User and Installation Guide (Group 6)

- Ashley Nguyen
- Christopher Chan
- Ruoyu Li

Requirements

- Ubuntu 17.10
- Languages: Java 8 and Python 3

Install packages

```
$  sudo apt-get install openjdk-8-jdk-headless maven openssl python3 sqlite3
```

About the components

The server, IDS, and client are in the `server/`, `ids/`, and `client/` directories, respectively. The server and client are implemented in Java 8 in the `edu.columbia.group6` package and require maven to install dependencies. These dependencies include:

- Sqlite JDBC driver
- Apache commons (to ease digest and base64 handling)

The IDS does not have dependencies beyond the standard installation provided by Ubuntu 17.10's python3.

Data between the client and server is sent via a simple text-based protocol based on HTTP and FTP. File data (not headers) sent between the client and server is base64 encoded to ease parsing of sent data.

Installation and Usage

The IDS runs on port 9999 for clients and proxies all connections between the client and server.

Note:

- You can only have one connection open between a client and server at one time.

Server (Java 8) and IDS (Python 3)

1. Navigate to server directory (`server/`)

2. Fetch dependencies and compile the server. This will also automatically create the document root where files are stored.
   ```
   $ mvn install
   ```
3. Start the server
   ```
   $ java -jar target/server.jar --port 10000
   ```
4. Navigate to the IDS directory (`ids/`)
5. Start the IDS
   ```
   $ python3 ids.py [pattern file] [listen port] [server port] [log file]
   ```
   or
   ```
   $ python3 ids.py pattern_config.txt 9999 10000 log.txt
   ```

Client

1. Navigate to the client directory (`client/`)
2. Fetch dependencies and compile the server
   ```
   $ mvn install
   ```
3. Start the server
   ```
   $ java -jar target/client.jar [ip address] [port]
   # For example:
   $ java -jar target/client.jar 127.0.0.1 9999
   ```

Purpose

The Purpose of this program is to create a simple FTP server which will allow the client to send and retrieving files from it. Each request will be passed through to an intrusion detection system(IDS) which is check to make sure all of the information passed is valid. Once it has been inspected in the IDS it will then be sent to its destination.

The client supports 4 commands: `get`, `put`, `ls`, and `exit`.

- **get** – Usage: `get [File]`
  The `get` command is used to send a string containing the name of a file to the IDS. If everything is working correctly, the IDS will forward the request to the server who will then send back a hash file along with the file name requested. The client will then proceed to hash the file using SHA256 and compare it to the hash value it was sent. If the two values are the same, it will write the contents to a file. It is base64 encoded.

- **put** – Usage: `put [File] [FileName]`
  The `put` command is used by the client to send a file as well as the hash value to the server. Once the client calls the command, it will be send to the IDS which will inspect it and forward the information to the server if it is complete. The server will save the file.

- **ls** – Usage: `ls`
  The ls command is used to list all of the files on the server's directory. It does not take any additional arguments and will return a list of files present in the server's directory. An example of using `ls` will be

  ```
  hello.txt
  goodbye.txt
  greetings.txt
  columbia.jpeg
  file
  ```

- **exit** – Usage: `exit`
  The `exit` command is used to exit the program. This does not shut down the server; to shut the server down, use `Ctrl+c` in the server CLI.

IDS

The purpose of the IDS is to check the information travelling through from the client to the server and look for signatures. Those signatures will then be compared to a given configuration file and if it matches, the packet will be dropped. The IDS will be able to log that information including the IP address from which it came as well as the timestamp. However there are certain restrictions as to what the IDS can check for.

- Maximum of 32 bytes
- Hexadecimal Notation
- The IDS uses two threads to handle the transmission from to client to server and from server to client respectively.
- The max buffer size is 1380 since MTU is 1500 and the maximum of IP header and TCP header is both 60 bytes, so `1500 - 60 * 2 = 1380`.
- All the patterns in the default pattern configuration file are generated by random bytes.
- The IDS only detects patterns in each packet, ie, data in every max buffer size, but won't do any other format checking that the server should take care of.
- The IDS will close the socket if both client and server close their socket, or you can use `Ctrl+c` to shut it down.

Server

The Server side of the program must be the first one started in order for the connections to work and acts as a way to process the client's requests. The server is able to get requests from the client that have been passed through the IDS and send back the corresponding files. In addition to the files themselves, the server also stores the hash files which will be sent with when requested to compare ensure the file's integrity which checked against the file. It simply acts as a place of storage for the files created by the client and is able to retrieve it whenever it is necessary.

- Files are stored in the `docroot` directory.
- Will only accept file names and will reject any path requests that are sent.
- Use `Ctrl+c` to shut down
- Files up to 10MB have been tested and it's possible the server supports larger files
- Hashes and filenames are stored in a sqlite database in the `sqlite` directory.

Client

The Client side of the program is responsible for populating the server with files using the PUT command as well as requesting the files from the server using the GET command. All of its information is passed through the IDS and it receives the files back from the server.

- Before sending the files the client will hash it using SHA-256 and send both the hash and the file to the server
- It will check to make sure the commands used are valid and send back an error if not
- It will continuously check for the input commands until EXIT is called
- Files received will be hashed with SHA-256 and compared with the hash sent from the server to ensure integrity.
- It will be able to see the list of files contained on the server by calling the LS command
- Checks for the status code and responds in the appropriate manner.

Protocol design

- All content (files) is base64-encoded to support binary content.
- Data is transmitted in the UTF-8 Unicode character set.

Client request

The first line is the command, followed by options specific to that command. Then, there are two line breaks and the content is sent. The client is done when a NUL or EOF character is sent. For instance, the client sends:

```
{COMMAND} [options]


{content}
```

Server response

The server responds with a code to signify the status of the request. The code is then followed by the command that was requested and then the response specific to the command.

```
{STATUS CODE}
Request: {COMMAND} [options]
Hash: (for get requests)

{content}
```

The status codes are a subset of FTP status codes:

- `250`: successful put, get, or ls command
- `501`: syntax error
- `502`: command not recognized
- `503`: bad sequence
- `550`: file not found
- `552`: file too big

Changelog

Changes after review (2018-04-13)

- Documentation
    - Correct client start parameters
    - Correct case of client commands
    - Clarify how to shutdown the server
- Server code changes
    - Add creation of docroot to `$ mvn install`
- Client code changes
    - Change uppercase commands to lowercase
- IDS code changes
    - Add support for specifying port numbers
    - Improve error handling and exception resilience