

# 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).

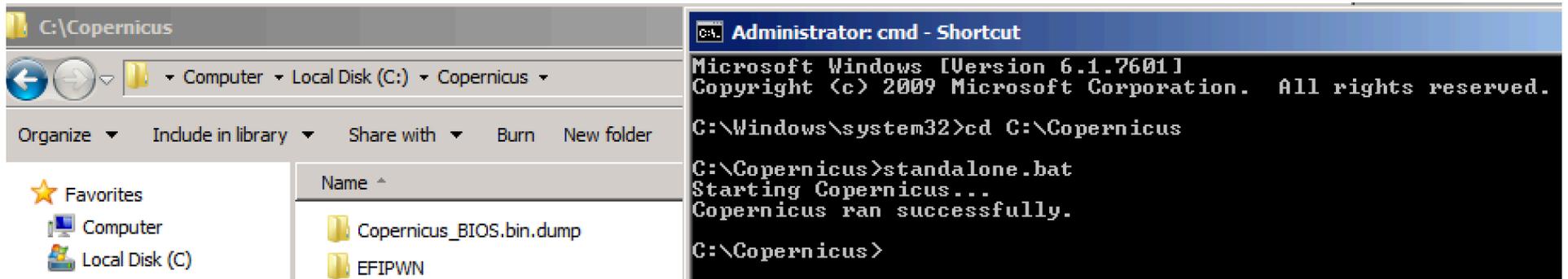


**Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work

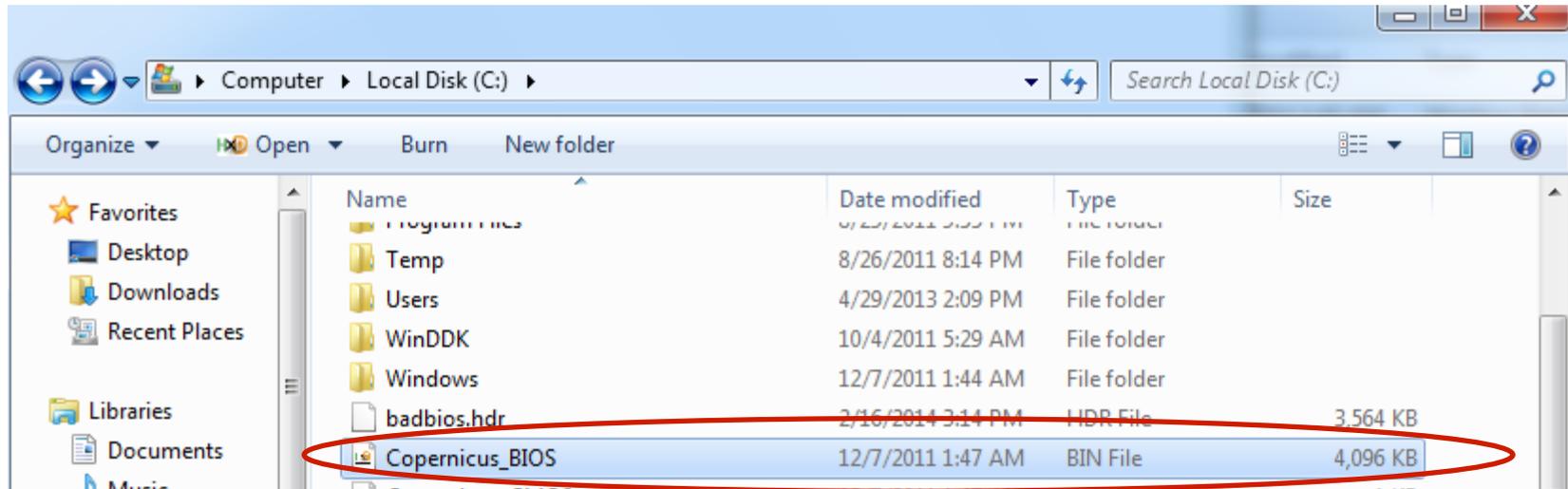
"Is derived from John Butterworth & Xeno Kovah's 'Advanced Intel x86: BIOS and SMM' class posted at <http://opensecuritytraining.info/IntroBIOS.html>"

# Running Copernicus



- Can be downloaded from:
- <http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about>
  - ***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

```
Offset (h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 BA A5 F0 0F 01 00 04 04 06 02 10 02 20 01 00 00 Zy8.....
00000010 13 00 30 00 00 00 00 00 00 00 00 00 FF FF FF FF ..0.....YYYY
00000020 FF YYYYYYYYYYYYYYYY
00000030 FF YYYYYYYYYYYYYYYY
00000040 00 00 00 00 60 02 FF 03 0B 00 5F 02 01 00 02 00 ....`y..._.....
00000050 03 00 0A 00 FF ...YYYYYYYYYYYY
00000060 00 00 1B 1A 00 00 0D 0C 18 02 08 08 FF FF FF FF .....YYYY
00000070 FF YYYYYYYYYYYYYYYY
00000080 FF YYYYYYYYYYYYYYYY
00000090 FF YYYYYYYYYYYYYYYY
000000A0 FF YYYYYYYYYYYYYYYY
000000B0 FF YYYYYYYYYYYYYYYY
000000C0 FF YYYYYYYYYYYYYYYY
000000D0 FF YYYYYYYYYYYYYYYY
000000E0 FF YYYYYYYYYYYYYYYY
000000F0 FF YYYYYYYYYYYYYYYY
00001000 89 64 99 00 0F 01 00 00 FF FF FF FF FF FF FF FF %d™.....YYYYYYYY
00001100 FF YYYYYYYYYYYYYYYY
```

- 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

# Copernicus\_CSV\_Out.csv

	A	B	C	D	E	F	G	H	I	J
1	CHIPSET_DEVICE_ID=2917	CHIPSET_FAMILY=ICH9	FREG0=00000000	FREG1=03FF0260	FREG2=025F000B	FREG3=00020001	FREG4=000A0003	FLASH_DESCRIPTOR_MODE=True	FLASH_CHIP_SIZE=00400000	BIOSWE_LOCK=False
2										
3										

- 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

```
F:\Copernicus_Log.txt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Copernicus_CSV_Out.csv Copernicus_Log.txt
1 MITRE Copernicus Loading
2
3 Thank you for using Copernicus!
4 If you'd like to help us build a master list of vulnerable BIOSes,
5 please email your .csv file to copernicus@mitre.org
6
7 Allocating memory for ICH Parameters object
8 Allocating memory for Flash Chip object
9 Allocating memory for SMBIOS Parameters object
10 Allocating memory for XROM_Summary_t object
11
12 Initializing ICH parameters
13 ICH9 detected, device ID: 0x2917 detected
14 ICH9 memory controller Vendor: 0x8086 ID: 0x2a40.
15 SPIBAR physical address 0xFED1B800
16 SPI Flash is in Descriptor mode
17 Determining size of SPI flash chip
18 SPI Region 0 (Flash Descriptor) base = 00000000, limit = 00000fff
19 SPI Region 1 (BIOS) base = 00260000, limit = 003fffff
20 SPI Region 2 (Management Engine) base = 0000b000, limit = 0025ffff
21 SPI Region 3 (Gigabit Ethernet) base = 00001000, limit = 00002fff
22 SPI Region 4 (Platform Data) base = 00003000, limit = 0000afff
23 SPI Flash chip size = 0x00400000
```

- 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:

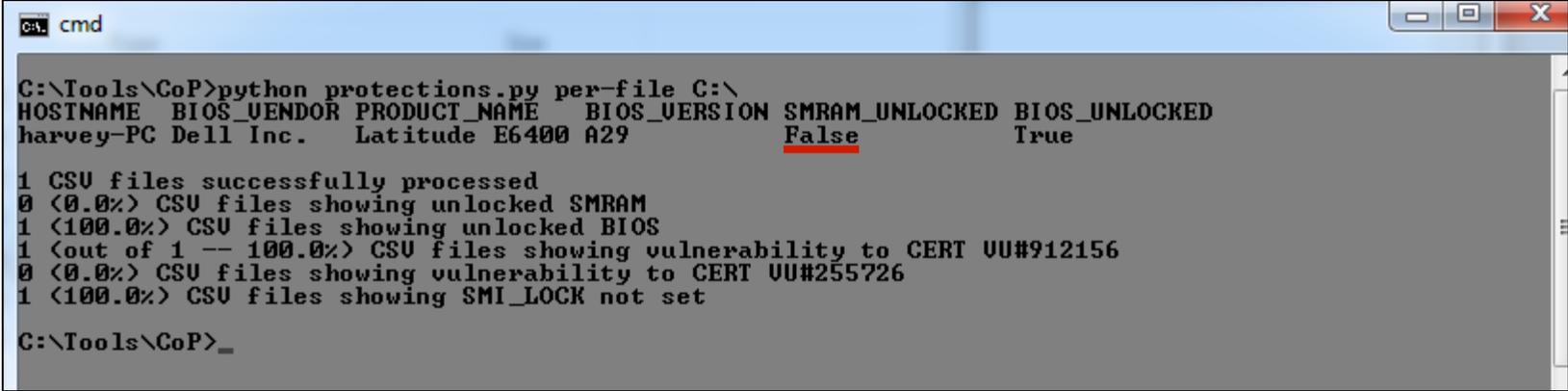
“Email the following to [copernicus@mitre.org](mailto:copernicus@mitre.org) so we can look into adding support for this architecture.

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



```
cmd
C:\Tools\CoP>python protections.py per-file C:\
HOSTNAME BIOS_VENDOR PRODUCT_NAME BIOS_VERSION SMRAM_UNLOCKED BIOS_UNLOCKED
harvey-PC Dell Inc. Latitude E6400 A29 False True

1 CSU files successfully processed
0 (0.0%) CSU files showing unlocked SMRAM
1 (100.0%) CSU files showing unlocked BIOS
1 (out of 1 -- 100.0%) CSU files showing vulnerability to CERT VU#912156
0 (0.0%) CSU files showing vulnerability to CERT VU#255726
1 (100.0%) CSU files showing SMI_LOCK not set

C:\Tools\CoP>
```

- 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\
  - > python protections.py per-version 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

```
[+] imported chipsec.modules.common.bios_wp
[x] [ =====
[x] [ Module: BIOS Region Write Protection
[x] [ =====
BIOS Control (BDF 0:31:0 + 0xDC) = 0x2A
[05]   SMM_BWP = 1 (SMM BIOS Write Protection)
[04]   TSS     = 0 (Top Swap Status)
[01]   BLE     = 1 (BIOS Lock Enable)
[00]   BIOSWE  = 0 (BIOS Write Enable)

[+] BIOS region write protection is enabled (writes restricted to SMM)

[*] BIOS Region: Base = 0x00500000, Limit = 0x00FFFFFF
SPI Protected Ranges
-----
PRx (offset) | Value      | Base      | Limit     | WP? | RP?
-----
PR0 (74)    | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)    | 8FFF0F40 | 00F40000 | 00FFF000 | 1   | 0
PR2 (7C)    | 8EDF0EB1 | 00EB1000 | 00EDF000 | 1   | 0
PR3 (80)    | 8EB00EB0 | 00EB0000 | 00EB0000 | 1   | 0
PR4 (84)    | 8EAF0C00 | 00C00000 | 00EAF000 | 1   | 0

[!] SPI protected ranges write-protect parts of BIOS region (other parts of BIOS can be
modified)

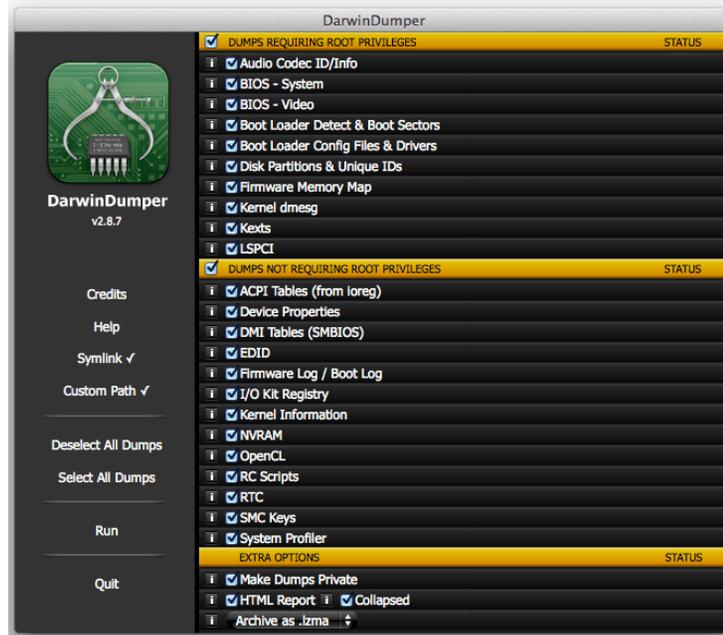
[+] PASSED: BIOS is write protected
```

- Intel's open source firmware measurement tool
- Yuriy Bulygin and John Loucaidis (Intel) introduced this tool at CanSecWest 2014
- <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

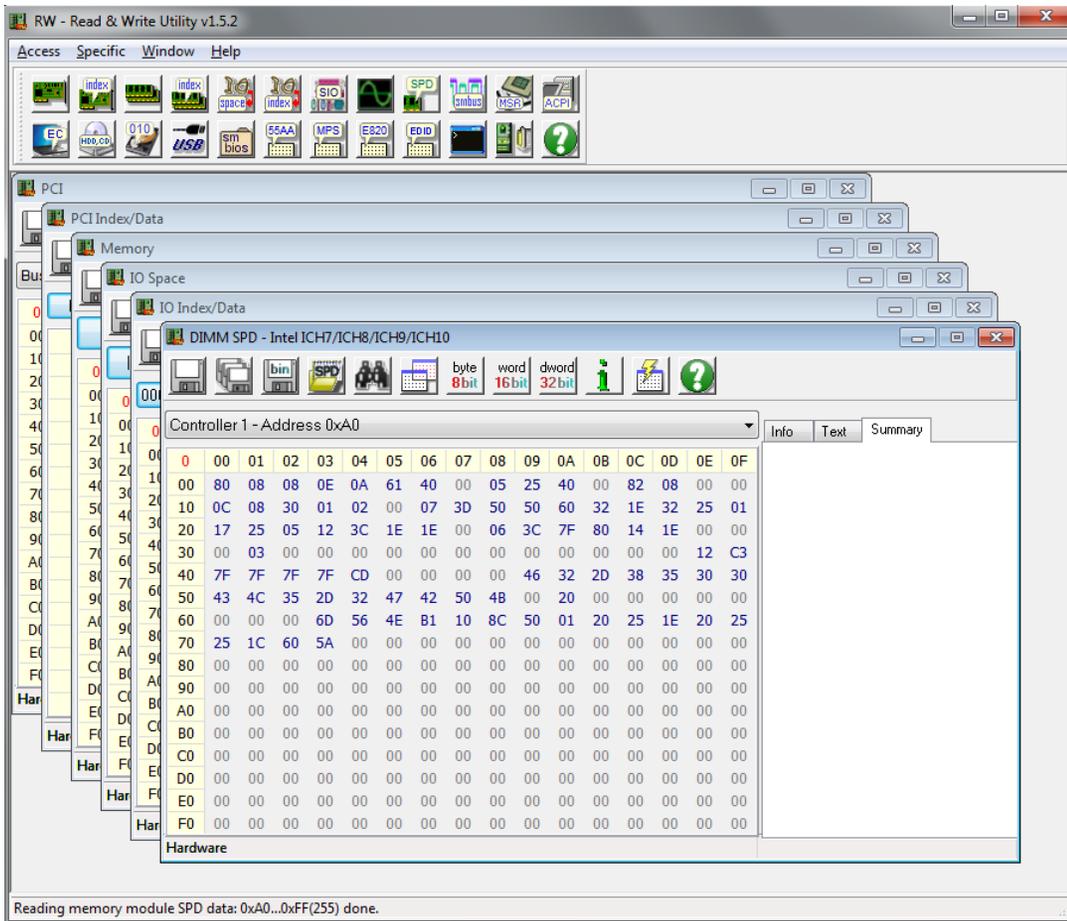
# Darwin Dumper



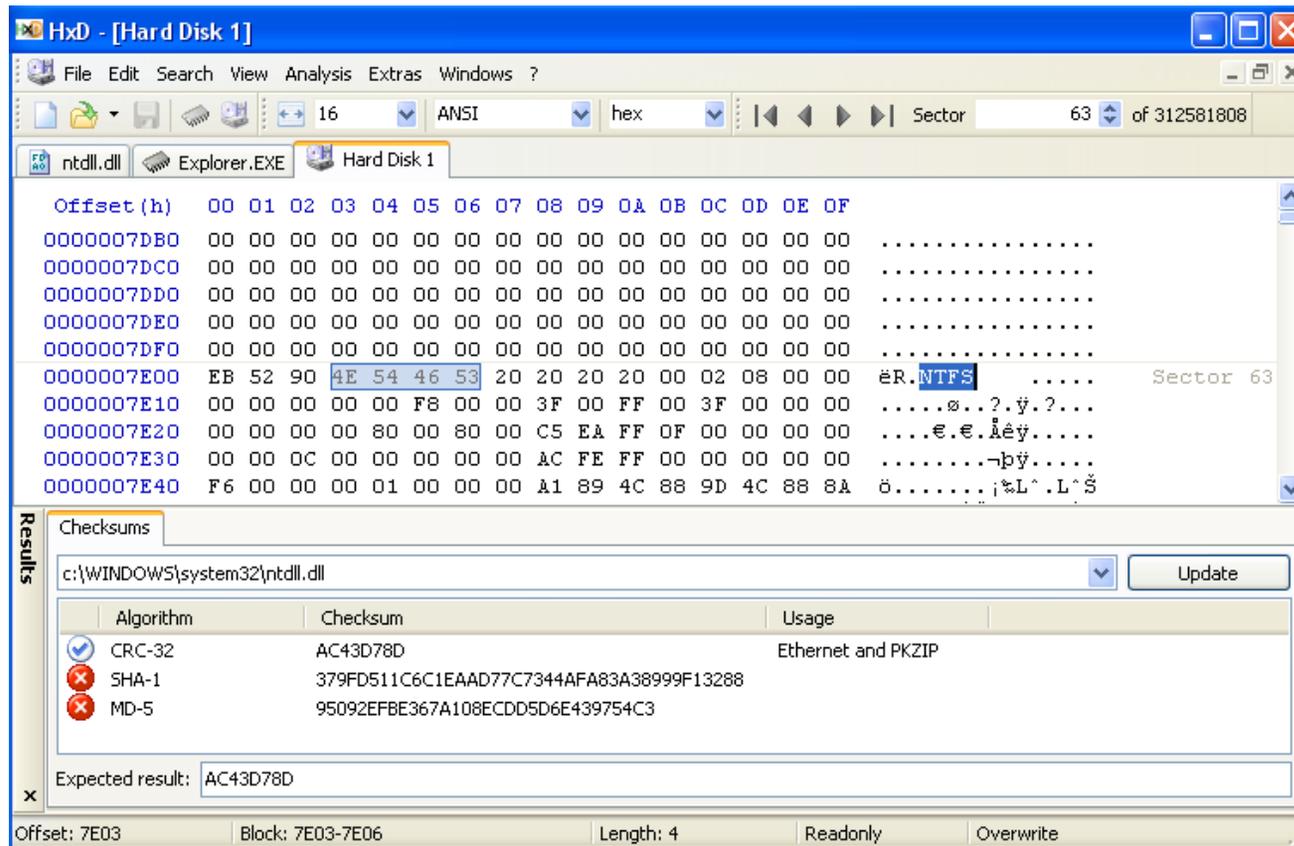
- <https://bitbucket.org/blackosx/darwindumper/downloads>
- Finally got around to using this for our Apple vulnerability research work :)
- Dumps SPI chip & a bunch of other stuff (also pulls EFI variables off the SPI chip, not through parsing, but through using Apple's API)
- Uses a precompiled flashrom 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/>
- 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)



- <http://mh-nexus.de/en/hxd/>
- Good & solid hex editor for Windows
- Some useful features include file diffing & raw HD access

# CFF Explorer

The screenshot shows the CFF Explorer VII interface for a file named 'body.bin'. The left sidebar displays the file's structure, including headers and section headers. The main window shows a table of sections and a disassembly view of the '.text' section.

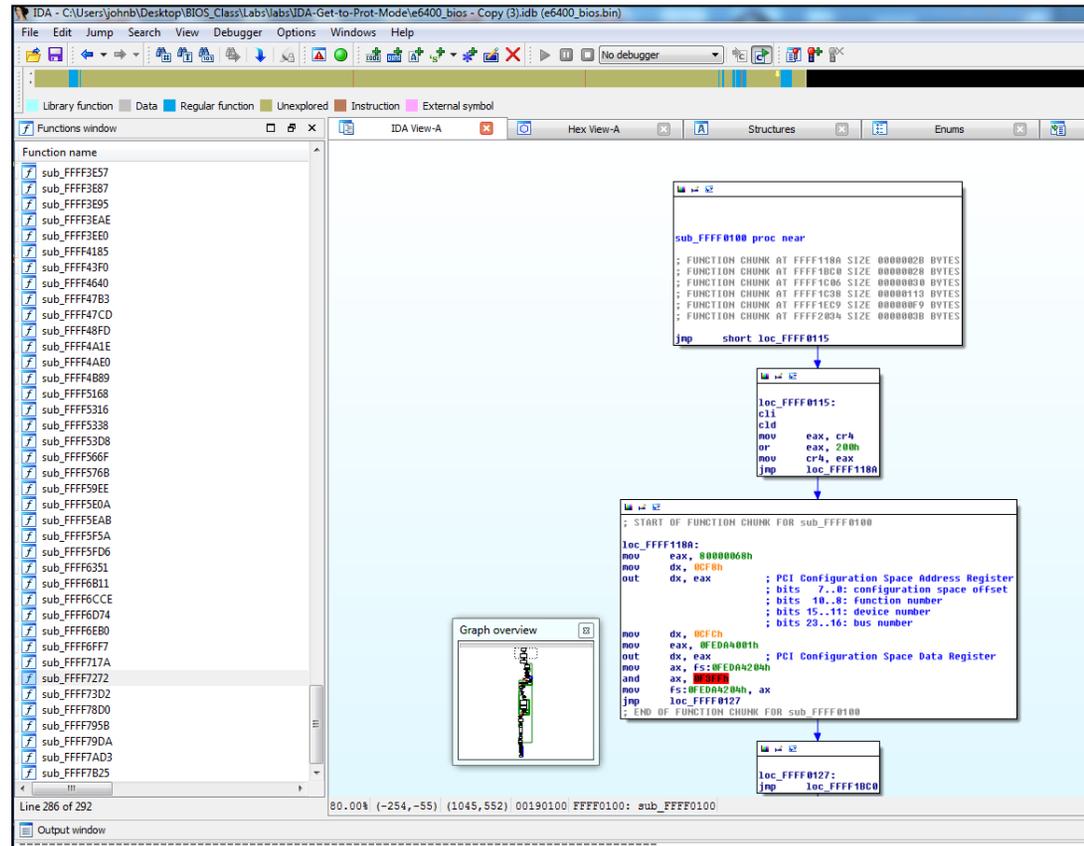
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
000001B8	000001C0	000001C4	000001C8	000001CC	000001D0	000001D4	000001D8	000001DA	000001DC
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000AB5	00000220	00000AC0	00000220	00000000	00000000	0000	0000	68000020
.reloc	00000034	00000CE0	00000040	00000CE0	00000000	00000000	0000	0000	42000040

This section contains:  
Code Entry Point: 0000042B

Offset	0	1	2	3	4	5	D	E	F	Ascii
00000000	10	00	00	80	E0	B8	00	00	80	+... à,ýý...+...
00000010	F0	B8	FD	FF	00	00	00	00	00	ä,ýý.....
00000020	06	D3	C9	88	00	09	DA	1F	89	-OE ...µN >-%Ü
00000030	50	A6	5E	60	5C	C6	B6	18	C6	P ^^\BáB° *# †æ
00000040	95	28	D4	AB	CF	78	0B	FB	DA	( Ó<IxrH D- } e-úÜ
00000050	90	6F	4C	1F	6B	B0	0L	k°@Hc°ãñ }V		
00000060	67	FD	9F	AF	10	EC	gý +i H ü ó^â..			
00000070	10	00	00	80	3C	B9	+... <¹ýý...%IIm			

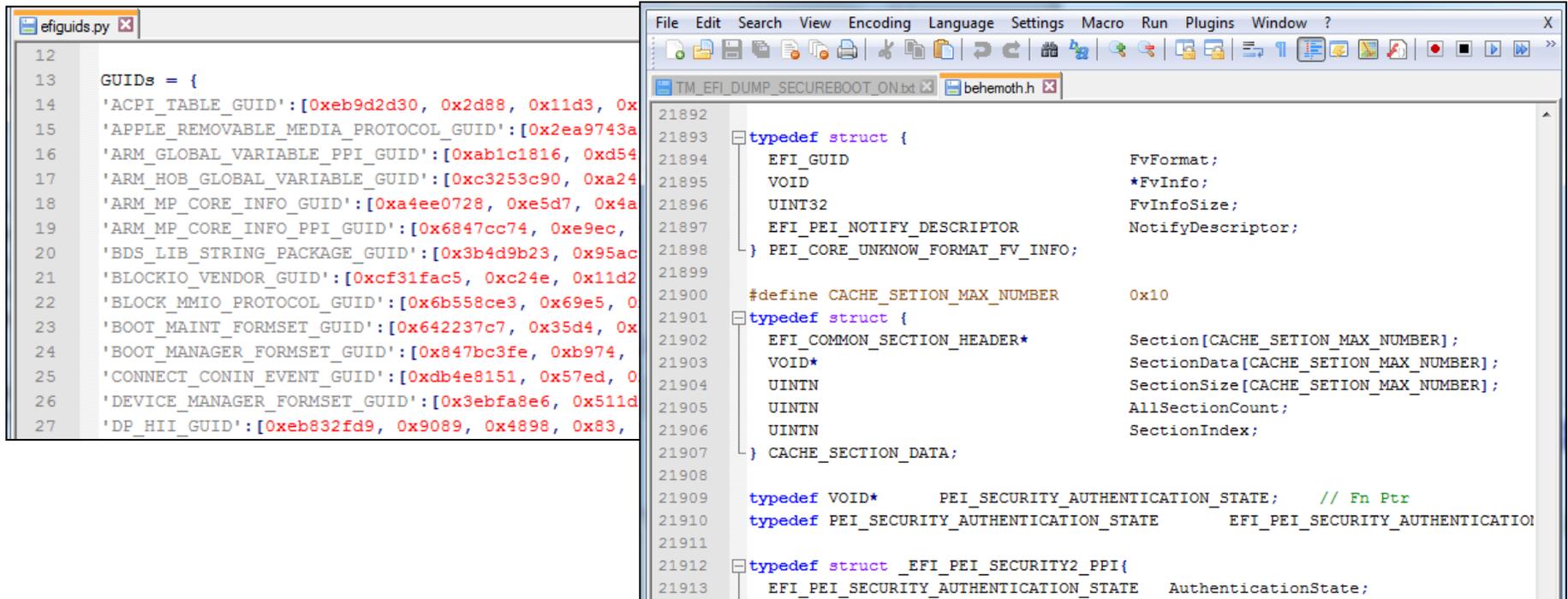
- <http://www.ntcore.com/exsuite.php>
- For for analyzing PE files (which lots of BIOS files turn out to be).
- Has a hex view and a basic 16/32/64 bit disassembler which can be useful for disassembly some arbitrary bytes
- Covered more extensively in the Life of Binaries class

# IDA Pro



- <https://www.hex-rays.com/products/ida/>
- Free version (5.0) works for this class (minus pseudo-code of course)
  - [https://www.hex-rays.com/products/ida/support/download\\_freeware.shtml](https://www.hex-rays.com/products/ida/support/download_freeware.shtml)

# Snare's ida-efi Utilities

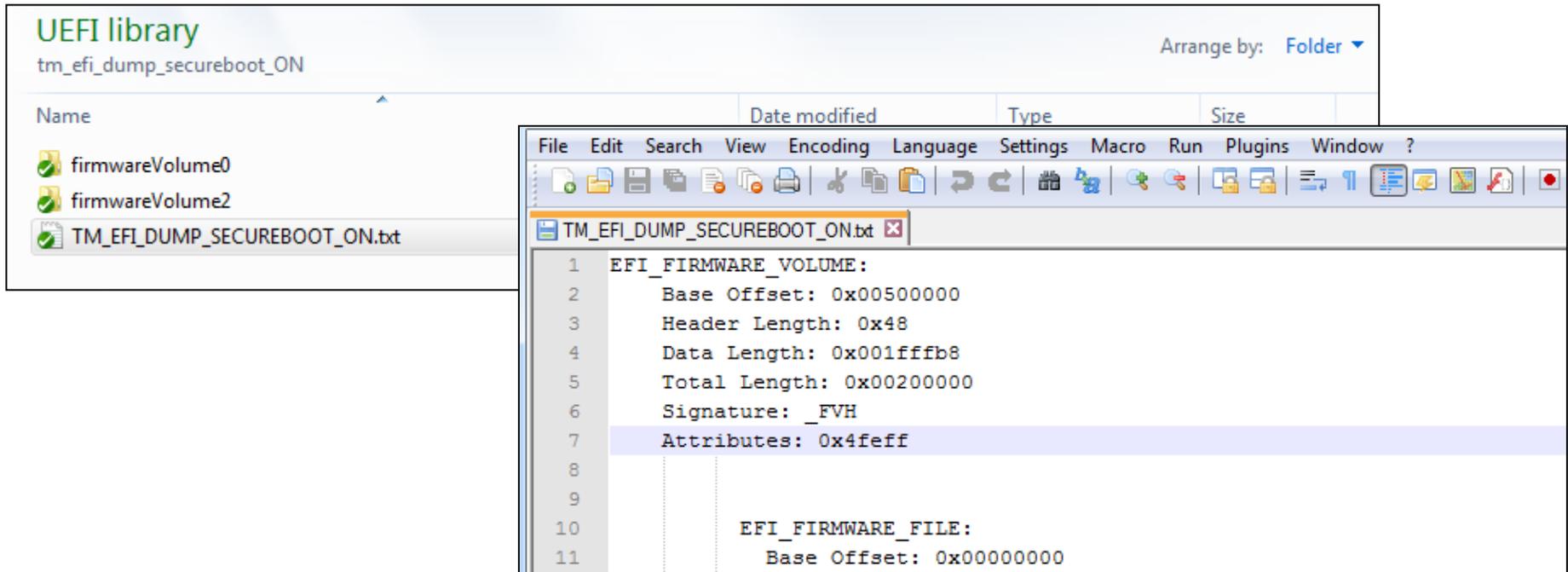


```
efiguids.py
12
13 GUIDs = {
14 'ACPI_TABLE_GUID':[0xeb9d2d30, 0x2d88, 0x11d3, 0x
15 'APPLE_REMOVABLE_MEDIA_PROTOCOL_GUID':[0x2ea9743a
16 'ARM_GLOBAL_VARIABLE_PPI_GUID':[0xab1c1816, 0xd54
17 'ARM_HOB_GLOBAL_VARIABLE_GUID':[0xc3253c90, 0xa24
18 'ARM_MP_CORE_INFO_GUID':[0xa4ee0728, 0xe5d7, 0x4a
19 'ARM_MP_CORE_INFO_PPI_GUID':[0x6847cc74, 0xe9ec,
20 'BDS_LIB_STRING_PACKAGE_GUID':[0x3b4d9b23, 0x95ac
21 'BLOCKIO_VENDOR_GUID':[0xcf31fac5, 0xc24e, 0x11d2
22 'BLOCK_MMIO_PROTOCOL_GUID':[0x6b558ce3, 0x69e5, 0
23 'BOOT_MAINT_FORMSET_GUID':[0x642237c7, 0x35d4, 0x
24 'BOOT_MANAGER_FORMSET_GUID':[0x847bc3fe, 0xb974,
25 'CONNECT_CONIN_EVENT_GUID':[0xdb4e8151, 0x57ed, 0
26 'DEVICE_MANAGER_FORMSET_GUID':[0x3ebfa8e6, 0x511d
27 'DP_HII_GUID':[0xeb832fd9, 0x9089, 0x4898, 0x83,

TM_EFI_DUMP_SECUREBOOT_ON.txt behemoth.h
21892
21893 typedef struct {
21894     EFI_GUID FvFormat;
21895     VOID *FvInfo;
21896     UINT32 FvInfoSize;
21897     EFI_PEI_NOTIFY_DESCRIPTOR NotifyDescriptor;
21898 } PEI_CORE_UNKNOW_FORMAT_FV_INFO;
21899
21900 #define CACHE_SECTION_MAX_NUMBER 0x10
21901 typedef struct {
21902     EFI_COMMON_SECTION_HEADER* Section[CACHE_SECTION_MAX_NUMBER];
21903     VOID* SectionData[CACHE_SECTION_MAX_NUMBER];
21904     UINTN SectionSize[CACHE_SECTION_MAX_NUMBER];
21905     UINTN AllSectionCount;
21906     UINTN SectionIndex;
21907 } CACHE_SECTION_DATA;
21908
21909 typedef VOID* PEI_SECURITY_AUTHENTICATION_STATE; // Fn Ptr
21910 typedef PEI_SECURITY_AUTHENTICATION_STATE EFI_PEI_SECURITY_AUTHENTICATION_STATE;
21911
21912 typedef struct _EFI_PEI_SECURITY2_PPI{
21913     EFI_PEI_SECURITY_AUTHENTICATION_STATE AuthenticationState;
```

- <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)

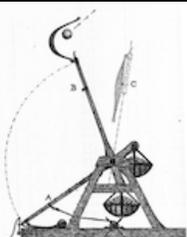


- <https://github.com/G33KatWork/EFIPWN>
- EFI image parser, pulls out .efi modules/drivers
- Written in Python, so it can be a bit slow
- Prints the PE file structure of an image
- Can Dump the PE file contents into a file system structure
- This is used behind the scenes for Copernicus' bios\_diff.py. 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>
- 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:
Firmware Volume: 7a9354d9-0468-444a-ce81-0bf617d890df attr 0xffff8eff, rev 1, cksun 0x4c89, size 0x150
Firmware Volume Blocks: (21, 0x10000)
File 0: 4a538818-5ae0-4eb2-ebb2-488b23657022 type 0x05, attr 0x40, state 0x07, size 0x134490 (126273
Section 0: type 0x01, size 0x134478 (1262712 bytes) (Compression section)
Section 0: type 0x19, size 0x700010 (7340048 bytes) (Raw section)
Firmware Volume: 7a9354d9-0468-444a-ce81-0bf617d890df attr 0xffff8eff, rev 1, cksun 0x4bd3, si
Firmware Volume Blocks: (112, 0x10000)
File 0: 35b898ca-b6a9-49ce-728c-904735cc49b7 (LENOVO_DXE_MAIN_GUID) type 0x05, attr 0x40, st
Section 0: type 0x10, size 0xe304 (58116 bytes) (PE32 image section)
Section 1: type 0x15, size 0x14 (20 bytes) (User interface name section)
Name: DxeMain
File 1: 4d37da42-3a0c-4eda-ebb9-bc0e1db4713b (LENOVO_SYSTEM_PPIS_NEEDED_BY_DXE_CORE_GUID) ty
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: PpisNeededByDxeCore
File 2: fea2bc49-33d1-4d3c-229b-8b0d4a798109 type 0x07, attr 0x40, state 0x07, size 0x687c (
Section 0: type 0x02, size 0x6864 (26724 bytes) (Guid Defined section)
Guid-Defined: fc1bcd0-7d31-49aa-6a93-a4600d9dd083 offset= 0x1c attrs= 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 0x67c4 (26564 bytes) (PE32 image section)
Section 3: type 0x15, size 0x2e (46 bytes) (User interface name section)
Name: HpQuickLookTrebuchet
```



# UEFITool (by NikolajSchlej)

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

<https://github.com/LongSoft/UEFITool>

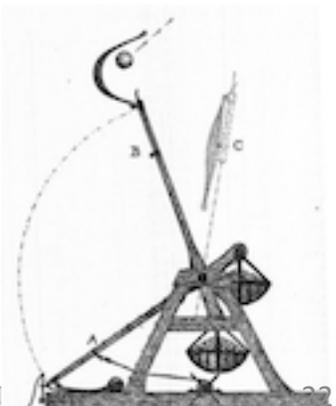
The screenshot shows the UEFITool 0.17.10 application window. The main area displays a tree view of the firmware structure with columns for Name, Action, Type, and Subtype. The 'User interface' section is highlighted. The information panel on the right shows details for the selected entry: Type: 15, Size: 00002a, Text: HpQuickLookTrebuchet.

Name	Action	Type	Subtype
▼ Intel image		Image	Intel
Descriptor region		Region	Descriptor
GbE region		Region	GbE
ME region		Region	ME
▼ BIOS region		Region	BIOS
Padding		Padding	
▼ 7A9354D9-0468-444A-81CE-0...		Volume	
▼ 4A538818-5AE0-4EB2-B2EB...		File	DXE core
▼ Compressed section		Section	Compressed
▼ Raw section		Section	Raw
Padding		Padding	
▼ 7A9354D9-0468-444A...		Volume	
▶ 35B898CA-B6A9-49C...		File	DXE core
▶ 4D37DA42-3A0C-4ED...		File	PEI module
▼ FEA2BC49-33D1-4D3...		File	DXE driver
▼ FC1BCDB0-7D31-4...		Section	GUID defined
DXE dependenc...		Section	DXE dependency
Unknown secti...		Section	Unknown
PE32+ image s...		Section	PE32+ image
User interfac...		Section	User interface
▶ 7367B3B6-DAAF-4A2...		File	DXE driver
▶ ED31ED0E-E974-456...		File	DXE driver
▶ 53AC75F6-D2F2-453...		File	DXE driver
▶ 8F2B9D3B-9AE8-446...		File	DXE driver
▶ 85D5DF6C-E3A1-11D...		File	DXE driver
▶ 85D5DF6C-E3A1-11D...		File	DXE driver
▶ 1C4709C6-CC81-461...		File	DXE driver
▶ B2DE1A6E-3045-409...		File	DXE driver
▶ 325F9B82-45C8-446...		File	DXE driver
▶ 470B83AC-33C7-49B...		File	DXE driver

Information

Type: 15  
Size: 00002a  
Text: HpQuickLookTrebuchet

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



# UEFIExtract

- Run as “UEFIExtract <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 else



## Search firmware by hash or upload for analysis



🔍 Search

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.

## › Firmware Details

## › Copernicus Report

› Structure 3

## › Data Parts

› UEFI Files 177

## › Variables

› Strings 9314

## › Similarity

## › Analysis

## Copernicus Report

Report ID: ac00401fe39121bb2a0ace91addce61818976b3c

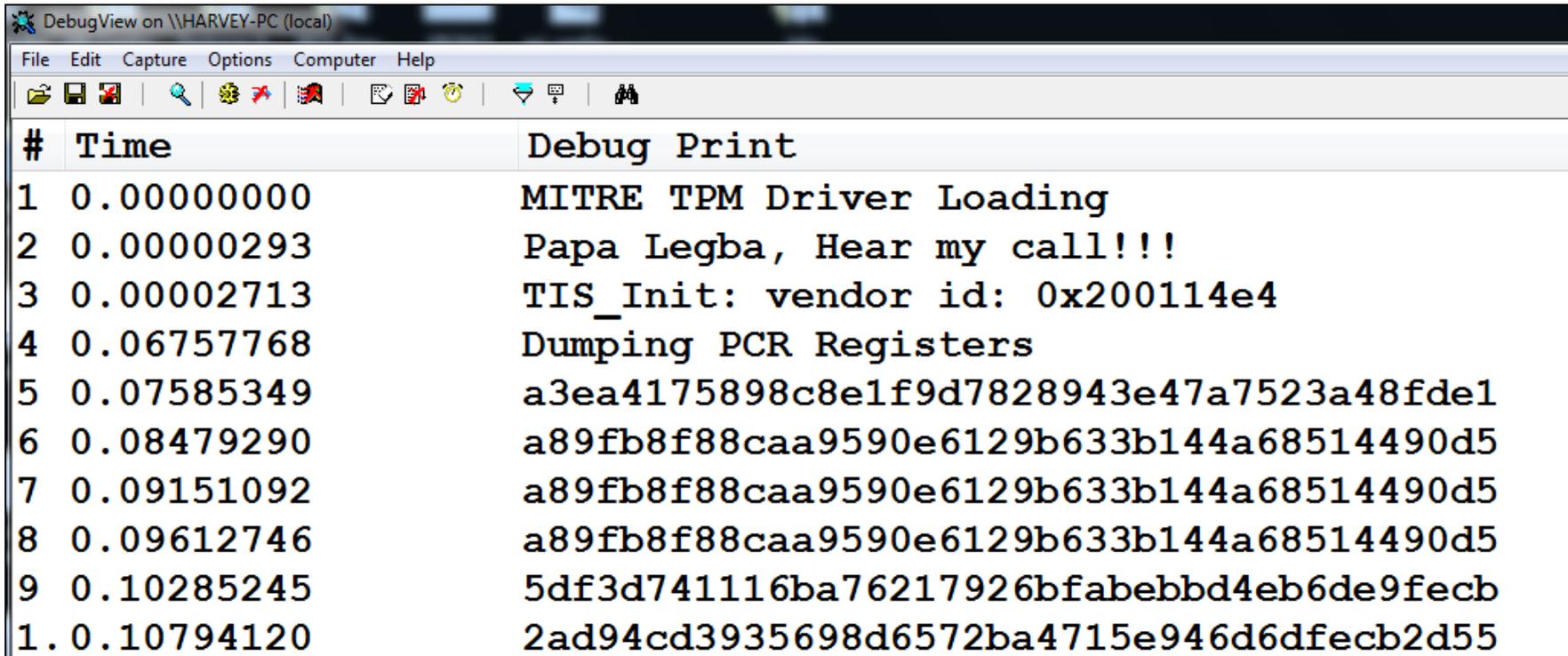
## Protections

Protection	Setting	Description
BIOSWE_LOCK	False	BIOS write enable is unlocked
D_LCK	False	SMRAM writable
PR_COVERS_BIOS	False	BIOS writeable
SMI_LOCK	False	SMI is unlocked
SMM_BWP	False	SMM Write Protect disabled
SMRR_ENABLED	True	System Management Range Register

## System Attributes

Attribute	Setting	Description
BIOSWE_LOCK	False	No description available
BIOS_CNTL	00	No description available

# OpenTPM



The screenshot shows a DebugView window titled "DebugView on \\HARVEY-PC (local)". The window contains a table of debug print events. The table has three columns: "#", "Time", and "Debug Print". The events are as follows:

#	Time	Debug Print
1	0.00000000	MITRE TPM Driver Loading
2	0.00000293	Papa Legba, Hear my call!!!
3	0.00002713	TIS_Init: vendor id: 0x200114e4
4	0.06757768	Dumping PCR Registers
5	0.07585349	a3ea4175898c8e1f9d7828943e47a7523a48fde1
6	0.08479290	a89fb8f88caa9590e6129b633b144a68514490d5
7	0.09151092	a89fb8f88caa9590e6129b633b144a68514490d5
8	0.09612746	a89fb8f88caa9590e6129b633b144a68514490d5
9	0.10285245	5df3d741116ba76217926bfabebbd4eb6de9fecb
1.0	0.10794120	2ad94cd3935698d6572ba4715e946d6dfecb2d55

- <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/>
  
- 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