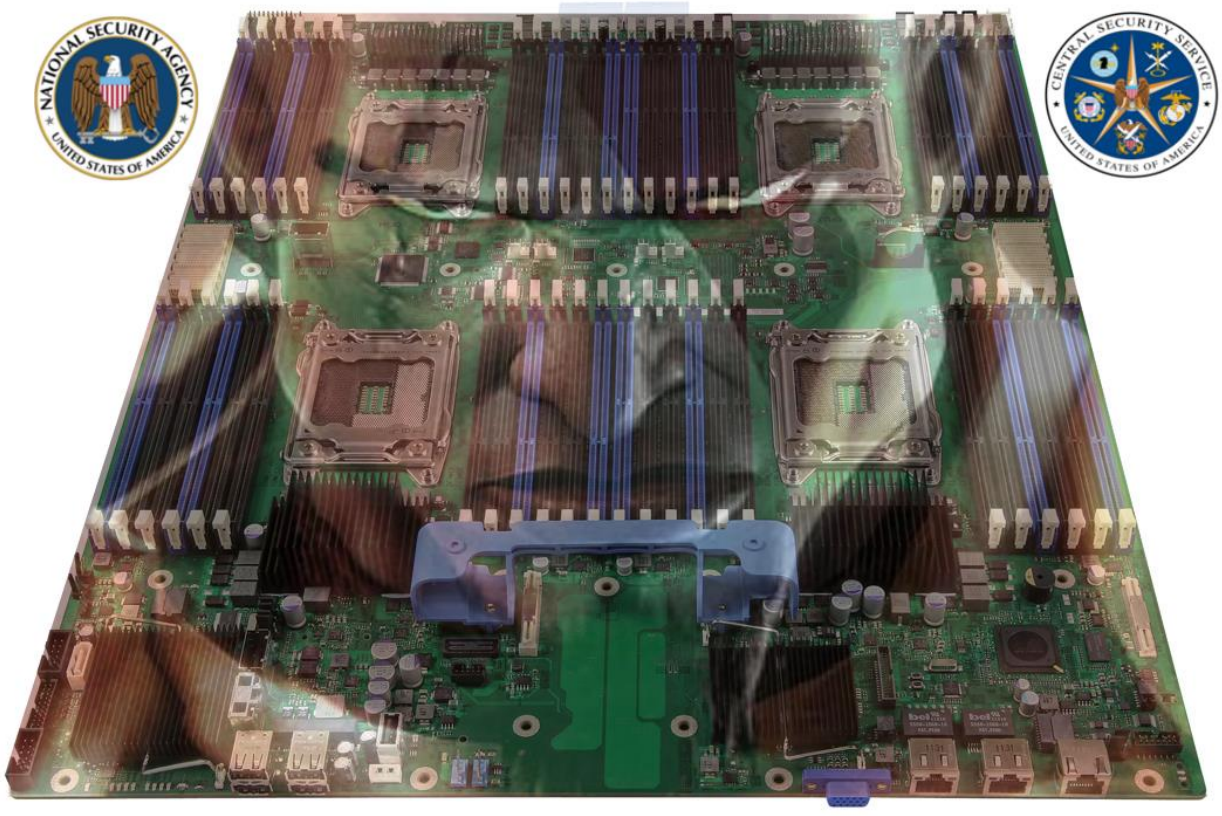
1/22/2014

INFOSEC INSTITUTE

NSA BIOS BACKDOOR A.K.A GOD MODE MALWARE PART 1: DEITYBOUNCE



DEITYBOUNCE | Darmawan Salihun

NSA BIOS Backdoor a.k.a God Mode Malware Part 1: DEITYBOUNCE

This article is the first part of a series on NSA BIOS Backdoor internals. Before we begin, I'd like to point out why these malware are classified as "god mode" malware. First, most of the malware use internal (NSA) codename in the realms of "gods", such as DEITYBOUNCE, GODSURGE, etc. Secondly, these malware have capabilities similar to "god mode" cheat in video games which made the player using it close to being invincible. This is the case with this type of malware because they are very hard to detect and remove even with the most sophisticated anti malware tool during its possible deployment timeframe.

This part of the series focuses on the DEITYBOUNCE malware described in the NSA ANT Server document, leaked by Edward Snowden. The analysis presented in this article is based on technical implications of the information provided by the document. The document lacks in many technical specifics, but based on the BIOS technology at the day DEITYBOUNCE started to become operational, we can infer some technically sound hypothesis—or conclusions, if you prefer :-).

Introduction to DEITYBOUNCE Malware

DEITYBOUNCE operates as part of the system shown in Figure 1. Figure 1 shows several peculiar terms, such as ROC, SNEAKERNET, etc. Some of these terms are internally used by NSA. ROC is an abbreviation for Remote Operation Center. ROC acts as NSA point of control of the target system and it's located outside NSA's headquarter. SNEAKERNET is a fabulous term for physical delivery of data, i.e. using human to move data between computers by moving removable media such as magnetic tape, floppy disks, compact discs, USB flash drives (thumb drives, USB stick), or external hard drives from one computer to another.

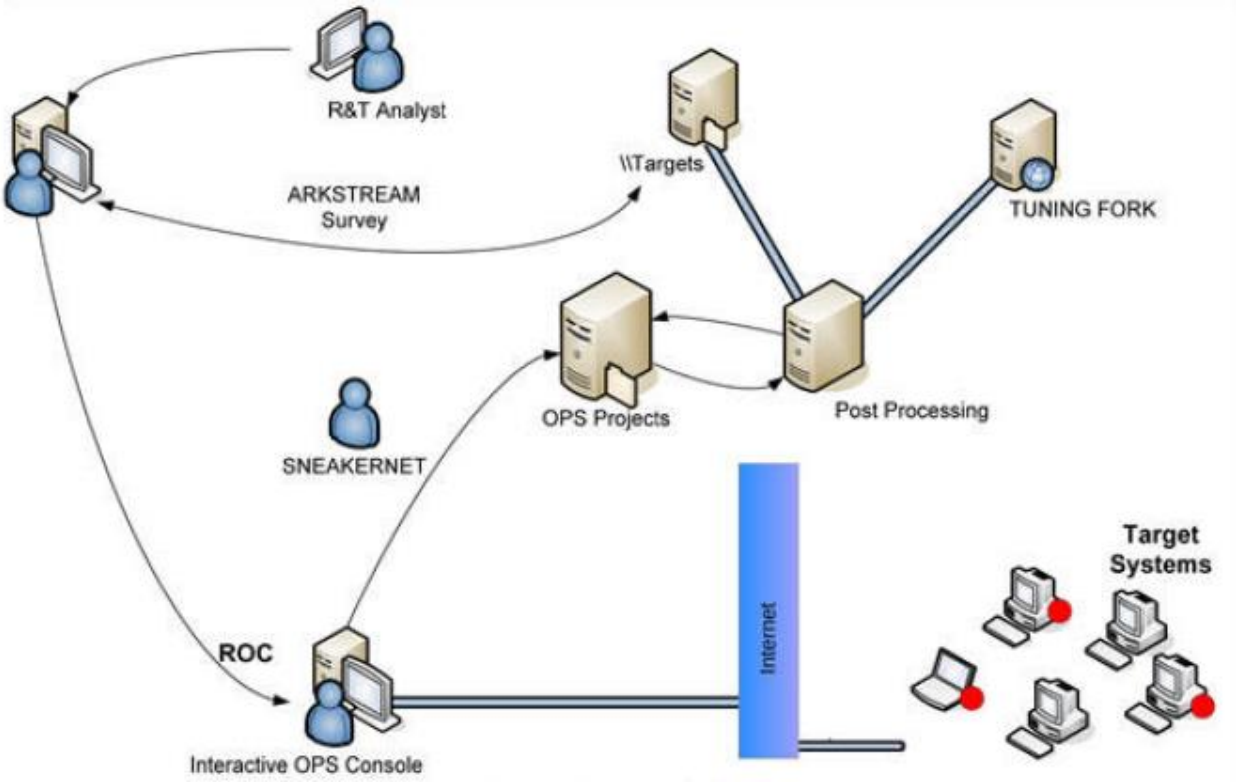


Figure 1 DEITYBOUNCE Extended Concept of Operation

Figure 1 shows DEITYBOUNCE targets the machines marked with red dot. DEITYBOUNCE itself is not installed in those machines because DEITYBOUNCE targets Dell PowerEdge 1850/2850/1950/2950 RAID server series with BIOS versions A02, A05, A06, 1.1.0, 1.2.0, or 1.3.7, not laptops or desktop/workstations. This means DEITYBOUNCE is installed in servers accessed by the laptops or desktop/workstations marked with red dot. The red dot also means that the target systems could act as "jump host" to implant DEITYBOUNCE in the target servers.

Figure 1 also shows the presence of ARKSTREAM. ARKSTREAM is basically a malware dropper which contains BIOS flasher and malware dropper functions. ARKSTREAM can install DEITYBOUNCE to the target server either via exploits controlled remotely (network infection) or via USB thumb drive infection. This infection method in a way is very similar to the STUXNET malware dropper. ARKSTREAM installs DEITYBOUNCE via BIOS flashing, i.e. replacing the PowerEdge server BIOS with the one that is "infected" by DEITYBOUNCE malware.

The NSA ANT server document doesn't divulge minute details explaining DEITYBOUNCE "technical context" of operation. However, we can infer some parts of the "technical context" from DEITYBOUNCE technical requirements mentioned in the document. These are the important technical details revealed by the NSA ANT server document:

1. DEITYBOUNCE provides software application persistence on Dell PowerEdge servers by exploiting the motherboard BIOS and utilizing System Management Mode (SMM) to gain periodic executions while the operating system (OS) loads.

2. DEITYBOUNCE supports multiprocessor systems with RAID hardware and Microsoft Windows 2000, XP and 2003 Server.
3. Once implanted, DEITYBOUNCE's frequency of execution (dropping the payload) is configurable and will occur when the target machine powers on.

In later sections we will look into DEITYBOUNCE malware architecture in more details based on these technical details. These details provide a lot more valuable hints than you think :-). But before that, you need to know some important background knowledge.

A Closer Look at Dell Power-Edge Hardware

Now, let's start with the first background knowledge. In the previous section, we learned that DEITYBOUNCE targets Dell PowerEdge 1850/2850/1950/2950 RAID server series. Therefore, we need to look into the servers more closely to understand DEITYBOUNCE execution environment. One can download the relevant server specifications in these links:

1. Dell PowerEdge 1950 specification: http://www.dell.com/downloads/global/products/pedge/en/1950_specs.pdf
2. Dell PowerEdge 2950 specification: http://www.dell.com/downloads/global/products/pedge/en/2950_specs.pdf

The server specifications are rather ordinary. However, if you look closer to the storage options of both server types, you'll notice the option to use RAID controller in both server types. The RAID controller is of the type PERC 5/i, PERC4e/DC, or PERC 5/e. We focus on the RAID controller hardware because DEITYBOUNCE technical details mentioned the presence of RAID as one of its hardware "requirement".

Now, let's move into more detailed analysis. You can download Dell PERC family of RAID controller user guide from: ftp://ftp.dell.com/Manuals/Common/dell-perc-4-dc_User's%20Guide_en-us.pdf. Despite the document is only a user guide, it provides important information as follows:

1. PERC stands for PowerEdge Expandable RAID Controller. This means the PERC series of RAID controller are either white-labeled by Dell or developed internally by Dell.
2. There several types of PERC RAID controller. The ones with XX/i moniker is the integrated version, those with XX/SC moniker means the RAID controller is single channel, those with XX/DC moniker means the RAID controller is Dual Channel, and those with XXe/XX moniker signifies that the RAID controller uses PCI Express (PCIe) instead of PCI bus—those without the last moniker implies that the RAID controller uses PCI, not PCIe.
3. All PERC variants have 1MB of onboard flash ROM. Mind you, this is **not** the PowerEdge server motherboard flash ROM but the PERC RAID controller (exclusive) flash ROM. It's used to initialize and configure the RAID controller.
4. All PERC variants have NVRAM to store their configuration data. The NVRAM is located in the PERC adapter board except when the PERC is integrated into the motherboard.

The PERC RAID controller flash ROM size (1MB) is huge from firmware code point of view. Therefore, anyone can insert advanced—read: large in code -size—malicious firmware-level module into it.

I can't find information on the Dell PowerEdge 1850/2850/1950/2950 BIOS chip size. However, the size of their compressed BIOS files is larger than 570KB. Therefore, it's safe to assume that the motherboards BIOS chip size are at least 1MB, because AFAIK, there is no flash ROM chip—used to store BIOS code—that has size between 570KB and 1MB. The closest flash ROM size to 570KB is 1MB. This fact also present a huge opportunity to place BIOS malware into the motherboard BIOS besides to the RAID controller expansion ROM.

Initial Program Loader (IPL) Device Primer

The second background knowledge you need to know is regarding IPL device. RAID Controller or other storage controller is an attractive victim for firmware malware because they are IPL device as per BIOS Boot Specification. The BIOS Boot Specification and PCI specification dictates that IPL device firmware must be executed at boot if the IPL device is in use. IPL device firmware is mostly implemented as PCI expansion ROM. Therefore, IPL device firmware is always executed—assuming the IPL device is in use. This fact opens a path for firmware-level malware to reside in the IPL device firmware, particularly if the malware is required to be executed at every boot or on certain trigger at boot.

For more details on IPL device firmware execution, see: https://sites.google.com/site/pinczakko/low-cost-embedded-x86-teaching-tool-2. You need to take a closer look into Boot Connection Vector (BCV) in the PCI expansion ROM in that article. The system BIOS calls/jumps-into the BCV during bootstrap to start the bootloader, which then loads and executes the OS. BCV is implemented in the PCI expansion ROM of the storage controller device. Therefore, PERC RAID controller in Dell PowerEdge servers should implement BCV as well to conform to the BIOS boot specification.

IPL device PCI expansion ROM also has a peculiarity. In some BIOS implementation, it's always executed whether or not the IPL device is being used. The reason is because the BIOS code very probably only checks for the PCI device Subclass Code and Interface Code in its PCI configuration register. See "PCI PnP Expansion ROM Peculiarity" section at https://sites.google.com/site/pinczakko/low-cost-embedded-x86-teaching-tool-2#pci_pnp_rom_peculiarity.

System Management Mode (SMM) Primer

The third background knowledge needed to understand DEITYBOUNCE is SMM. A seminal work on SMM malware can be found in Phrack ezine article, i.e. *A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers* at http://www.phrack.com/issues.html?issue=66&id=11#article. SMI in this context means System Management Interrupt. The Phrack article contains required knowledge to understand how SMM rootkit might work.

There is one thing that needs to be updated in that Phrack article though. Recent and present day CPUs don't use High Segment (HSEG) anymore to store SMM code. Only Top-of-memory Segment (TSEG) is used for that purpose. If you're not familiar with HSEG and TSEG, you can refer to http://resources.infosecinstitute.com/system-address-map-initialization-in-x86x64-architecture-part-1-pci-based-systems/ and http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/ for details on HSEG and TSEG location in the

CPU memory space. This means, for maximum forward compatibility, DEITYBOUNCE very possibly only use TSEG in its SMM component.

Entering SMM via software SMI in x86/x64 is quite simple. All you need to do is write certain value to a particular I/O port address. Write transaction to this I/O port is interpreted by the chipset as a request to enter SMM, therefore the chipset sends an SMI signal to the CPU to enter SMM. There are certain x86/x64 CPU that directly "traps" this kind of I/O write transaction and interpret it directly as request to enter SMM—without passing anything to the chipset first. The complete algorithm to enter SMM is as follows:

1. Initialize DX register with the SMM "activation" port. Usually, the SMM "activation" port is port number 0xB2. However, it could be different port though, depending on the specific CPU and chipset combination. You have to resort to their datasheet for the details.
2. Initialize AX register with the SMI command value.
3. Enter SMM by writing the AX value to the output port contained in the DX register.

As for the methods to pass parameter to the SMI handler routine, it's not covered extensively by the Phrack article. Therefore, we will have a look the methods here.

Some SMM code (SMI handler) needs to communicate to other BIOS modules or to software running inside the OS. The communication mechanism is carried-out through parameter passing between the SMM code and code running outside SMM. Parameter(s) passing between BIOS module and SMI handler in general are carried out using one of these mechanisms:

1. Via the Global Non-Volatile Storage (GNVS). GNVS is part of ACPI v4.0 Specification and named ACPI Non-Volatile Storage (NVS) in ACPI specification. However, in some patents description NVS stands for Non-Volatile Sleeping memory because the memory region occupied by NVS in RAM stores data that's preserved even if the system is in sleep mode. Either term refer to the same region in RAM. Therefore, the discrepancies in naming can be ignored. GNVS or ACPI NVS is part of RAM managed by ACPI subsystem in the BIOS. It stores various data. GNVS is not managed by the OS, but is reachable to the OS through ACPI Source Language (ASL) interface. In Windows, parts of this region are accessible through the Windows Management Instrumentation (WMI) interface.

2. Via General Purpose Registers (GPRs) in x86/x64 architecture, i.e. RAX/EAX, RBX/EBX, RDX/EDX, etc. In this technique, *physical address pointer* is passed via GPR to the SMI handler code. Because the system state, including the register values are saved to the "SMM save state"—which area prior to entering SMM, the code in the SMM area (the SMI handlers) are able to read the pointer value. The catch is: both the SMI handler and the code that passes the parameter(s) must agree on the "calling convention", i.e. which register(s) to use.

Knowing parameter passing between BIOS module and SMI handler is important because this DEITYBOUNCE uses this mechanism extensively when it runs. We will look into it in more detail in later sections.

Precursor to DEITYBOUNCE Malware Architecture

As with other software, we can infer DEITYBOUNCE malware architecture from its execution environment. The NSA ANT server document mentions three technical hints, as you can see in the Introduction section. We'll delve into them one by one to uncover the possible DEITYBOUNCE architecture.

The need for RAID hardware means DEITYBOUNCE contains a malware implanted in the RAID controller PCI expansion ROM. The RAID controller used in Dell PowerEdge 1950 server is PERC 5/i, or PERC4e/DC, or PERC 5/e adapter card. All of these RAID controllers are either PCI or PCI Express (PCIe) RAID controller. You have to pay attention that PCI expansion ROM also includes PCIe expansion ROM because both are virtually the same. I have covered PCI expansion ROM malware basics in another article. See http://resources.infosecinstitute.com/pci-expansion-rom/ for the details.

The presence of a payload being dropped by DEITYBOUNCE means DEITYBOUNCE is basically a "second-stage" malware dropper—the first stage is the ARKSTREAM malware dropper. DEITYBOUNCE probably only provides these two core functions:

1. The first is as a persistent and stealthy malware modules dropper.
2. The second is to provide "stealth" control function to other OS-specific malware modules. The malware modules could be running during OS initialization or from inside a running OS or the malware modules can operate in both scenarios. The OS-specific malware communicates to DEITYBOUNCE via SMM calls.

DEITYBOUNCE core functions above imply that there are possibly three kinds of malware components required for DEITYBOUNCE to work as follows:

1. A persistent "infected" PCI expansion ROM. This module contains routine to configure DEITYBOUNCE frequency of execution. The routine possibly stores the configuration in the RAID controller NVRAM. This module also contains tainted interrupt 13h (Int 13h) handler that can call other routines via SMI to patch the kernel of the currently loading OS.
2. SMI handler(s) code implanted in the PowerEdge motherboard BIOS to serve the software (SW) SMI calls from the "infected" RAID controller PCI expansion ROM.
3. An OS-specific malware payload running in Windows 2000, Windows Server 2003 or Windows XP.

At this point we already know DEITYBOUNCE malware components. This doesn't imply that we would be able to know exact architecture of the malware because there are several possible pathways to implement the components. However, I present the most possible architecture here. This is an educated guess. There could be inaccuracies because I don't have a sample DEITYBOUNCE binary to back up my assertions. But, I think the guesses should be close enough given the nature of x86/x64 firmware architecture. If you could provide a binary sample with suspected DEITYBOUNCE in it, I'm open to analyze it though :-).

DEITYBOUNCE Malware Architecture

We need to make several assumptions before proceeding to DEITYBOUNCE architecture. This should make it easier to pin point DEITYBOUNCE technical details. These are the assumptions:

1. The BIOS used by the Dell PowerEdge targets are legacy BIOS, not EFI/UEFI. This assumption is strengthened by the NSA ANT server document in which it mentions the target OS as Windows 2000/XP/2003. None of these OS provides mature EFI/UEFI support. All user manuals, including BIOS setup user manual doesn't indicate any menu related to UEFI/EFI support, such as boot in legacy BIOS mode. Therefore, it's safe to assume that the BIOS is a legacy BIOS implementation. Moreover, during the launch time of DEITYBOUNCE, EFI/UEFI support in the market is still immature.
2. The custom SMI handler routines required to patch the OS kernel during bootstrap is larger than the empty space available in the motherboard BIOS. Therefore, the routines must be placed into two separate flash ROM storage, i.e. the PERC RAID controller flash ROM chip and the Dell PowerEdge motherboard flash ROM chip. This could be not the case, but just let's make an assumption here because even a rudimentary NTFS driver would require at least several tens of kilobytes of space when compressed. Not to mention a complicated malware designed to patch kernels of three different OS.

The assumptions above have several consequences on our alleged DEITYBOUNCE architecture. The first one is there are two stages in DEITYBOUNCE execution. The second one is the malware code that patches the target OS kernel during bootstrap (interrupt 19h) is running in SMM.

Now let's look into DEITYBOUNCE execution stages. DEITYBOUNCE execution stages are as follows:

1. Stage 1: PCI expansion ROM initialization during Power-On Self-Test (POST). In this stage DEITYBOUNCE installs additional malicious SMI handlers to the System Management RAM (SMRAM) range in the RAM on the motherboard. The assumption here is the SMRAM range is not yet locked by the motherboard BIOS. Therefore SMRAM range is still writeable. SMRAM is a range in system memory (RAM) that is used specifically for SMM code and data. Contents of SMRAM is only accessible through SMI after it has been locked. On most Intel northbridge chipsets or recent CPUs, SMRAM is controlled by register that controls the TSEG memory region. Usually, the register is called TSEG_BASE in Intel chipset documentation.
2. Stage 2: Interrupt 13h execution during bootstrap (interrupt 19h). Interrupt 13h handler in the PERC RAID controller PCI expansion ROM is "patched" with malicious code to serve interrupt 19h invocation (bootstrap). Interrupt 19h copies the bootloader to RAM by calling interrupt 13h function 02h—read sectors from drive—and jump into it. DEITYBOUNCE doesn't compromise the bootloader. However, DEITYBOUNCE compromise interrupt 13h handler. The "patched" interrupt 13h handler will modify the loaded OS kernel in RAM.

Figure 2 shows stage 1 of DEITYBOUNCE execution and Figure 3 shows stage 2 of DEITYBOUNCE execution.

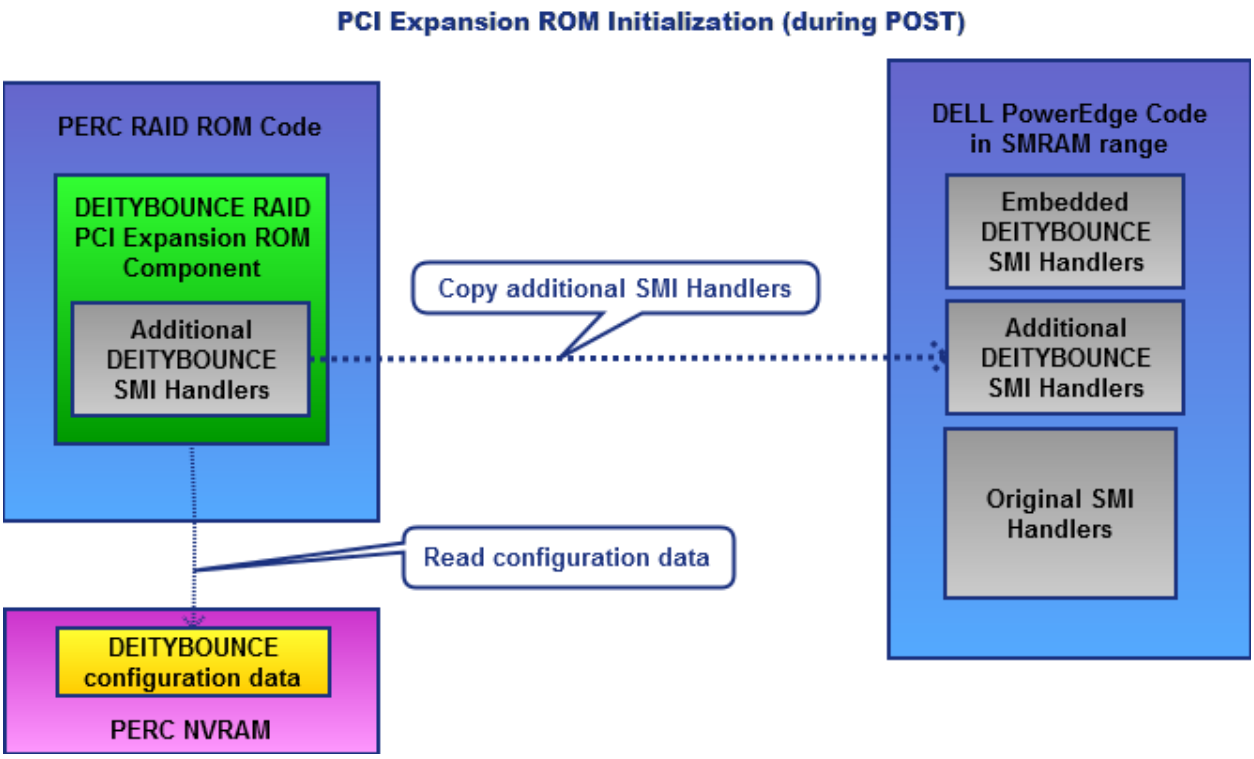


Figure 2 DEITYBOUNCE Execution Stage 1

DEITYBOUNCE stage 1 execution shown in Figure 2 happens during PCI expansion ROM initialization stage at POST. If you're not familiar with detailed steps carried out by the BIOS to initialize an x86/x64 system, a.k.a Power-On Self-Test, please read my System Address Map Initialization on x86/x64 Part 1 article at http://resources.infosecinstitute.com/system-address-map-initialization-in-x86x64-architecture-part-1-pci-based-systems/.

We know that PCI expansion ROM initialization is initiated by the motherboard BIOS from the *Malicious Code Execution in PCI Expansion ROM* article (http://resources.infosecinstitute.com/pci-expansion-rom/). The motherboard BIOS calls the INIT function (offset 03h from start of the PCI expansion ROM) with a far call to start add-on board initialization by the PCI expansion ROM. This event is the stage 1 of DEITYBOUNCE execution. In the DEITYBOUNCE case, the add-on board is the PERC PCI/PCIe board or the PERC chip integrated to the PowerEdge motherboard.

Figure 2 illustrates the following execution steps:

1. PERC RAID PCI expansion ROM executes from its INIT entry point.
2. The infected ROM code reads DEITYBOUNCE configuration data from the RAID controller NVRAM.
3. The infected ROM code copies DEITYBOUNCE additional SMI handlers to SMRAM range in the motherboard main memory (system RAM).
4. The infected ROM code fixes checksums of the contents of SMRAM as needed.

Once the steps above are finished, the SMRAM contains all of DEITYBOUNCE SMI handlers. Figure 2 shows the SMRAM contains “embedded” DEITYBOUNCE SMI handlers that’s already present in SMRAM before the DEITYBOUNCE component in the RAID controller PCI expansion ROM copied into SMRAM. The embedded DEITYBOUNCE component is injected into the motherboard BIOS. That’s why it’s already present in SMRAM early-on.

Figure 2 shows DEITYBOUNCE configuration data stored in the PERC add-on board NVRAM. This is an elegant and stealthy way of storing configuration data. How many anti-malware tools that scans add-on board NVRAM? I’ve never heard any.

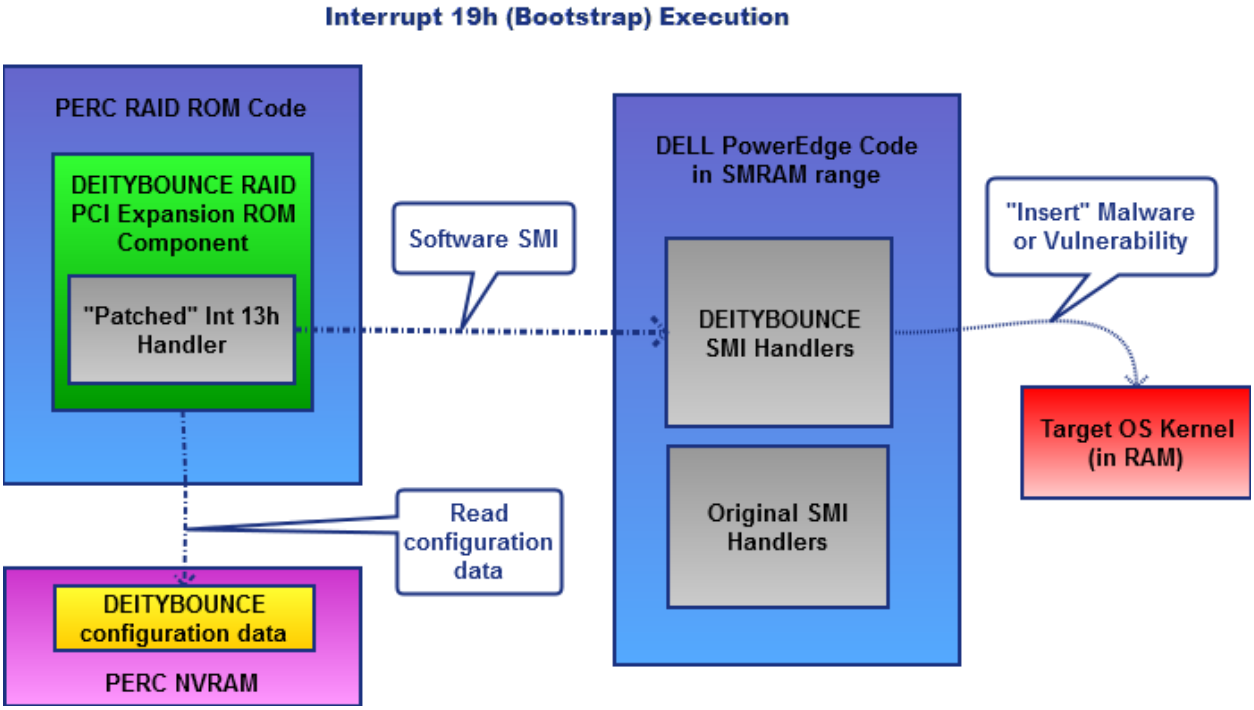


Figure 3 DEITYBOUNCE Execution Stage 2

Now, let’s move to DEITYBOUNCE execution stage 2. There are several steps in this execution stage as follows:

1. The mainboard BIOS executes the PERC RAID controller PCI expansion ROM routine at bootstrap via interrupt 19h (bootstrap). This time the entry point of the PCI expansion ROM is its BCV, not the INIT function. Interrupt 19h invokes interrupt 13h function 02h to read the first sector of the boot device—in this case the HDD controlled by the RAID controller—to RAM and then jump into it to start the bootloader.
2. The infected PCI expansion ROM routine contains a custom interrupt 13h handler. This custom interrupt handler executes when interrupt 13h is called by the bootloader and part of the OS initialization code. The custom routines contain the original interrupt 13h handler logic. But the custom one adds routines to infect the kernel of the OS being loaded. Interrupt 13h provides services to read/write/query to the storage device. Therefore, malicious coder can modify

contents of interrupt 13h handler routine to tamper the content being loaded to RAM. Figure 3 shows the custom interrupt 13h handler contains routine to call the DEITYBOUNCE SMI handler via software SMI. The DEITYBOUNCE SMI handler contains routine to install malware or activates certain vulnerabilities in the target OS kernel while the OS is still in initialization phase. Execution of the custom interrupt 13h handler depends on DEITYBOUNCE configuration data. Figure 3 DEITYBOUNCE configuration data related to the custom interrupt 13h handler is stored in the PERC RAID controller NVRAM.

The target OS contains vulnerability or malware after step 1 and 2 in DEITYBOUNCE second stage execution. Keep in mind that the vulnerability or malware only exists in RAM because the instance of the OS that's modified by DEITYBOUNCE exists in RAM, not in permanent storage device (HDD or SSD).

At this point we know how DEITYBOUNCE might work in sufficient detail. However, you should be aware that this is only the result of my preliminary assessment on DEITYBOUNCE. Therefore, it's bound to have inaccuracies.

Closing Thoughts: Hypotheses on DEITYBOUNCE Technical Purpose

There are two undeniable strategic value possessed by DEITYBOUNCE compared to "ordinary" malware:

1. DEITYBOUNCE provides stealthy way to alter the loaded OS without leaving a trace on the storage device, i.e. HDD or SSD in order to avoid being detected via "ordinary" computer forensic procedures. Why? Because the OS is manipulated when it's loaded to RAM, the OS installation on the storage device itself is left untouched (genuine). SMM code execution provides a way to conceal the code execution from possible OS integrity checks by other party scanners. In this respect, we can view DEITYBOUNCE as a very sophisticated malware dropper.
2. DEITYBOUNCE provides a way to preserve the presence of the malware in the target system because it is persistent against OS reinstallation.

Given the capabilities provided by DEITYBOUNCE there could possibly a stealthy Windows rootkit that communicates with DEITYBOUNCE via software SMI to call DEITYBOUNCE SMI handler routine at runtime—from within Windows—the purpose of such rootkit is unclear at this point. Whether the required SMI handler is implemented by DEITYBOUNCE is unknown at the moment.