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

- 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)
Select output method
Start fuzzing
Instrumentation and triage
Other inputs: PCAP files
Wireshark captures

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>10.33.33.104</td>
<td>88.151.219.102</td>
<td>TCP</td>
<td>44367 &gt; 443</td>
</tr>
<tr>
<td>68</td>
<td>88.151.219.102</td>
<td>10.33.33.104</td>
<td>SSL</td>
<td>Continuation</td>
</tr>
<tr>
<td>69</td>
<td>DevoLo_d4:e37:e0</td>
<td>Broadcast</td>
<td>HomePl</td>
<td>Vendor Speci</td>
</tr>
<tr>
<td>70</td>
<td>DevoLo_d4:e37:e0</td>
<td>Broadcast</td>
<td>HomePl</td>
<td>Network Stat</td>
</tr>
<tr>
<td>71</td>
<td>Dell_2a:2c:98</td>
<td>Broadcast</td>
<td>ARP</td>
<td>Who has 10.3</td>
</tr>
<tr>
<td>72</td>
<td>QuantaCo_9a:a7:ae</td>
<td>Dell_2a:2c:98</td>
<td>ARP</td>
<td>10.33.33.117</td>
</tr>
<tr>
<td>73</td>
<td>10.33.33.104</td>
<td>10.33.33.117</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>74</td>
<td>QuantaCo_9a:a7:ae</td>
<td>Broadcast</td>
<td>10.33.33.104</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>75</td>
<td>Dell_2a:2c:98</td>
<td>Broadcast</td>
<td>10.33.33.104</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>76</td>
<td>Dell_2a:2c:98</td>
<td>Broadcast</td>
<td>10.33.33.104</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>77</td>
<td>10.33.33.104</td>
<td>10.33.33.117</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>78</td>
<td>10.33.33.104</td>
<td>10.33.33.117</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>79</td>
<td>10.33.33.104</td>
<td>10.33.33.117</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>80</td>
<td>10.33.33.104</td>
<td>10.33.33.117</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>81</td>
<td>10.33.33.338</td>
<td>88.151.219.102</td>
<td>UDP</td>
<td>3389 &gt; 45427</td>
</tr>
<tr>
<td>82</td>
<td>fe80::9504:dle5:7e31:dc6d</td>
<td>ff02::c</td>
<td>UDP</td>
<td>3389 &gt; 45427</td>
</tr>
<tr>
<td>83</td>
<td>88.151.219.102</td>
<td>10.33.33.104</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
<tr>
<td>84</td>
<td>88.151.219.102</td>
<td>10.33.33.104</td>
<td>TCP</td>
<td>445427 &gt; 3389</td>
</tr>
</tbody>
</table>

Frame 73: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
Ethernet II, Src: Dell_2a:2c:98 (5c:26:a2:2c: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)

Follow TCP Stream
- Mark Packet (toggle)
- Ignore Packet (toggle)
- Set Time Reference (toggle)
- Manually Resolve Address
- Apply as Filter
- Prepare a Filter
- Conversation Filter
- Colorize Conversation
- SSH
- Copy

Decode As...
Print...
Show Packet in New Window
Importing a PCAP
File module
Select input file
Select file fuzzer + fuzz process
Fuzz process + debugging

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

Fuzzer selected: File Fuzzer
USB module
Graphic USB

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>16</td>
<td>Valid Length</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>1</td>
<td>DEVICE</td>
</tr>
<tr>
<td>bcdUSB</td>
<td>0x110</td>
<td>Spec Version</td>
</tr>
<tr>
<td>bDeviceClass</td>
<td>0x00</td>
<td>Class Information in Interface Descriptor</td>
</tr>
<tr>
<td>bDeviceSubClass</td>
<td>0x00</td>
<td>Class Information in Interface Descriptor</td>
</tr>
<tr>
<td>bDeviceProtocol</td>
<td>0x00</td>
<td>Class Information in Interface Descriptor</td>
</tr>
<tr>
<td>bMaxPacketSize</td>
<td>8</td>
<td>Max EP0 Packet Size</td>
</tr>
<tr>
<td>idVendor</td>
<td>0x112C</td>
<td>Dell Inc.</td>
</tr>
<tr>
<td>idProduct</td>
<td>0x2005</td>
<td>Unknown</td>
</tr>
<tr>
<td>bcdDevice</td>
<td>0x0104</td>
<td>Device Release No</td>
</tr>
<tr>
<td>iManufacturer</td>
<td>1</td>
<td>Index to Manufacturer String</td>
</tr>
<tr>
<td>iProduct</td>
<td>2</td>
<td>Index to Product String</td>
</tr>
<tr>
<td>iSerialNumber</td>
<td>0</td>
<td>Index to Serial Number</td>
</tr>
</tbody>
</table>

Data Content

```
00000000: 12 01 10 01 00 00 00 00 08 3C 41 05 20 04 ......<CA ...>
00000000: 01 01 02 00 01 ...
```

--- End of Capture ---
Import generator script
Select USB fuzzer
Fuzzer running
Serial module
Serial settings

- Zulu - the interactive fuzzer
- Port Settings
  - port is COM5
  - baudrate is 9600
  - bytesize is 8
  - parity is N
  - stopbits is 1
  - RTS/CTS: off
  - xon/xoff: on
Serial data capture

- Connect to serial port
- *EMRDY: 1
- at1
- R1K06
- OK

Zulu - the interactive fuzzer

Input data:
Packet #0000 In (0011 bytes)
Packet #0001 Out (0003 bytes)
Packet #0002 In (0007 bytes)
Packet #0003 In (0004 bytes)

Mutations:

Mutators:
- Long strings
- Format strings
- Single byte brute force
- Double byte attacks
- Quad byte attacks
- Null representations
- Linux 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 words
All words
Add offsets:
0 0 0

atil.
Serial fuzzing
Wireshark integration
Point to Wireshark binary
Auto-load Wireshark
VMware integration
Select file fuzzer + fuzz process
GUI-power
Adding a length field
No need to watch! Email alerts
Select email settings

Zulu - the interactive fuzzer

Email Notification Settings

VMware Settings

Configure email settings:

- SMTP Server address: smtp.gmail.com:587
- SMTP Username: username@gmail.com
- SMTP password: ********
- SMTP From address: username@gmail.com
- SMTP To address: username@ngssecu
- Use TLS

OK
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