Helltrap: Transforming physical machines into UEFI rootkit trap

Abstract

Sophisticated rootkits targeting the UEFI boot process are used in attacks to infiltrate devices and maintain persistence across reboots. Despite efforts to protect the boot process, challenges such as a bloated UEFI environment, insecure configurations, and firmware vulnerabilities have made these defenses insufficient. Thus, innovative detection and prevention methods are needed.

This work introduces Helltrap, a framework that transforms devices into honeypots capable of detecting rootkits that compromise the UEFI boot process. By leveraging firmware update capabilities, Helltrap automatically integrates monitoring systems within UEFI firmware and Baseboard Management Controller software found on server-grade systems. This allows it to detect modifications to UEFI firmware, Option ROM, or bootloaders, preventing rootkit payloads from executing. The payload is then automatically removed and made available for analysis.

To demonstrate the need for Helltrap, we collected real-world rootkit payloads that target UEFI firmware or the ESP partition. Our experiments show that these rootkits can evade detection by standard malware detection tools. To further underscore the danger of such malware, we also developed a proof-of-concept rootkit, GuardDown, which compromises even systems protected by Intel Boot Guard or Secure Boot. In contrast, Helltrap successfully detected all tested rootkits before execution.

1 Introduction

Malware capable of compromising devices persistently poses one of the greatest threats to the security of modern systems. In particular, rootkits targeting the boot process have proven to be especially stealthy and dangerous [18, 37, 38, 48, 52, 53, 60, 64]. Rootkits like LoJax [18] and CosmicStrand [60] have remained undetected for years, compromising devices through the flash storage used by UEFI. In this paper, we refer to all malware that infects one or more components of the Unified Extensible Firmware Interface (UEFI) boot process as UEFI *rootkits*.

These persistent rootkits target either (i) the bootloader stored in an ESP disk partition or (ii) the firmware flash storage that contains UEFI firmware, PCI card Option ROMs, and non-volatile UEFI variables. In the first case, the rootkit could have been deployed by any compromised software capable of writing arbitrarily to disk storage. In contrast, to compromise UEFI flash storage, a rootkit payload must be deployed by a remote adversary during runtime by compromising a signed software flashing process, or by a local adversary using an SPI flash programmer when the system is offline. However, in both cases, once deployed, the rootkit persists across system reboots and can compromise the device's OS before it executes. For example, it took years for Lojax to be discovered through manual analysis of UEFI firmware.

To address these identified attacks, device manufacturers have implemented various solutions aimed at protecting different stages of the boot process from rootkit compromise, including Intel BootGuard [21], Secure Boot [65], and Intel VT-d [4]. While these solutions strengthen the security of the boot process, they are often undermined by vulnerabilities [3, 16, 26, 63], misconfigurations [40], or their limited focus on specific attack vectors exploited by rootkits. Consequently, advanced rootkits such as BlackLotus [52] and LogoFail [46] have successfully bypassed these defenses. Furthermore, the increasing complexity of the modern UEFI boot process has introduced new attack surfaces, such as Option-ROM compromise, which can evade detection by malware scanning tools that focus solely on disk or UEFI flash storage.

The experiments described in Section 5.3 demonstrate that standard malware detection tools are generally effective at identifying rootkits only after their payloads have been discovered and analyzed. However, advanced rootkit payloads or novel threats, such as GuardDown (discussed in 5.2), can infect modern systems while evading detection. Once installed, these rootkits can disable Secure Boot and other protective measures, including antivirus programs and intrusion detection systems, which safeguard the bootloader and operating

system. Therefore, mitigating the impact of rootkit attacks requires deploying innovative solutions capable of detecting persistent malware targeting the UEFI boot process. Such solutions must be able to capture these threats for further analysis, enabling the identification and elimination of vulnerabilities they exploit.

This work introduces Helltrap, the first rootkit-capturing framework that employs physical honeypots to detect and capture rootkits. The Helltrap framework converts standard devices into honeypots by incorporating monitoring components that control the UEFI boot process. These components can identify rootkit payloads within UEFI flash storage or the ESP partition and capture them before they execute.

Under Helltrap, transforming devices into honeypots begins by integrating a set of Helltrap components into the software stack of the Baseboard Management Controller (BMC). The BMC, a dedicated processor commonly found in server-grade devices, is used for remote monitoring and management of the host system. These BMC components start the monitoring process by verifying the UEFI firmware and automatically deploying additional UEFI monitoring components to the BIOS flash storage at every boot. These components ensure the integrity of the entire boot process by verifying all its elements before execution.

The UEFI monitoring components examine UEFI variables, OptionROMs, and the bootloader for modifications that could indicate rootkit attacks. Any altered components are automatically sent to an external Helltrap server that collects them for analysis. The server extracts the modified UEFI files or variables and reports them for investigation.

Notably, the Helltrap BMC components remain completely hidden from the host system. Similarly, the Helltrap-introduced components within the UEFI firmware automatically uninstall themselves after completing their monitoring tasks by reflashing the firmware prior to OS execution. This approach maximizes rootkit detection by ensuring that no trace of Helltrap monitoring remains visible to the host system during runtime. It also makes it virtually impossible for remote adversaries or compromised software to detect the transformation of their targeted device into a honeypot. Detecting Helltrap's presence would require an adversary to gain physical access to the BMC flash storage.

The Helltrap design for physical honeypots offers two key advantages over existing high-interaction honeypots. First, unlike virtual honeypots, Helltrap honeypots are indistinguishable from real security-sensitive devices in both their hardware stack and host software stack. This eliminates the possibility of detection by rootkits based on exposed hardware, execution overhead, virtualization artifacts, or similar anomalies. Second, unlike existing honeypot frameworks that are not equipped to address boot process compromises, Helltrap includes components that can identify rootkit payloads before they execute. Additionally, Helltrap can automatically restore the honeypot to its pre-infection state by re-writing the compromised components, effectively resetting the device to its original state prior to the compromise.

The primary limitation of the Helltrap honeypot design is its reliance on a BMC processor to bootstrap the boot process, restricting Helltrap to transforming only server-grade devices into honeypots. However, as discussed in Section 6, ongoing efforts aim to develop alternative deployment methods that enable Helltrap to operate on devices without a BMC.

To assess the effectiveness of Helltrap, publicly available UEFI rootkit payloads discovered in the wild were collected and installed on systems protected by various malware detection tools. While these widely known payloads were detected by all the tested tools, simple encryption of the payloads allowed them to be stealthily installed without detection by any of the tools—except for Helltrap. As demonstrated in Section 5.2, even the correct deployment of state-of-the-art boot process protection mechanisms is insufficient to fully prevent UEFI rootkit infections. This underscores the critical importance of Helltrap honeypots for detecting sophisticated rootkits in the wild that target the UEFI boot process.

To summarize, this work introduces:

- The first design for physical honeypots capable of detecting rootkits that compromise UEFI storage.
- A novel mechanism for automatically transforming server-grade devices into honeypots.
- GuardDown, a proof-of-concept rootkit capable of compromising systems protected by both Secure Boot and Intel BootGuard.
- A comprehensive evaluation of Helltrap, demonstrating its effectiveness in detecting UEFI rootkits.

2 Background

2.1 UEFI boot process

Figure 1 illustrates the execution flow of the various software components involved in the UEFI process of most modern systems. When a BMC processor is present, the execution is typically started by software running on the BMC processor, either when the power button is pressed or when remote commands are received via the Intelligent Platform Management Interface (IPMI) [43]. In both scenarios, the BMC software triggers the start of a series of UEFI boot phases, which are further detailed in Section2.1 and are loaded from the BIOS flash storage (as described in Section 2.3).

The UEFI boot process scans and executes any PCI OptionROMs present on PCI expansion cards, as discussed in Section 2.4. After the UEFI firmware completes its execution, the bootloader stored in the ESP disk partition is loaded, which then starts either another bootloader or the operating system.

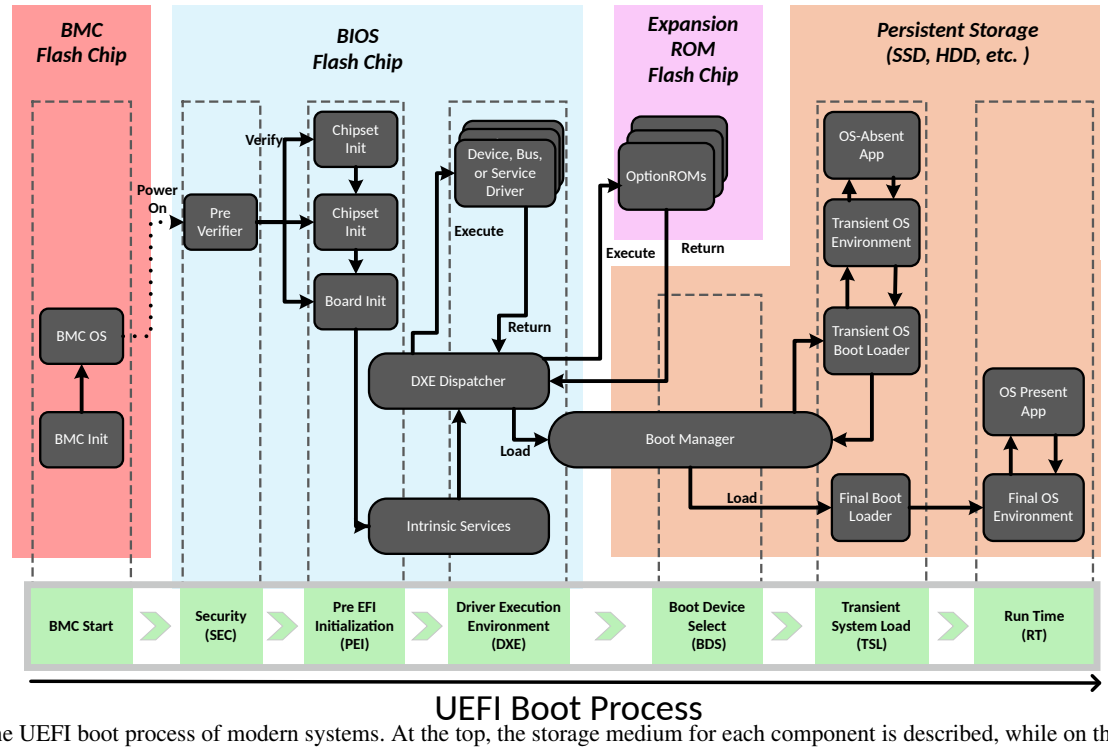

Figure 1: The UEFI boot process of modern systems. At the top, the storage medium for each component is described, while on the bottom the BMC start and standard UEFI phases are depicted. Note, the BMC flash chip components are standard only for server-grade devices and are typically used to control the power on and off process.

Baseboard Management Controller (BMC). The Baseboard Management Controller (BMC) is a dedicated microcontroller embedded in most server-grade motherboards. It provides remote management capabilities, including system monitoring, power management, system recovery, and firmware updates. In the UEFI boot process, the BMC operates as an out-of-band component, starting the UEFI phases by powering on the host system.

The rest of the UEFI process, as depicted in Figure 1, has been widely adopted as the standard boot process and comprises four primary stages:

Security (SEC). This phase serves as the UEFI root of trust, preparing the processor by establishing temporary memory storage for the next phase.

Pre-EFI Initialization (PEI). During this phase, main memory is initialized [41], and an in-memory description of the firmware image is prepared for the subsequent stage. PEI modules within the UEFI firmware are executed by a central module called the PEI Core.

Driver Execution Environment(DXE). In this stage, most system initialization tasks are performed by loading DXE drivers, which initialize platform components such as device controllers and peripherals. As shown in Figure 1, the DXE Core attempts to execute all DXE drivers whose dependencies it can resolve. It also loads and executes OptionROMs stored within PCI Expansion ROMs, completing this process by the end of the DXE phase.

Boot Device Selection (BDS). This phase is managed by the final DXE driver to execute. Its role is to discover and load the bootloader from a device specified via UEFI variables. If no device with a valid EFI System Partition (ESP) label is identified, the UEFI code scans connected storage devices for disks with EFI System Partition labels and loads the bootloader from one of them.

Finally, all boot services are terminated, and all DXE drivers are cleaned up before the operating system begins execution. Only those marked as runtime services remain available for OS use.

2.2 EFI system partition (ESP)

An EFI System Partition (ESP) typically stores a bootloader and its configuration data (e.g., the Boot Configuration Data, or BCD) in a FAT32 format. This partition is usually created during OS installation and is essential for booting the respective operating system. Crucially, all information stored in the ESP, including the bootloader, is accessible to the operating system and cannot be shielded from OS-level access.

To address this vulnerability, UEFI introduced a protection mechanism known as Secure Boot [65]. Secure Boot prevents the UEFI firmware from loading any bootloader that has been modified by an untrusted operating system. Under this system, the integrity of the bootloader is verified at every boot by checking its code hashes against signature databases stored in non-volatile UEFI variables. These signature databases are themselves validated through a chain of trust that originates

from a trusted platform key.

2.3 BIOS Flash Storage

Firmware contains the first instructions used by devices to initialize the hardware and perform system boot. Hence, it resides in fast, persistent, and easily accessible flash memory. The flash memory typically uses the SPI protocol to communicate with the result of the system, which reads and writes to the registers of the SPI flash controller.

Updating UEFI firmware. UEFI firmware stored in BIOS flash storage is updated to address software bugs, introduce new functionality, or patch security vulnerabilities. Firmware updates can be performed using either in-band or out-of-band updates. In-band updates are performed via available software flashing tools such as Chipsec [28], RWEverything [49], and flashrom [20], which allow updates from within the host OS. Out-of-band updates leverage either the BMC (Baseboard Management Controller) or external flash programmers.

BMC processors, in particular, enable remote BIOS flash updates through the IPMI interface. However, on security-sensitive systems, access to the IPMI interface must be restricted to local, trusted users and never exposed to remote adversaries, who could exploit it to fully compromise the system. Similarly, external flash programmers require physical access, as they must be attached to the BIOS flash chip using a clamp. In contrast, software flashing tools can be used by anyone with appropriate OS-level permissions to update the UEFI firmware. To mitigate risks from malicious firmware updates, several protection mechanisms have been introduced. Flash chip protection bits control CPU access to specific regions of the BIOS flash, while global flash protection bits restrict access to code running in System Management Mode (SMM), a highly privileged execution mode [36]. Additionally, protected range registers define writable address ranges within the BIOS flash and are secured by the FLOCKDN bit, which locks flash chip registers until the system is rebooted.

Introducing these protection mechanisms should ensure that only trusted software can write verified, signed UEFI firmware to the BIOS flash chip. However, these mechanisms are often improperly implemented or suffer from exploitable vulnerabilities, as showed in prior research [7, 31]. For example, the FLOCKDN bit is often not enabled in commercial off-the-shelf systems to allow users and administrators to perform firmware updates via the OS.

2.4 Option ROMs

During system startup, the UEFI boot process scans for and executes Option ROMs to initialize connected PCI devices. In the DXE phase of the UEFI boot process, the firmware begins by probing each PCI device's Base Address Registers (BARs) and Expansion ROM Base Address Registers (XROMBARs) to detect the presence of Option ROM firmware. Once located, the Option ROM code is copied into RAM and executed to complete device initialization.

In practice, some Option ROMs (e.g. embedded video card, embedded network) can also be stored within specific regions of the motherboard's BIOS flash chip. These Option ROMs are integrated into the motherboard firmware and are typically loaded by vendor-specific EFI modules and not detected by probing during the DXE phase.

Importantly, the firmware for both Option ROMs types is usually stored in EEPROM memory. This non-volatile storage allows hardware vendors to update Option ROM contents directly from the host system, using a firmware update process.

2.5 EFI variables

During the UEFI boot process, volatile and non-volatile variables are accessed by EFI modules through the Variable Runtime Service. This service allows EFI modules to read, write, and delete EFI variables identified by unique GUIDs used as keys. This service also grants limited access to certain EFI variables in the bootloader and the OS. In particular, variables that are non-volatile or flagged for runtime access by System Management Mode (SMM) modules remain available during system operation. Consequently, the firmware exposes a restricted set of EFI variables during boot, allowing the OS and bootloader to read and modify them as needed.

The Variable Runtime Service manages EFI variables as key-value data stored in either volatile or non-volatile memory. The volatile variables reside in RAM and are cleared when the system restarts, while non-volatile variables are kept in designated regions of the BIOS chip, preserving their values even when power is lost. These persistent variables are vital for retaining essential UEFI settings, including Secure Boot configurations and the boot device order, which the Variable Service and EFI modules rely on throughout the boot process.

3 Problem description

The rise of persistent rootkit attacks has revealed serious weaknesses in the security of the UEFI boot process. Modern rootkits can persist across reboots by embedding themselves in the firmware or modifying files within the ESP. By doing so, they can compromise the operating system even before traditional antivirus or integrity-verification tools are initialized.

Once a rootkit gains a foothold within the UEFI firmware, detecting it becomes extremely difficult. Operating at such a low level gives it full control over the system, allowing it to hide its presence and disable security tools before they can respond. The discovery of rootkits in the wild underscores the urgent need for proactive detection and prevention systems capable of identifying UEFI-targeting rootkits before they execute—blocking them from gaining control during the earliest stages of the boot process.

To defend against rootkit threats, hardware and firmware vendors have deployed security mechanisms like Secure Boot, Intel BootGuard, and BIOS write protection. These features are intended to ensure that only verified firmware and boot components are executed. However, analyses of real-world systems show that manufacturers often misconfigure these protections or introduce flaws during implementation—issues that stem largely from the complexity and fragmentation of the UEFI firmware supply chain.

Meanwhile, adversaries continue to discover new ways to exploit vulnerabilities within UEFI, such as executing unverified code from PCI Expansion ROMs or compromising verified EFI modules. Rootkit proof-of-concepts such as LogoFail [46] demonstrate how easily rootkits can bypass even advanced defenses focused solely on the BIOS or ESP regions. As these threats evolve, securing the UEFI boot process requires both improved firmware hardening and early detection strategies. Crucially, to defend against these threats systems must be capable of identifying the presence of rootkits before they can execute and gain control over the monitored system.

4 Helltrap

Helltrap is a rootkit-capturing framework that transforms standard server-grade devices into honeypots designed to detect and collect rootkit payloads. This transformation begins by modifying the BMC software stack to introduce components that verify the UEFI firmware before each boot of the host OS. These modifications include adding monitoring components that validate the integrity of all phases of the UEFI boot process. However, these monitoring components are removed before the OS boots, ensuring that the Helltrap honeypots remain indistinguishable from regular devices to remote adversaries or compromised software running on the host OS.

Figure 2 depicts the Helltrap architecture and how its components function to detect and capture rootkit installations within monitored UEFI storage. The detection process begins when the Helltrap server converts a target device into a honeypot by automatically installing several modules into its Baseboard Management Controller (BMC). These modules include monitoring components that validate the UEFI firmware and NVRAM variables prior to each system boot, as well as a UEFI boot controller responsible for loading specialized UEFI modules that track changes to both the Option ROMs and the EFI System Partition (ESP).

The introduced UEFI modules are executed during the DXE phase and verify the integrity of the bootloader and any present Option ROMs before they are executed. Crucially, any modified content is reported back to the Helltrap server for analysis using Helltrap networking modules integrated into the BMC and UEFI firmware. Each aspect of the honeypot and its components is further detailed in the following subsections.

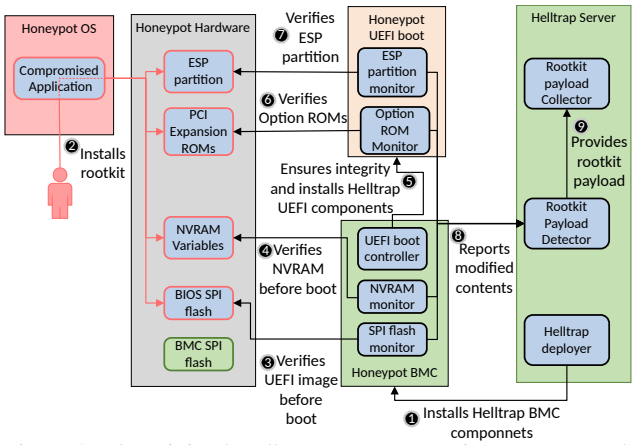

Figure 2: The minimal Helltrap system comprises a server and a device transformed into a honeypot. Root-of-trust components such as the trusted server and BMC are highlighted in green. Potential adversary pathways (attack vectors) for introducing UEFI rootkit payloads are shown in red.

4.1 Threat model

The Helltrap honeypot is designed to capture UEFI rootkits under these premises:

1. The BMC's hardware and firmware stack is trusted and cannot be accessed by attackers (for example, through IPMI).

2. Attackers compromise the target's boot sequence by injecting malicious code or data into the EFI System Partition (ESP) or into flash regions that hold UEFI firmware or Option ROMs.

3. Adversaries do not have physical access to the Helltrap honeypot hardware.

The first premise treats the BMC software and hardware as a secure, separate environment: it runs on its own processor and memory independent of the host, and is assumed to be immune to attacker tampering. Consequently, attack vectors that would update the BMC itself (such as remote flashing via IPMI) are excluded from this work. This is a practical restriction because interfaces like IPMI can typically be disabled on server-class machines.

The second premise narrows Helltrap's detection goals. It is not intended to catch every kind of malware—for example, transient OS-only infections that vanish after a reboot are outside its target. Instead, Helltrap focuses on persistent threats that gain lasting control of a device by altering NVRAM variables or by corrupting UEFI-related storage (SPI flash, Option ROMs, or the ESP). Rootkits that solely affect user-space or runtime applications need distinct, OS-level detection approaches and so are not covered here.

The third premise assumes attackers cannot physically access deployed honeypots. With physical access, an adversary

could read or overwrite BMC flash using a hardware programmer, discover Helltrap's presence, or remove its monitoring components. This assumption is reasonable in many deployments because physical access can be prevented through measures like secure facilities, continuous monitoring, and tamper-resistant enclosures. Furthermore, most real-world malware is deployed remotely, as requiring physical access poses a significant obstacle for the majority of adversaries.

4.2 Honeypot setup

The setup of new honeypots within the Helltrap framework is automated for server-grade devices equipped with an IPMI interface that allows modifications to the BMC software stack. In such cases, the Helltrap server introduces its monitoring components into the BMC firmware and starts a self-update process for the BMC to apply the modified firmware. However, the process becomes more challenging when device vendors restrict access to their proprietary BMC software stacks or limit updates to their own closed-source firmware.

For devices without IPMI access to the BMC, the honeypot transformation follows an alternate approach. In this scenario, an SPI flash programmer controlled by the Helltrap server is connected directly to the BMC chip of the targeted device. The Helltrap server reads the existing BMC firmware from the chip using the programmer, incorporates the honeypot-specific monitoring components, and writes the modified firmware back to the chip. Crucially, this process needs to be performed only once. Once deployed, the BMC components can independently communicate with the Helltrap server and update themselves via the network interface.

After the BMC components are installed, they complete the honeypot setup by booting the device once to record the contents of the UEFI firmware, EFI variables, Option ROMs, and ESP partition. These recorded contents serve as a baseline to verify future modifications. In the event of any detected infection, the honeypot automatically restores the device to a trusted state using these baseline records.

4.3 Rootkit detection

What sets the Helltrap honeypots apart is their ability to monitor all data and code utilized during the UEFI boot process before it can compromise the device or detect the monitoring mechanisms. This section provides a detailed explanation of how this monitoring is achieved.

UEFI firmware monitoring. To ensure that no UEFI firmware executes without first being scanned by Helltrap, the BMC components integrated into the honeypots intercept the command used to power on the host system. This command is implemented within the BMC firmware as a set of *ioctl* functions exposed via APIs to the IPMI interface. Helltrap modifies this process by preventing these commands from directly powering on the host and initiating the UEFI boot process. Instead, Helltrap first triggers a scan of the UEFI flash chip. If the scan detects no modifications, the power-on process proceeds as normal. However, if modifications are identified, the altered UEFI firmware is forwarded to the Helltrap server for analysis. The server determines whether the firmware has been compromised and, if necessary, restores it to its original state.

The UEFI firmware scan is enabled by the BMC's ability to access the BIOS flash chip directly. This functionality is standard in all server-grade systems, allowing for remote BIOS updates. Typically, this access is facilitated through a dedicated multiplexer controlled by the BMC. The multiplexer determines whether the BIOS flash chip is accessible to the host software stack or the BMC firmware. Under Helltrap, the multiplexer is configured to place the BIOS chip under BMC control before each boot, allowing Helltrap to read or write its contents as required.

Unlike host OS flashing tools, which involve manual, time-consuming, and flash chip-specific operations, the Helltrap BMC components access the SPI chip directly as a memory-mapped device. This approach significantly improves the efficiency and portability of the UEFI firmware scanning process across a variety of server-grade devices.

EFI variable monitoring. As mentioned earlier, EFI variables are stored within the BIOS flash chip on modern devices and are read and verified from the NVRAM partition before each host boot. However, some of these variables may change between boot sessions.

To prevent false-positive rootkit detections when non-security-critical EFI variables are modified, Helltrap employs a whitelist database on its server. This whitelist identifies EFI variables that should be excluded from integrity verification. The database is reusable across multiple devices and straightforward to implement due to the UEFI standard, which mandates that each variable be uniquely identified by a GUID. These excluded variables typically serve non-critical purposes, such as logging.

Additionally, Helltrap's ability to read and write EFI variables allows it to manipulate boot configurations strategically. For instance, Helltrap can automatically disable Secure Boot to lure adversaries into deploying rootkits, such as Lojax, which target the ESP partition. This functionality helps in the capture and analysis of malicious activity.

PCI Expansion ROM monitoring. Unlike UEFI firmware and variables, which can be monitored directly by the BMC, PCI Expansion ROMs are only accessible to the host software and firmware. As a result, monitoring PCI Expansion ROMs within Helltrap honeypots requires a different approach.

Fortunately, the process of verifying UEFI firmware provides a valuable opportunity for Helltrap honeypots to re-establish trust in the loaded UEFI firmware during every boot. This capability also enables Helltrap to dynamically introduce additional UEFI modules at each boot. These modules can verify all Option ROMs during the DXE phase, ensuring their

integrity before they execute.

The Option ROM verification process is handled by a Helltrap UEFI module that scans connected PCI devices for the presence of any Option ROMs. For each detected Option ROM, the module computes a hash and forwards it to the Helltrap server, where it is compared against the stored Option ROM hash recorded during the honeypot setup. If discrepancies are found, the complete Option ROM image is sent to the server for further analysis.

Importantly, introducing the Option ROM scanning module is feasible even on systems protected by Intel BootGuard. However, this requires an additional step during the honeypot setup: the updated UEFI firmware images must be signed by the device manufacturer to pass BootGuard's verification during startup. Notably, this Option ROM verification process is performed regardless of the device's Secure Boot configuration, ensuring consistent monitoring across all systems.

ESP partition monitoring. Helltrap introduces a dedicated UEFI module to address the challenge of monitoring the ESP. This module is automatically deployed by the UEFI boot components running on the BMC.

In the Helltrap framework, ESP monitoring is triggered automatically by hooking into the *ExitBootServices* UEFI event, which is executed during the UEFI boot process just before the bootloader is launched. At this stage, the introduced UEFI module scans the system hardware for connected disks and verifies each one. When a disk is identified, the Helltrap module searches for any ESP partitions by locating the standard *EFI* directory used by EFI boot managers. Once identified, the files within each ESP partition are validated against a set of pre-computed hashes collected during the honeypot setup phase. Any discrepancies are promptly forwarded to the Helltrap server for analysis.

It is important to note that some non-security-critical files, often used for logging purposes, may reside within the ESP partition. To prevent false positives, such files are excluded from verification using a whitelist, following the same process described for handling NVRAM variables.

Notably, the ESP monitoring process operates independently of the Secure Boot mechanism and cannot be disabled by an adversary under any circumstances.

UEFI monitoring removal. In the standard UEFI boot process, all Helltrap-introduced UEFI modules are automatically removed from device memory before the bootloader executes. However, this is insufficient to erase traces of Helltrap from the host system completely at runtime, as the BIOS flash chip still contains the Helltrap-modified UEFI firmware. Therefore, at the conclusion of the BDS phase, all introduced UEFI modules must remove themselves from the stored UEFI firmware.

Fortunately, the UEFI boot process provides the capability to interact directly with the BIOS flash chip. Leveraging this feature, Helltrap introduces an additional UEFI module that re-flashes the original, signed, pre-modified UEFI firmware back into the BIOS flash chip after all monitoring activities are completed. This module can write to the flash chip without issue because BIOS write protections are designed to permit the flashing of legitimate UEFI firmware.

The BMC monitoring components, however, do not require removal since they remain isolated and inaccessible to any software running within the host system.

4.4 Rootkit payload collection

Helltrap automatically detects and captures all changes made to UEFI firmware by performing comprehensive scans of the entire BIOS flash chip. However, the impact of these changes varies depending on which part of the BIOS chip is modified. Manually analyzing all altered UEFI images collected by honeypots would be a slow and resource-intensive process. To address this, Helltrap incorporates an automated parsing system that analyzes modified UEFI firmware, Option ROMs, ESP partitions, and EFI variables. This process leverages established techniques commonly employed by open-source tools, such as UEFITool [50].

UEFI firmware modification. Helltrap utilizes the UEFI firmware flash descriptors located at the beginning of the image to identify and parse each UEFI region. For each region, Helltrap takes advantage of the well-defined modular structure of UEFI images. For instance, the BIOS region organizes UEFI modules as separate files within firmware volumes using the Firmware File System (FFS). These volumes typically group files related to specific UEFI execution phases, with each file containing one or more sections that store data. Notably, the most critical sections include PEI modules executed during the Pre-EFI Initialization (PEI) phase, and DXE drivers or applications, executed later during the Driver Execution Environment (DXE) phase. Other file types, such as free-form and raw formats, are treated as generic binaries since their proprietary nature makes their inner workings largely unknown.

To streamline the analysis process, Helltrap parses each volume and file within a modified UEFI firmware and compares them against the baseline collected during the honeypot setup. This approach enables Helltrap to generate detailed, fine-grained reports that pinpoint which files have been altered by a potential rootkit. Consequently, it can detect even the smallest modifications to UEFI firmware that could compromise system integrity.

EFI variables. Any modification to EFI variables that persist across reboots is also captured within the UEFI firmware. Unlike older systems, where NVRAM variables are stored in battery-backed memory, modern devices store these variables in the NVRAM volume of the UEFI firmware, allowing them to persist through power cycles. While these variables can be cleared by removing the CMOS battery, such an attack requires physical access to the device's motherboard. Additionally, clearing the NVRAM volume also erases any malicious content intended to execute during the device boot process. Moreover, Helltrap detects and reports any clearing

of NVRAM variables to the Helltrap server, as this action can be part of an attack. For example, clearing NVRAM variables may disable Secure Boot by erasing its configuration.

Helltrap can easily extract and report only the modified EFI variables. The NVRAM volume is uniquely identified by its GUID ("CEF5B9A3-476D-497F-9FDC-E98143E0422C") within the BIOS region. Once identified, the contents of each variable are compared against their corresponding values recorded during the honeypot setup, using their unique GUIDs. Some systems store multiple copies of NVRAM variables, such as default factory values that are loaded when the CMOS battery is removed, triggering a reset. However, this duplication is not an issue for Helltrap, as it accurately identifies and differentiates each copy of the NVRAM variables during its parsing of the UEFI firmware.

Fine-grained identification of NVRAM modifications is crucial for detecting sophisticated rootkit payloads. For instance, as demonstrated by LogoFail, altering the NVRAM variable containing a device's logo image—or even its file path—can be enough to hijack a vulnerable UEFI boot process entirely.

Option ROMs. Unlike the well-documented UEFI firmware format, which can be parsed using open-source tools, the structure and operation of Option ROMs are entirely proprietary and closed-source. Option ROMs stored as distinct regions within the BIOS flash chip are extracted in their entirety whenever a byte-level comparison reveals modifications. This process is robust against false negatives, as Option ROMs typically remain unchanged across device boots unless explicitly updated. Similarly, Option ROMs located within PCI Expansion ROMs are also compared in their entirety for the same reason. Identifying modifications to Option ROMs is crucial for detecting sophisticated attacks. For example, at DEF CON [51], a proof-of-concept rootkit was concealed within the Intel GbE Option ROM region.

ESP partition. Modifications to the filesystem of the ESP partition contents are automatically extracted at the granularity of individual modified files by iterating through each folder inside this partition recursively and comparing the file hashes against those stored during the honeypot setup.

5 Evaluation

To evaluate Helltrap, all publicly available physical rootkit payloads that target either UEFI firmware stored in SPI flash or files within the ESP partition were collected. These payloads include nine rootkits discovered in the wild, five of which compromise UEFI firmware (LoJax [18], MossaicRegressor [38], CosmicStrand [60], MoonBounce [37], and VectorEDK [23]), and four that target the ESP partition (FinSpy [64], ESPecter [53], BlackLotus [52], and Gluteba [48]). Additionally, a novel proof-of-concept rootkit, dubbed *GuardDown*, was developed. This rootkit can compromise even systems protected by Intel BootGuard and Secure Boot.

Table 1 summarizes the effectiveness of Helltrap and other commonly used malware detection tools in identifying rootkits that compromise UEFI firmware. Similarly, Table 2 presents the results for rootkits that target the ESP partition. Finally, Section 5.2 demonstrates how, in contrast to Helltrap, most commonly deployed protection mechanisms fail to detect or fully prevent certain rootkit infections.

5.1 Rootkit deployment

For most rootkits, the actual method used for their deployment remains unknown. However, the white papers that present an analysis of Lojax and VectorEDK reveal that most have been likely deployed remotely using malicious software running on the host. Thus, a generic rootkit deployment method has been constructed for the introduction of rootkits from a host OS either into the UEFI firmware stored inside the BIOS flash chip or the ESP disk partition.

Deployment of rootkits that target UEFI firmware is as follows: First, the existing UEFI firmware image is read using *CHIPSEC* [28], a platform security assessment framework that can read and write data to the SPI device. Next, the rootkit files are generated by adding the required header data to every rootkit DXE payload using the open-source edk2 *GenFfs* [15] tool. These files are then inserted into the read UEFI firmware image using the open-source *UEFITool* [50], commonly used to read and modify UEFI firmware images. Finally, the resulting malicious UEFI firmware image is written back to the device using *CHIPSEC*. This process is automated using a simple Python script.

Similarly, for rootkits targeting the ESP partition another Python script with administrator privileges simply mounts the ESP partition and copies the malicious rootkit files, overwriting existing ones when necessary. Of course, this ESP dropper only works when Secure Boot is disabled. However, as shown by the discovery of Black Lotus [52], this limitation can be removed by exploiting Secure Boot vulnerabilities.

Finally, to evaluate the deployment of more sophisticated or unknown rootkits, a generic rootkit hardening tool has also been created. This tool simply encrypts the malicious UEFI modules or ESP files used by rootkits using the AES CBC encryption algorithm. These encrypted modules are then inseted using the same python scripts described alongside a dedicated DXE module or EFI application that will automatically decrypt and execute them during the boot process. Importantly, this hardening tool and decryption modules are very generic and are not flagged as malicious by tested malware detection tools. The encryption and decryption process itself is a standard technique that is commonly used by software developers to protect proprietary code against reverse engineering.

Malware Detector \ Rootkit	Lojax		Mossaic-Regressor		Cosmic-Strand		Moon-Bounce		Vector-EDK		**GuardDown**	
	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc
Hybrid Analysis	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Virus Total	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
MetaDefender Cloud	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Jotti malware scan	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Windows Defender	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
ESET antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Kaspersky antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Bitdefender antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
FwHunt	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗
Helltrap	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: UEFI firmware rootkit detection

5.2 GuardDown - Novel rootkit attack vector

The primary defense against rootkit infections in modern systems is ensuring the integrity of the boot process through mechanisms such as Intel BootGuard and Secure Boot. However, these defenses are not universally enabled, and even when active, they are not foolproof.

This section demonstrates that modern devices remain vulnerable to UEFI rootkit infections by detailing the implementation of *GuardDown*, a proof-of-concept rootkit that employs a novel attack vector to bypass signed firmware verification, circumvent Intel BootGuard, and even disable Secure Boot. The target device for this proof-of-concept is the Supermicro X12SDV-8C-SPT4F [54] motherboard. This motherboard was selected solely because it supports all modern defense mechanisms; no board-specific vulnerabilities were exploited.

Signed firmware verification bypass. The attack begins by exploiting a vulnerability in the BIOS flash chip write restrictions to bypass signed firmware verification and perform arbitrary writes to UEFI firmware. This vulnerability involves a race condition introduced when trusted flashing tools temporarily enable write operations for software running on the host operating system. An adversary can exploit this race condition by either compromising the flashing tool itself or performing malicious writes while the flashing tool is active.

For example, AMI Firmware Update (AFU) [29] is a trusted flashing tool provided by most hardware vendors for updating UEFI firmware from the host OS. This tool consists of a user-space application that only allows UEFI firmware updates through vendor-signed binaries and kernel drivers that memory-map the BIOS flash chip into the userspace of an application. The kernel driver is responsible for unlocking write access to the BIOS flash chip. This is achieved by performing specific System Management Mode (SMM) calls. Once these SMM calls finish, write access to the BIOS flash chip remains unlocked until the driver execution completes.

In the GuardDown attack, the AMI Firmware Update process was paused after it erased the NVRAM section of the BIOS flash chip, leaving write operations unlocked. At this point Intel's open-source CHIPSEC tool [28] could be used to inject a malicious rootkit payload into the BIOS flash chip.

Intel BootGuard bypass. On systems protected by Intel BootGuard, firmware code integrity is verified at every boot before execution is permitted. Additionally, Intel BootGuard cannot be disabled via the host operating system and can be permanently enabled using a hardware fuse. Consequently, attacks that involve modifying or introducing malicious UEFI modules are not feasible on such systems. Instead, attackers can only target unprotected areas of the UEFI firmware, such as the NVRAM variable storage.

NVRAM variables are often modified across multiple boot cycles, typically through the BIOS setup interface (e.g., for selecting a boot disk). As a result, Intel BootGuard does not verify these variables against pre-stored hashes, even though some of them, like Secure Boot configurations, are critical to the boot process's security.

The GuardDown rootkit exploits this limitation in Intel BootGuard's firmware verification by combining it with the signed firmware verification bypass. This allows it to modify the NVRAM variables stored in the BIOS flash chip that are associated with Secure Boot. While this constitutes a modification of the UEFI firmware, Intel BootGuard's verification process does not detect or prevent it.

Secure Boot bypass. Once adversaries gain write access to Secure Boot NVRAM variables, compromising or disabling Secure Boot becomes straightforward. GuardDown opts for the latter, fully disabling Secure Boot by using CHIPSEC to set its enabled bit to false. This action allows a malicious bootloader to be deployed by simply writing it to the ESP partition, as there are no longer any mechanisms preventing the loading of unsigned bootloaders or Option ROMs.

Stealthier attack vectors are also possible without fully disabling Secure Boot. For instance, the Secure Boot signature databases stored within NVRAM variables can be modified to include additional keys, which can then be used to sign malicious bootloaders or Option ROMs. This approach allows

the adversary to bypass Secure Boot while maintaining its appearance of being enabled.

A more sophisticated attack vector involves leveraging a malicious Option ROM after disabling Secure Boot. In this scenario, the malicious Option ROM deploys a temporary bootloader at the end of the DXE phase of each boot cycle. This bootloader performs its intended malicious activities and removes itself from the ESP partition before the operating system boots, rendering the compromise virtually undetectable to the host OS.

GuardDown detection. As shown in Table 2 and Table 1, GuardDown is not detected by any of the tested malware detection tools. This is because the attack leverages only open-source software to modify the UEFI firmware, and the malicious bootloader lacks any previously known signature. While it would be straightforward to create a signature for the malicious bootloader through analysis, it is equally trivial to modify or obfuscate the bootloader to evade detection.

Importantly, none of the methods used in the proof-of-concept attack can bypass detection within a Helltrap honeypot. First, any modification to the UEFI firmware is immediately identified at every device boot by the BMC monitoring components, which also track changes to NVRAM variables. Second, even if Secure Boot is disabled, the Helltrap components temporarily introduced into the UEFI firmware during boot detect any attempt to load untrusted Option ROMs or introduce unauthorized data or code into the ESP partition. As a result, Helltrap honeypots effectively detect and prevent GuardDown deployment.

GuardDown disclosure. The vulnerability was discovered when testing the version 5.16.01.0109 of AFU and disclosed to AMI's Product Security Incident Response Team (PSIRT). In later AFU tools this vulnerability seems to have been fixed, as neither we or the PSIRT team has been able to reproduce it with newer AFU tool versions. For AFU 5.16.01.0109, we are currently coordinating with the PSIRT team and trying to provide all the necessary information for them to reproduce the discovered vulnerability and publish an advisory.

5.3 Rootkit deployment detection

The payloads of GuardDown and the nine collected rootkits were subjected to scanning by four popular antivirus programs: *Windows Defender* [42], *ESET Antivirus* [17], *Kaspersky Antivirus* [32], and *Bitdefender Antivirus* [11]. This scanning was performed at runtime on the host system before introducing the respective payloads using the Python scripts described in Section 5.1.

The rootkits were also submitted to four widely used online file-scanning platforms: *Hybrid Analysis* [27], *VirusTotal* [22], *MetaDefender Cloud* [45], and *Jotti Malware Scan* [30]. These tools perform more in-depth analyses, including execution of the rootkits in virtual sandboxes. Some platforms also aggregate results from multiple malware detection services, including antivirus engines, to enhance their evaluations.

Additionally, the payloads of the five rootkits targeting UEFI firmware were scanned using *FwHunt* [10], an open-source tool designed specifically for analyzing UEFI firmware images. Unlike the other tools, *FwHunt* was used to scan the entire UEFI firmware compromised by each rootkit rather than individual payloads. This approach maximizes detection potential, as firmware scanning tools rely on additional rules based on UEFI module GUIDs, which are effective only when analyzing the entire firmware.

As shown in Table 1, most detection tools successfully identified all known rootkit payloads when they matched the exact format previously discovered in the wild. These tools leverage malware signatures, heuristics, and manually crafted detection rules to recognize previously analyzed rootkits. However, the results changed drastically when the payloads were encrypted, as described in Section 5.1. In this scenario, no rootkit payload was detected by any of the tested tools. A notable exception is GuardDown: its modified UEFI firmware, with Secure Boot disabled, was never flagged as malicious by any existing tool. However, Helltrap easily identified this subtle modification to the NVRAM storage when scanning the BIOS flash chip during the boot process.

The evaluation of ESP partition-targeting rootkits, as presented in Table 2, tells a similar story. All payloads were detected in their original format, while none of the encrypted versions were flagged as malicious by conventional detection tools. In contrast, during the honeypot boot process, Helltrap successfully detected the introduction of both plaintext and encrypted payloads in the ESP partition. While Helltrap does not directly classify these payloads as explicitly malicious—its primary purpose is to collect suspicious artifacts rather than evaluate their behaviour—it flags any unauthorized changes. In the context of a honeypot deployment, any modifications to UEFI firmware or the ESP partition that are not performed by administrators or permitted under the whitelists described in Section 4 are highly suspicious and indicative of an attack.

These experiments underscore how easily adversaries can create sophisticated rootkits that evade detection by runtime malware detection tools. Simple obfuscation techniques, such as encryption or compression, are often sufficient to bypass such tools. However, these techniques do not affect Helltrap's ability to detect rootkits. Regardless of the deployment methods or obfuscation techniques used, Helltrap honeypots reliably detect unauthorized modifications to UEFI firmware, Option ROMs, UEFI variables, or the ESP partition. While obfuscation may complicate the subsequent analysis of detected artifacts, it does not impede Helltrap's detection capabilities.

5.4 Rootkit collection validation

Accurately capturing rootkit payloads is critical for initiating a proper analysis of their behaviour within controlled sandboxes. To validate this process, each rootkit payload collected by

Malware Detector \ Rootkit	FinSpy		ESPecter		Black Lotus		Gluteba		Bootkitty		**GuardDown**	
	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc	Plain	Enc
Hybrid Analysis	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Virus Total	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
MetaDefender Cloud	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Jotti malware scan	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Windows Defender	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
ESET antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Kaspersky antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Bitdefender antivirus	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗
Helltrap	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: ESP rootkit detection. *FwHunt* was not designed to scan ESP file systems and was therefore not utilized.

Helltrap was reintroduced into a separate validation system with the same hardware configuration as the honeypot.

In our experiments, all the collected rootkits listed in Table 1 and Table 2 successfully replicated their behaviour within the validation systems. This outcome highlights Helltrap's ability to automatically extract and report payloads with precision, significantly reducing the effort required for analysis.

Crucially, due to compatibility issues between the rootkit logic and virtual machines (VMs), each payload had to be tested on physical systems. For instance, Lojax failed to execute properly in a QEMU [9] VM and could not compromise the OS. Similarly, rootkits such as FinSpy and Gluteba contain anti-VM detection logic that alters their behaviour when running in virtualized environments. These findings emphasize the importance of using physical honeypots that precisely replicate the hardware and software configurations of security-sensitive systems.

5.5 Boot process protection

One might argue that the boot process itself could be sufficiently hardened to prevent rootkit infections altogether. However, despite numerous efforts in this area, no foolproof system has been developed. Table 3 provides a breakdown of the boot protection systems evaluated for their effectiveness in preventing the deployment of the collected rootkit payloads. These evaluations were conducted on two server-grade motherboards: an older model, the SM-X10SDV-TLN4F [55] (referred to as X10), which offers UEFI boot protection through Secure Boot and BIOS flash chip protection pins; and a newer model, the SM-X12SDV-8C-SPT4F (X12), which incorporates additional protections such as Intel BootGuard and a secure BIOS flashing process that restricts updates to trusted UEFI firmware binaries.

As expected, when configured properly, Intel BootGuard and BIOS chip protection bits successfully blocked adversaries from using tools such as CHIPSEC or RWEverything to deploy rootkits into UEFI firmware. Conversely, the signed BIOS flashing mechanism failed to prevent rootkit deployment, as it only verifies updates initiated via the BMC and does not secure the direct write process used by these tools. Notably, on the X10 motherboard, the BIOS chip write protection appeared to be misconfigured, allowing unrestricted writes to the BIOS flash chip by both tools. More importantly, GuardDown was able to bypass all BIOS protection systems by exploiting the vulnerabilities described in Section 5.2.

To demonstrate Helltrap's adaptability, we also tested a version of the framework designed to prevent rootkit infections rather than detect and collect them, referred to as the "defensive mode." In this configuration, the whitelists and data saved during the honeypot setup are used by the Helltrap BMC and UEFI components to enforce boot integrity. Specifically, the system prevents booting if any unexpected modifications are detected in the monitored UEFI firmware, UEFI variables, Option ROMs, or ESP partition. Additionally, the Helltrap components embedded within the UEFI firmware are made persistent to avoid frequent removals that could wear down the SPI flash write cycles.

As shown in the rightmost column of Table 3, this "defensive mode" of Helltrap successfully prevents the deployment of all tested rootkits, including GuardDown, outperforming existing boot protection systems.

6 Discussion

In this section, the threat model assumptions are further discussed, with a focus on the limitations of the Helltrap framework prototype.

BMC compromise. Compromising the BMC software stack is theoretically possible. Several vulnerabilities [57–59] have been identified in the IPMI (Intelligent Platform Management Interface) used by proprietary BMCs. These vulnerabilities could allow adversaries to execute arbitrary code within the BMC. However, such attacks are mitigated in Helltrap honeypots by completely disabling the IPMI interface and ensuring the BMC is not exposed to any adversary-accessible network. Within the Helltrap framework, the BMC only communicates

Rootkit \ Boot protection	Intel BootGuard	Secure Boot	Signed BIOS Flashing	SuperMicro X10 BIOS chip write-protect	SuperMicro X12 BIOS chip write-protect	Helltrap (defensive mode)
Lojax	✓	✗	✗	✗	✓	✓
MossaicRegressor	✓	✗	✗	✗	✓	✓
CosmicStrand	✓	✗	✗	✗	✓	✓
MoonBounce	✓	✗	✗	✗	✓	✓
VectorEDK	✓	✗	✗	✗	✓	✓
FinSpy	✗	✓	✗	✗	✗	✓
ESPecter	✗	✓	✗	✗	✗	✓
BlackLotus	✗	✗	✗	✗	✗	✓
Gluteba	✗	✓	✗	✗	✗	✓
GuardDown	✗	✗	✗	✗	✗	✓

Table 3: Effectiveness of boot protection measures against rootkits that infect either the UEFI firmware or ESP partition

with the Helltrap server.

While disabling the IPMI interface effectively reduces the BMC's attack surface, it also limits its functionality, as the BMC can no longer manage the host system. Future versions of the framework aim to address this limitation by enabling IPMI functionality through the Helltrap server, ensuring both security and usability.

Rootkit deployment via IPMI interface. Any rootkit introduced into the BIOS flash chip, including those deployed through the IPMI interface, is automatically detected by the BMC components before boot. However, this detection process relies on the integrity of the BMC software stack. If the IPMI interface is vulnerable, it could also be exploited to compromise the BMC's monitoring functionality. To address this risk, the current Helltrap design disables the IPMI interface entirely, thereby eliminating this attack vector.

Devices without a BMC. The current Helltrap prototype is designed for server-grade devices that include a BMC processor on their motherboard. Devices such as standard workstations and laptops, which lack a BMC, cannot currently be converted into honeypots for capturing UEFI rootkits.

To overcome this limitation, ongoing work focuses on developing a hardware-based BMC alternative. This alternative leverages existing technologies such as flash chip programmers and low-cost KVM systems like PiKVM [14] to connect directly to the BIOS flash chip and manage the power-on and power-off functionality of host systems. Early experiments indicate that this approach is both feasible and cost-effective.

The hardware-based BMC alternative also provides additional capabilities, such as protecting the BMC processor itself by monitoring its firmware stored in the BMC flash chip. Furthermore, this alternative can detect and capture rootkits deployed via a compromised IPMI interface, as it eliminates reliance on the BMC software stack.

Legacy BIOS. The Helltrap framework currently focuses on devices using the UEFI boot process, which has largely replaced the older, non-standardized legacy BIOS. While the Helltrap design could be extended to support systems with legacy BIOS, this would require significant engineering effort. Given that approximately 80% of modern systems now utilize UEFI [1], the current focus on UEFI is justified.

Evading Helltrap detection. Rootkits can only evade detection in Helltrap honeypots by avoiding infection of the UEFI firmware, UEFI variables, Option ROMs, or ESP partition. Certain types of malware, such as those targeting only kernel memory, fall outside the scope of Helltrap, as they require runtime protection mechanisms like antivirus software, fine-grained access controls, and kernel integrity protection systems. These protections, while orthogonal to Helltrap's boot process verification, can benefit from Helltrap by ensuring the OS is not compromised during boot.

Detection of Helltrap monitoring. Using physical devices as honeypots instead of virtual machines offers significant advantages for Helltrap. First, physical honeypots can be configured with hardware identical to that used in security-critical systems, ensuring realistic testing conditions. Second, no runtime overhead is introduced, which could otherwise be detected by sophisticated adversaries. Third, common VM-detection techniques employed by malware are entirely ineffective against physical honeypots. Finally, the monitoring components embedded in the BMC or UEFI firmware are invisible to rootkits and adversaries with access to the host system.

Crucially, monitoring components within the UEFI firmware are removed after completing verification tasks, which occur prior to OS boot, to prevent their detection by rootkits at runtime.

7 Related Work

Hardware-assisted system monitoring. Helltrap is not the first system to leverage an isolated processing environment to monitor host system execution. Hypercheck [70] uses the CPU's System Management Mode (SMM), an isolated high-privilege execution mode, to capture the monitored system's state and transmit it to a remote server. Similarly, Nighthawk [72] employs the Intel Management Engine (ME) processor in the Platform Controller Hub (PCH) to intro-

duce SMM monitoring code, enabling introspection of system software at runtime. However, both solutions are limited to runtime monitoring and depend on the integrity of UEFI firmware to initialize their monitoring environments. Consequently, they are unable to detect or prevent rootkit infections targeting the UEFI boot process.

In contrast, Helltrap functions even when UEFI firmware is compromised and detects rootkits embedded within the firmware before they can execute. Furthermore, while Intel ME and SMM monitoring code reside in UEFI firmware—making it susceptible to detection or removal by UEFI rootkits in the absence of Intel BootGuard—Helltrap ensures its monitoring code is always removed after UEFI storage verification. This guarantees that the presence of the honeypot remains undetectable by rootkits during runtime.

UEFI firmware integrity. In response to the increasing prevalence of rootkit attacks, commercial device manufacturers have introduced firmware protection mechanisms such as Intel Boot Guard, OpenBMC verified boot [44], and AMD Platform Secure Boot (PSB) [2]. These solutions use public keys stored in one-time programmable fuses to prevent the execution of untrusted firmware. However, as Alex Matrosov demonstrated [40], supply chain complexities can result in unprotected firmware in modern devices. Moreover, vulnerabilities like those reported in CVE-2018-12169, CVE-2019-11098, and related work [16] have shown that Boot Guard can be bypassed, enabling the execution of unsigned firmware.

Additionally, the cryptographic keys used during every boot to verify firmware signatures are vulnerable to leaks, as evidenced in [3]. Therefore, relying solely on such solutions is insufficient to protect devices, necessitating robust rootkit detection systems like Helltrap. Moreover, during the DXE (Driver Execution Environment) phase, malicious code can be loaded from PCI Expansion ROMs. Such code remains unchecked unless Secure Boot is enabled.

There are also various hardware-based solutions, such as Caliptra [12], AWS Nitro [5], Lenovo ThinkShield Firmware Resilience [39], HP Sure Start [25], that take control of the boot process and verify the UEFI firmware integrity before boot, some even repairing it using a backup copy. However, these solutions are vendor-specific and not widely deployed. They are subject to the same supply chain problems that can result in leaked cryptographic keys or unpatched vulnerabilities. Thus, Helltrap ability to automatically capture new rootkits is essential in identifying any unknown vulnerabilities they might leverage and improving these solutions.

Firmware write protection. Modern systems restrict access to UEFI firmware modifications through System Management Mode (SMM), a highly privileged execution mode for x86 processors. Only SMM drivers executing in this mode can control low-level hardware and modify firmware, with access restricted via special interrupt handlers. To strengthen these protections, fuzzing-based approaches have been developed to harden the SMI (System Management Interrupt) interface against arbitrary code execution [19, 67, 68].

However, limiting arbitrary SMM access addresses prevents only one avenue for compromising UEFI firmware. Comprehensive solutions like Helltrap, which monitor all writes to SPI flash and UEFI variables, are essential for detecting and preventing rootkit installations.

Firmware honeypots. Helltrap is not the first honeypot designed for rootkit capture. However, existing honeypots rely on virtualized hardware and focus on detecting malicious code within the kernel. For instance, Argos, an x86 emulator developed by Pauna [47], performs dynamic taint analysis to detect kernel rootkit installations. Similarly, the Honware framework [61] captures malware using IoT honeypots, replacing Linux-based firmware kernels with custom ones for hardware virtualization. However, this approach is limited to Linux-based IoT devices and cannot replace non-standard components in UEFI firmware.

Malware detection. Commercial antivirus solutions typically employ a mix of malware detection techniques. While effective at identifying known rootkit payloads, our evaluation shows they struggle to detect hardened rootkits designed to evade detection. In contrast, Helltrap remains effective in identifying the deployment of such rootkits through its continuous monitoring of non-volatile storage.

As shown by Aslan and Samet [8], behavior-based malware detection approaches [13, 35, 62] outperform traditional signature-based methods [56, 71, 73], heuristics [6, 66, 69], or model-checking [24, 33, 34]. However, these approaches require runtime execution and monitoring of rootkit samples within controlled environments. As shown in Section 5, Helltrap's ability to automatically collect rootkits facilitates the use of behaviour-based detection techniques, enabling the analysis and identification of rootkits across other systems.

8 Conclusion

To prevent rootkit infections, it is essential to protect modern machines not only during runtime but also throughout the boot process. However, the discovery of sophisticated rootkit attacks that compromise the bootloader or UEFI firmware of security-sensitive machines highlights the inadequacy of existing boot protection systems. This issue is further underscored by GuardDown, a novel proof-of-concept rootkit introduced in this work. These challenges demonstrate the urgent need for new systems capable of detecting vulnerabilities in protection mechanisms and identifying rootkit infections.

This work introduced Helltrap, a framework that transforms server-grade devices into honeypots capable of detecting rootkits which compromise the UEFI boot process. Helltrap not only identifies UEFI rootkits before they execute but also captures their payload automatically and reports it for further analysis. Experimental results demonstrate that, unlike standard malware detection tools, Helltrap effectively detects GuardDown and all previously identified rootkits.

9 Ethics considerations

All rootkits in the evaluation section, except GuardDown, are publicly available and have been extensively analyzed and documented in whitepapers or blogs detailing their behaviour. Experiments confirm that these rootkits' binaries are widely recognized, with most malware detection tools capable of identifying them in the wild. The encrypted versions of these rootkits will be reported to malware detection tools and made publicly available along with the generic rootkit hardening tool. This tool employs a straightforward encryption method that neither requires disclosure nor exploits any unknown vulnerabilities.

For GuardDown, the write protection race vulnerability used to bypass firmware verification has been reported to the AFU tool vendor and CERT. We are actively coordinating its public disclosure with these entities.

References

[1] Trusting in the cpu: Getting to the roots of security. https://uefi.org/sites/default/files/resources/Getting%20a%20Handle%20on%20Firmware%20Security%2011.11.17%20Final.pdf.

[2] John Abbott. Trusting in the cpu: Getting to the roots of security. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Trusting-in-the-CPU.pdf, 2017.

[3] Lawrence Abrams. Intel investigating leak of intel boot guard private keys after msi breach, May 2023.

[4] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel virtualization technology for directed i/o. *Intel technology journal*, 10(3), 2006.

[5] Amazon. Lightweight Hypervisor - AWS Nitro System - AWS.

[6] William Arnold and Gerald Tesauro. Automatically generated win32 heuristic virus detection. In *Proceedings of the 2000 international virus bulletin conference*, 2000.

[7] Paul Asadoorian. Firmware security realizations - part 3 - spi write protections. https://eclypsium.com/blog/firmware-security-realizations-part-3-spi-write-protections/, Sep 2022.

[8] ömer Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020.

[9] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX annual technical conference, FREENIX Track*, volume 41, page 46. Califor-nia, USA, 2005.

[10] Binarly. Fwhunt. https://github.com/binarly-io/FwHunt.

[11] Bitdefender. https://www.bitdefender.com.

[12] Chipsalliance. GitHub - chipsalliance/Caliptra: Caliptra IP and firmware for integrated Root of Trust block.

[13] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE transactions on information forensics and security*, 11(2):289–302, 2015.

[14] Maxim Devaev. Pikvm. https://pikvm.org/.

[15] Edk ii project. https://github.com/tianocore/edk2, 2023.

[16] Mark Ermolov and Maxim Goryachy. How to hack a turned-off computer, or running unsigned code in intel management engine. *Black Hat Europe*, 2017.

[17] ESET. Eset antivirus. https://www.eset.com.

[18] ESET. Lojax first uefi rootkit found in the wild, courtesy of the sednit group. Technical report, 2018.

[19] Francesco Evangelista. *Automatic Extraction of Exploitation Primitives in UEFI*. PhD thesis, Politecnico di Torino, 2023.

[20] Flashrom. https://github.com/flashrom/flashrom, 2023.

[21] Jessie Frazelle. Securing the boot process: The hardware root of trust. *Queue*, 17(6):5–21, feb 2020.

[22] Google. Virustotal. https://www.virustotal.com.

[23] HackingTeam. vector-edk. https://github.com/hackedteam/vector-edk, 2004.

[24] Andreas Holzer, Johannes Kinder, and Helmut Veith. Using verification technology to specify and detect malware. In *International Conference on Computer Aided Systems Theory*, pages 497–504. Springer, 2007.

[25] HP. HP Sure Start.

[26] Trammell Hudson. CVE-2018-12169. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12169, Jun 2018.

[27] Hybrid analysis. https://www.hybrid-analysis.com.

[28] Intel. Chipsec: Platform security assesment framework. https://github.com/chipsec/chipsec#chipsec-platform-security-assessment-framework, 2023.

[29] American Megatrends International. Ami firmware update (afu). https://www.ami.com/bios-uefi-utilities/.

[30] Jotti malware scan. https://virusscan.jotti.org.

[31] Corey Kallenberg and Rafal Wojtczuk. Speed racer: Exploiting an intel flash protection race condition. *Bromium Labs (January 2015)*, 2015.

[32] Kaspersky. Kaspersky antivirus. https://www.kaspersky.com.

[33] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Detecting malicious code by model checking. In *Detection of Intrusions and Malware, and Vulnerability Assessment: Second International Conference, DIMVA 2005, Vienna, Austria, July 7-8, 2005. Proceedings 2*, pages 174–187. Springer, 2005.

[34] Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Proactive detection of computer worms using model checking. *IEEE transactions on dependable and secure computing*, 7(4):424–438, 2008.

[35] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, XiaoFeng Wang, et al. Effective and efficient malware detection at the end host. In *USENIX security symposium*, volume 4, pages 351–366, 2009.

[36] Xeno Kovah and Corey Kallenberg. Advanced x86: Bios and system management mode internalsspi flash protection mechanism. https://opensecuritytraining.info/IntroBIOS_files/Day2_03_Advanced%20x86%20-%20BIOS%20and%20SMM%20Internals%20-%20SPI%20Flash%20Protection%20Mechanisms.pdf.

[37] Mark Lechtik, Vasily Berdnikov, Denis Legezo, and Ilya Borisov. Moonbounce: the dark side of uefi firmware. Technical report, Kaspersky, 2022.

[38] Mark Lechtik, Igor Kuznetsov, and Yury Parshin. Mosaicregressor: Lurking in the shadows of uefi. Technical report, Kaspersky, 2020.

[39] Lenovo. Firmware Resiliency with Lenovo ThinkShield.

[40] Alex Matrosov. Betraying the bios: Where the guardians of the bios are failing. https://www.blackhat.com/docs/us-17/wednesday/us-17-Matrosov-Betraying-The-BIOS-Where-The-Guardians-Of-The-BIOS-Are-Failing.pdf, 2017.

[41] Alex Matrosov, Eugene Rodionov, and Sergey Bratus. *Rootkits and bootkits: reversing modern malware and next generation threats*. No Starch Press, 2019.

[42] Microsoft. Windows defender. https://www.microsoft.com/en-us/windows/comprehensive-security.

[43] Corey Minyard. Ipmi–a gentle introduction with openipmi. *XP055165227, Software Montavista*, pages 1–238, 2006.

[44] OpenBMC. OpenBMCSecureBoot.

[45] OPSWAT. Metadefender cloud. https://www.hybrid-analysis.com.

[46] Fabio Pagani and Alex Matrosov. Logofail: Security implications of image parsing during system boot. https://i.blackhat.com/EU-23/Presentations/EU-23-Pagani-LogoFAIL-Security-Implications-of-Image_REV2.pdf, 2023.

[47] Adrian Pauna. Improved self adaptive honeypots capable of detecting rootkit malware. In *2012 9th International Conference on Communications (COMM)*, pages 281–284, 2012.

[48] Lior Rochberger and Dan Yashnik. Diving into glupteba's uefi bootkit. Technical report, Palo Alto Networks, 2024.

[49] Rweverything - read & write everything. http://rweverything.com/.

[50] Nikolaj Schlej. Uefitool. https://github.com/LongSoft/UEFITool, 2023.

[51] Mickey Shkatov and Jesse Michael. Bytes in disguise. https://www.youtube.com/watch?v=KDo3CExd8Ns, Aug 2020.

[52] Martin Smolár. BlackLotus UEFI Bootkit: Myth Confirmed. https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/, 2023.

[53] Martin Smolár and Anton Cherepanov. UEFI Threats Moving to ESP: Introducing ESPecter Bootkit. https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/, October 2021.

[54] SuperMicro. Supermicro x12sdv-8c-spt4f. https://www.supermicro.com/en/products/motherboard/x12sdv-8c-spt4f.

[55] SuperMicro. Supermicro x12sdv-tln4f. https://www.supermicro.com/en/products/motherboard/X10SDV-TLN4F.

[56] Yong Tang, Bin Xiao, and Xicheng Lu. Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. *Computers & Security*, 28(8):827–842, 2009.

[57] Binarly Research Team. Brly-2023-001. https://www.binarly.io/advisories/brly-2023-001, Oct 2023.

[58] Binarly Research Team. Brly-2023-007. https://www.binarly.io/advisories/brly-2023-007, Oct 2023.

[59] Binarly Research Team. Cve-2024-36435 deep-dive: The year's most critical bmc security flaw. https://www.binarly.io/blog/cve-2024-36435-deep-dive-the-years-most-critical-bmc-security-flaw, Sept 2024.

[60] Global Research & Analysis Team. Cosmicstrand: the discovery of a sophisticated uefi firmware rootkit. Technical report, Kaspersky, 2022.

[61] Alexander Vetterl and Richard Clayton. Honware: A virtual honeypot framework for capturing cpe and iot zero days. In *2019 APWG symposium on electronic crime research (eCrime)*, pages 1–13. IEEE, 2019.

[62] Gérard Wagener, Radu State, and Alexandre Dulaunoy. Malware behaviour analysis. *Journal in computer virology*, 4:279–287, 2008.

[63] Jian Wang. CVE-2019-11098. https://cve.mitre.org/cgi-bin/cvename.cgi?name=2019-11098, Apr 2019.

[64] WikiLeaks. FinSpy. https://wikileaks.org/spyfiles4/customers/finspy/, 2015.

[65] Richard Wilkins and Brian Richardson. Uefi secure boot in modern computer security solutions. In *UEFI forum*, pages 1–10, 2013.

[66] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang. An intelligent pe-malware detection system based on association mining. *Journal in Computer Virology*, 4:323–334, 11 2008.

[67] Jiawei Yin, Menghao Li, Yuekang Li, Yong Yu, Boru Lin, Yanyan Zou, Yang Liu, Wei Huo, and Jingling Xue. Rsfuzzer: Discovering deep smi handler vulnerabilities in uefi firmware with hybrid fuzzing. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2155–2169, 2023.

[68] Jiawei Yin, Menghao Li, Wei Wu, Dandan Sun, Jianhua Zhou, Wei Huo, and Jingling Xue. Finding smm privilege-escalation vulnerabilities in uefi firmware with protocol-centric static analysis. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1623–1637, 2022.

[69] Boyun Zhang, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. Malicious codes detection based on ensemble learning. In *Autonomic and Trusted Computing: 4th International Conference, ATC 2007, Hong Kong, China, July 11-13, 2007. Proceedings 4*, pages 468–477. Springer, 2007.

[70] Fengwei Zhang, Jiang Wang, Kun Sun, and Angelos Stavrou. Hypercheck: A hardware-assistedintegrity monitor. *IEEE Transactions on Dependable and Secure Computing*, 11(4):332–344, 2014.

[71] Min Zheng, Mingshen Sun, and John C.S. Lui. Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 163–171, 2013.

[72] Lei Zhou, Fengwei Zhang, Jidong Xiao, Kevin Leach, Westley Weimer, Xuhua Ding, and Guojun Wang. A coprocessor-based introspection framework via intel management engine. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1920–1932, 2021.

[73] Mohamad Fadli Zolkipli and Aman Jantan. A framework for malware detection using combination technique and signature generation. In *2010 Second International Conference on Computer Research and Development*, pages 196–199. IEEE, 2010.