# UNIX Security Mechanisms: Section 1 Transcript

## Introduction



Welcome to the UNIX Security Mechanisms module. In this Module we will briefly discuss host-based firewalls on UNIX systems, and then focus on how iptables and firewalld are used to manage firewall rules. Once you understand how iptables and firewalld are structured and intended to work, then you will be able to read and write firewall rules into the appropriate chains and zones. We also give a brief introduction to Snort, a common intrusion detection system (IDS) on UNIX systems, and discuss how to understand an installed Snort configuration.

Throughout this module, you'll be presented with opportunities to assess and apply what you've learned.

At the end of this module, you will be able to:

- Identify tables, chains, and zones in UNIX firewalls, and understand their uses,
- Define UNIX firewall command structures and use them to manage firewall rules,
- Analyze UNIX firewall configurations,
- Define packet filtering, stateful inspection, and application proxy firewall technologies,
- Define Intrusion Detection Systems (IDS), and
- Analyze IDS rules

## Prerequisites



As you are traversing through this module, you are expected to use Internet search engines and UNIX "man" pages to discover and learn about a variety of UNIX commands. Some of the commands you need to research and become proficient with are: iptables, service, ifconfig, and firewall-cmd. Note that some commands are common across UNIX and UNIX-like variants, yet other commands are specific to Solaris, Linux, etc.

Before continuing on, take the time to research the UNIX commands displayed - especially any that may be unfamiliar to you. You may print a copy of the commands (see Resources on the toolbar) and/or use the "Notes" option on the toolbar to record data you learn during your research of each command.

Click the Next Section button when you are ready to continue.

## Bypass Exam Introduction



If you are already familiar with the subject matter presented in this module, you can choose to take a Bypass Exam to skip this module.

The Bypass Exam option provides a single opportunity to successfully demonstrate your competence with the material presented within the module. If you pass, you'll receive credit for completing the module, unlocking the content within, and you will be free to proceed to the next module. If you do not pass, you will need to successfully complete the module, including all

exercises and the Module Exam, to receive credit.

Click the Next Section button to continue.

# UNIX Security Mechanisms: Section 2 Transcript

## Firewalls and Intrusion Detection Systems (IDS)



Firewalls and Intrusion Detection Systems (IDSs) are common security measures used to filter unwanted network traffic and inspect accepted network traffic, respectively. A security solution designed with these technologies is designed to block unauthorized access while permitting authorized communication.

The main function of a firewall is to filter or block inbound and/or outbound traffic according to a configured set of rules. A firewall can be deployed on an endpoint host, in which case it is protecting only that host. A firewall can also be configured to route; therefore, it can filter traffic through the machine, protecting hosts behind the firewall. Many firewalls also perform other functions such as Network Address Translation (NAT) and Virtual Private Network (VPN) termination, though VPN termination is not covered here.

An IDS inspects network traffic, often after it has passed through a firewall, looking for anomalies and attacks. Functionally an IDS is much like a sniffer, paired with an analytic engine that analyzes traffic against both pre-defined signatures (patterns of bytes that appear within network traffic) and heuristic rules (looking for risky behaviors).

## Common Firewall Terms: Packet Filtering



On the next few screens we will discuss three common types of firewalls including packet filter, stateful inspection, and application proxy.

First, packet filtering, also referred to as stateless inspection, is a firewall technology that examines packets individually against matching criteria and makes a pass/do not pass decision on a packet-by-packet basis. Early firewalls used this technology, and it is often still used in network infrastructure devices such as routers, where access control lists (or ACLs) implement packet filtering.

The problem with packet filtering is that it does not consider information about the state of the connection.

- In our diagram, suppose a packet filter firewall is configured to allow hosts on the "inside" (or to the left side of the firewall) to browse the Internet via HTTP.
- The first packet of this exchange will be a TCP SYN packet sent from the inside computer to the web server. The firewall must be configured to allow packets out to TCP destination port 80. This configuration allows the packet to reach the web server.
- The second packet of the exchange is a TCP SYN-ACK packet from the web server back to the inside host. To ensure this SYN-ACK packet can reach the inside computer, the firewall must be configured to allow packets in from TCP source port 80; since our firewall uses only packet filter technology, there is no finer criterion we can use to allow only a narrower set of traffic through.

- Here's the problem: an attacker now has full access to your network! As long as the attacker sets his TCP source port to 80, he can connect to any service in your network. In our example, we see that the hacker is connecting to destination port 3389 on the inside host with source port 80. Now the hacker has access to the host.

## Common Firewall Terms: Stateful Inspection



Let's walk through a few quick examples of stateful inspection. The first is a "normal" session in which the inside host sends a packet with destination port 80 to the web server. The firewall tracks the connection as the packet traverses through the firewall to the web server.

In our next example of a stateful inspection, the web server is sending a SYN_ACK packet back, and then the inside host is sending an ACK, and then SYN_SENT is replaced with ESTABLISHED.

In our third example of stateful inspection we see data packets going back and forth between the inside host and web server. If a hacker attempts to get through, sending a TCP SYN with source port 80, the firewall will inspect the packet and determine "No session established." The firewall will then drop the packet.

## Common Firewall Terms: Application Proxies



Finally, let's briefly discuss the third common type of firewall, Application Proxy.

Further generations of firewalls have added additional filtering capabilities, with the modern buzzword being "deep packet inspection." One example of this is an **application proxy**, which examines the content of the communication and proxies the connection to the intended destination, so that all traffic passes through the proxy. This eliminates the possibility of a "covert channel," where an attacker can hide a connection so that it looks like web traffic at the transport layer, but the actual data is not HTTP.

## Common UNIX Firewalls



Let's discuss a few common UNIX firewalls.

**Sunscreen** is the original Solaris firewall. It was available in Solaris 8, and bundled with Solaris 9. It provided a full suite of capabilities including stateful inspection, NAT, and VPN termination.

**IP Filter** (or ipf) is the new default firewall of Solaris, as of Solaris 10. The functionality is substantially similar to sunscreen.

Next, **iptables** is the most common firewall deployed with Linux distributions. It performs stateful inspection and NAT, but not VPN termination.

A proposed replacement for iptables is **firewalld**, which is very popular in the RedHat/Fedora Linux community; however, firewalld does not seem to be having much market success outside of this community.

Lastly, **nftables** is the up and coming successor to iptables.

Conceptually, and often functionally, these firewalls work pretty much the same way, so for this module we will predominantly focus on iptables.

## Introduction to iptables: Tables and Chains



As mentioned, the iptables package offers stateful and stateless firewall filtering on UNIX hosts, with an implementation that sets up and inspects IP packet filter rules using the concepts of tables and chains.

In iptables there are 5 tables: **filter**, **nat**, **mangle**, **raw**, and **security**, and each table has a specific purpose and manipulates traffic in a particular way. For example, the filter table is specifically designed to filter packets, while the nat table is specifically designed to rewrite addresses.

Tables contain chains. A chain contains a set or list, of rules that are applied on packets that traverse the chain. Any packet that matches a pattern will have an action applied to it. Each chain has a specific purpose (for instance, which table it is connected to, which specifies what this chain is able to do), as well as having a specific application area (for instance, only forwarded packets, or only packets destined for this host). Note that you can also have user-defined chains, which are useful, by providing alternative processing paths for certain types of traffic.

## The Filter Table in Detail



Let's take a moment to briefly define the five tables in iptables: filter, nat, mangle, raw, and security. Note that each iptables table contains one or more chain, each of which defines a set of rules to be processed. Processing may jump from one chain to another, depending on how the rules are written.

Click the images displayed to learn about each of the five tables in iptables, along with their functions.

The **filter** table is primarily responsible for filtering traffic, or in other words, making pass/drop decisions for accepting or rejecting traffic. This is the default table when using the iptables command, and it contains the built-in chains INPUT (for packets destined to the local system), FORWARD (for packets being routed through the system), and OUTPUT (for local system generated packets).

The **nat** table implements network address translation. This table is consulted when the system encounters a new connection packet. It consists of three built-in chains PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering local system packets before they are routed), and POSTROUTING (for altering packets as they are departing the system).

The **mangle** table implements specialized packet rewriting rules, primarily for IP quality of service. This table is used for specialized packet alterations such as INPUT (for packets that are coming in to the system), FORWARD (for altering packets that are being routed through the system), and POSTROUTING (for altering packets as they are about to go out of the system).

The **raw** table is rarely used, but it can circumvent the kernel's connection tracking mechanism. The raw table contains PREROUTING (for packets arriving via any network interface) and OUTPUT chains (for packets generated by local processes).

The **security** table is also rarely used, but it can be used to implement Mandatory Access Control networking rules. The security table contains INPUT (for packets destined to the local system), FORWARD (for packets being routed through the system), and OUTPUT (for local system generated packets).

## The Filter Table in Detail



When using the default filter table, iptables is fundamentally making a pass/do-not-pass decision for each packet, applying both packet filtering and stateful inspection logic. Let's follow the processing path for a packet after it has been received by the iptables host and then de-encapsulated from its Ethernet framing.

Upon receipt of the packet, the iptables host examines the destination IP address. If the destination IP address is assigned to the iptables host, then the packet will be analyzed against the rules in the INPUT chain. If the destination IP address is not assigned to the iptables host, and that host is not configured to route packets, the packet will be discarded. If the destination IP address is not assigned to the iptables host, and that host is configured to route packets, then the packet will be analyzed against the rules in the FORWARD chain.

Regardless of which chain is applied, the end result is to pass the packet (generally using the target ACCEPT) or not to pass the packet (usually implemented with one of the targets REJECT or DROP).

If the packet is passed by the INPUT chain, the packet goes to the network stack of the iptables host for further de-encapsulation and processing. If the packet is accepted by the FORWARD chain, the destination IP address is compared to the routing table of the iptables host, and routed accordingly. In either case, iptables has no more involvement with packet processing.

## The Filter Table in Detail



Note that iptables can also be used to inspect packets that originate on the iptables host. In this case, the packet is analyzed against the rules in the OUTPUT chain, which makes a pass/do-not-pass decision. If a pass decision is made, the host consults the routing table and forwards the packet accordingly.

It is possible to create user-defined chains, but they will not be used for processing unless they are specifically referred to as a target for a rule in one of the three default chains of the filter table.

Each of the default chains in the filter table has an associated "default policy," which defines what the default decision should be if a packet does not match any of the rules in the chain. The choices for this policy are ACCEPT or DROP.

## The Filter Table in Detail



Here is an example of the default iptables filter table. We could have typed `iptables -t filter -L`; however, the `-t filter` is not necessary as this is the default. This diagram illustrates a fairly typical default iptables filter table installation that comes with Linux CentOS 6.

We will talk more about the specifics of the iptables rules shortly, but roughly we can see that this machine will not route any packets, since the FORWARD chain rejects all packets.

Also, the machine can send any packets it wants, but the only incoming packets it can accept are packets belonging to existing connections, ICMP packets, and incoming connections via SSH.

## The NAT Table in Detail



Network Address Translation (NAT) is used for rules that translate source and destination IP addresses into other network addresses, and can be applied to IP packets incoming, outgoing, or transiting the iptables host. You can also use NAT to redirect traffic from one port to another port. NAT maintains the integrity of the packet during any modification or redirection done on the packet, altering header checksums as necessary.

As shown in the diagram, the NAT table interacts very closely with the Filter table to affect packet forwarding.

The PREROUTING chain is generally used for Destination NAT (DNAT), where the iptables host is changing the destination IP address of the host, as well as for port forwarding, where the iptables host needs to translate the destination TCP/UDP port to a different value.

The OUTPUT and POSTROUTING chains are generally used for Source NAT (SNAT), with OUTPUT usually used for packets originating from the iptables host, and POSTROUTING used for packets transiting the iptables host. Later in this module, we will study the NAT table more through an exercise.

## The iptables Command Structure



Let's discuss the general structure of an iptables invocation: iptables [options] [action] [matching rules] [target]. Note that the items must be in this order!

A common option you may want to use is the **-t [table]** to define, or specify, the table to which the command should be applied. This option allows the user to select a table, other than the default filter table, to use with the command. Default is the filter table if left blank.

Some common actions include listing firewall rules; adding, inserting, replacing, and deleting rules; and finally setting the default policy.

## Your Current Ruleset



In this module we will cover a few important iptables commands and options. This is not an exhaustive list, but we will discuss several of the most common commands.

To list all of the rules in the chain specified after the command you would use the `-L` (or `--list`) command. If you would like to list all rules in all chains in the default filter table, then simply do not specify a chain or table; if no chain is selected, all chains are listed.

Verbose listing, or `-v` gives usage statistics with each rule.

Another important listing option is `-n`, for numeric output. You would use this option to obtain a list of firewall entries without doing name resolution. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display the information as host names, network names, or services (whenever applicable).

Lastly, you would use `--line-numbers` to show rule numbers next to each firewall rule. When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in the chain. This is important and should be used when determining where to insert or replace a rule.

Let's discuss how to run and view your current rule set. Before we get started, search the man pages for iptables.

Next, run iptables `-L -n -t [table] -v` to view the current rule set. The default table is the filter table.

One feature this example illustrates is the use of user-defined chains. In this case, the RH-Firewall-1-INPUT chain is created and referenced in both the INPUT and FORWARD chains, allowing the same ruleset to apply to both chains without the need to duplicate the ruleset.

## Exercise Introduction



It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

# UNIX Security Mechanisms: Section 3 Transcript

## Matching Options



The following parameters make up a rule specification (as used in the previously described add, delete, insert, replace, and append commands).

Filtering requires two items, specifically criteria to match and an action to take. Let's discuss a few examples of matching criteria, noting that this is not a complete list. Each of these criteria consists of first a standard UNIX option, such as `-p`, which defines what attribute of the inspected packet should be compared and second an argument to that option which defines the value to which to compare the packet attribute.

Multiple criteria are combined with a logical "and" implying that a packet must match all criteria for a rule to apply. Note that criteria can also be negated, so that the criterion is true only if there is not a match; this is indicated with an exclamation point (!) or bang sign before the option. Be aware that the exclamation point "**!** " is not a shell-safe character, so you will need to escape it. Next we will review a few of the most common rules.

## Matching Options (Continued)



First, **-p** PROTOCOL, refers to the protocol of the inspected packet as defined by the protocol field in the IP header. The specified protocol can be one of TCP, UDP, ICMP, or all. Alternatively, it can be a numeric value between 0 and 255 representing one of these protocols or a different one; for example, **-p tcp** and **-p 6** are synonymous.

Next, **-s** ADDRESS, refers to the source IP address of the inspected packet. Related is the **-d** ADDRESS, which refers to the destination address. For both of these criteria, the ADDRESS can be a fully-qualified domain name (FQDN), a hostname (as defined in the /etc/hosts file), or a dotted decimal IP address.

Finally, we have **-i** INTERFACE (inbound) and **-o** INTERFACE (outbound). These commands evaluate on which local interface the inspected packet was found (inbound or outbound) with respect to the host. The INTERFACE is an interface designation listed in the output of **ifconfig -a**, and will typically be named something in the order of xxx#, where xxx is "eth", "p2p1", or "vxnet"; and # is a single digit.

Let's look at an example rule:

```
iptables -A INPUT -d 192.168.11.13 \! -i eth0 -j DROP
```

There are two matching criteria here: `-d 192.168.11.13` requires the packet to have destination IP address 192.168.11.13. The second `\! -i eth0` requires that the packet be incoming on an interface different from `eth0`.

Combining these with a logical "and" we can interpret this as matching all packets with destination

IP address 192.168.11.13 that arrive on an interface other than `eth0`. Though we have not yet covered this, `-j DROP` instructs iptables to drop matching packets.

## Matching Options



One of the reasons for the popularity of iptables is its flexibility. There are numerous advanced and customized matching criteria that can be used to filter packets on a very granular basis. Specifically, advanced criteria are usually loaded using the **-m** or **-p** options.

A common example of this feature is the protocol specific extensions for TCP, UDP, and ICMP. With **-p tcp** or **-p udp**, you can use the options **--sport** and **--dport** to match source and destination ports. The TCP extension also offers options to match the various flags (such as SYN and ACK), as well as other particular options. As another example, **-p icmp**, allows use of **--icmp-type** matcher to look for specific ICMP message types.

There are numerous other matching modules, which can match on MAC address, IP quality of service flags, and even packet statistics such as number of packets per second. Stateful matching is a critically useful matching criterion, which we discuss next.

## Stateful Matching



State is a Match Extension. States are mainly used in conjunction with the state match which will then be able to match packets based on their current connection tracking state. Note that connection tracking is a UNIX kernel feature that is only relevant for stateful protocols such as TCP. The valid states are **NEW**, **ESTABLISHED**, **RELATED**, and **INVALID** where state is a comma separated list of the connection states to match. Let's define the possible states.

The **NEW** state means that the packet is starting a new connection and tells us this is the first packet that we see. In other words, the packet is not otherwise associated with a connection. This may be useful when we need to pick up lost connections from other firewalls or when a connection has already timed out, but in reality is not closed. In a TCP context, you can expect a TCP SYN packet to be NEW.

**ESTABLISHED** means that the packet is associated with a connection, which has seen packets in both directions, and will continuously match those packets. ESTABLISHED connections are fairly easy to understand. The only requirement for an ESTABLISHED state is that one host has sent a packet, and received a reply from the other host. Note that the NEW state will change to the ESTABLISHED state upon receipt of the reply packet to or through the firewall. In a TCP context, packets subsequent to the initial SYN and SYN-ACK are expected to be in the ESTABLISHED state.

A connection is considered **RELATED** when it is related to another already ESTABLISHED connection. In order for a connection to be considered RELATED, we must first have a connection that is considered ESTABLISHED. The ESTABLISHED connection will then spawn a connection outside of the main connection; a typical example of this is File Transfer Protocol (FTP), where the ESTABLISHED connection is the FTP control channel, and the RELATED connection would be a separate data channel that is opened to transfer a file. The newly spawned connection will then be considered RELATED. Simply stated, RELATED means that the packet is starting a new

connection, but is associated with an existing connection, such as an FTP data transfer or an ICMP error.

**INVALID** means that the packet could not be identified for some reason, which includes running out of memory and ICMP errors that do not correspond to any known connection. This category also includes TCP packets where the flags do not correspond to the connection state, such as a SYN-ACK packet when no SYN packet has been sent. Generally, it's a good idea to DROP everything in this state.

These states can be used with the **-m state** syntax (which loads the matcher), as part of the **--state** match option to match packets based on their connection-tracking state. A typical example seen in machines that allow outbound connections is a rule of the form: `iptables -A INPUT -p tcp -m state --state ESTABLISHED, RELATED -j ACCEPT`. This rule accepts any incoming TCP packet that is part of an ESTABLISHED or RELATED connection, but does not allow NEW connections to be originated by a machine outside the host attempting to connect in. In a TCP context you can expect a TCP syntax to be new.

## Stateful Matching



Let's take a closer look at the states and how they are handled for TCP and UDP. We will start out with the TCP protocol since it is a stateful protocol in itself, and it provides a lot of interesting details with regard to the state machine in iptables.

A TCP connection is always initiated with the 3-way handshake, which establishes and negotiates the actual connection over which data will be sent. The whole session begins with a SYN packet, then a SYN/ACK packet, and finally an ACK packet to acknowledge the whole session establishment. At this point, the connection is established and able to start sending data.

Connection tracking works basically the same for all connection types. In this example, notice the state as the stream proceeds through the different stages of the connection. The connection tracking code does not really follow the flow of the TCP connection, from the user's viewpoint.

Once it sees one packet (the SYN), the firewall considers the connection as NEW. When it sees the return packet (SYN/ACK), it considers the connection as ESTABLISHED. You can allow NEW and ESTABLISHED packets to leave your local network, but only allow ESTABLISHED connections back.

## Stateful Matching



Now let's look at a UDP example. UDP connections are inherently not stateful connections, but rather stateless. There are several reasons why, mainly because they don't contain any connection establishment or connection closing; furthermore, UDP connections lack sequencing.

Receiving two UDP datagrams in a specific order does not say anything about the order in which they were sent; however, the kernel connection tracking feature analyzes an entire flow of UDP packets to provide a reasonable analog of connection states for UDP communications. Note that iptables can take advantage of this capability to perform stateful matching against UDP streams.

## Filter Targets



Targets specify an action to take on the packet in question. In other words, the target tells the rule what to do with a packet that is a perfect match with the match section of the rule. Let's review a few basic targets, or actions.

**ACCEPT**, simply stated, is used to let the packet through. As soon as the match specification for a packet has been fully satisfied, and we specify ACCEPT as the target, the rule is accepted and will not continue traversing the current chain or any other chain in the same table. Note however, that a packet that was accepted in one chain might still travel through chains within other tables. To use this target, we simply specify **-j ACCEPT**.

The **DROP** target does just what it says, it silently drops packets dead on the floor and will not carry out any further processing. A packet that matches a rule perfectly and is then dropped will be blocked.

The **REJECT** target works basically the same as the DROP target, but it also sends back an error message to the host sending the packet that was blocked.

The **LOG** target is specially designed for logging detailed information about packets and can be used for bug hunting and error finding. The LOG target will return specific information on packets, such as most of the IP headers and other information considered interesting. Note that LOG does not cease processing of further rules; the packet continues being evaluated against later rules and may be passed or not depending on those rules.

**CHAIN** is used to continue processing the packet in this user-defined chain.

**RETURN** is used to stop the packet from traversing this chain and resume at the next rule in the previous calling chain.

## NAT Targets



The NAT table primarily uses three targets: SNAT, DNAT, and MASQUERADE.

The **SNAT** target is used to do Source Network Address Translation, which means that this target will rewrite the Source IP address in the IP header of the packet. This is usually performed in the POSTROUTING and OUTPUT chains, just before the packet is put onto the wire.

The **DNAT** target is used to do Destination Network Address Translation, which means that it is used to rewrite the Destination IP address of a packet. The DNAT target is usually used in the PREROUTING chain, before a routing decision is made, so that the translated address can be used to route the packet correctly.

The **MASQUERADE** target allows iptables to implement port address translation. Often used with the **-i** and **-o** interface criteria, the MASQUERADE target causes an outbound TCP or UDP packet to have both the source IP address and source port rewritten, with the source IP address changed to that of the iptables firewall, and the source port used to track the originating host. When a return packet is received, the destination port (which was the source port of the originating packet) is used

by the firewall to determine how to rewrite the incoming destination IP address and destination port. This is usually performed in the POSTROUTING and OUTPUT chains.

You will explore writing NAT rules to perform these functions in the exercises.

## Default Policy



In the case that a packet is not matched by any rules, there is a way to set up a default policy to a certain table or chain. This default policy configures a target to be executed in the case no other terminating rule on the same target or table match the traffic.

Each **CHAIN** has a default policy, and **-P** sets the default policy. The default is set to ACCEPT. DROP is the only other option.

In this example we see iptables **-P INPUT DROP**

What table does this policy apply to? Only the filter table.

## Flushing Firewall Rules



When working with iptables, it is often useful to get the rules back to a known good state. The following commands allow you to do this:

`iptables -t filter -F` (Flush all rules from the filter table)

`iptables -t nat -F` (Flush all rules from the NAT table)

`iptables -X` (Remove all user-defined chains from the filter table)

`iptables -P INPUT ACCEPT` (Set the default policy on the INPUT chain of the filter table)

`iptables -P OUTPUT ACCEPT` (Set the default policy on the OUTPUT chain of the filter table)

`iptables -P FORWARD ACCEPT` (Set the default policy on the FORWARD chain of the filter table)

## Writing Rules



Let's review the concepts we have just introduced. Using the iptables command we append a rule to a chain using the following syntax: `iptables -t [TABLE] -A [CHAIN][RULES] -j [TARGET]`

- Default table is filter
- Must identify a CHAIN
- Rules can be in ANY order
- The `-j [TARGET]` MUST be at the end
- Using a "!" (NOT) in front of the elements negates that element when evaluated by the system

Let's apply what we have learned using a few examples. In this example we allow the network

192.168.0.0/24 connected through the network interface named eth0, to send traffic to the network 192.168.1.0/24, which is connected to the network interface eth1. Note that the `-t filter` could be omitted.

You can practice by adding rules that will allow loopback to communicate freely using the following:

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

iptables -A OUTPUT -o lo -j ACCEPT
```

When the rule is evaluated, the system looks at it as having a logical "AND" between each criteria, which means that each part of the rule must be matched for the rule to get a hit.

## Writing Rules (Continued)



As another example, suppose we wish to write a rule that restricts access to the firewall's SSH port to only hosts on the subnet 10.18.80.64/26. Assume that our default policy on all filter table chains is to DROP traffic.

Since we are trying to restrict traffic whose destination is the firewall itself, we need to use the INPUT chain.

The traffic we are trying to restrict is SSH traffic, which will be directed to TCP port 22. Hence, we need to use the -p option to match TCP, and the --dport option to match port 22.

We want to allow traffic from the 10.18.80.64/26 network, and deny traffic from other hosts. Since the default policy is to drop traffic, we need to explicitly allow traffic from the source network. Thus we want to match using the -s option.

This defines all of our matching criteria, and we finally need to define a target. In this case, we want to ACCEPT this traffic.

Will SSH work? NO! The default policy on all chains in the filter table is to DROP, which means that while a packet from our allowed network to our firewall on port 22 is allowed, the return packet is not. This rule needs to be paired with the following rule to allow SSH from this network.

## Writing Stateful Rules



Let's try another approach using stateful matching. Recall that in the previous example, we flatly accepted all inbound TCP packets to port 22 from the 10.18.80.64/26 network. This is fine for a private network, but as we have discussed before rules like this can be dangerous in public situations. Now we will rewrite that example using stateful matching.

We certainly want to retain all of our existing matching criteria. But in addition, we need to think about what sorts of TCP packets we want to allow. Since hosts in the 10.18.80.64/26 network will be initiating connections into our firewall, we certainly need to allow NEW connections. If we want the connection to continue, we have to allow ESTABLISHED and RELATED packets as well, so to create a typical stateful match criteria we would add the following.

For the OUTPUT chain, we do not expect that the firewall should ever be initiating connections in this situation, so we should be able to allow just ESTABLISHED and RELATED by adding the following.

In this example, the rule says that any forwarding traffic RELATED to an ESTABLISHED, or already existing, connection will be allowed.

## Knowledge Check Introduction



It is time for a Knowledge Check. This Knowledge Check will not be scored, but may indicate areas that you need to review prior to the Module Exam.

# UNIX Security Mechanisms: Section 4 Transcript

## Introduction to firewalld



The iptables command has been the defacto method of firewall management for Linux systems since around the year 2000. However, with the advent of the new systemd initialization method over the past several years, many UNIX industry leaders have chosen a new method of firewall management named firewalld.

While it has not yet been adopted by all Linux distribution developers, many of the leading distributions, such as Red Hat Enterprise Linux (RHEL) 7, CentOS 7, Fedora 18+, and OpenSUSE 15+ implement firewalld today.

Firewalld aims to simplify and abstract many of the complexities involved with iptables. However, it also introduces some of its own complexities.

## The relationship between iptables and firewalld



Like iptables, firewalld is a method that Linux administrators can use to interact with the Linux network filtering subsystem, called netfilter. The netfilter subsystem allows kernel modules to inspect, modify, drop, or reject packets that are incoming, outgoing, or routing through a Linux system.

Both iptables and firewalld use the ip_tables kernel module to interact with netfilter, as displayed on screen. Although the command syntax, capabilities, and features are different between the two programs, the firewall rule implementation is the same at the kernel level.

A key differentiator between the two is that firewalld includes a layer of abstraction on top of the typical iptables firewall rules. This feature was intended to make adding firewall rules with firewalld more straightforward and intuitive than the previous approach taken by iptables.

The firewalld daemon runs as a systemd service daemon and can be used to configure firewall rules on a system via the `firewall-cmd` command, a command line tool used to manage the firewalld daemon.

Let's discuss the functionality of firewalld in more detail.

## Firewalld Zones



One of the ways firewalld simplifies firewall management is by first grouping all network traffic into one of nine pre-defined zones, based on either the source IP address of the incoming packet or the NIC that receives the incoming packet.

Depending on the zone to which firewalld diverts the packet, the packet is matched against a list of firewall rules in the same manner as iptables. Each zone has an individual list of rules. The rules within a zone are, effectively, no different than the rules within an iptables chain. In fact, under the

hood, firewalld includes the same tables and chains with which you are familiar.

In summary, the firewalld zones allow an administrator to assign a level of "trust" to a network interface.

Displayed are the nine firewalld zones and their associated details.

## Packet Analysis- Source Address



Let's discuss the packet analysis order of operations in a little more detail.

Each packet that enters the system is first checked for its source address. If that source address matches the criteria of a particular zone, the packet will be sent to that zone and parsed.

In this example, the 'work' zone has a source address specified within it. The IP address of the incoming packet matches the 'work' zone address, which is why that zone is parsed.

## Packet Analysis – Incoming Interface



If the source address does *not* match a zone, then the zone that matches the interface the packet came in on will be parsed.

In this example, the interface of the incoming packet matches the one configured in the 'trusted' zone, so that is the zone that gets parsed.

## Packet Analysis – Incoming Interface



If neither the source address *nor* the interface the packet came in on match a current zone, then the default zone will be parsed.

Note that while the 'public zone' is typically the established default zone, this can be changed and customized by an administrator.

In summary, the firewalld zone that applies to each packet is determined by matching the source address, incoming interface, and then the default zone - in that order.

Displayed is an example of the 'public' zone.

## firewall-cmd



The old format of tables and chains can still be viewed by using the iptables command: `iptables -L -n -v`. However, `firewall-cmd` command abstracts out the tables and chains to provide a more straightforward method of managing the firewall. Viewing the firewall rules with the `firewall-cmd` command yields different results compared to the iptables command.

Displayed is an example that briefly highlights some of the formatting differences between iptables and firewalld. This example is taken from a CentOS7 system running firewalld.

## Firewalld Configuration Locations



Looking at the firewalld configuration locations, the `/usr/lib/firewalld` directory contains the default configuration for firewalld. The system, or user defined firewalld configuration files, are located within the `/etc/firewalld` directory.

However, if no user defined configurations have been made, then there may not be any XML configuration files within the `/etc/firewalld` directory or its sub-directories.

If the user has made defined configurations to firewalld, the associated configuration file is named `firewalld.conf`.

Amongst other configurations, you can create custom, user defined services and zones which are typically stored in XML files within `/etc/firewalld/services` or `/etc/firewalld/zones` respectively. This is an example of a zone XML configuration file.

While the firewalld rules can actually be managed by editing the XML files contained within the directory, the `firewall-cmd` command is easier to use.

## Runtime vs. Permanent Configuration



Continuing with the firewalld configuration, firewalld is separated into a runtime and a permanent configuration.

The runtime configuration is the active and effective configuration in memory. However, all runtime settings will be lost if the firewalld service is stopped or the system is rebooted. The permanent configuration is read and implemented when the service starts or when reloaded via the `firewall-cmd --reload` command.

You can modify both the permanent and runtime configurations using the `firewall-cmd` command although you must add the `--permanent` option if you want to affect the permanent configuration.

Unlike iptables, you can change the configuration of the firewall and reload it into the runtime configuration with zero downtime or interruptions to established connections using the `firewall-cmd` command.

Please note, reloading will remove all runtime configurations and replace them with the permanent configuration.

## The firewall-cmd Command



`firewall-cmd` is installed as part of the firewalld package and has well written, detailed, and easy to interpret documentation and manual pages. Almost all `firewall-cmd options`, by default, work on runtime configuration, unless you specifically add the `--permanent` option to make the changes persistent. To view the current rules for all zones on a system, use the displayed command (`firewall-cmd --list-all-zones`.)

On most systems that have firewalld, the firewall-cmd command supports tab completion. Meaning, you can type `firewall-cmd --` and then hit the TAB key twice to get a list of possible options.

Let's briefly review a few examples of how you can use the `firewall-cmd` command to modify the firewall.

This command sets the default zone to the dmz zone.

In this example, we are configuring firewalld to route all traffic from the displayed network (192.168.1.0/24) to the internal zone.

This command allows MySQL traffic into the internal zone.

And finally, the displayed table includes a short list of some of the more commonly used firewall-cmd options. Take a moment to familiarize yourself with each option and its purpose.

## The firewalld daemon



By default, the firewalld daemon is enabled and started via systemd. The daemon allows dynamic management of the firewall and provides support for the firewall zones.

The daemon also enables the separation of the runtime and permanent configurations, discussed earlier.

Additionally, the firewalld daemon allows applications and services to add their own firewall rules directly.

## Rich and Direct Syntax



If the basic syntax for `firewall-cmd` won't address your needs, you can use rich or direct syntax, which are more expressive than the `firewall-cmd` syntax and can handle complex rules. To learn more about these advanced firewalld configuration options review the "Firewalld Rich Syntax" and the "Firewalld Direct Syntax" resources for this module.

To learn more about firewalld, please review the displayed man pages.

```
firewall-cmd(1)
firewall-config(1)
firewalld(1)
firewalld.zone(5)
firewalld.richlanguage(5)
firewalld.service(5)
firewalld.direct(5)
firewalld.zones(5)
```

## Exercise Introduction



It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

# UNIX Security Mechanisms: Section 5 Transcript

## IDS Tools



Intrusion Detection Systems (IDS) is another common security technology that can be implemented with UNIX. There are multiple different types of IDS:

1. Network-based IDS analyze network traffic. Snort is an example of a Network-based IDS.

2. Host-based IDS monitor system activity and state. Several examples of Host-based IDS include Tripwire, the Open Source Host-based Intrusion Detection System (OSSEC), and Basic Audit Reporting Tool (BART). OSSEC performs log analysis, file integrity checking, policy monitoring, rootkit detection, and real-time alerting. BART can detect alterations to files and file metadata and is available on Solaris 10 and newer.

3. Security scanners are used to detect vulnerabilities. Nessus and OpenVAS are two examples of security scanner IDS.

Note that with IDS, detection can be statistical or signature-based. In this section we discuss Snort and Tripwire, IDS tools that are most commonly associated with UNIX operating systems.

## Tripwire



Following the UNIX philosophy of doing one thing well, the original Tripwire tool worked very simply. For all critical system files which should not change on a regular basis, Tripwire calculated a cryptographic hash and stored those hashes in a secure location. If an attacker were successful in penetrating the machine, an administrator could run Tripwire again and compare the list of hashes. With high probability, any change to one of these files would cause its hash to change so Tripwire very easily detected modified system files.

Tripwire, the company that markets the original tool, has grown much bigger than the original tool. But the original Tripwire technique should definitely make you think twice before changing any system files if Tripwire is installed.

## Snort Rules



Unlike iptables, where learning how to read and write rules are critical skills, our interest in Snort is mostly in analyzing rule sets that might apply in a target network. Snort is rather complex software, and we are only providing an introduction here.

There are two key informational elements you need in order to analyze a Snort installation. The first is the **snort.conf** file, usually stored in **/etc/snort/snort.conf**. This file contains many configurations, but the most important for our purposes are the rule definitions.

As you can see from the listing, there are many different rulesets that could be applied in a Snort installation, but often most of these rules are not active. To be active, there needs to be a line in the

snort.conf file of the form "include $RULE_PATH/<name>/rules". The only rules that are active are those referred to in the snort.conf file.

Other important settings in the snort.conf file are the $EXTERNAL_NET and $INTERNAL_NET variables. These variables may be used to define two networks of interest that are generally regarded as "the wild" (which is, $EXTERNAL_NET) and "our stuff" (which is,$INTERNAL_NET).

Once you know which rules are active, you can look in the $RULE_PATH directory, usually /etc/snort/rules, to examine the applicable rule sets.

## Reading Snort Rules



A single Snort rule generally occupies a single line of a rule set file. The rule has three main sections:

**Action:** Each rule starts with an action that defines what Snort will do if the rule matches. Typical actions are alert (which generates an alert using a system-wide defined method), log (which logs the packet, but otherwise takes no action), and pass (which simply does nothing).

**Network/transport layer information:** The second part of each rule is a specification of the traffic to which the rule applies, from a network and transport layer perspective. This specification contains six items:

- Protocol: specifies the protocol of the traffic. TCP, UDP, ICMP, and IP are allowed
- Source IP: specifies the source IP information, be it a single host, or a network defined using CIDR notation, or "any" for any source
- Source port: may be a single port number, a range of ports (limits separated by a colon), or "any"
- Direction: May be either "->" indicating the rule should match only one-way traffic, or "<>" indicating bi-direction matching
- Destination IP just like source IP
- Destination port is just like source port

Note that all IP and port values or ranges can be negated with the NOT operator.

**Rule options:** The real power of Snort is its ability to dig into the application information of the inspected packets, which is configured with rule options. These rule options provide a wide variety of characteristics that Snort can examine, including both non-payload and payload characteristics.

Some common non-payload characteristics are:

- Flags which matches TCP flags
- Flow which is similar to iptables state matching and detects established versus new connections, and
- TTL the time-to-live of the packet

Some common payload matching criteria are:

- Content: Many Snort rules contain a content matcher, which examines the packet looking for

the associated string. For example, content:"your text here" will inspect packets looking for this string. Binary matches are usually given as a hexdump enclosed in pipes, for example content:"|00 45 ab dc ef|". This is very typical for performing signature-based analysis for known malware.
- Depth: the depth option limits how far into a packet to look for a content string.
- HTTP elements: There are a variety of options that limit the content matcher to specific segments of an HTTP message, such as http_client_body and http_cookie.

There are many, many more options to Snort than we can cover here. The website manual.snort.org provides a much more comprehensive guide to the elements you may see.

## Exercise Introduction



It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

# UNIX Security Mechanisms: Section 6 Transcript

## Summary



You have completed the UNIX Security Mechanisms module. In this module, we briefly discussed host-based firewalls on UNIX systems and how iptables is used to manage firewall rules. We discussed how iptables and firewalld are structured and how to read and write firewall rules into the appropriate chains and zones. We also provided a brief introduction to Snort and reviewed an installed Snort configuration. In addition to these topics, as you traversed through this module, you were expected to conduct research to learn about a variety of UNIX commands. You should now be able to:

- Identify tables, chains, and zones in UNIX firewalls, and understand their uses,
- Define UNIX firewall command structures and use them to manage firewall rules,
- Analyze UNIX firewall configurations,
- Define packet filtering, stateful inspection, and application proxy firewall technologies,
- Define Intrusion Detection Systems (IDS), and
- Analyze IDS rules

To receive credit and advance to the next module, you must achieve a passing score on the Module Exam. Click the Next Section button to begin the Module Exam.