Port knocking with knockd

By **Greg Sabino Mullane**
November 16, 2009

One of the best ways to secure your box against SSH attacks is the use of port knocking. Basically, port knocking seals off your SSH port, usually with firewall rules, such that nobody can even tell if you are running SSH until the proper "knock" is given, at which time the SSH port appears again to a specific IP address.

Popular Tags

In most cases, a “knock” simply means accessing specific ports in a specific order within a given time frame.

Let’s step back a moment and see why this solution is needed. Before SSH there was telnet, which was a great idea way back at the start of the Internet when hosts trusted each other. However, it was (and is) extremely insecure, as it entails sending usernames and passwords “in the clear” over the internet. SSH, or Secure Shell, is like telnet on steroids. With a mean bodyguard. There are two common ways to log in to a system using SSH. The first way is with a password. You enter the username, then the password. Nice and simple, and similar to telnet, except that the information is not sent in the clear. The second common way to connect with SSH is by using public key authentication. This is what I use 99% of the time. It’s very secure, and very convenient. You put the public copy of your PGP key on the server, and then use your local private SSH key to authenticate. Since you can cache your private key, this means only having to type in your SSH password once, and then you can ssh to many different systems with no password needed.

So, back to port knocking. It turns out that any system connected to the internet is basically going to come under attack. One common target is SSH—specifically, people connecting to the SSH port, then trying combinations of usernames and passwords in the hopes that one of them is right. The best prevention against these attacks is to have a good password. Because public key authentication is so easy, and makes typing in the actual account password such a rare event, you can make the password something very secure, such as:

perl 108
open-source 101
hosting 93
tips 78

All Tags

Archive

Posts by date
Posts by author

Search our blog

```
gtsmef#3ZdbVdAebAS@9e[AS4fed';8fS14S0A8d!!9~d1aAQ5.81sa0'ed
```

However, this won't stop others from trying usernames and passwords anyway, which fills up your logs with their attempts and is generally annoying. Thus, the need to "hide" the SSH port, which by default is 22. One thing some people do is move SSH to a "non-standard" port, where non-standard means anything but 22. Typically, some random number that won't conflict with anything else. This will reduce and/or stop all the break-in attempts, but at a high cost: all clients connecting have to know to use that port. With the ssh client, it's adding a -p argument, or setting a "Port" line in the relevant section of your .ssh/config file.

All of which brings us to port knocking. What if we could run SSH on port 22, but not answer to just anyone, but only to people who knew the secret code? That's what port knocking allows us to do. There are many variants on port knocking and many programs that implement it. My favorite is "knockd", mostly because it's simple to learn and use, and is available in some distros' packaging systems. My port knocking discussion and examples will focus on knockd, unless stated otherwise.

knockd is a daemon that listens for incoming requests to your box, and reacts when a certain combination is reached. Once knockd is installed and running, you modify your firewall rules (e.g. iptables) to drop all incoming traffic to port 22. To the outside world, it's exactly as if you are not running SSH at all. No break-in attempts are possible, and your security logs stay nice and boring. When you want

to connect to the box via SSH, you first send a series of knocks to the box. If the proper combination is received, knockd will open a hole in the firewall for your IP on port 22. From this point forward, you can SSH in as normal. The new firewall entry can get removed right away, cleared out at some time period later, or you can define another knock sequence to remove the firewall listing and close the hole again.

What exactly is the knock? It's a series of connections to TCP or UDP ports. I prefer choosing a few random TCP ports, so that I can simply use telnet calls to connect to the ports. Keep in mind that when you do connect, it will appear as if nothing happened—you cannot tell that knockd is logging your attempt, and possibly acting on it.

Here's a sample knockd configuration file:

```
[options]
  logfile = /var/log/knockd.log

[openSSH]
  sequence    = 32144,21312,21120
  seq_timeout = 15
  command     = /sbin/iptables -I INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
  tcpflags    = syn

[closeSSH]
  sequence    = 32144,21312,21121
  seq_timeout = 15
  command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
  tcpflags    = syn
```

In the above file, we've stated that any host that sends a TCP syn flag to ports 32144, 21312, and 21120, in that order, within 15 seconds, will cause the iptables command to be run. Note that the use of iptables is completely not hard-coded to knockd at all. Any command at all can be run when the port sequence is triggered, which allows for all sorts of fancy tricks.To close it up, we do the same sequence, except the final port is 21221.

Once knockd is installed, and the configuration file is put in place, start it up and begin testing. Leave a separate SSH connection open to the box while you are testing! If you are really paranoid, you might want to open a second SSH daemon on a second port as well. First, check that the port knocking works by triggering the port combinations. knockd comes with a command-line utility for doing so, but I usually just use telnet like so:

```
[greg@home ~] telnet example.com 32144
Trying 123.456.789.000...
telnet: connect to address 123.456.789.000: Connection refused
[greg@home ~] telnet example.com 21312
Trying 123.456.789.000...
telnet: connect to address 123.456.789.000: Connection refused
[greg@home ~] telnet example.com 21120
Trying 123.456.789.000...
telnet: connect to address 123.456.789.000: Connection refused
```

Note that we reveived a bunch of "Connection refused"—the same message as if we tried any other random port. Also the same message that people trying to connect to a port knock protected SSH will see. If you look in the logs for knockd

(set as /var/log/knockd.log in the example file above), you'll see some lines like this if all went well:

```
[2009-11-09 14:01] 100.200.300.400: openSSH: Stage 1
[2009-11-09 14:01] 100.200.300.400: openSSH: Stage 2
[2009-11-09 14:01] 100.200.300.400: openSSH: Stage 3
[2009-11-09 14:01] 100.200.300.400: openSSH: OPEN SESAME
[2009-11-09 14:01] openSSH: running command: /sbin/iptables -I INPUT -s 100.200
```

Voila! Your iptables should now contain a new line:

```
$ iptables -L -n | grep 100.200
ACCEPT     tcp  --  100.200.300.400  anywhere            tcp dpt:ssh
```

The next step is to lock everyone else out from the SSH port. Add a new rule to the firewall, but make sure it goes to the bottom:

```
$ iptables -A INPUT -p tcp --dport ssh -j DROP
$ iptables -L | grep DROP
DROP       tcp  --  anywhere             anywhere            tcp dpt:ssh
```

You'll note that we used "A" to append the DROP to the bottom of the INPUT chain, and "I" to insert the exceptions to the top of the INPUT chain. At this point, you should try a new SSH connection and make sure you can still connect.If all is working, the final step is to make sure the knockd daemon starts up on boot, and that the DROP rule is added on boot as well. You can also add some hard-coded

exceptions for boxes you know are secure, if you don't want to have to port knock from them every time.

One flaw in the above scheme the sharp reader may have spotted is that although the SSH port cannot be reached without a knock, the sequence of knocks used can easily be intercepted and played back. While this doesn't gain the potential bad guy too much, there is a way to overcome it. The knockd program allows the port knocking combinations to be stored inside of a file, and read from, one line at a time. Each successful knock will move the required knocks to the next line, so that even knowing someone else's knock sequence will not help, as it changes each time. To implement this, just replace the 'sequence' line as seen in the above configuration file with a line like this:

```
one_time_sequences = /etc/knockd.sequences.txt
```

In this case, the sequences will be read from the file named "/etc/knockd.sequences.txt". See the manpage for knockd for more details on one_time_sequences as well as other features not discussed here. For more on port knocking in general, visit **portknocking.org**.

While the one_time_sequences is a great idea, I'd like to see something a little different implemented someday. Specifically, having to pre-populate a fixed list of sequences is a drag. Not only do you have to make sure they are random, and that you have enough, but you have to keep the list with you locally. Lose that list, and

you cannot get in! A better way would be to have your port knocking program generate the new port sequences on the fly. It would also encrypt the new port sequences to one or more public keys, and then put the file somewhere web accessible. Thus, one could simply grab the file from the server, decrypt it, and perform the port knocking based on the list of ports inside of it. Is all of that overkill for SSH? Almost certainly. :) However, there are many other uses for port knocking that simple SSH blocking and unblocking. Remember that many pieces of information can be used against your server, including what services are running on which ports, and which versions are in use.

hosting networking security

Comments

Visit the **GitHub Issue** to comment on this post.

Haruchai commented on November 18, 2009

This is worth a read:
http://www.cipherdyne.org/fwknop/

It argues for Single Packet Authorisation over knock sequences.

Jon Jensen commented on November 18, 2009

Nice overview of port knocking, Greg. Aren't there port-knocking clients out there too, so you don't have to manually telnet to the ports in the right order?

I like to be careful about how I disallow access to a particular TCP port to the world. Simply dropping all packets makes a "hole" in the responses of your TCP ports which a port scanner can easily see.

Normally a TCP port without any listening service will send a TCP RST response, saying there's nothing there. Dropping the packet entirely says clearly "I'm blocking this with a firewall so it may be really interesting to investigate further".

For example, you could do something like:

-A INPUT -p tcp -j REJECT --reject-with tcp-reset
-A INPUT -p udp -j REJECT
-A INPUT -j DROP

Using a default -j REJECT on a TCP port also results in abnormal behavior, sending an ICMP port unreachable reply instead of a TCP RST, hence separate rules for each protocol type.

Checking with nmap or similar, that looks a lot more like a server not specifically blocking port 22, but just not running sshd there.

Greg Sabino Mullane commented on November 18, 2009

Haruchai, thanks for the link. The only thing I don't like about implementations such as that one is the need for a custom client to do the authentication. Whereas with generic port knocking you can use ping, telnet, and other standard tools likely to be on any box you are on. Looks like fwknop is a neat idea though.

Greg Sabino Mullane commented on November 18, 2009

Jon, yes, knockd comes with a simple "knocker" client, but I prefer telnet as I don't have to worry about if the machine I happen to be on has knockd installed or not. Plus, I can make a bash alias, so it's not much typing at all.

Interesting example about the response, but overall I'm not too worried about revealing much extra information that way. These days, a server not running *something* on port 22 is already "interesting", no? :)

Jon Jensen commented on November 18, 2009

Ok, Greg, then consider it an argument in favor of consistency. Do you really want to get a hanging connection attempt using ssh to the machine before you portknock, but get a normal immediate disconnect when going to other

closed ports? It just feel lame to me to DROP certain ports but leave others at the default reset behavior.

Greg Sabino Mullane commented on November 18, 2009

Jon:

I do love consistency in my servers (/me waves to puppetd). Point taken, I'll adapt your changes.

Jon Jensen commented on November 18, 2009

Heh, ok.

I should note that half the reason I wrote about this here is I expect some reader to point out how the iptables rules I gave still make it possible to detect iptables' involvement, compared to a bare TCP stack. Come help me, internets! How can this be improved?

Ezekiel commented on January 20, 2010

http://www.thoughtcrime.org/software/knockknock/

"knockknock is a simple, secure, and stealthy port knocking implementation that does not use libpcap or bind to a socket interface."

Very Interesting.

Subscribe to our newsletter

email address

SUBSCRIBE

SOLUTIONS

- Enterprise Security
- Ecommerce
- Custom Applications
- Managed Infrastructure

TECHNOLOGY

- Hardware
- Cloud
- Database
- Backend
- Frontend

ABOUT

- Clients
- Our Team
- Contact Us

BLOG

LIQUID GALAXY