13

INTRODUCING FWKNOP

The FireWall KNock OPerator (fwknop, see http://www.cipherdyne.org/fwknop) was released as an open source project under the GNU Public License (GPL) in June 2004. It was the first port-knocking implementation to combine encrypted port knocking with passive OS fingerprinting, making it possible to allow only Linux systems to connect to your SSH daemon. (The TCP stack of the port-knocking client system acts as an additional authentication parameter.) fwknop's port-knocking component is based on iptables log messages, and it uses iptables as the default-drop packet filter.

In May 2005, I released the Single Packet Authorization mode for fwknop, so fwknop became the first publicly available SPA software. As of this writing, fwknop-1.0 is the latest available release, and the SPA method of authentication is the default, even though fwknop continues to support the old port-knocking method. MadHat coined the term *Single Packet Authorization* at Black Hat Briefings in July 2005. I submitted a similar proposal for presentation at the same conference, but *Single Packet Authorization* rolls off the tongue a lot easier than my title, which was *Netfilter and Encrypted, Non-replayable, Spoofable,* *Single Packet Remote Administration.* It is also worth noting that a protocol implemented by the tumbler project (http://tumbler.sourceforge.net) is similar to SPA in the sense that it only uses a single packet to transmit authentication and authorization information; its payload is hashed instead of encrypted, however, and this results in a significantly different architecture.

NOTE *fwknop really supports both* authentication*—the process of verifying the digital identity of an entity that is communicating something—and* authorization*—the process of trying to determine whether an entity has permission to perform an operation—of remote clients that wish to access a service behind the default-drop packet filter. These two processes are not the same, and both are important in their own right.*

fwknop Installation

Installing fwknop begins with downloading the latest source tarball or RPM from http://www.cipherdyne.org/fwknop/download. As usual, it is prudent to verify the MD5 sum; it is even better, from a security perspective, to use GnuPG to see if the GnuPG signature checks out.[1] Once you're sure that the downloaded file is safe, you can proceed with the installation. Here's how to install the source tarball of fwknop version 1.0:

```
$ cd /usr/local/src
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2.md5
$ md5sum -c fwknop-1.8.1.tar.bz2.md5
$ fwknop-1.8.1.tar.bz2: OK
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2.asc
$ gpg --verify fwknop-1.8.1.tar.bz2.asc
gpg: Signature made Wed Jun 6 01:27:16 2007 EDT using DSA key ID A742839F
gpg: Good signature from "Michael Rash <mbr@cipherdyne.org>"
gpg:                 aka "Michael Rash <mbr@cipherdyne.com>"
$ tar xfj fwknop-1.8.1.tar.bz2
$ su -
Password:
# cd /usr/local/src/fwknop-1.8.1
# ./install.pl
```

As with the installation of psad in Chapter 5, the install.pl script will prompt you for several bits of information, such as the authorization mode (i.e., whether you want to use the SPA mode or the legacy port-knocking mode) and the interface on which you would like fwknop to sniff packets.

You can install fwknop on a system that only supports sending SPA packets as an SPA client, or on a system with full support for sending SPA packets as well as sniffing them from the network (this is the default). A full installation of fwknop results in the creation of several files and directories in the filesystem in order to support normal operations, as follows.

[1] As mentioned in Chapter 5, my GnuPG key is available from http://www.cipherdyne.org/public_key. It is necessary to import this key with `gpg --import` in order to verify the GnuPG signature for each software distribution file at http://www.cipherdyne.org.

/usr/bin/fwknop This is the client program responsible for accepting password input from the user; constructing SPA packets that conform to the fwknop packet format; encrypting packet data with the Rijndael symmetric cipher or by interfacing with GnuPG for asymmetric encryption; and sending the encrypted SPA packet via UDP, TCP, or ICMP. By default, fwknop sends SPA packets over UDP port 62201, but this can be changed from the command line.

/usr/sbin/fwknopd This is the main daemon responsible for sniffing and decrypting SPA packet data, guarding against replay attacks, decoding the fwknop SPA packet format, verifying access rights, and reconfiguring the local iptables policy to grant temporary access to service(s) requested within SPA packets.

/usr/bin/fwknop_serv This is a simplistic TCP server that is only used if SPA packets are sent over the Tor anonymizing network (http://tor.eff.org). Use of this server results in bidirectional communication, so it technically breaks the usual unidirectional nature of the SPA protocol; see "SPA over Tor" on page 254 for more information.

/usr/lib/fwknop The Perl modules fwknop uses are installed within this directory in order to keep the system Perl library tree clean. Among the installed modules are `Net::Pcap`, `Net::IPv4Addr`, `Net::RawIP`, `IPTables::Parse`, `IPTables::ChainMgr`, `Unix::Syslog`, `GnuPG::Interface`, `Crypt::CBC`, and `Crypt::Rijndael`. The install.pl script is careful to install only Perl modules that do not already exist within the system Perl library tree, in order to minimize disk utilization. However, you can force install.pl to install all required Perl modules by using the `--force-mod-install` command-line argument. The `IPTables::Parse` and `IPTables::ChainMgr` modules are never installed on systems running the ipfw firewall, or on client-only installs of fwknop on Windows under Cygwin.

/etc/fwknop This is the main directory for fwknop daemon configuration files such as fwknop.conf and access.conf. This directory is used by fwknop daemons when running in server mode, and it is not needed to generate an SPA packet in client mode.

/usr/sbin/knopmd This is a daemon used to parse iptables log messages out of the /var/lib/fwknop/fwknopfifo named pipe. This daemon is only used if fwknop is being run in the legacy port-knocking mode.

/usr/sbin/knoptm This is a daemon that removes rule entries from the iptables chains to which fwknop has added access rules for legitimate SPA clients. This daemon is necessary because the main fwknopd daemon is sniffing from a live interface and the OS does not schedule it to run until a packet is received by the interface. The knoptm daemon is not used if fwknopd is reading packet data from a PCAP file that is being updated either by a separate sniffer process or by ulogd. In this case, fwknopd is periodically scheduled to run, regardless of whether a packet is received on an interface; hence, fwknopd can enforce timeouts against iptables rules on its own.

/usr/sbin/knopwatchd This is a monitoring daemon that restarts a daemon if it dies. However, fwknop is generally quite stable, so knopwatchd does not usually have very much work to do; it exists merely as a precautionary measure, since running SPA implies that nothing can access a protected service unless fwknopd is also running.

/etc/init.d/fwknop This is the initialization script for fwknop. It allows the user to start fwknop in a manner that is consistent with most Linux distributions—by executing `/etc/init.d/fwknop start`. Using the init script only makes sense in the context of starting fwknop in server mode.

fwknop Configuration

In server mode, fwknop references two main configuration files, fwknop.conf and access.conf, for configuration directives. Like the psad configuration files (see Chapter 5), within these files each line follows the simple key-value convention for defining configuration variables. As usual, comment lines begin with a hash mark (`#`). I'll present a selection of the more important configuration variables from these files in the following sections.

/etc/fwknop/fwknop.conf

The fwknop.conf file defines critical configuration variables such as the authentication mode, the firewall type, the interface to sniff packets from, whether packets should be sniffed promiscuously (i.e., whether or not fwknop processes Ethernet frames that are not destined for the MAC address of the local interface), and the email address(es) to which alerts are sent.

AUTH_MODE

The `AUTH_MODE` variable tells the fwknop daemon how to collect packet data. Several collection modes are supported, including sniffing packets from a live interface via the `Net::Pcap` Perl module, reading PCAP-formatted packets from a file in the filesystem that is written by ulogd (see http://www.netfilter.org), using a separate Ethernet sniffer such as tcpdump, or parsing iptables log messages from the file /var/log/fwknop/fwdata. Possible values for the `AUTH_MODE` variable are `PCAP`, `FILE_PCAP`, `ULOG_PCAP`, and `KNOCK`; `PCAP` is the default.

```
AUTH_MODE                   PCAP;
```

PCAP_INTF

The `PCAP_INTF` variable defines the live interface the fwknop daemon uses to monitor packets. This is only used if `AUTH_MODE` is set to `PCAP`; the default setting is the `eth0` interface.

```
PCAP_INTF                   eth0;
```

PCAP_FILTER

A live interface may transmit or receive lots of packet data that is completely unrelated to SPA traffic, and there is no need to force the fwknop daemon to process it. The `PCAP_FILTER` variable allows you to restrict the types of packets libpcap passes into fwknop based upon criteria such as network layer addresses or transport layer port numbers. Because, by default, fwknop transfers SPA packets over UDP port 62201, this variable is set as follows (this can be modified to acquire SPA packets over different ports and/or protocols).

```
PCAP_FILTER                 udp port 62201;
```

ENABLE_PCAP_PROMISC

When set to `Y`, this variable instructs the fwknop daemon to monitor all Ethernet frames that are sent past the live packet capture interface (i.e., the interface is operating in *promiscuous mode*). This is enabled by default when `AUTH_MODE` is set to `PCAP`; however, if the interface where the fwknop daemon is sniffing is active and has an IP address assigned—meaning SPA packets can be sent directly to this interface—then this feature can be disabled as follows:

```
ENABLE_PCAP_PROMISC         N;
```

FIREWALL_TYPE

The `FIREWALL_TYPE` variable tells fwknopd about the type of firewall that it is responsible for reconfiguring after receiving a valid SPA packet. Supported values are `iptables` (the default), and `ipfw` for FreeBSD and Mac OS X systems.

```
FIREWALL_TYPE               iptables;
```

PCAP_PKT_FILE

If `AUTH_MODE` is set to either `FILE_PCAP` or `ULOG_PCAP`, then the fwknop daemon acquires packet data from a PCAP-formatted file within the filesystem. The path to this file is defined by the `PCAP_PKT_FILE` variable and is set to the following default:

```
PCAP_PKT_FILE               /var/log/sniff.pcap;
```

IPT_AUTO_CHAIN1

The `IPTables::ChainMgr` Perl module is used by fwknop to add and remove `ACCEPT` rules for legitimate SPA clients. The `IPTables::ChainMgr` is also used by psad, but instead of adding `ACCEPT` rules, psad adds `DROP` rules against IP addresses that send malicious traffic. The default configuration for the `IPT_AUTO_CHAIN1` variable is to add `ACCEPT` rules into the custom iptables chain `FWKNOP_INPUT` and jump packets into this chain from the built-in `INPUT` chain.[2]

```
IPT_AUTO_CHAIN1         ACCEPT, src, filter, INPUT, 1, FWKNOP_INPUT, 1;
```

[2] A detailed explanation of the `IPT_AUTO_CHAIN{n}` variables can be found in "Configuration Variables" on page 135. The `IPT_AUTO_CHAIN{n}` variables provide an interface to the `IPTables::ChainMgr` module, and this interface is used in both psad and fwknop.

ENABLE_MD5_PERSISTENCE

One of the most important features of the SPA protocol is the ability to detect and ignore replay attacks. The `ENABLE_MD5_PERSISTENCE` variable controls whether or not the fwknop daemon writes the MD5 sums of all successfully decrypted SPA packets to disk. This allows fwknop to detect replay attacks across restarts of fwknop and even across system reboots. This feature is enabled by default, but can be disabled if you wish to verify that replay detection functions correctly (requires sending a duplicate SPA packet over the network to the SPA server).

```
ENABLE_MD5_PERSISTENCE      Y;
```

MAX_SPA_PACKET_AGE

The `MAX_SPA_PACKET_AGE` variable defines the maximum age, in seconds, for which the fwknop server will allow an SPA packet to be accepted. The default is two minutes. This variable is only used if `ENABLE_SPA_PACKET_AGING` is enabled.

```
MAX_SPA_PACKET_AGE          120;
```

ENABLE_SPA_PACKET_AGING

By default, the fwknop daemon requires that an SPA packet sent from the fwknop client is less than 120 seconds (two minutes) old, as defined by the `MAX_SPA_PACKET_AGE` variable discussed above. The fwknop client includes a timestamp within each SPA packet (see "fwknop SPA Packet Format" on page 241), which the fwknop server uses to determine the age of all SPA packets. This feature requires loose time synchronization between the fwknop client and server, but the robust Network Time Protocol (NTP) makes this easy to do.

If `ENABLE_SPA_PACKET_AGING` is disabled, an attacker inline with an SPA packet could stop the packet from being forwarded, thus preventing the fwknop server from seeing it and calculating its MD5 sum. Later, the attacker could send the original SPA packet against its destination, and the fwknop server would honor it. Further, if the fwknop `-s` command-line argument was used to generate the original SPA packet, fwknop would honor the SPA packet from whichever source IP address it came from (see the variable `REQUIRE_SOURCE_ADDRESS` below), and the attacker would gain access through the iptables policy.[3] Therefore, it is highly recommended that you leave this feature enabled.

```
ENABLE_SPA_PACKET_AGING     Y;
```

REQUIRE_SOURCE_ADDRESS

The `REQUIRE_SOURCE_ADDRESS` variable tells the fwknop server to require that all SPA packets contain the IP address within the encrypted payload that is to be granted access through iptables. With this feature enabled, the `0.0.0.0`

[3] This attack was called to my attention by Sebastien Jeanquier, and the result was the `ENABLE_SPA_PACKET_AGING` feature (first available in the 0.9.9 release) to implement the time window in which an SPA packet would be accepted by the fwknop server.

wildcard IP address placed within an SPA packet with the -s argument on the fwknop client command line will not be accepted.

```
REQUIRE_SOURCE_ADDRESS       Y;
```

EMAIL_ADDRESSES

The fwknop server sends email alerts under various circumstances, such as when SPA packets are accepted and access to a service is granted, when access is removed, and when a replay attack has been thwarted. Multiple email addresses are supported as a comma-separated list, like so:

```
EMAIL_ADDRESSES              root@localhost, mbr@cipherdyne.org;
```

GPG_DEFAULT_HOME_DIR

The GPG_DEFAULT_HOME_DIR variable specifies the path to the directory where GnuPG keys are kept for digital signature verification and decryption of SPA packets. The default is to use the .gnupg directory in root's home directory.

```
GPG_DEFAULT_HOME_DIR        /root/.gnupg;
```

ENABLE_TCP_SERVER

The ENABLE_TCP_SERVER variable controls whether or not fwknop binds a TCP server to a port to accept SPA packet data. If you want to route SPA packets over the Tor network, which only uses TCP for data transport, you must enable this feature. (You'll find more on this topic in "SPA over Tor" on page 254.) This feature is disabled by default.

```
ENABLE_TCP_SERVER           N;
```

TCPSERV_PORT

The TCPSERV_PORT variable specifies the port on which the fwknop_serv daemon listens for TCP connections. This is only used by fwknop if ENABLE_TCP_SERVER is enabled. The default is the following:

```
TCPSERV_PORT                62201;
```

/etc/fwknop/access.conf

The section on the fwknop.conf file gave lots of information about macro-level configuration options for fwknop, but it left out a discussion of important topics such as decryption passwords and authorization rights assigned to users. I'll rectify this by presenting the fwknop access.conf file, which defines all usernames, authorization rights, decryption keys, iptables rule time-outs, and command channels that the fwknop server uses.

SOURCE

Authorization of multiple users from arbitrary IP addresses is supported by fwknop; each user may use different encryption keys (and associated encryption algorithms). `SOURCE` is the main partitioning variable that allows fwknop to determine the access level of a valid SPA packet, and each group of configuration variables within the access.conf file defines a complete `SOURCE` access definition. The access.conf file supports multiple `SOURCE` access definitions. The default value for the `SOURCE` variable instructs fwknop to validate an SPA packet from any source IP address as shown below, but individual IP addresses and CIDR networks are also supported.

```
SOURCE: ANY;
```

OPEN_PORTS

The `OPEN_PORTS` variable instructs fwknop to grant access to the specified ports by reconfiguring the local iptables policy. Unless the `PERMIT_CLIENT_PORTS` variable (see below) is set to `Y`, the client cannot gain access to any services other than those listed by `OPEN_PORTS`. The following definition allows a valid SPA packet to reconfigure iptables to allow access to TCP port 22 (SSHD).

```
OPEN_PORTS: tcp/22;
```

PERMIT_CLIENT_PORTS

When set to `Y`, this variable allows the fwknop client to dictate to the fwknop server the set of traffic (i.e., ports and protocols) that will be allowed through the iptables policy, instead of the fwknop server only reconfiguring iptables to allow the traffic defined by the `OPEN_PORTS` variable. An SPA packet may contain several ports that the client wishes to access (see "fwknop SPA Packet Format" on page 241 for more information).

```
PERMIT_CLIENT_PORTS: Y;
```

ENABLE_CMD_EXEC

When enabled, this variable allows authorized SPA clients to have the fwknop server execute a command on their behalf. This feature is controversial because fwknop (as of the 1.0 release) executes these commands as root, although the ability to run commands as less privileged users is in development. The `ENABLE_CMD_EXEC` feature must be explicitly and deliberately enabled if you want to use it.

```
ENABLE_CMD_EXEC: Y;
```

CMD_REGEX

The `CMD_REGEX` variable allows you to provide a regular expression that must match a command supplied by an fwknop client before the fwknop server will execute it. It only makes sense to use this variable in the context of setting ENABLE_CMD_EXEC to Y. For example, to limit the commands the fwknop server will execute on behalf of an fwknop client to variations on the mail command, you could use the following:

```
CMD_REGEX: ^mail\s+\-s\s+\"\w+\"\s+\w+\@\w+\.com;
```

DATA_COLLECT_MODE

The DATA_COLLECT_MODE variable accepts the same packet collection modes as the AUTH_MODE variable in the fwknop.conf file. This allows each SOURCE access definition in the access.conf file to be independently enabled or disabled, depending on the value of the AUTH_MODE variable. Only those SOURCE access definitions with a DATA_COLLECT_MODE value that matches the AUTH_MODE variable are enabled. However, the DATA_COLLECT_MODE variable is optional, and if it is left out of the access.conf file, the fwknop daemon assumes that it is set to PCAP, the most common setting.

```
DATA_COLLECT_MODE: PCAP;
```

REQUIRE_USERNAME

The REQUIRE_USERNAME variable refers to the username of the user on a remote system who executes the fwknop client to generate an SPA packet. This username is included within all SPA packets (see "fwknop SPA Packet Format" on page 241 for more information). The remote username allows fwknop to apply authorization rules to incoming SPA packets. The REQUIRE_USERNAME variable supports multiple usernames, which can be useful if there is a site or system-wide encryption key for multiple users on the client side.

```
REQUIRE_USERNAME: mbr,mrash;
```

FW_ACCESS_TIMEOUT

The FW_ACCESS_TIMEOUT variable tells the fwknop server the number of seconds for which any iptables ACCEPT rules should be instantiated within the FWKNOP_INPUT chain, allowing access to the services requested by a valid SPA packet.

```
FW_ACCESS_TIMEOUT: 30;
```

KEY

The KEY variable defines the encryption key used for decrypting SPA packets that have been encrypted with the Rijndael block cipher. It requires an argument that is at least eight characters long.

```
KEY: yourencryptkey;
```

GPG_DECRYPT_ID

The GPG_DECRYPT_ID variable specifies a unique identifier for the fwknop server's GnuPG public key, which is used by an fwknop client to encrypt the SPA packet. This unique identifier can be obtained from the output of the gpg --list-keys command and is normally a string of eight hex characters.

```
GPG_DECRYPT_ID: ABDC1234;
```

GPG_DECRYPT_PW

The GPG_DECRYPT_PW variable holds the decryption password for the fwknop server's GnuPG public key, which is used by an fwknop client for encryption. Because this password is contained within a plaintext file, you should generate a new GnuPG key to be used only as the fwknop server key, rather than using a valuable GnuPG key that you might also use for other things, like confidential email communications.[4]

```
GPG_DECRYPT_PW: gpgdecryptionpw;
```

GPG_REMOTE_ID

The GPG_REMOTE_ID variable contains a unique identifier for the GnuPG key that an fwknop client uses to digitally sign an SPA packet. This key needs to be imported into the fwknop server key ring (see "SPA via Asymmetric Encryption" on page 246).

```
GPG_REMOTE_ID: DEFG5678;
```

Example /etc/fwknop/access.conf File

Next, you'll put all of this information together and create a complete access.conf file that you can use to protect your SSH server. (You'll find operational examples in "Deploying fwknop" on page 243.)

With your favorite editor, open the /etc/fwknop/access.conf file and add the configuration directives listed below.

```
# cat /etc/fwknop/access.conf
SOURCE: ANY;
OPEN_PORTS: tcp/22;
FW_ACCESS_TIMEOUT: 30;
REQUIRE_USERNAME: mbr;
KEY: mypassword;
GPG_DECRYPT_PW: gpgdecryptpassword;
GPG_HOME_DIR: /root/.gnupg;
GPG_REMOTE_ID: 5678DEFG;
GPG_DECRYPT_ID: ABCD1234;
```

[4] fwknop can acquire secret key information from gpg-agent.

SOURCE: ANY means that the fwknop daemon will accept a valid SPA packet from any source IP address. This is handy if you are on the road and cannot predict which network your laptop or other system will be connected to.

OPEN_PORTS: tcp/22 means that the fwknop daemon will grant temporary access through the local iptables firewall with an ACCEPT rule to the SSH port. The ACCEPT rule is removed after 30 seconds, as specified by the FW_ACCESS_TIMEOUT variable.

REQUIRE_USERNAME: mbr forces the remote username that runs the fwknop client to be mbr. In this case, the fwknop daemon is configured to accept an SPA packet that has been symmetrically encrypted with Rijndael (KEY: mypassword) or asymmetrically encrypted (GPG_DECRYPT_PW: gpgdecryptpassword) with a GnuPG key (usually with the Elgamal cipher). For SPA packets that are encrypted with GnuPG, the fwknop daemon requires that the ID of the remote signing key is 5678DEFG, and the ID of the local decryption key is ABCD1234—see the GPG_REMOTE_ID and GPG_DECRYPT_ID variables, respectively.

fwknop SPA Packet Format

Every SPA packet is constructed according to a well-defined set of rules. These rules allow the fwknop server to be confident about the type of access that is being requested through the iptables firewall and who is requesting it. After accepting user input from the fwknop client command line (see "SPA via Symmetric Encryption" on page 244 and "SPA via Asymmetric Encryption" on page 246), each SPA packet contains the following:

Random data (16 bytes) This provides enough random information to ensure that every SPA packet fwknop generates is unique—at least, the packets are unique to the degree of randomness that the Perl function rand() is able to conjure with each invocation. (For Perl versions 5.004 and later, the srand() function is called implicitly at the first utilization of the rand() function.)

Username This is the name of the user that is executing the fwknop command, as returned by getlogin()—or getpwuid() if getlogin() fails. The fwknop server uses this username to determine whether the remote user is authorized to gain access to a service or run a command. (Note that by the time the fwknop server sees the username, the SPA packet has been successfully decrypted, which implies that the SPA packet has been authenticated and the process of verifying authorization can begin.)

Timestamp This is the timestamp on the local system. The fwknop server uses this value to determine whether the SPA packet falls within the timed access window defined by the MAX_SPA_PACKET_AGE variable.

Software version This is the version of the fwknop client:

```
[mbr@spaclient ~]$  fwknop --Version
[+] fwknop v1.8.1 (file revision: 694)
      by Michael Rash <mbr@cipherdyne.org>
```

For example, the software version field in this case would contain the value 1.0. The fwknop server uses this information to maintain backward compatibility with older clients if the SPA packet format changes.

Mode This tells the fwknop server whether or not the SPA client wishes to run a command. The default value is 1 for access mode; command mode is denoted by 0.

Access directive This string tells the fwknop server which type of traffic the client wishes to have accepted by the iptables firewall when the policy is modified. The fwknop server parses this string for ports and protocols to instruct iptables to accept, and the policy is reconfigured accordingly. For example, if the client wishes to access both TCP port 22 and UDP port 1194 (which is used by OpenVPN), the string would be `*client IP*,tcp/22,udp/1194`. The fwknop server controls whether or not users can request to open specific ports. If only certain ports are allowed to be opened, they must be defined within the access.conf file. (For more information, see "OPEN_PORTS" and "PERMIT_CLIENT_PORTS" on page 238.)

Command string This string is a full command that the fwknop client would like to execute on the server; for example, `/etc/init.d/apache2 restart` or `w |mail -s "w output" *you@domain*.com`. This feature can open the fwknop server to a security risk if it is not used wisely, and it is disabled by default. (For more information, see "ENABLE_CMD_EXEC" and "CMD_REGEX" on page 238.)

Packet MD5 sum This MD5 sum is calculated by the fwknop client and is included within the SPA packet for an added degree of confidence that the packet has not been altered while en route over the network. Normally, the encryption algorithm itself provides adequate security, because decrypting altered ciphertext does not normally result in valid plaintext; however, including the MD5 sum allows the fwknop server to independently agree that the data the client received is what the server actually receives.

Server authentication method The fwknop 0.9.6 release added this field to the packet format to allow the fwknop server to require an additional authentication parameter in the SPA packet. For example, the server may require the remote fwknop client to enter the local user's `crypt()` password. In this case, the authentication method string would be something like `crypt,*password*`.

Before SPA packets are encrypted and sent, by default, over UDP port 62201, the fields discussed above are Base64-encoded and then concatenated with colons. This encoding ensures that the colon delimiters remain unique, even across fields that may have contained colons before the encoding. When you combine all these fields without Base64 encoding, you get something like this:

```
9562145998506823:mbr:1161142204:1.0:1:0.0.0.0,tcp/22:koEtBtDLOze22sNRyfASoA
```

Once you Base64-encode the individual fields, you get this:

```
9562145998506823:bWJy:1161142204:1.0:1:MC4wLjAuMCxOY3AvMjI=:koEtBtDLOze22sNRyfASoA
```

Finally, the packet data is encrypted either with the Rijndael symmetric cipher or an asymmetric cipher supported by GnuPG (the Elgamal asymmetric cipher is used by GnuPG by default). If you encrypt with Rijndael, this is the result:

```
U2FsdGVkX18O3i3n8BfSpgM6wCaf8zC4CgLsSlf2STIQTNWxaC9Q3IP1NSW91nSj5zr8Juz7YyX1oFzMu2FDZgbYAJUOxre
e7WyzHJdYl3ympcEPxpd/Qx5Wo3D8uS/AD8WyaV232srRCNWcsPUc9Q
```

Every SPA packet is encrypted and decrypted with either a symmetric-key cipher or an asymmetric-key cipher. A *symmetric-key cipher* is an algorithm that encrypts and decrypts data using the same key (hence the *symmetric* designation). The Rijndael cipher, which has been selected as the Advanced Encryption Standard (AES), is an important example of a symmetric-key cipher. An *asymmetric-key cipher*, on the other hand, is an algorithm that encrypts and decrypts data with a pair of keys: the public key, which is published publicly, and the private key, which is kept secret. The two keys are related via a mathematical conundrum, but they are not identical (hence the *asymmetric* designation).

Deploying fwknop

Now that you have a good understanding of the configuration options available in fwknop, it's time for a few meaty operational examples. In each case, the fwknop client is used to gain access to SSHD through a default-drop iptables policy after reconfiguration by the fwknop server. The network diagram in Figure 13-1 should help you to visualize these scenarios.

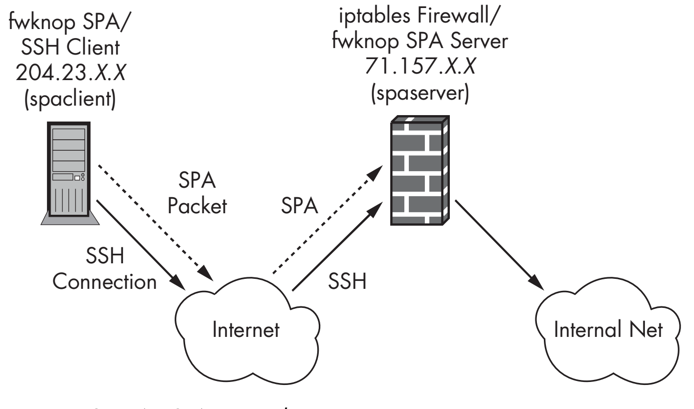

Figure 13-1: An SPA network

In each scenario below, the fwknop client is executed on the system labeled spaclient, and the SPA packet is sent to the system labeled spaserver. The dotted line in Figure 13-1 represents the SPA packet, and the follow-on

SSH connection can only take place after the SPA packet has communicated the desired access to the spaserver system and iptables can be reconfigured to allow the access.

SPA via Symmetric Encryption

The fwknop client has a rich set of command-line options that allow you to tell the fwknop server the exact access that you would like the iptables policy to grant. If you use these command-line options, you must include the access or command string, a source IP address resolution method, and the fwknop server target IP address.

You can assume that the local iptables policy drops all packets in the fwknop server's `INPUT` chain that are destined for TCP port 22. Start by configuring the fwknop.conf file with `AUTH_MODE` set to `PCAP`, make sure `PCAP_INTF` is set to `eth0`, and set the access.conf file to the following. (Note that there are no GnuPG directives, such as `GPG_REMOTE_ID` or `GPG_DECRYPT_PW`, included in this example.)

```
[root@spaserver ~]# cat /etc/fwknop/access.conf
SOURCE: ANY;
OPEN_PORTS: tcp/22;
REQUIRE_USERNAME: mbr;
KEY: myencryptkey;
FW_ACCESS_TIMEOUT: 30;
```

Use the commands below to ❶ start the fwknop server and ❷ verify that it is running. By examining syslog messages, you'll see that fwknopd is ready to accept SPA packets from ❸ one `SOURCE` block (which is derived from within the access.conf file listed above), and that ❹ an existing disk cache of SPA packet MD5 sums is imported. Finally, make sure that ❺ SSHD is running on the local system.

```
❶ [root@spaserver ~]# /etc/init.d/fwknop start
  Starting fwknop ... [ ok ]
❷ [root@spaserver ~]# /etc/init.d/sshd status
   * status:  started
  [root@spaserver ~]# tail /var/log/messages
  Oct 17 23:59:53 spaserver fwknopd: starting fwknopd
  Oct 17 23:59:53 spaserver fwknopd: flushing existing Netfilter IPT_AUTO_CHAIN
  chains
❸ Oct 17 23:59:53 spaserver fwknopd: imported access directives (1 SOURCE
  definitions)
❹ Oct 17 23:59:53 spaserver fwknopd: imported previous md5 sums from disk cache:
  /var/log/fwknop/md5sums
❺ [root@spaserver ~]# /etc/init.d/sshd status
   * status:  started
```

With the fwknop server up and running, you can test to see if SSHD is accessible from the fwknop client system, and then use fwknop to gain access to it. The `-A tcp/22` command-line argument at ❶ tells the fwknop server that the client wishes to access TCP port 22; the `-R` argument at ❷ instructs the fwknop client to automatically resolve the externally routable address from which the SPA packet will originate (this is accomplished by querying http://www.whatismyip.com); and the -k argument at ❸ tells the fwknop client to send the SPA packet to the spaserver host.

```
[mbr@spaclient ~]$ nc -v spaserver 22
[mbr@spaclient ~]$ fwknop ❶-A tcp/22 ❷-R ❸-k spaserver
[+] Starting fwknop in client mode.
[+] Resolving hostname: spaserver
    Resolving external IP via: http://www.whatismyip.com/
    Got external address: 204.23.X.X

[+] 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: 2282553423001461
        Username:    mbr
        Timestamp:   1161146338
        Version:     1.0
        Action:      1 (access mode)
        Access:      204.23.X.X,tcp/22
        MD5 sum:     wvWqr/qKuZdZ+xaqPO1KwA

[+] Sending 150 byte message to 71.157.X.X over udp/62201...
[mbr@spaclient ~]$ ssh spaserver
Password:
[mbr@spaserver ~]$
```

The last line in the listing above shows that you are now logged into the spaserver host, verifying your access to SSHD. Below, the messages written to syslog on the fwknop server tell you ❶ that fwknopd has successfully received and decrypted the SPA packet sent by the fwknop client, and ❷ that an ACCEPT rule has been added to allow TCP port 22 connections for the 204.23.*X.X* IP address for 30 seconds. The ACCEPT rule is removed in ❸. (Although not displayed here, emails are also sent to the addresses defined by the EMAIL_ADDRESSES variable in fwknop.conf to inform you when fwknop grants and removes access to an SPA client.)

```
❶ Oct 18 00:38:58 spaserver fwknopd: received valid Rijndael encrypted packet
  from: 204.23.X.X, remote user: mbr
❷ Oct 18 00:38:58 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
  204.23.X.X -> tcp/22 (30 seconds)
❸ Oct 18 00:39:29 spaserver knoptm: removed iptables FWKNOP_INPUT ACCEPT rule
  for 204.23.X.X -> tcp/22, 30 second timeout exceeded
```

The fwknop server adds and deletes all SPA access rules within the custom chain FWKNOP_INPUT instead of within any of the built-in chains, such as INPUT or FORWARD. This strictly separates rules in an existing iptables policy from the rules it manipulates, which means that you don't have to worry about fwknop rules conflicting with any existing rules in your iptables policy. You can execute the following command on the fwknop server before the 30-second timer has expired to see the iptables rule that grants access to SSHD.

```
[root@spaserver ~]# fwknopd --fw-list
[+] Listing chains from IPT_AUTO_CHAIN keywords...

Chain FWKNOP_INPUT (1 references)
 pkts  bytes  target  prot  opt  in  out  source      destination
 11    812    ACCEPT  tcp   --   *   *    204.23.X.X  0.0.0.0/0   tcp dpt:22
```

In this example, the fwknop server has reconfigured iptables to allow access to SSHD for 30 seconds; then fwknopd will delete the `ACCEPT` rule from the `FWKNOP_INPUT` chain. Although most SSH connections last longer than 30 seconds, this isn't a serious limitation as long as the Netfilter connection tracking facilities are used, allowing the established TCP connection to remain open between the client and the server:

```
[root@spaserver ~]# iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

SPA via Asymmetric Encryption

The problem of key exchange is a central one in the field of cryptography and the novel solution provided by public key cryptosystems distinguishes itself. In contrast to symmetric ciphers where the key must be shared between two parties in the clear over an insecure channel,[5] asymmetric ciphers rely on a system whereby people actively publish the public portion of a public/private key pair. For example, when person A encrypts data with person B's public key, person B, and only person B, can decrypt the ciphertext by combining the public and private key via an operation that breaks the lock on the data. This lock is built from a mathematical puzzle that is computationally expensive to solve without access to both the public and private keys.[6]

GnuPG Key Exchange for fwknop

In order to use GnuPG keys within fwknop, you must create and import the server's public key into the client's key ring, and vice versa. Because the decryption password for the client's key is never stored in a file, it is safe to use any GnuPG key with the fwknop client. However, for this discussion, I'll generate new client and server keys and import them as follows (some of the output has been removed for brevity).

[5] Transmitting keys over an insecure medium is an abstract notion that includes things like writing the shared key down on a piece of paper and mailing it between the parties.

[6] The puzzle is usually derived from a classic computational problem such as integer factorization of products of two large prime numbers, or computing discrete logarithms over a cyclic group. The latter method is used by the Elgamal cryptosystem in GnuPG; see http://en.wikipedia.org/wiki/ElGamal_encryption for a brief overview.

```
[mbr@spaclient ~]$  gpg --gen-key
gpg (GnuPG) 1.4.5; Copyright (C) 2006 Free Software Foundation, Inc.

Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire

Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Michael Rash
Email address: mbr@cipherdyne.org
Comment: Linux Firewalls fwknop_client key
You selected this USER-ID:
    "Michael Rash (Linux Firewalls fwknop_client key) <mbr@cipherdyne.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a passphrase to protect your secret key.
Enter passphrase:

[mbr@spaclient ~]$ gpg --list-keys "fwknop_client"
pub   1024D/AB743C36 2007-10-18
uid                  Michael Rash (Linux Firewalls fwknop_client key)
<mbr@cipherdyne.org>
sub   2048g/1035BC5C 2007-10-18
```

The length of ciphertext data associated with an SPA message that is encrypted with a 4,096-bit Elgamal key is usually well over the 1,500-byte MTU of Ethernet networks, so a key length of 2,048 bits is chosen (shown in bold above).

Now we export the client public key to a file:

```
[mbr@spaclient ~]$ gpg -a --export-key "fwknop_client" > fwknop_client.asc
```

A similar process is performed on the fwknop server with the key generation and exporting commands duplicated on the server side:

```
[root@spaserver ~]# gpg --gen-key
[root@spaserver ~]# gpg --list-keys "fwknop_server"
pub   1024D/25801B3A 2007-10-18
```

```
uid                  Michael Rash (Linux Firewalls fwknop_server key)
<mbr@cipherdyne.org>
sub   2048g/39E2FDC6 2007-10-18
[root@spaserver ~]# gpg -a --export "fwknop_server" > fwknop_server.asc
```

Finally, you need to transfer the public keys to each respective system, import them, and sign them. The import step is required so that the server's public key is available on the client's GnuPG key ring, and vice versa. The signing step is necessary for fwknop to verify the identity of signed SPA packet data. Even though I'll transfer the public keys over scp, given the nature of public-key cryptosystems, I could have published the keys on a web page for all to see without any negative security impact. It is also important to note that SSHD may not always be accessible (in fact, it will intentionally be firewalled off by the fwknop setup), so other transfer mechanisms for the public keys may sometimes be required. Here's some abbreviated command output (the scp transfers are in ❶ and ❷, and the import and signing commands begin in ❸ and ❹).

```
❶ [mbr@spaclient ~]$ scp fwknop_client.asc root@spaserver:
  Password:
❷ [mbr@spaclient ~]$ scp root@spaserver:fwknop_server.asc .
  Password:
❸ [mbr@spaclient ~]$ gpg --import fwknop_server.asc
  gpg: key 25801B3A: public key "Michael Rash (Linux Firewalls fwknop server key)
  <mbr@cipherdyne.org>" imported
  gpg: Total number processed: 1
  gpg:               imported: 1
  [mbr@spaclient ~]$ gpg --default-key "fwknop_client" --sign-key "fwknop_server"
  [mbr@spaclient ~]$ ssh -l root spaserver
  Password:
❹ [root@spaserver ~]# gpg --import fwknop_client.asc
  gpg: key AB743C36: public key "Michael Rash (Linux Firewalls fwknop client key)
  <mbr@cipherdyne.org>" imported
  gpg: Total number processed: 1
  gpg:               imported: 1
  [root@spaserver ~]# gpg --default-key "fwknop_server" --sign-key "fwknop_client"
```

Running fwknop with GnuPG Keys

With the GnuPG keys imported and signed within both the fwknop client's and the server's key rings, it is time to see fwknop in action with GnuPG. To begin, the access.conf file on the fwknop server must contain the proper GnuPG access definitions. The `SOURCE` block begins in ❶ and instructs fwknopd to require that SPA packets are encrypted with the fwknop_server key and signed with the fwknop_client key. In addition, iptables must be deployed to shut down access to SSHD, as shown in ❷, and fwknop must be running, as shown in ❸.

```
  [root@spaserver ~]# cat /etc/fwknop/access.conf
❶ SOURCE: ANY;
  OPEN_PORTS: tcp/22;
  REQUIRE_USERNAME: mbr;
  GPG_HOME_DIR: /root/.gnupg;
```

```
GPG_DECRYPT_ID: fwknop_server;
GPG_DECRYPT_PW: GPGdecryptpw;
GPG_REMOTE_ID: fwknop_client;
FW_ACCESS_TIMEOUT: 30;
❷ [root@spaserver ~]# iptables -I INPUT 1 -p tcp --dport 22 -j DROP
[root@spaserver ~]# iptables -I INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
❸ [root@spaserver ~]# /etc/init.d/fwknop start
Starting fwknop ... [ ok ]
```

Now, from the spaclient system, you can use Netcat to check that SSHD is indeed unreachable, and use fwknop to gain access through iptables. Below, the last line indicates that you have successfully logged into the spaserver system.

```
[mbr@spaclient ~]$ nc -v spaserver 22
[mbr@spaclient ~]$ fwknop -A tcp/22 -gpg-recip "fwknop_server" --gpg-sign
"fwknop_client" -R -k spaserver
[mbr@spaclient ~]$ ssh -l root spaserver
Password:
[root@spaserver ~]#
```

As was the case when fwknop was instructed to use the Rijndael symmetric cipher, the fwknop server writes several messages to syslog. This time, however, there is new information indicating that the GnuPG-encrypted SPA message was signed by ❶ the required key ID (defined by the `GPG_REMOTE_ID` variable in access.conf). As usual, an iptables `ACCEPT` rule is ❷ added and ❸ deleted after 30 seconds.

```
Oct 18 15:48:07 spaserver fwknopd: received valid GnuPG encrypted packet
(signed with required key ID: ❶"fwknop_client") from: 204.23.X.X, remote
user: mbr
❷ Oct 18 15:48:07 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
204.23.X.X -> tcp/22 (30 seconds)
❸ Oct 18 15:48:08 spaserver knoptm: removed iptables FWKNOP_INPUT ACCEPT rule
for 204.23.X.X -> tcp/22, 30 second timeout exceeded
```

Detecting and Stopping a Replay Attack

Until now, you have seen fwknop put to legitimate uses in an effort to reduce the attack surface of SSHD. When an SPA packet travels over an untrusted network, anyone who can watch the packet on the wire can save it, analyze it, and replay it. I have mentioned that the fwknop SPA implementation is well-suited to thwarting replay attacks by comparing MD5 sums of incoming SPA messages, but here's a concrete example.

In Figure 13-2, an attacker is placed within the Internet cloud and monitors an SPA packet in transit from the spaclient system to the spaserver system. The attacker uses tcpdump to capture the SPA packet to a file (spa.pcap) and examines it enough to see that the packet is encrypted gibberish. Then the attacker replays the packet back over the network with tcpreplay, which is depicted by the dotted line labeled *Replayed SPA Packet* in Figure 13-2.

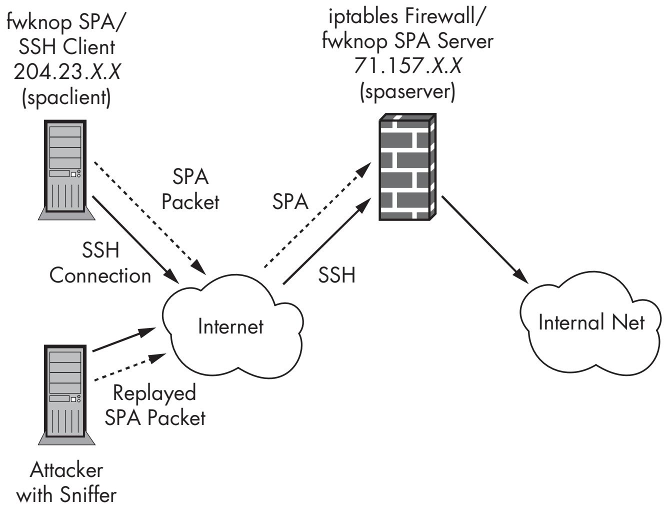

Figure 13-2: An attacker monitors and replays an SPA packet

The command sequence to accomplish the SPA packet replay appears below. First, the spaclient system sends a valid SPA packet to the spaserver system at ❶. The fwknop `-L` command-line argument allows fwknop to recall the last command-line options that were used against the fwknop server host. This is handy for simplifying the relatively complex fwknop command-line interface. As the SPA packet is en route over the network, the attacker ❷ captures the packet with tcpdump, and ❸ finds that it appears to be unintelligible. The attacker hence deduces that this packet may be an SPA packet (particularly since the packet is captured on the default port UDP 62201 that fwknop uses to communicate). Another tip-off that the packet may be part of an SPA scheme is that SSHD is not accessible from the attacker's IP address, but an SSH session may be established between the spaclient and spaserver. The attacker then ❹ replays the SPA packet on the network against the spaserver system in an effort to connect to the SSH server. The fwknop daemon running on spaserver has detected the replayed SPA packet as indicated by the syslog message in ❺, and the iptables policy does not grant the attacker any access. Although not displayed here, fwknop also sends an email alert to highlight the fact that a previous SPA packet was replayed, since this is not something that should happen under any reasonable circumstances.

```
❶ [mbr@spaclient ~]$ fwknop -L spaserver
  [+] Running with last command-line args: -A tcp/22 --gpg-recip fwknop_server
  --gpg-sign fwknop_client -R -k spaserver
  [+] Starting fwknop in client mode.
  [+] Resolving hostname: spaserver
      Resolving external IP via: http://www.whatismyip.com/
      Got external address: 204.23.X.X

  [+] Enter the GnuPG password for signing key: fwknop_client
  GnuPG signing password:
```

```
  [+] Building encrypted Single Packet Authorization (SPA) message...
  [+] Packet fields:

          Random data: 2018495891979939
          Username:    mbr
          Timestamp:   1161229378
          Version:     1.0
          Action:      1 (access mode)
          Access:      204.23.X.X,tcp/22
          MD5 sum:     1P53i1YNdwou/xA+361T3w

  [+] Sending 1010 byte message to 71.157.X.X over udp/62201...
❷ [root@attacker ~]# tcpdump -i eth0 -l -nn -s 0 udp port 62201 -w spa.pcap
❸ [root@attacker ~]#  tcpdump -l -nn -X -r spa.pcap | head
  reading from file spa.pcap, link-type EN10MB (Ethernet)
  23:31:43.883144 IP 204.23.X.X.42245 > 71.157.X.X.62201: UDP, length 1010
       0x0000:  4500 040e e5ff 4000 0000 0000 0000 0000  E.....@.@.......
       0x0010:  0000 0000 a505 f2f9 03fa 1d59 6851 494f  ...-.......YhQIO
       0x0020:  4177 7668 5165 7735 3476 3347 4541 662f  AwvhQew54v3GEAf/
       0x0030:  5754 6335 4279 736b 5544 5a76 5830 6873  WTc5ByskUDZvX0hs
       0x0040:  6b59 5047 7774 6664 7349 5774 4948 3548  kYPGwtfdsIWtIH5H
       0x0050:  5658 4c49 4731 656a 562b 3639 7057 6866  VXLIG1ejV+69pWhf
       0x0060:  4474 7443 7541 626b 4941 474c 3665 4c33  DttCuAbkIAGL6eL3
       0x0070:  426f 3632 5757 4231 3867 7975 7141 5a72  Bo62WWB18gyuqAZr
       0x0080:  2f71 687a 3234 614e 7042 596a 4a2f 524d  /qhz24aNpBYjJ/RM
❹ [root@attacker ~]# tcpreplay -i eth0 spa.pcap
  sending on: eth0
   1 packets (1052 bytes) sent in 0.15 seconds
   6831169.0 bytes/sec 52.12 megabits/sec 6493 packets/sec
  [root@attacker ~]# ssh -l root 71.157.X.X
  [root@spaserver ~]# tail /var/log/messages
❺ Oct 18 23:32:50 spaserver fwknopd: attempted message replay from: 204.23.X.X
```

Spoofing the SPA Packet Source Address

The SPA protocol supports spoofed source IP addresses. This is a consequence of two factors: the ability of the fwknop server to acquire the real source address from within the SPA packet payload, and the fact that SPA packets are sent over UDP with no expectation of return traffic.

fwknop uses the `Perl Net::RawIP` module to send SPA packets via a raw socket, which allows you to set the source IP address to an arbitrary value from the fwknop client command line. (This requires root access.) In Figure 13-3, the spaclient system sends the SPA packet, but the source IP address in the IP header is crafted to make the packet appear to originate from the 207.132.*X.X* IP address. When fwknopd is running on the spaserver system, it sniffs the SPA packet off the wire, but it grants access to SSHD from the real fwknop client IP address 204.23.*X.X* instead of from the spoofed source IP address, 207.132.*X.X*.

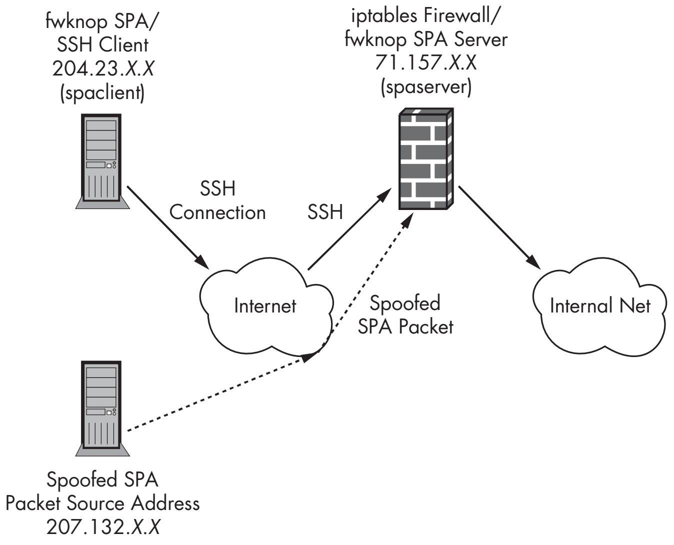

Figure 13-3: An SPA packet from a spoofed source address

Notice that the fwknop client command shown below has become more complicated. This is to support spoofing the source IP address of the SPA packet (as root), but to also build the encrypted payload using the fwknop_client key, which is owned by the `mbr` user and located within the /home/mbr/ .gnupg directory.

```
[root@spaclient ~]#  fwknop --Spoof-src 207.132.X.X -A tcp/22 --gpg-home-dir
/home/mbr/.gnupg --Spoof-user mbr --gpg-recip "fwknop_server" --gpg-sign
"fwknop_client" --quiet -R -k spaserver
GnuPG signing password:
```

The syslog messages below indicate that the fwknop server sniffed the SPA packet, that it originates from ❶ the spoofed source address 207.132.*X.X*, and that access is granted to the IP address contained within ❷ the encrypted packet, 204.23.*X.X*.

```
[root@spaserver ~]# tail /var/log/messages
 Oct 18 23:31:37 spaserver fwknopd: received valid GnuPG encrypted packet
(signed with required key ID: "fwknop_client") from: ❶207.132.X.X, remote
user: mbr
 Oct 18 23:31:37 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
❷204.23.X.X -> tcp/22 (30 seconds)
```

fwknop OpenSSH Integration Patch

The fwknop project hopes to make the use of SPA as easy and user friendly as possible. One thing that can help reduce the burden on the user is to integrate seamlessly with a variety of client applications. Because the most common application of SPA is to protect SSH communications, fwknop provides a patch against the OpenSSH source code, which integrates the ability to execute the fwknop client directly from the OpenSSH client command line. For this to work, you must first apply the patch to the OpenSSH source code and recompile it. The following illustrates how to accomplish this for the OpenSSH-4.3p2 release, assuming the source code is located in /usr/local/src.

```
$ cd /usr/local/src/openssh-4.3p2
$ wget http://www.cipherdyne.org/LinuxFirewalls/ch13/openssh-4.3p2_SPA.patch
$ patch -p1 < openssh-4.3p2_SPA.patch
patching file config.h.in
patching file configure
patching file configure.ac
patching file ssh.c
$ ./configure --prefix --with-spa-mode && make
$ su -
Password:
# cd /usr/local/src/openssh-4.3p2
# make install
```

The most important thing to note about the commands above is that the `--with-spa-mode` argument to the configure script ensures that the SPA patch code is included within OpenSSH when it is compiled.

Now, with the modified SSH client installed, the fwknop client can be invoked directly from the SSH command line, eliminating the need to run fwknop manually before using SSH to make a connection. The patch adds the new command-line argument `-K` *fwknop args* to SSH; this argument can be used as follows to gain access to the spaserver system without separately running the fwknop client.

```
[mbr@spaclient ~]$ ssh -K "--gpg-recip ABCD1234 --gpg-sign DEFG5678 -A tcp/22
-R -k spaserver" mbr@spaserver
GnuPG signing password:
Password:
Last login: Wed Oct 17 15:48:19 2007 from spaclient
[mbr@spaserver ~]$
```

Familiar log messages on the fwknop server side indicate receipt of the SPA packet and confirm that the packet checks out (i.e., it was encrypted with a required key ID and not replayed on the network).

```
Oct 17 15:53:39 spaserver fwknopd: received valid GnuPG encrypted packet
(signed with required key ID: A742839F) from: 204.23.X.X, remote user: mbr
Oct 17 15:53:39 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
204.23.X.X -> tcp/22 (30 seconds)
```

The new SSH `-K` option passes its arguments down to the fwknop command line, so all functionality provided by fwknop is exposed to the SSH command line. This includes the `-L` *host* argument, which, as mentioned earlier in this chapter, allows a previously used fwknop command line to be leveraged against the same host. Therefore, the following command would work.

```
ssh -K "-L host" user@host
```

SPA over Tor

The Onion Router (Tor), is an anonymizing network composed of a globally dispersed set of nodes called onion routers (see http://tor.eff.org). The Tor network is designed to harden TCP-based services against a type of Internet surveillance called traffic analysis. *Traffic analysis* is used to determine who is talking to whom over the Internet, and it is easily deployed by any organization—particularly ISPs—with access to Internet traffic. Even encrypted application traffic is subject to traffic analysis because IP addresses are transmitted in the clear.

NOTE *I am not considering IPSEC or other VPN protocols here, but even these protocols can reveal information through traffic analysis as well.*

The information that can be gleaned simply from watching two parties communicate is often underestimated, and this has implications for everything from keeping passwords secure to revealing the identities of supposedly anonymous remailers.

Tor works by setting up a separate virtual circuit through the router cloud for each TCP connection. A virtual circuit is established between an *entry router* and a randomly selected *exit router*. Every circuit is unique, and each hop within the circuit only knows the hop from which traffic originates and the hop to which traffic must be sent. Further, traffic is encrypted when it is within the router cloud.

The end result is that a client may communicate with a server over the open Internet via this virtual circuit, and any third party that can monitor the traffic going into or coming out of the router cloud will see IP addresses talking to each other that seem totally unrelated.[7]

Is there a benefit to sending SPA packets over the Tor network? Decidedly so, as it extends the service-cloaking nature of fwknop, making it more difficult to determine that an SPA is being used at server locations.

But there is one catch: Tor uses TCP for transport. This implies that Tor is incompatible with SPA, because SPA packets are transferred over UDP by default. Even though fwknop supports sending SPA packets over blind TCP ACK packets,[8] this alone is not enough to get an SPA packet to traverse the Tor network. A virtual circuit is created through Tor only after the initial TCP connection with the entry router has been fully established, implying that bidirectional communication is required.

fwknop solves this problem by breaking the single packet nature of SPA and sending SPA packets over fully established TCP connections with the fwknop_serv daemon. This daemon spawns a minimal TCP server that runs as user `nobody`, does a `bind()` and `listen()` on TCP port 62201, and then loops over successive calls to `accept()`. With each `accept()`, a single `recv()` is made so

[7] There have been some attacks against Tor in order to reduce the strength of its resistance to traffic analysis; see http://www.cl.cam.ac.uk/users/sjm217/papers/oakland05torta.pdf.

[8] A blind TCP ACK (or other TCP packet with other flags set) is not part of an established TCP connection.

that only a single TCP segment may be sent across by a client before the session is shut down. This allows a client to send the SPA payload, but nothing else, across the established TCP connection. Then, by using the socat program, which functions as the `socks4` proxy that Tor requires, together with the `--TCP-sock` argument on the fwknop command line, the SPA packet can be sent over the Tor network.

NOTE *For more information on socat, see http://www.dest-unreach.org/socat.*

Concluding Thoughts

This chapter and Chapter 12 have illustrated powerful techniques in computer security, showing how a server can be protected by a default-drop packet filter, through which access is granted only to clients able to prove their identities to a passively monitoring device. Port knocking was the first technology to implement this idea, but due to some serious limitations in the port-knocking architecture (including the difficulty of adequately addressing the replay problem and the inability to transmit more than a few tens of bytes), SPA has proved itself a more robust technology. The notion of an authorizing Ethernet sniffer combined with a default-drop packet filter is a relatively new one in the computer security field, but it seems that new implementations are springing up every day.[9]

Based on iptables, fwknop is an open source implementation of SPA that provides a flexible mechanism for managing multiple users within the SPA paradigm.

[9] There is even a project to put HMAC-based SPA directly into iptables; see http://svn.berlios.de/svnroot/repos/portknocko, and a discussion thread in the Netfilter development list archives, http://lists.netfilter.org/pipermail/netfilter-devel/2006-October/thread.html.