1.9.6) (cont'd)

- Look for base64-encoded version as 0x8502 in the first two bytes for GnuPG-encrypted data:

$ perl -MMIME::Base64 -e 'print encode_base64("\x85\x02\n")'

hQIK

Snort rule:

alert udp any any -> any **62201** (msg:"fwknop GnuPG encrypted SPA traffic"; **content:"hQ"**; depth:2; **dsize:>1000**; sid:20080003; rev:1;)

# GnuPG SPA Packet

- 1044 bytes long including one trailing “=” char

***hQ***IOA3yoH1L5ONECEAgAktg0GJNRrBno4DFaSUSiZhrZ9BIqt
aehcfV4F7oimk1WFLo5F0Jn2YoA/zbYbNKQc3vo3hCFnvf5P4a
F1slwp0pw0zl4JXyPB0bILm1OaWytNWLBNtL/iIWX0qKVIRGbl
mDsItffpu3xCqjTvTQ7GF4stqHp..........gKSJ6SVDVM0JM5nGw
m3gCgnZYPZvoMxJlIs3YeywGEoPVC/lkAGByWTnuWvG9QwN
Zt1eZlIgx3733WTuh0/XxpY***=***

# SPA Detection Countermeasures

- Removal of base-64 closing “=” characters – the client strips them out and the server pads incoming data to a multiple of four before decoding
- Removal of “Salted__” prefix
- Removal of “hQ” prefix
- Destination port randomization for the SPA packet and local NAT rules to meet services on randomized ports as well (i.e. use 'ssh -p <port> user@host')
- Randomization of SPA packet source port

(All of these are implemented in fwknop as of the 1.9.6 release)

# SPA and NAT + Port Randomization

[client]$ fwknop -A tcp/22 --NAT-access 192.168.10.22 --NAT-rand-port --rand-port -R -D 22.2.2.2

[+] Sending 216 byte message to 22.2.2.2 over udp/49672...

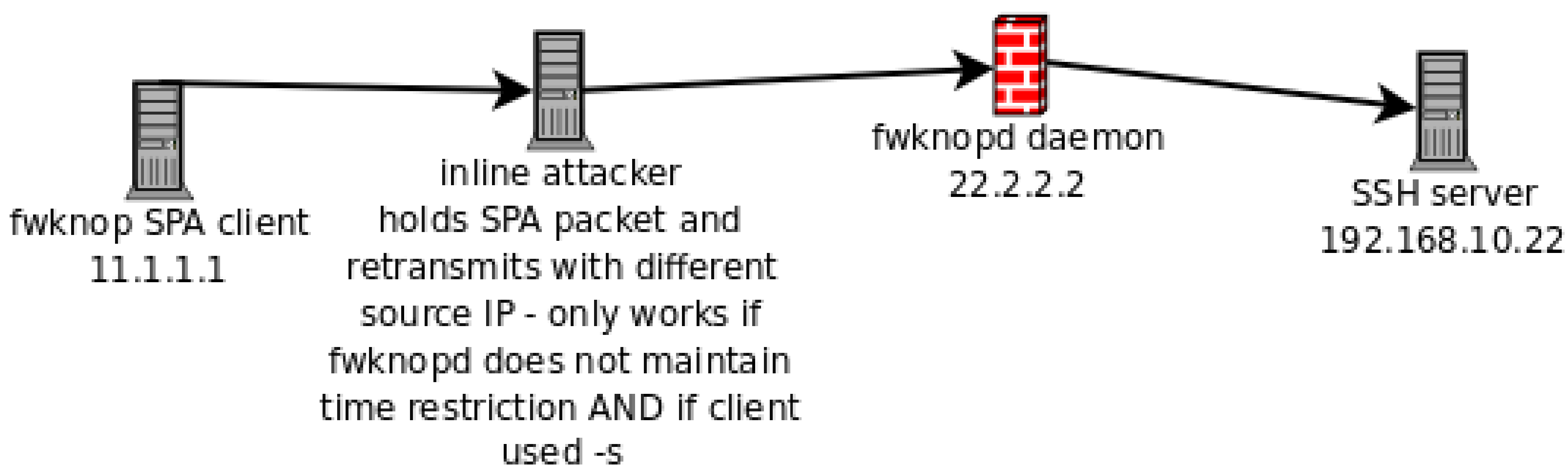
Requesting NAT access to tcp/22 on 192.168.10.22 via port 33914

[client]$ ssh -p 33914 mbr@22.2.2.2

<now have an SSH connection to the internal 192.168.10.22 system over randomly assigned port 33914>

# Old SPA Man-In-The-Middle Attack

- We've concentrated on SPA detection, so it's only fair to present an attack as well
- fwknop has ***not*** been vulnerable since 2006



# fwknop-1.9.6 release

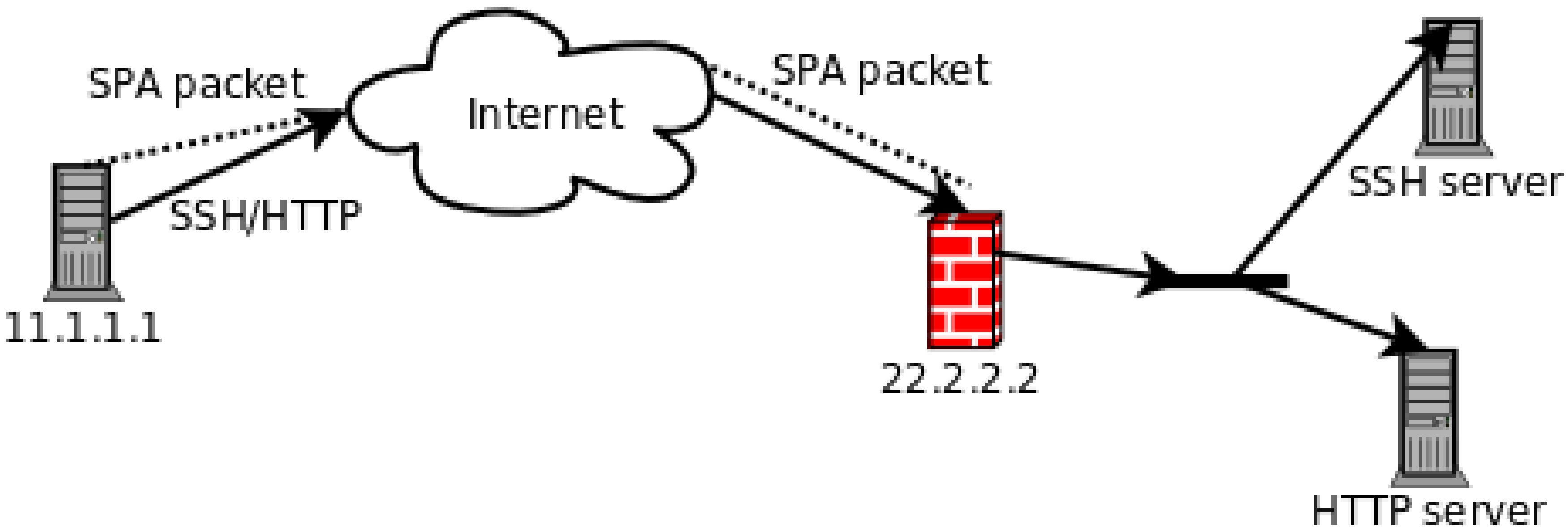
- Anti-detection measures by removing invariant sections of base64-encoding and artifacts encryption algorithms on SPA message data
- Randomized source ports even on client OS stacks that don't support this directly (this includes pre-2.6.24 Linux)
- Test suite support for port knocking mode
- For PK mode, fwknopd no longer requires syslog to communicate with a named pipe

# Upcoming Developments

- Re-write fwknop in C so that it's portable to embedded Linux distributions such as OpenWRT on Linksys routers
- Additional UI development in Java
- Web server PK/SPA proxy
- SPA proxy support within fwknopd directly so that chains of NAT rules are built up to allow access to deeply buried services on internal networks
- Port fwknop PK mode to ipfw and pf firewalls

# SPA Access Examples

# SPA Network Diagram



# SPA access.conf File

```
# cat /etc/fwknop/access.conf
SOURCE: ANY;
ENABLE_FORWARD_ACCESS: Y;
PERMIT_CLIENT_TIMEOUT: Y;
REQUIRE_USERNAME: root;
REQUIRE_SOURCE_ADDRESS: Y;
OPEN_PORTS: tcp/22, http/80;
GPG_HOME_DIR: /root/.gnupg;
GPG_DECRYPT_ID: 361BBAD4;
GPG_DECRYPT_PW: fwknoptest;
GPG_REMOTE_ID: 6A3FAD56;
FW_ACCESS_TIMEOUT: 60;
```

# SPA-hardened SSH

***20:08:24***.279221 IP 11.1.1.1.24593 > 22.2.2.2.***40301***: UDP, length 204

20:08:37.147128 IP 11.1.1.1.51165 > 22.2.2.2.***21059***: S 1444531903:1444531903(0) win 65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 1119837 0,sackOK,eol>

20:08:37.147209 IP 22.2.2.2.***21059*** > 11.1.1.1.51165: S 1950465138:1950465138(0) ack 1444531904 win 5792 <mss 1460,sackOK,timestamp 326110 1119837,nop,wscale 7>

20:08:37.148488 IP 11.1.1.1.51165 > 22.2.2.2.***21059***: . ack 1 win 33304 <nop,nop,timestamp 1119838 326110>

20:08:37.183911 IP 22.2.2.2.***21059*** > 11.1.1.1.51165: P 1:41(40) ack 1

*<Firewall no longer allows new connections, but keeps the existing connection open>*

21:11:18.781283 IP 11.1.1.1.51165 > 22.2.2.2.***21059***: P 2880:2912(32) ack 4657

21:11:18.785610 IP 11.1.1.1.51165 > 22.2.2.2.***21059***: F 2912:2912(0) ack 4657

21:11:18.786351 IP 22.2.2.2.***21059*** > 11.1.1.1.51165: F 4657:4657(0) ack 2913

***21:11:18***.793925 IP 11.1.1.1.51165 > 22.2.2.2.***21059***: . ack 4658

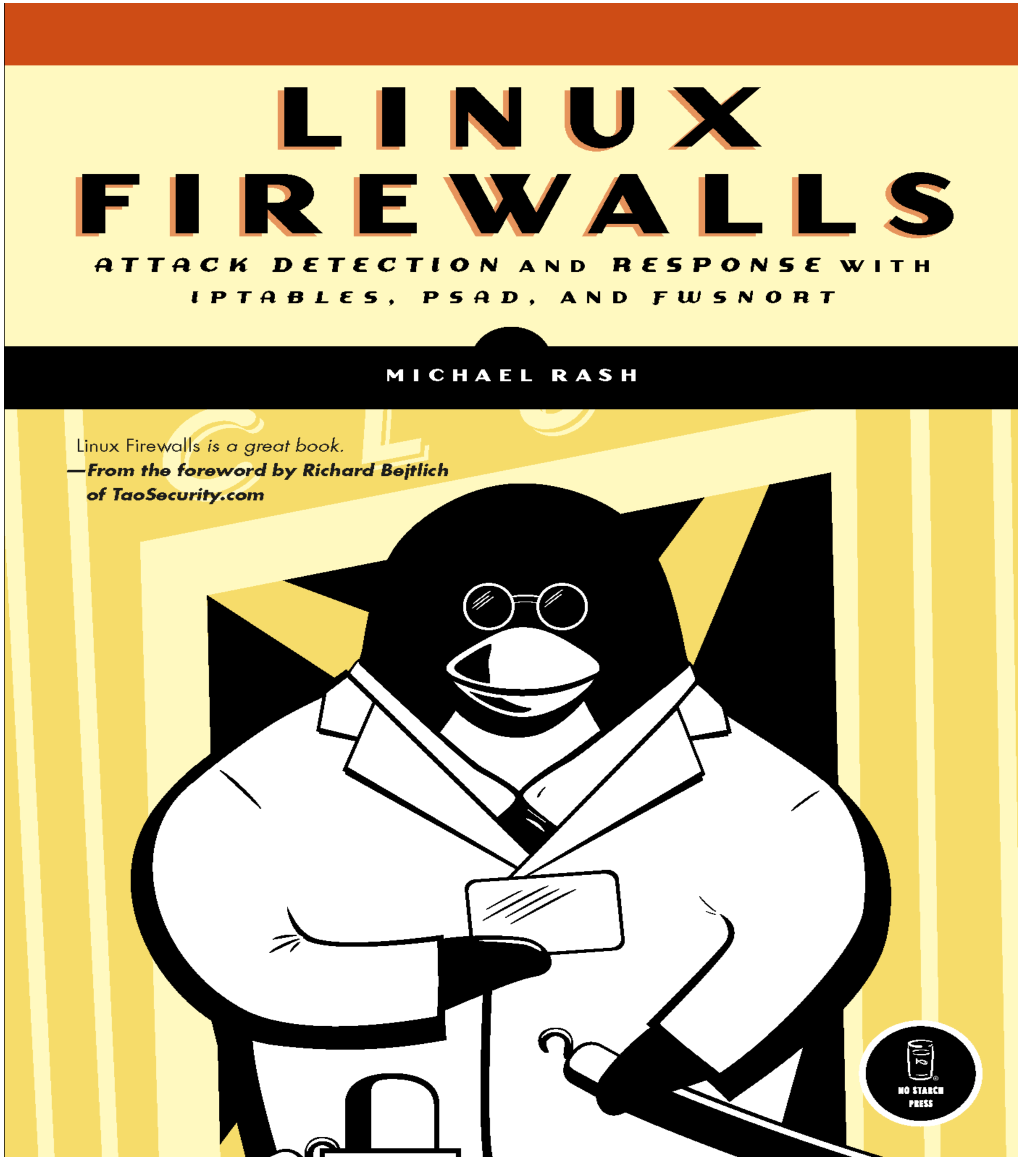
# SPA-hardened HTTP?

- HTTP is chatty at the transport layer
- Use client-side timeouts to extend firewall accept rules
- Can make sense for some deployments as web applications become more important

# Live Demo...

# No Starch Press, Oct 2007



# Questions?

http://www.cipherdyne.org/

mbr@cipherdyne.org