3

FILTERING NETWORK TRAFFIC WITH FIREWALLS

A *firewall* monitors and filters incoming and outgoing network traffic. There's a general misconception that the firewall is always the last line of defense; in reality, a perimeter firewall should be the first obstacle adversaries encounter when they try to penetrate any network, large or small. Every time a web browser accesses a website, a messaging program sends a message, or your email client sends and receives email, the traffic generated should pass through at least one firewall along its journey.

In this chapter, you'll explore two firewall solutions: iptables and pfSense. In Linux, iptables is a common firewall often used as a *host firewall* (that is, a firewall that allows or denies traffic on a specific endpoint). pfSense, which can be implemented either as an open source software firewall or as a hardware firewall using the appliances sold by Netgate, is used as a perimeter or boundary firewall responsible for filtering traffic for entire networks or network segments.

Types of Firewalls

A *hardware firewall* can be physically and logically placed in a network. A *software firewall*, installed as an application on an endpoint, requires more configuration of both the firewall and its connected devices to filter traffic effectively. By using one or both of these, you're able to effectively reduce your *attack surface*, which comprises the points where an adversary can try to infiltrate, compromise, or exploit your network. Ideally, attack surfaces should be as small as possible.

A *perimeter firewall*, installed between your private network and other networks like the internet, can be either software- or hardware-based. Perimeter firewalls are placed at the physical and logical border of the network, making it the first thing with which traffic bound for your internal network from the public internet communicates, as well as the last thing in your network that traffic bound for the internet passes through, as shown in **Figure 3-1**.

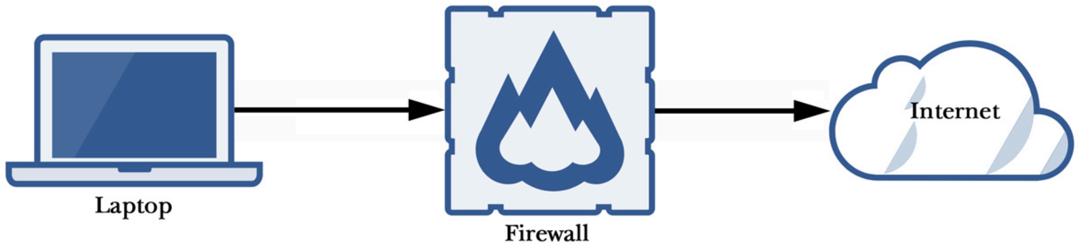


Figure 3-1*: A perimeter firewall*

Firewalls allow or deny (block) traffic based on a *ruleset* containing a configured list of rules. The way those rules are applied to traffic depends on the type of firewall you're using. The most common type, a *packet-filtering firewall*, inspects each packet of data attempting to make it into (or out of) your internal network and then checks that packet against its ruleset. If the packet contents match a rule in the firewall ruleset, the firewall will either allow or deny that traffic, depending on what that rule indicates it should do.

There are also stateful and stateless firewalls. A *stateful* firewall tracks all inbound and outbound connections and monitors each connection as a unique conversation between two endpoints. This method provides the firewall with context about any given connection and allows more granular control of traffic. By contrast, a *stateless* firewall doesn't record information about each connection. Both iptables and pfSense are stateful firewalls.

Almost all operating systems come with a built-in software firewall, known as a *host-based firewall*, that filters traffic specific to that host. Most Windows and Mac devices ship with an out-of-the-box host-based firewall whose basic ruleset is functional, if not exhaustive. By design, this firewall works as is for ordinary purposes; users don't need to configure their own firewall, lessening confusion as well as the need for technical support from computer manufacturers. On Linux devices, you'll have to configure a firewall—you'll see how to do this in the next section.

It's best to use both a host firewall and a perimeter firewall and to configure them correctly for your network to add multiple layers of defense.

iptables

Linux's iptables utility offers incredible flexibility in filtering traffic entering, traversing, or leaving a network. The firewall organizes its rules in *policy chains*, lists of rules that analyze and match packets based on their contents. Each rule determines what the firewall will do with a packet that matches its definition—it might allow, reject, or drop the packet. When a packet is allowed, it passes through the firewall unhindered. When dropped, the firewall discards the packet and sends no response back to the sender. If a packet is rejected, the firewall discards the packet and sends a rejection message back to the sender, providing context about your network and the firewall you're using.

There are three main types of policy chain: *input chains*, *output chains*, and *forward chains*. Input chains determine whether to allow certain traffic into the network from an external source, such as a *virtual private network (VPN)* connection from a remote location. A VPN is a method for logically—rather than physically—connecting to disparate networks, usually for remote access from one network to the other. VPNs are covered in greater detail in **Chapter 5**.

Output chains indicate whether the firewall should allow certain outbound traffic to an external network. For example, *Internet Control Message Protocol (ICMP)* is primarily used to diagnose network communication issues. ICMP ping packets are outbound traffic that pass through the output chain. A ping is a query from one device to another, usually to determine whether a connection can be made between the two. You'd need to allow the ping packets to travel from your device, through your firewall, and across several other devices on the public internet, to finally reach their destination. If your output chain blocks ICMP traffic, your device would be unable to ping anything, as the firewall would block or drop those packets.

In most cases, your stateful firewall rules should allow both new and established connections. For example, if you create an output chain to allow your device to ping Google, you need to tell the firewall to allow inbound traffic related to established connections. Otherwise, your device will send a ping out to Google that passes through your firewall, but the response from Google would get blocked by your firewall.

Forward chains forward the traffic your firewall receives to another network. In a small office or home network, host-based firewalls rarely use forward chains, unless the firewall is configured to serve as a router. A perimeter firewall would use the forward chain to route traffic from your internal network to the external network, or from one network segment to another, likely using network address translation (NAT), as discussed in **Chapter 1**. However, a configuration of this type is more complicated than necessary for small networks and would better fit an enterprise network.

By using these policy chains, you'll be able to control the traffic traversing your network at a very granular level. In the following chapters, you'll create several Linux servers, each of which would benefit from its own host-based firewall. I recommend configuring iptables on each of these servers using the following instructions.

NOTE *iptables isn't capable of securing IPv6 networks and traffic. If you plan to use IPv6 in your network, you'll need to use ip6tables in addition to iptables. Unless you have a strong use case for IPv6 in your network, I recommend disabling IPv6 completely. Disabling IPv6 is covered in* **Chapter 4**.

#12: Installing iptables

If you've already built a standard Ubuntu server following the steps from **Chapter 1**, you can start configuring its iptables firewall. Once you've mastered the basics, use that knowledge to configure iptables on all your Linux endpoints. Otherwise, go back and create your Ubuntu system now.

Recent versions of Ubuntu have iptables installed by default, so log in via SSH as a standard, non-root user, and check for iptables by running a version check:

```
$ sudo iptables -V
[sudo] password for user:
iptables v1.8.7 (nf_tables)
```

If iptables is installed, the server should return version information, as shown here. Your version may be different.

If iptables isn't installed, you'll receive an error, in which case, install iptables:

```
$ sudo apt install iptables
```

Once it's installed, run the same version check to confirm that the installation succeeded.

Next, install `iptables-persistent`, a tool that allows you to save your firewall configurations and automatically reload them after a reboot of the server:

```
$ sudo apt install iptables-persistent
```

An installation wizard should take over your terminal window. You'll be shown the file in which your server will save the firewall rules (the default file is */etc/iptables/rules.v4*) and told that rules from this file will load at system startup. Also, you'll need to save any changes to firewall rules manually beyond this installation process. Select **Yes** to save any current firewall rules. If you don't install this component, you'll have to reconfigure your firewall every time you restart the server.

You can now check the current policy chains like so:

```
$ sudo iptables -L

[sudo] password for user:

Chain INPUT (policy ACCEPT)

target     prot opt source               destination

Chain FORWARD (policy ACCEPT)

target     prot opt source               destination
```

```
Chain OUTPUT (policy ACCEPT)

target     prot opt source               destination
```

In the output, `policy ACCEPT` indicates that, by default, iptables accepts all traffic for input, output, and forwarding. This default behavior is desirable because it'll work without any user configuration. However, it's an insecure solution, so let's modify it.

iptables Firewall Rules

When creating iptables rules, keep in mind that order matters. As traffic reaches your firewall, iptables checks its rules one after the other *in the order they appear*. If the traffic matches a rule, iptables won't check any further rules—if the first rule in your list of 50 denies all traffic, the firewall will interpret this rule, reject the traffic, and stop processing, which effectively isolates your device entirely. Alternatively, if you have the same 50 rules but the first rule allows all traffic, all traffic will be allowed to pass through the firewall. You should avoid both of those situations.

To understand how to construct an iptables firewall rule, take a look at this example:

```
$ sudo iptables -A INPUT -p tcp --dport 22 -m conntrack \

     --ctstate NEW,ESTABLISHED -j ACCEPT
```

Immediately after `sudo`, iptables is invoked to begin the rule definition. The next argument determines whether the rule will be appended to (`-A`), deleted from (`-D`), or inserted into (`-I`) the specified policy chain. You can also specify `-R` in this position when replacing or updating an existing rule. The `INPUT` indicates that a rule in the input chain is being modified. You also can specify `OUTPUT`, `FORWARD`, or other policy chains.

In most cases, iptables needs to know the protocol and port to which the rules relate. In this example, `-p tcp` indicates the rule will apply only to TCP traffic, and `--dport 22` tells iptables that the rule applies to packets with a destination port of 22. Both of those settings are optional. You can specify multiple ports with this syntax: `--match multiport --dports` `port1,port2,port3`.

NOTE Transmission Control Protocol (TCP) *is a* reliable *transmission protocol, designed to ensure successful delivery of packets over a network. If a computer experiences packet loss during communication that's using TCP, those lost packets will be retransmitted, ensuring all of the data sent is eventually received by the destination host.* User Datagram Protocol (UDP) *is an* unreliable *protocol and does not ensure successful transmission of data or retransmit lost packets. UDP is used when some packet loss is acceptable and usually results in a faster connection. TCP is used when reliability matters, and every packet must be transmitted successfully.*

The iptables firewall offers multiple matching modules, and you can specify the module to use with the `-m` argument. In this example, `conntrack`, a tool that allows stateful packet inspection, is used (also optional). Some other tools include `connbytes`, which creates rules based on the amount of traffic transferred, and `connrate`, which matches on the transfer rate of the traffic. See the iptables man page for more details: ***https://linux.die.net/man/8/iptables/***.

Next, `--ctstate` tells iptables to allow and track traffic for the types of connections that follow—in this case, `NEW` and `ESTABLISHED`. Many values are available for connection state, but the most frequently used are `NEW`, `ESTABLISHED`, `RELATED`, and `INVALID`. New and established states are self-explanatory; the packets are part of new or established traffic flows. Related packets don't necessarily match an established connection, but they are expected by the firewall because an existing connection necessitates it (that is, it's expected based on the firewall's existing context). Invalid packets are any packets that don't match the criteria for any other states.

Finally, iptables will interpret `-j` and whatever follows it as the action to jump to (perform) when this rule is matched. Most commonly, it will be either `ACCEPT` to allow traffic matching this rule; `DROP`, or `REJECT`, to deny or block the traffic; or `LOG` to log the traffic to a logfile (more details on that later).

Now that you understand the fundamentals of iptables rules, you'll configure your firewall to allow and deny traffic.

Configuring iptables

When configuring iptables, first add rules to drop invalid traffic:

```
$ sudo iptables -A OUTPUT -m state --state INVALID -j DROP

$ sudo iptables -A INPUT -m state --state INVALID -j DROP
```

Then, add rules to accept traffic related to existing connections, as well as established connections and the loopback address to avoid any issues later (a *loopback address* is an internal address that computers use for testing and diagnosing network issues):

```
$ sudo iptables -A INPUT -m state --state RELATED,ESTABLISHED
-j ACCEPT

$ sudo iptables -A OUTPUT -m state --state
RELATED,ESTABLISHED -j ACCEPT

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

This allows the firewall to accept traffic matching a known connection or related to a connection in progress and discard any unexpected packets (which can protect your network from unsolicited or malicious network scanning activity).

Once you've run these commands to enter the rules into the policy chains, rerun the list command to ensure they've been accepted:

```
$ sudo iptables -L

Chain INPUT (policy ACCEPT)

target     prot opt source          destination

DROP       all  --  anywhere        anywhere        state INVALID

ACCEPT     all  --  anywhere        anywhere        state
RELATED,ESTABLISHED

ACCEPT     all  --  anywhere        anywhere

Chain FORWARD (policy ACCEPT)

target     prot opt source          destination

Chain OUTPUT (policy ACCEPT)

target     prot opt source          destination

DROP       all  --  anywhere        anywhere        state INVALID

ACCEPT     all  --  anywhere        anywhere        state
RELATED,ESTABLISHED
```

Notice the rules have been added under the `INPUT` and `OUTPUT` chains. The `FORWARD` chain remains empty.

Next, ensure that your firewall allows SSH traffic. You can do this in two ways: by broadly allowing SSH or by allowing SSH only from a subset of devices in your network. To allow SSH traffic originating from all devices in your network, use the following command:

```
$ sudo iptables -A INPUT -p tcp --dport 22 -m conntrack --ct-
state NEW -j \ ACCEPT
```

Creating broad rules can be helpful when connecting to or from multiple devices using SSH within your network. However, allowing the uninhibited use of programs and leaving protocols completely open is not the most secure solution. You should allow services like SSH only to and from specific IP addresses or ranges, as allowing remote access or file transfer between your endpoints and any other device is risky.

You can reduce your attack surface by specifying a source IP address or range (for example, 192.168.1.25) in your input chain with the `-s source` option, so if you're configuring iptables on a virtual machine, you might choose to allow connections from a single host for management purposes and deny access to all other endpoints in your network:

```
$ sudo iptables -A INPUT -p tcp -s 192.168.1.25 --dport 22 -m
\

      conntrack --ctstate NEW -j ACCEPT
```

We append this rule to the `INPUT` policy chain using `-A`, destination port `22`, and protocol `TCP`. For `NEW` connections, iptables will `ACCEPT` traffic matching this rule. The port can be one of your choosing; just be sure that your SSH configuration matches your firewall rule. If the rule allows SSH on port 22 but your SSH configuration allows connections on port 2222, the firewall will block your SSH connections.

If you make a mistake, delete the rule by running the same command, substituting the `-D` option in place of `-A`:

```
$ sudo iptables -D INPUT -p tcp -s 192.168.1.25 --dport 22 -
m conntrack \ --ctstate NEW,ESTABLISHED -j ACCEPT
```

Alternatively, you can delete all the rules you've specified for any of your policy chains by using the `-F chain`, or `--flush chain`, parameter:

```
$ sudo iptables -F INPUT
```

With this basic set of rules, now you can tell iptables what to do with all other traffic (that you don't want entering or leaving your server or network). Once you've created rules to allow the specific traffic you want or need the firewall to allow, you can most likely block, deny, or drop everything else. You should do this after you've configured your firewall rules; otherwise, you might interrupt your connection and be unable to reconnect via SSH. Using the `-P` argument sets the default behavior of your policy chains and lets iptables know what to do with traffic that doesn't match your rules. To achieve this, set the policy chains' default behaviors to `DROP` this traffic:

```
$ sudo iptables -P INPUT DROP

$ sudo iptables -P FORWARD DROP

$ sudo iptables -P OUTPUT DROP
```

Using `-P` in this way is different from `-A` and `-I` used previously, because it doesn't affect the firewall rules themselves; instead, it deals with the overarching policies that govern traffic in your network. Where `-A` and `-I` append or insert rules for your firewall, respectively, `-P` configures the firewall behavior one level higher.

At this point, checking your iptables chains should return:

```
Chain INPUT (policy DROP)
```

```
target     prot opt source               destination

DROP       all  --  anywhere             anywhere             state INVALID

ACCEPT     all  --  anywhere             anywhere             state
RELATED,ESTABLISHED

ACCEPT     all  --  anywhere             anywhere

ACCEPT     tcp  -- 192.168.1.25          anywhere             tcp dpt:22 ctstate
NEW

Chain FORWARD (policy DROP)

target     prot opt source               destination

Chain OUTPUT (policy DROP)

target     prot opt source               destination

DROP       all  --  anywhere             anywhere             state INVALID

ACCEPT     all  --  anywhere             anywhere             state
RELATED,ESTABLISHED
```

Notice that the policy for all three chains has changed from `ACCEPT` to `DROP`, indicating the default behavior for each chain is to drop traffic that doesn't match any of the rules you've created. You should also be able to identify the rules you've added to the chains by comparing this output to the previous output listing the iptables rules. You may receive an error that DNS is failing, because the firewall is now blocking everything not explicitly allowed, including DNS (which runs on port 53). Resolve this issue by adding the following new rules:

```
$ sudo iptables -A OUTPUT -p udp --dport 53 -m conntrack --
ctstate NEW -j ACCEPT
```

```
$ sudo iptables -A OUTPUT -p tcp --dport 53 -m conntrack --ctstate NEW -j ACCEPT
```

These commands append rules to the output chain, allowing this server to make outbound requests for domain name resolution on UDP and TCP port 53. With the addition of these rules, the server can resolve domain names.

Test your firewall by trying to ping the server from another device in your network; you should receive an error, as ICMP isn't allowed through the firewall. Likewise, if you try to ping anything from the server itself, you should receive a similar error:

```
$ ping google.com -c 5

PING google.com (<ip_address>): 56(84) bytes of data.

ping: sendmsg: Operation not permitted

ping: sendmsg: Operation not permitted

ping: sendmsg: Operation not permitted

ping: sendmsg: Operation not permitted

ping: sendmsg: Operation not permitted

--- google.com ping statistics ---

5 packets transmitted, 0 received, 100% packet loss, time 4000ms
```

ICMP can be such a useful troubleshooting tool that you might decide to allow ping through your iptables firewall. To do so, add the following rules:

```
$ sudo iptables -A INPUT -p icmp -j ACCEPT
$ sudo iptables -A OUTPUT -p icmp -j ACCEPT
```

You may discover that you need to open additional ports in the firewall. For example, if you have a proxy installed or if you build one after reading **Chapter 6**, you'll need to open the proxy port (3128) in your firewall:

```
$ sudo iptables -A OUTPUT -p tcp --dport 3128 -m conntrack --ctstate NEW -j ACCEPT
```

In most cases, you should block web browsing in general from servers—there are few, if any, legitimate reasons to use servers for this type of activity. Ideally, from both an administrative and a security standpoint, servers should be single-purpose. Allowing any additional service—especially browsing the internet—on a server results in a larger attack surface and creates potential vulnerabilities in your network.

If you decide to allow this traffic from your server(s) so the server can retrieve software updates, create output rules for ports 80 and 443, the default ports for HTTP and HTTPS traffic, respectively:

```
$ sudo iptables -A OUTPUT -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

$ sudo iptables -A OUTPUT -p tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

The only difference between the HTTP and HTTPS rules is the port number.

Every time you add a rule, you should test it. The easiest way to do so, in this case, will be to first test your ability to browse the internet by using a web browser on the server (if you have the GUI installed) or by using `curl` in the bash terminal. Start by installing `curl`:

```
$ sudo apt install curl
```

If you don't have rules allowing HTTP and HTTPS, the install command will fail, as updates are typically done over HTTP. However, if you do have those rules in place, `curl` should have installed successfully, so you can now ensure ports 80 and 443 are open:

```
$ curl http://icanhazip.com

ipaddress
```

The address ***http://icanhazip.com/*** is a public service provider that will return your current public IP address when queried with `curl`. If you're shown your current public IP address, your firewall is configured correctly.

If you receive an error, one of your rules may have a problem. Check for typos, and if all else fails, delete your rules and start again using the `-D` or `-F` parameters discussed earlier. Once the firewall is correctly configured, feel free to add further rules as you deem necessary.

One particular set of rules to add are those that block traffic to specific IP addresses. Since most public websites can have multiple IP addresses, however, blocking a site using iptables isn't the best option, as you'd have to create rules for each unique IP address. In most cases, you'd be better off using a proxy, which we'll cover in **Chapter 6**.

If you want to use iptables to block sites—say, for example, to block all traffic to and from ***https://www.squirreldirectory.com/***, which currently resides at IP address 206.189.69.35—you would add the following rules to your `INPUT` and `OUTPUT` chains:

```
$ sudo iptables -A INPUT -s 206.189.69.35 -j DROP

$ sudo iptables -A OUTPUT -s 206.189.69.35 -j DROP
```

Typically, you'd add this type of rule to allow or deny traffic from a static, private IP address that isn't expected to change, and use a proxy for public IP addresses or URLs.

Logging iptables Behavior

You've now installed and configured the iptables firewall, but you haven't told it to log anything, so it produces no records of its behavior, which can make it difficult to troubleshoot issues or determine whether blocked traffic should have been blocked.

First, create a new, custom policy chain. Note that this configuration is an example of where rule order is critical. You can name the chain whatever you like, but here, we'll call it `LOGGING`:

```
$ sudo iptables -N LOGGING
```

The `-N` parameter is used to create new chains.

Next, add a rule at the end of each of the `INPUT` and `OUTPUT` chains that tells iptables to send any traffic that hasn't yet matched a rule to the new `LOGGING` chain:

```
$ sudo iptables -A INPUT -j LOGGING

$ sudo iptables -A OUTPUT -j LOGGING
```

Then, tell iptables to log only once per minute for each type of dropped packet:

```
$ sudo iptables -A LOGGING -m limit --limit 1/minute -j LOG \
    --log-prefix "FW-Dropped: " --log-level 4
```

This limit is optional, and you can set it to any period, such as `1/second`, `1/minute`, `1/hour`, or `1/day`. Limiting the number of log entries reduces both the noise within and the size of the logfiles. Add a prefix (`"FW-Dropped: "`) to the log information so the firewall log entries are easy to identify. Setting the logging level to `4` will log up to warning-level events, indicating an event that has a material effect on the server or the firewall. Increasing the number results in more events with lower severity being logged, which is useful when troubleshooting. Log levels 1 to 3 will log only events or errors with higher than warning-level severity.

Finally, the following command indicates to the firewall that, once logged, the packets should be dropped:

```
$ sudo iptables -A LOGGING -j DROP
```

Your firewall will now log all the dropped packets both inbound to and outbound from the server. By default, those logs will be kept in */var/log/messages*.

The last step is to save your firewall configuration. Remember that iptables configurations are temporary by default and won't survive a reboot, which is why we installed `iptables-persistent` in **Project 12**. To save your configuration, run the following command (`netfilter` is the command used by `iptables-persistent` for this purpose):

```
$ sudo netfilter-persistent save

run-parts: executing /usr/share/netfilter-persistent/plugins.d/15-ip4tables save

run-parts: executing /usr/share/netfilter-persistent/plugins.d/25-ip6tables save
```

With that, the firewall is ready to go.

You may consider adding temporary rules to your firewall, but remember the adage that "nothing is more permanent than a temporary firewall rule" (Austin Scott). In the case of adding a temporary rule to allow a user to download a file from the internet, for example, it would be better to find a different workaround, like using another host. If a rule like this is created and left in the firewall configuration, it creates a vulnerability and reduces the security offered by the firewall. Avoid temporary rules whenever possible.

pfSense

In addition to a firewall securing each endpoint in your network with iptables, you should implement a firewall like pfSense to secure your entire network at its border. Together, these firewalls add layers to your defense-in-depth strategy, making the job of any adversary more difficult with each level of complexity. You should place a perimeter firewall at the physical edge of your network—that is, as close to the internet as possible relative to the other endpoints in your network. For most, that position will be directly behind the modem/router or network boundary point that connects your network to your internet service provider. It is possible to achieve this logically, using virtual machines and the correct routing configuration. However, the best and most secure way to set up a perimeter firewall is to use a physical device.

Like iptables, the pfSense firewall is stateful. However, where iptables works as a feature installed on top of a base operating system like Ubuntu, pfSense is a fully fledged operating system. It's based on FreeBSD, an open source version of Unix (an operating system similar to Linux that uses its own kernel) that has user-friendly features like a web management interface and can be deployed as either a virtual machine or a physical appliance.

You have a few options when it comes to creating a physical firewall. The first is to build a device that suits this purpose from a computer with a small footprint, like the Intel Next Unit of Computing (NUC). However, for the same cost or far less, Netgate sells pfSense appliances that are easy to configure and basically ready to go out of the box.

For the sake of simplicity (and security), we'll discuss using a prebuilt device. This book will not cover building a device from scratch because the risk of misconfiguration is too high, especially when an inexpensive, secure solution is readily available. The Netgate 2100 Base pfSense+ costs around $400 at the time of writing. It's powerful enough to be capable of most anything you can throw at it, short of a full-blown enterprise network. The SG-3100 is a step up from the entry-level 1100 pfSense+ and is more fully featured. It also has higher bandwidth and is capable of greater throughput, so it's the ideal choice for smaller networks.

#13: Installing the pfSense Firewall

Upon receiving your pfSense device, remove it from the box and plug it into power. Connect an Ethernet cable from the WAN port on the device to any port on your cable, DSL modem, or network boundary point device. Connect another Ethernet cable from the LAN1 port to the Ethernet port on your computer.

To access the pfSense configuration page from your computer, browse to 192.168.1.1, the default IP address of the SG-3100. If that doesn't work, you may need to disconnect your computer from your regular network and manually set its IP address to 192.168.1.2 (or any other address in the 192.168.1.*x* range, except the pfSense IP of 192.168.1.1) using the following instructions. This is necessary only for the initial configuration of the device and should need to be done only once on the computer you use to set up the pfSense appliance.

macOS

1. Open **System Preferences**.
2. Click **Network**.
3. Select the Ethernet connection between your pfSense device and your computer, and then set the Configure IPv4 drop-down box to **Manually**.
4. Enter **192.168.1.2** into the IP Address field, set Subnet Mask to **255.255.255.0**, and enter **192.168.1.1** into the Router field.
5. Click **Apply**.
6. Open your web browser and browse to 192.168.1.1. You should be presented with the pfSense login page.

Windows

1. Open **Network and Internet Settings**.
2. Click **Change Adapter Options**.
3. Open the Ethernet connection between your pfSense device and your computer, and then click **Properties** ▸ **Internet Protocol Version** ▸ **(TCP/IP)** ▸ **Properties**.
4. Select the **Use the following IP address** radio button.
5. Enter **192.168.1.2** into the IP Address field, set Subnet Mask to **255.255.255.0**, and enter **192.168.1.1** into the Default Gateway field.
6. Click **OK** and close the remaining windows.
7. Open your web browser and browse to 192.168.1.1. You should be presented with the pfSense login page.

Linux

1. Open **Settings**.
2. Click **Network**.
3. On the Ethernet connection between your pfSense device and your computer, click the configure **Cog**.
4. Select the **IPv4** tab.
5. Select the **Manual** radio button.
6. Enter **192.168.1.2** into the IP Address field, set Netmask to **255.255.255.0**, and enter **192.168.1.1** into the Gateway field.

7. Click **Apply** and close the Settings windows.
8. Open your web browser and browse to 192.168.1.1. You should be presented with the pfSense login page.

NOTE *If you receive a warning message indicating the site is not private or is unsafe, click through to the login page. This warning appears because there's no SSL certificate configured, and you can ignore it for now. However, be wary of errors like this elsewhere; generally, an SSL certificate error (especially on the internet) is a serious warning that the page you're trying to access is insecure.*

On the pfSense login page, log in with the credentials provided when you received your device. Once you're logged in, accept the end-user license agreement (EULA) if one is presented. Take a moment to review the system information, and then click the **System** menu at the top of the page and start the **Setup Wizard**. Use the following steps to finish setting up pfSense:

1. At the welcome screen, click **Next**.
2. If the Support screen is displayed, click **Next**.
3. On the General Information screen, choose a hostname for the device, or leave it as the default, pfSense.
4. If you have a domain configured in your environment, enter it in the Domain field.
5. Ignore the DNS settings for now and click **Next**.
6. On the Time Server Information screen, accept the default Time server hostname, unless you have a time server in your environment, in which case enter its details here.
7. Be sure to select the correct time zone, and then click **Next**.

You should now see the Configure WAN Interface page. You can use this page to configure your pfSense appliance to connect to your internet service provider. We'll cover the most common configuration here, called *PPPoE*, that will most likely match the settings in your current modem/router. If not, contact your internet service provider for the correct configuration details for your connection.

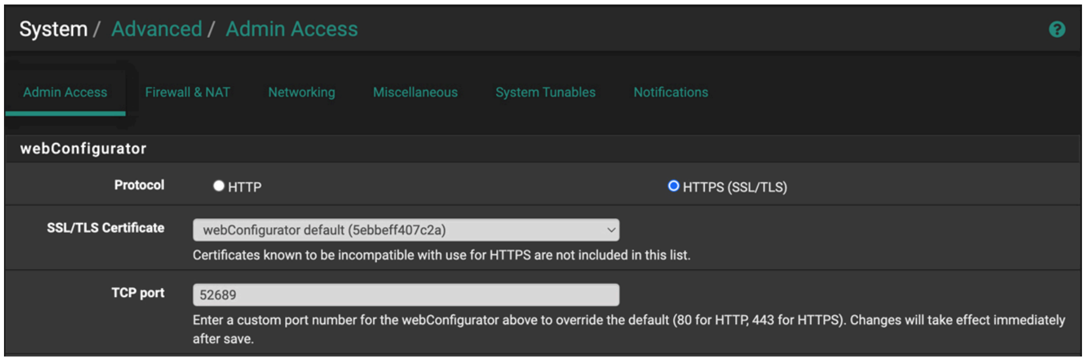
8. In the SelectedType box, select **PPPoE**.
9. Skip the General configuration options to accept the default settings.
10. Static IP Configuration and DHCP client configuration should be grayed out, so move on to PPPoE configuration.
11. Enter the username and password provided to you by your internet service provider.
12. Accept all other settings as default and click **Next**.
13. Set the LAN IP address of the pfSense appliance. You can choose to keep the IP addressing scheme you identified in Chapter 1 by giving this device the first IP in the address range (192.168.1.1 in the case of an address scheme of 192.168.0.0/16), or you can change it by specifying a different LAN IP address on this page. If you'd like addresses in the 10.0.0.0/8 range, specify 10.0.0.1, and so on. Then click **Next**.
14. Change the administrator password. Be sure to select a strong passphrase at least 12 characters in length or longer, and save it in a password safe (we'll discuss this further in Chapter 11). Once done, click **Next** ▸ **Reload** ▸ **Finish**.

Your initial configuration is now complete. Assuming the device has been able to connect to your internet service provider with your credentials, you should be able to browse the internet. If not, you may have to do some troubleshooting. The best place for troubleshooting any issues is in the System Logs page within the Status menu at the top of the web interface. With any luck, any issues will become evident once you've looked over the logs. If you're sure you entered all of the configuration details correctly, reach out to your internet service provider to ensure your settings are correct.

Hardening pfSense

Your firewall is now configured and running, and it should already do a brilliant job of rejecting unsolicited traffic attempting to enter your network. However, you can take additional steps to ensure your device and network are even more secure.

While logged in to your pfSense device, click **System ▸ Advanced**.



webConfigurator

Protocol: HTTP [✓ HTTPS (SSL/TLS)]

SSL/TLS Certificate: webConfigurator default (5ebbeff407c2a)
Certificates known to be incompatible with use for HTTPS are not included in this list.

TCP port: 52689
Enter a custom port number for the webConfigurator above to override the default (80 for HTTP, 443 for HTTPS). Changes will take effect immediately after save.

Figure 3-2*: Advanced pfSense menu*

In the Advanced menu tabs, you can change the protocols, ports, and proxy settings that pfSense uses, among other things. Click **Save** before leaving a tab if you change any settings.

In the Admin Access tab shown in **Figure 3-2**, set webConfigurator Protocol to **HTTPS** to ensure a secure, encrypted connection to the device. It's always preferable to use HTTPS instead of the unencrypted HTTP protocol because the added encryption ensures that, even if the network traffic is intercepted by an adversary, the adversary can't decrypt it.

In the next section of the Admin Access page (not shown in **Figure 3-2**), you can change the SSH options. I recommend not allowing SSH access to the device all the time—that would be similar to leaving your front door unlocked at night. If you allow SSH access only while you're actively using it, adversaries are able to attempt to access your network this way only while the service is available. Having the service turned off 99 percent of the time means attackers have only 1 percent of the time to attempt to breach the network. Disable this option unless you're actively connecting to the device via SSH. Once these settings have been updated, click **Save**.

On the Networking tab, you can enable or disable IPv6 traffic. If you're not actively using IPv6, disable it here to reduce your attack surface. Doing so should make the remaining settings on this page moot.

If you're using a proxy for your web traffic, enter your proxy details in the Miscellaneous tab. If you're planning to build your own proxy server using the steps detailed in **Chapter 6**, revisit this chapter and enter the proxy details at that stage.

pfSense Firewall Rules

The default pfSense firewall rules will block traffic from both RFC1918 private network connections and *bogon networks* from entering your network from the internet. RFC1918 addresses, discussed in **Chapter 1**, are IP address ranges reserved for private, internal network use only, meaning addresses in these ranges should not appear on the public internet. They include the following ranges: 192.168.0.0/16, 10.0.0.0/8, and 172.16.0.0/12. If any of these happen to appear on the internet, your firewall should find this suspicious and discard that traffic. Similarly, bogon networks or bogon addresses are those that are public but haven't been assigned to anyone by IANA. If an as-yet-unassigned address or address range is sending your network traffic, this is also suspicious, and the firewall should discard it.

While the default firewall rules are a good start, you should add a few rules manually to provide a higher level of security. For example, you shouldn't allow services such as *Server Message Block (SMB)*, the service that allows Windows computers to share files across a network, to send or receive outbound or inbound traffic from your network to the internet or receive inbound traffic from the internet.

NOTE *The WannaCry ransomware of May 2017 spread using an SMB vulnerability known as EternalBlue; blocking SMB at your perimeter firewall significantly reduces your risk of exposure to this vulnerability and the risk of other vulnerabilities like it being used to compromise your network.*

To add a rule that blocks SMB traffic, follow these steps:

1. In pfSense, at the top of the page, click **Firewall ▸ Rules**.
2. Click **LAN ▸ Add** to begin adding a rule.
3. Set the action to either **block** (drop the packets) or **reject** (drop the packets and notify the sender).
4. Set Address Family to **IPv4** and Protocol to **TCP**.
5. Set Source to **Any**, Destination to **Any**, and Destination Port Range (to and from) to **(other) 445**.
6. Ensure the **Log** box is ticked to log any dropped packets, and then click **Save**.

Once you're done, your firewall should no longer allow SMB traffic to pass your network boundary. Follow this same process for ports 137, 138, and 139, as these services (NetBIOS Name Resolution, NetBIOS Datagram Service, and NetBIOS Session Service) should never be allowed to cross the network boundary either, as all of these protocols are used for processes internal to a local network.

COMMON PROTOCOLS TO BLOCK

Several network protocols should never cross the network boundary or perimeter, for example:

- NetBIOS Name Resolution, TCP and UDP port 137: Precursor to DNS, resolves hostnames to IP addresses
- NetBIOS Datagram Service, UDP port 138: Enables unicast, multicast, and broadcast messages within a network

- NetBIOS Session Service, TCP port 139: Facilitates communication between two computers on a network
- MS RPC, TCP and UDP port 135: Facilitates communication between client/server applications
- Telnet, TCP port 23: An insecure plaintext protocol used for remote access and maintenance of systems
- SMB, TCP port 445: Allows Windows computers to share files across a network
- SNMP, UDP ports 161 and 162: Used for remote system management and monitoring
- TFTP, TCP and UDP port 69: Enables file transfer between computers on a network

---

#14: Testing Your Firewall

With one or more of these rules in place, test the firewall to ensure the blocked traffic is actually being blocked. The best tool for this purpose is *Nmap*, which is used for network scanning or network mapping. It's available in GUI form on Windows, Linux, and Mac (called *Zenmap*) and also as a command line tool. Installing the GUI version will make it available on the command line, so download and install the latest version from *https://www.nmap.org/*.

Otherwise, you can install it using the command line on Ubuntu:

```
$ sudo apt install nmap
```

Once you've installed Nmap, use the following command to scan port 445 from the command line, which we've told the firewall to block:

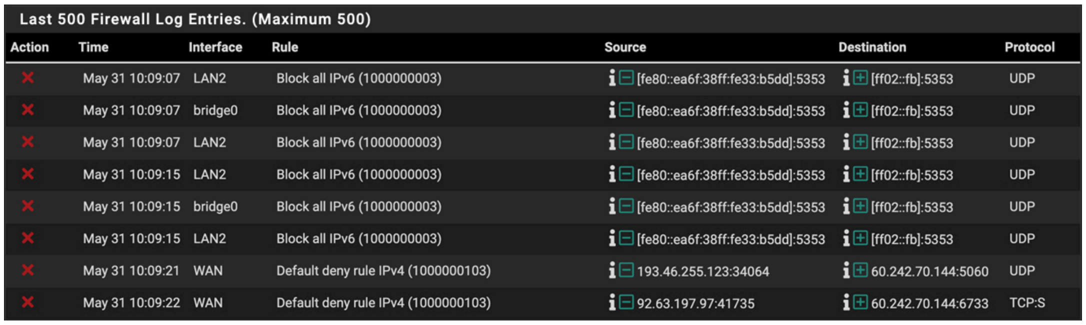
```
$ sudo nmap -p 445 -A scanme.nmap.org

--snip--
```

```
Nmap scan report for scanme.nmap.org (45.33.32.156)

Host is up (0.20s latency).

Other addresses for scanme.nmap.org (not scanned):
2600:3c01::f03c:91ff:fe18:bb2f

PORT    STATE    SERVICE     VERSION

445/tcp filtered microsoft-ds

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/.

Nmap done: 1 IP address (1 host up) scanned in 2.54 seconds
```

You can use the same command in the Zenmap GUI—just exclude `sudo`. This command will perform a port scan from your device, which is behind your firewall, on the ***http://scanme.nmap.org/*** website, a public web page available for testing purposes from the creators of Nmap.

The command breaks down like this: `nmap` is the name of the program. The `-p 445` argument specifies the port or ports to be scanned, which can be either a comma-separated list (such as `-p 445,137,138,22`), a specific port as shown, or a port range like `-p1-1024`. The `-A` argument tells Nmap to try to identify the service and operating system on each scanned port, and `scanme.nmap.org` is the website or system to scan. If the results come back and the `STATE` shown for the port is `filtered`, the firewall has blocked the traffic, and the firewall rules are working. If the `STATE` shows `closed`, the firewall is allowing the traffic through, and the website itself, rather than the firewall, was returning a response saying the port is closed. If you receive this result, your firewall rule either isn't configured or isn't working.

Once your rules are working, go to the firewall logs to see the blocked packets. In pfSense, at the top of the page, click **Status ▸ System Logs ▸ Firewall** to see the last 500 entries in the firewall log, as shown in Figure 3-3.



Last 500 Firewall Log Entries. (Maximum 500)

Action	Time	Interface	Rule	Source	Destination	Protocol
✗	May 31 10:09:07	LAN2	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:07	bridge0	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:07	LAN2	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:15	LAN2	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:15	bridge0	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:15	LAN2	Block all IPv6 (1000000003)	[fe80::ea6f:38ff:fe33:b5dd]:5353	[ff02::fb]:5353	UDP
✗	May 31 10:09:21	WAN	Default deny rule IPv4 (1000000103)	193.46.255.123:34064	60.242.70.144:5060	UDP
✗	May 31 10:09:22	WAN	Default deny rule IPv4 (1000000103)	92.63.197.97:41735	60.242.70.144:6733	TCP:S

Figure 3-3: *pfSense firewall log*

In all likelihood, you'll see a lot of blocked traffic. At this stage, it's difficult to know what this blocked traffic could be. As an example, one of the entries at the top of my log shows a blocked connection from the IP address 80.82.77.245 on port 46732.

Upon further investigation, it appears as though this is a service that performs regular network scans of public IP addresses for "research purposes." That said, it could be anything; how do I know whether this "research" is legitimate or an adversary attempting to find holes in my firewall to penetrate my network? In most cases, it's impossible to know, but at least my firewall is actively blocking this activity, and I can find it in the firewall logs if I need to review it and act on it. We'll discuss what you can do with this information in greater detail in Chapter 10, which covers network security monitoring.

Summary

Your network and hosts are demonstrably more secure for having host- and network-based firewalls in place. In the projects for this chapter, you've created rules and rulesets to make it significantly more difficult for an adversary trying to infiltrate your network, and even more challenging to do so undetected.

While this chapter has armed you with the fundamentals and a greater understanding of firewalls, it’s in your best interest to further research the ports and protocols you’d like to allow or deny within, as well as into and out of, your network. Every network will be different and have different requirements.