Lowering the USB Fuzzing Barrier by Transparent Two-Way Emulation

Rijnard van Tonder
Herman Engelbrecht

Stellenbosch University
Motivation

➢ High-impact security bugs reside in the USB attack surface

➢ Challenging to explore due to
  ○ limited pure software solutions
  ○ hardware acquisition
  ○ inflexible hardware for security testing
  ○ knowledge requirement of USB

➢ Can we do better?
Existing Solutions

Software:
➢ Qemu emulation (MWR Labs, ‘11)
➢ Frisbee Lite (Davis, ‘12)

Hardware:
➢ USB Analyzer, Frisbee and GraphicUSB (Davis, ‘11)
➢ Arduino (Ose, ‘11, Davis, ‘11)
➢ Facedancer (Goodspeed, Bratus, ‘12), umap (Davis, ‘13)
➢ BeagleBone and USBProxy (Spill ‘14)
<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>✓</td>
<td>Fast</td>
<td>✗</td>
</tr>
<tr>
<td>Read and Write ability</td>
<td>✓</td>
<td>Read and Write ability</td>
<td>✓</td>
</tr>
<tr>
<td>Man-in-the-middle</td>
<td>✓</td>
<td>Man-in-the-middle</td>
<td>✗</td>
</tr>
<tr>
<td>Knowledge requirement</td>
<td>✓</td>
<td>Knowledge requirement</td>
<td>✗</td>
</tr>
<tr>
<td>Cost</td>
<td>✗</td>
<td>Cost</td>
<td>✓</td>
</tr>
<tr>
<td>Flexible</td>
<td>✗</td>
<td>Flexible</td>
<td>✓</td>
</tr>
</tbody>
</table>
Contributions

➢ The TTWE USB fuzzing framework that
  ○ Is flexible,
  ○ Is cost-effective, and
  ○ Lowers the knowledge requirement

➢ Initial results and analysis of bug-hunting with TTWE

➢ New possibilities for USB fuzzing and attacks
USB Protocol Primer

➢ Consists of requests and descriptors exchanged between host and peripheral
➢ USB defines device classes for peripherals
➢ Endpoints designate data direction and address
➢ Control transfers and Non-control transfers
➢ Packets
  o Token
  o Data
  o Handshake
TTWE

➢ Tap into the communication between host and peripheral
➢ Modify communication
➢ The Facedancer can emulate host or peripheral devices
➢ Emulate both simultaneously
Design

Peripheral Emulation

Mediating Computer

Peripheral Emulation

Host Emulation

Host

Peripheral

EP2OUT

EP1IN

EP0

EP1OUT

EP0

EP3IN
Hardware Implementation

![Diagram showing USB controller, 16-bit Microcontroller, and USB/Serial adapter](image)

- USB controller
- 16-bit Microcontroller
- USB/Serial adapter

Target → 16-bit Microcontroller → USB/Serial adapter → MC
Software Implementation

➢ Emulation drivers
  ○ Host and Peripheral mode
  ○ Communicate via named pipes

Two challenges:
➢ Endpoint Hijacking
➢ Handshake emulation
Endpoint Hijacking

➢ Problem: hardcoded endpoint descriptors

EP1: IN  ➔  EP3: IN
EP2: OUT  ➔  EP1: OUT
Design

Peripheral | Host Emulation | Mediating Computer | Peripheral Emulation | Host

- EP0 (in)
- EP1IN (in)
- EP1OUT (out)
- EP2OUT (out)
- EP3IN (in)
Transparent Emulation Results

➢ Mass storage device
  ○ Enumeration
  ○ SCSI data
  ○ Mount, read, and write ability
Fuzzing Results

➢ “Dumb” fuzzing setup
➢ Printer Driver bug
  ○ Memory corruption
➢ Application DoS on print
  ○ Waits for ACK
➢ WiFi dongle
  ○ Invalid response to clear_feature
➢ Mass storage driver bug in printer
  ○ Malformed SCSI response
Limitations

➢ Slow
➢ Device timeouts
➢ Number of endpoints
Conclusion

➢ Flexible and inexpensive way to explore the USB attack surface
➢ Record and replay when fuzzing

Further avenues:
➢ TOCTTOU RIT attack (Mulliner, ‘12)
➢ Devices-as-seed-files
Questions

@rvtond

https://github.com/rvantonder/ttwe-proto

rvantonder@ml.sun.ac.za