HiTB '09: How Low Will Malware Go?

Very Low Apparently!

Nishad Herath <nishad.herath@novologica.com>

Special Thanks To...

- My wonderful family for tolerating my numerous obsessions (see, I thanked you first honey! – sorry for missing your birthday again...).
- HiTB organizers, staff and sponsors – we won't be here without you. Keep up the good work!
- HiTB participants – thank you for your interest and support.
- Zynamics GmbH – for providing access to some of the best reverse engineering tools in the market not to mention listening to some of the most unreasonable feature requests.
- Some of the old school ninjas here today – thank you for years of inspiration, collaboration, friendship, and entertainment!

The Modern Computer

- The modern computer is not one, but a system of computers, with one or more main CPUs, one or more GPUs and other computational elements embedded in various peripheral devices.

The Modern Computer: GPU

- Virtually all consumer GPUs are general purpose and programmable today – in C no less!
- NVIDIA CUDA
- ATI/AMD stream computing
- Apple OpenCL

The Modern Computer: GPU

- Code is compiled into vendor specific intermediate language (ex: NVIDIA PTX, AMD IL).
- Device specific GPU code binaries are generated and embedded into platform executables.

The Modern Computer: GPU

- No Debugging at the hardware level.
- Source level debugging is performed either through an emulator which emulates a vendor IL or through a specially compiled binary which generates equivalent CPU code for debugging on the host CPU.

The Modern Computer: GPU

- No self-modifying code possible.
- However, a virtual machine can be coded which in turn supports self modifying code :-)
- *“GPU Powered Malware”* by Daniel Reynaud – Ruxcon '08

The Modern Computer

- Computational elements in many peripheral devices contain programmable, general purpose microcontrollers!
  - Keyboard (remember the good old 8051?)
  - Optical drive
  - Hard disk
  - Some network controllers
  - Some multimedia devices
  - etc.

The Modern Computer

- Even many smartphones have at least two programmable microcontrollers and a DSP.
  - Application processor (typically an ARM core).
  - Baseband processor.
  - Media encoder/decoder (DSP).

Security Implications

- Peripheral devices have low level access to the system.
- Peripheral devices generally perform significant tasks with serious security impact.
- Programmable microcontrollers in peripheral devices can be vulnerable to:
  - Firmware update attacks.
  - Runtime code injection attacks.

Firmware Update Attack Example

- Keyboard sniffer running on the Apple aluminium keyboard by modifying an Apple firmware update, thus injecting the sniffer code.
  - *“Reversing and Exploiting an Apple® Firmware Update”* by K. Chen – BlackHat Briefings Las Vegas, USA '09

Runtime Code Injection Example

- Apple iPhone baseband unlock ultrasn0w.
- Exploits a vulnerability that exists in the iPhone baseband processor firmware, where the communications interface between the baseband processor and the application processor is implemented.
  - Stack overflow in AT+XLOG=1,"..." command.
- Exploit modifies the baseband code in memory to achieve the unlock.
- http://www.theiphonewiki.com/wiki/index.php?title=Ultrasn0w

Evolution of Malware

- Constant innovation to avoid detection by security products.
- Increasing in sophistication.
- 100% exploitation rate is not required.
- Increasingly making use of various niche opportunities on a global scale.
- Targeted attacks are getting more prevalent.
- So how low will malware go?

Why Run Malcode on Peripheral Processors?

- Current security products are not geared towards dealing with malcode running on peripheral processors.
  - Threats are not mainstream.
  - Lacks OS infrastructure to extend the reach.
  - Supporting the highly diverse device space is problematic.

Why Run Malcode on Peripheral Processors?

- Malware analysts are not generally familiar with the vast number of devices out there, with various microcontrollers and instruction sets (ARM, MIPS, Motorola 68k, Fujitsu FR, MCS 51, H8, MELPS740 etc.)
- Largely undocumented.
- Runtime debugging tools for analyzing code on peripheral microcontrollers are extremely primitive, if available at all to the malcode analyst.
- Device level debugging aids such as JTAG is sometimes disabled once the device leaves the factory.

Why Shouldn't Malcode Run on Peripheral Processors?

- For the very same reasons!
- BUT while defensive aspects of security needs to cast the net far and wide, offensive aspect can focus deep and narrow.
- Targeted attacks against a particular set of devices in a general sense or as used by a particular victim of interest.

Sorting Through The Mess

- Who can help? Who should we try to convince?
  - Device manufacturers
  - Computer vendors
  - OS vendors
  - Security and Systems management software vendors

Device Manufacturers

- Adhere to a strong security development lifecycle, for all aspects of a device – hardware, firmware and drivers.
  - Verify your hardware.
  - Audit your firmware and drivers!
  - Convince your chipset, OS, SDK, library and compiler vendors to audit their code too!
  - Focus developer attention to security when producing code.
  - Ensure that the developers are aware of the greater security model in which the device is a part of.

Device Manufacturers

- Adhere to a strong security development lifecycle, for all aspects of a device – hardware, firmware and drivers.
  - Device Developers – DIY: Don't trust device drivers to only pass valid commands/data.
  - Device Driver Developers - DIY: Don't trust the device to return valid data/responses.
  - Distrust and the “Do It Yourself” ethic are wonderful things for security!
  - For what it's worth, hardware should not trust firmware to not do things that are not meant to be done in the greater scheme of things either ;-)

Device Manufacturers

- Adhere to a strong security development lifecycle, for all aspects of a device – hardware, firmware and drivers.
  - Don't trust anyone to adhere to the correct standard/protocol. Validate, then validate again, and then validate some more.
  - Pay especially close attention to host <-> device communication handlers and structured data parsing code.
  - Ensure appropriate security access controls at the driver level.

Device Manufacturers

- Implement some security in the firmware update process
  - No, (trivially) obfuscating the update program doesn't count :-p
  - Please use decent crypto!!!
  - Perhaps downgrading firmware is not a good thing at times. E-fuses are cheap?
- Implement a firmware integrity validation process
  - It must be resilient against manipulation by rogue firmware.
  - Don't ask firmware to reveal itself or validate its own integrity.
  - Provide access to this process to computer vendors and/or security vendors.

Device Manufacturers

- In a perfect world, device manfacturers would also:
  - Provide notifications of security related updates in a uniform and accessible fashion, consistently.
  - Coordinate with computer vendors and security vendors regarding the latest updates in a uniform and accessible fashion, consistently.

Computer Vendors

- Coordinate with device vendors to promptly and proactively deal with security related hardware, firmware and driver updates.
- Coordinate with device vendors to provide the means to validate device integrity and security for peripheral devices.
- Open this functionality to the operating system, security and systems management software.

OS Vendors

- Coordinate with device vendors and computer vendors to promptly and proactively deal with security related hardware, firmware and driver updates.
- Extend device integrity beyond the driver binary integrity and make it an integral part of the greater system security model.
- Enable security and systems management software to seamlessly integrate into this model.

Security and Systems Management Software Vendors

- Lobby for greater access to prompt, security related device update information.
- Lobby for an operating system security model that takes comprehensive device security and integrity as an integral component of the security model.
- Invest in technology to ensure an acceptable standard of device security and integrity.
- Create greater visibility for administrators.

Chasing Ghosts

- Illustrate what is achievable by (ab)using firmware on a very capable device.
- Firmware sourced from the vendor's support site in the form of a firmware updater
- The device is sourced from... ;-)

Chasing Ghosts: Bus Mastering DMA

- Bus mastering (a.k.a first-party DMA) is a feature of modern buses (ex: PCI).
- Doesn't require a DMA controller under instruction from the CPU, to perform a DMA transfer on behalf of a device.
- Allows a peripheral device to master the bus exclusively for the purpose of directly communicating with another device without involving the CPU or...

Chasing Ghosts: Bus Mastering DMA

- Allows a peripheral device to master the bus in order to perform DMA into system RAM!
- Many high throughput, fast access devices that require as little CPU utilization as possible use this feature.
- Want to guess a few examples?

Chasing Ghosts: 1 + 1 = ?

- Peripheral device with Bus Mastering DMA capability + programmable microcontroller = total pwn4ge!
- Also add a dash of field updateable firmware into the mix for persistence!

Chasing Ghosts

- Updater is packed with ASPack.
- Encrypted firmware included in the updater as a binary resource.
- Part of the code where the encrypted firmware is “processed” and uploaded to the device is riddled with garbage code.
- Zynamics BinNavi is our friend. BinNavi NaviPython script quickly removes garbage code and makes it nicer to read.

Chasing Ghosts

- Extract the firmware once decrypted, but prior to being “processed” for uploading into the device.
- Identify the instruction set.
- Disassemble with IDA Pro.
- Refine the memory map.
- Analyse the key functions, name them, add comments, understand the necessary features.
- Knowledge of the device commands and their formats are extremely helpful in this process.

Chasing Ghosts

- Compare different versions of the firmware and check against bug fix information to understand functionality of some specific parts of the code.
- Compare firmware from related devices to understand device specificities.
- Zynamics BinDiff is invaluable in porting analysis across the various firmware versions – identifying common routines, parts of the RTOS, parts of the SDK etc.
- Understand command handling, responding and setting up bus mastering DMA transfers.
- Embed our “injection” code and payloads into the firmware.

Chasing Ghosts

- Write code to “process” the new firmware and upload it to the device.
- Pray that it works!
- If you've bricked the device, perhaps you could get a warranty replacement if you're lucky?
- In general, it is highly advisable to initially craft a simple mechanism which provides dynamic code injection into the device to ensure any additional code can be tested prior to being written into firmware.
- Again, BinNavi comes in very handy because it may make the discovery of such an existing mechanism (v-u-l-n-e-r-a-b-i-l-i-t-y) a lot easier!

Chasing Ghosts: “Unofficial Firmware”

- After the n number of normal requests made to the device since device initialization, the device “injects” initial payload into the kernel non-paged pool.
- Payload execution is achieved via another carefully placed, but somewhat tricky “injection” ;-)
- Upon completion of payload execution, payload, by calling a custom command implemented in the “unofficial firmware”:
  - Notifies completion status.
  - Instructs the device which payload to “inject” next.
  - Instructs the device how many legitimate requests to the device should occur before the next “injection”.

Chasing Ghosts: “Unofficial Firmware”

- In response, the device erases the previously “injected” payload.
- After the specified number of normal device requests, the newly specified payload is injected into the kernel non-paged pool.
- Payload execution is achieved again via another carefully placed, but somewhat tricky “injection” ;-)
- And so on it goes...

Chasing Ghosts

- The payload exists in the system memory only for a very short time.
- When it does exist in the system memory, it is only one of many payloads.
- Survives reboots and OS re-installations.
- If the device is removed and attached to another computer, that computer is affected.

Questions?

- If you think it's too late in the day for questions, you can reach me later via nishad.herath@novologica.com.
- On the other hand, I'm known for being persuaded time and again to talk tech over drinks ;-)
- Have a great evening and thank you for listening!
- **Invaluable Tools of the Trade:**
  - IDA Pro - http://www.hex-rays.com/idapro/
  - Zynamics BinDiff - http://www.zynamics.com/bindiff.html
  - Zynamics BinNavi - http://www.zynamics.com/binnavi.html