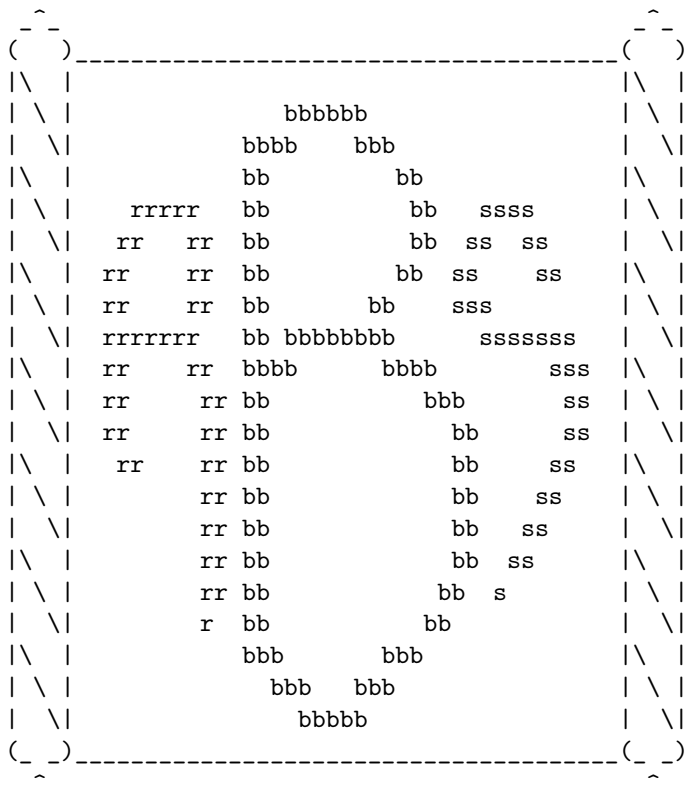

Python Programming Interview Challenge

Overview

The goal of this challenge is to write a small program that handles the parsing and verification of UDP packets, to demonstrate a small example of overall design architecture and code aesthetic.

Provided are a few files required for bootstrap as well as example input for testing purposes. Note that a variety of code inputs will be used to evaluate the efficacy of the program, the example provided should be used as a general test-case (but should work for any general variation of input).

In the course of solving the challenge you are free to use any library on PyPI, which should be included in a `requirements.txt` file at the root of your project structure.

Your submission should be implemented in **Python ~2.7.15**. If you wish to implement it in **Python 3**, please detail that specifically, and include a write-up on why you chose Python 3, plus any intricacies and modifications writing such a program in Python 3 would have to handle coming from Python 2.

Packet Structure

Below is an outline of the structure of an incoming packet. These incoming packets have been created with proper network byte ordering.

```
================================================================
( 4 bytes )      Unique Packet ID for the checksummed binary
================================================================
( 4 bytes )      Packet Sequence # (Total Checksums Processed)
================================================================
( 2 bytes ) Multibyte Repeating XOR Key | ( 2 bytes ) # of Checksums
================================================================
```

```
( Variable ) Repeating key XOR'd Cyclic Checksum CRC32 DWORDs
....
....
....
=====================================================================
( 64 bytes ) RSA 512 SHA-256 Digital Signature (for above fields)
=====================================================================
```

Objectives

Using the included files, you should create a UDP server that handles the following criteria:

1. Verifies the structural integrity of the packet
2. Verify the packet has a valid digital signature

- Failing this, the server should write to a log file in the root directory of the project in a log file named `verification_failures.log`

3. Verifies checksums are being sent correctly

- Failing this, the server should write to a log file in the root directory of the project in a log file named `checksum_failures.log`

4. Introduce an artificial "delay" for writing to the log file, the duration will be passed as a command line argument (in seconds)

Log File Structure

For verification failures, the log format should follow the structure:

```
0x42 (Packet ID - in hex)
3703 (Packet sequence number)
fd2bc562a95c4924d27f9f81de052fbab650f0c2989ee9f4e826244e7c1f0e66 (received hash)
26a4fcaa2167342136272e1d2814b7c73ac995e1229fea8bffa536600cc57921 (expected hash)
```

For checksum failures, the log format should follow the structure:

```
0x42 (Packet ID - in hex)
1109 (Packet sequence number)
1119 (Cyclic checksum iteration)
2165e3dd (received crc32)
2165e24d (expected crc32)
\n (trailing newline)
```

Command Line Arguments

Several command line arguments will be passed at runtime, which your submission must handle:

- `--keys`: a dictionary of `{packet_id: key_file_path}` mappings
  - ex: `--keys '{"0x42": "key.bin", "0x1337": "super_secret_key.bin"}'`
- `--binaries`: a dictionary of `{packet_id: binary_path}` mappings
  - ex: `--binaries '{"0x42": "cat.jpg", "0x1337": "kitten.jpg"}'`
- `-d`: delay, (in seconds) for writing to log files
  - ex: `-d '180'`
- `-p`: port, to receive packets on
  - ex: `-p '1337'`

Things To Note

The sample packet dump contains a stream from only a single packet ID (in this case, `0x42`) corresponding to a single binary (`cat.jpg`). Your server should be capable of handling multiple streams of different {packet id, binary path, key file} combination streams at once.

The packets contain cyclic checksums, to keep track of this pay attention to the packet sequence header. This header starts at 0, and represents the cyclic iteration of the first checksum in it's body. The packet sequence header tells you what iteration the first checksum in its body is on. Packets contain a variable number of checksums (possibly more than one), so the book keeping for this one is on you.

Every field is an unsigned value where appropriate.

You are not expected to handle wrap around (overflow) for the packet sequence number (if it ever occurs), although doing so will definitely earn you some bonus points!

Thank about how you will verify checksums. Note that simply solving the problem with the assumption of the simulated packets is not the goal, the server should work the same if it runs for 2 minutes or 2 days, whether the packets are sent within 50ms or within the span of an hour. Also while in the sample the packet sequences are in the linear range of [0, N), be able to handle out of order sequences and initial packets for up to any (reasonable) number N on the fly!

Your process for calculating cyclic checksums (as well as validation and logging) should not block the main execution of your server.

Evaluation Criteria

We will test your program as is against the include packet sample as well as our suite of test cases (the specification remains the same). Please make sure your submission plays nicely and terminates when asked to.

We will evaluate your submission on:

- General performance, design and algorithmic choices.
- Concurrent packet handling.
- Edge case handling, generalized application to larger, varied data sets.
- Handling binary Python data, nuances of conversion, processing.

NOTE: Please do not publish your solution publicly!

Good luck, and may the Force be with you!

Submission Details

Please submit a `.zip` file containing only this directory at the root. Inside, should be your implementation and write-up for this part of the challenge.

Included Files

- **send.py** - Will send prerecorded UDP packets to `127.0.0.1` on port `1337`. First packet is guaranteed to be correct. Packet IDs are `0x42`.
- **server.py** - The main entry point of your server.
- **payload_dump.bin** - Pickled raw pre-recorded packet data that is sent out (used for simulation testing).
- **key.bin** - Raw binary bytes of RSA 512 bit public key and exponent. Used to verify signature of incoming packets.
- **cat.jpg** - The source binary used for reference against the cyclic checksum values. Corresponds to packet ID `0x42`.
- **requirements.txt** - Your list of all required packages (`pip freeze` format)
- **verify.py** -Will validate that your server correctly logs the first checksum & verification failures.