Universal Serial aBUSE

Rogan Dawes & Dominic White
research@sensepost.com
Important Note

• This is the text-based version of the slides, and not the version we plan to present. We wanted to give you something meaningful to read through.

• The toolset will be released on our github:
  • [https://github.com/sensepost/USaBUSE](https://github.com/sensepost/USaBUSE)

• Details of the talk, latest slides and code will be written up at our blog:
  • [https://sensepost.com/blog](https://sensepost.com/blog)
Overview

In this talk, we’ll cover some novel USB-level attacks, that can provide remote command and control of, even air-gapped machines, with a minimal forensic footprint, and release an open-source toolset using freely available hardware.
The Meta Point

• It’s hard to defend at the best of times.
• Doing so well, requires a realistic threat model.
• Too often, that threat model is driven by vendor marketing rather than real attacks.
  • For example, advanced attackers have always existed, it’s not clear “APT” would have been a thing, or as much of a thing without the significant vendor marketing spend put behind it.
• Penetration testers need to emulate real threats or they’re just wasting your time
Why we’re highlighting this issue

• We’ve seen real attackers doing it, but defenses haven’t adapted.
  • The NSA’s COTTONMOUTH toolkit showed these sorts of USB attacks
  • Technically unsophisticated criminals have defrauded banks using simple IP KVMs
• If your apex predators and low level bottom feeders are using the same sort of attacks; physical bypasses of software/network security via hardware, then you best pay attention.
• Plus, it makes sense, software is getting harder to exploit and changes more rapidly than hardware.
But we know about these attacks?

• Do we? Because the defenses in this space seem to be poor in our experience at clients.

• Hardware keyloggers have been around for decades, and are still near impossible to practically detect in software.

• Most organisations seem to think USB is about malware or tethering/wifi and rely on protections elsewhere in the stack:
  • malware deployment – proxy
  • malware on device – AV/endpoint
  • comms from device – FireEye and friends

• But there’s little defense specific to malicious devices, something the USB standard makes very easy to implement.

• Finally, there wasn’t an end-to-end implementation of this attack when we started.
Prior Work

• This work stands on the shoulders of giants. While numerous researchers have produced USB related work, prior work specific to this project includes:
  • Travis Goodspeed’s Facedancer2 [http://goodfet.sourceforge.net/hardware/facedancer21/]
  • Michael Ossman & Dominic Spill’s NSA Playset, TURNIPSCHOOL [http://www.nsaplayset.org/turnipschool]
  • Samy Kamkar’s USBDriveBy [http://samy.pl/usbdriveby/]
  • USB Rubber Ducky Wiki [http://usbrubberducky.com/]
  • Seunghun Han’s Iron-HID [https://github.com/kkamagui/IRON-HID] (released after our Defcon CFP submission)
Objectives of our Work

• Build and end-to-end attack that’s usable in a pentest
• Allow it to be remotely triggered and updated
• Work without requiring victim interaction
• Exclude typical USB malware vectors (for which typical defenses exist) e.g. malware via mass storage
• Don’t sent any traffic via the victim’s network
  • Avoids environmental complexities (firewalls, etc)
  • Avoids detection (IDS)
• Create a stealthy bi-direction pipe over innocuous USB devices (something forensic tools are unlikely to spot)
• Minimise forensic artefacts (e.g. execute in memory where possible)
So what’s different/new?

• Simpler networking; TCP/IP interface over WiFi
  • TURNIPSCHOOL uses custom RF protocol
  • IRON-HID uses Bluetooth
• More complete implementation
  • TURNIPSCHOOL never completed firmware for cc1111 against host
  • Numerous small improvements over IRON-HID
• Enables reuse of existing tools
  • Implements a VNC clients for keyboard and mouse input
  • Compatible with metasploit generated payloads
• More stealthy
  • No use of mass storage devices to load malware
  • No use of host’s network i.e. works on airgapped hosts too
• An end-to-end attack
  • From plug in to remote network & command access
• Open Hardware
  • While it works on available hardware, we’re releasing our open hardware design
• Minimal custom bootstrap
Initial Hardware

• April Brother Cactus Micro Rev2
  • Atmega32u4 on one side - host
  • ESP8266 WiFi on the other – our exfil
  • Compact enough to be a flash drive

• Advantages
  • Cheap & available
  • AVR & ESP gives us host side and wifi side

• Disadvantages
  • No I2C
  • No USB A
  • Can’t program ESP directly
  • Minimal storage
  • Can’t reset when in a case
  • LED not controllable from Atmega
  • Not open hardware
New Hardware

• Very similar to the Cactus Micro, but with:
  • USB A male connector!
  • Micro SD Card slot for storage
  • I2C connected with pull up resistor
  • Programmable LED
  • Hall-effect switch to trigger reset when in case
Firmware

• Lightweight USB Framework for AVR (LUFA)
  • Running on the atmega32u4
  • Implements the various USB interfaces seen by the victim

• ESP-Link (UART to TCP firmware)
  • Running on the ESP8266
  • Provides Wifi to device, connects to attacker’s AP
  • Added a VNC implementation to receive key & mouse events to pass to the host
  • Added a multiplexing protocol over the UART to allow communication between various functions
USB Implementation

• Traditional Keyboard and Mouse
  • Emits events received via VNC
  • Can programmably emit events ala RubberDucky
    • Used to stage initial payload on host
    • Used to prevent screensaver engaging

• Generic HID
  • Allows bidirectional packet transfers
  • 64 byte packets
  • 1000 per second (in theory)
Alternate USB Implementations

• For stealthier, more innocuous bi-directional comms

• Text-only printer
  • “Prompt-less” driver installation in Windows

• Sound card
  • Gives us audio out and mic in
  • Problems: Might interfere with primary audio device

• Depends on default permissions
Targets

• Targeting Windows PCs at the moment, plans to expand to OSX then Linux hosts
  • Keyboard/Mouse is generic
  • Payload is platform dependent
• Powershell
  • Available on most Windows workstations
  • C# API available
  • P/Invoke CreateFile, ShowWindowAsync
• Staged approach
• Can avoid touching disk for the most part
  • Excludes P/Invoke-d function definitions above!
Payload Stages

• Stage 1 (PowerShell typed via the keyboard)
  • Optimised for size
  • Open device
  • Read Stage 2
  • Clear-History
  • Hide!

• Stage 2 (PowerShell read from device)
  • Arbitrary complexity
Stage 2 payload examples

- CMD.exe

- TCP Listener/Relay/Proxy
  - Enables existing network-based exploits
  - Localhost-only avoids firewalls/alerts

- Metasploit
  - Currently being redirected through the proxy
  - Can use arbitrary msf payloads; meterpreter, cmd, vnc etc.

- Purpose-built payloads
  - Knows how to access the USB device directly
  - Future development for meterpreter
Difficulties experienced

• Programming errors on the ESP8266 result in reboots, any debug logs disappear!

• Flow control
  • TCP is much faster than the UART, and ESP8266 triggers watchdog to reboot if you take too long to process the data. We had to rewrite the ESP-Link TCP handlers to support “resume-able” processing of data
  • UART is faster than the Keyboard
  • HID interrupt transfers occur regardless of a read()

• Disappearing UART interrupts
  • Data received by the ESP8266 would get stuck in the UART FIFO
Demonstrations

• We’ll provide a demo of the toolset in the talk.
• The software will be released at Defcon at:
  • https://github.com/sensepost/USaBUSE
Contacts

- Rogan Dawes
  - rogan@sensepost.com
  - @rogandawes

- Dominic White
  - dominic@sensepost.com
  - @singe