UEFI Firmware Rootkits: Myths and Reality
(Revisited for ZeroNights)

Alex Matrosov
@matrosov

Eugene Rodionov
@vxradius
Agenda

- Historical overview of BIOS rootkits
- Threat Model for UEFI Rootkits
- BIOS Rootkits In-The-Wild
  - HackingTeam Rootkit
  - BIOS Implants
  - Computrace/LoJack
- BIOS Update Issues
- Secure Boot Issues
- Forensic Approaches
History of BIOS rootkits
History of BIOS rootkits

In the Wild

1998 WinCH, ACPI Rootkit, PCI OptRom Rootkit
2006 Mebromi/BIOSkit, Computrace
2007 ACPI Rootkit, POI OptRom Rootkit
2008 SMM Rootkit, BIOS Patching
2009 Rakshasa
2010 1st SecureBoot Bypass, Darth Venamis
2011 Dream Boot, DEITYBOUNCE
2012 BadBIOS Hysteria, Thunderstrike 1
2013 SecureBoot Bypass, HT Killer
2014 Memory Sinkhole, ThinkPwn
2015 Memory Sinkhole, Thunderstrike 2
2016 PEI Backdoor, SMM->VMM Backdoor
2017 BANANABALLOT

Proof of Concept

Move to UEFI world with Secure Boot
MS Win10: Virtualization Based Security Era
History of BIOS rootkits

Low Threat Activity

Targeted Attacks

Low Research Activity

High Research Activity
Threat Model for UEFI Rootkits
OS Kernel-Mode (Ring 0)

- **Mitigations:** PatchGuard, Code Signing Policy
- **Prevention:** AV HIPS

Boot code (MBR/VBR)

- **Mitigations:** Secure/Measured Boot
- **Prevention:** AV HIPS

BIOS/UEFI Firmware SMM (Ring -2)

- **Mitigations:** ??? (Intel BIOS/Boot Guard? STM?)
- **Prevention:** ???
Legacy BIOS vs. UEFI

- No more MBR and VBR/IPL code
- Different hard drive partitioning scheme: GPT (GUID Partition Table)
- Secure Boot and Measured Boot
UEFI BIOS Firmware Rootkits

Patching UEFI “Option ROM”
UEFI DXE Driver in Add-On Card (Network, Storage ..)
Non-Embedded in FV in ROM

Adding/Replacing DXE Driver
Modified DriverOrder / Driver##### EFI variables

Replacing Windows Boot Manager
EFI System Partition (ESP) on Fixed Drive
ESP\EFI\Microsoft\Boot\bootmgfw.efi

Replacing Fallback Boot Loader
ESP\EFI\Boot\bootx64.efi

Adding New Boot Loader (bootkit.efi)
Modified BootOrder / Boot##### EFI variables
EFI_RUNTIME_SERVICES and HAL

EFI_SYSTEM_TABLE

Pointers

EFI_RUNTIME_SERVICES
EFI_BOOT_SERVICES
EFI_DXE_SERVICES

Module: hal.dll

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>HalpsEFIRuntimeActive</td>
<td>FFFFF00476329E0</td>
</tr>
<tr>
<td>Hal EFI Runtime Services Block</td>
<td>FFFFF00476690C0</td>
</tr>
<tr>
<td>Hal EFI Bugcheck Callback Next Runtime Service Index</td>
<td>FFFFF004766910B</td>
</tr>
<tr>
<td>Hal EFI Runtime Services Table</td>
<td>FFFFF0047669118</td>
</tr>
<tr>
<td>Hal EFI Runtime Callback Record</td>
<td>FFFFF004766BC58</td>
</tr>
</tbody>
</table>
Firmware Rootkit

- **Stage 1:**
  - ✓ Client-Side Exploit drop installer (1)
  - ✓ Installer Elevate Privileges to System

- **Stage 2:**
  - ✓ Bypass Code Signing Policies
  - ✓ Install Kernel-Mode Payload (2)

- **Stage 3:**
  - ✓ Execute SMM exploit
  - ✓ Elevate Privileges to SMM
  - ✓ Execute Payload (3)

- **Stage 4:**
  - ✓ Bypass Flash Write Protection
  - ✓ Install Rootkit into Firmware
Module: BIOS Interface Lock (including Top Swap Mode)

BiosInterfaceLockDown (BILD) control = 1
BIOS Top Swap mode is disabled (TSS = 0)
RTC TopSwap control (TS) = 0
PASSED: BIOS Interface is locked (including Top Swap Mode)

running module: chipsec.modules.common.bios_wp
Module path: c:\Chipsec\chipsec\modules\common\bios_wp.pyc

Module: BIOS Region Write Protection

BC = 0x08 << BIOS Control (b:d.f 00:31.0 + 0xDC)
[00] BIOSWE = 0 << BIOS Write Enable
[01] BLE = 0 << BIOS Lock Enable
[02] SRC = 2 << SPI Read Configuration
[04] TSS = 0 << Top Swap Status
[05] SMM_BWP = 0 << SMM BIOS Write Protection

BIOS region write protection is disabled!

BIOS Region: Base = 0x00A00000, Limit = 0x00FFFFFF
SPI Protected Ranges

<table>
<thead>
<tr>
<th>PRx (offset)</th>
<th>Value</th>
<th>Base</th>
<th>Limit</th>
<th>WP?</th>
<th>RP?</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR0 (74)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR1 (78)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR2 (7C)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR3 (80)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR4 (84)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

None of the SPI protected ranges write-protect BIOS region
BIOS Region Write Protection

BC = 0xA2 << BIOS Control (b:d.f 00:31.0 + 0xDC)

- [00] BIOSWE = 0 << BIOS Write Enable
- [01] BLE = 1 << BIOS Lock Enable
- [02] SRC = 0 << SPI Read Configuration
- [04] TSS = 0 << Top Swap Status
- [05] SMM_BWP = 1 << SMM BIOS Write Protection

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

BIOS Region: Base = 0x00000000, Limit = 0x00000FFF

SPI Protected Ranges

<table>
<thead>
<tr>
<th>PRx (offset)</th>
<th>Value</th>
<th>Base</th>
<th>Limit</th>
<th>WP?</th>
<th>RP?</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR0 (74)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR1 (78)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR2 (7C)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR3 (80)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PR4 (84)</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

None of the SPI protected ranges write-protect BIOS region

PASSED: BIOS is write protected
Firmware Rootkit

Stage 1:
- Client-Side Exploit drop installer (1)
- Installer Elevate Privileges to System

Stage 2:
- Bypass Code Signing Policies
- Install Kernel-Mode Payload (2)

Stage 3:
- Execute SMM exploit
- Elevate Privileges to SMM
- Execute Payload (3)

Stage 4:
- Bypass Flash Write Protection
- Install Rootkit into Firmware

Madness, as you know, is a lot like gravity, all it takes is a little push.
Recent BIOS Vulns For Persistent Infection

- **SMI Handlers (always an issue)** - memory corruption vulnerabilities can lead arbitrary SMM code execution

- **S3BootScript (VU #976132)** - arbitrary modification of platform firmware. Allows attacker arbitrarily read/write to the SMRAM region

- **ThinkPwn (LEN-8324)** - arbitrary SMM code execution exploit for multiple BIOS vendors. Allows attacker to disable flash write protection and modify platform firmware
  - https://github.com/Cr4sh/ThinkPwn

- **Aptiocalypsis (INTEL-SA-00057)** - arbitrary SMM code execution exploit for AMI Aptio based firmware. Allows attacker to disable flash write protection and modify platform firmware
  - https://github.com/Cr4sh/Aptiocalypsis
BIOS Rootkits In-The-Wild
HakingTeam Vector-EDK
Hacking Team UEFI Implant

➢ First* discovery of non-PoC UEFI Malware

➢ Persistent copy of malicious agent inside BIOS

http://www.intelsecurity.com/advanced-threat-research/content/data/HT-UEFI-rootkit.html
Hacking Team UEFI Implant: Modules

- **rkloader**
  - DXE module
  - Bootkit trigger

- **fsbg**
  - UEFI application
  - Main bootkit functionality

- **ntfs**
  - DXE module
  - NTFS driver

Diagram showing the relationships between the modules.
Hacking Team UEFI Implant: How It Works

- **RkLoader** is executed at DXE phase by Firmware
  - Load and execute main bootkit module *fsbg*

- Application *fsbg* is executed
  - Initialize NTFS protocol by loading NTFS driver

- Drop malware onto NTFS volume
  - Application *fsbg* installs malware onto NTFS volume
Hacking Team UEFI Implant: How It Works

```c
EFI_STATUS
EFI_API
_ModuleEntryPoint(
    IN EFI_HANDLE ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
)
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loader.\n"));
    InitializeLib(ImageHandle, SystemTable);

    gReceived = FALSE;  // reset event!

    //CpuBreakpoint();

    // wait for EFI EVENT GROUP READY TO BOOT
    gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);

    return EFI_SUCCESS;
}
```
EFI_GUID LAUNCH_APP =
{
  0xea9a9c,
  0xc9c1,
  0x66e2,
  { 0x96, 0x52, 0x43, 0x2a, 0xd2, 0x5a, 0x9b, 0x6b }
};

NewFilePathProtocol = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength);
NewFilePathProtocol->Type = 0x04;
NewFilePathProtocol->SubType = 0x06;
NewFilePathProtocol->Length[0] = 0x14;
NewFilePathProtocol->Length[1] = 0x00;
gBootServices->CopyMem(((CHAR8 *))(NewFilePathProtocol) + 4), &LAUNCH_APP, sizeof(EFI_GUID));

NewDevicePathEnd = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength + sizeof(EIFI_GUID) + 4);

NewDevicePathEnd->Type = 0x7f;
NewDevicePathEnd->SubType = 0xff;
NewDevicePathEnd->Length[0] = 0x04;
NewDevicePathEnd->Length[1] = 0x00;
Status = gBootServices->loadImage(FALSE, gImageHandle, NewDevicePathProtocol, NULL, 0, &ImageLoadedHandle);

EFI_STATUS EFIAPI
_ModuleEntryPoint (IN EFI_HANDLE ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
)
{
  EFI_EVENT Event;

  DEBUG((EFI_D_INFO, "Running RK loader.\n"));
  InitializeLib(ImageHandle, SystemTable);
  gReceived = FALSE; // reset event

  // wait for EFI_EVENT GROUP READY TO BOOT
  gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);
  return EFI_SUCCESS;
}
Hacking Team UEFI Implant: How It Works

```c
#define FILE_NAME_SCOUT L"\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"
#define FILE_NAME_SOLDIER L"\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup"
#define FILE_NAME_ELITE L"\AppData\Local"
#define DIR_NAME_ELITE L"\AppData\Local\Microsoft"

// (20 * (6+5+2)) + 1) unicode characters from EFI FAT spec (doubled for bytes)
#define MAX_FILE_NAME_LEN 512
#define FIND_XXXXX_FILE_BUFFER_SIZE (SIZE_OF_EFI_FILE_INFO + MAX_FILE_NAME_LEN)
#define CALC_OFFSET(type, base, offset) (type)((UINTN)base + (UINT32)offset)

#ifdef FORCE_DEBUG
UINT16 g_NAME_SCOUT[] = L"scoute.exe";
UINT16 g_NAME_SOLDIER[] = L"soldier.exe";
UINT16 g_NAME_ELITE[] = L"elite";
#else
//32 byte per inserire 16 caratteri unicode
UINT16 g_NAME_SCOUT[] = L"6To_60S7K_FU06yjEhjh5dpFw96549UU";
UINT16 g_NAME_SOLDIER[] = L"kdfas7835jfw09j29FKFLDOR3r35fJR";
UINT16 g_NAME_ELITE[] = L"eorpekf3904kLDKQ0023iosdn93smMXK";
#endif
```
Hacking Team UEFI Implant: How It Works

- https://github.com/hackedteam/soldier-win
- https://github.com/hackedteam/scout-win
- https://github.com/hackedteam/core-win64
- https://github.com/hackedteam/core-win32
- https://github.com/hackedteam/vector-edk
Hacking Team : Results

How can I deploy the Agent?

• Via SPI programmer circuit (physical access to motherboard);

• Via Service Mode (recovery device);

• Via firmware upgrade (actually SecureFlash limitation to bypass);

• Via exploitation of firmware vulnerability
<table>
<thead>
<tr>
<th>Name</th>
<th>At Type</th>
<th>Subtype</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>03C1F5C8-48F1-416E-A6B6-992D3FBACA6</td>
<td>File</td>
<td>DXE driver</td>
<td>AD15mmServiceBody</td>
</tr>
<tr>
<td>4AF43F1CA-604F-493A-99BE-1E0E72A0767</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13C1</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13D1</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13C3</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13D3</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13C4</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A883-76DBBE1E13D4</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A884-77DBBE1E13C6</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A884-77DBBE1E13C8</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>37946B52-4C48-46AF-A884-77DBBE1E13C9</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>CC243581-112F-441C-8150-60DB3696919</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>4AC4C731-7C53-4DC1-86FA-A2425260A69A</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>F306F4D0-2D95-465D-9410-8355E51A2D88</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>C9963F83-F593-4C82-926C-C310F44238</td>
<td>File</td>
<td>DXE driver</td>
<td>MenTest</td>
</tr>
<tr>
<td>4262A82-4C8F-4D9A-94CE-82ED69AFC93</td>
<td>File</td>
<td>DXE driver</td>
<td>MemoryDiagnosticBios</td>
</tr>
<tr>
<td>A91CC287-4871-41EB-AE92-69C9CBBEBB8</td>
<td>File</td>
<td>DXE driver</td>
<td>HddDiagnostic</td>
</tr>
<tr>
<td>F780E92D-A847-441D-8BDE-41E529EB5A70</td>
<td>File</td>
<td>DXE driver</td>
<td>UnlockPsud</td>
</tr>
<tr>
<td>466C4F97-2C53-4513-99E7-5A6739FC31C1</td>
<td>File</td>
<td>DXE driver</td>
<td>VGAInformation</td>
</tr>
<tr>
<td>88647F11-1A15-48C8-AB07-33E4D852086</td>
<td>File</td>
<td>DXE driver</td>
<td>A01mmMmShandler</td>
</tr>
<tr>
<td>C74233C1-96FD-4CB3-9453-55C0D77C3EC8</td>
<td>File</td>
<td>DXE driver</td>
<td>WMMmMmShandler</td>
</tr>
<tr>
<td>FS824B49-2F4D-4DE9-86AE-0D484D74A4C</td>
<td>File</td>
<td>DXE driver</td>
<td>Ntfs</td>
</tr>
<tr>
<td>4F3025BA9-2F4D-4D09-861E-0D484D74A4C</td>
<td>File</td>
<td>DXE driver</td>
<td>rklloader</td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume free space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
<td></td>
<td>Non-empty</td>
</tr>
</tbody>
</table>

Messages
parseBios: one of volumes inside overlaps the end of data
parseBios: one of volumes inside overlaps the end of data
parseVolume: unknown file system FFF1288D-7696-4C8F-8985-2747075B4F50
<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>245 37946B52-EC4B-46AF-A883-76DBBE1E13D4</td>
<td>67,166</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>246 37946B52-EC4B-46AF-A884-77DBBE1E13C6</td>
<td>38,568</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>247 37946B52-EC4B-46AF-A884-77DBBE1E13C8</td>
<td>32,892</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>248 37946B52-EC4B-46AF-A884-77DBBE1E13C9</td>
<td>31,011</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>249 D2DRecovery</td>
<td>14,230</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>250 4CAC7381-7C53-4DC1-86FA-42A15260409A</td>
<td>1,107</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>251 F306F460-2DC9-4B5D-9410-83585F1ADD80</td>
<td>1,189</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>252 MemTest</td>
<td>40,700</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>253 MemoryDiagnosticBios</td>
<td>7,585</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>254 HddDiagnostic</td>
<td>27,211</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>255 UnlockPswd</td>
<td>27,196</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>256 VGLnformation</td>
<td>16,587</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>257 A01WMISmmHandler</td>
<td>87,950</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>258 WM00WMISmmHandler</td>
<td>34,182</td>
<td>7/6/2015 3:10:02 PM</td>
</tr>
<tr>
<td>259 Volume free space</td>
<td>6,943,865</td>
<td>7/6/2015 3:10:03 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>12,582,840</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>72</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>286</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>12,582,924</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>4</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>121</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>1,665,728</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>234</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>218</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>1 Volume free space</td>
<td>3,511,578</td>
<td>7/6/2015 3:10:03 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>5,177,272</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>72</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>301</td>
<td>7/6/2015 3:09:57 PM</td>
</tr>
<tr>
<td>2 Padding</td>
<td>127,054</td>
<td>7/6/2015 3:10:03 PM</td>
</tr>
<tr>
<td>3 FFF12B8D-7696-4CB8-A985-27A070568450</td>
<td>184,619</td>
<td>7/6/2015 3:10:03 PM</td>
</tr>
<tr>
<td>4 7A9354D9-0468-444A-81CE-08F617DB90DF</td>
<td>2,085,496</td>
<td>7/6/2015 3:10:03 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>12,582,924</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>72</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>286</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>1,665,728</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>234</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>218</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>1 Volume free space</td>
<td>3,045,738</td>
<td>7/6/2015 3:11:02 PM</td>
</tr>
<tr>
<td>body.bin</td>
<td>5,177,272</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>header.bin</td>
<td>72</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>info.txt</td>
<td>301</td>
<td>7/6/2015 3:10:56 PM</td>
</tr>
<tr>
<td>2 Padding</td>
<td>127,054</td>
<td>7/6/2015 3:11:02 PM</td>
</tr>
<tr>
<td>3 FFF12B8D-7696-4CB8-A985-27A070568450</td>
<td>184,619</td>
<td>7/6/2015 3:11:02 PM</td>
</tr>
<tr>
<td>4 7A9354D9-0468-444A-81CE-08F617DB90DF</td>
<td>2,085,496</td>
<td>7/6/2015 3:11:02 PM</td>
</tr>
</tbody>
</table>
DEITYBOUNCE
➢ Only Snowden-leaked documentation is available for analysis

➢ Safe to assume that servers use legacy BIOS¹

DEITYBOUNCE Workflow

ARKSTREAM

Computer Firmware (BIOS)

DEITYBOUNCE

SMRAM

Malicious SMI handler

System Initialization

Execute OS bootloader

Patch OS kernel/
Inject malicious module

OS Kernel is initialized
BANANABALLOT and JETPLOW (Equation Group)
(TS/SI/REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant. JETPLOW also has a persistent back-door capability.

Command, Control, and Data Exfiltration using
DNT Implant Communications Protocols (Typical)

Target Network

(TS/SI/REL) JETPLOW Persistence Implant Concept of Operations

(TS/SI/REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT’s BANANAGLEE software implant and modifies the Cisco firewall’s operating system (OS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE’s communications structure, so that full access can be reacquired at a later time. JETPLOW works on Cisco’s 500-series PIX firewalls, as well as most ASA firewalls (5505, 5510, 5520, 5540, 5550).

(TS/SI/REL) A typical JETPLOW deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. JETPLOW is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

Status: (C/I/REL) Released. Has been widely deployed. Current availability restricted based on OS version (require for details).

Unit Cost: $0

POC: 532222, @nsa.ic.gov

Derived From: NSA/CSSR 1-52
Date of: 20200126
Declassify On: 20320108

TOP SECRET/COMINT/REL TO USA, FVEY
<Interface>

<menuItem>
    <itemText>Read B865 Memory</itemText>
    <querylist>
        <query>Enter address:</query>
        <query>Enter number of bytes to read:</query>
    </querylist>
</menuItem>

<menuItem>
    <itemText>Write a file to B865 memory</itemText>
    <querylist>
        <query>Address to write data:</query>
        <query>Enter filename of binary data to write:</query>
    </querylist>
</menuItem>

</Interface>
if ( !isPixOS((NET + 4)) )
    return 1;
if ( bfl_fetchOsUns(NET + 8, "BiosClassAddr", &temp1) )
{
    fwrite("Bios Class Address information could not be read\n", 1, 49, stdout);
    fwrite("You will not be able to read or Write to Bios\n", 1, 46, stdout);
    a1[6] = 0;
    result = 0;
}
else
{
    u2 = NET;
    u3 = (NET + 4) < 0x7000u;
    u4 = (NET + 4) == 1792;
    a1[6] = temp1;
    if ( u3 || u4 )
    {
        if ( bfl_fetchOsUns(u2 + 8, "BiosReadE28F6", &temp1) )
        {
            u5 = stdout;
            fwrite("Bios Read Address information could not be read\n", 1, 48, stdout);
        }
        else
        {
            fwrite("You will not be able to read or Write to Bios\n", 1, 46, u5);
            a1[9] = 0;
            result = 0;
            a1[8] = 0;
            return result;
        }
    }
    a1[8] = temp1;
    if ( bfl_fetchOsUns(NET + 8, "BiosLockE28F6", &temp1) )
    {
        u5 = stdout;
        fwrite("Bios Lock Address information could not be read\n", 1, 48, stdout);
        goto LABEL_7;
    }
    a1[9] = temp1;
    if ( bfl_fetchOsUns(NET + 8, "BiosWriteAddr5", &temp1) )
    {
        u5 = stdout;
        fwrite("Bios Write Address information could not be read\n", 1, 49, stdout);
        goto LABEL_7;
    }
    a1[7] = temp1;
    return 1;
}
if( !ispixOS(x(NET + 4)) )
    return 1;
if( ( bfl_fetchOsUns(NET + 8, "BiosClassAddr", &temp1) )
{}
    fwrite("Bios Class Address information could not be read\n", 1, 49, stdout);
    fwrite("You will not be able to read or Write to Bios\n", 1, 46, stdout);
    a1[6] = 0;
    result = 0;
}
else
{
    u2 = NET;
    u3 = x(NET + 4) < 0x7000;
    u4 = x(NET + 4) == 1792;
    ...
    .got_loader:00000000 ; Source File : 'checksum_bios.c'
    .got_loader:00000000 ; Source File : 'entryPoint.c'
    .got_loader:00000000 ; Source File : 'pageTable.c'
    .got_loader:00000000 ; Source File : 'coreBiosModule.c'
    .got_loader:00000000 ; Source File : 'determineBios.c'
    .got_loader:00000000 ; Source File : 'writeSpeedPlow.c'
    .got_loader:00000000 ; Source File : 'asaBios.c'
    .got_loader:00000000 ; Source File : 'cmos.c'
    .got_loader:00000000 ; Source File : 'Components/Modules/BiosModule/Implant/ASABIOS/../asaBios_asm.$'
    .got_loader:00000000 ; Source File : 'checksum_uint32.c'
    .got_loader:00000000 ; Source File : 'byteOrdering.c'
    .got_loader:00000000 ; Source File : 'osVersionChecking.c'
    .got_loader:00000000 ; Source File : 'free_stub.c'
    u5 = &stdout;
    fwrite("Bios Lock Address information could not be read\n", 1, 48, stdout);
    goto LABEL_7;
}
    a1[9] = temp1;
    if( ( bfl_fetchOsUns(NET + 8, "BiosWriteAddr5", &temp1) )
{
    u5 = &stdout;
    fwrite("Bios Write Address information could not be read\n", 1, 49, stdout);
    goto LABEL_7;
}
    a1[7] = temp1;
    return 1;
}
Don’t miss the talk:

**JETPLOW is dead. Long live the JETPLOW!**

by Roman Bazhin and Maxim Malyutin

Main Track at 5pm

https://dsec.ru/ipm-research-center/research/architecture_jetplow/
Computrace/LoJack
Computrace/LoJack

- Legitimate application that provides anti-theft protection.

- Implements rootkit functionality to “persist” on the system.

- Contains UEFI BIOS components to perform its activities.
### Computrace Module Activation

<table>
<thead>
<tr>
<th>Current Setting</th>
<th>Current State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Disabled]</td>
<td>Not Activated</td>
</tr>
</tbody>
</table>

#### Item Specific Help

Enables or disables the BIOS interface to activate Computrace module. Computrace is an optional monitoring service from Absolute Software.

- [Enabled] Enables the Computrace activation.
- [Disabled] Disables the Computrace activation.
- [Permanently Disabled] Permanently disables the Computrace.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Subtype</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>8FEEECF1-8CFO-9A79-9231-48015666..</td>
<td>File</td>
<td>Application</td>
<td>AbsoluteComputraceInstaller</td>
</tr>
<tr>
<td>PE32 image section</td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4EFC51DA-23A6-4790-A292-4995C7F6..</td>
<td>File</td>
<td>DXE driver</td>
<td>LenovoComputraceEnablerDxe</td>
</tr>
<tr>
<td>PE32 image section</td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4580C8F3-03F0-4998-G66F-26343C6..</td>
<td>File</td>
<td>DXE driver</td>
<td>LenovoComputraceLoaderDxe</td>
</tr>
<tr>
<td>PE32 image section</td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18570E75-0D73-4203-9ED2-8768A87..</td>
<td>File</td>
<td>SMM module</td>
<td>LenovoComputraceSmiServices</td>
</tr>
<tr>
<td>PE32 image section</td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4CD188-5F7B-4636-A079-16F7F96..</td>
<td>File</td>
<td>SMM module</td>
<td>LenovoSecuritySmdDispatch</td>
</tr>
<tr>
<td>621D6C8-0F5E-4E31-AE02-60DE769..</td>
<td>File</td>
<td>DXE driver</td>
<td>LenovoRemoteConfigUpdateDxe</td>
</tr>
</tbody>
</table>
Computrace/LoJack

UEFI Environment

Computrace Configuration & Activation

LenovoComputraceLoaderDxe
LenovoComputraceEnablerDxe

AbsoluteComputraceInstallerDxe
LenovoComputraceSmiServices

OS Environment

Install Computrace Agent

Computrace Agent

OS Process
OS Process
OS Process
...

Network Interface

OS NTFS Volume

Computrace C&C Servers
ComputraceAgentInstaller

- Implements custom NTFS parsing functionality to install the “agent” onto NTFS volumes

```c
struct install_agent_on_system_volume()
{
    v0 = 0;
    get_block_protocol(0, &v3);
    b1d_str_cat_2(&v2, &aMulti0Disk0Rdi[35], "\System32"); // WINDOWS\System32
    do
        result = find_file_volume(0i64, --v0, &v2, store_image_in_file);
        while ( v0 > -v3 );
    aMulti0Disk0Rdi[33] = 0;
    return result;
}

__int64 InstallComputraceAgent()
{
    install_agent_on_system_volume();
    find_file_volume(0i64, 0i64, "\BOOT.INI", install_agent_on_volume);
    find_file_volume(0i64, 0i64, "\MSDOS.SYS", install_agent_on_volume);
    return 0i64;
}
```
Installs custom ACPI Configuration Table

BootServices->InstallConfigurationTable

EFI_ACPI_TABLE_PROTOCOL->InstallAcpiTable
ComputraceAgentInstaller

- Installs custom ACPI Configuration Table

BootServices->InstallConfigurationTable

EFI_ACPI_TABLE_PROTOCOL->InstallAcpiTable

Agent Image

Original ACPI RSDT Table

Original ACPI XSDT Table

New ACPI WBPT Table

Agent UEFI Shell Arguments
BIOS Update Issues
BIOS Update Validation Routine

Dell Client BIOS: Signed Firmware Update: [http://en.community.dell.com/techcenter/extras/m/white_papers/20287278/](http://en.community.dell.com/techcenter/extras/m/white_papers/20287278/)
BIOS Update Rollback Protection

Dell Client BIOS: Signed Firmware Update: [http://en.community.dell.com/techcenter/extras/m/white_papers/20287278/](http://en.community.dell.com/techcenter/extras/m/white_papers/20287278/)
Lenovo BIOS Update on MS Win10 with Device Guard
Forensic Approaches
Firmware Forensics with CHIPSEC

Live system firmware analysis

chipsec_util spi info
chipsec_util spi dump rom.bin
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util uefi var-list
chipsec_util uefi var-read db

D719B2CB-3D3A-4596-A3BC-DAD00E67656F db.bin

Offline system firmware analysis

chipsec_util uefi keys PK.bin
chipsec_util uefi nvram vss bios.bin
chipsec_util uefi decode rom.bin
chipsec_util decode rom.bin

https://github.com/chipsec/chipsec
Firmware Forensics with CHIPSEC


```
{
    "HT_rkloader" : { "guid": "F50248A9-2F4D-4DE9-86AE-BDA84D07A41C" },
    "HT_rkloader_name" : { "name": "rkloader" },
    "HT_Ntfs" : { "guid": "F50258A9-2F4D-4DA9-861E-BDA84D07A44C" },
    "HT_Ntfs_name" : { "name": "Ntfs" },
    "HT_app" : { "guid": "EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B" },

    "ThinkPwn_SmmRuntimeProtGuid" : { "regexp": "\\xA1\\x97\\x68\\xA5 ...\\x9A" },
    "ThinkPwn_SystemSmmRuntimeRt_name" : { "name": "SystemSmmRuntimeRt.efi" },
    "ThinkPwn_SystemSmmRuntimeRt" : { "guid": "7C79AC8C-5E6C-4E3D-BA6F-C260EE7C172E" },
    "ThinkPwn_SmmRuntime_name" : { "name": "SmmRuntime" },
    "ThinkPwn_SmmRuntime" : { "guid": "A56897A1-A77F-4600-84DB-22B0A801FA9A" }
}
```


```
chipsec_main.py -i -m tools.uefi.blacklist [-a <fw_image>,<blacklist>]
chipsec_main.py -i --no_driver -m tools.uefi.blacklist -a uefi.rom,blacklist.json
```

https://github.com/chipsec/chipsec
SMM Memory Forensics

- Diff original and infected SMRAM to hunt for malicious SMI
  - Parse `EFI_SMM_SYSTEM_TABLE` to get SMM memory layout
  - Find all SMI handlers based on `DATABASE_RECORD` signature

- Also another useful information can be extracted from SMRAM:
  - List of SMM drivers
  - List of SMM Protocols

- Kudos to Cr4sh for SMRAM forensic parser
  
  https://github.com/Cr4sh/smram_parse/blob/master/smram_parse.py
How to dump SPI Flash?
SPI Flash Dump – Reading from OS

- SPI Controller
  - Get SPI Base Address Register (refer to ICH/PCH documentation) -- SPIBAR
  [CHIPSEC] reading 0x100000 bytes from SPI Flash starting at FLA = 0x700000
  [CHIPSEC] it may take a few minutes (use DEBUG or VERBOSE logger options to see progress)
  ERROR: HSFS.FDU is 0, hardware sequencing is disabled
  - SPIBAR + 0x04: HSFS - Status Register
  - SPIBAR + 0x06: HSFC - Control Register
  - SPIBAR + 0x08: FADDR - Address Register
  - SPIBAR + 0x10: FDATAX - Data Registers
SPI Flash Dump – Reading from OS

• SPI Controller
  • Get SPI Base Address Register (refer to ICH/PCH documentation) -- SPIBAR

• Memory-mapped SPI Registers
  • SPIBAR + 0x04: HSFS - Status Register
  • SPIBAR + 0x06: HSFC - Control Register
  • SPIBAR + 0x08: FADDR - Address Register
  • SPIBAR + 0x10: FDATAx - Data Registers
SPI Flash Dump - Reading from OS

**Reader**
- Write start address to FADDR
- Write size of data to read to HSFC
- Write read command to HSFC
- Set FGO (0x0001) bit in HSFC
- Wait for SPI read cycle completion
- Read data from FDATAx registers

**SPI Controller**
- **FADDR**: Flash Linear Address
- **HSFC**:
  - FDBC
  - FCYCLE
  - FGO
- **HSFS**:
  - ... (other registers)
- **FDATAx**: DATA
SPI Flash Dump – Attacker’s Possibilities

Flash SPI SMI# Enable (FSMIE) — R/W. When set to 1, the SPI asserts an SMI# request whenever the Flash Cycle Done (FDONE) bit is 1.
SPI Flash Dump – Attacker’s Possibilities

Reader
- Write start address to FADDR
- Write size of data to read to HSFC
- Write read command to HSFC
- Set FGO (0x0001) bit in HSFC
- Wait for SPI read cycle completion
- Read data from FDATAX registers

SPI Controller

Attacker
- Set FSMIE bit to 1 in HSFC
- Once FDONE is set to 1 SMI is triggered
- Write fake data to FDATAX registers
How to dump BIOS firmware directly from chip?
How to dump BIOS firmware directly from chip?
How Debug UEFI Firmware?

How Debug UEFI Firmware?

Intel Virtual Platform

- Perfect simulation of hardware
- Boot after power on, sleep and hibernate
- Dump SMRAM, memory map and other parameters
- Disassembling
- Dynamic check of accesses out of allowable memory regions and SMRAM call-outs
Minnowboard Max

http://wiki.minnowboard.org/
“If you’re good at something, never do it for free.” - Joker
Intel XDP Hardware Debuggers
SMM Debug with Intel System Debugger

How to enter SMM

Debugger Commands

SPECIAL BREAK 0 ON "SMM Entry Break" : enabled (S=0,C=S=0)
SPECIAL BREAK 1 ON "SMM Exit Break" : enabled (S=0,C=S=0)
INFO: Resetting target, this may take a moment...
exection stopped by "Halt Command break"
>xdb > IA32CPU "read msr 0x9e"
ERROR: Couldn't read MSR 0x9e: The CPU faulted when accessing an MSR.
xdb > SET PORT 0x82 = 1
WARNING: Multiple breaks, context is set to the most interesting.
program stopped: SPECIAL BREAK 'SMM Entry Break' (ID=0) at "0x0900:0x00000000"
A few words about UEFI Firmware Mitigations
Intel Boot Guard Technology

- Intel Boot Guard – authenticated code module ACM-based secure boot (verified boot) that verifies a known and trusted BIOS is booting the platform

- Protect Secure Boot Root of Trust from firmware-based attacks

CPU reset → bootrom → ACM → initial boot block → BIOS → boot loader → OS
Intel Boot Guard Technology

Intel® Boot Guard

Intel® Device Protection Technology with Boot Guard


OEM PI Verification Using PI Signed Firmware Volumes
Vol 3, section 3.2.1.1 of PI 1.3 Specification

OEM UEFI 2.4 Secure Boot
Chapter 27.2 of The UEFI 2.4 Specification

https://firmware.intel.com/sites/default/files/STTS003%20-%20SF15_STTS003_100f.pdf
Intel Boot Guard Technology

http://vzimmer.blogspot.com/2013/09/where-do-i-sign-up.html
Don’t miss the talk:

**Safeguarding Rootkits: Intel BootGuard**
by Aleksandr Ermolov

Main Track at 4pm
Intel BIOS Guard Technology

- BIOS Guard – hardware-assisted authentication and protection against BIOS recovery attacks

- BIOS Guard can reduce SMI handler attack surface because of one signed and authenticated code module (ACM)

- BIOS Guard module is authenticated code module (ACM) executing in internal processor (isolated CPU) before letting the machine boot

- With active BIOS Guard only guarded module is able to modify SPI flash memory

- BIOS Guard can be useful protection from persistent rootkit infection
Intel BIOS Guard Technology

Intel SMI transfer monitor (STM) Technology

- STM - memory isolation approach for SMM
- Based on Intel VT-x and optional TXT support
- In the end, STM becomes very complex solution for SMM protection
- At this moment nobody from known BIOS vendors doesn't use STM approach 😞
## Windows SMM Security Mitigation Table (WSMT)

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte length</th>
<th>Byte offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACPI Standard Header</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td>4</td>
<td>0</td>
<td>Signature for the WSMT</td>
</tr>
<tr>
<td>Length</td>
<td>4</td>
<td>4</td>
<td>Length, in bytes, of the WSMT. Must be 40 for Revision 1.</td>
</tr>
<tr>
<td>Revision</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Checksum</td>
<td>1</td>
<td>9</td>
<td>Entire table, which must sum to zero</td>
</tr>
<tr>
<td>OEMID</td>
<td>6</td>
<td>10</td>
<td>Original equipment manufacturer (OEM) identifier (ID)</td>
</tr>
<tr>
<td>OEM Table ID</td>
<td>8</td>
<td>16</td>
<td>Manufacturer model ID</td>
</tr>
<tr>
<td>OEM Revision</td>
<td>4</td>
<td>24</td>
<td>OEM revision for supplied OEM table ID</td>
</tr>
<tr>
<td>Creator ID</td>
<td>4</td>
<td>28</td>
<td>Vendor ID of the ASL compiler utility that created the table</td>
</tr>
<tr>
<td>Creator Revision</td>
<td>4</td>
<td>32</td>
<td>Revision of the ASL compiler utility that created the table</td>
</tr>
<tr>
<td>Protection Flags</td>
<td>4</td>
<td>36</td>
<td>Container of a bitmask of the system implemented WSMT protections. Bits in this field represent that certain protections/enforcements are enabled and active for firmware executing in SMM context after ExitBootServices(). See Table 2 for a detailed description of this field.</td>
</tr>
</tbody>
</table>

## Windows SMM Security Mitigation Table (WSMT)

<table>
<thead>
<tr>
<th>Length</th>
<th>Bit offset</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1      | 0          | FIXED_COMM_BUFFERS  
If set, expresses that for all synchronous SMM entries, SMM will validate that input and output buffers lie entirely within the expected fixed memory regions. |
| 1      | 1          | COMM_BUFFER_NESTED_PTR_PROTECTION  
If set, expresses that for all synchronous SMM entries, SMM will validate that input and output pointers embedded within the fixed communication buffer only refer to address ranges that lie entirely within the expected fixed memory regions. |
| 1      | 2          | SYSTEM_Resource_PROTECTION  
Firmware setting this bit is an indication that it will not allow reconfiguration of system resources via non-architectural mechanisms. |
| 31:3   |            | Reserved; must return 0 when read. |

Modern Devices
Device Guard: Enabled

Setup feature to support Microsoft(R) Device Guard.

To complete the configuration of Device Guard, Supervisor Password must be set.
<table>
<thead>
<tr>
<th>Secure Boot Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Boot</td>
</tr>
<tr>
<td>Platform Mode</td>
</tr>
<tr>
<td>Secure Boot Mode</td>
</tr>
<tr>
<td>Reset to Setup Mode</td>
</tr>
<tr>
<td>Restore Factory Keys</td>
</tr>
<tr>
<td>Clear All Secure Boot Keys</td>
</tr>
</tbody>
</table>

* Unselectable for Device Guard
Rootkits and Bootkits
Reversing Modern Malware and Next Generation Threats

Alex Matrosov, Eugene Rodionov, and Sergey Bratus

nostarch.com/rootkits
Thank you for your attention!

Alex Matrosov
@matrosov

Eugene Rodionov
@vxradius