An NCC Group Publication

Fuzzing the easy way, using Zulu

Prepared by:
Andy Davis
Research Director
andy.davis ‘at’ nccgroup ‘dot’ com

Contents

1 Introduction 3
2 Tutorial One: Zulu basics 4
3 Tutorial Two: Proxy-based network fuzzing 6
4 Tutorial Three: Providing input using a PCAP file 13
5 Tutorial Four: File fuzzing 16
6 Tutorial Five: Using the USB fuzzer module 19
7 Tutorial Six: Using the Serial fuzzer module 24
8 Tutorial Seven: Integrating with Wireshark 28
9 Tutorial Eight: Integrating with VMware 30
10 Tutorial Nine: Adding length fields 31
11 Tutorial Ten: Configuring email notification settings 33
12 Tutorial Eleven: Writing and using ZuluScript 35

1 Introduction

There are many fuzzers and fuzzing frameworks available to test everything from files, to network protocols, to interface technologies. None of them are perfect, but they all offer a range of different capabilities to different user bases. The motivations behind the Zulu fuzzer lie in the rapid prototyping that normally happens on a client engagement where something needs to be fuzzed within tight timescales. Many of us will have created bespoke fuzzing scripts time and time again on client sites, maybe tweaking and changing these scripts as requirements change from job to job. There are various fuzzing frameworks available that provide potential solutions to this problem, but they often have quite a steep learning curve associated with them and therefore, are not always best suited to getting a fuzzer up and running quickly and easily. Also, the ability to see and manipulate the data graphically is attractive to many users, as purely command line-based tools can be intimidating to some.

Zulu is an interactive GUI-based fuzzer. It is as much as possible, input and output-agnostic so once you are happy with using the fuzzing engine that's driven by the GUI you are only limited by the input and output modules that have been developed for it. It is written purely in Python and has been released under AGPL on the NCC Group Github page:

https://github.com/nccgroup/zulu

This paper serves as an introduction to using Zulu and includes a number of tutorials explaining how to use the different features within the tool. The tutorials have been written with minimal duplication and therefore they are intended to be read in order.

2 Tutorial One: Zulu basics

Before we discuss the details of specific fuzzing sessions its worth spending some time looking at the GUI and all its elements, in addition to understanding the general workflow of the tool.

The GUI is split into a number of areas:

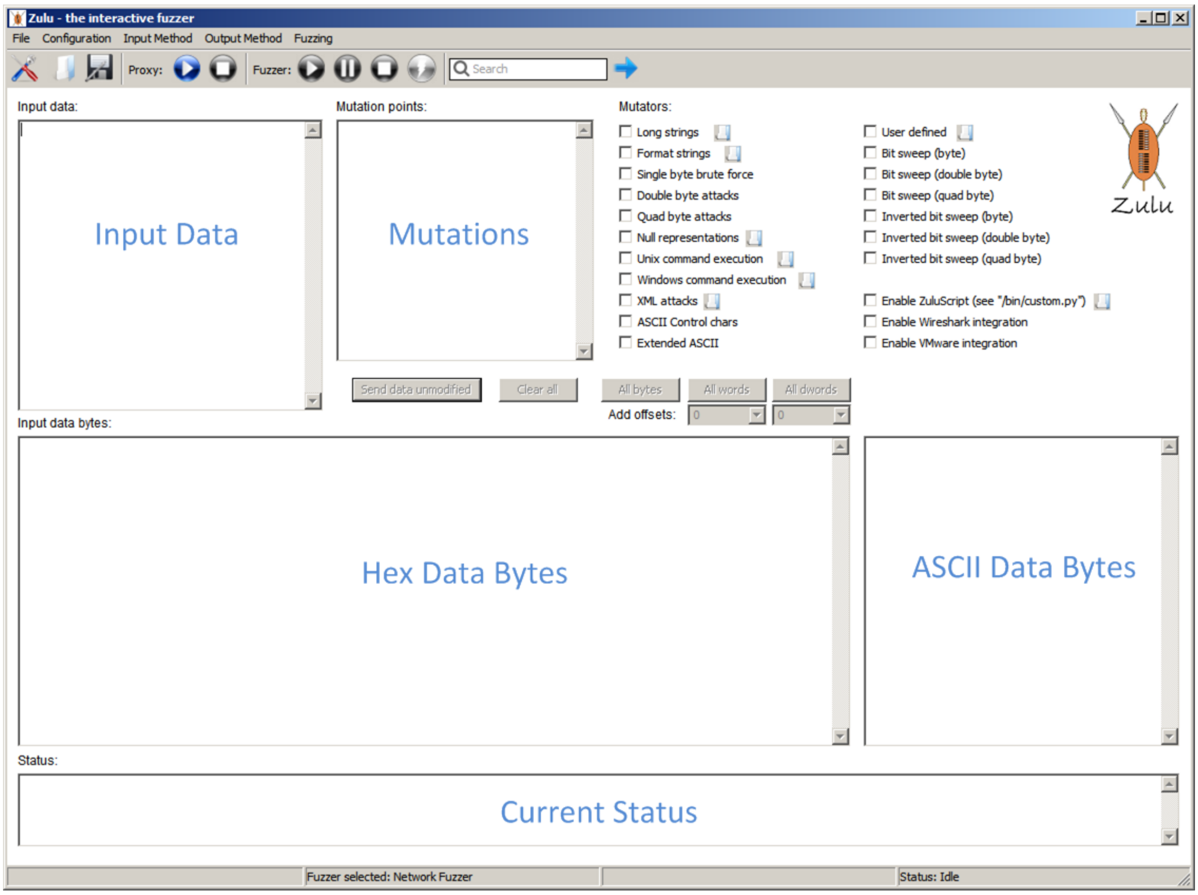
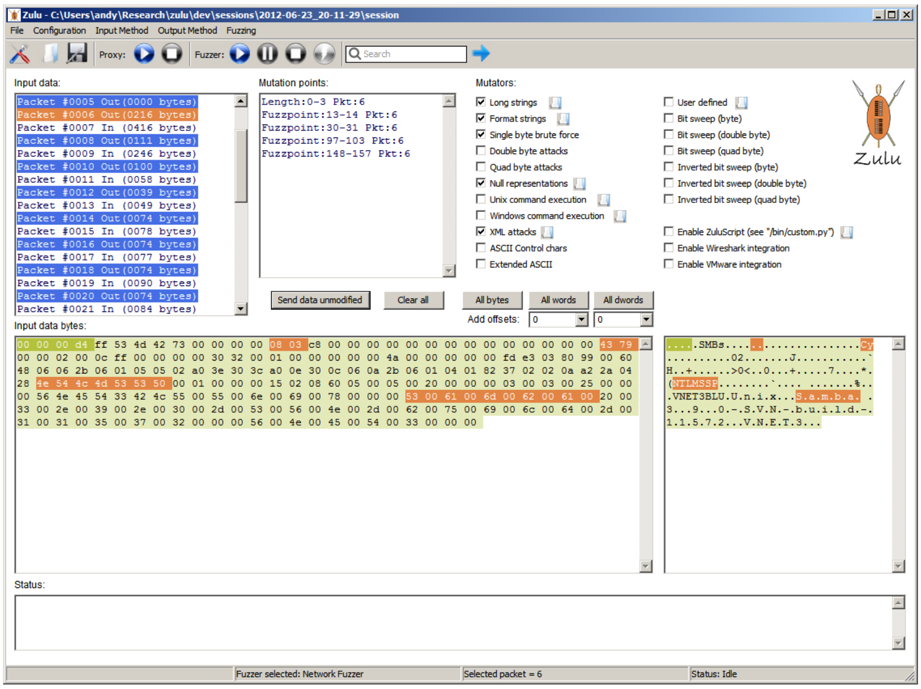


Figure 1: The Zulu GUI layout

- **Input Data** - This is where separate packets (both input and output) from a network capture are displayed. If the input is a single file then this will be displayed as a single entry.
- **Mutations** - In this section details of fuzzpoints, length fields and packets to be sent unmodified are displayed
- **Hex data bytes** - The hex data associated with the currently selected Input Data entry is displayed
- **ASCII data bytes** - The ASCII data associated with the currently selected Input Data entry is displayed
- **Current Status** - General status updates as to what Zulu is currently doing

Below is a screenshot to demonstrate the type of information likely to be present in each section of the GUI:



Input data:

```
Packet #0005 Out(0000 bytes)
Packet #0006 Out(0216 bytes)
Packet #0007 In  (0416 bytes)
Packet #0008 Out(0111 bytes)
Packet #0009 In  (0246 bytes)
Packet #0010 Out(0100 bytes)
Packet #0011 In  (0058 bytes)
Packet #0012 Out(0039 bytes)
Packet #0013 In  (0049 bytes)
Packet #0014 Out(0074 bytes)
Packet #0015 In  (0078 bytes)
Packet #0016 Out(0074 bytes)
Packet #0017 In  (0077 bytes)
Packet #0018 Out(0074 bytes)
Packet #0019 In  (0090 bytes)
Packet #0020 Out(0074 bytes)
Packet #0021 In  (0084 bytes)
```

Mutation points:

```
Length:0-3 Pkt:6
Fuzzpoint:13-14 Pkt:6
Fuzzpoint:30-31 Pkt:6
Fuzzpoint:97-103 Pkt:6
Fuzzpoint:148-157 Pkt:6
```

Mutators:

- ☑ Long strings
- ☑ Format strings
- ☑ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- ☑ Null representations
- □ Unix command execution
- □ Windows command execution
- ☑ XML attacks
- □ ASCII Control chars
- □ Extended ASCII
- □ User defined
- □ Bit sweep (byte)
- □ Bit sweep (double byte)
- □ Bit sweep (quad byte)
- □ Inverted bit sweep (byte)
- □ Inverted bit sweep (double byte)
- □ Inverted bit sweep (quad byte)
- □ Enable ZuluScript (see "/bin/custom.py")
- □ Enable Wireshark integration
- □ Enable VMware integration

Add offsets: 0 0

Input data bytes:

```
00 00 00 d4 ff 53 4d 42 73 00 00 00 00 08 03 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 43 79
00 00 02 00 0c ff 00 00 00 00 30 32 00 01 00 00 00 00 00 4a 00 00 00 00 00 fd e3 03 80 99 00 60
48 06 06 2b 06 01 05 05 02 a0 3e 30 3c a0 0e 30 0c 06 0a 2b 06 01 04 01 82 37 02 02 0a a2 2a 04
28 4e 54 4c 4d 53 53 50 00 01 00 00 00 15 02 08 60 05 00 05 00 20 00 00 00 03 00 03 00 25 00 00
00 56 4e 45 54 33 42 4c 55 00 55 00 6e 00 69 00 78 00 00 00 53 00 61 00 6d 00 62 00 61 00 20 00
33 00 2e 00 39 00 2e 00 30 00 2d 00 53 00 56 00 4e 00 2d 00 62 00 75 00 69 00 6c 00 64 00 2d 00
31 00 31 00 35 00 37 00 32 00 00 00 56 00 4e 00 45 00 54 00 33 00 00 00
```

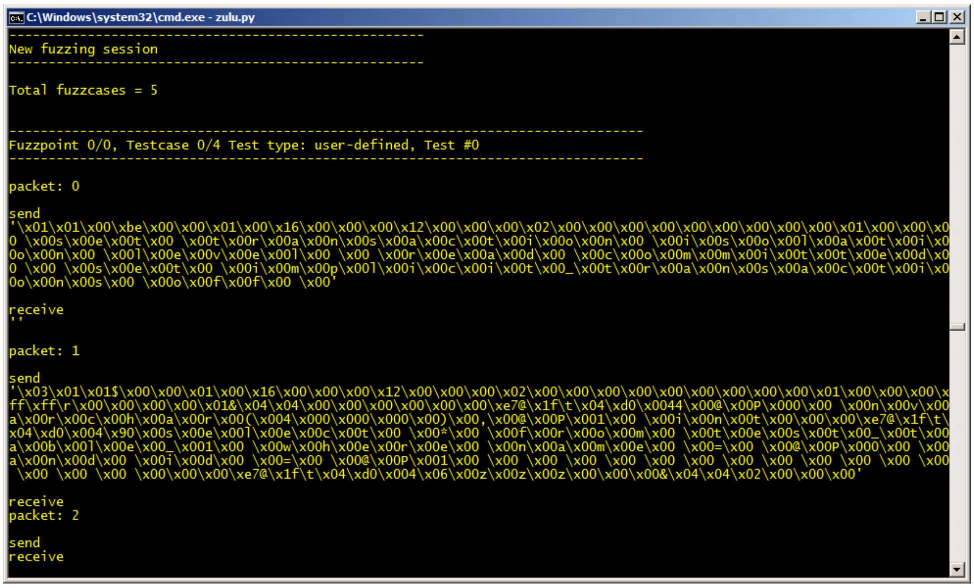
```
.....SMBs.......................Cy
..........02.......J...........`
H..+......>0<..0...+.....7....*.
(NTLMSSP........`.... .......%..
.VNET3BLU.U.n.i.x...S.a.m.b.a. .
3...9...0.-.S.V.N.-.b.u.i.l.d.-.
1.1.5.7.2...V.N.E.T.3...
```

Status:

Fuzzer selected: Network Fuzzer | Selected packet = 6 | Status: Idle

Figure 2: The Zulu GUI, populated with data

The majority of the large-volume output is displayed in the command-line console (and also logged to a file):



```
------------------------------------------------------
New fuzzing session
------------------------------------------------------

Total fuzzcases = 5


----------------------------------------------------------------------------------
Fuzzpoint 0/0, Testcase 0/4 Test type: user-defined, Test #0
----------------------------------------------------------------------------------

packet: 0

send
'\x01\x01\x00\xbe\x00\x00\x01\x00\x16\x00\x00\x00\x12\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x0
0 \x00s\x00e\x00t\x00 \x00t\x00r\x00a\x00n\x00s\x00a\x00c\x00t\x00i\x00o\x00n\x00 \x00i\x00s\x00o\x00l\x00a\x00t\x00i\x0
0o\x00n\x00 \x00l\x00e\x00v\x00e\x00l\x00 \x00 \x00r\x00e\x00a\x00d\x00 \x00c\x00o\x00m\x00m\x00i\x00t\x00t\x00e\x00d\x0
0 \x00 \x00s\x00e\x00t\x00 \x00i\x00m\x00p\x00l\x00i\x00c\x00i\x00t\x00_\x00t\x00r\x00a\x00n\x00s\x00a\x00c\x00t\x00i\x0
0o\x00n\x00s\x00 \x00o\x00f\x00f\x00 \x00'

receive
''

packet: 1

send
'\x03\x01\x01$\x00\x00\x01\x00\x16\x00\x00\x00\x12\x00\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x
ff\xff\r\x00\x00\x00\x00\x01&\x04\x04\x00\x00\x00\x00\x00\x00\xe7@\x1f\t\x04\xd0\x0044\x00@\x00P\x000\x00 \x00n\x00v\x00
a\x00r\x00c\x00h\x00a\x00r\x00(\x004\x000\x000\x000\x00)\x00,\x00@\x00P\x001\x00 \x00i\x00n\x00t\x00\x00\x00\xe7@\x1f\t\
x04\xd0\x004\x90\x00s\x00e\x00l\x00e\x00c\x00t\x00 \x00*\x00 \x00f\x00r\x00o\x00m\x00 \x00t\x00e\x00s\x00t\x00_\x00t\x00
a\x00b\x00l\x00e\x00_\x001\x00 \x00w\x00h\x00e\x00r\x00e\x00 \x00n\x00a\x00m\x00e\x00 \x00=\x00 \x00@\x00P\x000\x00 \x00
a\x00n\x00d\x00 \x00i\x00d\x00 \x00=\x00 \x00@\x00P\x001\x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00
 \x00 \x00 \x00 \x00\x00\x00\xe7@\x1f\t\x04\xd0\x004\x06\x00z\x00z\x00z\x00\x00\x00&\x04\x04\x02\x00\x00\x00'

receive
packet: 2

send
receive
```

Figure 3: The Zulu console

The following illustrates the Zulu file structure:

- **/bin** - Zulu binaries and custom.py (ZuluScript Python)
- **/crashfiles** - When file fuzzing, files that have caused the target to crash are placed in here
- **/fuzzdb** - the fuzzer testcase files (these can all be modified, but there is a user-defined.txt file which should ideally be used for custom testcases
- **/images** - images used by the GUI
- **/logs** - log files
- **/pcap** - when Wireshark integration is enabled, auto-generated PCAP files are placed in here
- **/PoC** - when a crash occurs and a PoC is auto-generated, it is placed in here (and emailed to the user if the email settings have been configured)
- **/sessions** - all the configuration options and captured packets can be saved in a session file
- **/tempfiles** - when file fuzzing, manipulated files are temporarily placed in this directory
- **/templates** - the template used to generate the PoC files is in here

3 Tutorial Two: Proxy-based network fuzzing

In this tutorial we will capture packets from a Windows Remote Desktop session via a proxy and then start fuzzing some of the data. First, you need to set the proxy settings, which can be found in the **Configuration** menu:

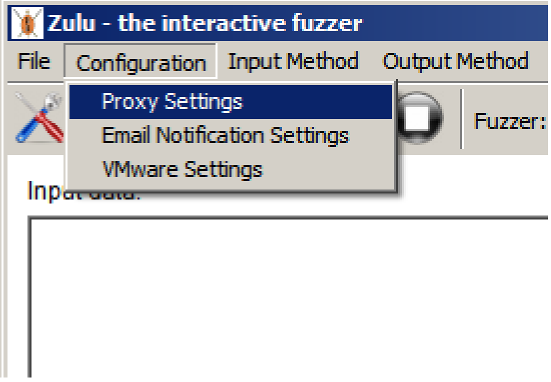
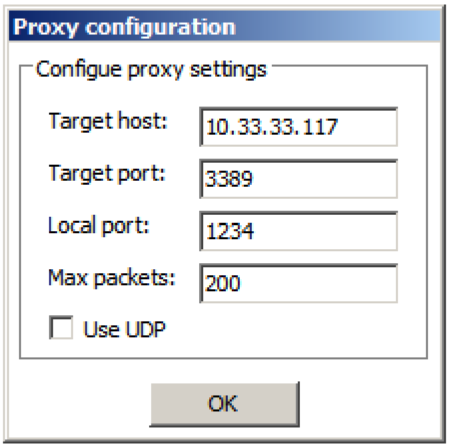


Figure 4: Selecting proxy settings

This will bring up the **Proxy configuration window**:



Target host: 10.33.33.117
Target port: 3389
Local port: 1234
Max packets: 200
□ Use UDP

Figure 5: The proxy configuration window

- **Target host** - the target host on which the network service is listening
- **Target port** - the port associated with the target service
- **Local port** - the local port used for the proxy
- **Max packets** - the maximum number of packets to capture
- **Use UDP** - if the network service uses UDP instead of TCP, select this checkbox

Next, select **Input Method > Start Network Capture**:

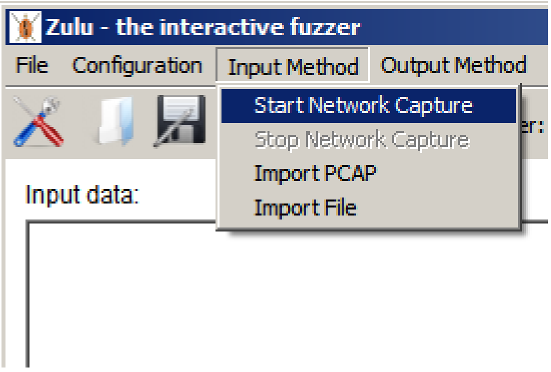
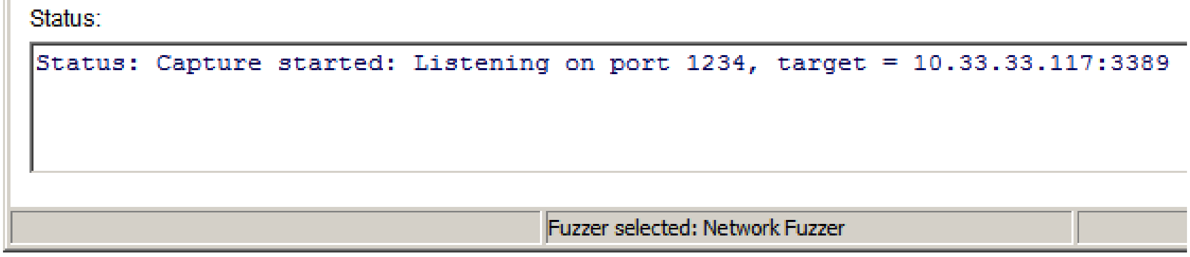


Figure 6: Starting the network capture

The status is now updated to show that the proxy is running:

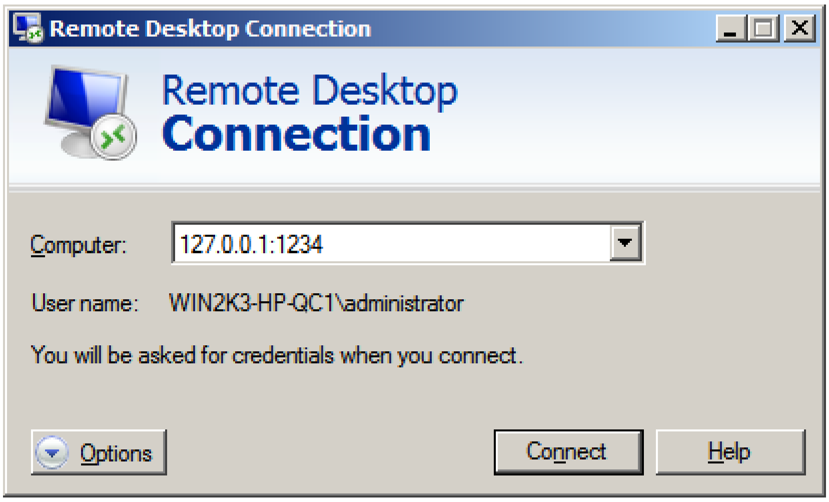

Status:

```
Status: Capture started: Listening on port 1234, target = 10.33.33.117:3389
```

Fuzzer selected: Network Fuzzer

Figure 7: Network capture status

Now configure your client to connect to the service, via the proxy:



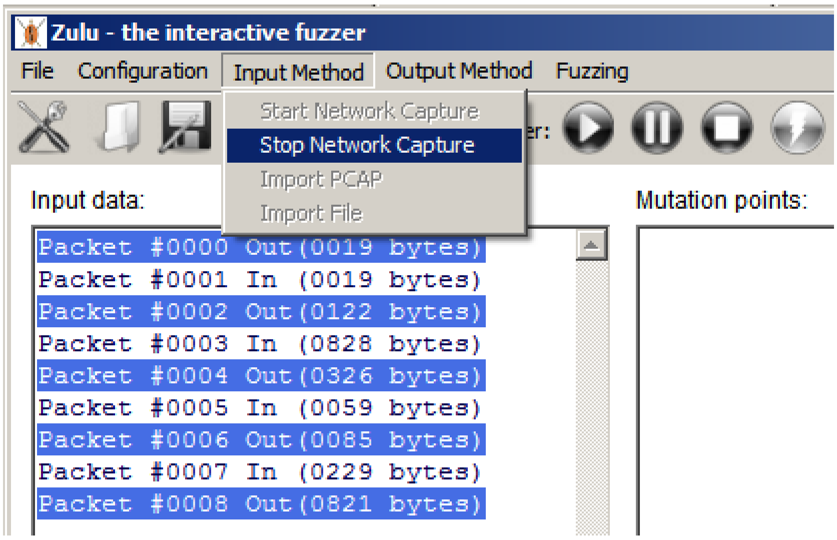
Computer: 127.0.0.1:1234

User name: WIN2K3-HP-QC1\administrator

You will be asked for credentials when you connect.

Figure 8: Configuring the network client software

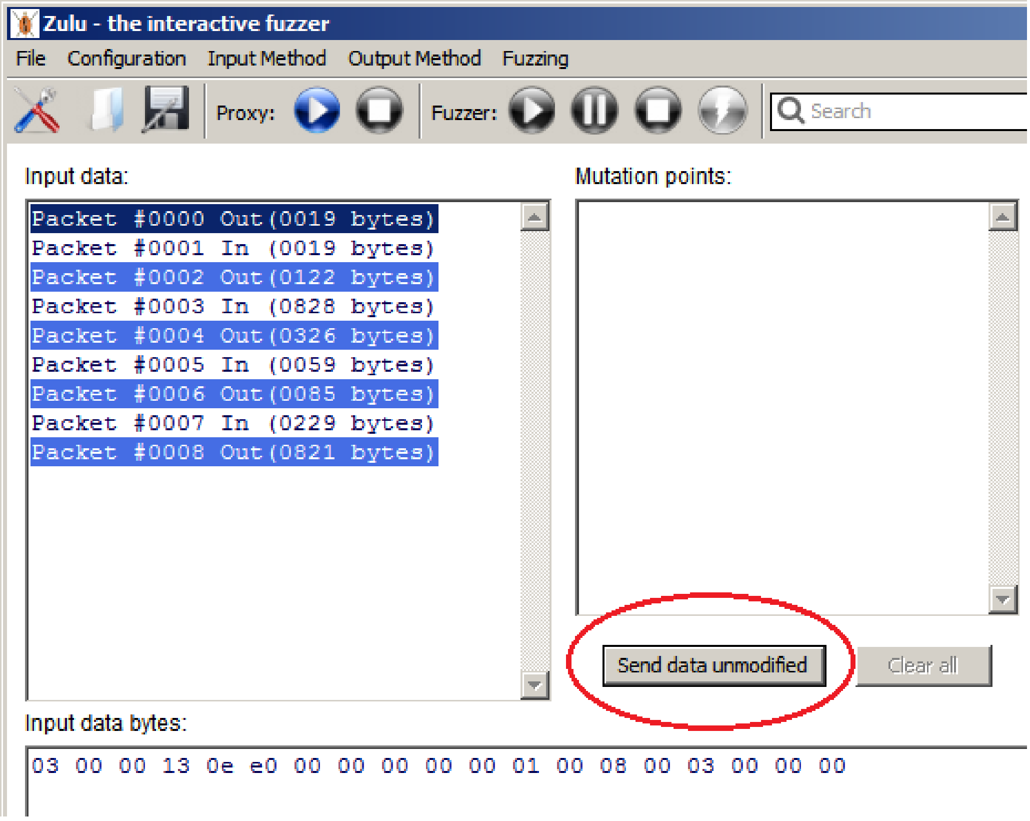
The packets are displayed as they are captured and the "output" packets (which are the ones sent to the target and can therefore be fuzzed) are highlighted in blue. When you are ready to end the capture select **Input Method > Stop Network Capture**:



```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

Figure 9: The displayed captured data

The currently selected packet is highlighted in dark blue and the data within the packet is displayed in the Hex and ASCII areas. For this fuzzing session we will send the first packet unmodified so with the first packet selected, click on the **Send data unmodified** button:



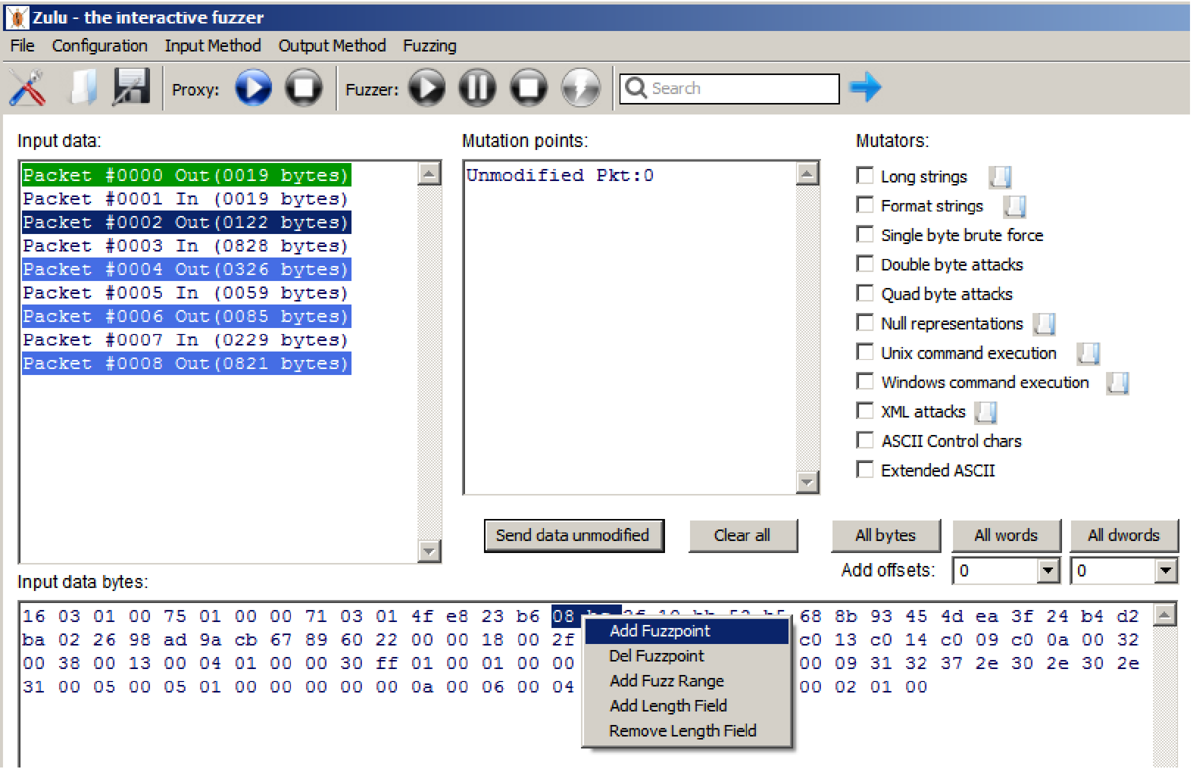
```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

```
03 00 00 13 0e e0 00 00 00 00 00 01 00 08 00 03 00 00 00
```

Figure 10: Specifying data to be sent unmodified

You can see that the first packet has changed colour to green to indicate that this will be sent unmodified. Next select some bytes in packet #2 to be a fuzzpoint (all the bytes you select to be a fuzzpoint will be replaced with each testcase when the fuzzer runs). Right click and select **Add Fuzzpoint**.

Note: If you want to select a series of consecutive bytes as individual fuzzpoints you can use *Add Fuzz Range*.



Input data:

```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

Mutation points:

```
Unmodified Pkt:0
```

Mutators:

- ☐ Long strings
- ☐ Format strings
- ☐ Single byte brute force
- ☐ Double byte attacks
- ☐ Quad byte attacks
- ☐ Null representations
- ☐ Unix command execution
- ☐ Windows command execution
- ☐ XML attacks
- ☐ ASCII Control chars
- ☐ Extended ASCII

Send data unmodified | Clear all | All bytes | All words | All dwords

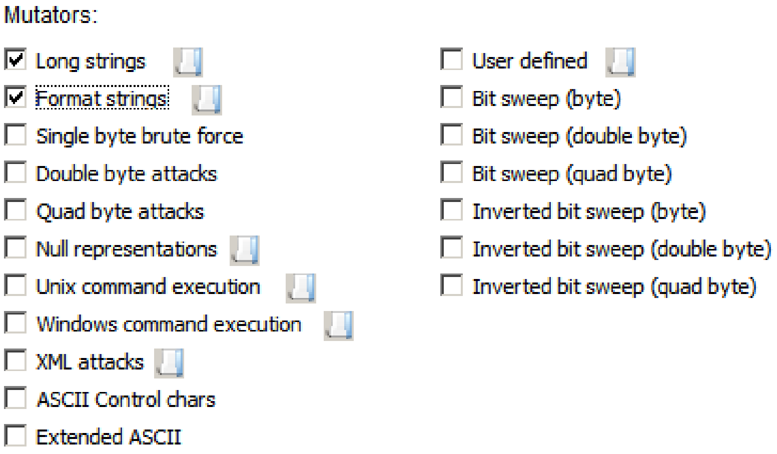
Add offsets: 0 | 0

Input data bytes:

```
16 03 01 00 75 01 00 00 71 03 01 4f e8 23 b6 08                68 8b 93 45 4d ea 3f 24 b4 d2
ba 02 26 98 ad 9a cb 67 89 60 22 00 00 18 00 2f                c0 13 c0 14 c0 09 c0 0a 00 32
00 38 00 13 00 04 01 00 00 30 ff 01 00 01 00 00                00 09 31 32 37 2e 30 2e 30 2e
31 00 05 00 05 01 00 00 00 00 00 0a 00 06 00 04                00 02 01 00
```

Figure 11: Adding a fuzzpoint

Next, select the Mutators that you wish to use for the fuzzing session (to see the actual mutator testcases click on the small light blue button next to each name and the information will be displayed in notepad):

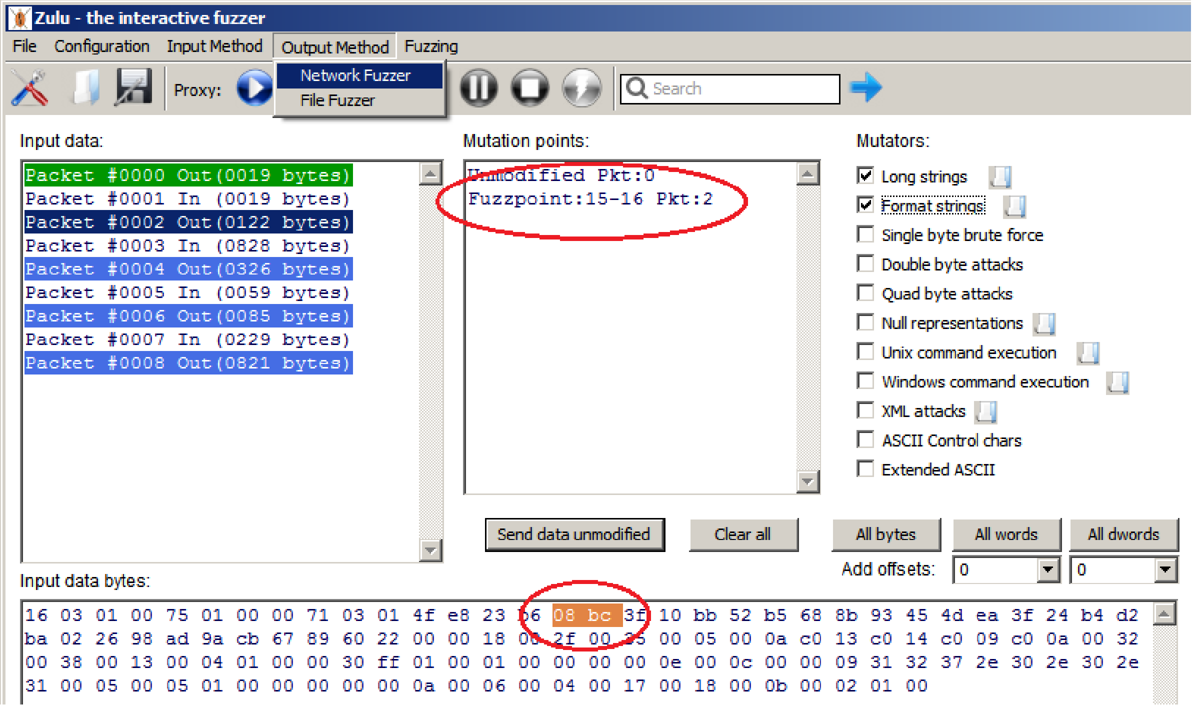


Mutators:

- ☑ Long strings
- ☑ Format strings
- ☐ Single byte brute force
- ☐ Double byte attacks
- ☐ Quad byte attacks
- ☐ Null representations
- ☐ Unix command execution
- ☐ Windows command execution
- ☐ XML attacks
- ☐ ASCII Control chars
- ☐ Extended ASCII
- ☐ User defined
- ☐ Bit sweep (byte)
- ☐ Bit sweep (double byte)
- ☐ Bit sweep (quad byte)
- ☐ Inverted bit sweep (byte)
- ☐ Inverted bit sweep (double byte)
- ☐ Inverted bit sweep (quad byte)

Figure 12: Selectable mutators

As you can see, the Fuzzpoint bytes have changed colour to orange and the Fuzzpoint information has been added to the Mutations area. Next, select **Output Method > Network Fuzzer**:



Input data:

```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

Mutation points:

```
Unmodified Pkt:0
Fuzzpoint:15-16 Pkt:2
```

Mutators:

- ☑ Long strings
- ☑ Format strings
- ☐ Single byte brute force
- ☐ Double byte attacks
- ☐ Quad byte attacks
- ☐ Null representations
- ☐ Unix command execution
- ☐ Windows command execution
- ☐ XML attacks
- ☐ ASCII Control chars
- ☐ Extended ASCII

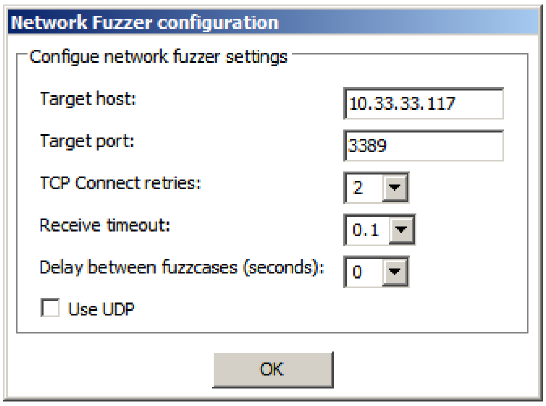
Add offsets: 0 0

Input data bytes:

```
16 03 01 00 75 01 00 00 71 03 01 4f e8 23 b6 08 bc 3f 10 bb 52 b5 68 8b 93 45 4d ea 3f 24 b4 d2
ba 02 26 98 ad 9a cb 67 89 60 22 00 00 18 00 2f 00 35 00 05 00 0a c0 13 c0 14 c0 09 c0 0a 00 32
00 38 00 13 00 04 01 00 00 30 ff 01 00 01 00 00 00 00 0e 00 0c 00 00 09 31 32 37 2e 30 2e 30 2e
31 00 05 00 05 01 00 00 00 00 00 0a 00 06 00 04 00 17 00 18 00 0b 00 02 01 00
```

Figure 13: Selecting the network fuzzer

This will bring up the **Network fuzzer configuration** window:



Network Fuzzer configuration

Configue network fuzzer settings

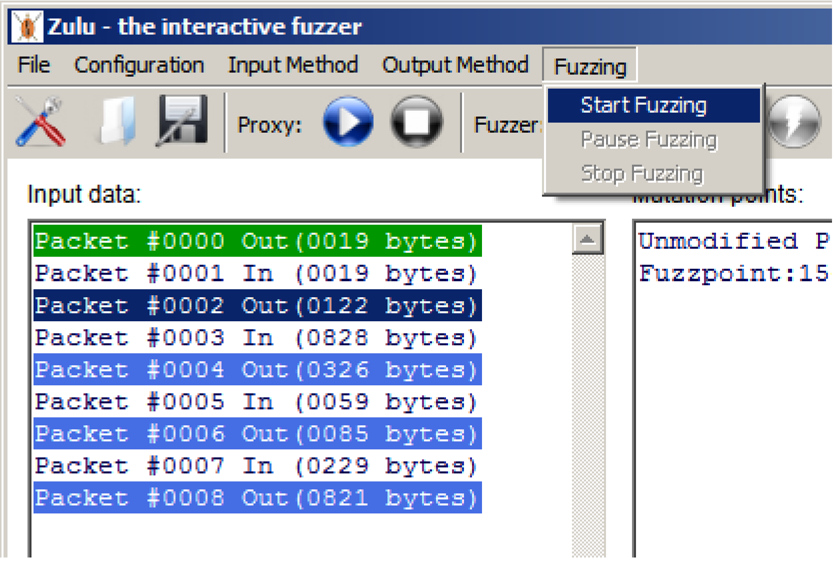
Target host:	10.33.33.117
Target port:	3389
TCP Connect retries:	2
Receive timeout:	0.1
Delay between fuzzcases (seconds):	0

☐ Use UDP

Figure 14: The network fuzzer settings

- **Target host** - this is by default the target that was used in the proxy capture, but you can change it to a different target if you wish
- **Target port** - again, this is by default the target port that was used in the proxy capture, but you can change this too
- **TCP connect retries** - Zulu detects if the service has crashed if a TCP connect fails (currently there is no detection for UDP). This is the number of times to try connecting before deciding a crash has occurred
- **Receive timeout** - the time to wait for the target to respond to a fuzz packet sent to it
- **Delay between fuzzcases** - self explanatory
- **Use UDP** - if the service is UDP-based, select this checkbox

Now, we can start fuzzing - select **Fuzzing > Start Fuzzing**:



Input data:

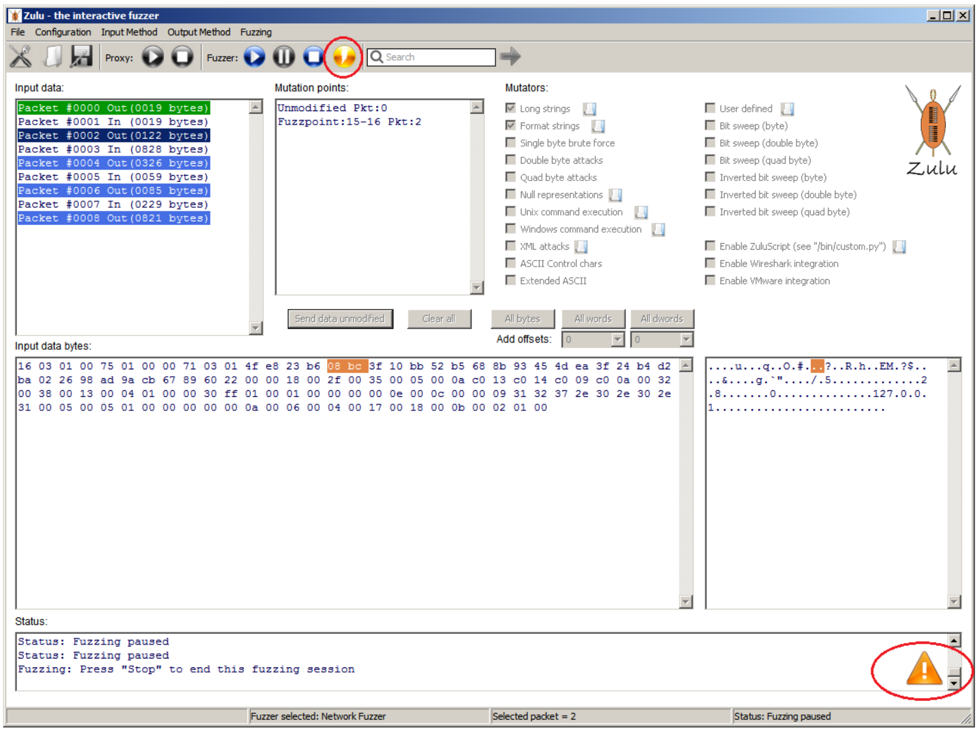
```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

Mutation points:

```
Unmodified P
Fuzzpoint:15
```

Figure 15: Starting the network fuzzer

If the target crashes (as has been demonstrated here), an orange warning triangle is displayed and a PoC (Proof-of-Concept) exploit is automatically generated based on the last fuzzcase that was sent to the target (Zulu makes the assumption that it is likely, although not guaranteed, that the last fuzzcase sent was the one that caused the crash). Also, if email settings have been configured (**Configuration > Email Notifcation Settings**) it will also send the PoC as an attachment to an email to inform you of the crash. If a PoC has been generated the **Launch Latest PoC** button is highlighted in orange and can be used to send the PoC to the target as many times as you like, just by pressing the button.



Input data:

```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

Mutation points:

```
Unmodified Pkt:0
Fuzzpoint:15-16 Pkt:2
```

Mutators:

- ☑ Long strings
- ☑ Format strings
- ☐ Single byte brute force
- ☐ Double byte attacks
- ☐ Quad byte attacks
- ☐ Null representations
- ☐ Unix command execution
- ☐ Windows command execution
- ☐ XML attacks
- ☐ ASCII Control chars
- ☐ Extended ASCII
- ☐ User defined
- ☐ Bit sweep (byte)
- ☐ Bit sweep (double byte)
- ☐ Bit sweep (quad byte)
- ☐ Inverted bit sweep (byte)
- ☐ Inverted bit sweep (double byte)
- ☐ Inverted bit sweep (quad byte)
- ☐ Enable ZuluScript (see "/bin/custom.py")
- ☐ Enable Wireshark integration
- ☐ Enable VMware integration

Add offsets: 0 0

Input data bytes:

```
16 03 01 00 75 01 00 00 71 03 01 4f e8 23 b6 08 bc 3f 10 bb 52 b5 68 8b 93 45 4d ea 3f 24 b4 d2
ba 02 26 98 ad 9a cb 67 89 60 22 00 00 18 00 2f 00 35 00 05 00 0a c0 13 c0 14 c0 09 c0 0a 00 32
00 38 00 13 00 04 01 00 00 30 ff 01 00 01 00 00 00 0e 00 0c 00 00 09 31 32 37 2e 30 2e 30 2e
31 00 05 00 05 01 00 00 00 00 00 0a 00 06 00 04 00 17 00 18 00 0b 00 02 01 00
```

```
....u...q..O.#...?..R.h..EM.?$..
..&....g.`"..../.5.............2
.8.......0..............127.0.0.
1........................
```

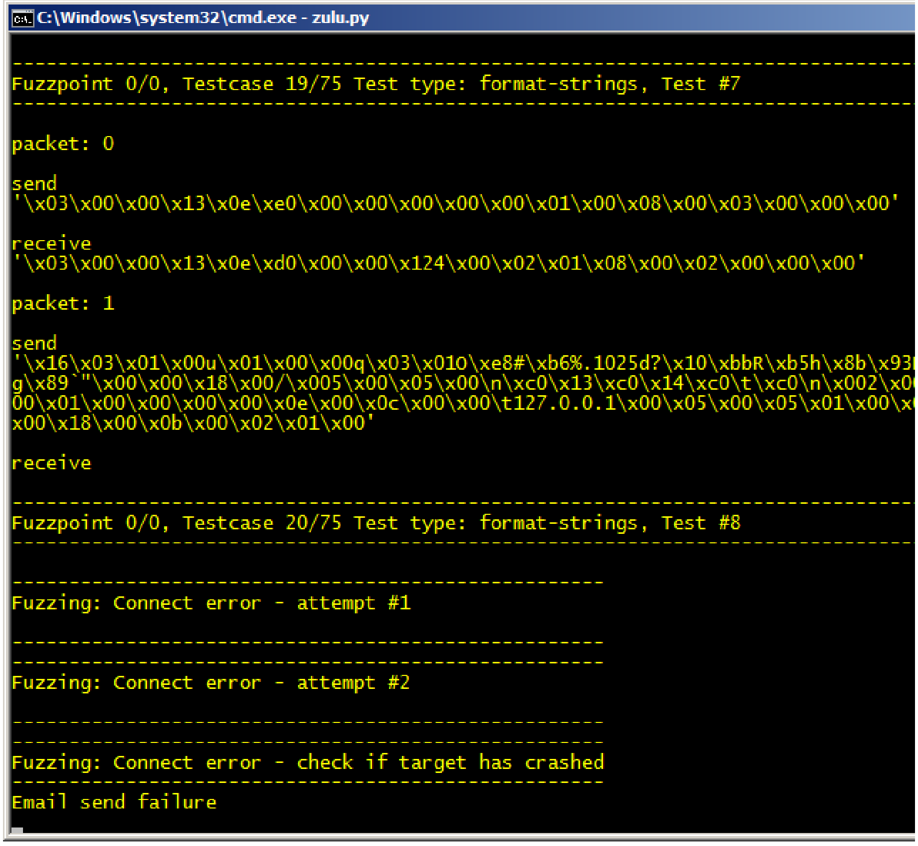
Status:

```
Status: Fuzzing paused
Status: Fuzzing paused
Fuzzing: Press "Stop" to end this fuzzing session
```

Fuzzer selected: Network Fuzzer | Selected packet = 2 | Status: Fuzzing paused

Figure 16: Indicators that the target has crashed

The console output can be seen below (the "Email send failure" message indicates that in this instance email settings have not been configured):

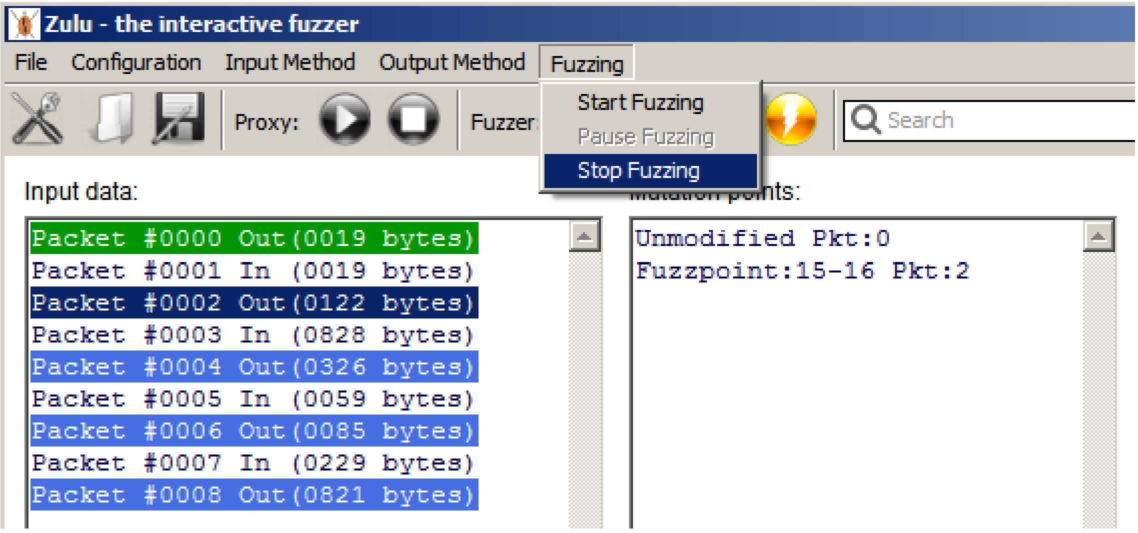


C:\Windows\system32\cmd.exe - zulu.py

```
-----------------------------------------------------------------------------
Fuzzpoint 0/0, Testcase 19/75 Test type: format-strings, Test #7
-----------------------------------------------------------------------------

packet: 0

send
'\x03\x00\x00\x13\x0e\xe0\x00\x00\x00\x00\x00\x01\x00\x08\x00\x03\x00\x00\x00'

receive
'\x03\x00\x00\x13\x0e\xd0\x00\x00\x124\x00\x02\x01\x08\x00\x02\x00\x00\x00'

packet: 1

send
'\x16\x03\x01\x00u\x01\x00\x00q\x03\x01o\xe8#\xb6%.1025d?\x10\xbbR\xb5h\x8b\x93
g\x89`"\x00\x00\x18\x00/\x005\x00\x05\x00\n\xc0\x13\xc0\x14\xc0\t\xc0\n\x002\x0
00\x01\x00\x00\x00\x00\x0e\x00\x0c\x00\x00\t127.0.0.1\x00\x05\x00\x05\x01\x00\x
x00\x18\x00\x0b\x00\x02\x01\x00'

receive

-----------------------------------------------------------------------------
Fuzzpoint 0/0, Testcase 20/75 Test type: format-strings, Test #8
-----------------------------------------------------------------------------

---------------------------------------------------
Fuzzing: Connect error - attempt #1

---------------------------------------------------
---------------------------------------------------
Fuzzing: Connect error - attempt #2

---------------------------------------------------
---------------------------------------------------
Fuzzing: Connect error - check if target has crashed
---------------------------------------------------
Email send failure
```

Figure 17: Console output when the target has crashed

When a Zulu thinks that a crash has occurred it pauses the fuzzing session, so if it turns out to be a false-positive you can just restart the fuzzing by selecting **Fuzzing > Start Fuzzing** again or stop the session by selecting **Fuzzing > Stop Fuzzing**:



```
Packet #0000 Out(0019 bytes)
Packet #0001 In  (0019 bytes)
Packet #0002 Out(0122 bytes)
Packet #0003 In  (0828 bytes)
Packet #0004 Out(0326 bytes)
Packet #0005 In  (0059 bytes)
Packet #0006 Out(0085 bytes)
Packet #0007 In  (0229 bytes)
Packet #0008 Out(0821 bytes)
```

```
Unmodified Pkt:0
Fuzzpoint:15-16 Pkt:2
```

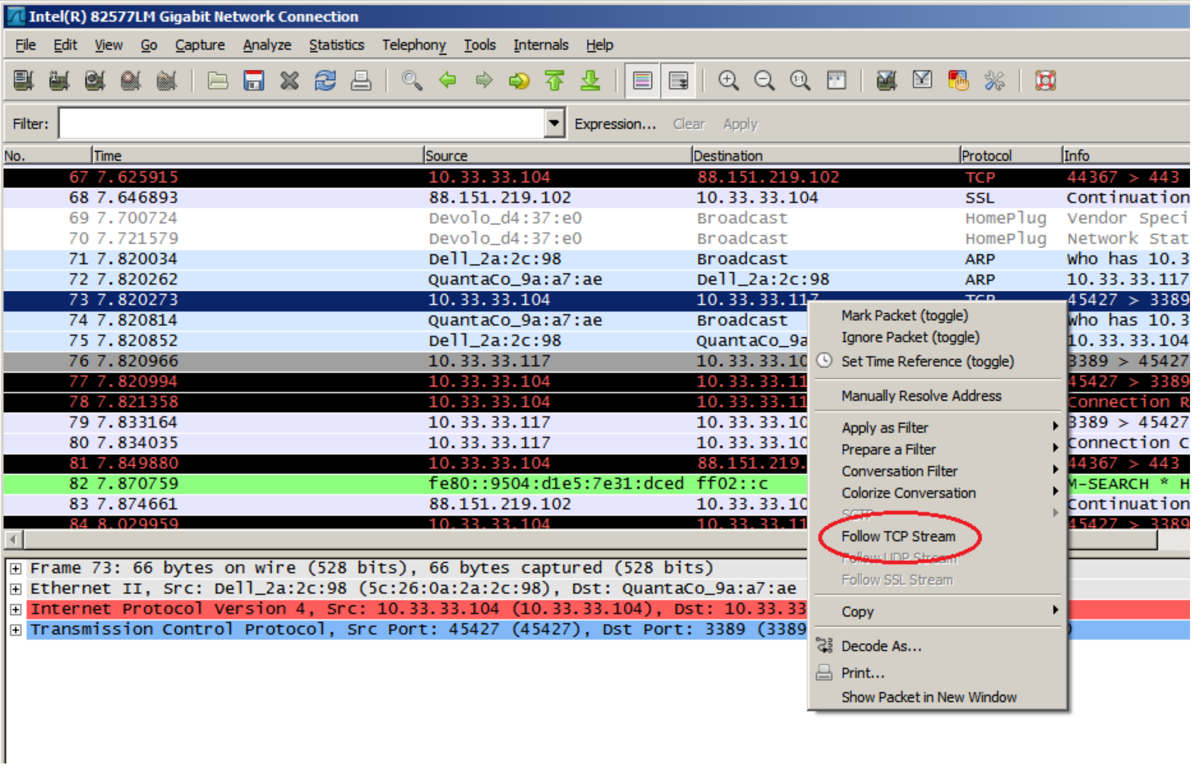
Figure 18: Stopping the fuzzer

4 Tutorial Three: Providing input using a PCAP file

In this tutorial we will provide input data to Zulu via a PCAP file generated by Wireshark .

Note: Please ensure that you have captured the data on a specific interface when capturing on Linux, rather than using the *Any* interface, which results in the Link layer not being *Ethernet II*, but instead *Linux Cooked Capture* - a pseudo-protocol used by libpcap on Linux.

First, capture your data in Wireshark, select a packet from the stream of data you wish to fuzz and **Follow TCP Stream**:

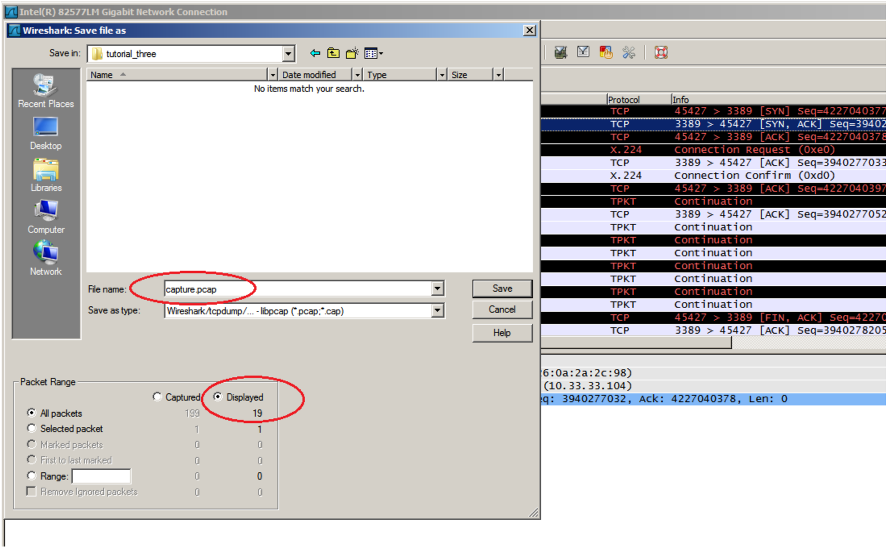


```
No.   Time       Source                        Destination        Protocol   Info
   67 7.625915   10.33.33.104                  88.151.219.102     TCP        44367 > 443
   68 7.646893   88.151.219.102                10.33.33.104       SSL        Continuation
   69 7.700724   Devolo_d4:37:e0               Broadcast          HomePlug   Vendor Speci
   70 7.721579   Devolo_d4:37:e0               Broadcast          HomePlug   Network Stat
   71 7.820034   Dell_2a:2c:98                 Broadcast          ARP        Who has 10.3
   72 7.820262   QuantaCo_9a:a7:ae             Dell_2a:2c:98      ARP        10.33.33.117
   73 7.820273   10.33.33.104                  10.33.33.117       TCP        45427 > 3389
   74 7.820814   QuantaCo_9a:a7:ae             Broadcast                     Who has 10.3
   75 7.820852   Dell_2a:2c:98                 QuantaCo_9a                   10.33.33.104
   76 7.820966   10.33.33.117                  10.33.33.10                   3389 > 45427
   77 7.820994   10.33.33.104                  10.33.33.11                   45427 > 3389
   78 7.821358   10.33.33.104                  10.33.33.11                   Connection R
   79 7.833164   10.33.33.117                  10.33.33.10                   3389 > 45427
   80 7.834035   10.33.33.117                  10.33.33.10                   Connection C
   81 7.849880   10.33.33.104                  88.151.219.                   44367 > 443
   82 7.870759   fe80::9504:d1e5:7e31:dced     ff02::c                       M-SEARCH * H
   83 7.874661   88.151.219.102                10.33.33.10                   Continuation
   84 8.029959   10.33.33.104                  10.33.33.11                   45427 > 3389

⊞ Frame 73: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: Dell_2a:2c:98 (5c:26:0a:2a:2c:98), Dst: QuantaCo_9a:a7:ae
⊞ Internet Protocol Version 4, Src: 10.33.33.104 (10.33.33.104), Dst: 10.33.33
⊞ Transmission Control Protocol, Src Port: 45427 (45427), Dst Port: 3389 (3389
```

Figure 18: Filtering out data stream using Wireshark

Then save the capture

Note: ensure that you have selected to save the packets *Displayed* rather than all packets *Captured*



Intel(R) 82577LM Gigabit Network Connection

Wireshark: Save file as

Save in: tutorial_three

No items match your search.

File name: capture.pcap

Save as type: Wireshark/tcpdump/... - libpcap (*.pcap;*.cap)

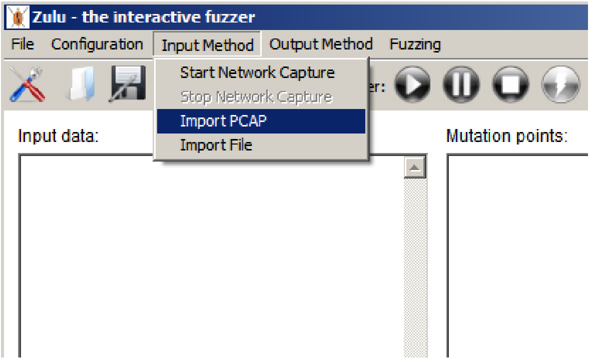
Packet Range	○ Captured	● Displayed
● All packets	199	19
○ Selected packet	1	1
○ Marked packets	0	0
○ First to last marked	0	0
○ Range:	0	0
□ Remove Ignored packets	0	0

```
Protocol  Info
TCP       45427 > 3389 [SYN] Seq=4227040377
TCP       3389 > 45427 [SYN, ACK] Seq=39402
TCP       45427 > 3389 [ACK] Seq=4227040378
X.224     Connection Request (0xe0)
TCP       3389 > 45427 [ACK] Seq=3940277033
X.224     Connection Confirm (0xd0)
TCP       45427 > 3389 [ACK] Seq=4227040397
TPKT      Continuation
TCP       3389 > 45427 [ACK] Seq=3940277052
TPKT      Continuation
TPKT      Continuation
TPKT      Continuation
TPKT      Continuation
TPKT      Continuation
TPKT      Continuation
TPKT      Continuation
TCP       45427 > 3389 [FIN, ACK] Seq=42270
TCP       3389 > 45427 [ACK] Seq=3940278205
```

```
6:0a:2a:2c:98)
(10.33.33.104)
q: 3940277032, Ack: 4227040378, Len: 0
```

Figure 19: Saving the captured data stream

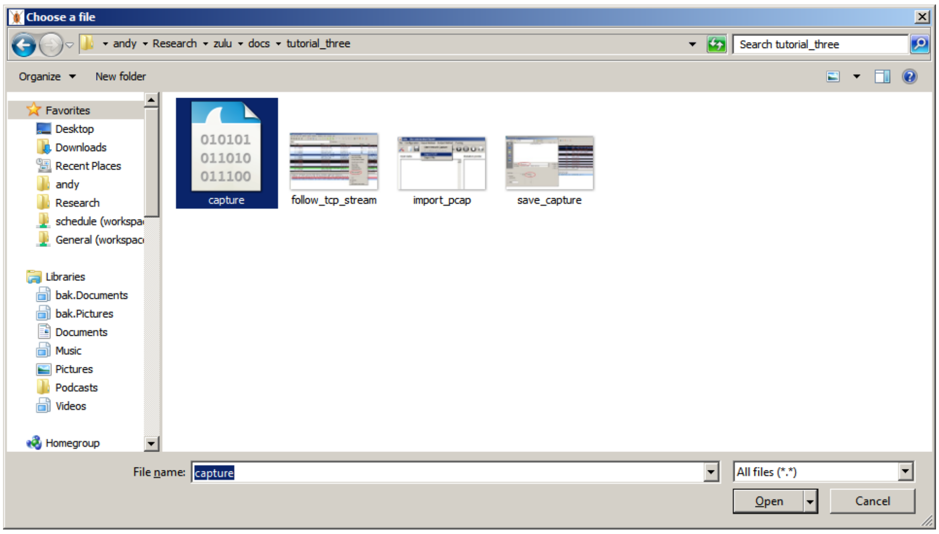
In Zulu, select **Input Method > Import PCAP**:



Zulu - the interactive fuzzer

Figure 20: Importing a PCAP file into Zulu

Select the PCAP file and click **Open**:



Choose a file

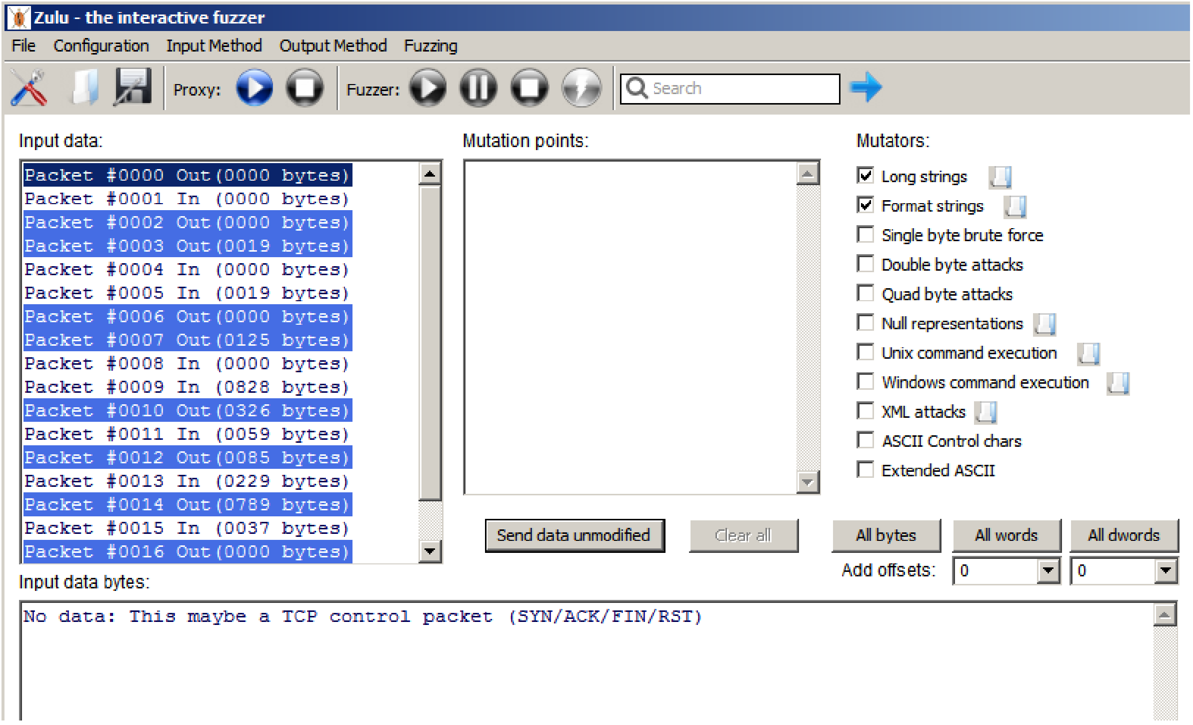
andy ▸ Research ▸ zulu ▸ docs ▸ tutorial_three

File name: capture

All files (*.*)

Figure 20: Selecting the PCAP file

The data is then displayed, as in Tutorial two, but with one difference - TCP control packets are also shown e.g. SYN, ACK etc. These can be ignored from a fuzzing perspective as they will automatically be regenerated where required by Zulu:



```
Packet #0000 Out(0000 bytes)
Packet #0001 In  (0000 bytes)
Packet #0002 Out(0000 bytes)
Packet #0003 Out(0019 bytes)
Packet #0004 In  (0000 bytes)
Packet #0005 In  (0019 bytes)
Packet #0006 Out(0000 bytes)
Packet #0007 Out(0125 bytes)
Packet #0008 In  (0000 bytes)
Packet #0009 In  (0828 bytes)
Packet #0010 Out(0326 bytes)
Packet #0011 In  (0059 bytes)
Packet #0012 Out(0085 bytes)
Packet #0013 In  (0229 bytes)
Packet #0014 Out(0789 bytes)
Packet #0015 In  (0037 bytes)
Packet #0016 Out(0000 bytes)
```

☑ Long strings
☑ Format strings
☐ Single byte brute force
☐ Double byte attacks
☐ Quad byte attacks
☐ Null representations
☐ Unix command execution
☐ Windows command execution
☐ XML attacks
☐ ASCII Control chars
☐ Extended ASCII

Add offsets: 0 0

```
No data: This maybe a TCP control packet (SYN/ACK/FIN/RST)
```

Figure 21: The imported PCAP file in Zulu

Note: If the PCAP file was generated on another network and involves a target with a different IP address or even a different port, this information can be changed in **Output Method > Network Fuzzer** otherwise it will just be inherited from the PCAP file.

5 Tutorial Four: File fuzzing

In this tutorial we will use Zulu to fuzz a target process which opens a file.
First we select **Input Method > import** File to select a file to use as the fuzzing template:

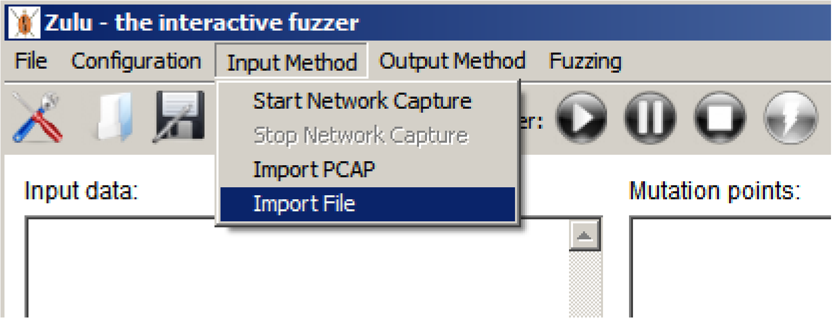
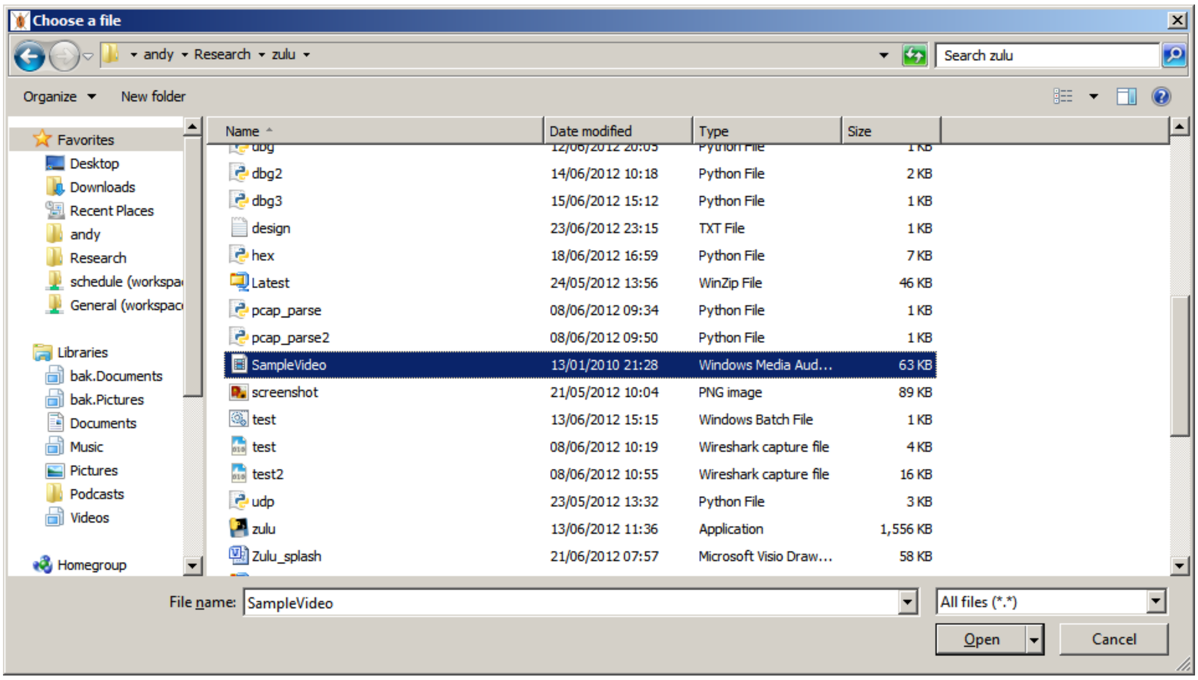


Figure 22: Importing a file to fuzz

Select a file then click **Open**:

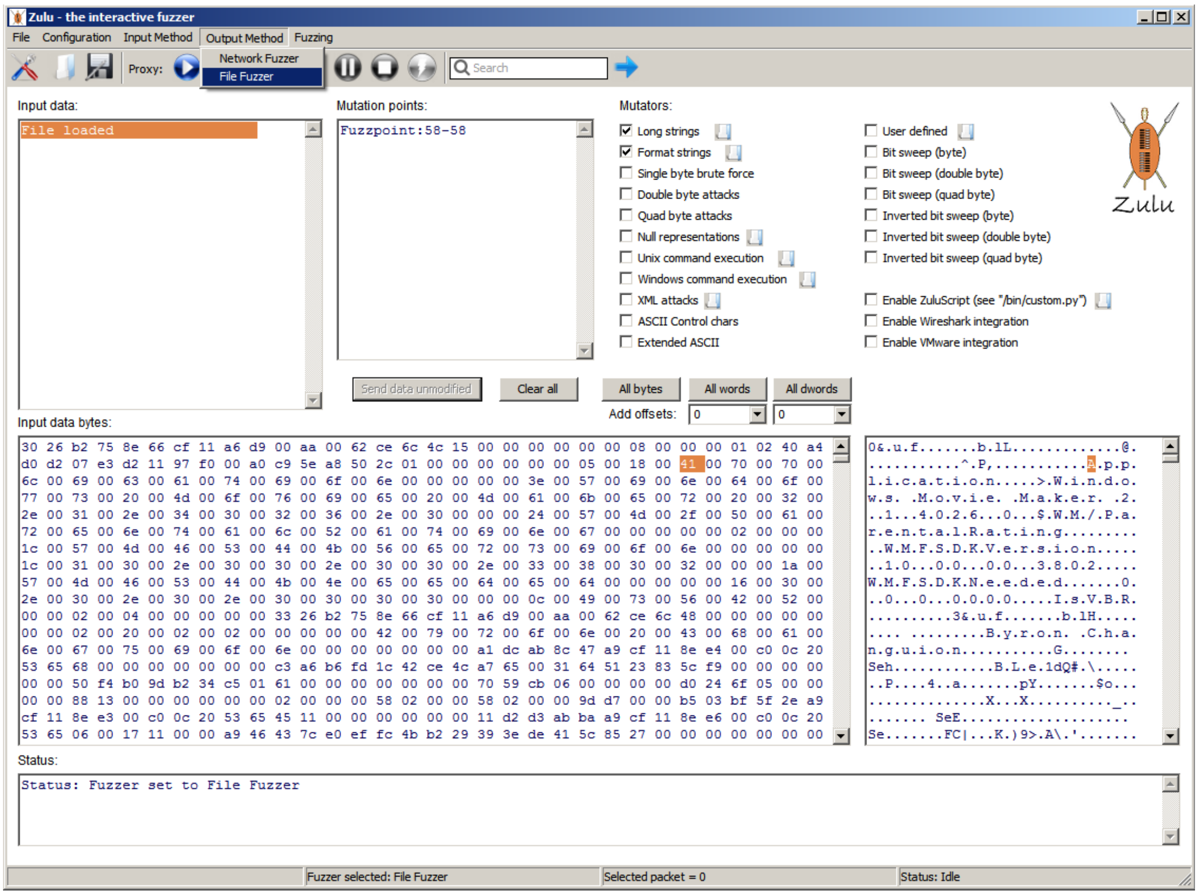


Name	Date modified	Type	Size
dbg	12/06/2012 20:05	Python File	1 KB
dbg2	14/06/2012 10:18	Python File	2 KB
dbg3	15/06/2012 15:12	Python File	1 KB
design	23/06/2012 23:15	TXT File	1 KB
hex	18/06/2012 16:59	Python File	7 KB
Latest	24/05/2012 13:56	WinZip File	46 KB
pcap_parse	08/06/2012 09:34	Python File	1 KB
pcap_parse2	08/06/2012 09:50	Python File	1 KB
SampleVideo	13/01/2010 21:28	Windows Media Aud...	63 KB
screenshot	21/05/2012 10:04	PNG image	89 KB
test	13/06/2012 15:15	Windows Batch File	1 KB
test	08/06/2012 10:19	Wireshark capture file	4 KB
test2	08/06/2012 10:55	Wireshark capture file	16 KB
udp	23/05/2012 13:32	Python File	3 KB
zulu	13/06/2012 11:36	Application	1,556 KB
Zulu_splash	21/06/2012 07:57	Microsoft Visio Draw...	58 KB

File name: SampleVideo

Figure 23: Selecting the source file

Then select the file fuzzer module with **Output Method > File Fuzzer**:



Input data:

```
File loaded
```

Mutation points:

```
Fuzzpoint:58-58
```

Mutators:

☑ Long strings
☑ Format strings
□ Single byte brute force
□ Double byte attacks
□ Quad byte attacks
□ Null representations
□ Unix command execution
□ Windows command execution
□ XML attacks
□ ASCII Control chars
□ Extended ASCII

□ User defined
□ Bit sweep (byte)
□ Bit sweep (double byte)
□ Bit sweep (quad byte)
□ Inverted bit sweep (byte)
□ Inverted bit sweep (double byte)
□ Inverted bit sweep (quad byte)

□ Enable ZuluScript (see "/bin/custom.py")
□ Enable Wireshark integration
□ Enable VMware integration

Add offsets: 0 0

Input data bytes:

```
30 26 b2 75 8e 66 cf 11 a6 d9 00 aa 00 62 ce 6c 4c 15 00 00 00 00 00 00 08 00 00 00 01 02 40 a4
d0 d2 07 e3 d2 11 97 f0 00 a0 c9 5e a8 50 2c 01 00 00 00 00 00 00 05 00 18 00 41 00 70 00 70 00
6c 00 69 00 63 00 61 00 74 00 69 00 6f 00 6e 00 00 00 00 00 3e 00 57 00 69 00 6e 00 64 00 6f 00
77 00 73 00 20 00 4d 00 6f 00 76 00 69 00 65 00 20 00 4d 00 61 00 6b 00 65 00 72 00 20 00 32 00
2e 00 31 00 2e 00 34 00 30 00 32 00 36 00 2e 00 30 00 00 00 24 00 57 00 4d 00 2f 00 50 00 61 00
72 00 65 00 6e 00 74 00 61 00 6c 00 52 00 61 00 74 00 69 00 6e 00 67 00 00 00 00 00 02 00 00 00
1c 00 57 00 4d 00 46 00 53 00 44 00 4b 00 56 00 65 00 72 00 73 00 69 00 6f 00 6e 00 00 00 00 00
1c 00 31 00 30 00 2e 00 30 00 30 00 2e 00 30 00 30 00 2e 00 33 00 38 00 30 00 32 00 00 00 1a 00
57 00 4d 00 46 00 53 00 44 00 4b 00 4e 00 65 00 65 00 64 00 65 00 64 00 00 00 00 00 16 00 30 00
2e 00 30 00 2e 00 30 00 2e 00 30 00 30 00 30 00 30 00 00 00 0c 00 49 00 73 00 56 00 42 00 52 00
00 00 02 00 04 00 00 00 00 00 33 26 b2 75 8e 66 cf 11 a6 d9 00 aa 00 62 ce 6c 48 00 00 00 00 00
00 00 02 00 20 00 02 00 02 00 00 00 00 00 42 00 79 00 72 00 6f 00 6e 00 20 00 43 00 68 00 61 00
6e 00 67 00 75 00 69 00 6f 00 6e 00 00 00 00 00 00 00 a1 dc ab 8c 47 a9 cf 11 8e e4 00 c0 0c 20
53 65 68 00 00 00 00 00 00 00 c3 a6 b6 fd 1c 42 ce 4c a7 65 00 31 64 51 23 83 5c f9 00 00 00 00
00 00 50 f4 b0 9d b2 34 c5 01 61 00 00 00 00 00 00 00 70 59 cb 06 00 00 00 00 d0 24 6f 05 00 00
00 00 88 13 00 00 00 00 00 00 02 00 00 00 58 02 00 00 58 02 00 00 9d d7 00 00 b5 03 bf 5f 2e a9
cf 11 8e e3 00 c0 0c 20 53 65 45 11 00 00 00 00 00 00 11 d2 d3 ab ba a9 cf 11 8e e6 00 c0 0c 20
53 65 06 00 17 11 00 00 a9 46 43 7c e0 ef fc 4b b2 29 39 3e de 41 5c 85 27 00 00 00 00 00 00 00
```

```
0&.u.f......b.lL..............@.
..........^.P,..........A.p.p.
l.i.c.a.t.i.o.n.....>.W.i.n.d.o.
w.s. .M.o.v.i.e. .M.a.k.e.r. .2.
..1...4.0.2.6...0...$.W.M./.P.a.
r.e.n.t.a.l.R.a.t.i.n.g.........
..W.M.F.S.D.K.V.e.r.s.i.o.n.....
..1.0...0.0...0.0...3.8.0.2.....
W.M.F.S.D.K.N.e.e.d.e.d.......0.
..0...0...0.0.0.0.....I.s.V.B.R.
..........3&.u.f.......b.lH.....
.... .........B.y.r.o.n. .C.h.a.
n.g.u.i.o.n...........G........
Seh............B.L.e.1dQ#.\.....
..P....4..a.......pY........$o...
..............X...X.........._..
....... SeE.....................
Se.......FC|...K.)9>.A\.'.......
```

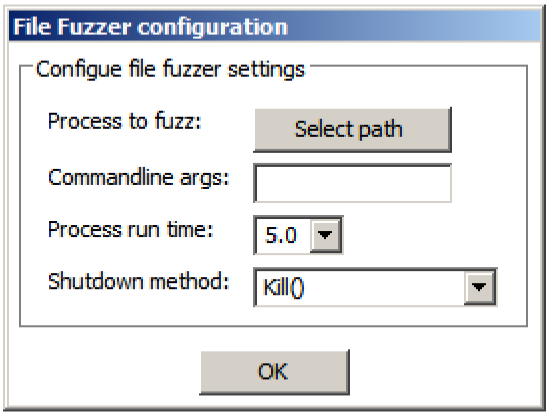
Status:

```
Status: Fuzzer set to File Fuzzer
```

Fuzzer selected: File Fuzzer | Selected packet = 0 | Status: Idle

Figure 24: Selecting the file fuzzer module

The **File fuzzer configuration** window is displayed:



Process to fuzz:

Commandline args:

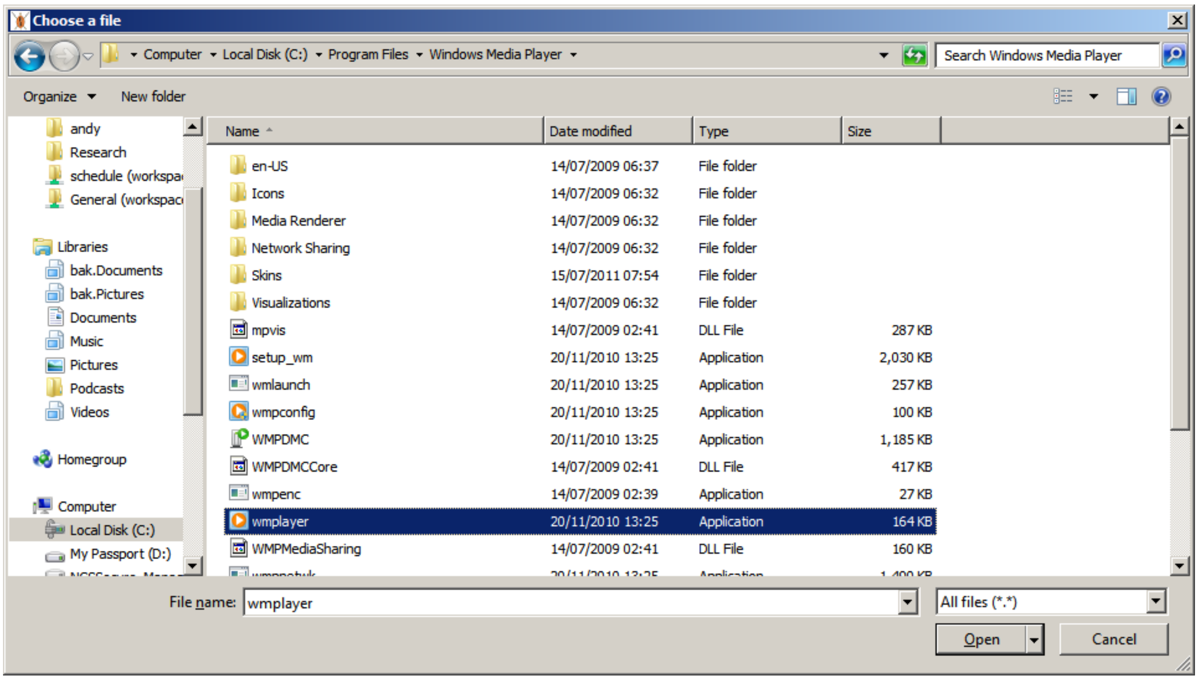
Process run time: 5.0

Shutdown method: Kill()

Figure 25: The file fuzzer configuration

- **Process to fuzz** - the target process that will be automatically spawned and provided with each fuzzcase
- **Command line args** - self explanatory
- **process run time** - the length of time the process will run for before it is killed
- **Shutdown method** - either *kill()* or *terminate()* (*terminate()* uses the win32 API)

Select the process to fuzz:



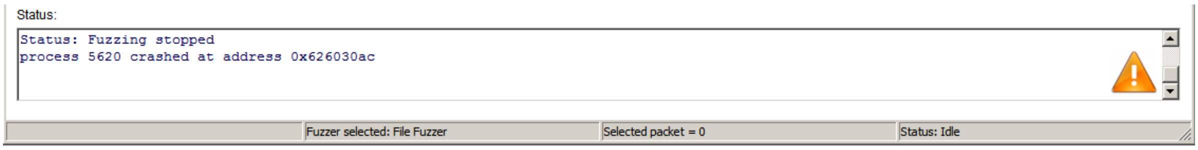
Name	Date modified	Type	Size
en-US	14/07/2009 06:37	File folder	
Icons	14/07/2009 06:32	File folder	
Media Renderer	14/07/2009 06:32	File folder	
Network Sharing	14/07/2009 06:32	File folder	
Skins	15/07/2011 07:54	File folder	
Visualizations	14/07/2009 06:32	File folder	
mpvis	14/07/2009 02:41	DLL File	287 KB
setup_wm	20/11/2010 13:25	Application	2,030 KB
wmlaunch	20/11/2010 13:25	Application	257 KB
wmpconfig	20/11/2010 13:25	Application	100 KB
WMPDMC	20/11/2010 13:25	Application	1,185 KB
WMPDMCCore	14/07/2009 02:41	DLL File	417 KB
wmpenc	14/07/2009 02:39	Application	27 KB
wmplayer	20/11/2010 13:25	Application	164 KB
WMPMediaSharing	14/07/2009 02:41	DLL File	160 KB

File name: wmplayer

All files (*.*)

Figure 26: Selecting the process to fuzz

Then select **Fuzzing > Start Fuzzing**. If the process dies as a result of opening the fuzzcase, the in-built debugger will trap the crash, display EIP at the time of the crash and copy the fuzzfile responsible for the crash to the **/crashfiles** directory:


Status:

```
Status: Fuzzing stopped
process 5620 crashed at address 0x626030ac
```

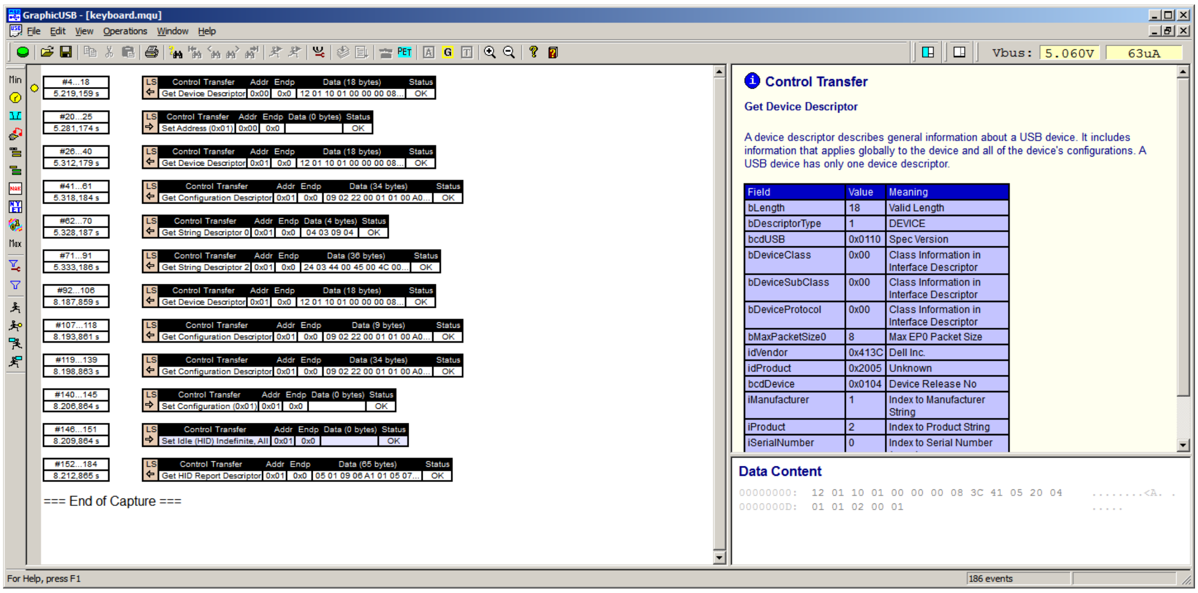
Fuzzer selected: File Fuzzer | Selected packet = 0 | Status: Idle

Figure 27: The debugger indicating the address where the process crashed

6 Tutorial Five: Using the USB fuzzer module

In this tutorial we will perform some USB fuzzing by integrating with external hardware - the Packet-Master USB500 AG from MQP Electronics (NCC Group has no affiliation with MQP Electronics): http://www.mqp.com/usb500.htm

We start by capturing some USB traffic using GraphicUSB (the GUI application shipped with the USB500):



GraphicUSB - [keyboard.mqu]
File Edit View Operations Window Help
Min
Max
PET
A
G
T

```
#4...18      LS  Control Transfer            Addr  Endp  Data (18 bytes)           Status
5.219,159 s      Get Device Descriptor       0x00  0x0   12 01 10 01 00 00 00 08...  OK

#20...25     LS  Control Transfer            Addr  Endp  Data (0 bytes)            Status
5.281,174 s      Set Address (0x01)          0x00  0x0                             OK

#26...40     LS  Control Transfer            Addr  Endp  Data (18 bytes)           Status
5.312,179 s      Get Device Descriptor       0x01  0x0   12 01 10 01 00 00 00 08...  OK

#41...61     LS  Control Transfer            Addr  Endp  Data (34 bytes)           Status
5.318,184 s      Get Configuration Descriptor 0x01 0x0   09 02 22 00 01 01 00 A0...  OK

#62...70     LS  Control Transfer            Addr  Endp  Data (4 bytes)            Status
5.328,187 s      Get String Descriptor 0     0x01  0x0   04 03 09 04                 OK

#71...91     LS  Control Transfer            Addr  Endp  Data (36 bytes)           Status
5.333,186 s      Get String Descriptor 2     0x01  0x0   24 03 44 00 45 00 4C 00...  OK

#92...106    LS  Control Transfer            Addr  Endp  Data (18 bytes)           Status
8.187,859 s      Get Device Descriptor       0x01  0x0   12 01 10 01 00 00 00 08...  OK

#107...118   LS  Control Transfer            Addr  Endp  Data (9 bytes)            Status
8.193,861 s      Get Configuration Descriptor 0x01 0x0   09 02 22 00 01 01 00 A0...  OK

#119...139   LS  Control Transfer            Addr  Endp  Data (34 bytes)           Status
8.198,863 s      Get Configuration Descriptor 0x01 0x0   09 02 22 00 01 01 00 A0...  OK

#140...145   LS  Control Transfer            Addr  Endp  Data (0 bytes)            Status
8.206,864 s      Set Configuration (0x01)    0x01  0x0                             OK

#146...151   LS  Control Transfer            Addr  Endp  Data (0 bytes)            Status
8.209,864 s      Set Idle (HID) Indefinite, All 0x01 0x0                           OK

#152...184   LS  Control Transfer            Addr  Endp  Data (65 bytes)           Status
8.212,865 s      Get HID Report Descriptor   0x01  0x0   05 01 09 06 A1 01 05 07...  OK

=== End of Capture ===
```

Vbus: 5.060V 63uA

Control Transfer

Get Device Descriptor

A device descriptor describes general information about a USB device. It includes information that applies globally to the device and all of the device's configurations. A USB device has only one device descriptor.

Field	Value	Meaning
bLength	18	Valid Length
bDescriptorType	1	DEVICE
bcdUSB	0x0110	Spec Version
bDeviceClass	0x00	Class Information in Interface Descriptor
bDeviceSubClass	0x00	Class Information in Interface Descriptor
bDeviceProtocol	0x00	Class Information in Interface Descriptor
bMaxPacketSize0	8	Max EP0 Packet Size
idVendor	0x413C	Dell Inc.
idProduct	0x2005	Unknown
bcdDevice	0x0104	Device Release No
iManufacturer	1	Index to Manufacturer String
iProduct	2	Index to Product String
iSerialNumber	0	Index to Serial Number

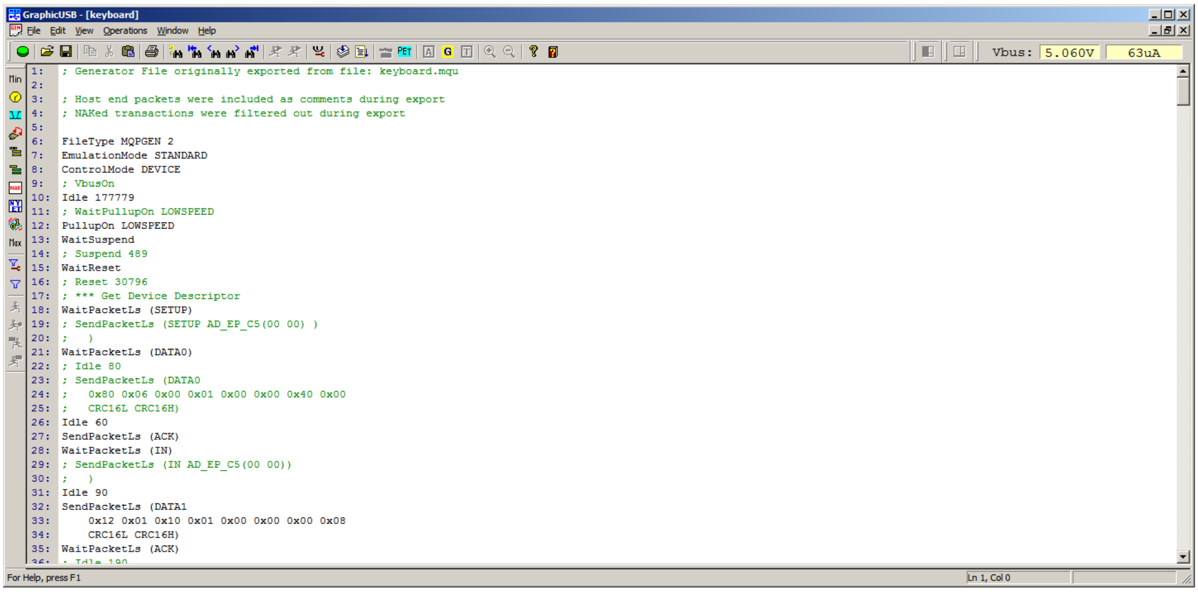
Data Content

```
00000000:  12 01 10 01 00 00 00 08 3C 41 05 20 04    ........<A. .
0000000D:  01 01 02 00 01                             .....
```

For Help, press F1 — 186 events

Figure 28: The GraphicUSB packet capture window

Next, you need to create a generator script:



```
; Generator File originally exported from file: keyboard.mqu

; Host end packets were included as comments during export
; NAKed transactions were filtered out during export

FileType MQPGEN 2
EmulationMode STANDARD
ControlMode DEVICE
; VbusOn
Idle 177779
; WaitPullupOn LOWSPEED
PullupOn LOWSPEED
WaitSuspend
; Suspend 489
WaitReset
; Reset 30796
; *** Get Device Descriptor
WaitPacketLs (SETUP)
; SendPacketLs (SETUP AD_EP_C5(00 00) )
;    )
WaitPacketLs (DATA0)
; Idle 80
; SendPacketLs (DATA0
;    0x80 0x06 0x00 0x01 0x00 0x00 0x40 0x00
;    CRC16L CRC16H)
Idle 60
SendPacketLs (ACK)
WaitPacketLs (IN)
; SendPacketLs (IN AD_EP_C5(00 00))
;    )
Idle 90
SendPacketLs (DATA1
    0x12 0x01 0x10 0x01 0x00 0x00 0x00 0x08
    CRC16L CRC16H)
WaitPacketLs (ACK)
; Idle 190
```

Figure 29: The GraphicUSB generator script window

Launch Zulu and select **Input Method > Import USB Generator Script**:

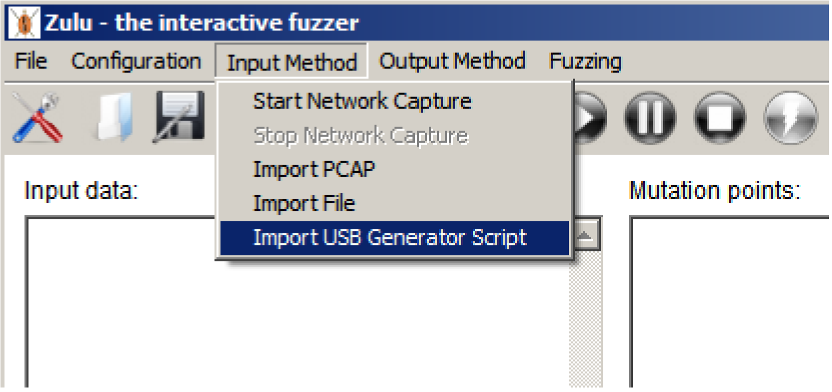
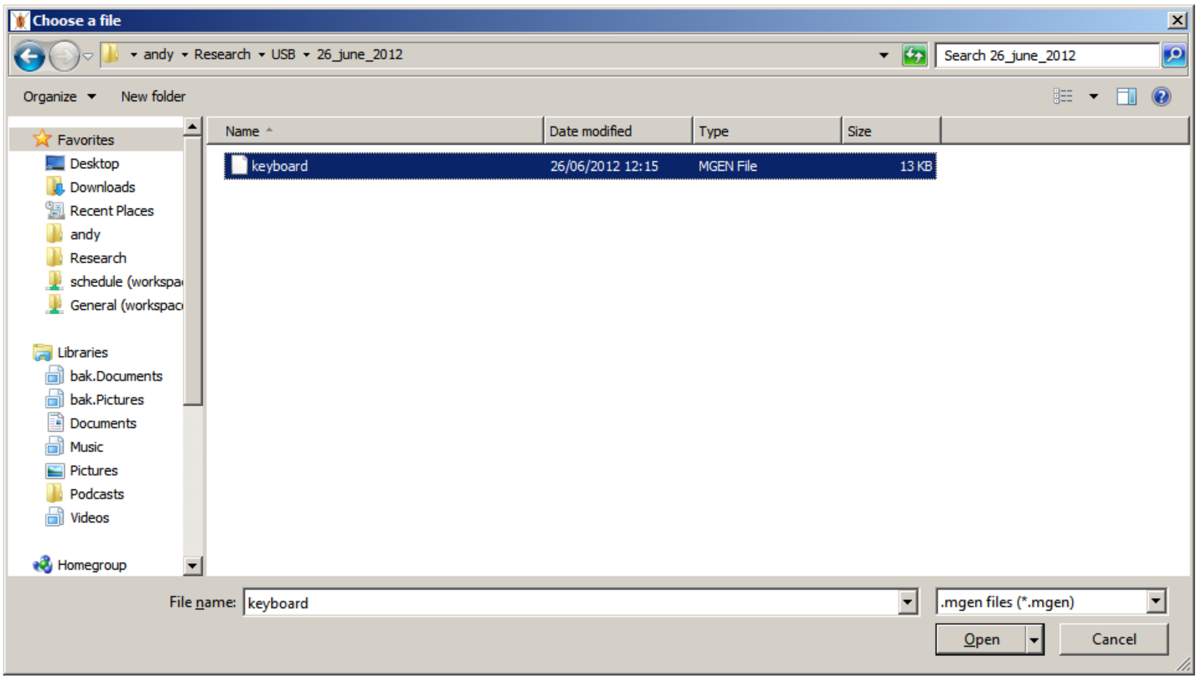


Figure 30: Importing a USB generator script into Zulu

Then select the saved script:



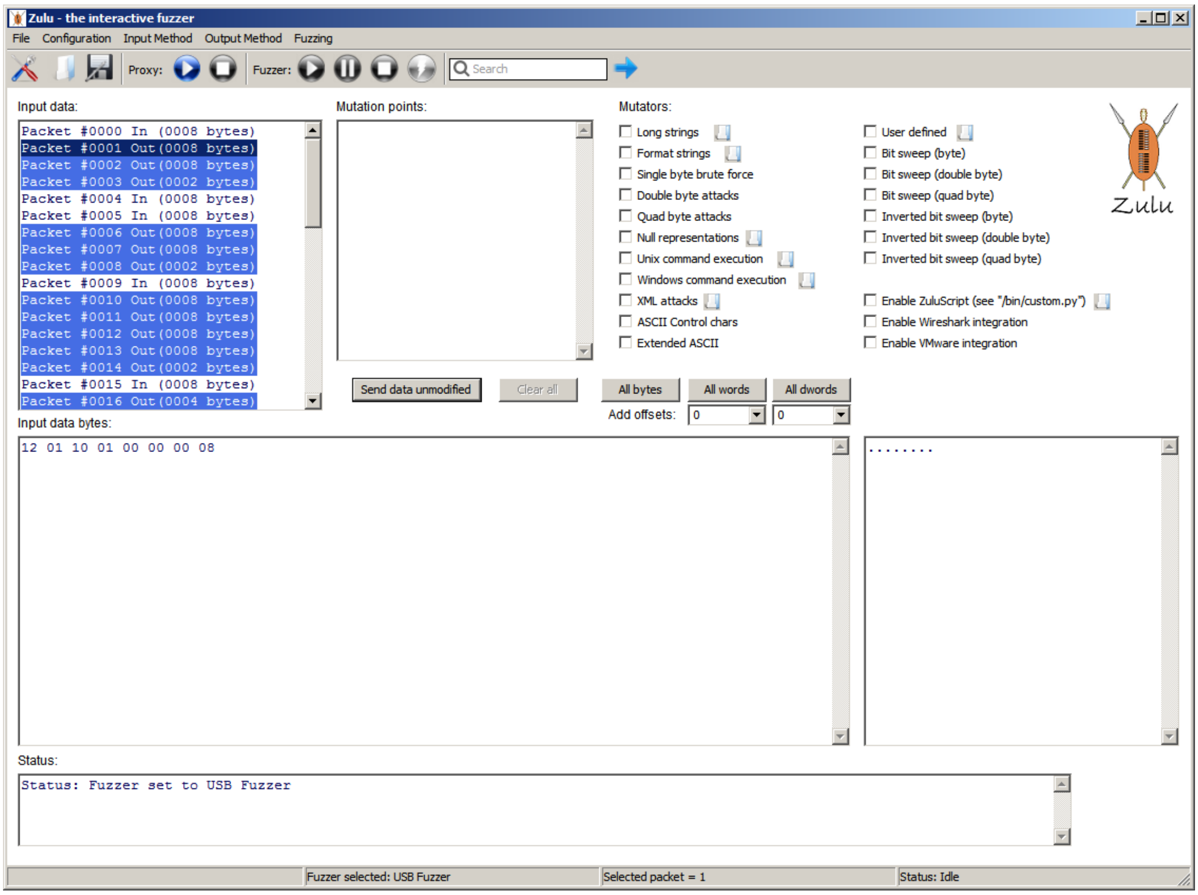
Name	Date modified	Type	Size
keyboard	26/06/2012 12:15	MGEN File	13 KB

File name: keyboard

.mgen files (*.mgen)

Figure 31: Selecting the generator script

The packets are loaded into Zulu just as if they were network packets:



```
Packet #0000 In  (0008 bytes)
Packet #0001 Out (0008 bytes)
Packet #0002 Out (0008 bytes)
Packet #0003 Out (0002 bytes)
Packet #0004 In  (0008 bytes)
Packet #0005 In  (0008 bytes)
Packet #0006 Out (0008 bytes)
Packet #0007 Out (0008 bytes)
Packet #0008 Out (0002 bytes)
Packet #0009 In  (0008 bytes)
Packet #0010 Out (0008 bytes)
Packet #0011 Out (0008 bytes)
Packet #0012 Out (0008 bytes)
Packet #0013 Out (0008 bytes)
Packet #0014 Out (0002 bytes)
Packet #0015 In  (0008 bytes)
Packet #0016 Out (0004 bytes)
```

- □ Long strings
- □ Format strings
- □ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- □ Null representations
- □ Unix command execution
- □ Windows command execution
- □ XML attacks
- □ ASCII Control chars
- □ Extended ASCII
- □ User defined
- □ Bit sweep (byte)
- □ Bit sweep (double byte)
- □ Bit sweep (quad byte)
- □ Inverted bit sweep (byte)
- □ Inverted bit sweep (double byte)
- □ Inverted bit sweep (quad byte)
- □ Enable ZuluScript (see "/bin/custom.py")
- □ Enable Wireshark integration
- □ Enable VMware integration

Add offsets: 0 0

```
12 01 10 01 00 00 00 08
```

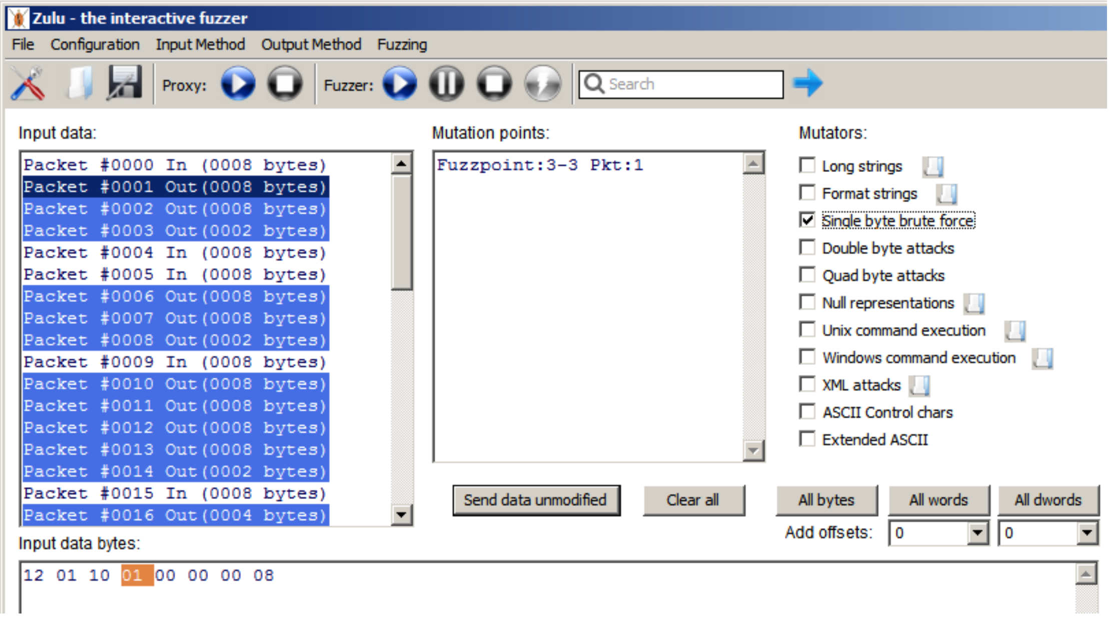
```
........
```

```
Status: Fuzzer set to USB Fuzzer
```

Fuzzer selected: USB Fuzzer | Selected packet = 1 | Status: Idle

Figure 32: USB generator script imported into Zulu

Add a fuzzpoint and select your Mutators:



```
Packet #0000 In  (0008 bytes)
Packet #0001 Out (0008 bytes)
Packet #0002 Out (0008 bytes)
Packet #0003 Out (0002 bytes)
Packet #0004 In  (0008 bytes)
Packet #0005 In  (0008 bytes)
Packet #0006 Out (0008 bytes)
Packet #0007 Out (0008 bytes)
Packet #0008 Out (0002 bytes)
Packet #0009 In  (0008 bytes)
Packet #0010 Out (0008 bytes)
Packet #0011 Out (0008 bytes)
Packet #0012 Out (0008 bytes)
Packet #0013 Out (0008 bytes)
Packet #0014 Out (0002 bytes)
Packet #0015 In  (0008 bytes)
Packet #0016 Out (0004 bytes)
```

```
Fuzzpoint:3-3 Pkt:1
```

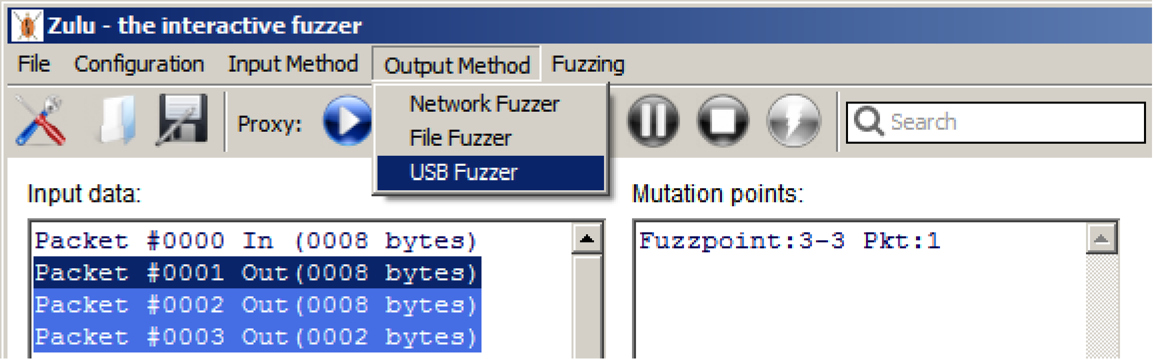
- □ Long strings
- □ Format strings
- ☑ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- □ Null representations
- □ Unix command execution
- □ Windows command execution
- □ XML attacks
- □ ASCII Control chars
- □ Extended ASCII

Add offsets: 0 0

```
12 01 10 01 00 00 00 08
```

Figure 33: Adding fuzzpoints and mutators to USB data

Select **Output Method > USB Fuzzer**:

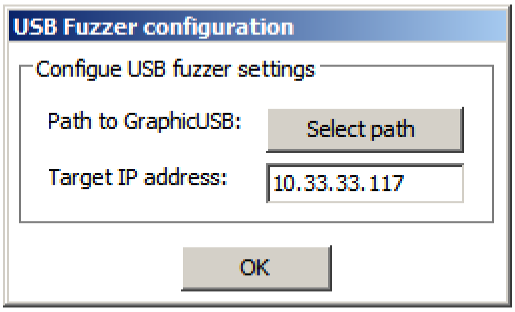


Packet #0000 In (0008 bytes)
Packet #0001 Out(0008 bytes)
Packet #0002 Out(0008 bytes)
Packet #0003 Out(0002 bytes)

Fuzzpoint:3-3 Pkt:1

Figure 34: Selecting the USB fuzzer module

The **USB Fuzzer configuration** window will be displayed:



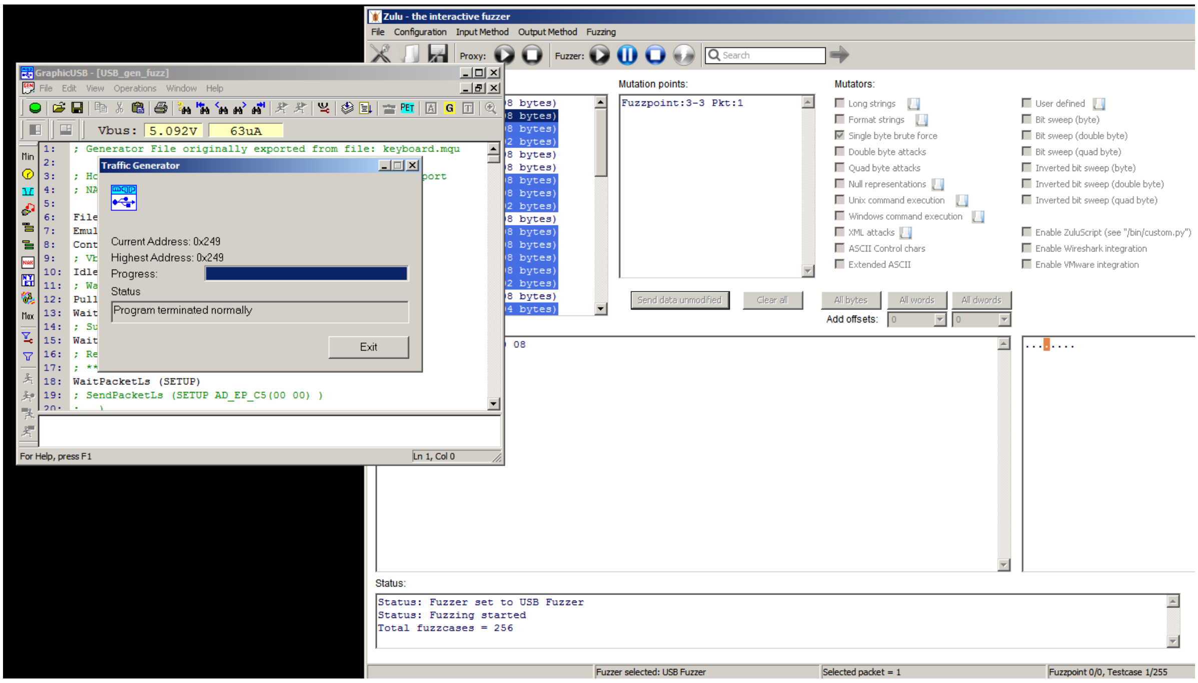
Path to GraphicUSB:

Target IP address: 10.33.33.117

Figure 35: USB fuzzer configuration

- **Path to GraphicUSB** - this is the USB application that will be controlled by Zulu
- **Target IP address** - If the target is connected to a network then ICMP is used for instrumentation, as most USB vulnerabilities result in kernel panics

Finally, start the fuzzer and Zulu will automatically control the GraphicUSB software in order to control the Packet-Master USB analyser hardware:



Vbus: 5.092V 63uA

```
; Generator File originally exported from file: keyboard.mqu

; Hc                                             port
; NA

File
Emul
Cont
; Vb
Idle
; Wa
Pull
Wait
; Su
Wait
; Re
; **
WaitPacketLs (SETUP)
; SendPacketLs (SETUP AD_EP_C5(00 00) )
```

Traffic Generator

Current Address: 0x249
Highest Address: 0x249
Progress:
Status
Program terminated normally

For Help, press F1 | Ln 1, Col 0

Mutation points:

```
Fuzzpoint:3-3 Pkt:1
```

Mutators:

- □ Long strings
- □ Format strings
- ☑ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- □ Null representations
- □ Unix command execution
- □ Windows command execution
- □ XML attacks
- □ ASCII Control chars
- □ Extended ASCII
- □ User defined
- □ Bit sweep (byte)
- □ Bit sweep (double byte)
- □ Bit sweep (quad byte)
- □ Inverted bit sweep (byte)
- □ Inverted bit sweep (double byte)
- □ Inverted bit sweep (quad byte)
- □ Enable ZuluScript (see "/bin/custom.py")
- □ Enable Wireshark integration
- □ Enable VMware integration

Send data unmodified | Clear all | All bytes | All words | All dwords

Add offsets: 0 | 0

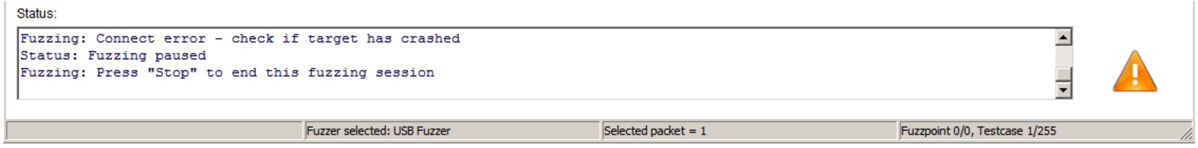
Status:

```
Status: Fuzzer set to USB Fuzzer
Status: Fuzzing started
Total fuzzcases = 256
```

Fuzzer selected: USB Fuzzer | Selected packet = 1 | Fuzzpoint 0/0, Testcase 1/255

Figure 36: The USB fuzzer running

If a crash is detected, the status will be updated as follows:



Status:

```
Fuzzing: Connect error - check if target has crashed
Status: Fuzzing paused
Fuzzing: Press "Stop" to end this fuzzing session
```

Fuzzer selected: USB Fuzzer | Selected packet = 1 | Fuzzpoint 0/0, Testcase 1/255

Figure 37: The target USB host crashing

7 Tutorial Six: Using the Serial fuzzer module

In this tutorial we will perform some serial fuzzing of a modem using the Hayes AT command set.

First, select **Input Method > Serial Data Capture**:

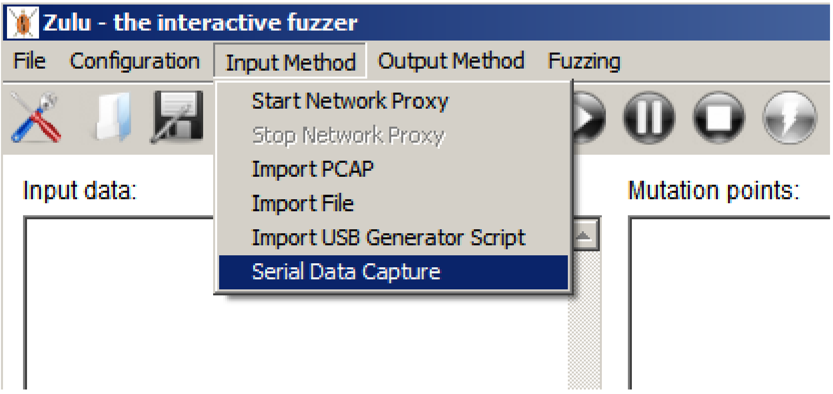


Figure 38: Selecting serial data capture

Click the **Port Settings** button:

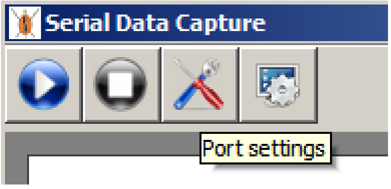
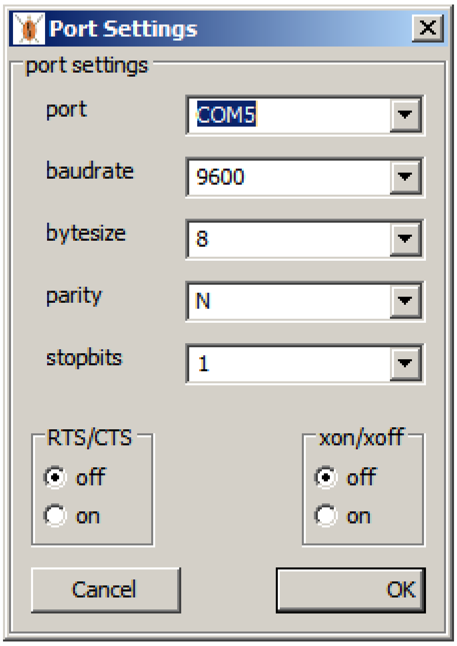


Figure 39: Selecting serial port settings

Select the serial port settings to use and click OK:



port settings

port: COM5

baudrate: 9600

bytesize: 8

parity: N

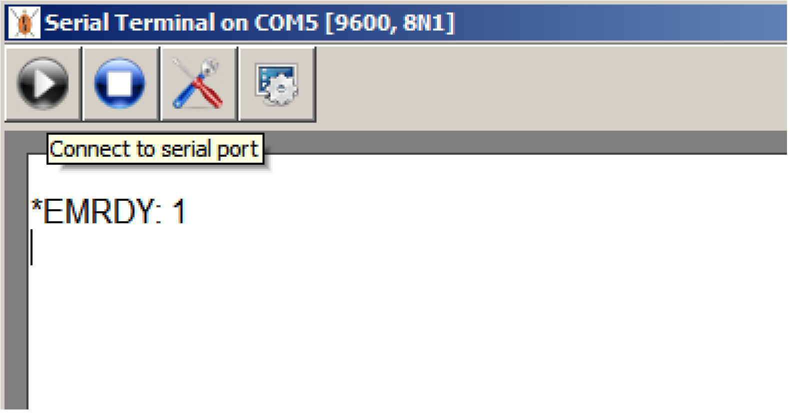
stopbits: 1

RTS/CTS: ● off ○ on

xon/xoff: ● off ○ on

Figure 40: Serial port settings for capture

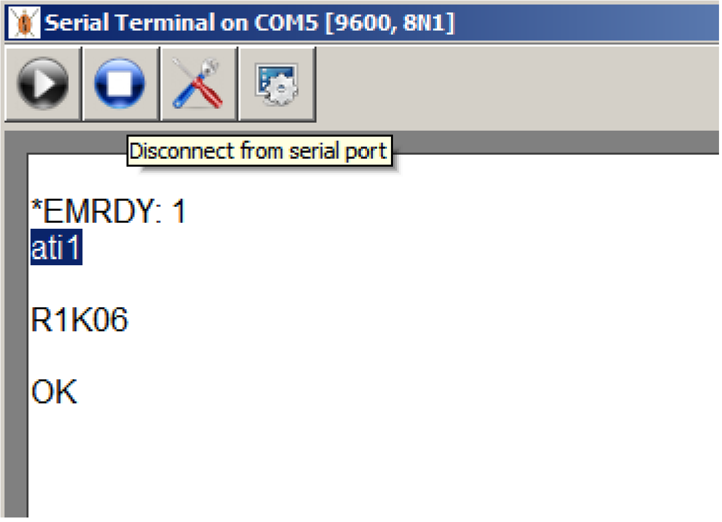
Click the **Connect to serial port** button:



```
*EMRDY: 1
```

Figure 41: Connecting to the serial port

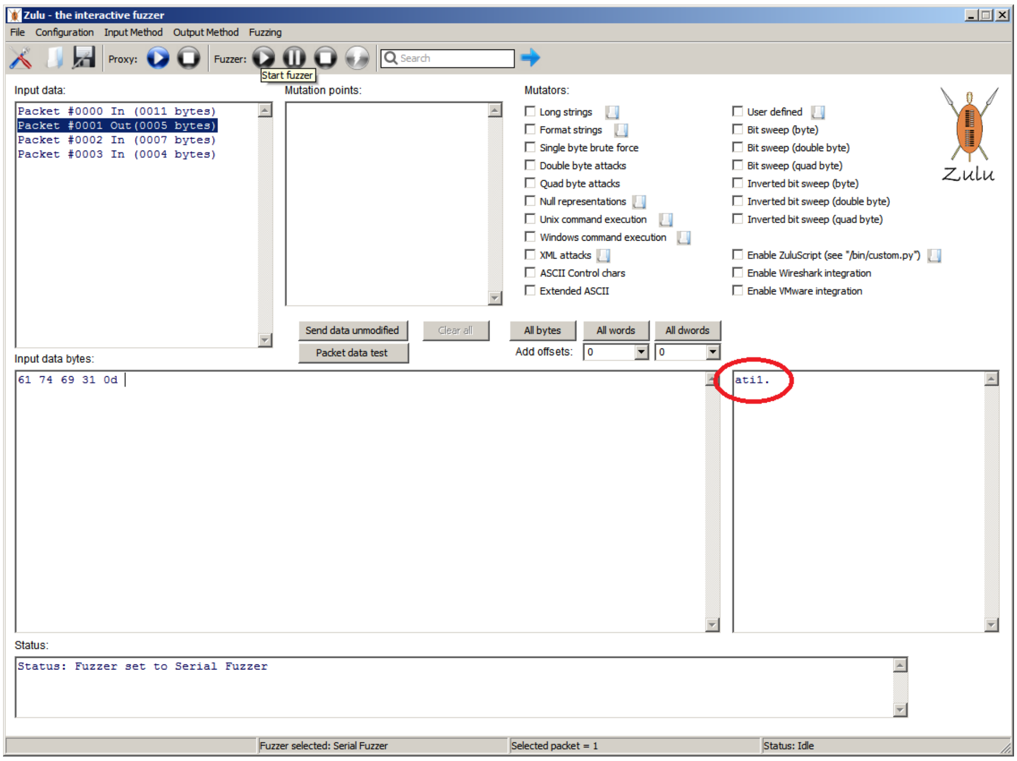
Enter some serial data - in this instance the AT command *ati1*:


Serial Terminal on COM5 [9600, 8N1]
Disconnect from serial port

```
*EMRDY: 1
ati1

R1K06

OK
```

Figure 42: Capturing serial data

When you have finished, click the **Disconnect from serial port** button:



```
Packet #0000 In (0011 bytes)
Packet #0001 Out(0005 bytes)
Packet #0002 In (0007 bytes)
Packet #0003 In (0004 bytes)
```

- □ Long strings
- □ Format strings
- □ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- □ Null representations
- □ Unix command execution
- □ Windows command execution
- □ XML attacks
- □ ASCII Control chars
- □ Extended ASCII
- □ User defined
- □ Bit sweep (byte)
- □ Bit sweep (double byte)
- □ Bit sweep (quad byte)
- □ Inverted bit sweep (byte)
- □ Inverted bit sweep (double byte)
- □ Inverted bit sweep (quad byte)
- □ Enable ZuluScript (see "/bin/custom.py")
- □ Enable Wireshark integration
- □ Enable VMware integration

Add offsets: 0 0

```
61 74 69 31 0d
```

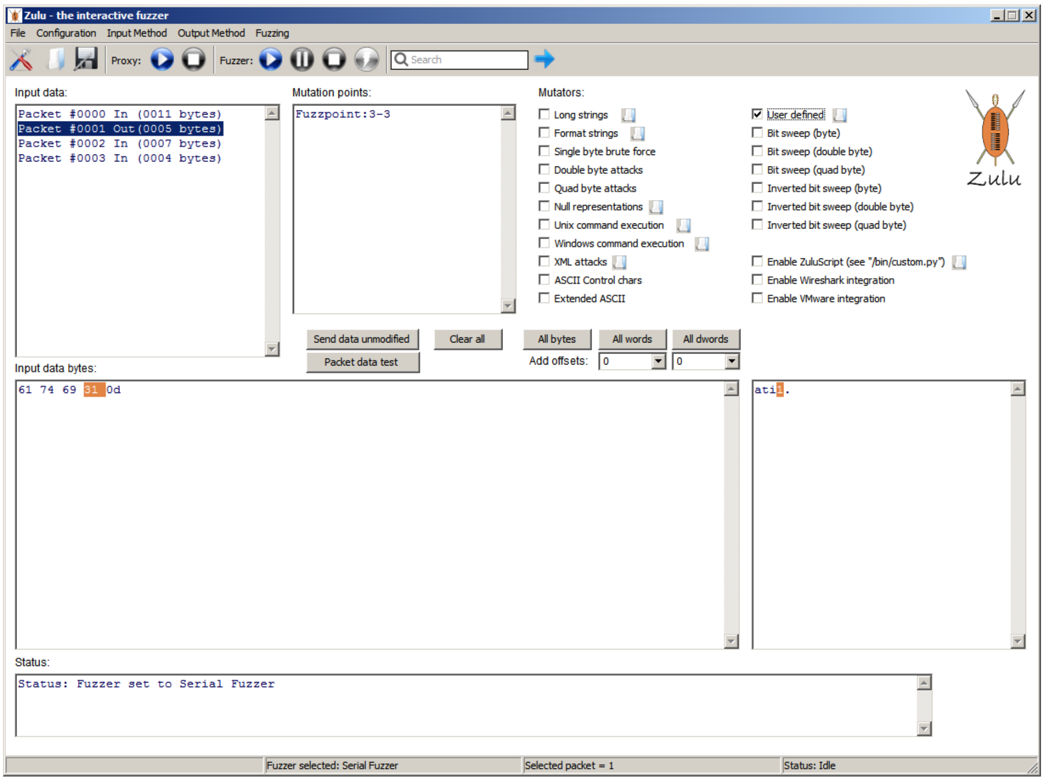
```
ati1.
```

```
Status: Fuzzer set to Serial Fuzzer
```

Fuzzer selected: Serial Fuzzer | Selected packet = 1 | Status: Idle

Figure 43: The captured serial data displayed in Zulu

Add any fuzzpoints you wish - here we are fuzzing the number after the *ati* command:



```
Packet #0000 In (0011 bytes)
Packet #0001 Out(0005 bytes)
Packet #0002 In (0007 bytes)
Packet #0003 In (0004 bytes)
```

```
Fuzzpoint:3-3
```

- □ Long strings
- □ Format strings
- □ Single byte brute force
- □ Double byte attacks
- □ Quad byte attacks
- □ Null representations
- □ Unix command execution
- □ Windows command execution
- □ XML attacks
- □ ASCII Control chars
- □ Extended ASCII
- ☑ User defined
- □ Bit sweep (byte)
- □ Bit sweep (double byte)
- □ Bit sweep (quad byte)
- □ Inverted bit sweep (byte)
- □ Inverted bit sweep (double byte)
- □ Inverted bit sweep (quad byte)
- □ Enable ZuluScript (see "/bin/custom.py")
- □ Enable Wireshark integration
- □ Enable VMware integration

Add offsets: 0 0

```
61 74 69 31 0d
```

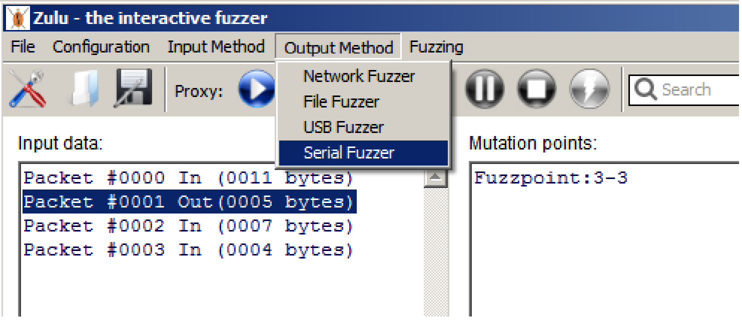
```
ati1.
```

```
Status: Fuzzer set to Serial Fuzzer
```

Fuzzer selected: Serial Fuzzer | Selected packet = 1 | Status: Idle

Figure 44: Selecting fuzzpoints in the serial data

Select **Output Method > Serial Fuzzer**:

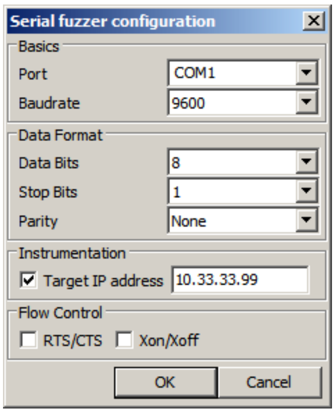


```
Packet #0000 In (0011 bytes)
Packet #0001 Out(0005 bytes)
Packet #0002 In (0007 bytes)
Packet #0003 In (0004 bytes)
```

```
Fuzzpoint:3-3
```

Figure 45: Selecting the serial fuzzer module

Select the serial port settings to use for fuzzing along with an IP address of the device (if appropriate) for instrumentation and click OK:


Serial fuzzer configuration

Basics
Port: COM1
Baudrate: 9600

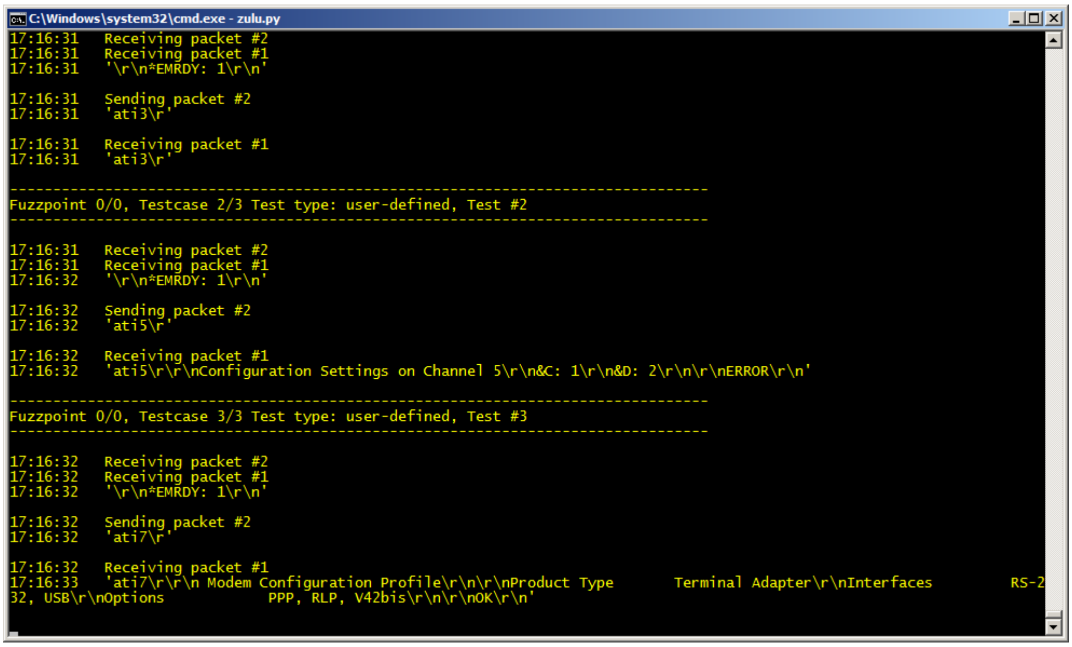
Data Format
Data Bits: 8
Stop Bits: 1
Parity: None

Instrumentation
☑ Target IP address: 10.33.33.99

Flow Control
☐ RTS/CTS ☐ Xon/Xoff

Figure 46: Selecting the target serial settings

Start fuzzing:

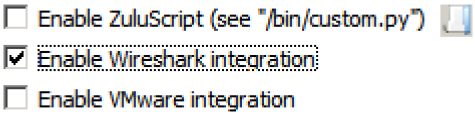


```
17:16:31   Receiving packet #2
17:16:31   Receiving packet #1
17:16:31   '\r\n*EMRDY: 1\r\n'

17:16:31   Sending packet #2
17:16:31   'ati3\r'

17:16:31   Receiving packet #1
17:16:31   'ati3\r'

--------------------------------------------------------------------------------
Fuzzpoint 0/0, Testcase 2/3 Test type: user-defined, Test #2
--------------------------------------------------------------------------------

17:16:31   Receiving packet #2
17:16:31   Receiving packet #1
17:16:32   '\r\n*EMRDY: 1\r\n'

17:16:32   Sending packet #2
17:16:32   'ati5\r'

17:16:32   Receiving packet #1
17:16:32   'ati5\r\r\nConfiguration Settings on Channel 5\r\n&C: 1\r\n&D: 2\r\n\r\nERROR\r\n'

--------------------------------------------------------------------------------
Fuzzpoint 0/0, Testcase 3/3 Test type: user-defined, Test #3
--------------------------------------------------------------------------------

17:16:32   Receiving packet #2
17:16:32   Receiving packet #1
17:16:32   '\r\n*EMRDY: 1\r\n'

17:16:32   Sending packet #2
17:16:32   'ati7\r'

17:16:32   Receiving packet #1
17:16:33   'ati7\r\r\n Modem Configuration Profile\r\n\r\nProduct Type      Terminal Adapter\r\nInterfaces        RS-2
32, USB\r\nOptions            PPP, RLP, V42bis\r\n\r\nOK\r\n'
```

Figure 47: The console output during serial fuzzing

8 Tutorial Seven: Integrating with Wireshark

In this tutorial we will integrate Zulu with Wireshark to take advantage of its dissectors to interpret network protocols. First, configure Zulu to perform a network capture, but before starting the capture select the **Enable Wireshark integration** checkbox:



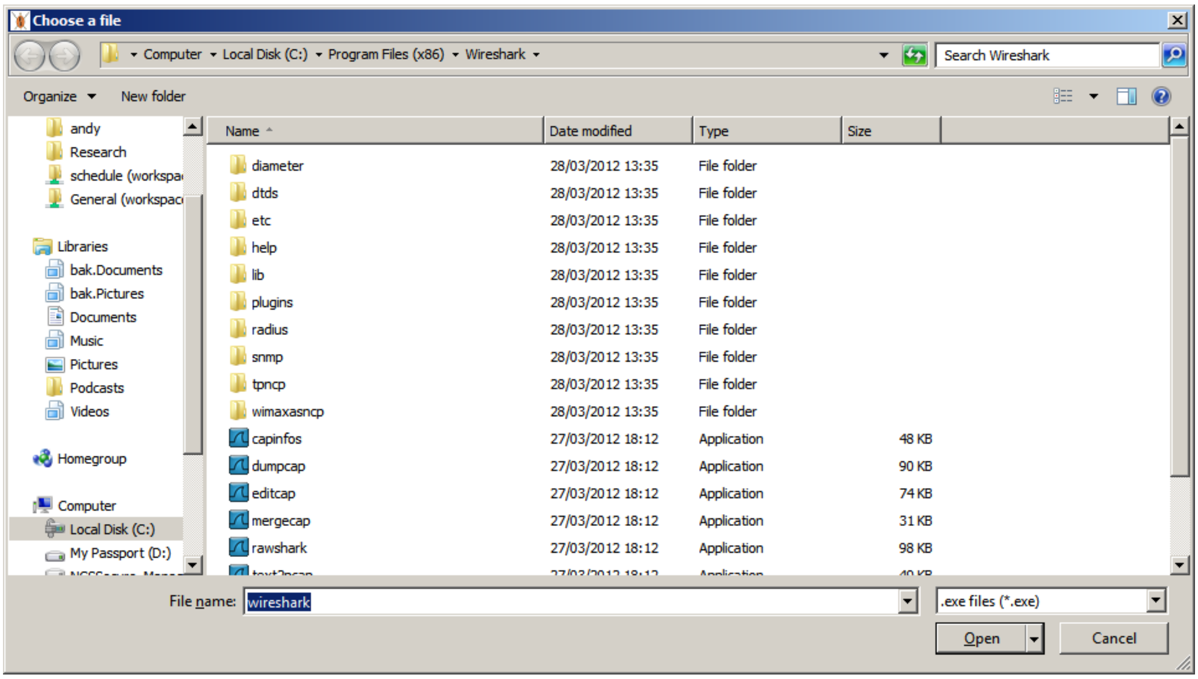
□ Enable ZuluScript (see "/bin/custom.py")

☑ Enable Wireshark integration

□ Enable VMware integration

Figure 48: Enabling wireshark integration

Zulu will then ask where the Wireshark executable is located (opening in the default location). Select the executable:



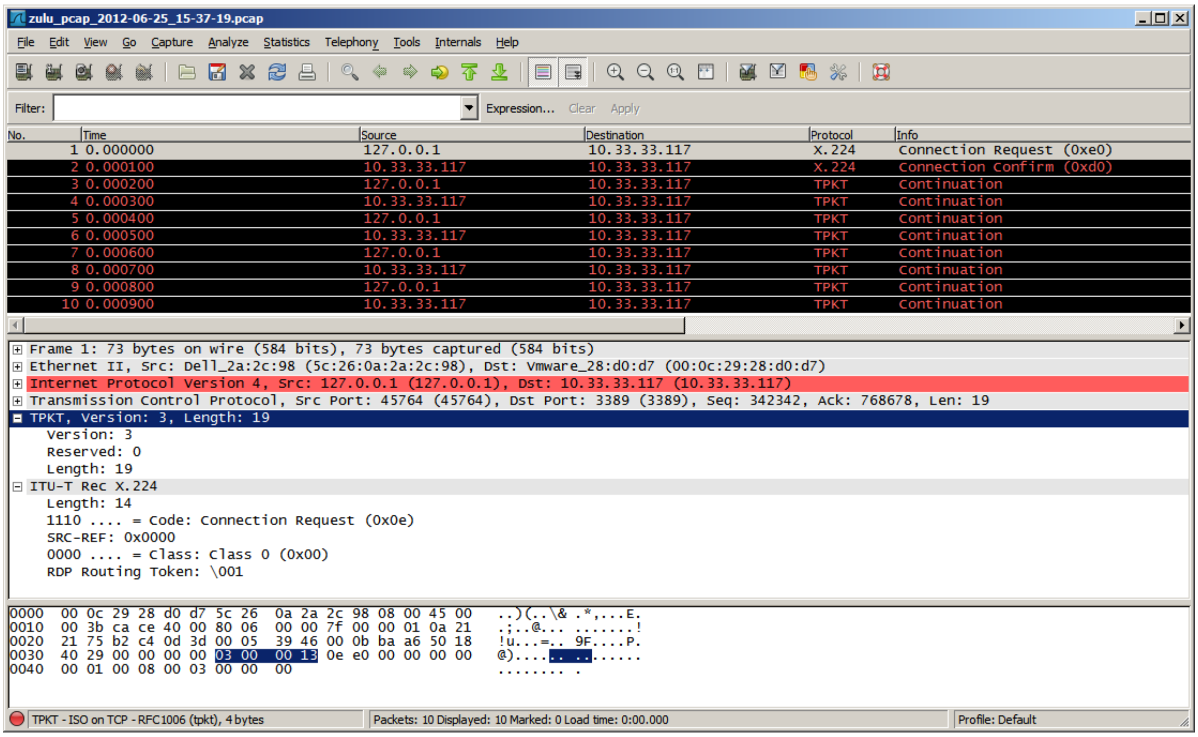
Name	Date modified	Type	Size
diameter	28/03/2012 13:35	File folder	
dtds	28/03/2012 13:35	File folder	
etc	28/03/2012 13:35	File folder	
help	28/03/2012 13:35	File folder	
lib	28/03/2012 13:35	File folder	
plugins	28/03/2012 13:35	File folder	
radius	28/03/2012 13:35	File folder	
snmp	28/03/2012 13:35	File folder	
tpncp	28/03/2012 13:35	File folder	
wimaxasncp	28/03/2012 13:35	File folder	
capinfos	27/03/2012 18:12	Application	48 KB
dumpcap	27/03/2012 18:12	Application	90 KB
editcap	27/03/2012 18:12	Application	74 KB
mergecap	27/03/2012 18:12	Application	31 KB
rawshark	27/03/2012 18:12	Application	98 KB
text2pcap	27/03/2012 18:12	Application	40 KB

File name: wireshark

.exe files (*.exe)

Figure 49: Selecting the Wireshark binary

Next start the network capture (**Input Method > Start Network Capture**). When all the traffic required has been captured and the capture has been stopped (**Input Method > Stop Network Capture**), a PCAP file of the captured traffic is generated and provided to Wireshark which is opened to display the packets captured by Zulu:


zulu_pcap_2012-06-25_15-37-19.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
No.   Time       Source          Destination     Protocol  Info
    1 0.000000   127.0.0.1       10.33.33.117    X.224     Connection Request (0xe0)
    2 0.000100   10.33.33.117    10.33.33.117    X.224     Connection Confirm (0xd0)
    3 0.000200   127.0.0.1       10.33.33.117    TPKT      Continuation
    4 0.000300   10.33.33.117    10.33.33.117    TPKT      Continuation
    5 0.000400   127.0.0.1       10.33.33.117    TPKT      Continuation
    6 0.000500   10.33.33.117    10.33.33.117    TPKT      Continuation
    7 0.000600   127.0.0.1       10.33.33.117    TPKT      Continuation
    8 0.000700   10.33.33.117    10.33.33.117    TPKT      Continuation
    9 0.000800   127.0.0.1       10.33.33.117    TPKT      Continuation
   10 0.000900   10.33.33.117    10.33.33.117    TPKT      Continuation
```

```
⊞ Frame 1: 73 bytes on wire (584 bits), 73 bytes captured (584 bits)
⊞ Ethernet II, Src: Dell_2a:2c:98 (5c:26:0a:2a:2c:98), Dst: Vmware_28:d0:d7 (00:0c:29:28:d0:d7)
⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 10.33.33.117 (10.33.33.117)
⊞ Transmission Control Protocol, Src Port: 45764 (45764), Dst Port: 3389 (3389), Seq: 342342, Ack: 768678, Len: 19
⊟ TPKT, Version: 3, Length: 19
    Version: 3
    Reserved: 0
    Length: 19
⊟ ITU-T Rec X.224
    Length: 14
    1110 .... = Code: Connection Request (0x0e)
    SRC-REF: 0x0000
    0000 .... = Class: Class 0 (0x00)
    RDP Routing Token: \001
```

```
0000  00 0c 29 28 d0 d7 5c 26  0a 2a 2c 98 08 00 45 00   ..)(..\& .*,...E.
0010  00 3b ca ce 40 00 80 06  00 00 7f 00 00 01 0a 21   .;..@... .......!
0020  21 75 b2 c4 0d 3d 00 05  39 46 00 0b ba a6 50 18   !u...=.. 9F....P.
0030  40 29 00 00 00 00 03 00  00 13 0e e0 00 00 00 00   @)...... ........
0040  00 01 00 08 00 03 00 00  00                        ........ .
```

TPKT - ISO on TCP - RFC1006 (tpkt), 4 bytes
Packets: 10 Displayed: 10 Marked: 0 Load time: 0:00.000
Profile: Default

Figure 50: Wireshark displaying the data captured by Zulu

Note: Ignore any Wireshark warnings about packet ordering – the PCAP file is generated from scratch so the TCP sequence numbers and packet timestamps are “invented”

9 Tutorial Eight: Integrating with VMware

In this tutorial we show how VMware can be controlled by Zulu to either restart guest processes or the whole VM.

Before starting any fuzzing perform the steps below, firstly, select **Configuration > VMware settings**:

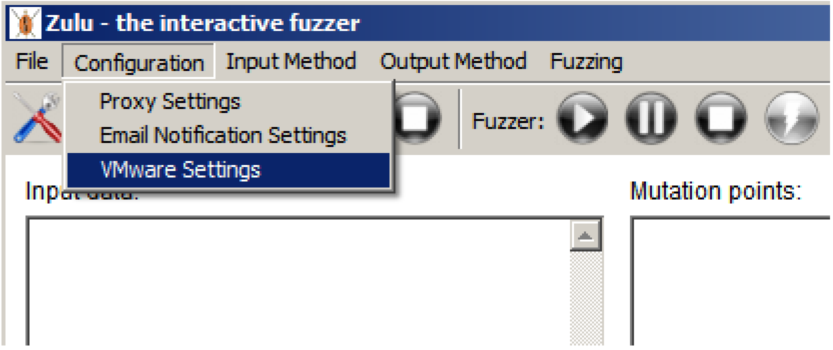
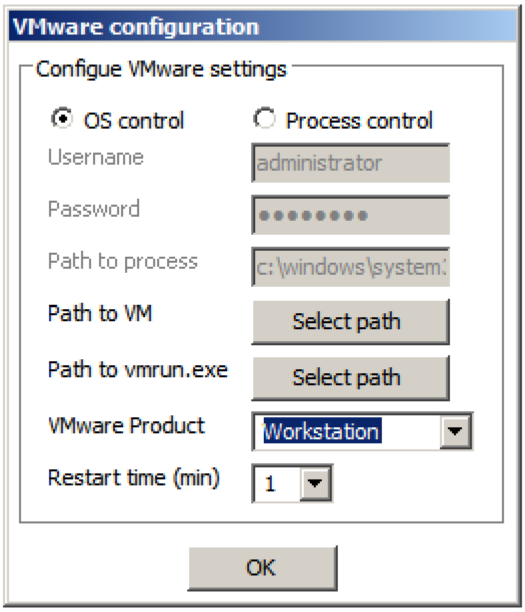


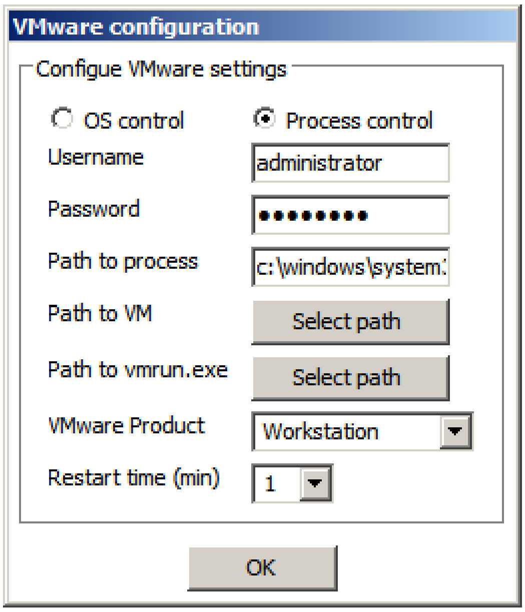
Figure 51: Selecting VMware settings

Depending on whether you want to control a VM guest process of the whole VM, choose either **OS Control** or **Process Control**:



● OS control ○ Process control
Username: administrator
Password: ••••••••
Path to process: c:\windows\system:
VMware Product: Workstation
Restart time (min): 1

Figure 52: VMware OS control settings



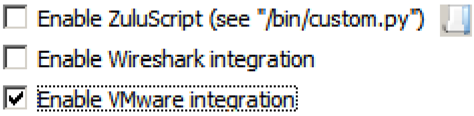
○ OS control ● Process control
Username: administrator
Password: ••••••••
Path to process: c:\windows\system:
VMware Product: Workstation
Restart time (min): 1

Figure 53: VMware process control settings

- **Username** - the username of a guest OS administrator
- **Password** - the password of the guest OS administrator
- **Path to process** - the full path to the process in the guest OS to control
- **Path to VM** - self explanatory
- **Path to vmrun.exe** - vmrun is the command line tool that Zulu uses to control VMware (it is installed by default)
- **VMware product** - self explanatory
- **Restart time** - the time to wait for the VM to restart (if **OS Control** is selected)

Note: If the Zulu session is saved, the password is currently stored as cleartext in the session file

Before starting fuzzing, select the Enable VMware integration checkbox:



☐ Enable ZuluScript (see "/bin/custom.py")
☐ Enable Wireshark integration
☑ Enable VMware integration

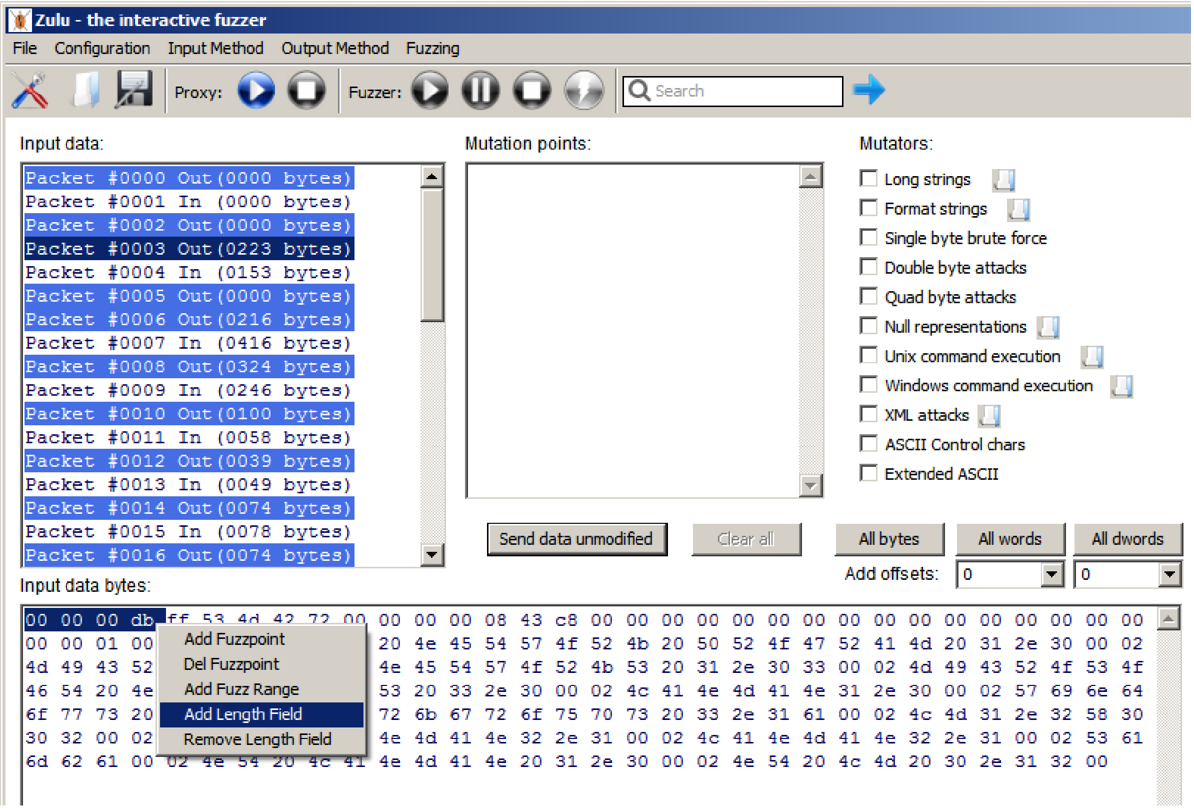
Figure 54: Enabling VMware integration

If during fuzzing the target crashes, Zulu will then control the VM as configured above.

10 Tutorial Nine: Adding length fields

In this tutorial we will add a length field to a network packet. This would be required if the mutator used changes the length of a packet and a length field needs to be updated, based on the new size of each fuzzcase.

First, select the bytes that represent the length field (this can be either one, two or four bytes). Then right click and select **Add Length Field**:



```
Packet #0000 Out(0000 bytes)
Packet #0001 In  (0000 bytes)
Packet #0002 Out(0000 bytes)
Packet #0003 Out(0223 bytes)
Packet #0004 In  (0153 bytes)
Packet #0005 Out(0000 bytes)
Packet #0006 Out(0216 bytes)
Packet #0007 In  (0416 bytes)
Packet #0008 Out(0324 bytes)
Packet #0009 In  (0246 bytes)
Packet #0010 Out(0100 bytes)
Packet #0011 In  (0058 bytes)
Packet #0012 Out(0039 bytes)
Packet #0013 In  (0049 bytes)
Packet #0014 Out(0074 bytes)
Packet #0015 In  (0078 bytes)
Packet #0016 Out(0074 bytes)
```

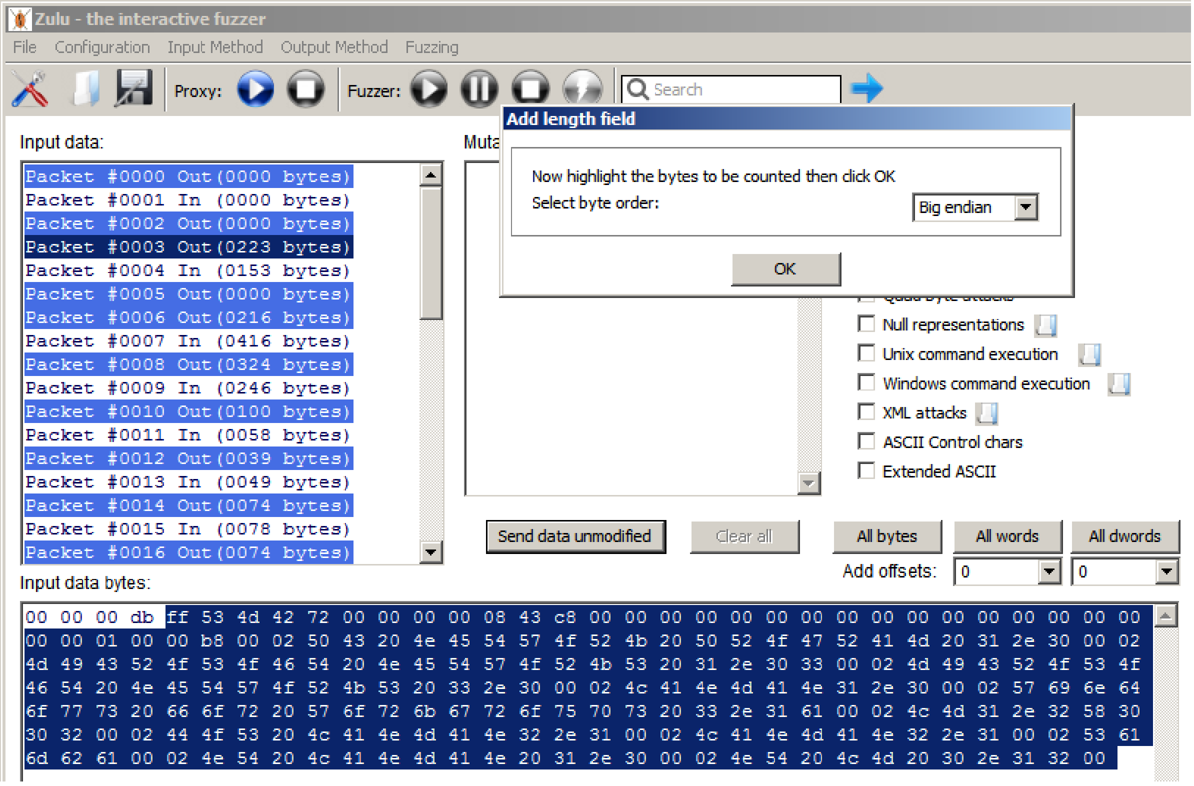
☐ Long strings
☐ Format strings
☐ Single byte brute force
☐ Double byte attacks
☐ Quad byte attacks
☐ Null representations
☐ Unix command execution
☐ Windows command execution
☐ XML attacks
☐ ASCII Control chars
☐ Extended ASCII

Add offsets: 0 0

```
00 00 00 db ff 53 4d 42 72 00 00 00 00 08 43 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00                   20 4e 45 54 57 4f 52 4b 20 50 52 4f 47 52 41 4d 20 31 2e 30 00 02
4d 49 43 52                   4e 45 54 57 4f 52 4b 53 20 31 2e 30 33 00 02 4d 49 43 52 4f 53 4f
46 54 20 4e                   53 20 33 2e 30 00 02 4c 41 4e 4d 41 4e 31 2e 30 00 02 57 69 6e 64
6f 77 73 20                   72 6b 67 72 6f 75 70 73 20 33 2e 31 61 00 02 4c 4d 31 2e 32 58 30
30 32 00 02                   4e 4d 41 4e 32 2e 31 00 02 4c 41 4e 4d 41 4e 32 2e 31 00 02 53 61
6d 62 61 00 02 4e 54 20 4c 41 4e 4d 41 4e 20 31 2e 30 00 02 4e 54 20 4c 4d 20 30 2e 31 32 00
```

Figure 55: Adding a length field to captured data

A dialogue box is then displayed, which asks you to now select the bytes to be counted (you can also select the byte order of the length field in the combo box). Select the bytes to be counted and click OK.



Now highlight the bytes to be counted then click OK
Select byte order: Big endian

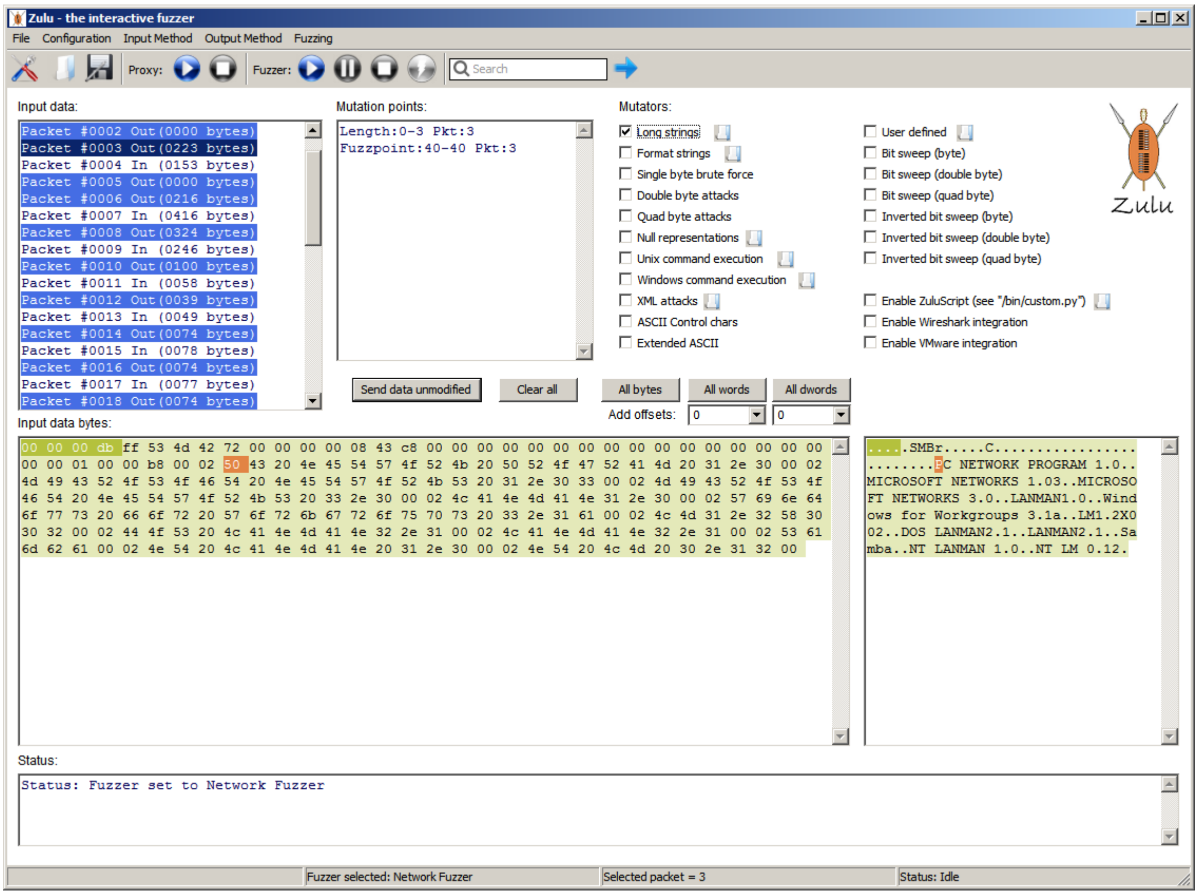
Add offsets: 0 0

```
Packet #0000 Out(0000 bytes)
Packet #0001 In  (0000 bytes)
Packet #0002 Out(0000 bytes)
Packet #0003 Out(0223 bytes)
Packet #0004 In  (0153 bytes)
Packet #0005 Out(0000 bytes)
Packet #0006 Out(0216 bytes)
Packet #0007 In  (0416 bytes)
Packet #0008 Out(0324 bytes)
Packet #0009 In  (0246 bytes)
Packet #0010 Out(0100 bytes)
Packet #0011 In  (0058 bytes)
Packet #0012 Out(0039 bytes)
Packet #0013 In  (0049 bytes)
Packet #0014 Out(0074 bytes)
Packet #0015 In  (0078 bytes)
Packet #0016 Out(0074 bytes)
```

```
00 00 00 db ff 53 4d 42 72 00 00 00 00 08 43 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 b8 00 02 50 43 20 4e 45 54 57 4f 52 4b 20 50 52 4f 47 52 41 4d 20 31 2e 30 00 02
4d 49 43 52 4f 53 4f 46 54 20 4e 45 54 57 4f 52 4b 53 20 31 2e 30 33 00 02 4d 49 43 52 4f 53 4f
46 54 20 4e 45 54 57 4f 52 4b 53 20 33 2e 30 00 02 4c 41 4e 4d 41 4e 31 2e 30 00 02 57 69 6e 64
6f 77 73 20 66 6f 72 20 57 6f 72 6b 67 72 6f 75 70 73 20 33 2e 31 61 00 02 4c 4d 31 2e 32 58 30
30 32 00 02 44 4f 53 20 4c 41 4e 4d 41 4e 32 2e 31 00 02 4c 41 4e 4d 41 4e 32 2e 31 00 02 53 61
6d 62 61 00 02 4e 54 20 4c 41 4e 4d 41 4e 20 31 2e 30 00 02 4e 54 20 4c 4d 20 30 2e 31 32 00
```

Figure 56: Highlighting the bytes to be counted

If a fuzzpoint is now selected within the configured length field calculation area and any of the mutators selected modify the length of the packet, the configured length field will now automatically be updated during the fuzzing process:



```
Packet #0002 Out(0000 bytes)
Packet #0003 Out(0223 bytes)
Packet #0004 In  (0153 bytes)
Packet #0005 Out(0000 bytes)
Packet #0006 Out(0216 bytes)
Packet #0007 In  (0416 bytes)
Packet #0008 Out(0324 bytes)
Packet #0009 In  (0246 bytes)
Packet #0010 Out(0100 bytes)
Packet #0011 In  (0058 bytes)
Packet #0012 Out(0039 bytes)
Packet #0013 In  (0049 bytes)
Packet #0014 Out(0074 bytes)
Packet #0015 In  (0078 bytes)
Packet #0016 Out(0074 bytes)
Packet #0017 In  (0077 bytes)
Packet #0018 Out(0074 bytes)
```

```
Length:0-3 Pkt:3
Fuzzpoint:40-40 Pkt:3
```

- ☑ Long strings
- ☐ Format strings
- ☐ Single byte brute force
- ☐ Double byte attacks
- ☐ Quad byte attacks
- ☐ Null representations
- ☐ Unix command execution
- ☐ Windows command execution
- ☐ XML attacks
- ☐ ASCII Control chars
- ☐ Extended ASCII
- ☐ User defined
- ☐ Bit sweep (byte)
- ☐ Bit sweep (double byte)
- ☐ Bit sweep (quad byte)
- ☐ Inverted bit sweep (byte)
- ☐ Inverted bit sweep (double byte)
- ☐ Inverted bit sweep (quad byte)
- ☐ Enable ZuluScript (see "/bin/custom.py")
- ☐ Enable Wireshark integration
- ☐ Enable VMware integration

Add offsets: 0 0

```
00 00 00 db ff 53 4d 42 72 00 00 00 00 08 43 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 b8 00 02 50 43 20 4e 45 54 57 4f 52 4b 20 50 52 4f 47 52 41 4d 20 31 2e 30 00 02
4d 49 43 52 4f 53 4f 46 54 20 4e 45 54 57 4f 52 4b 53 20 31 2e 30 33 00 02 4d 49 43 52 4f 53 4f
46 54 20 4e 45 54 57 4f 52 4b 53 20 33 2e 30 00 02 4c 41 4e 4d 41 4e 31 2e 30 00 02 57 69 6e 64
6f 77 73 20 66 6f 72 20 57 6f 72 6b 67 72 6f 75 70 73 20 33 2e 31 61 00 02 4c 4d 31 2e 32 58 30
30 32 00 02 44 4f 53 20 4c 41 4e 4d 41 4e 32 2e 31 00 02 4c 41 4e 4d 41 4e 32 2e 31 00 02 53 61
6d 62 61 00 02 4e 54 20 4c 41 4e 4d 41 4e 20 31 2e 30 00 02 4e 54 20 4c 4d 20 30 2e 31 32 00
```

```
.....SMBr.....C.................
........PC NETWORK PROGRAM 1.0..
MICROSOFT NETWORKS 1.03..MICROSO
FT NETWORKS 3.0..LANMAN1.0..Wind
ows for Workgroups 3.1a..LM1.2X0
02..DOS LANMAN2.1..LANMAN2.1..Sa
mba..NT LANMAN 1.0..NT LM 0.12.
```

```
Status: Fuzzer set to Network Fuzzer
```

Fuzzer selected: Network Fuzzer | Selected packet = 3 | Status: Idle

Figure 57: The length field is now inserted

11 Tutorial Ten: Configuring email notification settings

In this tutorial we will configure the email settings so that Zulu can send crash notifications via email. First, select **Configuration > Email notification** settings:

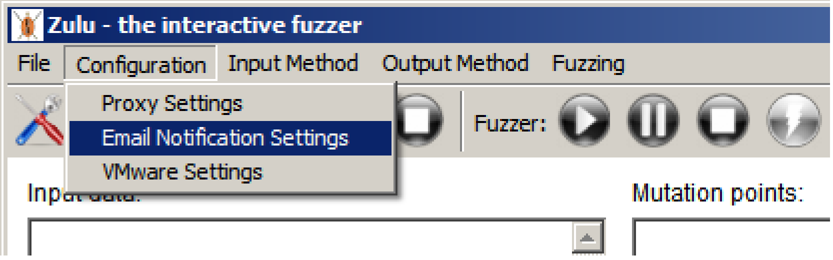
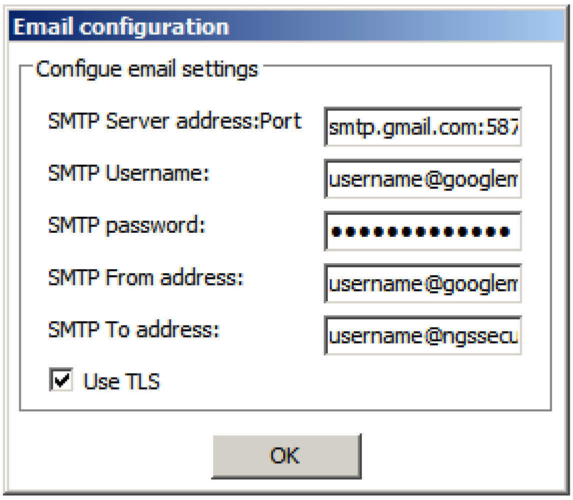


Figure 58: Selecting the email notification settings

This will display the **Email configuration** window:



Email configuration

Configue email settings

SMTP Server address:Port: smtp.gmail.com:587

SMTP Username: username@googlem

SMTP password: ●●●●●●●●●●●●●●

SMTP From address: username@googlem

SMTP To address: username@ngssecu

☑ Use TLS

Figure 59: The email notification settings

The example below, in brackets applies to Google Gmail

- **SMTP Server address:port** - the address and port that the SMTP server is listening on (smtp.gmail.com:587)
- **SMTP Username** - your login name (username@googlemail.com)
- **SMTP Password** - your password
- **SMTP From address** - your email address to send from (username@googlemail.com)
- **SMTP To address** - the address to send to (name.surname@nccgroup.com)
- **Use TLS** - select if encryption is used (select checkbox)

Note: If the Zulu session is saved then the SMTP password is currently stored in cleartext within the session file

If during network fuzzing a crash occurs, the auto-generated PoC will be emailed to the configured email account

12 Tutorial Eleven: Writing and using ZuluScript

In this tutorial we will introduce the concept of ZuluScript and look at an example.

There will sometimes be situations where a process needs to be performed on a packet after it has been modified by a mutator, but before it has been processed by the target - this is where ZuluScript can be used.

ZuluScript is just Python script stored in a special file (/bin/custom.py). The default file, which includes a *test()* function and an *UpdateContentLengthField()* function is shown below:

```
###################################################################################
# Zulu custom script file - when ZuluScript is enabled, this script will be executed prior to
each packet being sent
#                                (including unmodified packets), but after any fuzz data has
been applied to the packet
#
# Variables that can be referenced:
#
# self.packets_selected_to_send = list of packets selected to send [[packet number,
data],[packet number, data]...]
#
# self.all_packets_captured = list of all packets captured [[[source IP,source port],data],
[[source IP,source port],data]...]
#
# self.modified_data = list of all the data in the current packet (after any modification
with fuzzpoint data) [byte1, byte2, byte3...]
#
# self.current_packet_number = the number of the current packet being processed (packet 0 is
the first packet)
#
# Below are two example functions:
#
# test() just proves that ZuluScript is functioning correctly and demonstrates the data that
can be accessed
# UpdateContentLengthField() is an example script to update a Content Length field within an
HTTP packet
#
###################################################################################

class ZuluScript:
    """
    Zulu custom scripting interface
    """

    def   __init__(self,zulu):
          print "---ZuluScript started---"
          self.zulu = zulu

          #self.test()

          self.UpdateContentLengthField(0)

          print "---ZuluScript complete---"
          return



    def test(self):
          print "----------------------------"
          print "ZuluScript test:"
          print
          print "self.packets_selected_to_send"
          print
          print self.zulu.packets_selected_to_send
          print
          print "self.all_packets_captured"
          print
          print self.zulu.all_packets_captured
          print
          print "self.modified_data"
          print
          print self.zulu.modified_data
          print
          print "self.current_packet_number"
          print
          print self.zulu.current_packet_number
          print
          print "-----------------------------"
          return

    def   UpdateContentLengthField(self, packet_num):

          # UpdateContentLengthField() - Updates an HTTP Content length field within a packet
after fuzz data has been inserted

          # packet_num - the number of the packet containing the length field

          if self.zulu.current_packet_number != packet_num:
                   return

          length = 0
          lengthtext = ""
          length_lst = []
          data = ""
          temp_data = ""
          field_pos = 0

          x = 0
          while x < len(self.zulu.modified_data):
                   data += self.zulu.modified_data[x]
                   x+=1

          field_pos = data.find("Content-Length")
          if field_pos == -1:
                   print "No Content-Length header"
                   return

          field_pos += 15

          start = data.find("\r\n\r\n")
          if start == -1:
                   print "No POST data"
                   return

          start += 4

          end = len(self.zulu.modified_data)-1

          length = end - start
          lengthtext = "%d" % length

          x = 0
          while x < field_pos:
                   temp_data += self.zulu.modified_data[x]
                   x+=1
```

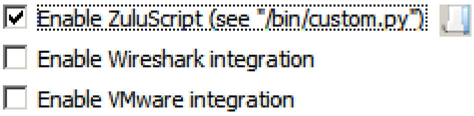
```
        temp_data += " "
        temp_data += lengthtext
        temp_data += "\x0a"

        while self.zulu.modified_data[x] != "\x0a":
                  x+=1

        x+=1

        while x < end:

                  temp_data += self.zulu.modified_data[x]
                  x+=1

        x=0
        self.zulu.modified_data = []
        while x < len(temp_data):
                  self.zulu.modified_data.append(temp_data[x])
                  x+=1
```

Figure 60: Example ZuluScript

As shown at the top of the file, there are a number of variables that can be referenced:

- **self.packets_selected_to_send** = list of packets selected to send [[packet number, data],[packet number, data]...]
- **self.all_packets_captured** = list of all packets captured [[[source IP,source port],data],[[source IP,source port],data]...]
- **self.modified_data** = list of all the data in the current packet (after any modification with fuzzpoint data) [byte1, byte2, byte3...]
- **self.current_packet_number** = the number of the current packet being processed (packet 0 is the first packet)
-

In the ZuluScript constructor you can see that it prints a message to say that ZuluScript has started, creates a "zulu" object and executes *self.UpdateContentLengthField(0)* which will update the HTTP Content-Length field in packet zero each time a fuzzcase is run. To enable ZuluScript, just select the **Enable ZuluScript** checkbox:



☑ Enable ZuluScript (see "/bin/custom.py")
□ Enable Wireshark integration
□ Enable VMware integration

Figure 61: Enabling ZuluScript

If you can write Python then this makes Zulu practically infinitely extendible.