Advanced x86:
BIOS and System Management Mode Internals

Tools

Xeno Kovah & Corey Kallenberg
LegbaCore, LLC
All materials are licensed under a Creative Commons “Share Alike” license.  
http://creativecommons.org/licenses/by-sa/3.0/

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from John Butterworth & Xeno Kovah’s ‘Advanced Intel x86: BIOS and SMM’ class posted at http://opensecuritytraining.info/IntroBIOS.html"
Running Copernicus

- Can be downloaded from:
  - [http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about](http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about)
    - *But in this class, use the one in the Tools folder*
  - From admin prompt cd C:\Copernicus and execute standalone.bat
Running Copernicus

- Copernicus drops its output into the base of the C:\ drive
- We’ll talk about a few of these files
- The size of this binary will equal the size of your SPI flash
- Copernicus dumps all the flash contents (whether readable or not)
- For regions which the CPU/BIOS has no permission to read, it writes all 1’s (0xFF)

*Note: badbios.hdr is just the joke-name of the vulnerable BIOS image John created for this class*
Copernicus_BIOS.bin

- This is a dump of the entire flash BIOS
- Flash is accessed via the programming registers
- Anything Copernicus can’t read (e.g. due to permissions) it just fills in with 0xFFs
  - Reason: to preserve the flash linear address offsets for each region within the binary
• This is a comma-separated file containing the configuration data that was read from the system
• Not all are security-related
  – what ICH/PCH is present
  – FREG registers (just to determine the size of each region and the chip)
  – Many others
  – Because you are attending this class you can access the full .CSV file measurements
  – otherwise you are limited only to the most simple SMM/BIOS lock/unlock measurements (but still the most pertinent!)
Copernicus_Log.txt

- Contains a log file of human-readable information about the system which was just measured
- Also exit status to determine which, if any, measurements failed
“see something, say something”

• If your output contains:

  “Email the following to copernicus@mitre.org so we can look into adding support for this architecture.

  Copernicus Error: Unidentified IO Controller Hub
  vendor=8086, device=9c45

  Memory Controller: vendor=8086, device=0a04”

• Then email it in so they can add support for the hardware
Running Protections.py

- Let's look at the .CSV file in Notepad
- By the end of the course you’ll know what these mean
- Open a CMD prompt in directory C:\Tools\CoP\n  > python protections.py per-file C:\
    – Can also run per-file, affects the sorting of output
- This analyzes our .CSV file for the most basic configuration settings
- SMRAM unlocked in the image above is a bug that should be fixed in the version you're using
- Anyway, let’s get started on the course!
Some Useful Tools of the Trade
Copernicus

• Question your assumptions
• Copernicus is a tool we wrote to determine how prevalent vulnerable BIOS’ are “in the wild”
• Collects the information we are discussing during this class
  – BIOS_CNTL, SPI, Chipset settings, etc.
  – Data can be analyzed offline to determine the vulnerability of BIOS’ in an organization
• So far it has been run on nearly 10,000 systems
• Runs as a Windows driver (32-bit and 64-bit supported)
• Will eventually be released to open source when we have collected enough data to support a whitepaper on our findings
  – Source code released in a code for data agreement
  – Data run on many, many systems
Chipsec

* Intel’s open source firmware measurement tool
* Yuriy Bulygin and John Loucaidis (Intel) introduced this tool at CanSecWest 2014
* [https://github.com/chipsec/chipsec](https://github.com/chipsec/chipsec)
* Developers can add new measurement modules to it
* Can be run from Windows, Linux, or UEFI Shell
Flashrom

• *NIX tool that gives you the ability to read from a variety of SPI flash chips

• I’ve only ever used this on Mac (where there’s no Cop/Chipsec support) or as a backup when combined with a BusPirate hardware reader when my DediProg hardware reader didn’t work

• I don’t think they ever patched the issue that makes it untrustworthy, even though we called it out specifically and showed the source code in our CanSecWest 2013 presentation
Darwin Dumper

- [https://bitbucket.org/blackosx/darwindumper/downloads](https://bitbucket.org/blackosx/darwindumper/downloads)
- Finally got around to using this for our Apple vulnerability research work :)
- Dumps SPI chip & a bunch of other stuff (also pulls EFI variables off the SPI chip, not through parsing, but through using Apple’s API)
- Uses a precompiled flashrom and supporting DirectHW.kext behind the scenes.
  - DirectHW.kext probably won’t work in the future since Apple recognizes it’s a problem
RW Everything (RW-E)

- [http://rweverything.com/](http://rweverything.com/)
- Powerful utility to observe and/or modify platform hardware configurations
- Scriptable so you can test ideas without writing a driver
- Access to PCI Config space, IO space, physical memory
- Freeware, not open source
HxD (hex editor for Windows)

- Good & solid hex editor for Windows
- Some useful features includ file diffing & raw HD access
CFF Explorer

- For analyzing PE files (which lots of BIOS files turn out to be).
- Has a hex view and a basic 16/32/64 bit disassembler which can be useful for disassembly some arbitrary bytes.
- Covered more extensively in the Life of Binaries class.
IDA Pro

- [Link](https://www.hex-rays.com/products/ida/)
- Free version (5.0) works for this class (minus pseudo-code of course)
  - [Download](https://www.hex-rays.com/products/ida/support/download_freeware.shtml)
Snare’s ida-efi Utilities

- [https://github.com/snarez/ida-efiutils](https://github.com/snarez/ida-efiutils)
- Behemoth.h
  - UEFI structures to import into IDA Pro, makes code readable
- EfiGuids.py
  - Big list of EFI Guids parsed from various sources
- And others, but the above two we use most often
EFIPWN (by G33KatWork)

- [https://github.com/G33KatWork/EFIPWN](https://github.com/G33KatWork/EFIPWN)
- EFI image parser, pulls out .efi modules/drivers
- Written in Python, so it can be a bit slow
- Prints the PE file structure of an image
- Can Dump the PE file contents into a file system structure
- This is used behind the scenes for Copernicus’ bios_diff.py. Not necessarily because it’s the best, but because it was the first available when we started the work.
UEFI Firmware Parser (by Teddy Reed)

- [https://github.com/theopolis/uefi-firmware-parser](https://github.com/theopolis/uefi-firmware-parser)
- Can now also extract some stuff that EFIPWN misses
- Prints UEFI non-volatile variables
- Automatically identifies the possible usage of a file based on the file GUID
  - But it can misidentify things which may use the same GUIDs across different vendors (but misidentification is not that big of a deal, since it still gives you a potentially useful name, which is valuable for knowing what it might do, if the file doesn’t have a proper name)

```
Flash Descriptor (Intel PCH) chips 0, regions 3, masters 2, PCH straps 16, PROC straps 0, ICC entries 0
Flash Region
  type= bios, size= 0x300000 (3145728 bytes) details[ read: 11, write: 10, base: 1280, limit: 1023200]
  Firmware Volume: 7a935d9-0468-44a-ce81-0bf617d890df attr 0xffff8eff, rev 1, cksum 0x3c39, size 0x1500
  Firmware Volume Blocks: (21, 0x1000)
  File 0: 4a538818-5ae0-4eb2-e8b2-4888b23657022 type 0x05, attr 0x40, state 0x07, size 0x134490 (126272 bytes)
    Section 0: type 0x01, size 0x134478 (1262712 bytes) (Compression section)
    Section 0: type 0x19, size 0x70010 (734048 bytes) (Paw section)
  Firmware Volume: 7a935d9-0468-44a-ce81-0bf617d890df attr 0xffff8eff, rev 1, cksum 0x3c39, size 0x1500
  Firmware Volume Blocks: (112, 0x1000)
  File 0: 35b898ca-b6a9-49c7-728c-904735cc49b7 (LENOVO_DXE_MAIN_GUID) type 0x05, attr 0x40, state 0x07
    Section 0: type 0x10, size 0x2e04 (58116 bytes) (PE32 image section)
    Section 1: type 0x14, size 0x14 (20 bytes) (User interface name section)
    Name: DxMain
  File 1: 4d37da42-3a0c-4eda-e8b9-bc0e1db4713b (LENOVO_SYSTEM_PPIS_NEEDED_BY_DXE_CORE_GUID) type 0x05, attr 0x40, state 0x07
    Section 0: type 0x1b, size 0x6 (6 bytes) (PEI dependency expression section)
    Section 1: type 0x10, size 0x2bc4 (11204 bytes) (PE32 image section)
    Section 2: type 0x15, size 0x2c (44 bytes) (User interface name section)
    Name: EpisNeededByDxeCore
  File 2: f8a2bc49-3dd1-4d3c-229b-8b0d4a798109 (Guid Defined) type 0x07, attr 0x40, state 0x07, size 0x687c (26724 bytes)
    Guid-Defined: fc1bcdb0-7d31-49aa-6a93-a460d9d0d83 offset= 0x1c attr= 0x2 (AUTH_VALID)
    Section 0: type 0x13, size 0x4c (76 bytes) (Dxe dependency expression section)
    Section 1: type 0x20, size 0x8 (8 bytes) (unknown section)
    Section 2: type 0x10, size 0x57c4 (26564 bytes) (PE32 image section)
    Section 3: type 0x15, size 0x2e (46 bytes) (User interface name section)
    Name: HqQuickLookTrebuchet
```
**UEFI Tool** (by NikolajSchlej)

- Frequently succeeds in extracting files that EFIPWN or UEFI Firmware Parser misses

HP's too cool for a trampoline, they use trebuchets!

https://github.com/LongSoft/UEFITool
UEFIE Extract

• Run as “UEFIEExtract <bios.bin>”
• Will create folder in current directory labeled <bios.bin>.dump
• Will then use the same core logic as UEFITool to extract *all* the files at once into their filesystem structure
• Good for when you want to search through all files, rather than extract a single target file
• Be warned, because of the way it extracts every level of binary at every level of the filesystem, this leads to a massive expansion
  — E.g. a 12MB BIOS can turn into a 200MB dump
Subzero.io

- Ted Reed has created a website that allows you to upload BIOS files, and they will be processed with his UEFI firmware parser
  - Similar to firmware.re, but PC BIOS specific
  - Does one thing and does it well
- Just in time for BH EUR, he also started parsing Copernicus CSV output with protections.py in order to report whether your BIOS is vulnerable or not
  - Run “submit.bat”
- The site will serve to crowd source what good BIOSes look like, so that we can report when we see something that doesn't look like everyone elses
Welcome to Subzero.IO, the largest repository of Flash, BIOS, UEFI volumes and other firmware-related content. You may immediately view/dissect firmware by searching a md5/sha1 or upload your own firmware to process. Learn more about how the dissecting and analysis works.
### Copernicus Report

**Report ID:** cc00401fe39121bb2a0ace91addce61818976b3c

#### Protections

<table>
<thead>
<tr>
<th>Protection</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOSWE_LOCK</td>
<td>False</td>
<td>BIOS write enable is unlocked</td>
</tr>
<tr>
<td>D_LCK</td>
<td>False</td>
<td>SMRAM writable</td>
</tr>
<tr>
<td>PR_COVERS_BIOS</td>
<td>False</td>
<td>BIOS writeable</td>
</tr>
<tr>
<td>SMI_LOCK</td>
<td>False</td>
<td>SMI is unlocked</td>
</tr>
<tr>
<td>SMM_BWP</td>
<td>False</td>
<td>SMMWrite Protect disabled</td>
</tr>
<tr>
<td>SMRR_ENABLED</td>
<td>True</td>
<td>System Management Range Register</td>
</tr>
</tbody>
</table>

#### System Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOSWE_LOCK</td>
<td>False</td>
<td>No description available</td>
</tr>
<tr>
<td>BIOS_CNTL</td>
<td>00</td>
<td>No description available</td>
</tr>
</tbody>
</table>
OpenTPM

- [https://code.google.com/p/opentpm/](https://code.google.com/p/opentpm/)
- Open Source utility written by Corey Kallenberg
- Allows you to retrieve quotes of your TPM PCRs (will be covered in Trusted Computing portion of the class)
- Linux users can use the tpm_bios and tpm modules to do this
Backup

• Tools I haven’t tested
FMEM (Linux)

- FMEM is a little known Linux kernel module that provides you access to all physical memory in a system.
- FMEM creates /dev/fmem.
- Very useful, since /dev/mem restricts access to various regions.
- [http://hysteria.sk/~niekt0/fmem/](http://hysteria.sk/~niekt0/fmem/)

Examples of its use:
- Dumps the last 2 MB of system RAM (FFE00000h – FFFFFFFFh).
  - `dd = /dev/fmem of=~/foo.bin bs=1048576 skip=4094 count=2`
- Therefore, to dump the last 4 MB of memory:
  - `dd = /dev/fmem of=~/bar.bin bs=1048576 skip=4092 count=4`

- Block size (bs) is 1 MB decimal, you can play with bs, skip, and count to make it less obfuscated (or script it).
- So we’re skipping <skip> blocks, then reading <count> blocks.