MICHAEL RASH

single packet authorization with fwknop

Michael Rash holds a master's degree in Applied Mathematics and works as a security research engineer for Enterasys Networks, Inc. He is the lead developer of the cipherdyne.org suite of open source security tools, including PSAD and FWSnort, and is co-author of the book *Snort-2.1 Intrusion Detection,* published by Syngress.

mbr@cipherdyne.org

ONE YEAR AGO, IN THE DECEMBER 2004 issue of *;login:*, in the article entitled "Combining Port Knocking and Passive OS Fingerprinting with Fwknop," I described a technique for combining passive OS fingerprinting with a method of authorization called port knocking Since that time I have implemented a new method of securing IP-based communications called Single Packet Authorization (SPA) [1], which draws on some of the strengths of port knocking and fixes some of its weaknesses. Fwknop retains the ability to generate encrypted port knock sequences and incorporate additional criteria on the OS required to honor such sequences, but the default authorization method has been switched to SPA due to the benefits this strategy has over traditional port knocking.

This article discusses Single Packet Authorization as implemented by fwknop, suggests why you would want to use it, and provides an example of using fwknop to provide an additional layer of security for OpenSSH. Fwknop is free software released under the GNU Public License (GPL) and can be downloaded from http://www.cipherdyne.org/projects/fwknop/.

The Chief Innovations of Port Knocking

When the concept of port knocking [2] was announced in 2003, many competing implementations were rapidly developed. At last count, portknocking.org lists nearly 30 different software projects dedicated to the specific visions of port knocking promoted by their respective authors. Some of these projects are more complete than others, but in general they stay true to port knocking's chief innovation, the communication of information across sequences of connections to closed ports. The port numbers themselves, instead of the application payload portion of TCP segments or UDP datagrams, transmits the information as it is sent from the port knocking client to the server. Of course, the term "server" only applies to the portion of the port knocking scheme that is designed to passively receive packets; there is no traditional server that listens via the Berkeley sockets interface. The information typically sent in a port knock sequence communicates desired access through a packet filter that is protecting a particular service or set of services. The knock server gathers the knock sequence via a passive monitoring mechanism such as firewall-log monitoring or using libpcap to monitor packets as they fly by on the wire. This allows a kernel-level packet filter (such as Netfilter in the Linux kernel) to be configured in a default drop stance so that the only connections to a protected service that are allowed to be established are those that have first been associated with a valid port knock sequence. This is a powerful concept, because the end result is that code paths available to a would-be attacker are minimized. Even if an attacker possesses an exploit (0-day or otherwise) for a service that is actually deployed on a system, it is rendered useless, since a connection cannot even be established without first issuing a valid knock sequence. When an attacker uses the venerable Nmap with all of its sophisticated machinery to enumerate all instances of a vulnerable service accessible throughout a network, services protected in such a manner will not appear in the list.

Single Packet Authorization vs. Port Knocking

So far we have discussed the two most important ways that port knocking is used to enhance security: the passive communication of authentication information, and the server-side use of a packet filter to intercept all attempts to connect with a real server that are not associated with a knock sequence. These two features are also used in Single Packet Authorization to increase security, but this is where the similarities between port knocking and SPA abruptly end.

In port knocking schemes, the communication of information within packet headers, as opposed to the packet payload, severely limits the amount of data that can stilll be transferred effectively. The port fields of the TCP and UDP headers are 16 bits wide, so only two bytes of information can be transferred per packet in a traditional port knock sequence. This assumes that other fields within the packet header are not also made significant in terms of the knock sequence, but any conceivable implementation would be able to transmit much less information than a protocol that makes use of payload data. If two bytes of information were all that were required to communicate the desired access to a knock server, this would not be a significant issue, but it is not enough to simply create a mapping between a knock sequence (however short) and opening a port. We also want our messages to resist decoding by an attacker who may be in the enviable position of being able to monitor every packet emanating from the knock client. This requirement can be satisfied by using an encryption algorithm, but even a symmetric block cipher with a reasonable key size of, say, 128 bits forces at least eight packets to be sent at two bytes per packet.

As soon as multiple packets become involved, we need to try to ensure that the packets arrive in order. This implies that a time delay is added between each successive packet in the knock sequence. Simply blasting the packets onto the network as quickly as possible might result in out-of-order delivery by the time the packets reach their intended target. Because the knock server is strictly passively monitoring packets and consequently has no notion of a packet acknowledgment scheme, a reasonable time delay is on the order of about a half-second. Given a minimum of eight packets to send, we are up to four seconds just to communicate the knock sequence. In addition, if there were ever a need to send more information, say on the order of 100 bytes, the time to send such a message is longer than most people would be willing to wait. Single Packet Authorization has no such limitation, because the application payload portion of packets is used to send authentication data. The result is that up to the minimum MTU number of bytes of all networks between the client and server can be sent in a single message, and no cumbersome time delays need to be introduced. Fwknop uses this relatively large data size to communicate not only detailed access requirements in SPA messages, but also entire commands to be executed by the fwknop SPA server. Of course, all SPA messages are encrypted, and the algorithm currently supported by fwknop is the symmetric Rijndael cipher, but the upcoming 0.9.6 release will also support asymmetric encryption via GPG key rings and associated asymmetric cipher(s).

An additional consequence of sending multiple packets in a slow sequence is that it becomes trivial for an attacker to break any sequence as it is being sent by the port knocking client. All the attacker needs to do is spoof a duplicate packet from the source address of the client during a knock sequence. This duplicate packet would be interpreted by the knock server as part of the sequence, hence breaking the original sequence. Programs like hping (see http://www.hping.org) make it exceedingly easy to spoof IP packets from arbitrary IP addresses. Single Packet Authorization does not suffer from this type of easy injection attack.

In addition to making it difficult for an attacker to decode our messages, we also require that it not be possible for the attacker to replay captured messages against the knock server. A mechanism should be in place that makes it easy for the server to know which messages have been sent before and not to honor those that are duplicates of previous messages. It is not enough just to encrypt knock sequences even if the IP address to which the server grants access is buried within the encrypted sequence; consider the case where a knock client is behind a NAT device and the attacker is on the same subnet. If a knock sequence is sent to an external knock server, then the IP address that must be put within the encrypted sequence is the external NAT address. Because the attacker is on the same subnet, any connection originating from the attacker's system to the external knock server will come from the same IP as the legitimate connection. Hence the attacker need only replay a captured knock sequence from the client in order to be granted exactly the same access.

In the world of traditional port knocking there are ways to prevent replay attacks, such as altering knock sequences based upon time, iterating a hashing function as in the S/KEY system [3], or even manually changing the agreed upon encryption key for each successful knock sequence. However, each of these methods requires keeping state at both the client and the server and does not scale well once lots of users become involved. It turns out that Single Packet Authorization facilitates a more elegant solution to the replay problem. By having the SPA client include 16 bytes of random data in every message and then tracking the MD5 (or other hashing function) sum of every valid SPA message, it becomes trivial for the server to not take any action for duplicate messages. The ability to send more than just a few bytes of data within an SPA message is the essential innovation that really makes this possible. Fwknop implements exactly this strategy, which will be demonstrated in the example below.

Port knocking schemes generally use the port number within the TCP or UDP header to transmit information from the knock client to the knock server. However, there are lots of IP protocols, such as ICMP and GRE, that have space reserved for application-layer data but have no corresponding notion of a "port." Theoretically, SPA messages can be sent over any IP protocol, not just those that provide a port over which data is communicated. One such protocol currently supported by fwknop is ICMP.

Finally, to an observer of network traffic, a port knock sequence is indistinguishable from a port scan— that is, it is a series of connections to various port numbers from a single IP address. Many network intrusion detection systems have the capability of detecting port scans, and have no way to know that a port knock sequence is not an attempt to enumerate the set of services that are accessible from the IP address of the client system. Hence, any intermediate IDS that has its port scan thresholds set low enough (i.e., the number of packets associated with a port knock sequence exceeds the thresholds within a given period of time) will generate port scan alerts for each port knocking sequence. Although this by itself does not create a problem for port knocking implementations in terms of the port knocking protocol, it can draw undue attention to anyone actually using port knocking on a network that is monitored by an IDS. By contrast, Single Packet Authorization does not create a significant enough network footprint to generate an IDS port scan alert.

Fwknop Single Packet Authorization Message Format

In order for an fwknop SPA client to authenticate and allow application of the subsequent authorization criteria [4], several pieces of information must be securely communicated to the knock server. An fwknop client transmits the following within each SPA message:

- 16 bytes of random data
- local username
- local timestamp
- fwknop version
- mode (access or command)
- desired access (or command string)
- MD5 sum

The 16 bytes of random data ensures that each SPA message has an extremely high probability of being unique and hence allows the fwknop server to maintain a cache of previously seen messages in order to thwart replay attacks. The local username enables the fwknop server to distinguish between individual users so that different levels of access can be granted on a per-username basis. The version number allows the fwknop message format to be extended while preserving backwards compatibility for older versions of the software. The mode value instructs the server that the client either wishes to gain access to a service or run a command, each of which is specified in the next field. The MD5 sum is calculated over the entire message and is then used by the server to verify message integrity after a successful message decrypt. All the above values are concatenated with ":" characters (with base64 encoding applied where necessary so as not to break the field separation convention), and the entire message is then encrypted with the Rijndael symmetric block cipher. A symmetric key up to 128 bits long is shared between the fwknop SPA client and the SPA server.

Fwknop in Action

Now let us turn to a practical example: we will illustrate how fwknop is used in the default Single Packet Authorization mode to protect and gain access to the OpenSSH daemon. First, we configure the fwknop server to allow access to TCP port 22 by the “mbr” username once a valid SPA message is monitored. This is accomplished by adding the following lines to the file /etc/fwknop/access.conf:

```
SOURCE: ANY;
OPEN_PORTS: tcp/22;
KEY: <encrypt_key>;
FW_ACCESS_TIMEOUT: 10;
REQUIRE_USERNAME: mbr;
DATA_COLLECT_MODE: ULOG_PCAP;
```

In server mode, fwknop can acquire packet data by using libpcap to sniff packets directly off the wire or out of a file that is written to by a separate sniffer process, or by using the Netfilter ulogd pcap writer [5]. In this case the configuration keyword DATA_COLLECT_MODE instructs the server to respect SPA messages that are collected via the ulogd pcap writer. For this example, let us assume that the IP address on the server system is 192.168.10.1, that fwknop is running in server mode, and that Netfilter has been configured to drop all packets destined for TCP port 22 by default.

Now, on the client (which has IP 192.168.20.2), we first verify that we cannot establish a TCP connection with sshd:

```
[client]$  nc -v 192.168.10.1 22
```

So far, so good. The netcat process appears to hang because we fail to even receive a reset packet back from the TCP stack on the server; Netfilter has dropped our SYN packet on the floor before it can hit the TCP stack. Having a completely inaccessible server is not of much use, of course, so now we execute the following to gain access to sshd:

```
[client]$  fwknop -A tcp/22 -w -k 192.168.10.1
[+] Starting fwknop in client mode.
[+] Enter an encryption key. This key must match a key in the file
/etc/fwknop/access.conf on the remote system.

Encryption Key:

[+] Building encrypted single-packet authorization (SPA) message...
[+] Packet fields:

Random data: 5628557594764037
Username:    mbr
Timestamp:   1132121405
Version:     0.9.5
Action:      1 (access mode)
Access:      192.168.20.2,tcp/22
MD5 sum:     q8vIpYY6q3qEflaFtU3Jag

[+] Sending 128 byte message to 192.168.10.1 over udp/62201...
```

Sure enough, we are now able to establish a TCP connection with port 22:

```
[client]$  nc -v 192.168.10.1 22
192.168.10.1 22 (ssh) open
SSH-2.0-OpenSSH_3.9p1
```

Fwknop running on the server has reconfigured Netfilter to allow the client IP address to talk to sshd. Even though fwknop will expire under the access rule after 10 seconds, by using the Netfilter connection-tracking capability to accept packets that are part of established TCP connections before packets are dropped, the SSH session remains active for as long as we need it.

Finally, to illustrate the ability of fwknop to detect and stop replay attacks, suppose that an attacker were able to sniff the SPA message above as it was sent from the client to the server (by default, fwknop sends SPA messages over UDP port 62201, but this can be changed via the -p command line argument):

```
[attacker]#  tcpdump -i eth0 -c 1 -s 0 -l -nn -X udp port 62201
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535
bytes
01:44:12.170787 IP 192.168.20.2.32781 > 192.168.10.1.62201: UDP,
length: 128
    0x0000:  4500 009c 246a 4000 4011 768f c0a8 1406  E...$j@.@.v.....
    0x0010:  c0a8 0a01 800d f2f9 0088 9fc3 6736 576a  ............g6Wj
    0x0020:  5234 7374 4941 4358 3935 4152 6541 4778
R4stIACX95AReAGx
    0x0030:  3342 7848 7569 7776 786e 557a 3531 5131  3BxHui-
wvxnUz51Q1
    0x0040:  5532 3976 4872 7144 6e69 3330 514f 4d72
U29vHrqDni30QOMr
    0x0050:  6661 5a48 4845 304c 3631 4767 636a 6e37
faZHHE0L61Ggcjn7
    0x0060:  6a64 7a6e 787a 726c 4f53 314c 5051 6877
jdznxzrlOS1LPQhw
    0x0070:  394b 424f 3963 6b61 5232 2b6f 5474 736c
9KBO9ckaR2+oTtsl
    0x0080:  574d 484c 574f 7736 7468 4161 7a58 3976
WMHLWOw6thAazX9v
    0x0090:  2b65 6746 6352 2f2f 6776 4352            +egFcR//gvCR
1 packets captured
2 packets received by filter
0 packets dropped by kernel
```

Now the attacker can replay the encrypted SPA message on the network as follows in an effort to gain the same access as the original message [6]:

```
[attacker]$ echo
"g6WjR4stIACX95AReAGx3BxHuiwvxnUz51Q1U29vHrqDni30QOMrfaZ
HHE0L61Ggcjn7jdznxzrlOS1LPQhw9KBO9ckaR2+oTtslWMHLWOw6th
AazX9v+egFcR//gvCR"
|nc -u 192.168.10.1 62201
```

On the server, this results in the following syslog message, indicating that fwknop monitored the message replay and took no further action:

```
Nov 16 01:50:11 server fwknop: attempted message replay from:
192.168.20.6
```

Conclusion

Single Packet Authorization has several characteristics that make it more powerful and flexible than port knocking for protecting network services. Its data transmission capabilities, coupled with its clean strategy for preventing replay attacks, make it an ideal candidate for expanding the configuration of packet filters to drop all connections to some critical services by default. This makes the exploitation of vulnerabilities within such services much more difficult, because an arbitrary IP address cannot enumerate or interact with these services until a valid SPA message is generated.

REFERENCES

[1] MadHat was the first person to coin the term “Single Packet Authorization” at the BlackHat Briefings in July of 2005. However, the first available implementation of SPA was in the 0.9.0 release of fwknop in May of 2005 (with SPA code available via the http://www.cipherdyne.org/ CVS repository dating back to March of 2005; see http://www.cipherdyne.org/cgi/viewcvs.cgi/fwknop/fwknop).

[2] M. Krzywinski, “Port Knocking: Network Authentication Across Closed Ports,” *SysAdmin Magazine* 12 (2003): 12–17.

[3] RFC 1760: The S/KEY One-Time Password System.

[4] The terms “authentication” and “authorization” in this context are commonly construed to mean the same thing. However, authentication refers to the verification that a communication from one party to another actually came from the first party, whereas authorization essentially refers to the process of verifying whether one party is allowed to communicate with a second party at all.

[5] See the Netfilter ulogd project: http://www.gnumonks.org/projects/ulogd/.

[6] Even if the replay were successful, access would only be granted for the IP address of the client, which is encrypted within the SPA message and hence not available to the attacker. If, however, the client is behind a NAT address, this may not matter because the external address would be the same, so it is important to stop replay attacks regardless of whether the client address is encrypted within the SPA message.