Exercise6EscapeRoomPackets

Jump to bottom

sethnielson edited this page 3 days ago · 5 revisions

Exercise 6: Escape Room with Packets

Assigned	9/18/2019
Due	9/25/2019
Points	25

Overview

Like we talked about in class, you need to start working with packets in your network communication. Playground provides a library to make this easier. In this lab, you're going to create a version of the escape room that uses network packets instead of just "strings".

You should first read up on the playground docs. They can be found here.

A bit about serialization

If you need a refresher about serialization or haven't understood the topic, look at this blurb to read more https://github.com/CrimsonVista/20194NetworkSecurity/wiki/Serialization

Part 1: Getting Set Up

First, go to the samples code and get the following two files:

1. `escape_room_006.py`
2. `autograde_ex6_packets.py`

As you might imagine, `escape_room_006.py` is for the game. It has a one-line change that makes no difference to anything, but will make something else work. Don't worry about this but do make sure you use this one.

The `autograde_ex6_packets.py` file has the packets you need for this assignment. The first two we are going to talk about are:

1. `AutogradeStartTest`
2. `AutogradeTestStatus`

The first of these is what you will send to the server to get the test started and the second is what the server will send back.

The `AutogradeStartTest` function is much like the data you have sent before except this time there is no command. The message type is the command. But otherwise, it has fields for `name`, `team`, `email`, and `port`. Please note the types of these fields when you look in the code. The `port` field, in particular, is an integer, not a string.

There is also one other field that might look strange: `packet_file`.

You will use this field to send your own packet definitions to the autograde server! But for now, we're going to get started just setting this field to the empty value of `b""`. This *will* cause an error, but it will help us get started communicating with the autograde server.

Part 2: Initializing a Test and Receiving a Test Status

You may recall from Exercise 5 that in our current version of Playground, we have no way of initiating a session. The server won't detect a new connection until the client sends data. So you're going to modify your client to not wait for the server, but just send the test start command right away.

In other words, in your `connection_made` function for your client, send a `AutogradeStartTest` message off to the server. Make sure to set all of the fields including `packet_file=b""`.

If you've successfully sent the message to the server, you should get back a response in `data_received`. You should have a deserializer (`PacketType.Deserializer`) that takes in the data bytes and uses `nextPackets()` to iterate through the responses. Initially, there should only be one. A single `AutogradeTestStatus`.

The `AutogradeTestStatus` packet type includes the `test_id` as well as three status fields for the submission part, the client test part, and the server test part. Each of these can have one of three values:

1. `0` represents `NOT_STARTED`. The client and server tests should be set to this value
2. `1` represents `PASSED`.
3. `2` represents `FAILED`. Your submit status should have this value.

Note that `AutogradeTestStatus` includes these as constants (e.g., `AutogradeTestStatus.PASSED`). So, again, for this test, you *should* have had some kind of failure. If the field `submit_status` is not `PASSED`, you should check the field `error`. You will see something like, `Failed module 'student_messages' has no attribute 'GameCommandPacket'`.

I'll explain this in a moment, but if you get this far, you're doing really well.

Part 3: Debugging

You will absolutely need to turn on debugging for this. When you do, however, you will start to see a *lot* of messages. You may find this overwhelming. I suggest that instead of importing and setting `PRESET_DEBUG` use `PRESET_VERBOSE`. This logs all the same messages, but to a file instead of to the screen. If you want to check your logging messages, you will find them under your `/.playground/logs/` directory. There will be a lot of files there, but your log will be named after your file.

In any event `PRESET_VERBOSE` will show you exceptions without printing out all the logging messages to the screen.

Part 4: Creating your own Game Packets

I want you to create a separate file that contains *only* the following code:

1. A definition of a packet called `GameCommandPacket`
2. A definition of a packet called `GameResponsePacket`

Here's the deal. You can create the internal fields of these packets, including the names and definitions, any way you choose. But, you must provide some *functions* that will allow me to access these.

By way of explanation, when you serialize an object, it serializes the *data* not the methods. But, when you restore the data on the other side, the methods defined can use the restored data.

This can be a little confusing, so let me help you. Your file should look something like this:

```
from playground.network.packet import PacketType
from playground.network.packet.fieldtypes import # whatever field types you need
```

```
class GameCommandPacket(PacketType):
    DEFINITION_IDENTIFIER = # whatever you want
    DEFINITION_VERSION = # whatever you want

    FIELDS = [
        # whatever you want here
    ]

    @classmethod
    def create_game_command_packet(cls, s):
        return cls( # whatever arguments needed to construct the packet)

    def command(self):
        # MUST RETURN A STRING!
        return # whatever you need to get the command for the game.

class GameResponsePacket(PacketType):
    DEFINITION_IDENTIFIER = # whatever you want
    DEFINITION_VERSION = # whatever you want

    FIELDS = [
        # whatever you want here
    ]

    @classmethod
    def create_game_response_packet(cls, response, status):
        return cls( # whatever you need to construct the packet )

    def game_over(self):
        # MUST RETURN A BOOL
        return # whatever you need to do to determine if the game is over

    def status(self):
        # MUST RETURN game.status (as a string)
        return # whatever you need to do to return the status

    def response(self):
        # MUST return game response as a string
        return # whatever you need to do to return the response
```

You can literally copy the above code into a file and save it. Then, fill in all the parts that are comments. Note that you can choose any fields you want, so long as you can construct the packet with a single command string for the command packet and a response string and status string for the response packet.

You can have more fields than these, by the way, so long as they are either optional or can be created with default values. For example, perhaps you'd like to include the time in a packet so you can see how long (approximately) it took you to get it. You can use `time.time()` to get the current time and automatically add that to a field inside the function `create_game_command_packet` or `create_game_response_packet`.

If your submit status is 1, then after your client sends the first packet and once your client receives the first packet in `data_received`, the client should start sending `GameCommand` packets and expect `GameResponse` packets in return

Also, note that if you have a field for the status, it cannot be called `status` because there is already a method with that name.

You **MUST** define the methods you see above. My code will rely on these functions.

Once you have them written, you should write a client and server for the game that use them. Test it out and make sure it works to send commands and responses back and forth. Note that your client can now use the `game_over` method on a packet to determine if the game has ended, without having to parse the response.

Part 5: Sending the Autograde Server your Packet File

WARNING: DO *NOT* put any code in the packets file except for your packet classes, and DO *NOT* write any "evil" code into your packet file. I will be importing your code directly. You could delete files, change the autograder, etc. Please don't do so. Although I am running this in a VM, and could restore any lost data, I can't have you, for example, poking around in other people's scores.

WARNING: If I find out that anyone has sent me an "evil" file for *any* reason, I will immediately fail you, send your name to the university, and potentially see about turning your name over to civil authorities for violations under the CFAA or other criminal statutes. We *will* do some ethical hacking in this class, but this assignment is NOT one of them. As part of this assignment, you will send an email to the TA acknowledging that you have read these warnings, understand them, and agree to not transmit any code for spying, altering data, or any other inappropriate activity and that doing so would result in the penalties listed above.

With that warning out of the way (please take it seriously), you can now complete the assignment. You need to modify your autograde client to read the data of your packets file and transmit it (as binary) in the `packet_file` field of the test start packet. The code might look something like this:

```
packet = AutogradeStartTest(name="my name", email="my email", team=n, port=nnnn)
with open("my_packet_file.py", "rb") as f:
    packet.packet_file = f.read()
```

If you've set up your client and server correctly for the game, my auto grader will be able to use them to interface with you. If you've done something wrong... hopefully you will get an error.

Upon completion of the client test, you will get another `AutogradeTestStatus` with both `submit_status` and `client_status` set. After completion of the server test, you will get an `AutogradeTestStatus` with all three status set.

Part 6: Checking Your Result

You can check your submission by submitting a `AutogradeResultRequest` with the appropriate `test_id`. You will receive a `AutogradeResultResponse` with the result.

▼ Pages 16

Find a Page...

Home

BackgroundOverlayNetworks

Exercise1GettingStarted

Exercise2EscapeRoomSockets

Exercise3EscapeRoomAsyncio

Exercise4EscapeRoomAsychUserInput

Exercise5EscapeRoomPlayground

Exercise6EscapeRoomPackets

Network Security Fall 2019

Playground

PlaygroundBankQuickstart

PlaygroundConverstionQuickstart

Readings

Serialization

Syllabus

Show 1 more pages...

Clone this wiki locally

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