CCNA Exam Prep: Back to Networking Basics with Hank Preston and Patrick Gargano

Fundamentals of ACLs

Access Control Lists, or ACLs, have been core to network security topics for CCNA candidates since the beginning. If you want to control what traffic is allowed to pass through an interface, ACLs are a go-to technology. But ACLs are used for far more than just simple interface filters. Access Control Lists, and the Access Control Entries (ACE) that make them up, provide network engineers with a powerful traffic matching language that can be used for everything from NAT to routing policies.

As a CCNA candidate, you must be comfortable using ACLs in your routers and switch configurations.

> 5.6 Configure and verify access control lists

In this lab, we'll start our exploration of ACLS with the exercises focused on:

- Standard and Extended ACLs with numbers and names
- Matching source and destination hosts and networks
- The impact of inbound and outbound filtering with ACLs

Setup and Scenario

This set of lab based demonstrations includes a small network made up of "Clients" and "Services" network locations. There are 2 client hosts that you'll create ACL policies to control where their traffic can reach. For services, there is a `server` that hosts a web page (`http://server` or `http://172.16.0.65`). Thre is also a "simulated internet" that hosts 3 simulated web locations, `internet100`, `internet200` and `internet210`.

Each exercise is about a "security policy" that you'll need to configure and apply ACLs to achieve. The exercises are designed to have you move through both Standard and Extended ACLs, so be sure to review the details at the start of each exercise before getting started.

> Note: The credentials for all devices and hosts are `cisco / cisco`.

*Be sure to **START** the lab before continuing to the demo labs.*

Pre-validation

The start state of the lab has the router and all hosts configured for IP connectivity end to end. Before we begin controlling access using ACLs, verify that traffic is flowing everywhere without problems.

Client to Server Access

Both client hosts should be able to communciate with the `server`. Open up a VNC connection from each `client`, launch Firefox from the "Applications > Internet" menu, and navigate to "**http://server**". You could also navigate to the IP address of the server with "**http://172.16.0.65**"

> Note: Don't use the "Web Browser" option from the menu or the launchbar.

If the VNC desktop view isn't performing well, you can also use a console connection and `ping server` to verify connectivity.

Client to Simulated Internet Access

Now verify that both clients can reach the "Internet" by navigating to "**http://internet100**", "**http://internet200**" and "**http://internet210**".

Alternatively, verify that both clients can ping the two simulated internet hosts.

```
ping -c 5 internet100
ping -c 5 internet200
ping -c 5 internet210
```

Server to Simulated Internet Access

The `server` is an Nginx container hosting the web page. The Console connection defaults to "Serial line 0", which shows the logs from the Nginx server. The interactive console connection is on "Serial line 1". To access "Serial line 1", right click on the tab for the `>_SERVER` after opening the console, choose "Change Serial" and select "Serial line 1". You should then see a prompt such as this:

```
root@276921a33475:/#
```

Once you have the interactive console connection, verify simulated internet access by pinging the two hosts from the the last test.

```
ping -c 5 internet100
ping -c 5 internet200
ping -c 5 internet210
```

Standard Access Control Lists

Standard Access Control Lists are the simpler type of ACLs available, supporting a only source IP address matching. In the following exercises, we will explore Standard ACLs by implementing 4 different security policies.

Consider these exercises as an evolving security policy for the network. Which means you may need to change or remove configurations used to complete a prior policy as you move forward. This evoluation of policy, and the changes needed in the configuration are done intentially to explore how Standard ACLs work on a network.

Policy 1: Only Clients 1 and 2 can talk to the Server

The first policy is pretty straight forward, we want to make sure that only Clients 1 and 2 on the "Clients" network can talk to the server. We have been told that we must use a numbered standard ACL, and apply it to the `router`. Futhermore, we should explicitly prevent any other *future* host on the Clients network from reaching the server.

Security policies are often provided to network engineers in descriptive details like the above. It is up to you as the network engineer to process the requirements and convert that to a proper network configuration. Let's look at the policy and pull out and review the important parts.

1. Clients 1 and 2 **CAN** talk to the server. This means we'll be *PERMITTING* traffic sourced from these hosts.

2. We must used a **NUMBERED** standard ACL. ACLs are named or numbered. And when they are numbered, there are specific number ranges valid for different types of ACLs.

ACL Type	Number Range
Standard	1-99, 1300-1999
Extended	100-199, 2000-2699

    - Let's keep it simple and use number **1** for the ACL.

3. We need to apply it to the `router`, but we are not told which interface to apply it to. That seems to be up to us.

4. We must **EXPLICITLY** prevent other hosts on the client network from accessing the server. ACLs include an *implicit* deny at the end, but we must create our own ACE *DENYING* the other traffic.

We are ready to create our ACL on `router`.

Creating the ACL

Open up the console on `router` so we can apply the configurations.

First we'll permit `client1` and `client2`. There are two options when matching an individual IP address with an ACE. You can use the `host` keyword followed by the IP address. Or you can use the "match all" wildcard mask following the IP address. We'll use both options here for practice.

```
access-list 1 permit host 192.168.0.17
access-list 1 permit 192.168.0.33 0.0.0.0
```

> ***Key Point:*** Access Control Lists use *wildcard masks* and not subnet masks in the statements. Wildcard masks are the inverse of "subnet masks", where `0` bits indicate a matching bit is required, and a `1` bit indicates that the bit doesn't need to match. So a wildcard mask of `0.0.0.0` (all `0`s) means that ALL bits, and therefore the entire IP address, must match.

Next we will handle the explicit deny for all other hosts on the client network.

```
access-list 1 deny 192.168.0.0 0.0.0.255
```

The use of the `0.0.0.255` wildcard mask means that any source IP address where the first three octects are `192.168.0.` will match this line and be "denied".

> ***Key Point:*** When an ACL is applied to traffic, each ACE in the list is checked starting from the top of the list. As soon as a match is hit, the processing stops and the action, either permit or deny, is taken. So best practice is to place the most specific ACEs nearer the top of the ACL. So hosts before networks.

Before we apply the ACL to an interface, let's run a couple of "verification" commands that are useful to know. Start with `show access-lists`.

```
router# show access-lists

Standard IP access list 1
    20 permit 192.168.0.33
    10 permit 192.168.0.17
    30 deny   192.168.0.0, wildcard bits 0.0.0.255
```

This command prints a "readable" view of all access-lists configured on the device. By "readable", I mean that this isn't the same as the configuration commands used to create the ACL.

Now look at the ACL in the running-configuration with `show running-config | sec access-list`.

```
router# show running-config | sec access-list

ip access-list standard 1
 10 permit 192.168.0.17
 20 permit 192.168.0.33
 30 deny 192.168.0.0 0.0.0.255
```

Notice how the configuration shown does NOT match the configuration we applied. This is because there are multiple syntactally correct ways to configure ACLs, but they are normalized to this view when stored in the configuration. And should you want to update this ACL in the future, you could use whichever syntax you prefer at the time.

Applying the ACL

With the ACL created, we can now apply it to an interface. But which interface, and in which direction?

We weren't told what to use in the policy, so we'll apply it to the interface connected to the `Client` network, or `Eth0/1`.

And since traffic from the clients comes *into* interface `Eth0/1`, we will apply the ACL `inbound` on the interface.

```
interface eth0/1
  ip access-group 1 in
```

Verify that it is applied to the interface with the command `show ip access-lists interface eth0/1`.

```
router# show ip access-lists interface eth0/1
Standard IP access list 1 in
    20 permit 192.168.0.33
    10 permit 192.168.0.17
    30 deny   192.168.0.0, wildcard bits 0.0.0.255
```

You will have different numbers of "matches" for each line, but you should see something similar to the above.

Testing that the policy is working

First, verify that each client can still reach the server by using Firefox from VNC, or by pinging at the console.

To verify the DENY line, we will simulate another client by adding a second IP address to `client1`. Open a console connection to `client1` and run this command.

```
sudo ip address add 192.168.0.134 dev eth0
```

You can verify the address was added with `ip address show dev eth0`

Now send 3 pings to the server from this new IP address.

```
ping -c 3 -I 192.168.0.134 server
```

You should get an output that indicates 100% packet loss.

```
PING server (172.16.0.65) from 192.168.0.134: 56 data bytes

--- server ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Change to `router` and check the ACL stats. You should now see 3 matches on the deny line.

```
show ip access-lists interface eth0/1

Standard IP access list 1 in
    20 permit 192.168.0.33 (123 matches)
    10 permit 192.168.0.17 (87 matches)
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (3 matches)
```

Great job! Time to move onto the next policy.

Policy 2: All hosts have Internet access except Client 2

After our success with the first policy, we've been given a new policy goal. Client 2 is a special purpose machine that needs to be BLOCKED from accessing the Internet, but *all other hosts* should have Internet access. Furthermore, we must ensure that the first policy is still in effect after our changes, however we can make changes to the way policy 1 was enforced if required. We can add or change named or numbered standard ACLs needed to achieve this new policy.

Let's highlight the key aspects of this challenge we must keep in mind.

1. We must use standard ACLs for this task, which means we can only match based on SOURCE addresses.
2. The simulated internet is reached through interface `eth0/2` on the `router`.
3. When combined with policy 1, there are some potentially conflicting elements.
    1. Policy 1 tells us to PERMIT access from `client2`, while policy 2 requires DENYING access from `client2`.
    2. Policy 1 tells us to DENY access from the `clients` network, while policy 2 requires PERMITTING access from ***all other hosts***, which would include the `clients` network.
4. We are allowed to make changes to the ACL applied in the first task.

This exercise is about exploring the importance of properly placing ACL enforcement on interfaces within a network. It is impossible to complete this policy with a single standard ACL applied on one router and interface.

There is a "rule of thumb" when using standard ACLs to enforce traffic policy. Standard ACLs match traffic based on the SOURCE address, and should be applied as close to the DESTINATION as possible.

The new ACL that will limit access to the Internet needs to be applied on the `router` on the interface connected to the simulated internet, `eth0/2`.

We've been told we can use *named* standard ACLs for this task. Named ACLs are often preferred over numbered because properly named ACLs are easier to understand and maintain.

Creating the ACL

Create this ACL on `router`.

```
ip access-list standard STANDARD-INTERNET-ACCESS
  deny 192.168.0.33
  permit any
```

> The order of ACEs in this ACL is important. Remember that ACLs are processed top down, so if the order was reversed, the PERMIT line would match traffic from `client2`. The more specific entries need to be placed above more general entries.
>
> ***Extra Techincal Detail:*** This exercise and policy very specifically says that "all other hosts" are permitted access to the internet so that the `permit any` statement is used in the ACL. This is because outbound ACLs filter all traffic, including ARP traffic coming from the router itself. If this policy directed us to only allow "client network" traffic, the ARPs from the router to the simulated internet hosts would be blocked.
>
> If you experiment with ACLs, and you absolutely should, keep the "implicit deny" at the end of every ACL in mind because it is gauranteed you'll find yourself troubleshooting something not working as expected that is caused by it.

Applying the ACL

Next we must apply the ACL on interface `eth0/2`, but in which direction? Consider the traffic flow through the network. Trace your finger from `client2` to the simulated internet.

1. Traffic goes into interface `eth0/1` on `router`
2. Traffic goes ***out*** interface `eth0/2` on `router`

So we'll apply the ACL *outbound* from the interface.

```
interface eth0/2
 ip access-group STANDARD-INTERNET-ACCESS out
```

Testing that the policy is working

Now let's verify that the policy is working as expected. We'll use ping for these tests, but you can also use Firefox to check if the web servers are available still.

First, test from `client1` that both `internet100` and `internet200` are reachable.

```
ping -c 3 internet100
ping -c 3 internet200
```

Now repeat the pings from `client2`. These should fail.

Next, take a look at the counters on the ACLs.

```
show ip access-list

Standard IP access list 1
    10 permit 192.168.0.17 (35 matches)
    20 permit 192.168.0.33 (34 matches)
```

```
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (42 matches)
Standard IP access list STANDARD-INTERNET-ACCESS
    10 deny   192.168.0.33 (14 matches)
    20 permit any (30 matches)
```

Excellent, we are seeing matches on both the `deny` and `permit` lines.

One last test, let's make sure that "all other hosts" can also reach the internet. We'll use the same trick as before, adding a 2nd IP address to `client1` that we can test from.

> Note: If you didn't remove the 2nd IP address after the first policy tests, you don't need to re-add it to the host.

```
sudo ip address add 192.168.0.134/24 dev eth0

ping -c 3 -I 192.168.0.134 internet100
```

Did it work? It probably didn't... can you figure out why?

.

.

.

.

.

.

That's right, the ACL we applied for policy 1 is still active and is configured to `deny` traffic from other `client` network addresses. We'll have to fix this next.

Updating policy 1 enforcement

We can use the same "rule of thumb" about applying standard ACLs close to the destination to resolve this issue.

First, remove the ACL from interface `eth0/1`.

```
interface eth0/1
 no ip access-group 1 in
```

Now try the ping from the secondary IP address again.

```
ping -c 3 -I 192.168.0.134 internet100
```

This time the pings should complete successfully.

> If the pings do NOT complete at this time, verify that you correctly removed the ACL from interface eth0/1. If it is still not working, stop/wipe/start the `client1` node. Occassionally something "odd" happens with the networking stack on the Alpine node when you are adding/remove IP addresses.

Policy 1 is now NOT being enforced. You can test this by attempting the ping from the secondary IP address.

```
ping -c 3 -I 192.168.0.134 server
```

These will work, but the policy says they need to be blocked.

Apply the ACL "outbound" on interface `eth0/0` which connects to the `server`. This will re-apply policy 1.

```
interface eth0/0
 ip access-group 1 out
```

Now we'll use this opportunity to see how the "Extra Techincal Detail" that was noted above is so important with a little explicit experiment.

Clear the ARP entry for the server from the `router`.

```
clear ip arp 172.16.0.65
```

> You can view the ARP table with `show ip arp`

Now test if you can reach the `server` from `client1`. According to policy, and the ACL we configured, this is `permitted` traffic.

```
ping -c 3 server
```

These pings should fail because the ARP from the router is being blocked by the new ACL. You can verify that with the `show ip arp` command.

```
router# sho ip arp

Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.16.0.65           0   Incomplete      ARPA
.
```

The `Incomplete` status is listed because the ARP is not completing.

To resolve this, we need to add the `permit any` to the end of the ACL, just like we have on the ACL for the interent interface. This should be the last entry in the ACL, and we need to keep all the existing entries in the list. You've probably noticed the numbers in the `show ip access-list` commands. We haven't specified a number yet, but we can use them to our advantage here. Planning for possible future changes to the ACL, we will give lots of "room" before the ending `permit any` by making it a high line number.

```
access-list 1 1000 permit any
```

Now try the ping to `server` again.

```
ping -c 3 server
```

It should succeed this time, and if you check the ARP table, you'll see the entry is now complete.

Final testing of Policies 1 and 2

Complete the testing of policy 1 and policy 2 enforcement with the following tests.

1. Attempt to ping `server` from `client1`. This should succeed.
2. Attempt to ping `server` from `client2`. This should succeed.
3. Attempt to ping `server` from `client1` using the secondary IP address. This should fail.
4. Attempt to ping `internet100` from `client1`. This should succeed.
5. Attempt to ping `internet100` from `client1` using the secondary IP address. This should succeed.
6. Attempt to ping `internet100` from `client2`. This should fail.

Policy 3: Internet host "internet100" is not allowed to talk to any internal host

For this third policy, you'll need to do the planning and work to implement it. Then you can verify your solution against the suggested solution to see how you did. Fun right!

We are now being asked to filter traffic from particular simulated internet hosts. Once again we must use standard access control lists, and all previous policies must remain in effect. After our changes. It has been determined that "internet100" is a malicous site, and we need to prevent any traffic from it reaching our network.

> Note: This new policy preventing access to "internet100" is an update to policy 2 that allowed access to it. So after this policy is applied, the clients should no longer be able to reach "internet100".

Your tasks are to:

1. Determine which interface, and in which direction to apply the standard ACL
2. Create the appropriate ACL
3. Apply the ACL

▼ Solution

Our "rule of thumb" about placing standard ACLs close to the destination may make you consider applying the ACL on the interfaces connected to the clients and server, but because this policy is for ***all*** internal hosts, we can place this inbound on `eth0/2` connected to the simulated internet.

We need an ACL that denies traffic from "internet100" (192.0.2.100), and permits all other traffic.

```
ip access-list standard STANDARD-BLOCK-INTERNET100
 deny host 192.0.2.100
 permit any
```

And to apply it

```
interface eth0/2
 ip access-group STANDARD-BLOCK-INTERNET100 in
```

You can verify the policy is being applied by attempting to `ping internet100` from `client1` (remember `client2` has no internet access).

```
client1:~$ ping -c 3 internet100
PING internet100 (192.0.2.100): 56 data bytes

--- internet100 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

And check the ACL counters on the `router`

```
router# show ip access-list
Standard IP access list 1
    10 permit 192.168.0.17 (13 matches)
    20 permit 192.168.0.33 (15 matches)
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (18 matches)
    1000 permit any (113 matches)
Standard IP access list STANDARD-BLOCK-INTERNET100
    10 deny   192.0.2.100 (10 matches)
    20 permit any
Standard IP access list STANDARD-INTERNET-ACCESS
    10 deny   192.168.0.33 (18 matches)
    20 permit any (165 matches)
```

> Note: They way the policies are written and applied, the `echo` from `client1` is permitted to leave `router`. It is the `echo-reply` from `internet100` that is blocked.

You can and should also verify that the other policies are still in affect and working. Access to `internet200` and `server` should still be working from relevant hosts.

Policy 4: Internet host "internet200" is allowed to talk to clients, but not to servers

We will continue the fun here with the fourth policy. Plan out and apply the solution, and then check your answer against the provided one.

The host `internet200` isn't malicious, but it has been decided that the `server` has no need to communicate with it. So we have been asked to use a standard ACL to prevent traffic from `internet200` from reaching the `server`. Clients can continue to visit this destination. As with other changes, previous policies must still be enforced after our changes.

Your tasks are to:

1. Determine which interface, and in which direction to apply this change
2. Configure the appropriate ACL
3. Apply the ACL

▼ Solution

In this case, the "rule of thumb" about applying close to the destination applies. And that would be interface `eth0/0` connected to the `server` network. And because the traffic in question is traveling outbound from the interface, it will be applied in the "out" direction.

However, we already have an ACL applied on this interface in the "out" direction.

```
router# show run interface eth0/0
Building configuration...

Current configuration : 127 bytes
!
interface Ethernet0/0
 description link to Services Network
 ip address 172.16.0.1 255.255.255.0
```

```
 ip access-group 1 out
end

router# show ip access-list 1
Standard IP access list 1
    10 permit 192.168.0.17 (13 matches)
    20 permit 192.168.0.33 (15 matches)
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (18 matches)
    1000 permit any (206 matches)
```

You cannot apply more than one ACL to an interface, but we can add an additional ACE to the access-list.

The key to this task is where in the ACL to add the rule. If we add it at end, rule `1000 permit any` would prevent the new `deny host 192.0.2.200` from being triggered. So we must add this new line before line 1000

```
router(config)#access-list 1 40 deny host 192.0.2.200
router(config)#end

router# show ip access-list 1
Standard IP access list 1
    10 permit 192.168.0.17 (13 matches)
    20 permit 192.168.0.33 (15 matches)
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (18 matches)
    40 deny   192.0.2.200
    1000 permit any (225 matches)
```

Now verify the policy is working by attempting to `ping internet200` from the `server`.

>See above in the "Server to Simulated Internet Access" pre-validation step for a reminder on how to access the interactive console for the server.

```
root@8cd5369bec52:/# ping -c 3 internet200

PING internet200 (192.0.2.200): 56 data bytes

--- internet200 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

And verify the counters on the ACL.

```
router# show ip access-list 1

Standard IP access list 1
    10 permit 192.168.0.17 (13 matches)
    20 permit 192.168.0.33 (15 matches)
    30 deny   192.168.0.0, wildcard bits 0.0.0.255 (18 matches)
    40 deny   192.0.2.200 (12 matches)
    1000 permit any (240 matches)
```

Finally, check to make sure the other policies are still in effect.

Extended ACLs and Destination Based Policies

So far in this lab we have focused on standard ACLs. Standard ACLs allow for matching solely based on the ***source*** IP address of the traffic. This limitation means that we can't create more granular policies that consider elements like the ***destination*** IP address of the traffic. Or even more granular ones where the protocol, port, or application are considered. That is where **extended ACLs** come into play.

Extended ACLs allow matching on many different aspects of network traffic. For CCNA candidates, the parts we will focus on are:

- Source IP address
- Destination IP address
- Protocol (ie IP, TCP, UDP, ICMP, etc)
- Source port number
- Destination port number

This extra flexibility is why it is nearly always the case that ACLs to filter traffic through an interface, like the policies we've been exploring in this lab, use extended ACLs. So you might be asking, "Why did we spend so much time using standard ACLs then?". The reason is all about learning and education.

In CCNA Prep Season 3 Episode 3, we will spend a lot of time exploring how this greater flexibility can be put to use. However, before we finish up this lab, let's look at how extended ACLs can include destination IP address in a policy.

Policy 5: Clients should not be able to access the new "internet210" host

A new host `internet210` has been discovered on the simulated internet. It hasn't been determined if it is malicious or not yet, so the security team has asked that you block access from the client network to this new destination. You've been asked to use an extended ACL to impliment this new policy. As before, all other policies must maintain in effect.

Now we've done similar restrictions before with standard ACLs. We could add it to the inbound ACL on the internet interface, but with extended ACLs, our "rule of thumb" changes.

Extended ACLs include details about the destination in the policy, so we want to apply them as close to the source as possible. No need to send traffic across the network that is going to be dropped later. That is inefficient.

Creating the ACL

So let's create the access-list first. We'll use a named access list, however we could use a valid numbered ACL as well.

```
ip access-list extended EXTENDED-INBOUND-CLIENT-NETWORK
 10 deny ip 192.168.0.0 0.0.0.255 host 192.0.2.210
 1000 permit ip any any
```

Some key points about this ACL

- This ACL might be changed later with other policies from the client network. So the name chosen isn't just about "internet210"
- For the same future planning for changes, line numbers are used and the `permit ip any any` statement is placed at line number 1000 to provide lots of "room" for future statements

- Extended ACEs have more details included. The `ip` refers to the protocol. By using `ip`, any type of traffic will match this line.
- We need the `permit ip any any` at the end to override the "implicit deny" at the end of every ACL.

Pre-testing connectivity

Before we apply the new ACL, check if `client1` can access `internet210`.

```
client1:~$ ping -c 3 internet210

PING internet210 (192.0.2.210): 56 data bytes
64 bytes from 192.0.2.210: seq=0 ttl=42 time=1.278 ms
64 bytes from 192.0.2.210: seq=1 ttl=42 time=1.411 ms
64 bytes from 192.0.2.210: seq=2 ttl=42 time=1.326 ms

--- internet210 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.278/1.338/1.411 ms
```

Applying the ACL

Okay, let's see if we can stop that traffic from working. Apply the new extended ACL inbound on `eth0/1`.

```
interface eth0/1
 ip access-group EXTENDED-INBOUND-CLIENT-NETWORK in
```

Testing that the policy is working

Try to ping `internet210` again.

```
client1:~$ ping -c 3 internet210
PING internet210 (192.0.2.210): 56 data bytes

--- internet210 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Okay, that looks good. Check the hits on the ACL.

```
router# show ip access-list EXTENDED-INBOUND-CLIENT-NETWORK

Extended IP access list EXTENDED-INBOUND-CLIENT-NETWORK
    10 deny ip 192.168.0.0 0.0.0.255 host 192.0.2.210 (3 matches)
    1000 permit ip any any (10 matches)
```

And finally, verify that `client1` can still reach the `server` and `internet200`, hosts that should be reachable. You can also verify the other policies are in effect if you'd like.

Bonus Homework Policy

Now that you've seen how to create a basic extended ACL that matches based on both source and destination addresses, go back through Policies 1 - 4 and consider how you might implement them

using extended ACLs instead of standard ACLs.

Great Job!

Excellent work on this lab! You should have a much deeper appreciation and understanding of how standard ACLs work and how to use them for filtering traffic. And you've gotten just a hint at the flexibility of extended ACLs. Be sure to continue your journey into ACLs with the next episode and lab of CCNA Prep where the fun continues.