Exercise5EscapeRoomPlayground

Jump to bottom

sethnielson edited this page 4 days ago · 7 revisions

Exercise 5: Escape Room over Playground

Assigned	9/11/2019
Due	9/16/2019
Points	25

Overview

This assignment is pretty easy. You're going to setup Playground, connect to Playground, and connect your Escape Room over the Playground Overlay to an Autograde server there.

And, just to add in a little bit more to the escape room, you'll add in some triggers to the open and unlock chest. Let's do that part first.

Updating the chest

If you play the Escape room, you'll notice that if you look at the chest, even after unlocking or opening it, it says that it appears to be locked. The code to get the correct description is *already in the sample*. Take a look at `create_chest_description()` and you'll see that it is capable of making a new description correctly.

But, it needs to be *told* to do so. The description function in the Escape Room is static. It doesn't change dynamically. Rather, upon a trigger being fired, the static description can be changed.

All you need to do for this part of the exercise is add in a trigger to the chest on "unlock" and "open" that update the description. Use the code as a guide. There are other examples there.

Installing Playground.

This quickstart is intended to get you up and running with playground *quickly*. You won't understand everything, but you will have a working system in a very short amount of time. As you follow the quick start, make mental notes of the things you want to understand better and see if the additional documentation helps you figure it out.

Playground Quickstart

Playground is an *overlay* network. An overlay network is a network that operates on top of another network with its own addressing and semantic schemes. For example, email is an overlay network. It has its own addressing (e.g., `sethjn@cs.jhu.edu`) and its own semantic connections (I'm connected to everyone in my address book). The fact that it *usually* runs over the IP network is irrelevant. When you're thinking about sending someone an email, do you think about the IP addresses involved? No! Of course not! And the addressing means that your message can get to your contact regardless what system they're using. In fact, email could be delivered without IP addresses at all.

Similarly, we are going to create a Playground network that has its own addressing and its own semantic connections. In this quick start, the entire network will run on your own personal computer. But later, we will build a network that connects the entire class. The playground network runs on top of TCP/IP, but that *doesn't matter*! Ignore that. Try to focus on the overlay network we're setting up here as much as possible. Occasionally, you do have to think about the TCP/IP pieces (just as email does), but it's minimal.

The first thing to do is get Playground installed.

Installing Playground

The current major version of Playground is Playground 3.0 (Playground3). It can be cloned out of github from https://github.com/CrimsonVista/Playground3. You should clone it to your local computer, even though you will typically not need to modify it.

Once you have it cloned, you can use pip (a python installation manager) to install it for your python to use. Please note, *I HIGHLY recommend* using python virtual environments as shown below (the current directory is shown as the prompt... your individual prompt will look different and your individual directories can be different):

```
[~] > mkdir netsec_env
[~] > cd netsec_env
[~/netsec_env] > python3 -m venv .
[~/netsec_env] > source bin/activate
[~/netsec_env] (netsec_env) > git clone https://github.com/CrimsonVista/Playground3
[~/netsec_env] (netsec_env) > pip install -e ./Playground3
```

Note, some versions of pip have an error about `bdist_wheel`. If you get this error, you may need to install `pip install wheel`.

Playground is now installed. There are two major components:

1. The underlying virtual hardware layer
2. The API to access the network from a Python program

We'll start by configuring some hardware

Playground Hardware

All playground networking controls are accessible through the `pnetworking` command. Once you've installed playground, you should have access to this utility. If not, make sure that you're in the right virtual environment (you will need to call `source bin/activate` from the directory you configured... read up on virtual environments if you need more info).

`pnetworking` has a number of subcommands. We won't explore them all today. It also makes use of a config file. For simplicity, we'll start by configuring a config file for your user name. Later, we'll be able to have more than one configuration, which is useful for connecting to different networks.

To initialize your playground network, execute the following command

```
[~/netsec_env] (netsec_env) > pnetworking initialize user
```

You should get a message that this succeeded, but you can always check by running the status command:

```
[~/netsec_env] (netsec_env) > pnetworking status

Configuration: /home/sethjn/.playground

Playground Networking Configuration:
------------------------------------
```

The configuration is currently blank because we have created no hardware.

To get communications going in playground, we need two virtual pieces of hardware: a switch and a nic (network interface card). In networking, a switch connects multiple devices together and a nic is how your device interacts with the switch.

We're going to have one switch and one nic. You might ask yourself, how will we have networking if we only have one nic? Don't we need at least two nic's? Sure. But just like you can network programs together on your computer, you can network playground programs together on your computer as well. It's all local for now.

To create a switch, execute the following command:

```
pnetworking add my_switch switch
```

The `add` command takes at least two arguments: the name of the device and the type. So this command created a device called `my_switch` that is of type `switch`. You can give it other arguments to specify exactly what IP address and port it should be using, but this is all we need for our initial tests.

if you do a `pnetworking status` now, you should see something like this:

```
Configuration: /home/sethjn/WIN_DEV/stage/20191/playground/.playground

Playground Networking Configuration:
------------------------------------

switch my_switch:
        Status: Disabled
        Auto-Enable: True
        PNMS Managed: True
        TCP Location: 127.0.0.1:63902
```

Most of this doesn't matter. The `Auto-Enable` and the `PNMS Managed` have to do with some auto-configuration stuff. The only thing worth noting is perhaps:

1. It is currently disabled
2. It is using a TCP/IP location

Let's talk about the TCP/IP first. As I said earlier, Playground creates an overlay network on top of TCP/IP. The virtual hardware pieces connect to each other over TCP/IP. But we will be writing playground programs that use the virtual hardware with playground addresses. We will *think* in terms of playground even though underneath, the actual network traffic goes over TCP/IP. Again, think of email. Email has to be sent over TCP/IP, but you don't think about it that way. You just say, "I want to send an email from me (sethjn@cs.jhu.edu) to a student (somestudent@jhu.edu)". You have a from and a to address and you let however it actually gets there be hidden and unseen.

Similarly, we will have playground addresses and playground ports. THESE ARE NOT TCP addresses and ports. I can only say this so many times, but every year students confuse the two. Please understand this. I'll say it one more time:

Playground Addresses and Ports are not TCP Addresses and ports

So what is a playground address and port? Let's start with the address. Playground addresses are any four numbers "dotted" together.

```
0.0.0.0
100.100.100.100
9999999.9999999.9999999.9999999
1.2.3.4
192.168.1.1
```

All of the above examples are legal playground addresses. We will typically not use addresses that look like IP addresses to avoid confusion. In fact, for our purposes, we will have our first number always be `20191` which represents the year plus the term. In other words, most of the "official" addresses you will see are:

```
20191.x.y.z
```

of course, this is just convention. There is nothing stopping you (or a classmate) from using something else. But, for example, we will allocate addresses for each team using this scheme. The class servers (such as the bank) will use this prefix as well.

But what *is* an address? A playground address is used by a nic to tell the switch what traffic it is interested in. The virtual nic sends messages with to addresses and from addresses. The switch takes the incoming data, looks at the destination address, and routes it to any device (or to any devices) that are claiming that address.

When you create a VNIC (virtual nic) device, you assign it a playground address.

```
pnetworking add my_nic vnic 20191.10.20.30
```

This adds a virtual nic to the hardware configuration that will claim `20191.10.20.30` as its address when it connects to the switch.

In addition to creating a vnic, we have to make two other configurations. First, we have to tell it what switch to connect to, and we have to tell it what playground addresses it routes for. This second piece is in case we have *two* (or more) nics at the same time. They can claim different outbound traffic. But when we just have one, we set it as the "default" route.

```
pnetworking config my_nic connect my_switch
pnetworking config my_nic route add default
```

Run `pnetworking status` again, and you should see both devices configured but disabled. Let's turn them on.

```
pnetworking on
pnetworking status

Configuration: /home/sethjn/.playground

Playground Networking Configuration:
------------------------------------

switch my_switch:
        Status: Enabled (PID: 1186)
        Auto-Enable: True
        PNMS Managed: True
        TCP Location: 127.0.0.1:63902

vnic my_nic:
        Status: Enabled (PID: 1193)
        Auto-Enable: True
        Playground Address: 20191.10.20.30
        Connected To: my_switch (127.0.0.1:63902) (Live)
        Routes:
                __default__
```

This status tells you that we have a switch running and a vnic connected to it claiming the address `20191.10.20.30`. That's good! We now have a working playground networking!

Connecting to the Switch

On your VM, please install playground like we talked about in class, and as covered in the tutorial above. Much of it will be exactly the same. The biggest difference is that you won't use a "managed" switch.

In pnetworking, nic's and switches are connected by *name*. So, how do you connect to a switch that *isn't in your local pnetworking config*? That is, the class switch isn't managed by your pnetworking. So how can you connect to it?

That answer is, you create a "dummy" version of the switch in your own configuration. The dummy version is to create a name that represents the real switch somewhere else. Because it's a dummy version, it will always say that it's "on". Maybe some day pnetworking will have something more advanced, but this is the best we can do for now.

So, once pnetworking is configured, add the class switch like this:

```
pnetworking add class_switch switch remote 192.168.200.52 9090
```

When you do pnetworking status, it will show something like:

```
switch class_switch:
         Status: Enabled
         Auto-Enable: True
         PNMS Managed: False
         TCP Location: 192.168.200.52:9090
```

Don't be fooled by "Status: Enabled". Remember, this is a dummy! You will be able to tell if the switch is "up" and "live" because when you connect your nic to it, it will say "live" if there's an actual connection.

Once this class switch is added, you connect your nic to it just like you would a local switch, e.g., `pnetworking config my_nic connect class_switch .`

To repeat, the class switch is at the following address:

- IP: 192.168.200.52
- Port: 9090

Test that you have working functionality by running the Playground echotest demo between you and a teammate on different computers.

We will also walk through some of this in class.

Conversion

In it's default form, playground can be used instead of typical TCP/IP **almost** exactly the same.

The easy part is to replace:

- `loop.create_server` with `playground.create_server`
- and `loop.create_connection` with `playground.create_connection`

But there are a couple of other minor changes you need to make and be aware of.

1. You CANNOT use `host=''` or `host='127.0.0.1'`. Playground does not use `127.0.0.1` and, in the interest of being explicit, it does not allow `host=''` either. For servers, please *leave out the host parameter altogether*, or use the Playground address, or use `localhost`. The `localhost` keyword is just a special word that tells Playground to look for the "default" NIC's address.

2. Currently, Playground is not using "sessions" like TCP is. You'll learn more about this when we talk about TCP and handshakes. But in short, *there is no connection*, and there won't be until you create the TCP replacement layer. This has two important consequences. (1), the server doesn't know the client is there until it receives data, so the client must ALWAYS speak first. (2) the client will happily report `connection_made()` without that meaning anything useful. So just because your `connection_made` is getting called on the client side doesn't mean you put in the right address, port, etc.

This last point means that you need to modify your client to speak first, even if the server normally speaks first. You can always send an empty command, e.g., `transport.write("<EOL>\n")` right in the client's `connection_made()` method.

That's a lot of explanation, but the changes are still very small. In short, they are:

1. Change `loop` to `playground` for all `create_connection` and `create_server`
2. Change any `host=""` or `host="127.0.0.1"` to `host="localhost"`
3. Add a `transport.write("<EOL>\n")` to your client protocol's `connection_made` method to alert the server to its presence

And as a final note, the playground address of the autograde server is `20194.0.0.19000`. Remember, once you're in Playground, TCP/IP doesn't exist anymore! You can't connect to this autograde server on 192.168.200.52, because that address doesn't exist in playground!

▼ Pages 16

Find a Page...

Home
BackgroundOverlayNetworks
Exercise1GettingStarted
Exercise2EscapeRoomSockets
Exercise3EscapeRoomAsyncio
Exercise4EscapeRoomAsychUserInput
Exercise5EscapeRoomPlayground
Exercise6EscapeRoomPackets
Network Security Fall 2019
Playground
PlaygroundBankQuickstart
PlaygroundConverstionQuickstart
Readings

Clone this wiki locally

```
https://github.com/CrimsonVista/20194NetworkSecurity.wiki.git
```