fuzzerwuzzer

Fuzz testing IP/TCP/App

Table of Contents

Setup

Prerequisites

Tool	Description
Python3	https://www.python.org/downloads/
Scapy	https://scapy.readthedocs.io/en/latest/installation.html

The recommended version of python is 3.6.7+. The only library that is needed that is not provided with python is scapy. That being said you should install all of the necessary prerequisites for scapy.

```
sudo apt-get install tcpdump graphviz imagemagick python-matplotlib python-
cryptography python-pyx python3-pip python3-scapy
sudo pip3 install -r requirements.txt
```

Configuring OS

All testings and developing was done on Ubuntu 18.04 so the code is not guaranteed to run on other environments. All testing was done with Python 3.6.7 and lower version have not been tested.

Since scapy functions not within the kernel, for it to be able to establish TCP connections we must disable the kernel from sending RST packets. To do this, we can modify iptables to drop RST packets.

```
sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
iptables -L
```

Note: You will have to modify the iptables every time you restart your OS unless you save your changes permentantly. I would not recommend dropping all RST packets when the fuzzer is not running.

High Level Usage

```
sudo python3 main.py [--sourceIP SOURCEIP] [--targetPort TARGETPORT]
                     [--defaultPayloadPath DEFAULTPAYLOADPATH]
                     targetIP  {app-rand-fixed,app-rand-range,app-file,ip}
```

The command line arguments are broken up into general parameters and positional arguments that are used to fuzz the IP and application layers. Note that you must run the fuzzer with sudo since the underlying scapy library needs root access to manipulate and send packets on the network without relying on the kernel to construct packets.

`--targetPort` is an optional argument to specify the port of the server you are fuzzing. By default this value is set to 80 unless specified otherwise

`targetIP` is a required argument of the target server that will be fuzzed.

`--sourceIP` argument is optional and by default the fuzzer will use the IP of the machine that it is running on.

The positional arguments are described in the sections below.

Fuzzing IP Layer

Default Tests and Methodology

When sending a packet to fuzz the ip layer, regardless of the protocol field the payload will contain valid TCP payload as well as a default data payload. A single packet is sent to the destination server containing a TCP SYN packet. The fuzzer does not wait for a response from the server.

```
sudo python main.py --targetPort <port> <ip> ip -h
usage: FuzzerWuzzer targetIP ip [-h] [--defaultPayloadPath DEFAULTPAYLOADPATH]
                                [--fall] [--fversion] [--fihl] [--fdscp]
                                [--fecn] [--flen] [--fid] [--fflags] [--ffrag]
                                [--fttl] [--fproto]
```

```
optional arguments:
  -h, --help            show this help message and exit
  --defaultPayloadPath  DEFAULTPAYLOADPATH
                        Path to the payload that will be sent to the server
                        with each request
  --fall                Will fuzz the all fields in IP header
  --fversion            Will fuzz the version field in IP header
  --fihl                Will fuzz the IHL field in IP header
  --fdscp               Will fuzz the DSCP field in IP header
  --fecn                Will fuzz the ECN field in IP header
  --flen                Will fuzz the Length field in IP header
  --fid                 Will fuzz the ID field in IP header
  --fflags              Will fuzz the Flags flags in IP header
  --ffrag               Will fuzz the Fragment offset field in IP header
  --fttl                Will fuzz the TTL field in IP header
  --fproto              Will fuzz the protocol field in IP header
```

Methodology

`--defaultPayloadPath` specifies the relative or absolute path to the file that contains a payload to send with each ip packet. When not specified, the default path *./IP_Settings/default_payload* is used. You may either change the existing file or use the argument to specify your own path. The payload is the ascii representation of the payload to send.

`--fall` runs all tests and fuzzes all of the ip layer fields one by one.

`--fversion` 16 packets are sent for each of the possible inputs 0-15 for the version header field

`--fihl` 16 packets are sent for each of the possible inputs 0-15 for the IHL header field

`--fdscp` 64 packets are sent for each of the possible inputs 0-63 for the dscp header field

`--fecn` 4 packets are sent for each of the possible inputs 0-3 for the ecn header field

`--flen` 256 packets are sent by randomly selecting values from the range 0-65535 for the len field

`--fid` 256 packets are sent by randomly selecting values from the range 0-65535 for the id field

`--fflags` 8 packets are sent for each of the possible inputs 0-7 for the flags field

`--ffrag` 256 packets are sent by randomly selecting values from the range 0-8191 for the fragment offset field

`--fttl` 256 packets are sent for each of the possible inputs 0-255 for the ttl field

`--fproto` 256 packets are sent for each of the possible inputs 0-255 for the protocol field

Tests From a File

File Format

Example:

```
version:4 ihl:10 dscp:1 ecn:0 len:50 id:1 flags:0 frag:6543 proto:6
version:4 ihl:10 dscp:1 ecn:0 len:50 id:1 flags:2 frag:0 ttl:30
```

The test file consists of one test per line. Each line is parsed individually and each parameter is validated. If a specific line contains a parameter that is not valid, that parameter is ignored and a default value is used instead. Any parameters that are not specified, the default value for that value will be used. In the example above, since the first line doesn't have an override for ttl, the default value of 54 is used. There is no limit to how many tests can be specified in the file but since python has memory limits when opening and reading files, large files might require adjustments to the python interpreter memory limits. Please refer to the documentation of the default IP tests for valid ranges for individual fields. An example file is included in the IP_Settings folder.

Default values:

```
version = 4
ihl = 0
dscp = 1
ecn = 0
len = 50
id = 1
flags = 0
frag = 0
ttl = 54
proto = 6
```

Usage

```
sudo python main.py --targetPort <port> <ip> ip-file -h
usage: FuzzerWuzzer targetIP ip-file [-h]
                                     [--defaultPayloadPath DEFAULTPAYLOADPATH]
                                     path

positional arguments:
  path                  Path to the file that specifies ip packets to send to
                        the server

optional arguments:
  -h, --help            show this help message and exit
  --defaultPayloadPath DEFAULTPAYLOADPATH
                        Path to the payload that will be sent to the server
                        with each request
```

`--defaultPayloadPath` specifies the relative or absolute path to the file that contains a payload to send with each ip packet. When not specified, the default path *./IP_Settings/default_payload* is used. You may either change the existing file or use the argument to specify your own path. The payload is the ascii representation of the payload to send.

`path` is the path to the file which contains the user specified test cases in the file format described in the file format section.

Fuzzing Application Layer

Application layer fuzzing is done by first establishing a TCP connection with the target server, and then sending payloads to the server and observing the responses. The fuzzer expects to receive either a x00 (valid) or xFF (invalid) response from the server and counts the occurrences of both. After all of the payloads have been sent to the server, the TCP connection is closed and the number of valid, invalid and other responses is printed to stdout. Payloads can either be generated randomly by the fuzzer or specified via arguments and their usage is detailed below.

Random Payloads

The results of the tests are written to the fuzz_tests folder with the unix timestamp when the tests were ran. Since the random values are generated by python secret library which doesn't accept a seed, you can replay a random test by reusing the file that is generated.

```
usage: FuzzerWuzzer targetIP app-rand-fixed [-h] numTests payloadSize

positional arguments:
  numTests     Number of tests to run
  payloadSize  The size of the fixed payload to include in each packet


usage: FuzzerWuzzer targetIP app-rand-range [-h]
                                         numTests payloadMinSize
                                         payloadMaxSize
positional arguments:
  numTests        Number of tests to run
  payloadMinSize  The min size of the fixed payload to include in each packet
  payloadMaxSize  The max size of the fixed payload to include in each packet
```

Random payload testing is provided by 2 positional arguments `app-rand-fixed` and `app-rand-range` to send either fixed size random payloads or a range of sizes respectively. Both require the user to provide the number of random payloads. Depending on if the fixed or the range version of the parameter is used, the following also need to be specified

`numTests` number of random payloads to generate and send

`payloadSize` used for fixed size random inputs. Maximum is 1024 bytes

`payloadMinSize` used for variable sized random inputs. Sets the minimum size the packet can be

`payloadMaxSize` used for variable sized random inputs. Sets the maximum size the packet can be. Maximum is 1024 Bytes

Payloads From a File

```
usage: FuzzerWuzzer targetIP app-file [-h] path

positional arguments:
  path        Path to the file which contains tests to run. Each line should
              contain the hex string representing bytes to send in a packet
```

The format of the file contains a payload per line. The payload is specified by an hex representation of bytes using human readable ascii. You can run a random test first and see the file the is generated for an example. If a payload is malformed (odd length or invalid hex) a message will be printed and the payload will be ignored. After payloads are sent the number of valid (x00), invalid (xFF) and other bytes the server responded with is totalled and displayed.

Example File:

```
0636ad972d33b4
1de410d494d0bc
24ee8ca554256c
72a38ca554256c
3effe8b692884e
e52453f381e792
72a3ad972d33b4
72a353f381e792
```

Included Server

The included server serves as an endpoint for the client to connect to and send packets. It uses a higher level socket library that takes care of the TCP connection and is primarily used to read payloads from the socket. The maximum number of bytes that the server will read from the pipe 1024 so any payload larger than this will not be parsed correctly for matching.

Validation of Pattern

The server reads the *matched_pattern* file for pattern to match against the bytes that it receives from the socket. The bytes are specified in the file by human readable HEX, i.e. a file containing the characters DEADC0DE will translate to the byte values 0xDEADCODE inside the server. The number of bytes that this pattern can be is 1024 which is the same as the size of the maximum payload that the server will process. A payload is considered valid if the initial bytes of the payload are the same as the pattern specified in the file. Else the payload is considered invalid. Note that the payload must match the entire pattern, if the payload is shorter than the pattern it is considered invalid.

The server keeps count of the number of valid and invalid payloads and will print out the accumulate counts when the server is terminated.