Exercise10MonitorPlayground

Jump to bottom

sethnielson edited this page 8 hours ago · 1 revision

Exercise 10: Monitoring Playground Traffic

Assigned	10/2/2019
Due	10/7/2019
Points	25

Overview

This exercise should be a lot of fun for three reasons.

1. You get to submit the assignment as a team for the first time.
2. You get to spy on your neighbors and the bank
3. You can try some password cracking.

I wanted every assignment this semester to be auto-graded, but this one is going to require you to submit a write-up to the TA for grading. Please pay careful attention to the requirements.

Part 1: Create your own escape room

As a team, you will deploy an escape room on Playground for other teams to play. You will create this as a team and post your playground address and port on slack in the `#labs_and_exercises` channel. Please *pin* your escape room's location.

You may start with the existing escape room or from scratch. But everyone in the team MUST create at least one "feature." A feature needs to be some recognizable part of the escape room process. The write-up must detail what each member of the team contributed to the escape room.

Please note, you can spend as much time on this as you want, but you do not have to spend much. Adding a "feature" might only take you 15 minutes of copy-cut-paste and modifying what's already provided in the escape room I've given you. But there is an extra credit portion of this assignment, so keep reading.

You need to have your escape room running and available for your peers to use by Friday at noon! You can keep adding to it, but you must have the room available to play and posted to Github by that time.

Also, you will be connecting to a different class switch as we discuss in the next section.

Part 2: Eavesdropping and Password Hash Collecting

For this assignment, we will be using what I will call the "Live Fire" switch. The "Live Fire" Switch allows network attacks as expressed in the Syllabus. If you look at the Syllabus, at this time in the class, we are at Attack Level 1. That means you are allowed to *eavesdrop* but you are not allowed to custom-code packets. As you will see in this assignment, you *could* log in as anyone you overhear, but I'm not going to permit you to do that just yet.

However, you *can* eavesdrop. Let's see how to do that in playground.

As you know, you typically connect to a switch in playground using a virtual nic. In fact, I need to tell you the new switch you should connect your nic to for accessing the Live Fire network.

```
IP Address: 192.168.200.52
TCP Port: 10666
```

The bank for this life fire network is at:

```
playground address: 20194.0.1.1
playground port: 888
```

You will need your playground networking to be able to connect to either the "safe"/autograde switch or the live fire switch, as we will have more auto-grade assignments in the future. There are two ways of switching back and forth.

First, you could simply add a new nic to your pnetworking, connect it to the live fire switch, and then use `pnetworking config` to set this nic to be your default route. This requires removing the default route from the current nic and setting the default route to the new nic.

The second option is to create a separate pnetworking installation using playground instances. In this scenario, you would create a special directory for this configuration, use `export PLAYGROUND_INSTANCE=<new directory>`, and then do `pnetworking initialize instance`. The instance can have its own configuration and its own connections.

Either way, make sure you know that you can connect to either network.

Now, we're going to learn how to eavesdrop. Remember, this is only allowed on the live fire network.

We've already discussed a simple way to eavesdrop in class. You can set your VNIC with any address you choose, including one that is already in use. So, for example, it would be simple to set your VNIC to the address 20194.0.1.1 and get a copy of all bank traffic. In fact, you could run your own bank on that very address.

The problem is two fold. First of all, how do you deal with duplicated writes (i.e., writes from your fake server and writes from the real server). Second, what if you want to listen to multiple addresses? After all, you might want to listen to the bank, but you might want to listen in on your opponents' escape rooms.

To solve this, we're going to set up a listening only server that listens to *all* traffic. In the real world, you can do something similar by putting your ethernet or wifi card into promiscuous mode. We can do the same thing with a VNIC. You'll use a new pnetworking command for this called `query`. While `configure` is for configuration before you turn a device on, `query` can interact with the device when it's already running.

To find out the promiscuous level of your current VNIC, do the following:

```
pnetworking query <VNIC name> get-promiscuity-level
```

To set the level:

```
pnetworking query <VNIC name> set-promiscuity-level <level>
```

The level can be from 0 to 4. For our purposes, you can set it to 4.

Now your VNIC will listen to *all* traffic. Don't worry about any clients or servers you have running, though. They will still only get traffic meant for them. To listen to all the traffic being collected, we need to connect with our VNIC directly and ask for the dump. Fortunately, I've provided a very simple interface.

First, you need to import the following two classes.

```
from playground.network.protocols.vsockets import VNICDumpProtocol
from playground.network.protocols.packets.switching_packets import WirePacket
```

The `VNICDumpProtocol` is designed to make it easy to get the data dumps we're looking for. the `WirePacket` is the packet used for transmitting data over Playground. Normally you *never* see these. These packets are stripped off before your protocols ever see anything. But for eavesdropping, it's nice to get the extra data.

To plug our dump protocol directly into the VNIC, we won't use `create_playground_server` or `create_playground_connection` because those are for the non-dump communications. Instead, using a sub-module named `connect`, I offer a `raw_vnic_connection` function. So, assuming you've imported playground, you can call:

```
playground.connect.raw_vnic_connection(protocol_factory, vnic_name="default")
```

The `protocol_factory` is a factory like any other protocol factory you've built so far. We'll talk about that in a minute. The `vnic_name` defaults to the default VNIC for your pnetworking (i.e., the one with the default route). But you can put in a vnic name explicitly if you are using a different VNIC for eavesdropping.

The main function of your python eavesdropper is pretty easy. You really just need something like:

```
loop.run_until_complete(playground.connect.raw_vnic_connection(...))
loop.run_forever()
```

As far as your eavesdropping protocol goes, it needs to deserialize `WirePacket` s. You can either use `PacketType` or `WirePacket` Deserializers, but the only thing that should come out of it should be a `WirePacket` . `WirePacket` s have the following fields:

```
("source",         STRING),
("sourcePort",     UINT16),
("destination",    STRING),
("destinationPort",UINT16),
("fragData",       ComplexFieldType(FragmentData, {Optional:True})),
("data",           BUFFER)
```

The `fragData` field has it's own components, which is why it's a `ComplexFieldType`. But most of the time you can ignore it for now. Most of our packets aren't large enough yet to need fragmentation. For a simple wire sniffing, you could just print out the source and destination data to see who is sending what to whom. But the next part of the lab will require you to do another step.

Collect User Password Hashes

Once you have your general network sniffer working, you need to start listening on the Live Fire network for bank traffic. Your job is to collect everyone's bank password hash. To do this, you will need to collect `data` from the `WirePacket` s and feed it into a *separate* deserializer buffer. The data is just serialized packets and you can reverse the process. Because you don't know what type will come out, you should use `PacketType.Deserializer`.

In real life, nobody sends password hashes. They send the actual password itself. Usually, it's sent over a secure channel (e.g., TLS) so it doesn't matter. Moreover, even if you send the hash, it's no more secure than sending the password. The only advantage to storing hashes instead of cleartext passwords on the server side is to protect *other* websites where you might be using the same password. We'll talk about this later in class.

But for now, I *am* sending password hashes because I'm forbidding you to re-play them. You may not custom code a packet with someone else's password hash. But I am giving you permission to try and brute-force passwords. If you do unlock someone else's password and steal all their bitpoints, you get extra credit.

For the write-up for this lab, you need to include the password hash of at least one person from every other team. (If a team doesn't participate, you can indicate that you listened but did not detect their login).

Creativity Extra Credit

Every team is required to play every other team's escape room at least once AFTER noon on Friday. You should all be sniffing the traffic at that time so that you can see everyone's login data.

In your report, rate each person's escape room on a scale of 1 to 10, with 10 being the most fun ever and 2 being absolutely boring. Rate it as a 1 if it's broken (throwing exceptions, interrupted, etc). The team with the highest rated escape room will get extra credit as well.

Report Summary

Your submitted report needs to include:

1. A list of each person's contributions to the escape room
2. A list of 1 password hash for one person from each team

3. (Optional) Any password hashes you could brute-force
4. A rating of every other team's escape room from 1 to 10 (only give a 1 if it's broken)

Please submit one report per team to the TA by the due date.

Clone this wiki locally

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