Improving Sendmail Security by Turning It Off · Hal Pomeranz · 8

Port Knocking — Network Authentication Across Closed Ports
Martin Krzywinski · 12

Freeware Forensics Tools
Kristy Westphal · 18

Lppbuild — AIX Package Creation
Charles Ritter · 23

CFS — The Cryptographic File System · Keith McEufee · 27

Questions and Answers · Amy Rich · 29

Integrating SpamAssassin into Qmail and Sendmail · Jason Camp · 35

Solaris™ Resource Management — The Fair Share Scheduler
Peter Baer Galvin · 39

Using the xargs Command
Ed Schaefer · 44

Protecting Your UNIX Systems — An Overview · Eric Cole · 47

Introduction to RAID · Henry Newman · 51

Stress Testing Jabber with the Jabber Test Suite · Dustin Puryear · 53
Port Knocking — Network Authentication Across Closed Ports

Martin Krzywinski

Two thousand years ago, information security was already a concern to Julius Caesar, who is said to have been one of the first people to use cryptography to secure his dispatches. To encrypt his messages, he applied a circular shift to each letter moving it three places further in the alphabet. In the day when few were literate, the Caesar cipher was probably sufficient. In fact, evidence exists to indicate that cryptography was used by Hebrew scribes as early as 600-500 BC in the form of a reverse-alphabet cipher named ATBASH. Long before the invention of the first computer network, information security was a preoccupation for those responsible for safeguarding data. Today, with the exponential growth of information systems and of the Internet population, effectively protecting information requires increasingly sophisticated methods. How do we keep our data protected from unauthorized access while keeping it accessible online to ourselves and to those we trust?

Keeping network applications updated and having a conservatively configured firewall can serve as the first line of defense. Equally important are intrusion detection systems (IDS), which are designed to spot known, derivative, or anticipated attacks. While no connected system can be rendered safe, short of severing the network cable (sometimes the right solution), a properly configured firewall/IDS combination is an effective bulwark. However, packet-filtering firewalls (e.g., the Linux kernel-based IPCHAINS/iptables) filter on the basis of IP address and ports and do not contain an authentication mechanism to discriminate users. As a result, applications behind open ports remain vulnerable to attacks by malevolent individuals from unfiltered IP addresses.

Port knocking is a firewall-based user authentication system that uses closed ports for authentication, making it possible to connect to a networked computer that has no open ports. Communication across closed ports is mediated through the firewall log, in which all connection attempts are recorded. The log is monitored for specific port sequences that encode information used to modify firewall rules, which are changed to open or close ports for a specific IP address. The system can be extended to implement the transfer of arbitrary information across closed ports. Port knocking is inherently stealthy because it is not possible to determine whether a host is listening to port knocks and because the authentication information flows in the form of connection attempts.

Authentication Mechanism

The application of the port knocking mechanism is illustrated in this section. A number of examples are used to show how knock sequences can be used to encode information.

In the simplest case, a small number of ports are used to encode short knock sequences that map to specific modifications of the firewall rules. For example, a host with IP address FIREWALL may have all privileged ports closed, while ports 100-109 are monitored for connection attempts. The IPCHAINS command to achieve this is:

```
ipchains -A input -p tcp -s 0/0 -d FIREWALL/32 100:109 -j DENY -i
ipchains -A input -p tcp -s 0/0 -d FIREWALL/32 1:1024 -j DENY
```

Any connections to ports 100-109 will be silently ignored, but logged. The client will not receive ICMP error packets and therefore cannot determine whether an authentication process is taking place. Now suppose a client from IP address IPCONTACT makes a series of connection attempts to FIREWALL’s ports 102,100,100,103, in that order. The firewall log will contain entries like:

```
Feb 12 00:13:26 ... Input DENY eth1 PROTO=6 IPCONTACT:64137 FIREWALL:102 ...
Feb 12 00:13:27 ... Input DENY eth1 PROTO=6 IPCONTACT:64138 FIREWALL:100 ...
Feb 12 00:13:27 ... Input DENY eth1 PROTO=6 IPCONTACT:64139 FIREWALL:100 ...
Feb 12 00:13:28 ... Input DENY eth1 PROTO=6 IPCONTACT:64140 FIREWALL:103 ...
```

Using a service daemon that monitors the firewall log, the port sequence 102,100,100,103 can be extracted and associated with the incoming IP address. This specific port sequence could act as a trigger to manipulate the firewall rules. For example, it could set off the following rule:

```
ipchains -I input -p tcp -s IPCONTACT/32 -d FIREWALL/32 22 -j ACCEPT
```

which would have the effect of opening port ssh/22 to TCP traffic from IPCONTACT. The client that performed the knock sequence would now be able to connect to the ssh application and carry out a session. After the session ends, another knock sequence would be sent to close the port.

The manner in which information is encoded in the port sequence is arbitrary. In the example above, a discrete mapping between a sequence and an action was illustrated. The sequence could also contain the port and the length of time it is held open. For example, the sequence could be:
The header and footer indicate to the monitoring daemon that a sequence is about to start and end. Four numbers in the sequence payload encode the port to be opened. A time value indicates whether to (1) open the port, (2) open the port and then close it after some time has passed, or (3) close the port. In this example, 105 could mean open port abcd and keep it open for 5^10=50 minutes. A final checksum field validates the authenticity of the sequence.

A more sophisticated sequence would be encrypted and would contain the IP address for which ports are opened. This type of knock sequence provides a two-fold advantage. First, anyone intercepting the sequence cannot use it directly unless they manage to decrypt it, replace the client IP with their own, and then re-encrypt it. Second, ports can be opened for a given IP address by using a different host, since the IP address of the client performing the knock is no longer used. This type of knock sequence could encrypt the following data, stored in seven unsigned chars, with the range of the port limited to 0-255 for simplicity:

```
IPbyte1 IPbyte2 IPbyte3 IPbyte4 port time checksum
```

For example, if port 22 were to be opened to the client from IP address 142,103,205.1 for 15 minutes, the decrypted sequence would be:

```
142 103 205 1 22 15 233
```

where the checksum is (142+103+205+1+22+15) mod 255 = 233. Using Perl, the encryption can be carried out using the cipher block chaining module Crypt::CBC and, to choose one algorithm, Crypt::Blowfish. In this example, the cipher object is created without using an initialization vector (IV). The IV is random data used to seed the algorithm to ensure that each encryption of the same data results in a different cipher text. Removing the IV reduces the length of the cipher text, and therefore of the encrypted knock sequence. In practice, an IV would be used, but in this example it is skipped:

```c
use constant KEY => "knock";
use constant CIPHER => "Blowfish";
$cipher = Crypt::CBC->new(key=>KEY,cipher=>CIPHER,
iv=>"01234567",prepend_iv => 0);
```

The knock data is stored as an array, with the checksum as the last element. The sum() function from Math::VecStat is used to calculate the checksum:

```perl
@data = (142,103,205,1,22,15,233);
push(@data, $data[6]); @data[6] = 255; @data = (142, 103, 205, 1, 22, 15, 233);
```

The data is packed into a string, encrypted, then encoded into a list of unsigned chars:

```perl
$cipher_text = $cipher->encrypt(pack("C*",@data));
@cipher_text = unpack("C*", $cipher_text); @221, 169, 50, 219, 86, 117, 66, 197
```

Finally, the unsigned chars are mapped onto actual port values. For example, ports 745-1000 could be used:

```perl
use constant PORTMIN => 745;
@knocks = map { PORTMIN+$_ } @cipher_text; @966, 914, 795, 964, 831, 992, 1000
```

The result is an encrypted 8-port knock sequence that contains the IP address, port, time flag, and checksum. Once the knock sequence is received by the daemon monitoring the log, it is decrypted. Malformed sequences can be detected because the decryption step does not produce the expected data structure.

**Benefits of Port Knocking**

The primary feature of port knocking is that it allows for stealthy authentication into a networked host that has no open ports. The method is stealthy because it is not possible to directly determine whether the host is listening for port knocks. While port probes could be employed to try to guess which ports are being monitored and the format of the knock sequence, success is highly improbable and such brute-force attempts can be easily discovered with IDS tools.

Since information is flowing in the form of connection attempts, rather than typical packet data payload, it is unlikely that this authentication method would be detected by packet sniffing methods. To minimize the risk of a functional knock sequence being reconstructed by a third party, the sequence should contain the client IP address and be encrypted.

The system is flexible because the authentication scheme is built into the knock sequence. Existing applications, such as...
Disadvantages of Port Knocking

As with any security system, the disadvantages present themselves as inconveniences, preferably small, which must be endured while the system is in use. These inconveniences should ideally pose less of a burden than the risk associated with not using the system. Port knocking requires a client script to perform the knock. This client and any associated data should be considered a secret and kept on removable media, such as a USB key. The use of the client imposes an overhead for each connection and users would require instruction on its appropriate use.

For obvious reasons, port knocking cannot be used to protect ports of public applications, such as Web or mail services. It is not reasonable to expect that each mail or web client will perform a port knock. This would require everyone to know the secret knock, or how to construct one. If port knocking is

---

Listing 1  portknocking client

```c
use strict;
use Math::Trig qw(sin);
use Crypt::CBC;
use IO::Socket::INET;
use Getopt::Long;
use vars qw($port $remoteport $opt_port $opt_time);
GetOptions("client=s","remote=s","port=t","time=t");

$port ||= "client"; $remoteport ||= "remote"; $port ||= "port";
$opt_port ||= "opt_port"; $opt_time ||= "opt_time";

sub portknock { 
    my $sock = &net($remoteport);
    my $data = &pack("C",$data);
    $sock->send($data);
    &timeout($sock); 
    &shutdown($sock);
} 

# portknock
sub net { 
    my $endpoint = &net($remoteport);
    my $sock = &net($remoteport);
    &timeout($sock); 
    &shutdown($sock);
} 

# timeout
sub timeout { 
    my $sock = &net($remoteport);
    &shutdown($sock);
} 

# shutdown
sub shutdown { 
    my $sock = &net($remoteport);
    &shutdown($sock);
} 

# pack
sub pack { 
    my $data = &net($remoteport);
    &shutdown($sock);
} 
```

---

Listing 2  portknocking server

```c
use strict;
use File::Tail;
use Crypt::CBC;
use Schedule::AT::;
use Math::Trig qw(sin);
use POSIX qw(strftime);
use Getopt::Long;
use constant PORTMIN => 745;
use constant KNOCKLENGTH => 8;
use constant CIPHER => "Blowfish";
use vars qw($port_file);

GetOptions("file=s");

die "Cannot open file line file (file "FILE_NAME") if i = $port_file || ! -r

my $file = File::Tail->new(name=>"/opt_file","maxinterval=>2);
my $QOVE;
while(defined($line=$file->read) or
      # no data or only GEN packets
      # next unless line = "GEN"
      ($line = "GEN")
      ||
      # if line = "PROTOCOL:id,ip1:id,ip2:id" then
      ($line = "PROTOCOL:id,ip1:id,ip2:id")
      ||
      # if line = "IP:SPORT=[1-128]
      ($line = "IP:SPORT=[1-128]"
      ||
      # if line = "IP:PORT=512"
      ($line = "IP:PORT=512"
      ||
      # if line = "IP:PORTMIN=255"
      ($line = "IP:PORTMIN=255"
      ||
      # print "Knock from ip to port sport in"
      print "Knock from ip to port sport in"
      ||
      # push this port to each queue item
      $QOVE[EQUE][1] = [$] if $QOVE[EQUE][1];
      push($QOVE[EQUE][1],[],Sport); 
      print "current queue "=";foreach $QOVE[EQUE][1],"in";
      # if the queue is of the length and process it
      if(@$QOVE[EQUE][1] = KNOCKLENGTH)
      { 
        ProcessQueue($QOVE[EQUE][1]);
        print "current queue @QOVE[EQUE][1]s";
      } 
      sub ProcessQueue { 
        my $queue = shift;
```
used on a host to protect some ports (e.g., ssh, telnet, ftp) while others remain open for the public (e.g., Web, mail), two options remain. Public services should preferably be delegated to a bastion host in the DMZ and isolated from the host on which sensitive information is stored. If the delegation is not possible, all public services must be regularly updated or patched to minimize vulnerability. Though some applications remain open to the public, the protection of others with port knocking would limit the number of possible types of attack.

The implementation of any system that manipulates firewall rules in an automated fashion must be robust to prevent legitimate users from being locked out. If knocks cannot be interpreted properly or if the service daemon crashes, the networked host may become isolated. Appropriate safeguards should be implemented to avoid such a scenario.

Implementation
Both the client and server scripts are available at:
http://mkweb.bcgsc.ca/portkno kelling

knockclient
A prototype client is shown in Listing 1. Lincoln Stein’s Crypt: CBC module is used as proxy to Crypt::Blowfish to carry out encryption. The unencrypted knock sequence comprises seven values: four IP bytes, a port (limited to the range 0-255 in this implementation), a time flag, and a checksum (mod 255). The time flag determines how the daemon will react: 0 to open the port, 255 to close the port, and any other value in the range 1-254 to open the port and then close it after that many minutes. The knock on the firewall (IP=FIREDWALL) to open port ssh/22 on IP=CLIENT and have the port closed after 15 minutes would be executed by calling the client as follows:
knockclient -c CLIENT -r FIREDWALL -p 22 -t 15
The client packs the list of seven integers, performs the encryption, and unpacks the string into unsigned chars (0-255). These values are then mapped onto a sequence of ports in the range 745-1000.

knockdemon
The accompanying knock daemon is shown in Listing 2. This script will work with any firewall compatible with IPCHAINS syntax. The File::Tail module is used to continually look for new lines in the firewall log. Lines corresponding to connection attempts to ports 745-1000 are parsed for the remote IP and port number. Because the client produces encrypted sequences of 8 digits, a queue of this length is used to store the ports. Each connecting IP has its own queue. Every time the queue size reaches 8 an attempt is made to decrypt it. If the decryption is successful, and the checksum is correct, appropriate action is taken and the queue is cleared. If the decryption fails, the oldest queue port element is removed.

The firewall rules are manipulated by a system call to the IPCHAINS binary, although the IPChains Perl module by Jonathan Schatz can also be used. The “at” queue system provides a simple way to trigger closing a port after a specified amount of time. The interface to this is provided by Jose Rodrigues’ Schedule::At module.

Conclusion
Port knocking is a stealthy network authentication system that uses closed ports to carry out identification of trusted users. This method provides the means of establishing a connection to an application running on a completely isolated system on which no ports are initially open. The use of port knocking to safeguard information makes it possible to dramatically increase the level of protection without sacrificing accessibility.

Resources
Port Knocking Page:
http://mkweb.bcgsc.ca/portkno kelling
Randomness Recommendations for Security (RFC 1750):
Codes and Ciphers: Julius Caesar, the Enigma and the Internet, Robert Churchhouse. Cambridge University Press, 2002.
Registered Ports List (RFC 3232):
http://www.iana.org
IPCHAINS/IFTABLES
http://www.metfilter.org

Marvin Krzywinski is a bioinformatics research scientist at the Genome Sciences Centre in Vancouver, British Columbia, Canada. He spends his time applying Perl to problems in physical mapping and data processing automation. In his spare time he can be found encouraging his cat to stick to her diet. He can be reached at: martink@bcgsc.ca.

June 2003