



Fuzzing the easy way: Using Zulu

Andy Davis, Research Director NCC Group



Who am I?

- NCC Group Research Director
- >20 years in information security
- Still very hands-on
- Enjoy testing more unusual technologies
- Also developing tools to test them

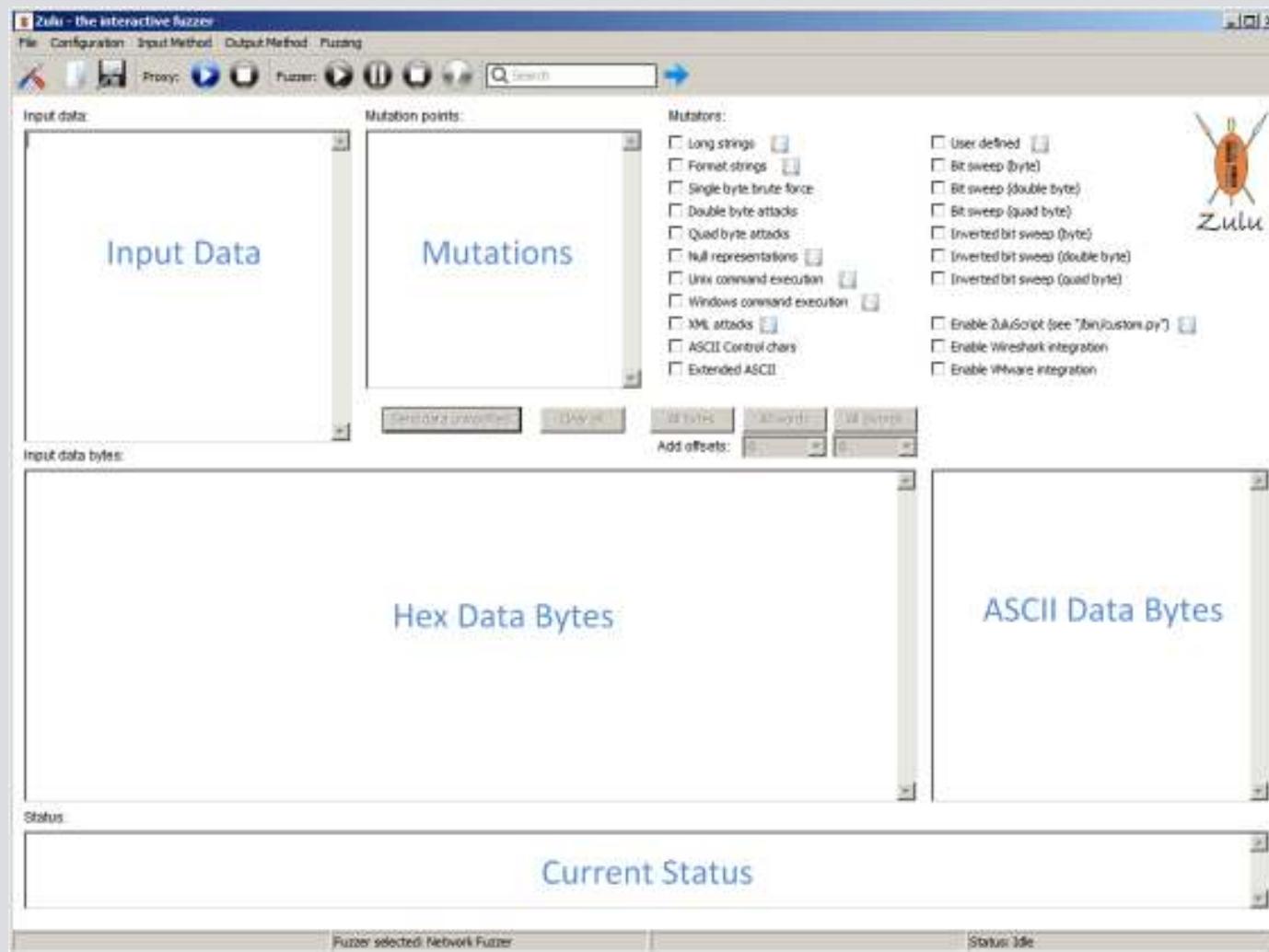
What is Zulu?

- Zulu is an interactive GUI-based fuzzer
- Written in Python
- As much as possible, input and output-agnostic
- Multiple modules
- Extendible via ZuluScript

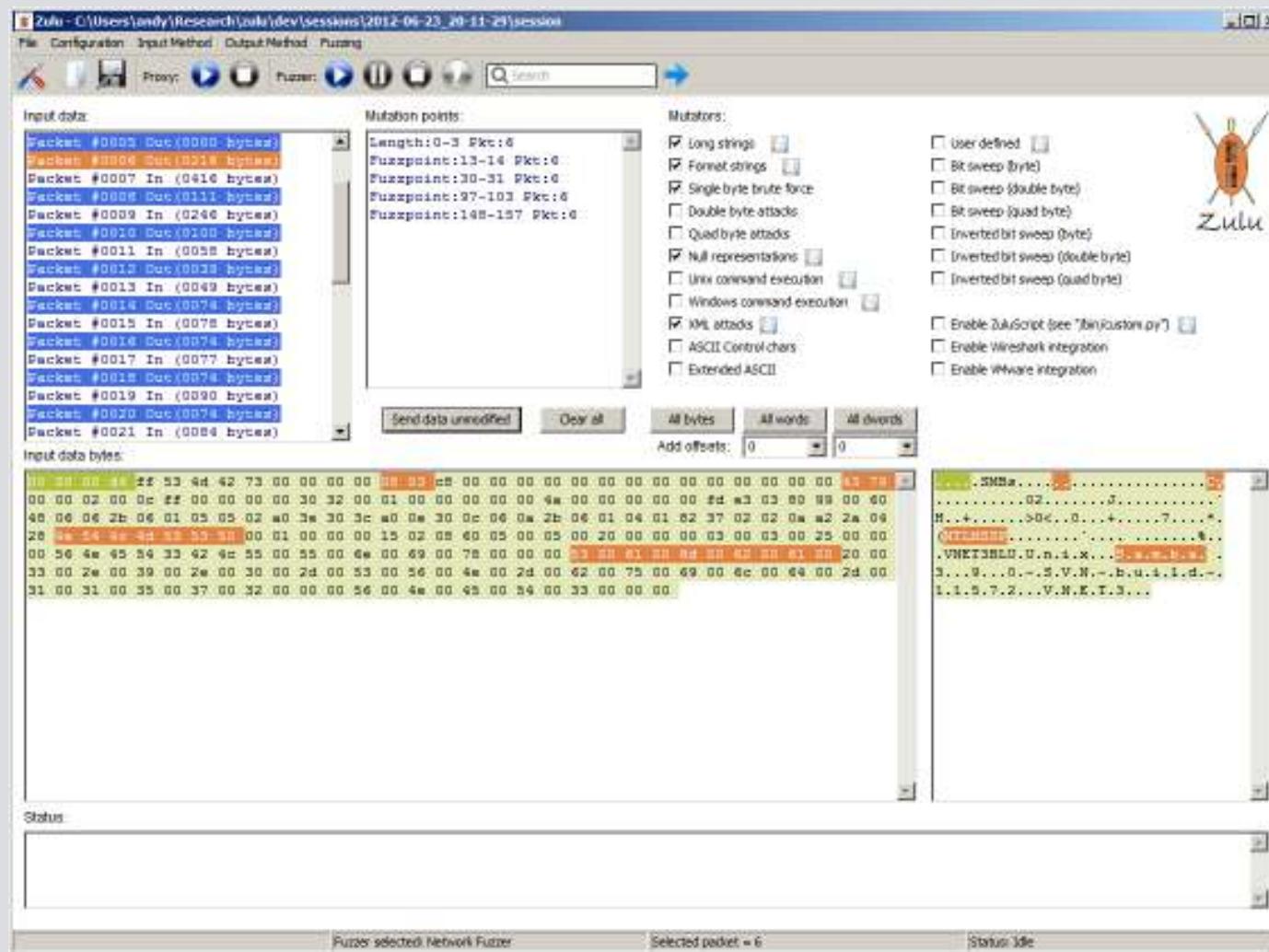
Motivations behind the tool

- I had lots of unique “fuzzer scripts”
- Fuzzing frameworks have a steep learning curve
- Fuzzers should be quick and easy to setup
- Wanted a point-and-click solution
- Needed to be scriptable to add complexity where required

Zulu basics – the GUI



Zulu basics – typical data



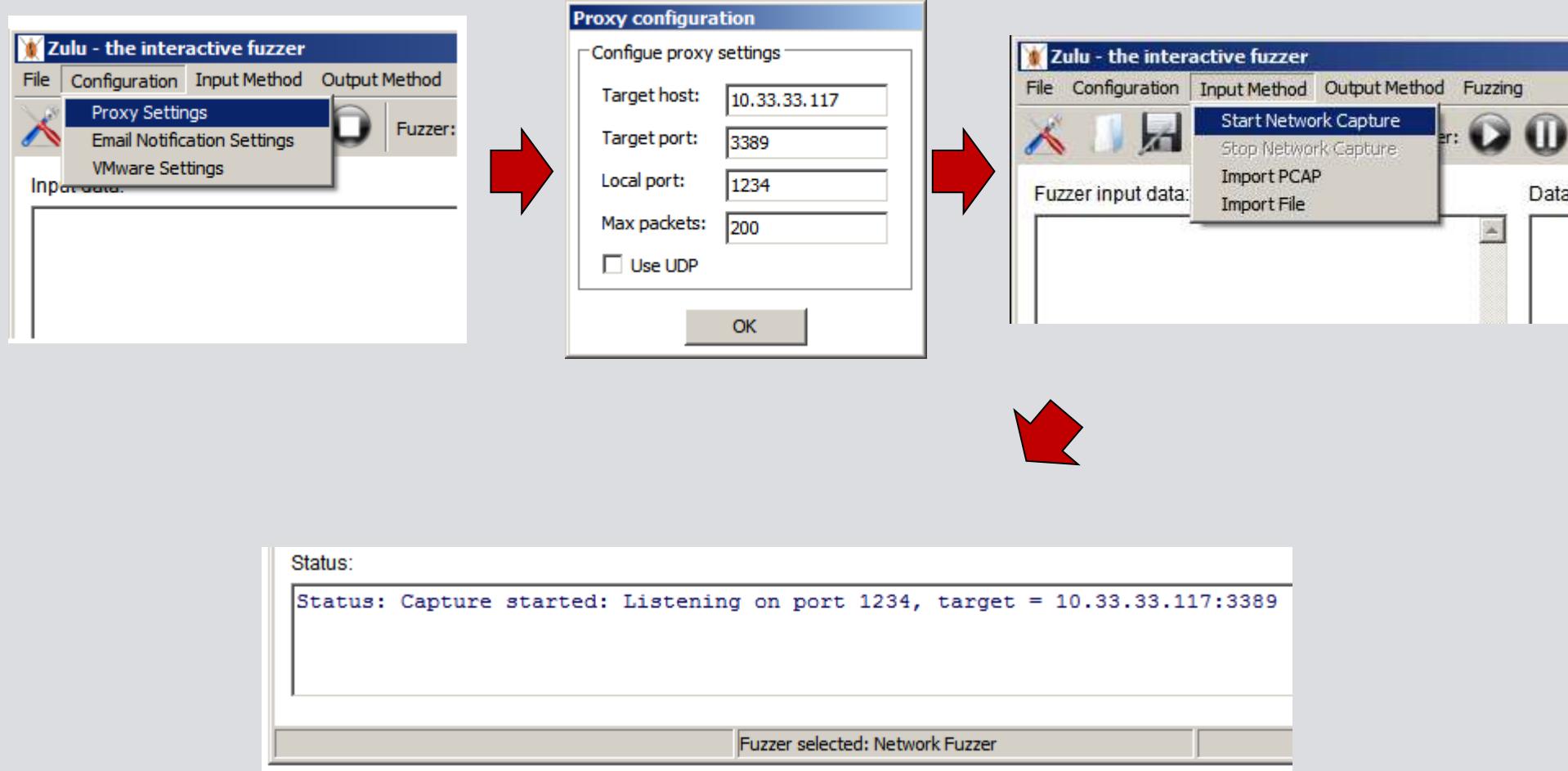
Zulu basics – the console

File structure

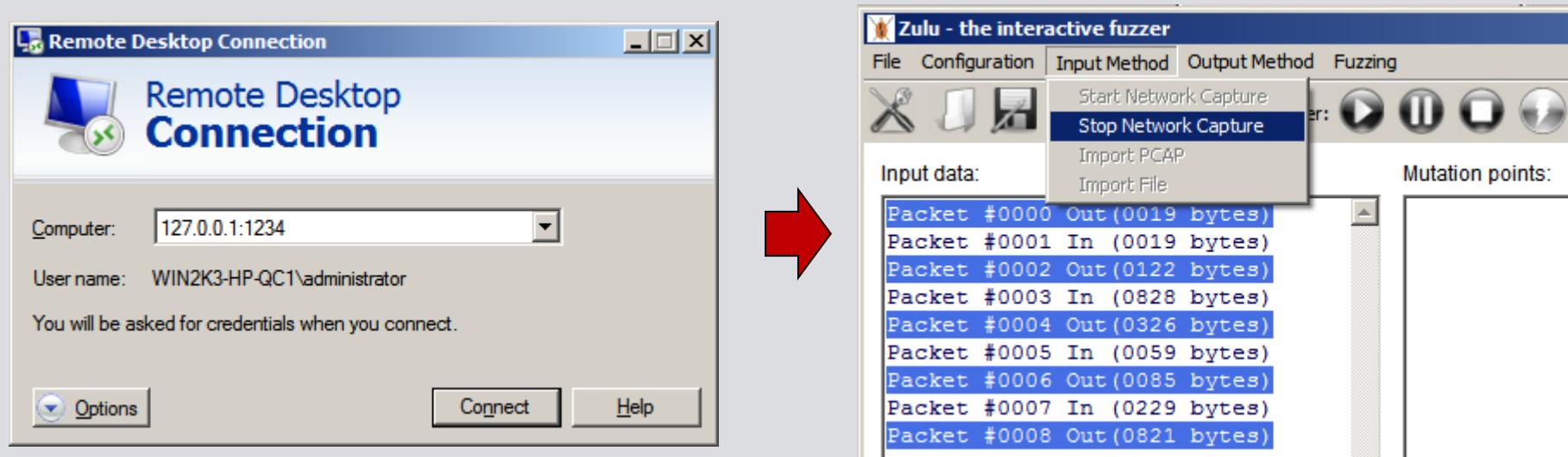
- **/bin** - Zulu binaries and custom.py (ZuluScript Python)
- **/crashfiles** - When file fuzzing, files that have caused the target to crash
- **/fuzzdb** - the fuzzer testcase files
- **/images** - images used by the GUI
- **/logs** - log files
- **/pcap** - when Wireshark integration is enabled, auto-generated PCAP files
- **/PoC** - when a crash occurs a PoC is auto-generated
- **/sessions** - configuration options and captured packets
- **/tempfiles** - when file fuzzing, temp manipulated files are stored here
- **/templates** - the template used to generate the PoC files is in here

Proxy-based network module

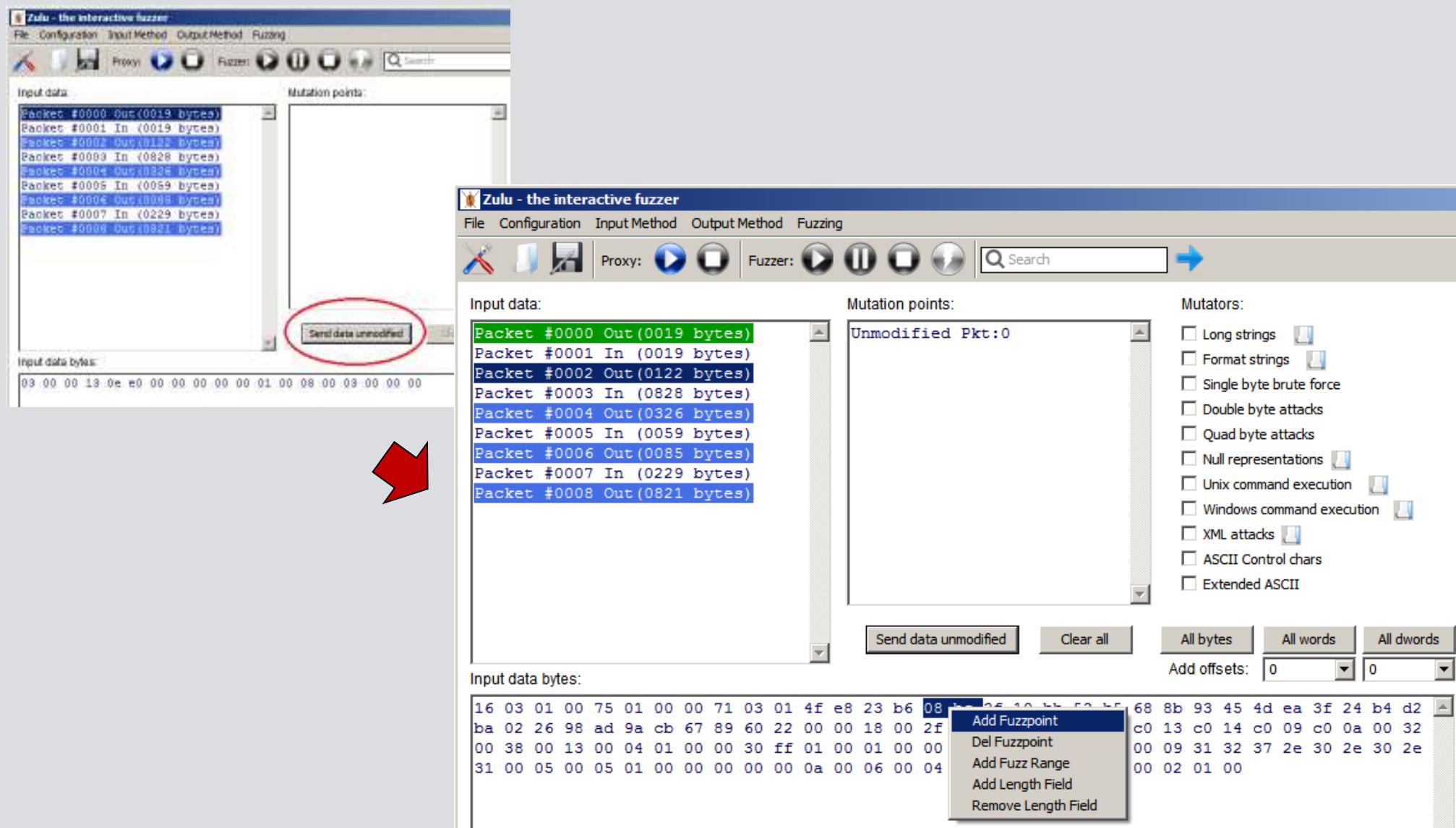
Configure the proxy



Use the standard network client



Select some fuzz points

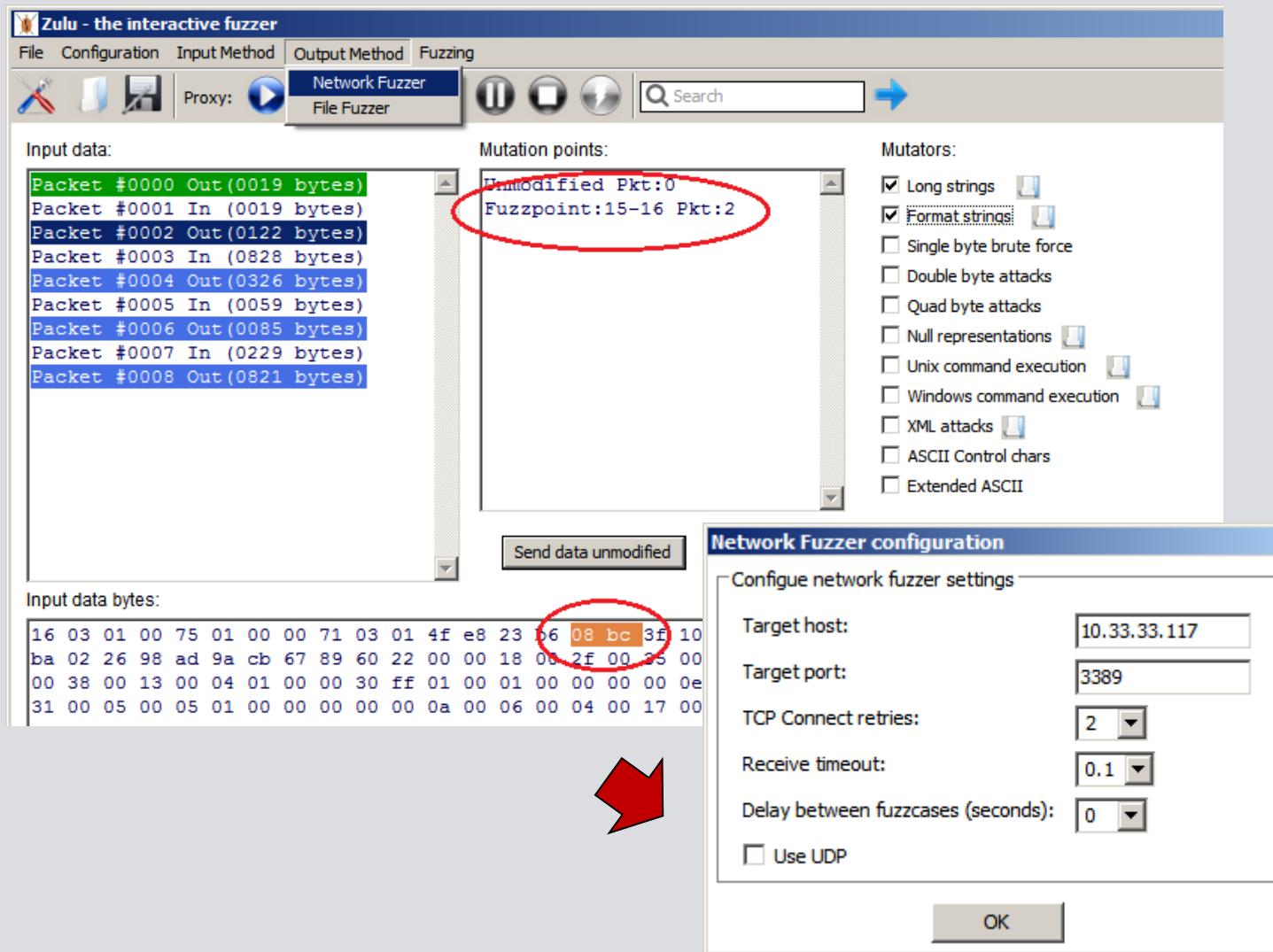


Select mutators

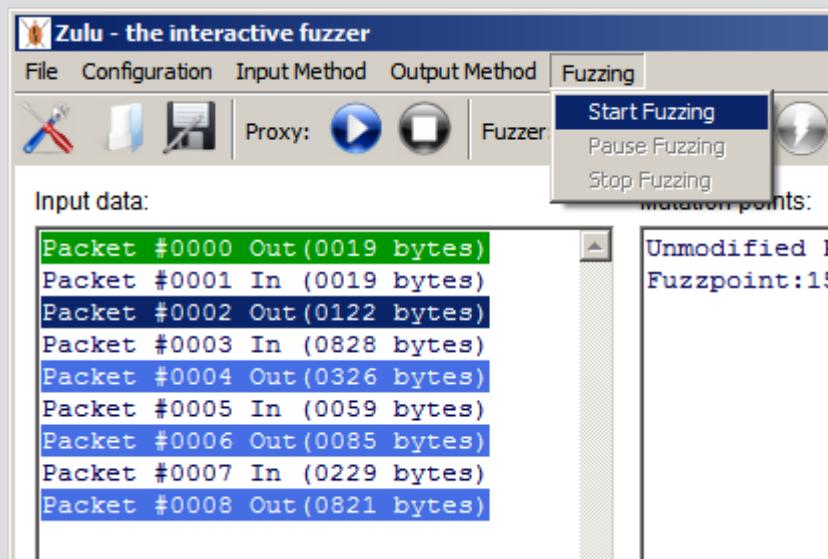
Mutators:

- | | |
|---|---|
| <input checked="" type="checkbox"/> Long strings  | <input type="checkbox"/> User defined  |
| <input checked="" type="checkbox"/> Format strings:  | <input type="checkbox"/> Bit sweep (byte) |
| <input type="checkbox"/> Single byte brute force | <input type="checkbox"/> Bit sweep (double byte) |
| <input type="checkbox"/> Double byte attacks | <input type="checkbox"/> Bit sweep (quad byte) |
| <input type="checkbox"/> Quad byte attacks | <input type="checkbox"/> Inverted bit sweep (byte) |
| <input type="checkbox"/> Null representations  | <input type="checkbox"/> Inverted bit sweep (double byte) |
| <input type="checkbox"/> Unix command execution  | <input type="checkbox"/> Inverted bit sweep (quad byte) |
| <input type="checkbox"/> Windows command execution  | |
| <input type="checkbox"/> XML attacks  | |
| <input type="checkbox"/> ASCII Control chars | |
| <input type="checkbox"/> Extended ASCII | |

Select output method



Start fuzzing



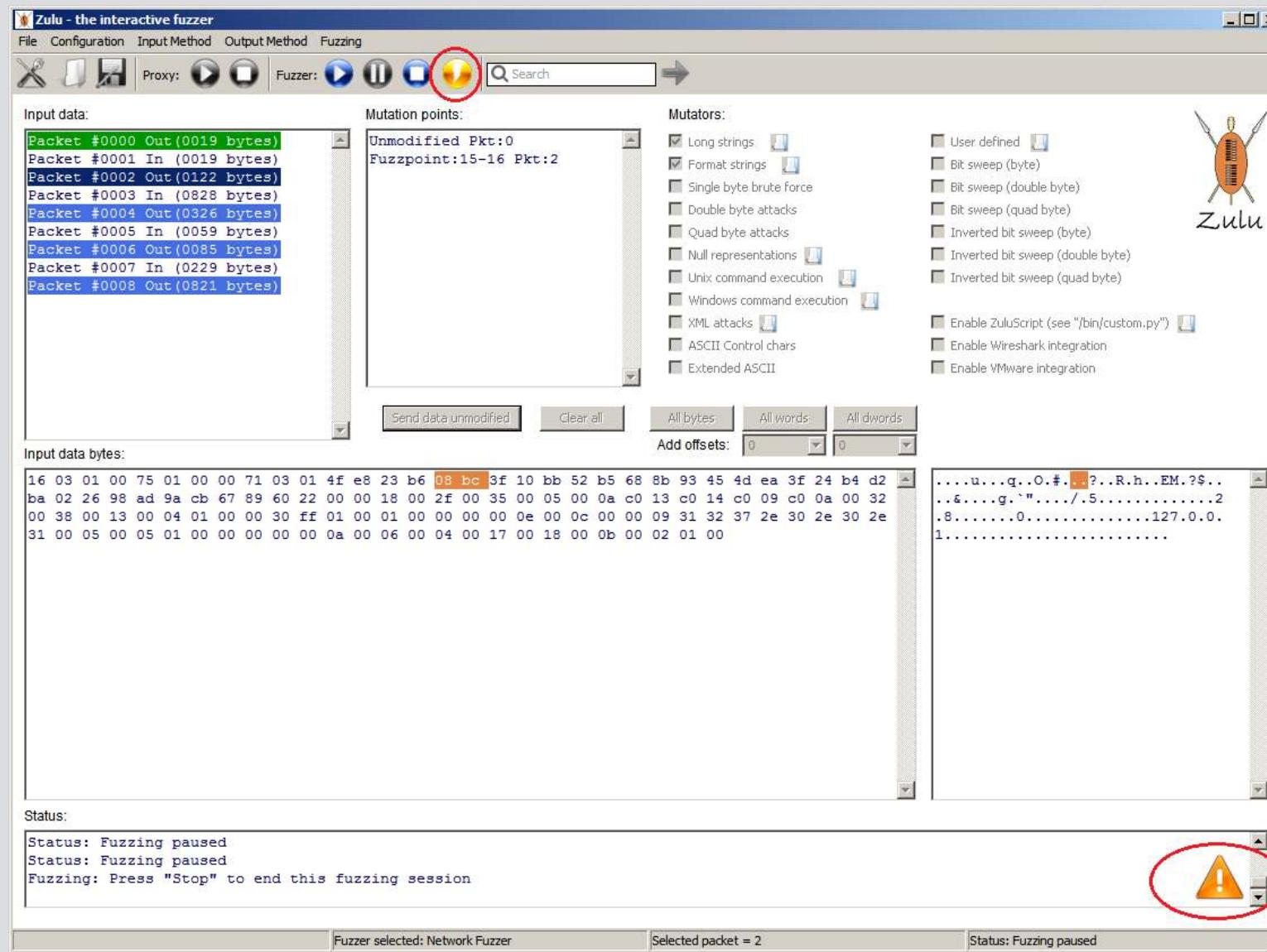
```
C:\Windows\system32\cmd.exe - zulu.py

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\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\x010\xe8#\xb6%.1025d?\x10\xbbR\xb5h\x8b\x93g\x89`"\x00\x00\x18\x00/\x005\x00\x05\x00\n\xc0\x13\xc0\x14\xc0\t\xc0\n\x002\x00\x01\x00\x00\x00\x0e\x00\x0c\x00\x00\t127.0.0.1\x00\x05\x00\x05\x01\x00\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
```

Instrumentation and triage



Other inputs: PCAP files

Wireshark captures

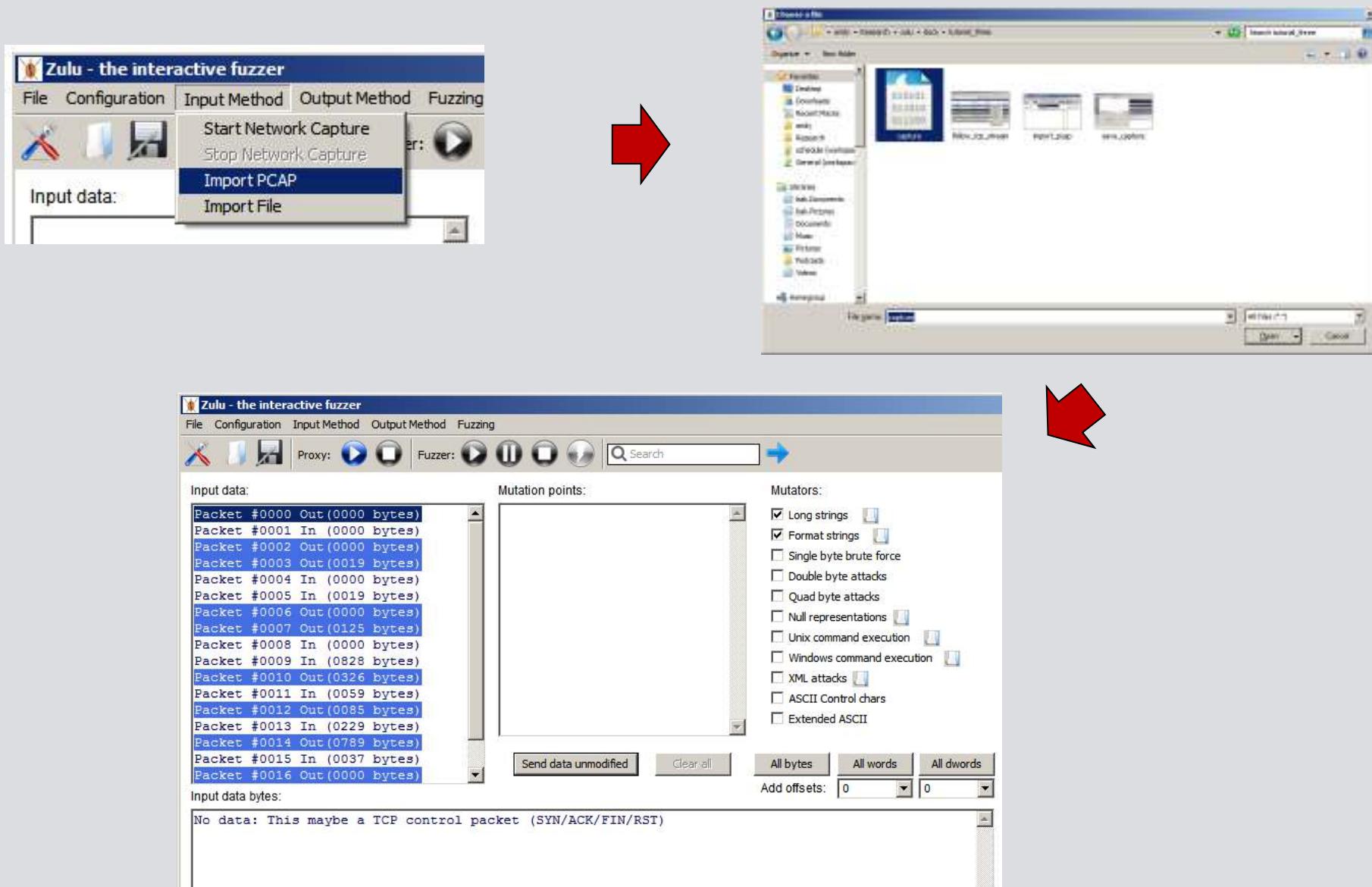
The screenshot shows a Wireshark interface with a list of network packets. A context menu is open over the 73rd packet, which is highlighted in blue. The menu options include:

- Mark Packet (toggle)
- Ignore Packet (toggle)
- Set Time Reference (toggle)
- Manually Resolve Address
- Apply as Filter
- Prepare a Filter
- Conversation Filter
- Colorize Conversation
- Follow TCP Stream (circled in red)
- Follow UDP Stream
- Follow SSL Stream
- Copy
- Decode As...
- Print...
- Show Packet in New Window

The packet details pane at the bottom shows the following information for the selected packet (Frame 73):

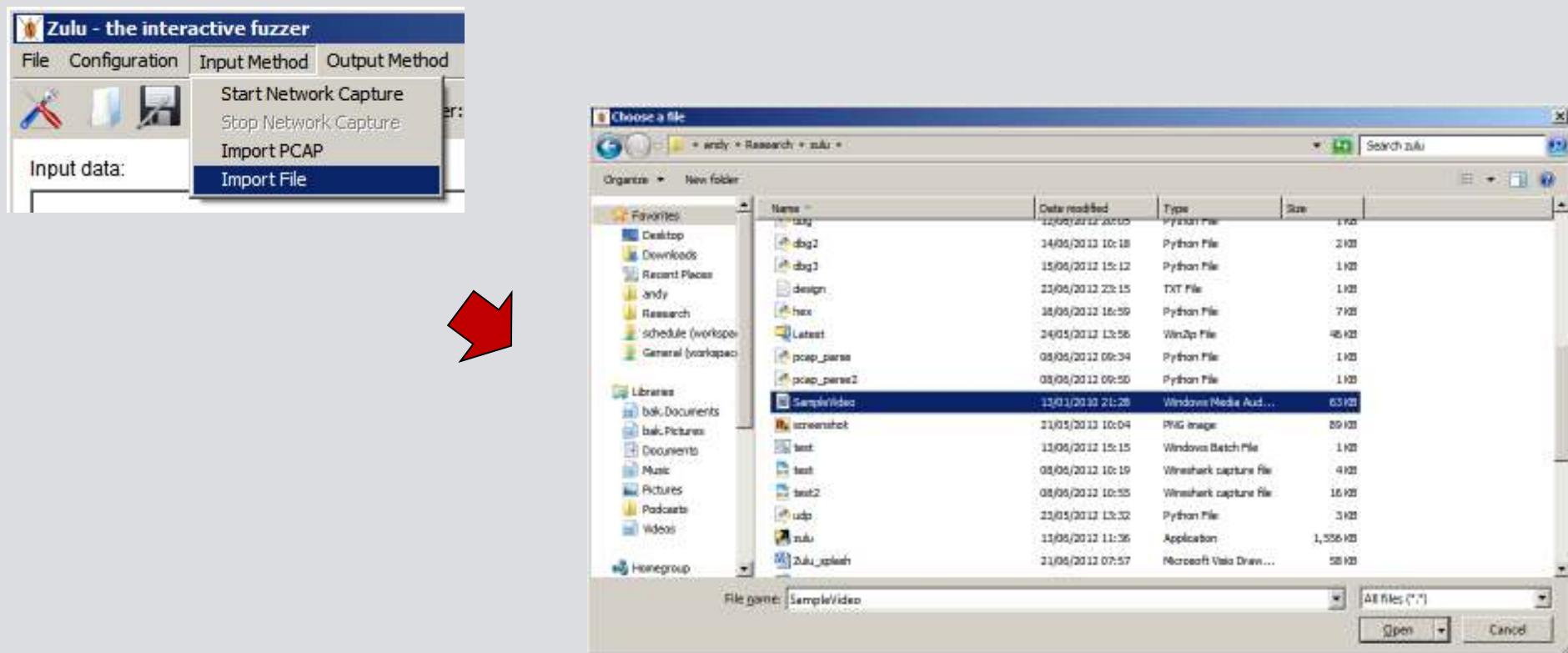
- Frame 73: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
- Ethernet II, Src: Dell_2a:2c:98 (5c:26:0a:2a:2c:98), Dst: QuantaCo_9a:a7:ae
- Internet Protocol version 4, Src: 10.33.33.104 (10.33.33.104), Dst: 10.33.33.117
- Transmission Control Protocol, Src Port: 45427 (45427), Dst Port: 3389 (3389)

Importing a PCAP

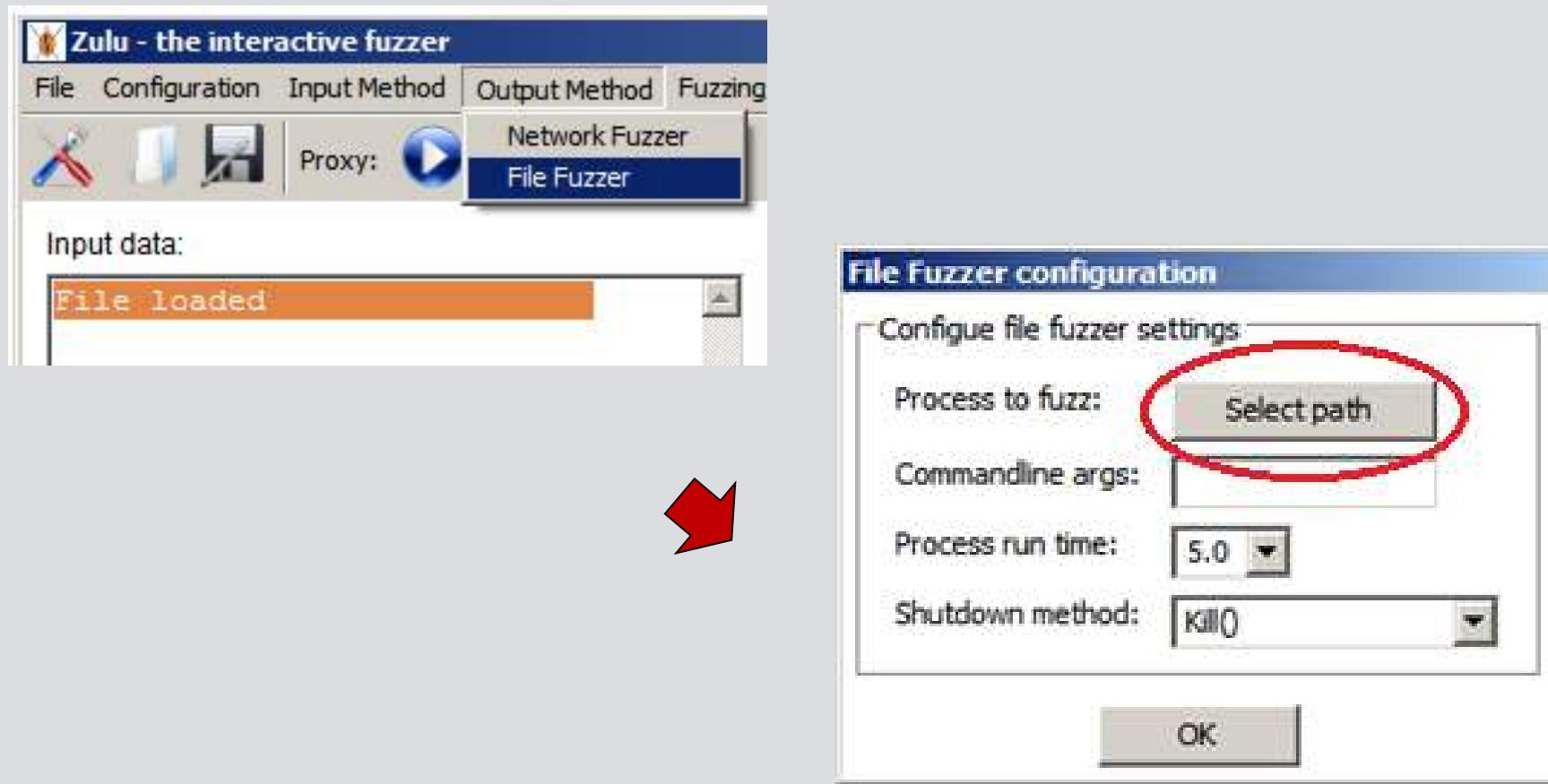


File module

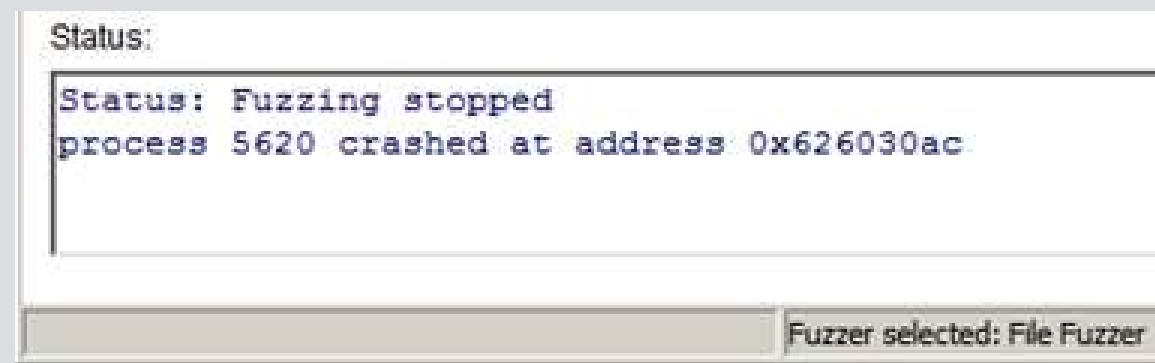
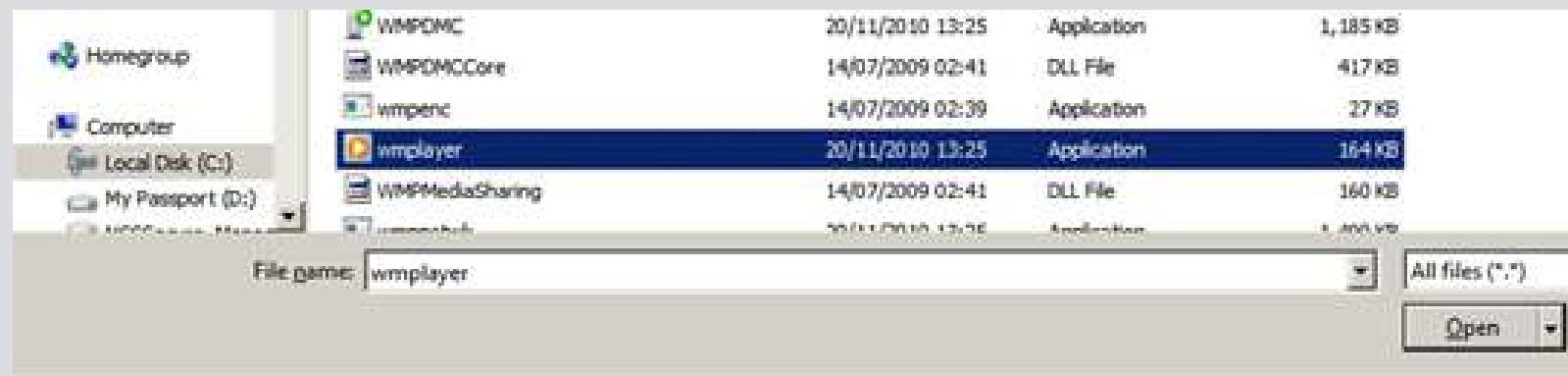
Select input file



Select file fuzzer + fuzz process



Fuzz process + debugging



USB module

Graphic USB

USB File Edit View Operations Window Help

Vbus: 5.060V 63uA

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

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

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

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

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

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

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

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

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

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

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

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

==== End of Capture ====

i 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.

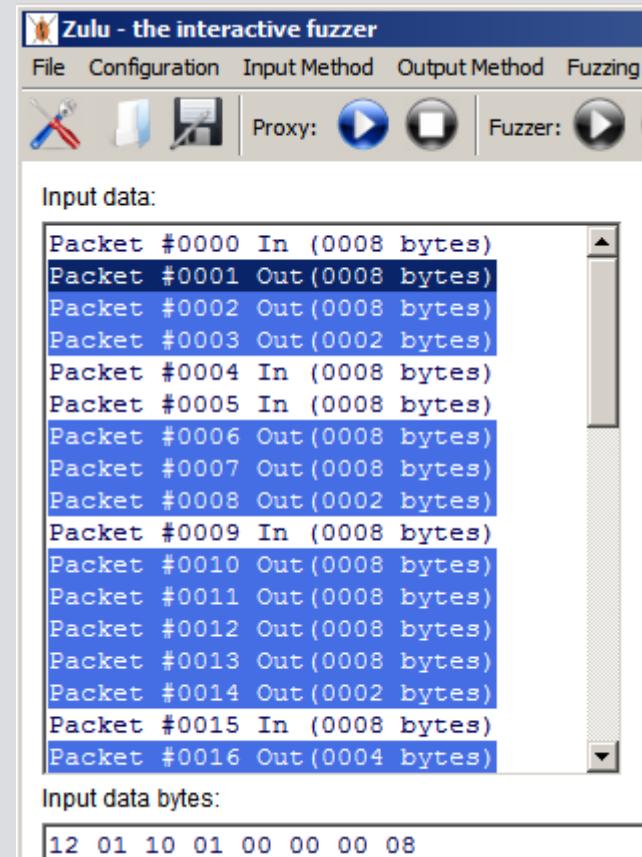
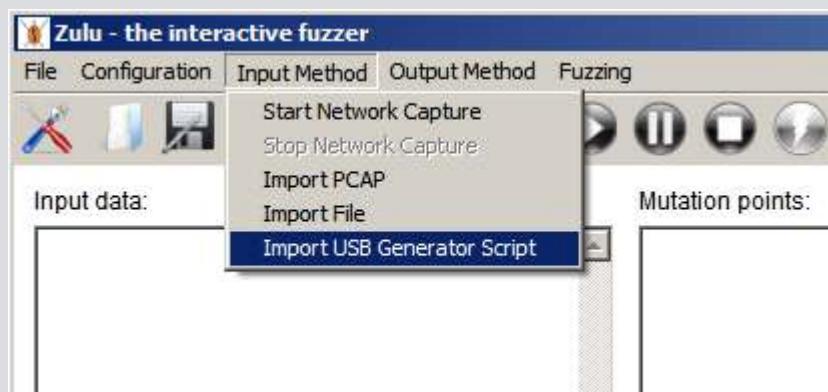
Field	Value	Meaning
bLength	18	Valid Length
bDescriptorType	1	DEVICE
bcdUSB	0x0110	Spec Version
bDeviceClass	0x00	Class Information in Interface Descriptor
bDeviceSubClass	0x00	Class Information in Interface Descriptor
bDeviceProtocol	0x00	Class Information in Interface Descriptor
bMaxPacketSize0	8	Max 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

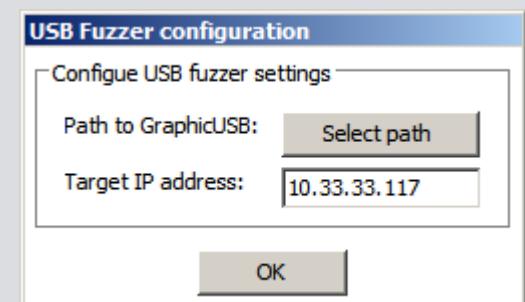
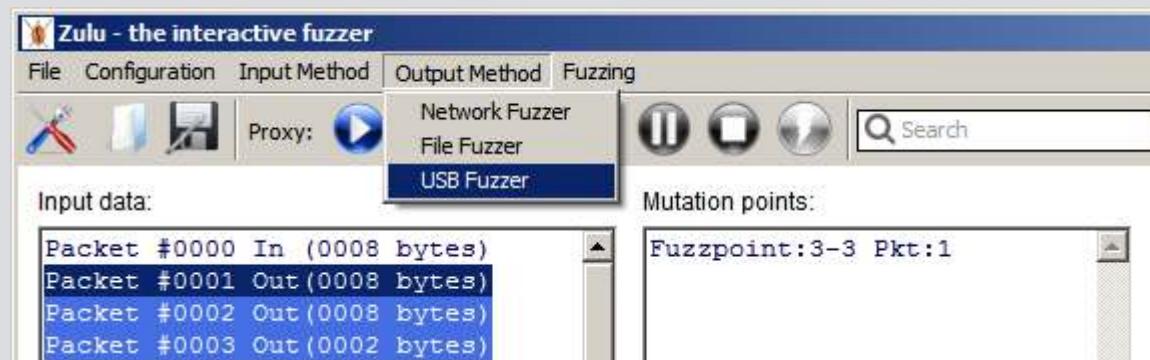
```
00000000: 12 01 10 01 00 00 00 08 3C 41 05 20 04 .....<A...
0000000D: 01 01 02 00 01 .....
```

For Help, press F1 186 events

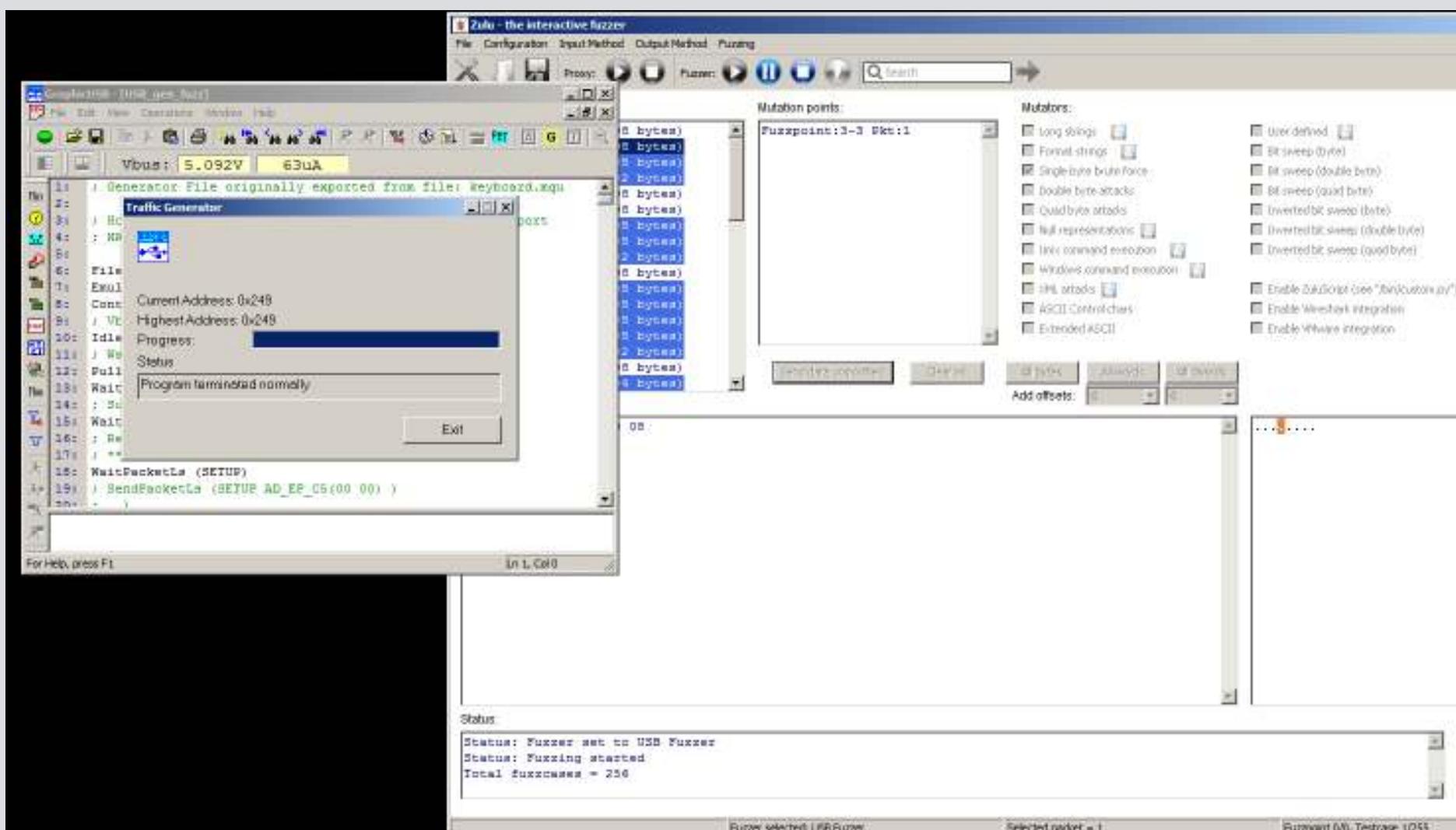
Import generator script



Select USB fuzzer

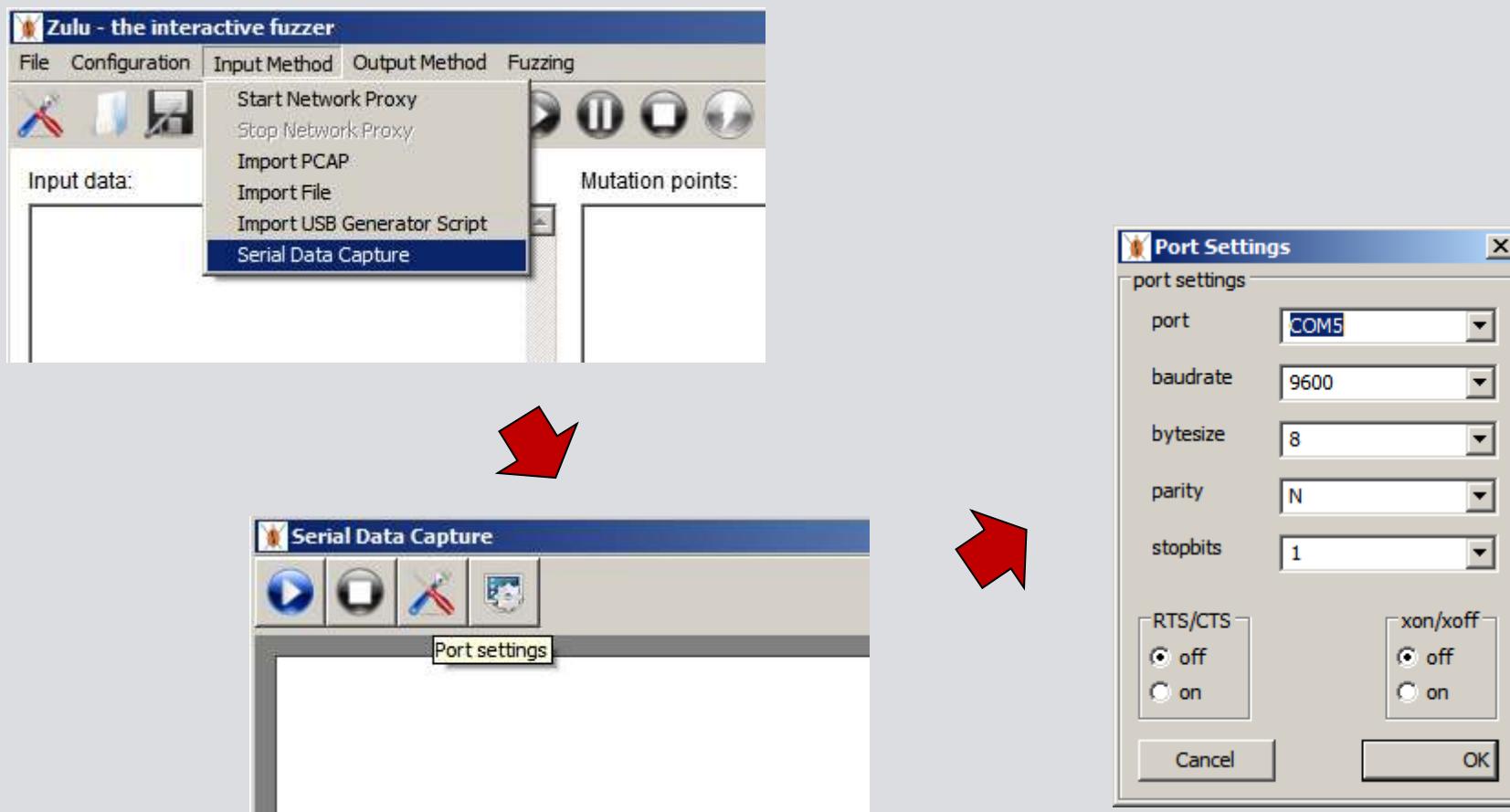


Fuzzer running

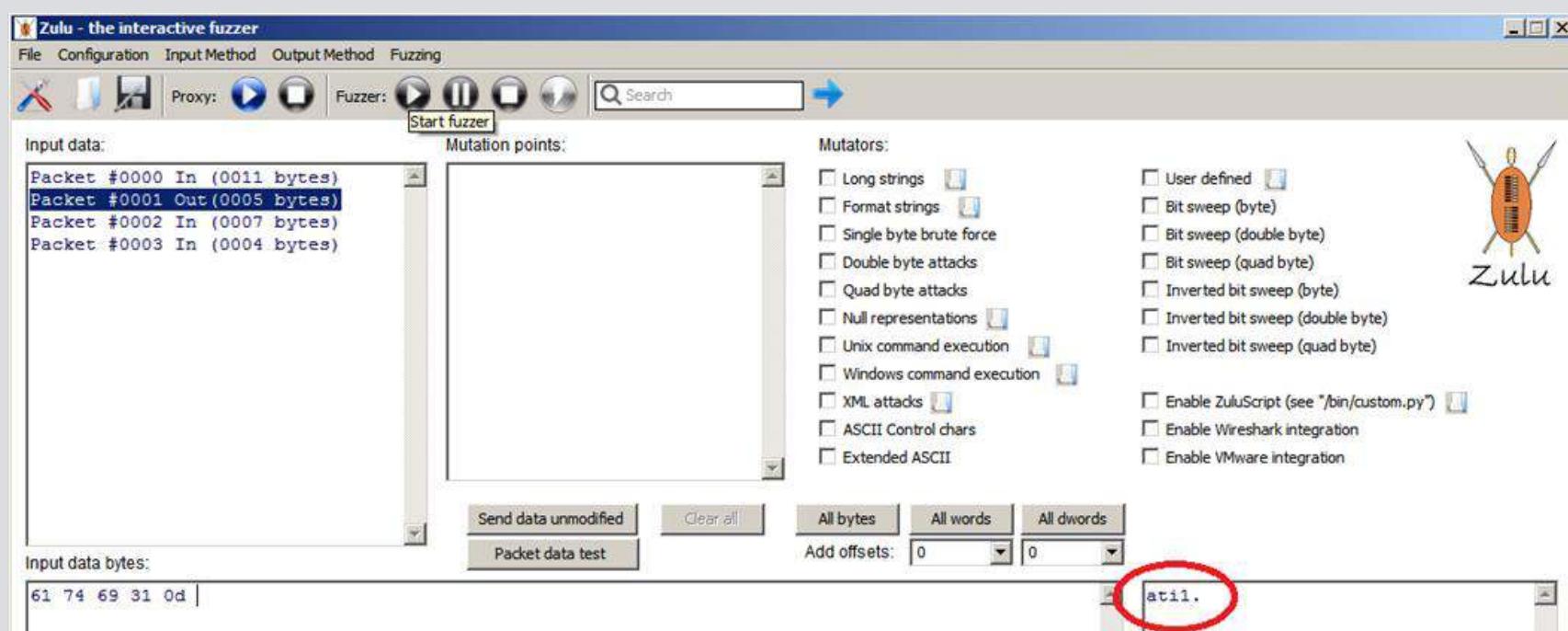


Serial module

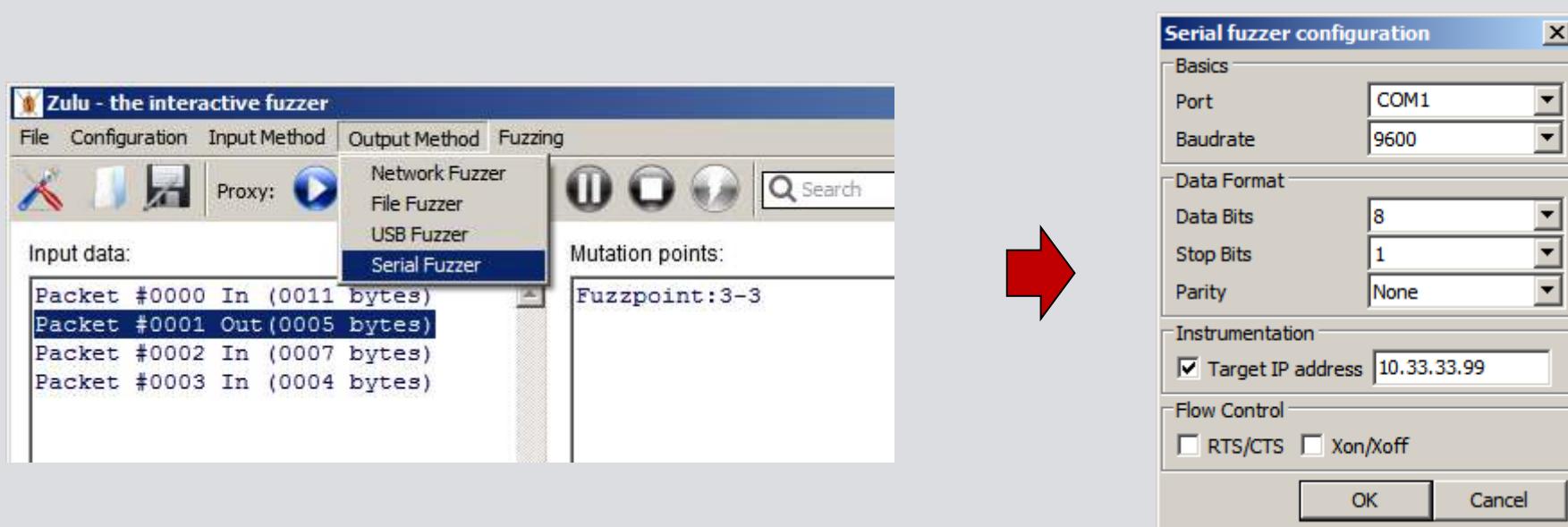
Serial settings



Serial data capture

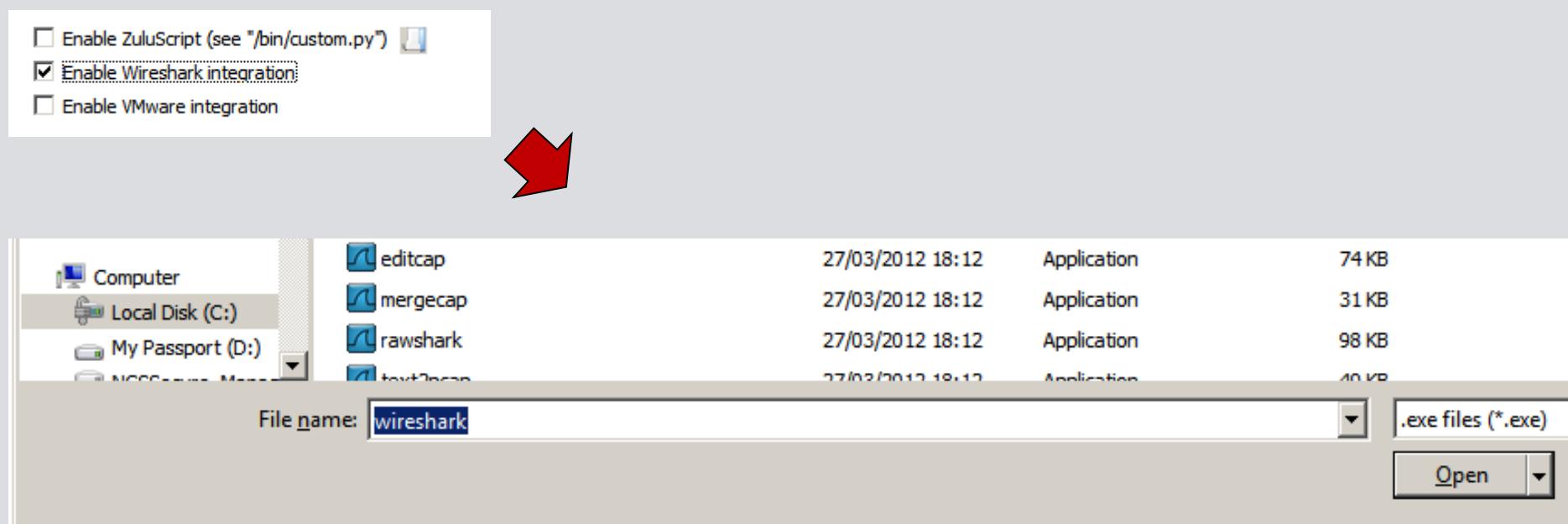


Serial fuzzing



Wireshark integration

Point to Wireshark binary



Auto-load Wireshark

The screenshot shows the Wireshark interface with a capture file named "zulu_pcap_2012-06-25_15-37-19.pcap". The packet list pane displays 10 captured frames, all of which are TPKT (X.224) packets. The details pane shows the structure of Frame 1, which is an ISO on TCP - RFC1006 (tpkt) packet. The selected packet is highlighted in red and shows the following details:

- 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

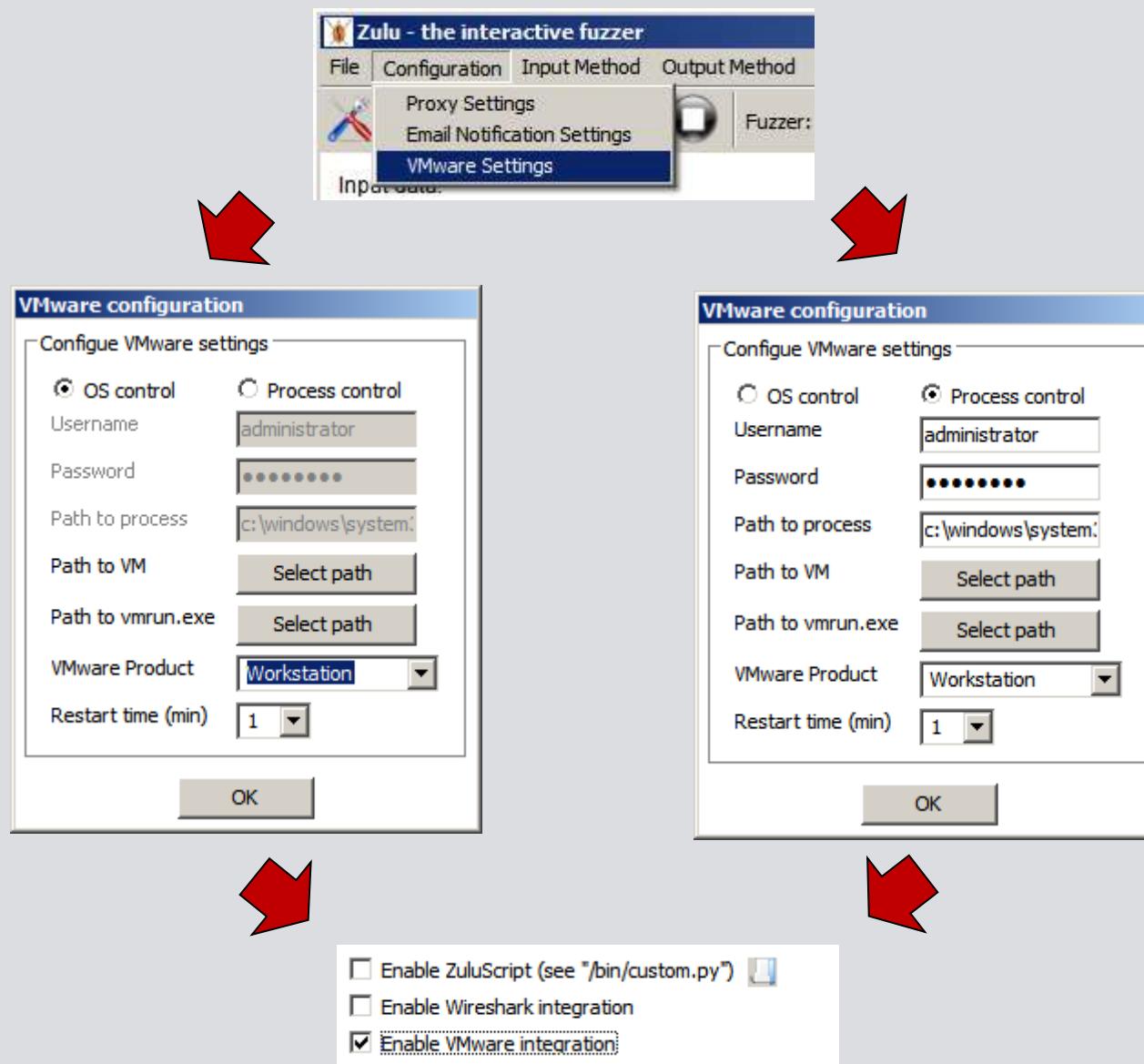
The selected packet is a TPKT, Version: 3, Length: 19. It includes fields for Version (3), Reserved (0), and Length (19). The ITU-T Rec X.224 section shows fields for Length (14), SRC-REF (0x0000), and Class (0x00). The RDP Routing Token is \001.

The bytes pane shows the raw hex and ASCII data for the selected packet. The ASCII dump shows characters like '.', '&', '*', 'E.', '!', '@', '=', '9F', and 'P.'.

At the bottom, the status bar indicates: TPKT - ISO on TCP - RFC1006 (tpkt), 4 bytes; Packets: 10 Displayed: 10 Marked: 0 Load time: 0:00.000; Profile: Default.

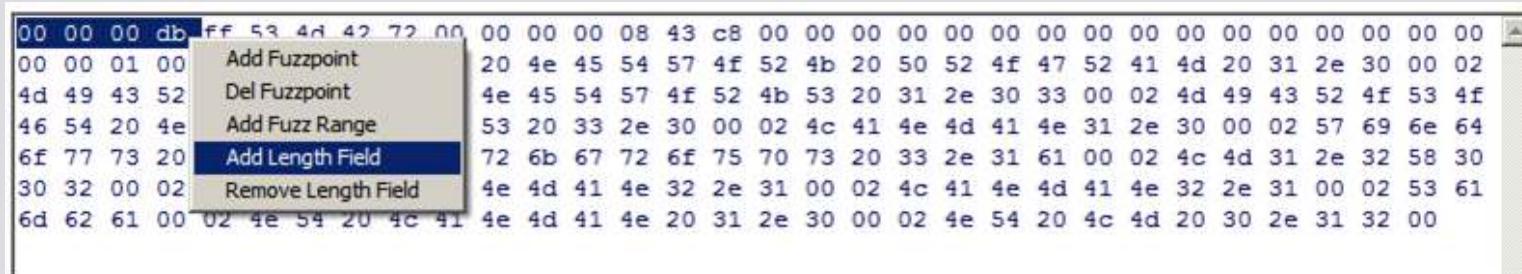
VMware integration

Select file fuzzer + fuzz process

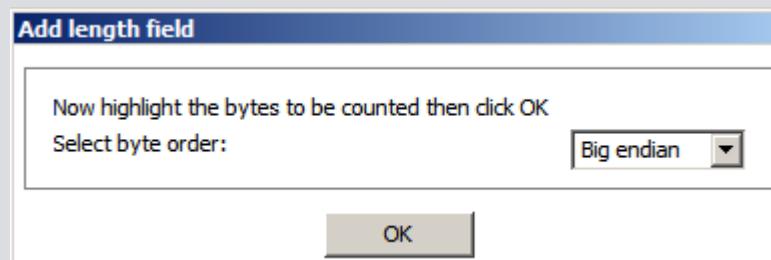
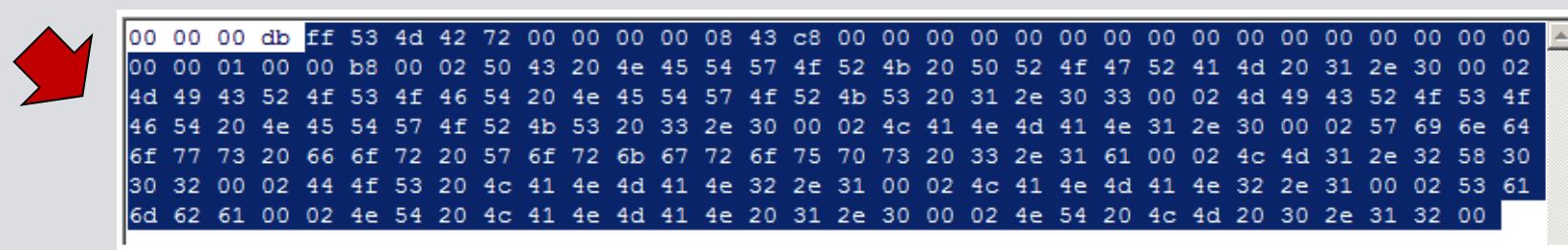


GUI-power

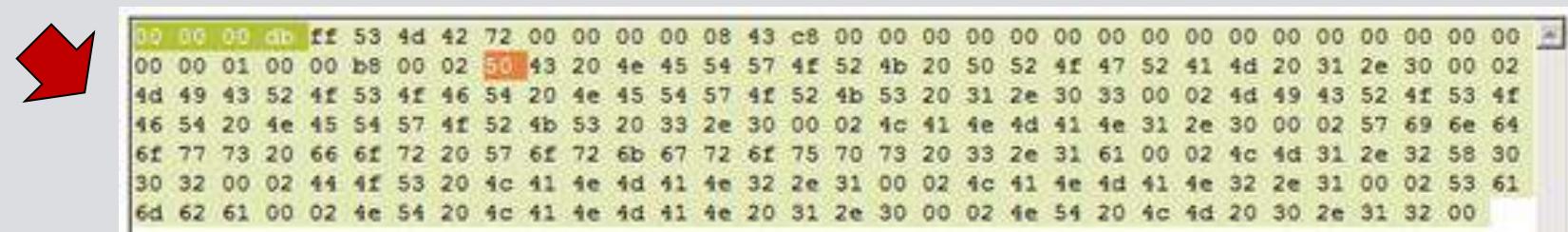
Adding a length field



00 00 00 db ff 53 4d 42 72 00 00 00 00 08 43 c8 00
00 00 01 00 Add Fuzzpoint
4d 49 43 52 Del Fuzzpoint
46 54 20 4e Add Fuzz Range
6f 77 73 20 Add Length Field
30 32 00 02 Remove Length Field
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

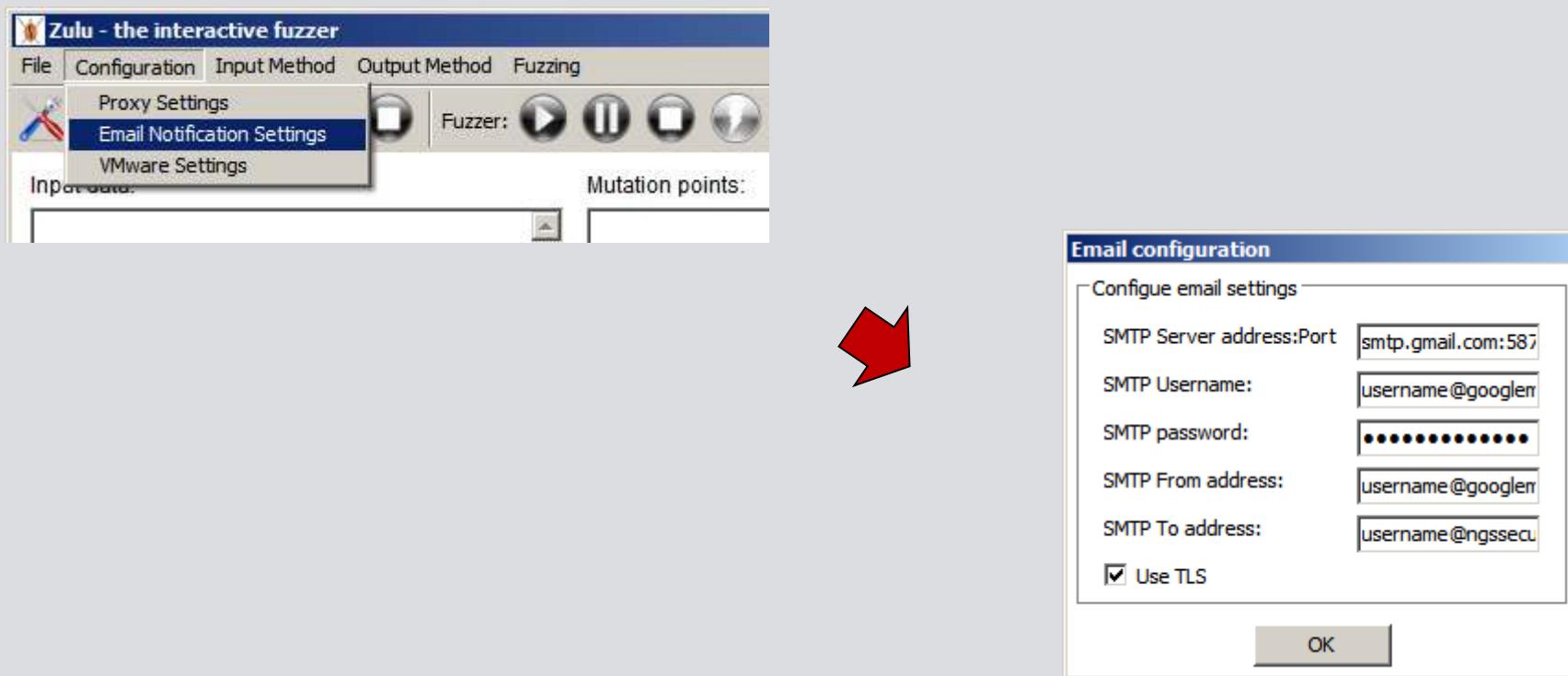
00 00 00 db ff 53 4d 42 72 00 00 00 00 00 08 43 c8 00
00 00 01 00 00 b8 00 02 50 43 20 4e 45 54 57 4f 52 4b 20 50 52 4f 47 52 41 4d 20 31 2e 30 00 02
4d 49 43 52 4f 53 4f 46 54 20 4e 45 54 57 4f 52 4b 53 20 31 2e 30 33 00 02 4d 49 43 52 4f 53 4f
46 54 20 4e 45 54 57 4f 52 4b 53 20 33 2e 30 00 02 4c 41 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



00 00 00 db ff 53 4d 42 72 00 00 00 00 08 43 c8 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

No need to watch! Email alerts

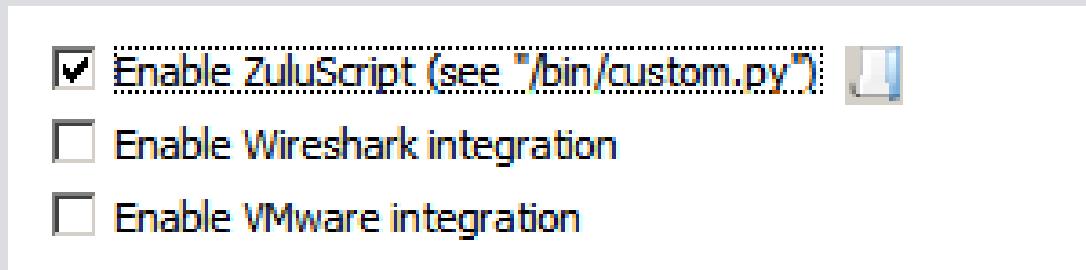
Select email settings



Advanced features - ZuluScript

Using ZuluScript

- How do you modify a packet after the mutator but before being processed by the target?
- The answer is by using ZuluScript
- Python script stored in a special file (/bin/custom.py)
- Includes a sample ***UpdateContentLengthField()*** function



Access to data

- ***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)

Bugs that Zulu has found

- Samba 'AndX' request remote heap overflow (CVE-2012-0870)
- Oracle 11g TNS listener remote null pointer dereference
- Apple OS X USB Hub Descriptor bNbrPorts Field Handling Memory Corruption
- ...and many others that haven't been fixed yet

Zulu is available on Github

Zulu can be downloaded today at:

<https://github.com/nccgroup/zulu>



Questions?

Andy Davis, Research Director NCC Group
andy.davis 'at' nccgroup 'dot' com

