Single Packet Authorization:
The Ghost in the Machine

Join the Discussion
Connect

By Russ McRee – ISSA member, Puget Sound (Seattle), USA Chapter

Prerequisites

Linux
build-essential, libpcap-dev, mailx for installation from source
iptables – consider firewall.sh to test SPA

Similar Projects

No such thing ;-)

Toolsmith covered Michael Rash's (Cipherdyne) fwsnort in October 2008,[1] and I knew then I'd be sure to bring Single Packet Authorization (SPA)[2] to your attention. When Michael recently discussed "Creating Ghost Services with Single Packet Authorization"[3] on his website, I knew it was time. Each time I engage with one of Michael's offerings, I am reminded of what it is to encounter real genius. It's not enough that Cipherdyne tools work really well; they're also insanely cool. Further, Michael's one of those people who'd much rather talk about detective security applications than himself. If you haven't had the opportunity to explore Cipherdyne.org or attend one of Michael's presentations, be sure to do so. Enough embarrassing Mr. Rash, let's make good use of SPA.

Perhaps you're familiar with Port Knocking[4] where a service is protected by a default-drop packet filer. SPA is an authorization scheme provided by the FireWall KNock OPerator (fwknop) that exhibits all the advantages of Port Knocking but offers significant enhancements:

1. SPA requires only a single encrypted packet in order to communicate various pieces of information, including desired access through a firewall policy and/or complete commands to execute on the target system.
2. fwknop keeps iptables in a "default drop" stance, thus protecting services such as OpenSSH with an additional layer of security, making exploitation of vulnerabilities (both 0-day and unpatched code) much more difficult.
3. With fwknop deployed, port scanners looking for sshd won't find it; it makes no difference if a 0-day vulnerability exists or not.
4. The authorization server passively monitors authorization packets via libcap; there is no "server" connection in the traditional sense.
5. Access to a protected service is only granted after a valid encrypted and non-replayed packet is received from an fwknop client.
6. SPA can utilize asymmetric ciphers for encryption.
7. SPA packets are non-replayable.
8. SPA cannot be broken by trivial sequence busting attacks.
9. SPA only sends a single packet over the network; IDS will not interpret it as a port scan.
10. SPA is much faster because it only sends a single packet.[5]

SPA has been under continued development since 2004. As part of such ongoing efforts, Michael has a continued vision regarding SPA and fwknop; his recent feedback follows:

When the fwknop project was first started in 2004, it was written in Perl and focused on combining encrypted port knocking and passive OS fingerprinting. Although port knocking is interesting, in 2005 the Single Packet Authorization (SPA) functionality was added in order to solve many of the inherent protocol deficiencies that port knocking implementations are forced to live with. SPA allows a chunk of encrypted and non-replayable authentication information to be transmitted and passively received, and this means SPA is immune to simple DoS attacks, inability to use asymmetric ciphers, and replay attacks that port knocking suffers from.

The development pace of the fwknop project is fairly rapid, with over 30 new releases made since 2004. The latest release is 1.9.12, but at this point the development focus has shifted. The entire fwknop code base is being re-written in C to make fwknop more portable to embedded devices and dedicated systems (such as firewalls) where perl is not even installed. This has been one of the most important requests over the years from the community – to offer SPA in a lightweight C implementation. Progress on the C port of fwknop has been rapid, and the client side code as well as a new library "libfko" has been completed. Damien Stuart, an expert developer and a colleague of mine at a previous employer, has developed the C port including all of the libfko code and is currently working on the server side implementation. With the development of libfko, there are perl bindings via a new perl module "FKO" and the current fwknop-1.9.12 perl

1 http://holisticinfosec.org/toolsmith/docs/october2008.pdf.

2 http://www.cipherdyne.org/fwknop/docs/SPA.html.

3 http://www.cipherdyne.org/blog/2009/11/creating-ghost-services-with-single-packet-authorization.html.

4 http://www.portknocking.org.

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

ISSA Information Systems Security Association ©2010 Information Systems Security Association • www.issa.org • editor@issa.org • Permission for author use only.

code defaults to using this module if it is installed. This means that libfko C code is used for all of the heavy lifting in terms of building SPA packet data, handling the encryption/decryption cycle, and determining authorization levels. The next release of fwknop will introduce the completed C implementation of both the client and server components, and the perl code will be maintained as legacy functionality.

*One emphasis for the fwknop project has been to offer a sophisticated interface to the underlying firewall. In the case of iptables, this allows non-obvious use cases to crop up, such as the ability to request access to a service (such as SSHD) *through* a port where another service (such as Apache) is already bound. This works because iptables operates from within kernel space and is therefore in a position to interact with sockets at a level where there is no conflict with a user space application. So, new DNAT rules can send a connection from the IP specified in an SPA packet to port 80 through to port 22 on the local system without Apache ever seeing the connection. To anyone who is scanning, all they will ever see Apache on port 80 - SSHD never appears in their world. But to the SPA client, SSHD is perfectly accessible via port 80.*

Our real goal for this piece, aside from pointing out how useful SPA can be, is to promote the concept of ghost services over SPA wherein we bind a service through an entirely unexpected port. This approach may prove to be of much use to you in situations where only a specific service is allowed through a firewall and you need to data quickly and inconspicuously without requiring firewall changes. "Er," you say? "Because iptables operates on packets within kernel space, NAT functions apply before there is any conflict with a user space application such as Apache. This makes it possible to create "ghost" services where a port switches for a short period of time to whatever service is requested within an SPA packet (e.g. sshd), but everyone else always just sees the service that is normally bound there (e.g. Apache on port 80)."[6] Michael covers accessing sshd over port 80 in his November 29, 2009 Cipherdyne post; I will build on this to give you additional ideas for ghost services.

Installing fwknop

No matter your preference of Linux distribution, I must recommend installing fwknop from source. As I favor Ubuntu, the methodology described below was utilized on Ubuntu 9.10 (Karmic Koala) systems and is borrowed entirely from Gilbert Mendoza on the Ubuntu Community Documentation site.[7] Visit the site for a fully detailed version; following is the *Reader's Digest* version for brevity's sake. Gilbert's document describes protecting SSH with SPA and fwknop.

Prepare the server as follows:

```
$ sudo apt-get install fwknop-server
```

Prepare the client as follows:

```
$ sudo apt-get install fwknop-client
```

If you choose to install from source, the installer will prompt you for some answers. As an example, when installing on your SPA server, answer the installer script as follows:

- server (mode)
- pcap (data acquisition)
- the network interface of your choosing (eth0 is default)
- root@localhost (access alerts sent to)
- no (access alerts to different address)
- yes (enable fwknop at boot)

Now configure GnuPG authentication.

On the client host:

```
$ gpg --gen-key
$ gpg --list-key <client key>@localhost
pub  1024D/<client key ID> 2009-12-13
$ gpg -a --export <client key ID> > fwknop-
client.asc
```

Upload the client key to the server.

On the server:

```
$ gpg --gen-key
$ gpg --list-key <server key>@localhost
pub  1024D/<server key ID> 2009-12-13
$ gpg -a --export <server key ID> > fwknop-
server.asc
```

Upload the server key to the client.

On each host import the uploaded key.

On the client host:

```
$ gpg --import fwknop-server.asc
$ gpg --sign-key fwknopd@localhost
```

On the server:

```
$ gpg --import fwknop-client.asc
$ gpg --sign-key fwknop-client@localhost
```

It is essential that you then edit the fwknop configuration file `/etc/fwknop/access.conf`.

As you consider protecting other services with fwknop, remember to add their port number here.

```
SOURCE: ANY;
OPEN_PORTS: tcp/22;
DATA_COLLECT_MODE: PCAP;
GPG_HOME_DIR: /home/<user>/.gnupg;
GPG_DECRYPT_ID: <server key ID>;
GPG_DECRYPT_PW: <decrypt password>;
GPG_REMOTE_ID: <client key ID>;
FW_ACCESS_TIMEOUT: 30;
```

Now you're ready to start the daemon: `$ sudo /etc/init.d/fwknop start`

Creating a connection with fwknop

If you're behind a NAT firewall, include the "-w" flag; it queries whatismyip.com for the client's real IP address to be used

6 http://www.cipherdyne.org/blog/2009/11/creating-ghost-services-with-single-packet-authorization.html.

7 https://help.ubuntu.com/community/SinglePacketAuthorization.

©2010 Information Systems Security Association • www.issa.org • editor@issa.org • Permission for author use only.

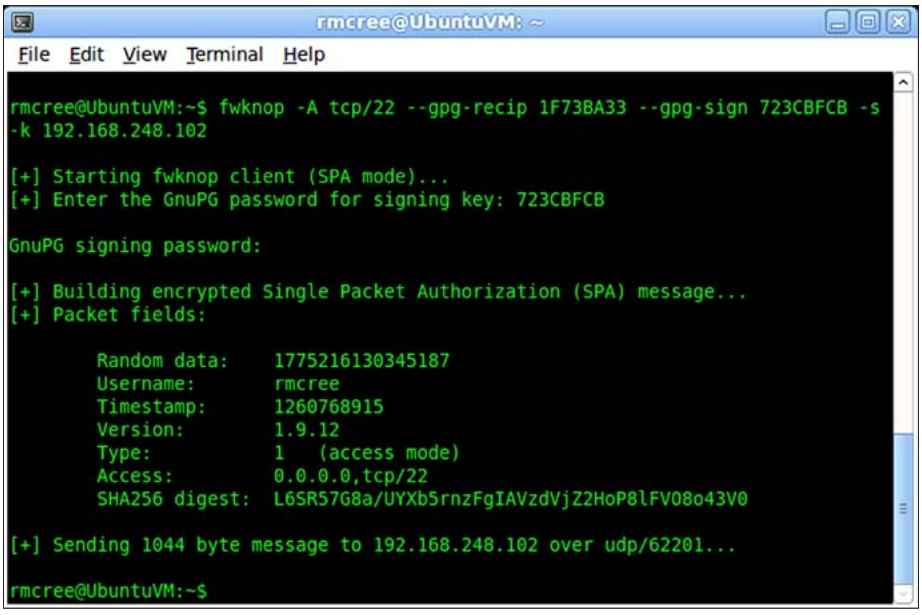


Figure 1 – Single Packet Authentication

as the source address. Otherwise the source IP would be a local address.

```
$ fwknop -A tcp/22 --gpg-recip <server key ID>
--gpg-sign <client key ID> -w -k <server IP
address>
```

If your client and server are on the same network, or NAT is not an issue, issue the following where the "-s" flag declares that the server should use the source address from which the SPA key originates.

```
$ fwknop -A tcp/22 --gpg-recip <server key ID>
--gpg-sign <client key ID> -s -k <server IP
address>
```

A properly authenticated session will produce results as seen in Figure 1.

After a successful SPA transaction you have 30 seconds to establish connection to the appropriate service, again in this case, SSH.

On the server, an excellent way to monitor activity while testing is to pass `$ tail -f /var/log/syslog | grep fwknop`.

You'll see results as exhibited in Figure 2, showing a successful SPA transaction and subsequent SSH session established.

Meanwhile a scan of the server host would show no open ports after 30 seconds even though an SSH session has been established.

There is also fwknop Windows UIs for you to consider including Daniel Lopez's Morpheus[8] and Sean Greven's older alpha client.[9]

8 http://cipherdyne.org/blog/2009/08/morpheus-fwknop-windows-ui-update.html.

9 http://cipherdyne.org/blog/2007/11/fwknop-windows-ui.html.

Creating an SPA ghost service

For this scenario two key changes need to be made to the configuration files:

In `/etc/fwnop/fwknop.conf` change ENABLE_IPT_FORWARDING to Y.

In `/etc/fwnop/access.conf` add ENABLE_FORWARD_ACCESS: Y; Also see PERMIT_CLIENT_PORTS reference below.

The prospects for creating ghost services with SPA are endless. The thought of hiding a service behind fwknop is useful enough, but bouncing ports to do so is all the more useful. Michael's discussion of SSH over port 80 should already have convinced you but there are endless additional options.

Here's one for the ultra-paranoid that mixes up all kinds of hide and seek. Work with me; it's a stretch but it shows how useful ghost services with SPA can be.

First, you're part of a highly secretive pen testing team working on site with a major corporate client. The client is surprisingly secure, monitors well, and locks outbound traffic to http and pop3. They proxy and content monitor http traffic as part of data leak prevention, so anything you do over port 80 is likely to be noticed. Further, they block the vast majority of web addresses, limiting access to just a few approved sites. All said and done, you'll get nowhere over port 80. However, traffic bound for port 110 is surprisingly not monitored due to an oversight. Helpful, but you still need to be very stealthy.

You've managed to sneak into their datacenter and deploy your workstation, but to further your foothold you need a customized Python script that didn't make it in your toolkit when you deployed to the client site. Your handler back at your office can present the file to you via a one-shot NetCat connection like this:

```
( cat PSKpredict.py; ) | nc -q 1 -l -p 6543
```

Your handler set the NetCat listener to an uncommon port (6543) and has left the firewall closed. Remember, you're a highly paranoid and hopefully stealthy organization. He has, however, allowed fwknop to open that port via access.conf (PERMIT_CLIENT_PORTS: Y;) when properly authenticated to.

Now you can bind fwknop to port 110 to sneak out of the client network and grab the file we need on the server back in the Batcave.

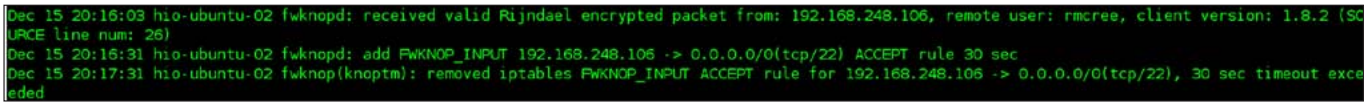


Figure 2 – Tail syslog for fwknop

©2010 Information Systems Security Association • www.issa.org • editor@issa.org • Permission for author use only.

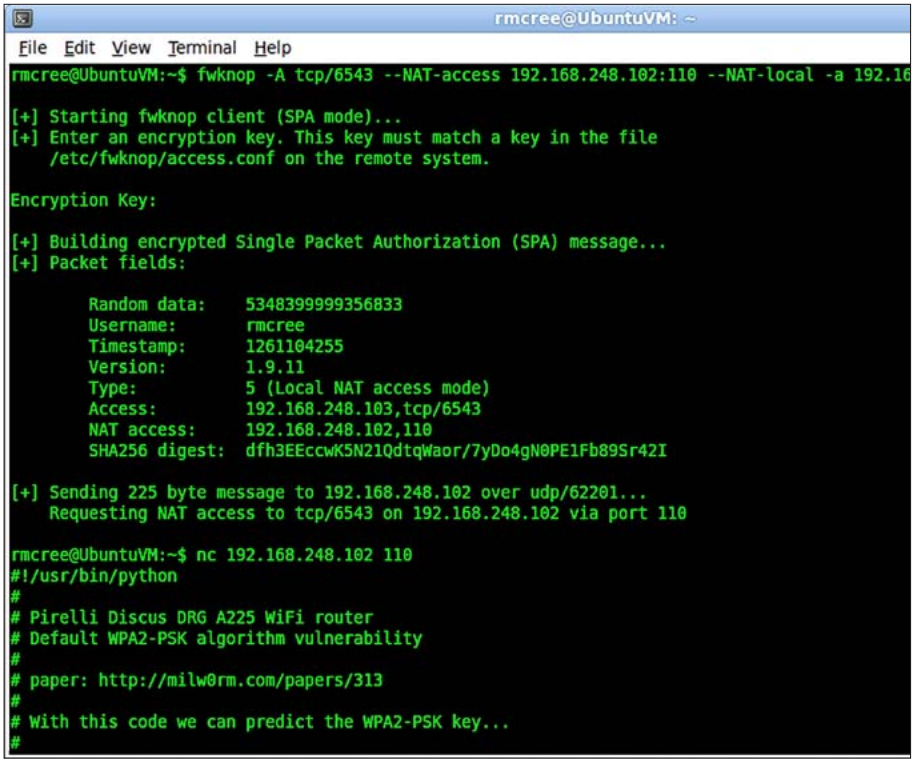


Figure 3 – NetCat one-shot file grab via port 110 with a closed firewall

Note: the time stamp embedded in the SPA packet must fall within 120 seconds of the servers clock or no love. Also remember to add relevant ports to `/etc/fwknop/access.conf`.

On the client system execute:

```
fwknop –A tcp/6543 –NAT-access <server IP>:110
–NAT-local –a <client IP> -D <server IP>
```

You'll be prompted for the password you set earlier. If you happened to be watching syslog on the SPA server you should see "received valid Rijndael encrypted packet…."

Now at the command prompt issue `nc <server IP> 110`.

Note that in Figure 3 fwknop builds the SPA message, then requests NAT access to tcp/6453 on the server via port 110. Then, the NetCat client requests a connection via port 110 and dumps the Python script to the console. Tidy, right?

Keep in mind that an nmap scan of the server would show it completely closed.

Wireshark analysis of a packet capture on the server system would appear as seen in Figure 4 where all connections are made to port 110, but we grabbed the file from a NetCat listener on port 6543. Figure 4 also shows the UDP request message via port 62201 (for the fwnopd passive listener). Stealthy, eh?

In Conclusion

Creating a "ghost in the machine" with SPA might serve you extremely well when traveling, or when conducting penetration tests and/or confidential audits. I've only offered one scenario; imagine moving database connections, syslog, or SNMP over ports such as 21, 25, or 443.

Look forward to the enhancement to fwknop as it is rewritten in C and watch ciberdyne.org for updates.

Cheers…until next month.

Acknowledgments

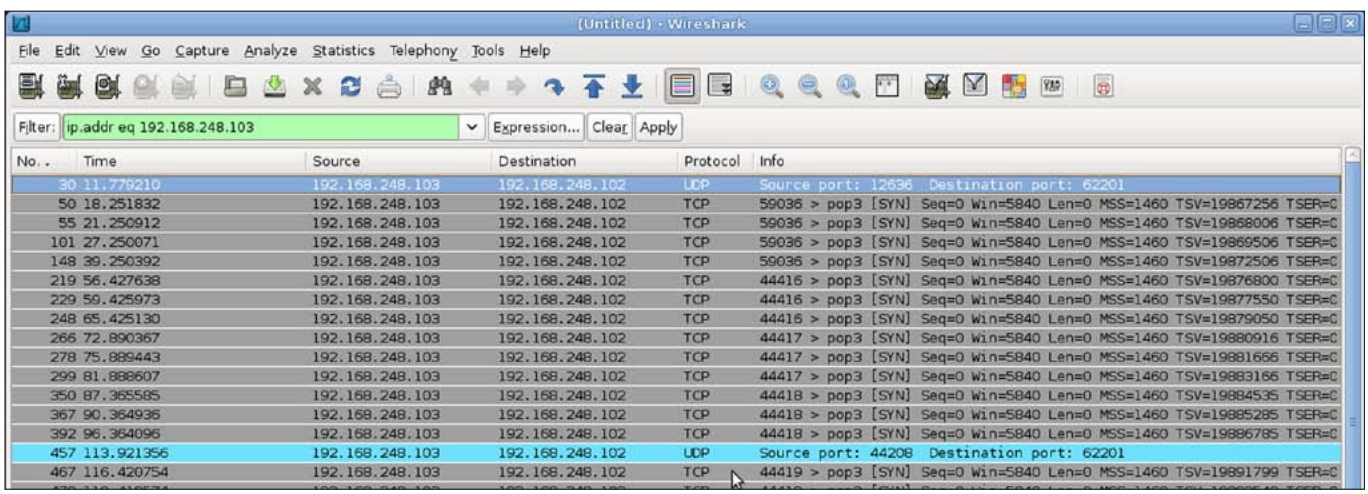
—Michael Rash: project lead

—Elliot Lazarus: incident handler, idea wrangler

About the Author

Russ McRee, GCIH, GCFA, GPEN, CISSP, is team leader and senior security analyst for Microsoft's Online Services Security Incident Management team. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ@holisticinfosec.org.

Figure 4 – Packet capture of SPA ghost session



©2010 Information Systems Security Association • www.issa.org • editor@issa.org • Permission for author use only.