Securing Your Server’s SSH Configuration

Originalartikel

Backup

<html> <p>Are your SSH log files flooding with failing login attempts? I&#8217;ve seen many questions on websites such as Stackoverflow and Stackexchange from worried people that someone is actively targeting their servers with brute-force password logins attempts. Let me get one thing straight first: <em>you are not special! </em>It&#8217;s part of internet life: many botnets constantly attempt to login to servers. These can be random IP addresses or known ranges such as Amazon AWS EC2 instances or DigitalOcean droplets. There&#8217;s nothing much you can do about this except for making sure that your server is securely set up.</p>&#13; &#13; <p>One of the key security measures you have to take is properly setting up your SSH configuration. If password login is enabled and you are using an insecure password, a brute-force method might give someone access to your server. In this blog post, I&#8217;ll discuss some methods for properly setting up SSH on your server.</p>&#13; &#13; <h2>Securing Your Server</h2>&#13; &#13; <p>There are different ways to tighten up the security on your server. By far, the safest approach is to disable password logins altogether, though there are situations where this isn&#8217;t practical. In total, I&#8217;ll discuss five different methods for hardening your server:</p>&#13; &#13; <ul><li>Disable password logins</li>&#13;

```
<li>Block too many login attempts</li>&#13;
<li>White-list specific IP addresses</li>&#13;
<li>Change your SSH port</li>&#13;
<li>Port knocking</li>&#13;
```

</ul><h2>Disable Password Logins</h2>&#13; &#13; <p>By far, the most secure method of protecting your servers is by disallowing password logins altogether. Passwords are many times less secure than logging in with a public token. The computing power required to brute-force an SSH key is not within reach for the coming years. A password though, and especially a weak one, can definitely potentially be cracked using brute force.</p>&#13; &#13; <p>Disable password logins by editing the <em>/etc/ssh/sshd_config</em> file with your favorite editor. Find the line that contains the

```
PasswordAuthentication
```

property and disable it with:</p>&#13; &#13; <pre lang=„text“>PasswordAuthentication no</pre>&#13; &#13; <p>Before restarting the ssh daemon to take the changes into effect, be sure you have created and configured an SSH key to use for login. Without an SSH key, you will not be able to login to your server anymore after disabling password logins. If you don&#8217;t know how to create and configure an SSH key, check out this <a href=„https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2“>DigitalOcean blogpost</a> that explains exactly how to do so.</p>&#13; &#13; <h2>Block To Many Login Attempts</h2>&#13; &#13; <p>It might very well be that disabling password logins is simply not suitable for your use case. It then becomes especially important to block brute force attacks. There are multiple ways to do this, but let me focus on two specific methods here.</p>&#13; &#13; <h3>Recent</h3>&#13; &#13; <p>First,

```
IPTables
```

has a built in module called &#8220;recent&#8221; for blocking too many connections attempts. We can specify both an interval and the number of attempts within that specified interval. An example configuration is as follows:</p>&#13; &#13; <pre lang=„text“>iptables -N SSHBLOCK&#13; iptables -A INPUT -p tcp -m tcp –dport 22 -m state –state NEW -j SSHBLOCK &#13; &#13; iptables -A SSHBLOCK -m recent –name blocks –set –rsource &#13; iptables -A SSHBLOCK -m recent –name blocks –update –seconds 60 &#13;

1. -hitcount 5 –rsource -j DROP</pre>&#13;

&#13; <p>I am going to assume at least some basic knowledge about

```
iptables
```

here. If you are not familiar with the syntax, be sure to find a tutorial somewhere that explains the basic concepts of configuring

```
iptables
```

. Always be <strong>very </strong>careful when working with firewalls; it&#8217;s possible to lock yourself out of your system if you make a mistake.</p>&#13; &#13; <p>We begin with creating a new chain in the first rule to which we forward all the SSH connections in the second rule. Though not required per s&#233;, it makes managing the rules related to the rate limiting a bit easier.</p>&#13; &#13; <p>We keep track of the attempted connections in a list called &#8220;

```
blocks
```

&#8221;. The third rule adds the connecting IP address to this list. If the IP address was already in there, it will be appended. The final rule checks if the IP address is already within the list and if it has tried to connect 5 times within 60 seconds. If so, the connection is dropped and the IP address will have to wait 60 seconds for another attempt to make a connection. Anything not picked up by these rules is forwarded and the default chain policy decides what to do with it.</p>&#13; &#13; <p>You can keep track of the connections being blocked by looking at the <em>/proc/self/net/xt_recent/blocks</em> file. The exact location of the file might differ with your distribution and kernel version. The &#8220;<em>blocks</em>&#8221; filename is the same as the name of the list we defined in the

```
iptables
```

rules. Using the

```
watch
```

command, we can watch this file while attempting some connection attempts:</p>&#13; &#13; <pre lang=„text“>watch -n 1 cat /proc/self/net/xt_recent/blocks</pre>&#13; &#13; <p>In another shell, attempt to connect to your server a few times. As you do this, you will see your IP address being listed in the file. Don&#8217;t be surprised if you see some other IP addresses popping up in the meantime.</p>&#13; &#13; <p>The total number of IP addresses and number of connections per IP address that are stored by the recent module can be configured. The following two parameters are especially of interest:</p>&#13; &#13; <ul><li>

```
ip_list_tot
```

: The total number of IP addresses stored</li>&#13;

```
<li><code>ip_pkt_list_tot</code>: The total number of connections per IP
address stored</li>&#13;
```

</ul><p>The current values for these parameters can be found in the <em>/sys/module/xt_recent/parameters/{ip_list_tot, ip_pkt_list_tot}</em> files. Changing these values requires changing the parameters for the

```
xt_recent
```

kernel module, which is out of scope for this blog post. It&#8217;s good to know though that the file won&#8217;t grow until your mount is completely full!</p>&#13; &#13; <h3>Fail2Ban</h3>&#13; &#13; <p>A second method for blocking too many SSH connection attempts is by using <a href=„https://github.com/fail2ban/fail2ban“>Fail2Ban</a>. Fail2Ban can do <em>much</em> more than blocking SSH login attempts. It also supports blocking connections to software such as Apache, Nginx, HaProxy and MySQL. For now, let&#8217;s setup a simple configuration for blocking our SSH login attempts.</p>&#13; &#13; <p>The GitHub page I linked to includes the installation instructions, however be sure to first check if a package exists for your distribution. Once installed, create the file <em>/etc/fail2ban/jail.local </em>with the following contents:</p>&#13; &#13; <pre lang=„text“>[DEFAULT]&#13; # ban for 60 seconds&#13; bantime = 60&#13; &#13; # ban when 3 attempts are made within 60 seconds&#13; findtime = 60&#13; maxretry = 3&#13; &#13; # block through iptables&#13; banaction = iptables-multiport&#13; &#13; [sshd]&#13; # enable the above settings for sshd&#13; enabled = true</pre>&#13; &#13; <p>I&#8217;ve added comments to the configuration to explain the different settings. Restart Fail2Ban to take the new settings into effect. Try to login a few times to your server and you will find that you cannot connect anymore. After 60 seconds, you will be able to attempt a login again.</p>&#13; &#13; <p>The advantage of IP tables compared with Fail2Ban is that you do not need to install an additional package to your system. On the other hand, Fail2Ban is easier to setup and it supports much more than just blocking ssh connections. Both tools do a good job, so pick the right one for your situation!</p>&#13; &#13; <p><strong>Important</strong>: I cannot stress enough that whatever brute-force-blocking mechanism you have set up, its use can totally be undone by using an insecure password. If you really cannot disable password logins, always be sure to use a secure password.</p>&#13; &#13; <h2>White-list Specific IP Addresses</h2>&#13; &#13; <p>If you only login to your server from one or a few specific IP addresses, it&#8217;s an option to white-list only those IP addresses in your firewall to the SSH port. You can open up your firewall for your home address and work address, for example. However, if you&#8217;re at a friends house and want to login to your server, you are out of luck. There will certainly be use cases where this is a perfectly viable setup though.</p>&#13; &#13; <p>There are two ways for whitelisting specific IP addresses for your SSH daemon. First, we can use

```
iptables
```

for this. If your current

```
iptables
```

rules list is empty, the following rules will suffice:</p>&#13; &#13; <pre lang=„text“>iptables -A

INPUT -p tcp -m tcp –dport 22 -s <strong>[your_ip_address]</strong> -j ACCEPT&#13; iptables -A INPUT -p tcp -m tcp –dport 22 -m state –state NEW -j DROP</pre>&#13; &#13; <p>The first rule will accept your IP address and allow you to make a connection. Anyone else goes through to the second rule and is blocked if they try to connect to port 22. What happens to any other connection attempt depends on the default chain policy. If you already have some existing rules, be sure to tailor this for your specific ruleset.</p>&#13; &#13; <p>Second, you can change your sshd configuration to allow only specific IP addresses. In <em>/etc/ssh/sshd_config</em>, we can use the following syntax:</p>&#13; &#13; <pre lang=„text“>AllowUsers [username]@[ip_address]</pre>&#13; &#13; <p>The

```
[username]@[ip_address]
```

part can be used multiple times to allow multiple IP addresses and users.</p>&#13; &#13; <h2>Change your SSH Port</h2>&#13; &#13; <p>By default, you connect to port 22 to connect with the SSH daemon (

```
sshd
```

) on a server. Changing this port will still allow you to connect and will probably block at least 90% of the automated scripts that try to break into your server. However, whatever software you use to connect with your server over SSH will need to be changed to connect with the new port. If you have any automation scripts or external software that uses SSH to connect, be sure to check beforehand if they have support for connecting to a different port.</p>&#13; &#13; <p>Changing your SSH port is very easy. Open up the <em>/etc/ssh/sshd_config</em> file in your favorite editor and change the following line:</p>&#13; &#13; <pre lang=„text“>#Port 22</pre>&#13; &#13; <p>Uncomment the line and change the port to a new value. It&#8217;s recommended to change your port to a value &gt; 49152. Following the <a href=„http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml“>IANA port assignment guidelines</a>, this will minimize the chance of your port colliding with an another service that also uses this port. Finally, be sure to restart your sshd deamon to take the change into effect (

```
systemctl restart sshd
```

when using

```
systemd
```

). Connecting to a different port with SSH is easy with the

```
-p
```

parameter:</p>&#13; &#13; <pre lang=„text“>ssh -p [port] [user]@[host]</pre>&#13; &#13; <p>Remember that whoever knows your SSH port can still execute any brute force attempts if no other mechanisms are in place to battle that. A port scanner will still show your open port, so keep that in mind.</p>&#13; &#13; <h2>Port Knocking</h2>&#13; &#13; <p>This is by far the most creative way to add an additional layer of protection. By default, we close our SSH port which will block anyone who tries to connect. Only when a specific sequence of connection attempts (the &#8220;knocking&#8221;) on specific ports is made, the SSH port is opened temporarily so that someone can connect. These other ports don&#8217;t need to be open; the

```
knockd
```

service will also listen on closed ports.</p>&#13; &#13; <p>As

```
knockd
```

will open up your SSH port after a correct sequence, we need to make sure that this port is closed by default. Closing your SSH port is always tricky as it might block you out of your server. It&#8217;s best to first attempt this on a test server before doing it on your actual server. Given that no other

```
iptables
```

rules are set, the following rule will block any <em>new</em> incoming SSH connections. This means that your current open connection will not be dropped:</p>&#13; &#13; <pre lang=„text“>iptables -A INPUT -p tcp -m tcp –dport 22 -m state –state NEW -j DROP</pre>&#13; &#13; <p>The

```
knockd
```

service is available for Debian distributions through installation with

```
apt-get
```

. For other distributions, you might need to install from source which you can find on the <a href=„http://www.zeroflux.org/projects/knock“>knockd website</a>. On Ubuntu, install the service with the following command:</p>&#13; &#13; <pre lang=„text“>apt-get install knockd</pre>&#13; &#13; <p>Next, open up the

```
knockd
```

configuration file (<em>/etc/knock.conf</em>) with your favorite text editor. As you will see, there is already an example configuration set. This configuration requires us to close the SSH port with a different sequence of connection attempts. We can change this so that the port is automatically closed after a defined interval, such as 20 seconds. Yes, you will have 20 seconds to login to your server after the knocking sequence! You can use the following configuration for this:</p>&#13; &#13; <pre lang=„text“>[options]&#13; UseSyslog&#13; &#13; [SSH]&#13; sequence = 8888,7777,6666&#13; tcpflags = syn&#13; seq_timeout = 10&#13; start_command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp –dport 22 -j ACCEPT&#13; cmd_timeout = 20&#13; stop_command = /sbin/iptables -D INPUT -s %IP% -p tcp –dport 22 -j ACCEPT</pre>&#13; &#13; <p>The configuration is pretty self explanatory. We configure three ports that consequently need to be connected to. We also specify that this sequence needs to be performed within 10 seconds and that the port will close again after 20 seconds. Opening and closing (or &#8220;starting&#8221; and &#8220;stopping&#8221;) the SSH port is done through two

```
iptables
```

rules. Of course, change this line if you have changed your SSH port to another one.</p>&#13; &#13; <p>Before we can start

```
knockd
```

, we need to edit the <em>/etc/default/knockd</em> file. Find the line that says

```
START_KNOCKD=0
```

and change this to

```
START_KNOCKD=1
```

. Save the file and start knock with

```
service knockd start
```

.</p>&#13; &#13; <p>&#8220;Knocking&#8221; on this server from another server can be done with the same

```
knockd
```

package. On a different server, install the same package. We can use the following command now:</p>&#13; &#13; <pre lang=„text“>knock [ip_address] 8888 7777 6666</pre>&#13; &#13; <p>Now, connect to the server with SSH and you should be able to connect to the server. Logout, wait a second or 20 and try to login again. The port should now be closed again.</p>&#13; &#13; <p>Similar to changing your SSH port, this approach requires you to change the way you login to your server. Again, if you use any external services that login to your server, make sure they will be able to do so with

```
knockd
```

enabled. You do not need to use the

```
knockd
```

service per s&#233; to &#8220;knock&#8221; on a server. Connecting to the port sequence with

```
nmap
```

will yield the same result. Keep this in mind if you need to automate the &#8220;knocking&#8221; for a different service that connects with your server.</p>&#13; &#13; <h2>Conclusion</h2>&#13; &#13; <p>There are many ways to properly harden the security of your server. In this blog post, we specifically looked at SSH. The internet is a scary place with many botnets looking to get into any poorly configured servers. It&#8217;s a good thing then that there are many ways to protect your server:</p>&#13; &#13; <ul><li>Whenever possible: disable password logins</li>&#13;

```
<li>If that really isn&#8217;t possible: use a strong password!</li>&#13;
<li>Block brute force attacks by blocking too many login attempts</li>&#13;
<li>Consider adding some extra <em>obscurity</em> such as changing your SSH
port or enabling port knocking</li>&#13;
```

</ul><p>Both changing your SSH port and enabling port knocking can be considered a <em>security through obscurity</em> approach: hiding our service instead of really securing it. When used in combination with a solid security mechanism such as a strong password, I consider such an approach as an additional layer of security. Practicalities aside, it will only make it harder to get into your server.</p>&#13; &#13; <p><img height=„1“ src=„https://sanderknape.com/?feed-stats-post-id=200“ width=„1“/></p>&#13; &#13; <p>The post <a href=„https://sanderknape.com/2016/11/securing-your-server-ssh-configuration/“ rel=„nofollow“>Securing your server&#8217;s SSH configuration</a> appeared first on <a href=„https://sanderknape.com/“ rel=„nofollow“>Sander Knape</a>.</p>&#13; &#13; &#13; </html>