Exploiting Smart-Phone USB Connectivity For Fun And Profit

Angelos Stavrou & Zhaohui Wang
Department of Computer Science
George Mason University
Talk Outline

- Background – Why USB attacks? What’s new here?
  - New attack vectors, different from simple USB storage

- Phone-to-Computer Attack

- Computer-to-Phone Attack

- Phone-to-Phone Attack

- Demo & Discussion Points

- Defenses & Future Work
USB is Pervasive in Gadgets

- All Smart-Phone devices use USB
  - Google Android Devices (HTC, Motorola, …)
  - Apple iPhone
  - Blackberry
  - Others

- Multi-purpose Usage
  - Charging the Device Battery
  - Data & Media Transfer
  - Control external Devices (new capability)
USB-borne Threats only focused on Auto-Mounting

Defense official discloses cyberattack

By Ellen Nakashima
Tuesday, August 24, 2010, 11:16 PM

Now it is official: The most significant breach of U.S. military computers was caused by a flash drive inserted into a U.S. military laptop on a post in the Middle East in 2008.

In an article to be published Wednesday discussing the Pentagon's cyberstrategy, Deputy Defense Secretary William J. Lynn III says malicious code placed on the drive by a foreign intelligence agency uploaded itself onto a network run by the U.S. military's Central Command.

"That code spread undetected on both classified and unclassified systems, establishing what amounted to a digital beachhead, from which data could be transferred to servers under foreign control," he says in transforming data from additional sources...
USB-borne Threats are much more complex…

- USB protocol can be (ab)used to connect *any* device to a computing platform *without* authentication
  - Desktops, Laptops, phones, kiosks, tables (ipad)

- USB Storage is just the tip of the iceberg and it is usually locked-down and scanned by anti-virus and other defenses

- USB Human Interface Devices (HIDs) are one class of devices that are *much* more appealing
  - Keyboard/Mouse/?/? on your Android Phone
  - Other USB devices?
USB-borne Threats are much more complex…

Many other devices:

- Ethernet/Wireless Network Adapter
  - No password, man in the middle for your network traffic installed as the default “gateway”

- Printer
  - Capture all the documents printed

- Joystic(!)

- Biometric USB Reader
  - Brute force your way into a protected system(?)
Phone-to-Computer Attacks

• Program the Phone with USB Gadget API for Linux

• Pretend to be a USB Human Interface Driver,
  - Dell USB keyboard, VendorID=413C,ProductID=2105
  - Touchpad or Mouse

• Pre-programmed key code.
  - User-lever or System-level attacks
  - Anything you would imagine

• Transparent to Victim Machine
  - No Human Input or approval

HIDs are recognized automatically…
Phone-to-Computer Attacks (Cont)

- Traditional autorun attacks are easy but easily detectable

- Autorun and autoplay are default since Windows XP SP2
  - (MS KB967715) tries to address that

- Flash Autoplay Content exploitation by re-enumeration
  - Exploit different content (PDF, HTML, DOC, MP3)
  - ReMount/unmount MMC card controlled by device

- Exploit Autoplay feature of default Media Programs
  - Selectively prepare attack payload, i.e. Malicious mp3 files targeting MacOSX iTunes, pdf targeting unpatched Adobe Reader
  - Highly robust exploit, works for a variety of programs
Computer-to-Phone Attacks

- **Gaining Root Access to the Smart Phone Device**
  - Official: simulate screen tap event to the oem unlock menu on selected devices
  - Universal: linux local root exploit (CVE-2009-1185, RLIMIT_NPROC exhaustion) send via USB

- **Insert malicious payload**
  - Kernel-level: disassemble boot partition
  - Replace kernel zimage with your own
  - Replace Applications

- **Remove traces by un-rooting to avoid detection**
  - We can quickly cleanup, not need for traces
  - Next reboot, not traces at all
  - Very very difficult to identify, it has to happen before next reboot
Computer-to-Phone Attacks (Cont.)

- Kernel manipulation
  - Rootkits
  - Traffic Redirection to a known proxy
  - Data Exfiltration

- Native ARM ELF binary
  - Bypasses Android framework permissions and checks

- A complete phone provisioning process fully automated with evil payload
  - No application-level traces
Phone-to-Phone Attacks - OTG

USB OTG (On-the-Go) controller
- Capability to switch the controller and become a host or a gadget

Smart Phones are shipped with such OTG capable chipset
- Qualcomm QSD8250, Texas Instruments OMAP 3430

The 5th pin (ID) pin identifies the function of the controller host or gadget
- Floating ID denotes gadget, grounded ID denotes host
Smart Phone as a Host Controller

- Specially shorted USB mini-B dongle to signal the OTG controller behave as a host

- USB transgender or USB micro-A to Standard-A Female cable. (out-of-box cable is micro-B to Standard-A Male)
- Power hub, for additional power supply

- Host side software stack, UHCI/EHCI HCD driver, device driver, userland programs
USB Hacking 101

Crucial Steps for USB Hacking:

- Understand the USB Background (coming up)
  - Low-level “USB Hubs” VS device driver
- Good tools to help debugging (Demo)
  - Some tools are helpful but have flaws as we will show
  - Combination of tools much better
- (Some) Hardware hacking
  - Craft cables to put the phone in “Master” mode
  - Use the phone to connect and hack Other Phones
- Patience!
USB Reconnaissance

Operating System Fingerprinting using USB:

- Not all USB implementations are the same
  - Windows vs Linux vs Mac OS X
  - Flavors of Windows
- The protocol is the same but not the implementation
- USB devices in “slave”/gadget mode can identify the OS upon connection
- Smart (i.e. programmable USB devices) can do so much more as we will see.
## USB Reconnaissance

<table>
<thead>
<tr>
<th>USB Gadget Observations</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Windows</td>
</tr>
<tr>
<td>Full function probe</td>
<td>✓</td>
</tr>
<tr>
<td>Bare device w/o configuration retries</td>
<td>6</td>
</tr>
<tr>
<td>Device alive probe</td>
<td>✓</td>
</tr>
<tr>
<td>Single adb/umass interface bus reset</td>
<td>✓</td>
</tr>
</tbody>
</table>
USB Background: Hierarchical Topology
USB: Series of Events (Overview)

1. The host sends a Get Device Descriptor setup request.
2. The host sets up kernel data structures of the device descriptor.
3. The host continues to enumerate all the interfaces.
4. The host sets up endpoints for every interface.
5. USB data transfer starts.

Interrupt notifying the host that a device connected

The peripheral identifies itself

The peripheral supply the configuration, can be dynamically changed in smart gadget

The peripheral specify interface information

Standard USB Handshake
USB: Series of Events

Interrupt notifying the host that a device connected

Get Device Descriptor

The peripheral identifies itself

Standard USB Handshake
USB: Series of Events

The host sends a Get Device Descriptor setup request.

The peripheral identifies itself.

Interrupt notifying the host that a device connected.

Get Device Descriptor

Speed, VendorID, ProductID, Serial No., Manufacture

USB Host

USB Peripheral

Standard USB Handshake
USB: Series of Events

The host send *Get Device Descriptor* setup request

Interrupt notifying the host that a device connected

The peripheral identifies itself

The peripheral supply the configuration, can be dynamically changed in smart gadget

Get Device Descriptor

Get Configuration

Speed, VendorID, ProductID, Serial No., Manufacture

USB Host

USB Peripheral

Standard USB Handshake
USB: Series of Events

Interrupt notifying the host that a device connected

The host send **Get Device Descriptor** setup request

The host setup kernel data structures of the device descriptor

USB Host

USB Peripheral

**Get Device Descriptor**

Speed, VendorID, ProductID, Serial No., Manufacture

**Get Configuration**

Mass-storage, USB Ethernet etc.

**Standard USB Handshake**

The peripheral identifies itself

The peripheral supply the configuration, can be dynamically changed in smart gadget
USB: Series of Events

1. Interrupt notifying the host that a device connected
2. The host sends a *Get Device Descriptor* setup request
3. The host setup kernel data structures of the device descriptor
4. The peripheral identifies itself
5. The peripheral supply the configuration, can be dynamically changed in a smart gadget
6. The peripheral specify interface information

- Standard USB Handshake
- USB Host
- USB Peripheral

Get Device Descriptor
- Speed, VendorID, ProductID, Serial No., Manufacture
- Mass-storage, USB ether etc.
- Get Configuration
- Get Interface Descriptor
USB: Series of Events

- Interrupt notifying the host that a device connected
- The host sends a Get Device Descriptor setup request
- The host setup kernel data structures of the device descriptor
- The host continues to enumerate all the interfaces

**Standard USB Handshake**

- Get Device Descriptor
  - Speed, VendorID, ProductID, Serial No., Manufacture
- Get Configuration
  - Mass-storage, USB ether etc.
- Get Interface Descriptor
  - USB Interface Class, Subclass, Protocol

**USB Host**

**USB Peripheral**

**Standard USB Handshake**
USB: Series of Events

1. The host sends a Get Device Descriptor setup request.
2. The host setup kernel data structures of the device descriptor.
3. The host continues to enumerate all the interfaces.
4. The host sets up endpoints for every interface.

Interrupt notifying the host that a device connected

USB Host

USB Peripheral

Standard USB Handshake
USB: Series of Events (Overview)

1. **Interrupt notifying the host that a device connected**

2. The host sends the **Get Device Descriptor** setup request.

3. The host sets up kernel data structures of the device descriptor.

4. The host continues to enumerate all the interfaces.

5. The host sets up endpoints for every interface.

6. USB data transfer starts.

**USB Host** and **USB Peripheral**

**Standard USB Handshake**

- **Get Device Descriptor**
  - Speed, VendorID, ProductID, Serial No., Manufacture

- **Get Configuration**
  - Mass-storage, USB ether etc.

- **Get Interface Descriptor**
  - USB Interface Class, Subclass, Protocol

- The peripheral identifies itself.

- The peripheral supply the configuration, can be dynamically changed in smart gadget.

- The peripheral specify interface information.
Device Configuration Map
USB Host Enumeration

- Enumeration: How the host learns about devices
- All USB devices must support (HW/SW) control transfers, the standard requests, and endpoint zero.
- Smart gadgets are often composite devices
- Enumeration is transparent and automatic

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Description</th>
<th>Device Type</th>
<th>Vendor ID</th>
<th>Product ID</th>
<th>Service Name</th>
<th>Driver Filename</th>
<th>Serial Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>O2 SE Flash OMAP3430 MI</td>
<td>Motorola Flash Interface</td>
<td>Vendor Specific</td>
<td>22b8</td>
<td>41e0</td>
<td>MotDev</td>
<td>motodrv.sys</td>
<td></td>
</tr>
<tr>
<td>O2 SE Flash OMAP3430 MI</td>
<td>USB Composite Device</td>
<td>Unknown</td>
<td>22b8</td>
<td>41e1</td>
<td>usbccgp</td>
<td>usbccgp.sys</td>
<td></td>
</tr>
<tr>
<td>Palm Handheld</td>
<td>Palm Handheld</td>
<td>Vendor Specific</td>
<td>0630</td>
<td>0061</td>
<td>PalmUSBd</td>
<td>PalmUSBd.sys</td>
<td>Palm3N12345678</td>
</tr>
<tr>
<td>Nexus One</td>
<td>Google, Inc.Nexus One USB Device</td>
<td>Unknown</td>
<td>18d1</td>
<td>4e11</td>
<td>usbccgp</td>
<td>usbccgp.sys</td>
<td>HT9CNP804091</td>
</tr>
<tr>
<td>Nexus One</td>
<td>USB Mass Storage Device</td>
<td>Mass Storage</td>
<td>18d1</td>
<td>4e11</td>
<td>USBSTCR</td>
<td>USBSTCR.SYS</td>
<td></td>
</tr>
<tr>
<td>Nexus One</td>
<td>Android ADB Interface</td>
<td>Vendor Specific</td>
<td>18d1</td>
<td>4e11</td>
<td>WhUSB</td>
<td>WhUSB.sys</td>
<td></td>
</tr>
<tr>
<td>Nexus One</td>
<td>Gadget Serial</td>
<td>CDC Data</td>
<td>18d1</td>
<td>4e11</td>
<td>usbsr</td>
<td>usbsr.sys</td>
<td></td>
</tr>
<tr>
<td>Nexus One</td>
<td>Nexus One</td>
<td>Vendor Specific</td>
<td>18d1</td>
<td>4e11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola A855</td>
<td>Motorola A855 USB Device</td>
<td>Unknown</td>
<td>22b8</td>
<td>41db</td>
<td>usbccgp</td>
<td>usbccgp.sys</td>
<td>04038800000C01D</td>
</tr>
<tr>
<td>Motorola A855</td>
<td>USB Mass Storage Device</td>
<td>Mass Storage</td>
<td>22b8</td>
<td>41db</td>
<td>USBSTCR</td>
<td>USBSTCR.SYS</td>
<td></td>
</tr>
<tr>
<td>Motorola A855</td>
<td>Mct Composite ADB Interface</td>
<td>Vendor Specific</td>
<td>22b8</td>
<td>41db</td>
<td>androidusb</td>
<td>motoandroid.sys</td>
<td></td>
</tr>
</tbody>
</table>
USB Enumeration Hierarchy

- Device
  - Configuration
    - Interface
      - Endpoint

- Configuration changes the ProductID
  - USB debugging will Change N1’s ProductID from 4e11 to 4e12)
• Show Exploitation of Computer using the phone as Keyboard

• Android based
  but *any* smart phone device with modern USB controller can perform the attack

• *Any* operating system is vulnerable, core functionality not just a hack

• We can lunch, reboot, redirect, …
Discussion

• USB connections are unprotected in current USB 1.1/2.0/3.0 protocol

• USB is the new venue for emerging exploits due to trust in physical proximity

• Smart gadget can cause more damages than traditional passive USB devices.

• Mutual USB authentication

• Revise the USB protocol for security features
Phone-to-Computer Defenses

Potential Defense Strategies

• Disable autorun on USB storage device
  ➢ MS KB971029, non mandatory

• Disable all USB storage devices from automatically attaching
  ➢ MS KB823732

• Validate the Authenticity of the USB Devices once upon connect
  ➢ Bluetooth devices
  ➢ Does not prevent attacks from corrupted devices
Discussion – Defenses?

Potential Defenses – Adding Device Authentication is:

- Interrupt notifying the host that a device connected
- The host send Get Device Descriptor setup request
- The host setup kernel data structures of the device descriptor
- The host continues to enumerate all the interfaces
- The host sets up endpoints for every interface
- USB data transfer starts

a) Standard USB Handshake

b) USBSec I Handshake
Discussion– Defenses?

- Adding static token authentication is not enough
  - Guessable
  - Easy to bypass (wait for the USB device to get authenticated, swap to another device)
  - Data Exfiltration

- Mutual Dynamic Authentication is good but...
  - Passive and Dumb devices cannot cope with
  - Many devices support partially the protocols
  - Windows USB-Hub subsystem a problem…
Discussion– Defenses?

- Getting the Human in the loop
  - Bluetooth has tried that
    - It works but only to validate the device it cannot prevent a device which is “approved” but compromised from corrupting and taking over the other end.

- The Solution requires Human to verify both Type of Device and restrict its permissions
  - Very very difficult given the current user body
  - Can only be applied to enterprise settings
  - Disabling the USB not an option (Why? Recharging….)