COMS 4182 Project Part 1

Overview

Each group will be building a fuzzer using the scapy python library in part 1 of the project. In part 2 of the project, each group will evaluate and slightly modify another group's project.

Part 1 will account for 65% of the project grade. The grade for part 1 will be based on input from the testing/evaluation performed by another group and evaluation by the professor. Both the code and documentation will be evaluated. Important: **There will be no opportunity to fix bugs during part 2**. **What you provide to another group to evaluate must be a finished product.** The code (functionality, # of bugs, error handling, comments, ease of modifying/organization) will be 2/3rds of the part 1 points and the documentation (clarity, correctness/matching what the code does, installation working) will be 1/3rd of the part 1 points.

In part 2 of the project, another group will be installing, testing and altering your fuzzer. The other group will be evaluating the code on how well it functions and handles errors, and evaluating the documentation on the clarity and accuracy of the instructions, and evaluating how well the code is commented. The other group must be able to install and use your code without additional information from your group.

Part 2 will account for 35% of the project grade. An evaluation report will be written and given to the group whose project you evaluated. The testing and code modification your group performs and the report your group writes will determine your group's grade for part 2. The completeness and clarity of the report will be evaluated by the group whose receives the report and the professor. Details for part 2 will be provided when each group hands off its project to another group.

- Each group will build a fuzzer that is capable of testing the IP layer (IPv4 only), the TCP transport layer and the application layer. You may use scapy 3.0 for python 3 or 2.X for python 2.
- Each group will provide an installation and a user guide (each in pdf format). The installation guide will describe how to install your fuzzer from scratch on a new VM. The user guide will describe , how to use the fuzzer. You may provide an installation script. Points will not be deducted for having to manually perform multiple steps for the installation in place of a script.

Schedule On-Campus Students for Part 1:

CNV students refer to the CVN section at the end.

- March 27th 10pm: submission for part 1 is due in Courseworks. Have one member of the group upload the submission.
- March 29th in class: Each group will hand-off their code and documentation to another group. The hand-off will include a quick demonstration of the code and overview of the installation instructions.

Environment

Use Ubuntu 18.x VMs as the client on which the fuzzer runs and as the server to which the client sends packets.

Do not run the client and server on the same VM and use the loop back interface. Implementations that do not work with the client and server on separate VMs will not receive any credit.

Fuzzer Functionality

The following describes the general functionality required. You will need to make design decisions and work out details on how to implement the fuzzer.

The fuzzer will run on a client and the packets sent to a server. The fuzzer must be capable of receiving responses from the server.

- The source and destination IP addresses and ports will be valid addresses and ports in all tests. In the description below, any fields being fuzzed excludes these 4 fields.
- The user will be able to specify the source and destination IP addresses and destination port.
- The checksum is populated as the packet is being sent and this is not controlled by scapy, thus checksums will not be fuzzed.
- In the default tests for both the IP and transport layer, if you believe it is not feasible to test all possible values for a single field, the fuzzer should try a user specified number of random values.

IP Layer:

When testing the IP layer, the transport layer will be TCP and contain valid values in all fields. The fuzzer should provide the following capabilities for fuzzing the IP header:

- Running a default set of tests where each field is fuzzed individually, with all possible values tested for each field. The user is able to specify what field to fuzz or to say run through tests on all fields.
- Running a set of tests where the content of the fields is read in as hex from a file. The user must be able to specify the content for multiple tests in the same file. You must decide how the user is to specify the content for each field in the file. For example, one line of the file may be a series of tag value pairs to indicate the field and contents and each line corresponds to a separate packet the fuzzer will generate. The file must be easy to create and edit. For example, do not require that the user type the contents in json format into the file. You may set a maximum on how many tests can be specified in a single file.
- The fuzzer will include a small default payload in the packet. The payload must be in a file and can be edited by the user.

Transport Layer (TCP only):

When testing the transport layer, the IP layer will contain valid values in all fields. The fuzzer will provide the following capabilities for fuzzing the TCP transport header:

- Running a default set of tests where each field is fuzzed individually, with all possible values tested for each field. The user is able to specify what field to fuzz or to say run through tests on all fields.
- Running a set of tests where the content of the fields is read in as hex from a file. The user must be able to specify the content for multiple tests in the same file. You must decide how the user is to specify the content for each field in the file. For example, one line of the file may be a series of tag value pairs to indicate the field and contents and each line corresponds to a separate packet the fuzzer will generate. The file must be easy to create and edit. For example, do not require that the user type the contents in json format into the file. You may set a maximum on how many tests can be specified in a single file.

- The fuzzer will include a small default payload in the packet. The payload must be in a file and can be edited by the user.

Application Layer:

When testing the application layer (the packet payload), the IP and transport layers will be valid. The application layer for this project will just be bytes to send to the server. The fuzzer will provide the following capabilities for fuzzing the application layer:

- Running a default set of tests where random payloads are sent to the server. The user will be able to specify the number of tests to run and whether to use a fixed payload size or vary the payload size. The user will specify the number of bytes to include in the case of a fixed payload size and specify a range for the payload size in the case of a variable payload size.
- Running a set of tests where the payload is read in as hex from a file. The user must be able to specify the content for multiple tests in the same file. You must decide how the user is to specify the content in the file. For example, each line of the file may be a separate test. You may set a maximum on how many tests can be specified in a single file.
- Each test will be a single packet thus the payload must fit into a single packet.
- The fuzzer will process the response from the server (refer to the server description below for what is in a response) and maintain a count of how many packets sent to the server where viewed as valid by the server and how many were viewed as invalid. When the fuzzer is done with the tests, it will display the total number of tests, the valid count and the invalid count to the user.

Server:

You will create a server application to which the client will send the packets. You will use the server for testing when building your fuzzer and for testing the fuzzer you are given to evaluate in part 2 of the project.

The server only processes the application layer of packets. You will use a normal socket library and are not parsing the IP and transport layers. Thus with invalid IP layers and transport layers, the server application may not receive the packet and does not send a response to the server. The VM may send a response. For example when fuzzing the TCP header, if a flag is invalid, the relevant next message in the handshake (to continue or reset) may be sent and the fuzzer should receive it.

When the server application receives a packet, it will inspect the payload for a user specified a series of bytes (refer to this as the *pattern*) at the start of the payload. The pattern will be specified in hex in a file and read in when the server starts. The pattern may not exceed the maximum allowed payload length. Any payload beginning with the pattern is considered valid. The server will send the client a response with a payload containing the byte 0x00 if the payload is valid and 0xff if it is invalid. The server will maintain a count of the number of valid payloads and the number of invalid payloads received.

When the server is done receiving packets or is stopped, it will either display valid and invalid counts to the user or write them to a file before exiting.

Error Handling:

You are responsible for identifying possible errors that may occur due to the user given invalid input when starting the fuzzer and/or invalid content in an input file and properly handling errors. Any runtime error must also be handled by printing an appropriate message and exiting nicely.

CVN Students:

You will omit the TCP layer fuzzing portion of the project and have a few extra days to complete part 1.

Schedule: Part 1 is due on Wednesday April 3rd by 6pm Eastern DST. Upload your submission to courseworks. You will receive another student's project before Friday April 5th to evaluate for part 2 of the project. Your part 2 will be a subset of what the on-campus students are given, but you will have one less week to do part 2.