How To Use Port Knocking to Hide your SSH Daemon from Attackers on Ubuntu

Posted January 8, 2014 211.9k SECURITY UBUNTU

By Justin Ellingwood
Become an author

Status: Deprecated

This article covers a version of Ubuntu that is no longer supported. If you are currently operate a server running Ubuntu 12.04, we highly recommend upgrading or migrating to a supported version of Ubuntu:

- Upgrade to Ubuntu 14.04.
- Upgrade from Ubuntu 14.04 to Ubuntu 16.04
- Migrate the server data to a supported version

Reason: Ubuntu 12.04 reached end of life (EOL) on April 28, 2017 and no longer receives security patches or updates. This guide is no longer maintained.

See Instead:
This guide might still be useful as a reference, but may not work on other Ubuntu releases. If available, we strongly recommend using a guide written for the version of Ubuntu you are using. You can use the search functionality at the top of the page to find a more recent version.

Introduction

Servers, by definition, are implemented as a means of providing services and making applications and resources accessible to users. However, any computer connected to the internet is inevitably targeted by malicious users and scripts hoping to take

advantage of security vulnerabilities.

Firewalls exist and should be used to block access on ports not being utilized by a service, but there is still the question of what to do about services that you want access to, but do not want to expose to everybody. You want access when you need it, but want it blocked off otherwise.

Port knocking is one method of obscuring the services that you have running on your machine. It allows your firewall to protect your services until you ask for a port to be opened through a specific sequence of network traffic.

In this guide, we will discuss how to implement port knocking as a method of obscuring your SSH daemon on an Ubuntu 12.04 VPS using the `knockd` package.

Note: *This tutorial covers IPv4 security. In Linux, IPv6 security is maintained separately from IPv4. For example, "iptables" only maintains firewall rules for IPv4 addresses but it has an IPv6 counterpart called "ip6tables", which can be used to maintain firewall rules for IPv6 network addresses.*

If your VPS is configured for IPv6, please remember to secure both your IPv4 and IPv6 network interfaces with the appropriate tools. For more information about IPv6 tools, refer to this guide: How To Configure Tools to Use IPv6 on a Linux VPS

How Does Port Knocking Work?

Port knocking works by configuring a service to watch firewall logs or packet capture interfaces for connection attempts. If a specific sequence of predefined connection attempts (or "knocks") are made, the service will modify the firewall rules to open up connections on a certain port.

This allows you to keep your services hidden until you actually plan on using them. This would not be practical for something like an HTTP server because you would want connections available at all time. But it would be useful for services meant to be used only by known, legitimate users, like SSH.

Although a knocking sequence can be arbitrarily complex, it is not, in and of itself, usually the only set of security measures. Usually the service's own security and authentication methods are then exposed to a user who issues the correct sequence. In this way, port knocking adds an additional layer that a user must go through to even get to the regular authentication.

Even more helpful is that there is no feedback given on knocking attempts. An intruder scanning would see all of the usual ports closed and if they attempted a knocking sequence, would have to check between each attempt to see if a port was opened. This is often enough to dissuade or prohibit attackers.

For our purposes, we will be using the iptables firewall that comes with Ubuntu 12.04, and installing a daemon called `knockd` to provide the port knocking functionality.

Configure IPTables to Block Most Traffic

Before we get to the actual port knocking, we need to configure a basic firewall. We want to lock down most things.

By default, Ubuntu comes with iptables installed. However, there are no default rules in place, so all traffic is allowed. To design your own set of rules, you can learn how to set up a firewall with iptables here.

For our purposes, we will use most of the rules from that guide.

Begin by allowing traffic on the local machine. This means accepting traffic that the server generates and sends to itself. This allows services to talk to each other without being blocked:

```
sudo iptables -A INPUT -i lo -j ACCEPT
```

This appends a rule to the "INPUT" chain. This chain handles all connections coming into the server. This rule tells iptables to accept all traffic on the "lo" network interface, which is the local loopback interface used for internal communication.

Next, we want to make sure we allow all established connections and traffic related to established connections by typing:

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

This rule tells iptables to accept traffic that is associated with an already established connection. This is an important one, because once we start blocking connections, we don't want our current SSH session to be cut off.

Next, you'll want to allow persistent, world-consumable services. What I mean by this is, add rules for services that need to be always running and visible. For instance, if you have a website being served on the standard port 80, you want to allow that traffic all the time.

Do not add rules to iptables for the services that we will use the port knocker to open. We will instead use the knocking daemon to dynamically modify our rule set. For our tutorial, we will not add our SSH server in our initial iptables configuration.

Use this syntax to establish your own rules:

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

At this point, we've only added rules to accept connections, not drop them. We're still accepting everything, we've just been explicit about certain kinds of traffic.

Now, we will drop everything we haven't specifically allowed. Add this rule:

```
sudo iptables -A INPUT -j DROP
```

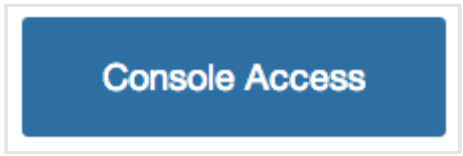
Any traffic that hasn't been handled by the above rules will be dropped. You can view your rules by typing:

```
sudo iptables -S
```

```
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -j DROP
```

As you can see, we still do not have any rules to accept new SSH connections.

If your connection is dropped at this point, you'll have to access your server through the control panel by clicking the "console access" button in the upper right corner:



This acts as a direct login and does not use SSH, so it won't be affected by your rules.

Once you have established your iptables rules, make them persistent with `iptables-persistent`. Install it by typing:

```
sudo apt-get install iptables-persistent
```

Afterwards, start the service by typing:

```
sudo service iptables-persistent start
```

Install the Knockd service

The port knocking aware service that we will be using is called `knockd`. We can install it by simply typing:

```
sudo apt-get install knockd
```

This will install the utility, but will not start the service by default.

This is a safety precaution in order to prevent the daemon from immediately blocking important traffic. You must configure and explicitly enable this service.

Configure Knockd to Use Port Knocking

To configure the service, we will have to edit the configuration file. Open this file with root privileges:

```
sudo nano /etc/knockd.conf
```

You should see a file that looks like this:

```
[options]
        UseSyslog

[openSSH]
        sequence    = 7000,8000,9000
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn

[closeSSH]
        sequence    = 9000,8000,7000
        seq_timeout = 5
        command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn
```

Immediately, you should be able to see some important information about how knockd works. You should also start to realize that the configuration is not too complex.

In the "options" section, we see a directive called `UseSyslog`. This tells knockd that it should log its information using the normal syslog methods. This will insert logs into `/var/log/messages`.

If you would like to specify a different log file, you can do so by using this option instead:

```
LogFile = /path/to/log/file
```

Underneath, we have two sections. The names for these sections can be anything. They are used to group a set of rules that will match a single event each.

For instance, in our file, we have a section that will open up our SSH port, and one that will close it again.

The parameter that sets the knocking pattern is here:

```
sequence    = 7000,8000,9000
```

This means that this set of rules will match if the same IP requests a connection on port 7000, followed directly by port 8000, followed finally by port 9000.

Two other parameters in this set also control whether the activity matches:

```
seq_timeout = 5
tcpflags = syn
```

The first option specifies an amount of time that the sequence must be completed in.

The second specifies a flag that must be present in the tcp packets in order for them to be considered valid. The value of `syn` that we see here is commonly used to distinguish the packets we want from those created in the background by programs like SSH.

Finally, we see command:

```
command = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
```

You should recognize this as an iptables rule. As the section label "openSSH" points out, this section will open up a port for SSH connections when the correct sequence is hit.

However, if you were paying attention during the iptables configuration, you'll see that this new rule uses the `-A` option to *append* this rule to the end of the INPUT chain. This will place this rule after the rule to drop all remaining connections.

To fix this situation, we need to modify this command. Replace the command with a rule to *insert* the new rule at the top of the list. We do this by using the `-I` option and referencing the location as rule 1:

```
command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
```

With this change, a new rule will be added to the top of the INPUT chain to accept SSH connections from the user who knocked. The `%IP%` portion of the rule will be replaced by the IP address that made the acceptable knock.

The second SSH section does almost the same thing, but it uses a different sequence, and removes the rule from iptables that opened connections to SSH. We can hit this sequence to close the gap we opened.

In practice, you should always change the sequences for these two sections to something that is essentially random. Keeping the default sequence is effectively removing any security that port knocking establishes.

Before we configure anything further, let's test out our current set up. Save and close the file.

Implement the Knockd Service

Now that we have configured knockd to have a valid rule set, we can test it out by implementing our daemon. Keep in mind that, although the configuration is valid, at this point, it is not secure unless you changed the port sequences for each knocking section.

We need to enable the service by editing another file. Open this file with root privileges:

```
sudo nano /etc/default/knockd
```

We need to change the `START_KNOCKD` option to be "1" in order to start the service:

```
START_KNOCKD=1
```

Save and close the file.

Now, we can start the service by typing:

```
sudo service knockd start
```

This will start the daemon and allow you to change the iptables rule sets by knocking on the sequences of ports.

Port Knocking Test

We should now test our ability to modify the iptables rules by using the port knocking sequence that we configured.

In a **new** terminal window, we can use tools to request these ports. It is best to keep your other session open in case there is a problem. Again, if you accidentally lock yourself out, use the "console access" button in the upper right corner of the droplet's page in the control panel.

We can use a variety of different tools to knock. Some popular choices are `netcat`, `nmap`, and a specially designed client called, appropriately, `knock`.

We will use `nmap` in this example, because it is installed by default on most Linux distributions and OS X.

Before we knock, let's confirm that our SSH port is, in fact, closed currently. Type the command you usually use to connect to the server:

```
ssh root@server_ip_address
```

```
sh: connect to host server_ip_address port 22: Operation timed out
```

You should receive no response from the server and the SSH client should timeout. This is because our SSH daemon is currently blocked by iptables. Type ctrl-C to end the SSH attempt if it does not time out automatically

Because of the sequence timeout parameter that is set, we actually have a very limited amount of time to hit the correct sequence. We will use a small, in-line bash script to knock on these ports quickly.

From your local machine, type a command like this:

```
for x in 7000 8000 9000; do nmap -Pn --host_timeout 201 --max-retries 0 -p $x server_ip_address; done
```

In the command, adjust the three numbers to the numbers you selected for your sequence to open the SSH port. Change the server*ip*address to reflect the address of your server.

This will call nmap sequentially on all of the ports you listed.

Once that is complete, you should be able to log into SSH regularly:

```
ssh root@server_ip_address
```

We can re-close the port by knocking on the other sequence that we configured:

```
for x in 9000 8000 7000; do nmap -Pn --host_timeout 201 --max-retries 0 -p $x server_ip_address; done
```

Port Knocking with the Knock Utility

A simpler way to knock is to use the `knock` utility that is provided by the makers of the `knockd`. This is included in the knockd package, so you can install it on our client machine just as we did with the server:

```
sudo apt-get install knockd
```

You can also get knock clients from the project's website under the "download" section. There are native OS X and Windows clients available (and even iOS and Android clients).

Once you have the knock client installed, you can perform a sequence easily by just using this syntax:

```
knock server_ip_address sequence
```

So for our example, you could open your SSH port by typing:

```
knock server_ip_address 7000 8000 9000
```

This is much quicker than the other method mentioned.

We can close the port by typing:

```
knock server_ip_address 9000 8000 7000
```

Configuring Knockd to Close Connections Automatically

Now that we have established that our port knocking daemon is functioning correctly, let's change some configuration details to be more robust.

Open the configuration file again:

```
sudo nano /etc/knockd.conf
```

We will take advantage of knockd's ability to establish a command timeout in order to condense our SSH match into one rule. This means that we won't have to remember to knock to close the SSH port after we are finished.

We can comment out or delete the "openSSH" and "closeSSH" sections. We will be replacing them with a single section that we'll simply call "SSH":

```
[options]
    UseSyslog

[SSH]
```

Under this new section, we will establish a sequence, the tcpflags, and the sequence timeout just like we did in the other sections. We will also include the command we used to open the SSH port:

```
[options]
    UseSyslog

[SSH]
```

```
    sequence = 5438,3428,3280,4479
    tcpflags = syn
    seq_timeout = 15
    start_command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
```

Choose a unique sequence of ports. As you can see, in this example we use four ports. We can increase the number of ports as long as they can all be knocked on in the time frame specified in the `seq_timeout` parameter.

The `start_command` parameter is the same as the `command` parameter used in the other example. We chose to use this variant simply to be more verbose about what we are doing.

After this, we will add some new parameters that will help us close the port:

```
[options]
    UseSyslog

[SSH]
    sequence = 5438,3428,3280,4479
    tcpflags = syn
    seq_timeout = 15
    start_command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
    cmd_timeout = 10
    stop_command = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
```

The `cmd_timeout` is the number of seconds that knockd will wait before executing the command contained in the `stop_command` variable.

The result is that when the correct sequence is used, the daemon will open the SSH port. It will then wait 10 seconds and then close the port again.

Save and close the file.

Implement your new rule by restarting the daemon:

```
sudo service knockd restart
```

We can use this port knocking rule to connect easily within the time specified. For instance, we could use this command to easily connect to our server:

```
knock server_ip_address 5438 3428 3280 4479 && ssh root@server_ip_address
```

The hole that we create in our firewall will close after us in 10 seconds.

Conclusion

Although port knocking is sometimes talked about in a disparaging tone as security through obscurity (hiding a service instead of actually securing it), it is a great way to add an additional layer of protection against random attacks.

You should always secure your services using the native tools available for the best results. However, adding something like a port knocking scheme in front of these methods can drastically cut back on the number of brute force attacks or intrusion attempts that your services experience.

By Justin Ellingwood

Tutorial Series

How To Implement Port Knocking to Obscure your SSH Daemon

Port knocking is a security concept that involves dynamically altering firewall rules to expose access to an otherwise protected service. This is done by sending a pre-configured special packet, or a pattern of packets that the port knocking software is listening for. In this series, we will discuss a variety of ways to configure port knocking to add an extra layer of security around your SSH daemon.

Show Tutorials

Related Tutorials

How To Protect Your Server Against the Meltdown and Spectre Vulnerabilities

How To Protect Your Linux Server Against the GHOST Vulnerability

How to Protect Your Server Against the Shellshock Bash Vulnerability

How to Protect Your Server Against the Heartbleed OpenSSL Vulnerability

How to Install TrueCrypt (CLI) on Linux

22 Comments

Leave a comment...

Log In to Comment

Sovietaced *January 17, 2014*

0

Awesome tutorial!

rick144256 *January 17, 2014*

2

Knock, Knock, Knock, who's there?, SSH

mkoistinen *January 21, 2014*

0

Great tutorial. Also, don't be dissuaded by anyone who claims this is an example of "security through obscurity". It is not. This is more akin to another password layer and absolutely provides an excellent layer of protection from zero-day exploits, should there ever be any, on your sshd server.

chaubner *January 21, 2014*

0

Great tutorial!!! Honestly never even thought of doing this....now enabled and part of my server setup security protocols :)

FMCB *March 7, 2014*

0

this is tut is good, but I got into weird issue...
The cpu jumps high because of this process "knock"... LOL my server got knocked out

FMCB *March 7, 2014*

0

by the way, does this knock process conflict with any iptables rule..?

sounds like that is what causing this issue...

any idea?

mnipp *July 25, 2014*

0 Using a virtual hosted ubuntu 14.04 LTS, knockd failed to start.
'ip a l' helped solve the issue.

```
/# ip a l

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: venet0: <BROADCAST,POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/void
    inet 127.0.0.2/32 scope host venet0
    inet 103.*.*.*/32 brd 103.*.*.* scope global venet0:0


/# vi /etc/default/knockd

START_KNOCKD=1

  # command line options

 # KNOCKD_OPTS="-i eth1"
KNOCKD_OPTS="-i venet0:0"
```

I first tried KNOCKD_OPTS="-i venet0" this let the knockd service start, but did not see the knocks.

cramhead *November 11, 2014*

0

I really like your writeup.

The configuration as you state it works well for me until I implement the automatically timing out connection.
When I restart the service it fails to restart. I've tried a number permutations, but unfortunately no luck.
When I check the docs at http://www.zeroflux.org/projects/knock/ they are quite different, i.e. key names have underscores, and capitals.

arist0v *November 20, 2014*

0

Hello, when i'm trying to use knock IP PORT PORT PORT && ssh root@IP, according to my syslog the knock stop on stage 1 or 2 9 time on 10 so i have to use the commande multiple time to be able to ssh. I have followed your (awesome thank you) howto but don't understand what append!

thank you

> pavelpetrov *April 15, 2015*
>
> 0
>
> Interestingly enough I stumbled upon the same problem as you and found no info online.
> The problem seem to be in the knock. For some reason i had to use separate commands instead of one.
>
> ```
> knock -v example.com 7000
> knock -v example.com 8000
> ```

```
knock -v example.com 9000
```

Instead of

```
knock -v example.com 7000 8000 9000
```

RuzsinszkyAttila *March 18, 2016*

0 I found this problem, too. :-(

My solution is:
knock <IP> port1 && sleep 1 && knock <IP> port2 && sleep 1 && knock <IP> portN && ssh -i my_key.prv username@IP

So I think the knock is tooo fast! :)

electronicsguy *February 27, 2015*

0 Very well written Justin, thank you!

KahaliTao *June 17, 2015*

0 I have a potential alternative to putting iptables command into knockd.
It would keep the tables a little cleaner and would take care of 'timeouts' by itself.
I'm talking about an 'ipset' .. iptables commonly supports ipsets as is.

You would put a rule in iptables like this:
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m set --match-set portknock src -j ACCEPT --comment "allow port knocked connections"
This rule will allow ANY ip address in the 'portknock' set to accept new ssh connections.
To add an IP to an ip set, you need to have ipset installed and you simply run: "ipset add portknock %IP% timeout 30"
When you first create the set you can define a default timeout for the address or you can define one yourself in the ipset command when you add an IP address. This will prevent messing with your iptables, and when using the timout, the ipset will automatically decline new ssh connections if the IP address has expired in the list.

> KahaliTao *June 17, 2015*
> 0
> I should mention, that if you are using ANY method that automatically closes the port you should set a keepalive option in your ssh client. Once you establish an ssh connection, the port can close and you can remain connected due to the rules above in the tutorial. (Main the Related,Established rule)
> If your client times out , you will need to knock again to reconnect.

bubakazouba *June 19, 2015*
0
amazing tutorial!!
Thank you

Arakiz *November 29, 2015*
0
Made a bash script to generate knockd.conf and an open port script and a close script. It needs 3 inputs:

1. How many pins / ports you want

2. Hostname of the server you want to lock down
3. User you will be using for connecting `Syntax: gen-knock.sh <pins> <hostname> <username>`

Output is then:

knockd.conf
<hostname>-OPEN.sh
<hostname>-CLOSE.sh

```
#!/bin/bash

PINS=$1
HOSTNAME=$2
USERNAME=$3

PINDIPS=`shuf -i 1024-65534 -n $PINS`
OUTLOCK="knockd.conf"
OPENKEYFILE="$HOSTNAME-OPEN.sh"
CLOSEKEYFILE="$HOSTNAME-CLOSE.sh"

# CHECK SYNTAX
if [ "$#" -ne 3 ]; then
    echo "Syntax Error!"
    echo "Syntax: $0 <number of pins> <hostname/ip> <username to login>"
    exit
fi

# CREATE FILES
touch $OUTLOCK
```

```
touch $OPENKEYFILE
touch $CLOSEKEYFILE

# GENERATE SERVER CONFIGURATION
echo "GENERATING KNOCKd CONFIG FILE"
echo "[option]" > $OUTLOCK
echo "   UseSyslog" >> $OUTLOCK
echo "" >> $OUTLOCK
echo "[openSSH]" >> $OUTLOCK
for x in `seq 1 $PINS`
do
    PIN=$(echo $PINDIPS | cut -d " " -f $x)
    SEQUENCE="$SEQUENCE,$PIN"
done
SEQUENCE=$(echo $SEQUENCE | sed -r 's/^.{1}//')
echo "   sequence = $SEQUENCE" >> $OUTLOCK
echo "   seq_timeout = 5" >> $OUTLOCK
echo "   command = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT" >> $OUTLOCK
echo "   tcpflags = syn" >> $OUTLOCK
echo "" >> $OUTLOCK
echo "[closeSSH]" >> $OUTLOCK
for x in `seq $PINS -1 1`
do
        PIN=$(echo $PINDIPS | cut -d " " -f $x)
        CLOSING="$CLOSING,$PIN"
done
CLOSING=$(echo $CLOSING | sed -r 's/^.{1}//')
echo "   sequence = $CLOSING" >> $OUTLOCK
echo "   seq_timeout = 5" >> $OUTLOCK
echo "   command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT" >> $OUTLOCK
```

```
echo "   tcpflags    = syn" >> $OUTLOCK
echo ""
echo ""

# GENERATE OPENKEY-$HOSTNAME.sh
echo "GENERATING OPEN KEY SCRIPT"
echo "#!/bin/bash" > $OPENKEYFILE
echo "" >> $OPENKEYFILE
for x in `seq 1 $PINS`
do
    OPENPIN=$(echo $PINDIPS | cut -d " " -f $x)
        OPENKEY="$OPENKEY $OPENPIN"
done
OPENKEY=$(echo $OPENKEY | sed -r 's/^.{0}//')
echo "knock $HOSTNAME $OPENKEY" >> $OPENKEYFILE
echo "sleep 1" >> $OPENKEYFILE
echo "ssh $USERNAME@$HOSTNAME -p 22" >> $OPENKEYFILE
echo ""
echo ""

#GENERATE CLOSEKEY-$HOSTNAME.sh
echo "GENERATING CLOSE KEY SCRIPT"
echo "#!/bin/bash" > $CLOSEKEYFILE
echo "" >> $CLOSEKEYFILE
for x in `seq $PINS -1 1`
do
    CLOSEPIN=$(echo $PINDIPS | cut -d " " -f $x)
        CLOSEKEY="$CLOSEKEY $CLOSEPIN" >> $CLOSEKEYFILE
done
CLOSEKEY=$(echo $CLOSEKEY | sed -r 's/^.{0}//')
```

```
echo "knock $HOSTNAME $CLOSEKEY" >> $CLOSEKEYFILE
echo ""
echo ""
echo ""
echo "3 Files has now been created: knockd.conf, $HOSTNAME-OPEN.sh, $HOSTNAME-CLOSE.sh"
echo "------------------------------------------------------------------------------"
echo "knockd.conf           - Copy and overwrite /etc/knockd.conf"
echo ""
echo "$HOSTNAME-OPEN.sh    - Use this script to open a connection"
echo ""
echo "$HOSTNAME-CLOSE.sh   - Use this script to close the connection"
```

Example output could be like this:

command: gen-knock.sh 8 demo.xample.com root

knockd.conf:

```
[option]
   UseSyslog

[openSSH]
   sequence = 4078,49668,18489,14340,11497,32969,30645,10145
   seq_timeout = 5
   command = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
   tcpflags = syn

[closeSSH]
   sequence = 10145,30645,32969,11497,14340,18489,49668,4078
```

```
    seq_timeout = 5
    command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
    tcpflags    = syn
```

demo.xample.com-OPEN.sh:

```
#!/bin/bash

knock demo.xample.com 4078 49668 18489 14340 11497 32969 30645 10145
sleep 1
ssh root@demo.xample.com -p 22
```

demo.xample.com-CLOSE.sh:

```
#!/bin/bash

knock demo.xample.com 10145 30645 32969 11497 14340 18489 49668 4078
```

idefix *June 17, 2016*

0

How do I combine this with IP white listing so it only opens up the port for my current IP address (from which I'm knocking)? I'm connected on a daily basis so I don't want the port to be open to all during that time. And I travel daily and connect with different IP addresses via my cell phone's hotspot. I want to knock on the ports and then be able to login via Putty, FileZilla, Stash, Git etc via SSH and only my current IP address should be allowed.

jellingwood *June 22, 2016*

0

@idefix: The `knockd` daemon should already be doing that. If you check out the `start_command` in the configuration file:

/etc/knockd.conf

```
. . .
start_command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
. . .
```

The option highlighted in red restricts the IP address allowed in to the current knocking IP. The `-s` option to `iptables` restricts by source IP. The `%IP%` is a variable that `knockd` uses to fill in the current knocking IP address when applying the rule. This should result in the port opening only to the IP address that correctly knocked.

> idefix *June 23, 2016*
>
> 0
>
> @jellingwood Ah great! Thx for the clarification!

cube *June 26, 2016*

0

@jellingwood - Any chance you could write up official tutorials for Ubuntu and CentOS using Web Knocker Firewall Service?

It's similar to fwknop but doesn't use libpcap.

cube *June 26, 2016*

Here is a very long blog entry/rant I ran into on port knocking while developing the Web Knocker Firewall Service:

http://bsdly.blogspot.com/2012/04/why-not-use-port-knocking.html

Even though it is a few years old, "That grumpy BSD guy" has some really good thoughts on what the problems are with (port) knocking. The main issue I see and mostly agree with is that blocking SSH is kind of problematic. However, if you are protecting other ports and don't want to set up a VPN (which just opens a VPN port), port or web knocking can be an extremely useful alternative to that lengthy process. Setting up SSH to only allow SSH keys to be used is going to be good enough for most people. If you set up a knocking tool, you should be 100% certain that it is working properly before closing off SSH port 22. That means making sure the service still starts and functions after rebooting, that the iptables rules are updating correctly (i.e. really know iptables), and have solid familiarity with the console should anything go horribly wrong. It's not exactly something for the average person to use. Anyone aiming to lock down port 22 should probably leave port 22 open for a few days after enabling a knocker and spot-checking the iptables rules to make sure that everything is doing exactly what it should be doing.

A better use for a knocking tool is to unblock e-mail servers on a host for a specific IP address. More specifically, the ability to retrieve e-mail and send authenticated e-mail. That allows just the authorized people who need to send e-mail through the system and retrieve their e-mail to even see the ports. Since port 22 isn't involved in this instance, it's no big deal to toss a knocking solution at the system. There's no need for anyone in China, Russia, or North Korea to even be attempting to access your e-mail inbox.

An alternative to knocking is fail2ban. It doesn't have IPv6 support yet - there is is an open issue in their issue tracker so it is coming. Enough failed connections results in an IP ban for a set amount of time. The tool limits the potential for an attacker to get into the system.

Overall, I feel like a combination of fail2ban and a knocker for restricted access non-port 22 ports is going to be a best-of solution for the vast majority of folks that is relatively easy to deploy with minimal repercussions should something go wrong.

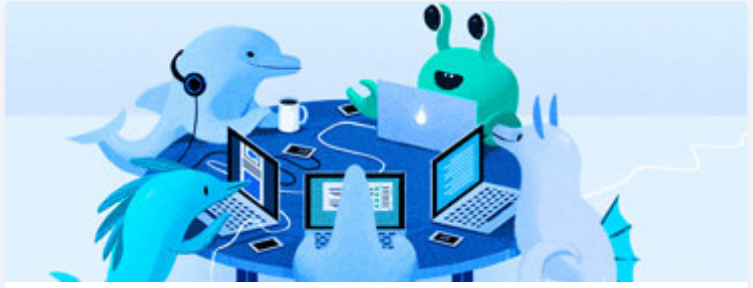
robsondantas *November 28, 2016*

Hey there! great tutorial.

Could you please fix the first sample knockd.conf where you are using iptables -A instead of iptables -I?

Was trying using that sample first and noticed rules were appended, not inserted.

Also, for some reason, maybe a bug on recent knockd version, only udp traffic is working properly.

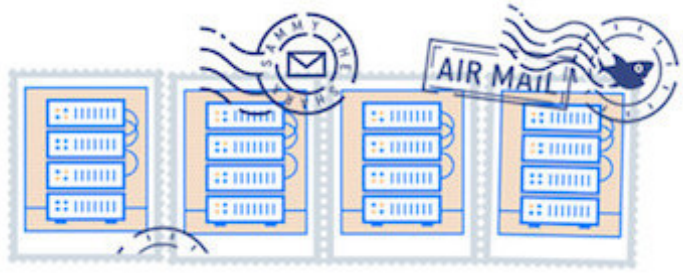


BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

CONNECT WITH OTHER DEVELOPERS

Find a DigitalOcean Meetup near you.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

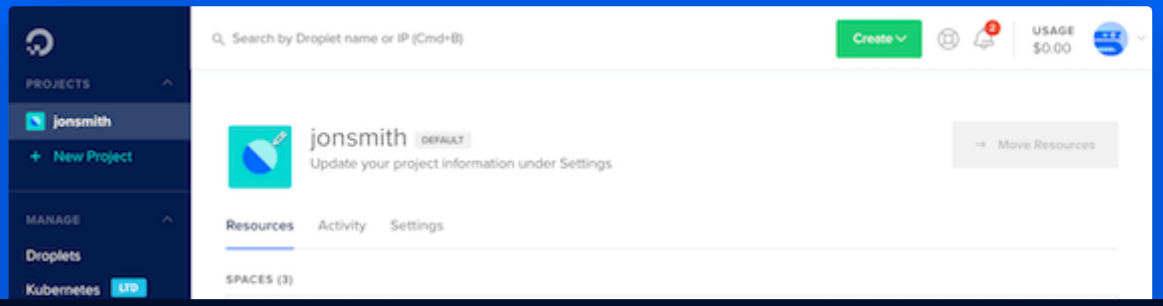
Featured on Community: Intro to Kubernetes, Learn Python 3, Machine Learning in Python, Getting started with Go, Migrate Node.js to Kubernetes

DigitalOcean Products: Droplets, Managed Databases, Managed Kubernetes, Spaces Object Storage, Marketplace

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More



Company

About
Leadership
Blog
Careers
Partners
Referral Program

Products

Products Overview
Pricing
Droplets
Kubernetes
Managed Databases
Spaces

Community

Tutorials
Q&A
Projects and Integrations
Tags
Product Ideas
Meetups