Bonanza: A Persistent Rootkit Capturing Framework For Virtual Honeypots

Abstract—**The UEFI boot process is the keystone of most modern computers making it an ideal target for malicious actors. Unfortunately, code and data used during the boot process are rarely adequately protected, and as a result, increasingly powerful malware is emerging in the form of rootkits that enable malicious actors to bypass existing defences and gain persistent control over devices stealthily. Thus it has become imperative to develop and deploy boot process monitoring methods tailored for detecting such malware.**

This work introduces Bonanza, a framework that enables virtual honeypots to automatically detect and collect persistent rootkits by (i) luring adversaries through the use of emulated non-volatile storage that is designed to seemingly provide access to UEFI data and code, (ii) immediately identifying emulated storage compromise, and (iii) automatic extraction of introduced rootkit payload for further analysis. To maximize the detection of persistent rootkits that utilize various attack vectors discovered in the wild, Bonanza allows virtual machines (VMs) to expose a fully virtualized UEFI firmware flash with "misconfigured" write protections that mimic the operations of SPI chips located on real devices, intercepts and logs emulated PCI device option ROM writes, replaces all VM-specific UEFI variables with those extracted from real devices, and monitors Extensible Firmware Interface system partition (ESP) modifications.

Experiments performed using the most prominent persistent rootkits discovered in the wild show that it is trivial for adversaries to evade detection from popular runtime malware detection tools when deploying them through standard obfuscation techniques such as encryption. In contrast, Bonanza allows VMs to not only always detect the deployment of such rootkits due to its continuous monitoring of non-volatile storage access and contents, but also automatically extract the rootkits for an analysis crucial for identifying such attacks. Importantly, Bonanza also the first framework to expose real device UEFI information to rootkits instead of the UEFI emulated data and code, hardening VMs against rootkit detection.

1. Introduction

UEFI boot process compromise has emerged in recent years as one of the most dangerous attack vectors utilized by adversaries to gain persistent and complete system control and numerous successful attacks have been discovered [1]–[8]. Figure 1 depicts four attack vectors discovered in the wild that have been used by adversaries to deploy malware on a device that compromises its boot process. In all four cases depicted, introduced malware infects the bootloader or UEFI boot process responsible for initializing the device, such as the UEFI firmware, PCI card Expansion ROMs, or non-volatile UEFI variables. Such malware persists across system reboots and can surreptitiously compromise the device's OS before it even starts executing. This enables the rootkit to bypass commonly deployed malware detection tools and obtain control over any executing software, including any security-sensitive applications. In this paper, we refer to all malware that infects one or more components of the UEFI boot process as UEFI *rootkits*.

The discovery of systems infected with rootkits in some of the most well-protected enterprises [1], [2] is a clear indication that existing malware detection systems are ill-equipped to handle this threat. As further confirmed in the experiments detailed in Section 5.2, once a rootkit compromises the UEFI firmware, it becomes exceedingly difficult if not impossible to mitigate the impact of its malicious actions without a complete *physical* disassembly of the affected systems. The rootkit can disable Secure Boot [9], malware detection tools (anti-viruses, intrusion detection systems, etc.), and other similar solutions designed to protect the bootloader and OS. Moreover, as depicted in Figure 1, rootkits can target various non-volatile storage used by UEFI code, making it difficult to detect and remove. For example, rootkits that infect the UEFI firmware or PCI Expansion ROMs can only be removed by re-flashing the respective flash memory. They survive the operations typically used to remove malware, such as reinstalling the operating system or wiping the disks. In contrast, ESP bootkits compromise the bootloader and are not affected by UEFI firmware re-flashing, requiring the cleaning of the ESP disk partitions instead. Thus, to mitigate the impact of rootkit attacks it becomes essential to identify malware that utilizes the attack vectors shown in Figure 1 and automatically capture it for analysis, such that detection tools can fingerprint it and quickly put an end to rootkit compromise.

This work introduces Bonanza, a UEFI rootkit capturing framework for virtual high-interaction honeypots [1]. This framework allows virtual honeypots to automatically detect when malware compromises the persistent storage used during the UEFI boot process and collect any malicious payload introduced, and collect the respective payload automatically.

To maximize the detection of various rootkit attack vectors, Bonanza introduces inside virtual honeypots emulated access to various fake persistent physical memory

1. A high-interaction honeypot is not simply emulating some vulnerable services but instead is "configured to mirror a production system, and is designed to give an attacker full reign of an operating system if they are lured into compromising it." [10]

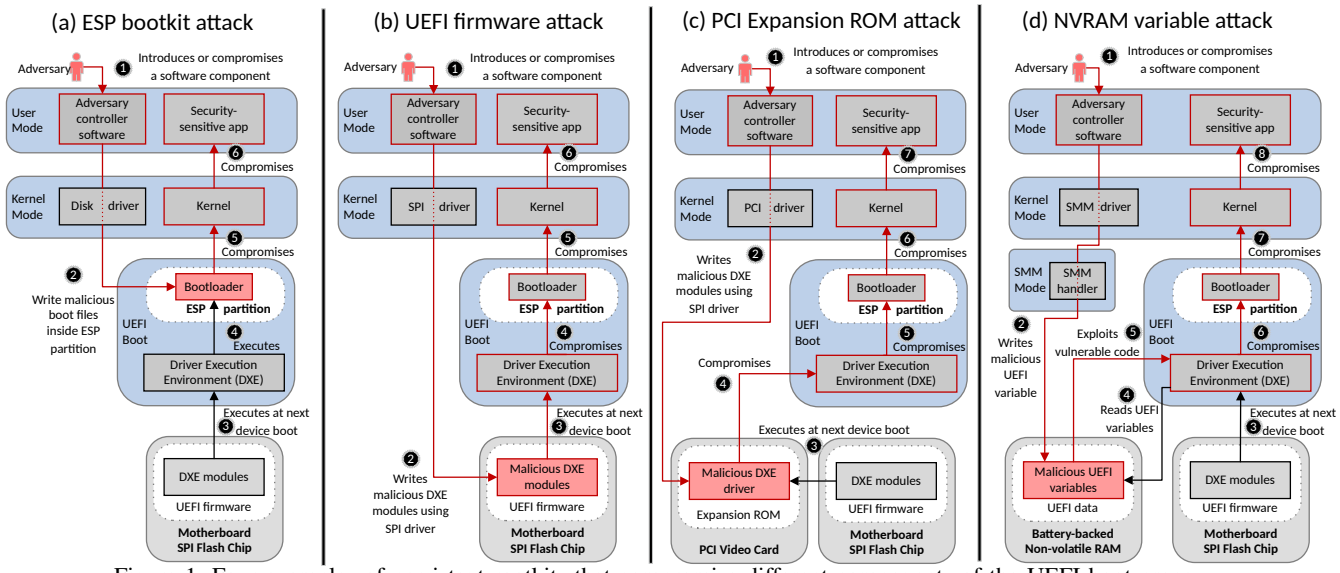

Figure 1: Four examples of persistent rootkits that compromise different components of the UEFI boot process.

designed to mimic the security-sensitive storage used by real physical devices, such as SPI flash chips, battery-backed memory, and PCI Expansion ROMs. Access to this memory is continuously monitored by Bonanza from the hypervisor level alongside writes to the ESP disk partition, and guest OS's operations. Further, the contents of this memory is automatically replaced by Bonanza prior to OS execution with firmware code and data extracted from real devices. This enables Bonanza to (i) trick rootkits into deploying their malicious payloads inside the monitored fake persistent storage, (ii) immediately identify a rootkit the moment it attempts to compromise the UEFI boot process, and (iii) capture the rootkit payload before it can execute on the next device boot. Through this process, Bonanza enables virtual honeypots to report both the introduced payload and crucial information regarding the methods utilized by rootkits to deploy it.

The Bonanza design for virtual honeypots has two main advantages over the honeypots based on real physical devices. First, the honeypot containing the fake virtual storage and OS to be continuously monitored from the hypervisor and allows Bonanza to seemingly expose various security-sensitive systems to rootkits. Second, the virtualized environments facilitate the execution of potential malicious files into a multitude of fake security-sensitive devices and OSes without the need for hundreds of expensive machines that are also difficult to manage and maintain.

A Bonanza evaluation using the rootkits previously discovered in the wild indicates that the proposed framework can automatically capture rootkit payload the moment they attempt to infect the UEFI boot process of a system. Further, the rootkit operations automatically collected by Bonanza provide critical information required to detect the presence of such rootkits in other systems, such as altered UEFI variables, introduced malicious EFI modules and modified files inside ESP disk partitions. To demonstrate the importance of detecting the very infection of UEFI persistent storage, an obfuscated version of each rootkit was created through a process that ensures their payload is encrypted before its execution during device boot. By testing the deployment of these obfuscated rootkit versions inside systems protected by nine popular malware detection tools, the experiments show that it is trivial for adversaries to harden rootkit deployment against detection from malware detection tools, even when their payload has been previously discovered and fingerprinted. In contrast to the detection tools evaluated, Bonanza introduced monitoring of persistent storage remains capable of detecting the deployment of all obfuscated rootkits. Finally, the LoJax [1] and MossaicRegressor [2] case studies highlight the importance of collecting rootkit payloads and executing them inside virtual and physical machines with various software and firmware components for the correct identification and fingerprinting of rootkit behavior.

To summarize, this work provides:

- A new storage emulation mechanism for virtual honeypots that seemingly provides access to vulnerable SPI chips, battery-backed memory and PCI extension ROMs.
- A novel automated UEFI rootkit detection framework that identifies and captures malware the moment it attempts to compromise persistent UEFI data or code.
- An evaluation of Bonanza that shows that it can automatically detect and collect obfuscated persistent rootkits that evade detection by popular malware detection tools.

2. Background

2.1. SPI Flash Memory

Firmware contains the first instructions used by devices to initialize the hardware and perform system boot. Hence, it must reside in fast, persistent, and easily accessible memory. EEPROM flash memory is the most common type used for

firmware that can be easily updated by manufacturers and consumers through a well-established flashing process.

Most EEPROM flash chips use an ubiquitous SPI protocol and are typically referred to as SPI flash or serial flash. A system communicates with the SPI flash by reading and writing to the registers of the SPI flash controller found within the chipset configuration space.

SPI Flash Regions. SPI flash contains several regions each holding firmware for distinct purposes, one of them being the UEFI firmware. We refer to the chip containing the UEFI firmware as the UEFI firmware flash chip, even though more firmware is present. While the content of this flash chip differs between systems, a common layout for Intel architecture can be seen in Figure 2. Firmware data is separated into regions, the top region is known as the Flash Descriptor Region and is used to identify the other regions and other system information. Next, the BIOS region highlighted in green which holds the UEFI firmware image, whose layout is also depicted in the Figure. The other regions (e.g ME, ACPI, etc.) contain various proprietary device-specific data and code and vary considerably between system architectures.

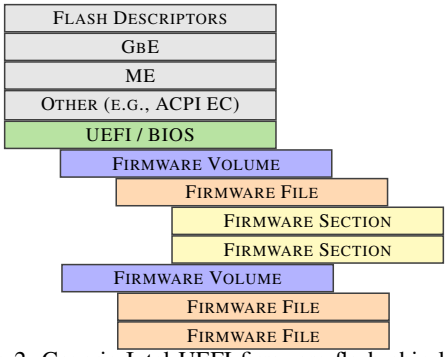

Figure 2: Generic Intel UEFI firmware flash chip layout.

2.2. Writing and Protecting SPI Flash

Firmware requires updates to patch software bugs, support additional functionality, or fix security vulnerabilities. While there exist multiple update methods, including freely available programs such as chipsec [11], RWEverything [12], and flashrom [13], there also exist several SPI write protection systems, which are designed to prevent malicious flashing of SPI flash. The following briefly describes these layers and highlights previously discovered problems with each of them.

First, there are protection bits located in the flash descriptor region. These bits indicate which regions can be accessed by the CPU, including the flash descriptor region itself. However, if the flash descriptor region itself is writable, an adversary can bypass this protection [14].

Next, the global flash protection bits restrict access to only code that executes when the processor is in a special mode known as System Management Mode (SMM) [15]. SMM is a highly privileged execution mode present on x86-based architectures that suspends the OS and has access to all of the system's memory. These protections have previously been bypassed by leveraging a race condition that exists between the two bits [16], motivating the addition of an extra bit to mitigate such attacks.

There are several vendor write protections in place to prevent attackers from writing to the flash chip which are chipset dependent. The chipset exposes settings that limit access to the BIOS region of the BIOS flash device: the BIOS Write Enable Bit (BIOSWE), the BIOS Lock Enable Bit (BLE) and the SMM BIOS Write Protect Disable (SMM_BWP). All settings are accessed through the BIOS control register (BIOS_CNTL) on the chipset. These protections are far from perfect as there exists a race condition between the BIOSWE bit and the BLE bit [16]. Another mechanism to protect writes is BIOS range write protection registers.

Finally, there exist protected range registers that determine the allowed address ranges for SPI flash chip writes. These range restrictions are enforced for all code (including SMM code) by the chipset and are based on the flash linear addresses (FLAs), ensuring they are consistent with the SPI flash chip memory itself. To prevent an update of these ranges at runtime, the chipset utilizes a flash lockdown (FLOCKDN) bit, that prevents register modification. Once the FLOCKDN is set at boot, the registers are locked and can not be changed until the next device boot. When the chipset write protection is set correctly, it prevents the operating system from updating the UEFI firmware. This protection is not typically desirable in commercial-off-the-shelf systems as the users or administrators need the ability to perform updates through the OS.

In practice, most of the existing SPI write protection introduced has been either been miss-configured in commercial systems or contains vulnerabilities that allowed malware to compromise the UEFI firmware.

There also exist external firmware update tools that bypass all SPI protection mechanisms. For example, server-grade systems typically have a dedicated Baseboard Management Controller (BMC) that enables remote SPI flash updates. Similarly, external programmers can update the firmware through a clamp physically attached directly to the SPI flash.

2.3. Option ROMs

It is common for PCI devices (e.g. video card, network card, etc.) to be initialized from firmware code located outside the motherboard BIOS region. This type of code is commonly referred to as Option ROM and is either embedded inside their dedicated regions of motherboard SPI flash memory (e.g. the GbE region depicted inside Figure 2), or on PCI Expansion ROMs located on the PCI devices themselves.

During the boot process, both types of Option ROM are copied into RAM and executed by firmware code. However, while the embedded Option ROMs integrity is protected by the systems described in 2.2, those stored inside PCI devices are maintained in external EEPROM memory. This allows

device vendors to update PCI device Option ROMs from the host device through a flashing process that is not subject to UEFI firmware write protection.

2.4. UEFI

The Unified Extensible Firmware Interface (UEFI) is the software interface between the operating system and the platform hardware. UEFI has widely been adopted as a replacement for legacy BIOS due to its efficiency and extensive features. While the term BIOS is commonly used to refer to the system firmware, this is a misnomer as UEFI is the modern replacement. There are four primary stages of the UEFI boot process:

Security (SEC). Prepares processor caches to create a temporary memory store for the next phase [17]. The code at this stage, executes from flash memory and is responsible for loading and running the next phase [18].

Pre-EFI Initialization (PEI). Prepares the platform for the later stages, mainly initializing memory and the memory controller [18], as well as preparing an in-memory description of the firmware image for the next stage [17].

DXE. Most of the system initialization is performed in this phase. DXE drivers are responsible for initializing platform components such as device controllers and peripherals. The DXE Core attempts to execute all DXE driver for which the DXE Core can resolve the respective dependencies. All drivers have access to the DXE runtime and boot services, which provide common functionality used by the drivers.

BDS. Discovers and loads the bootloader from the selected device. However, the boot services are torn down and all DXE drivers are cleaned up before transferring control to the bootloader. Only the runtime services remain available for use by the kernel.

2.5. Boot partition

During the BDS phase, the UEFI code loads a bootloader by scanning the connected storage devices for special partitions that contain an EFI System Partition (ESP) label. These ESPP partitions typically have a FAT32 format and are created when an OS is installed. They store both the bootloaders and boot configuration data (BCD) required to boot any of the OSes present on the storage device.

In contrast to firmware stored inside SPI flash, ESP-stored data, and code are always exposed to OS write access and cannot be made read-only before OS boot. Thus, to mitigate attacks that compromise the boot process, UEFI has introduced Secure Boot [9], where on boot UEFI code prevents the loading of untrusted OSes and verifies the bootloader integrity against code hashes maintained inside signature databases stored in non-volatile memory that are in turn verified through a chain of trust rooted in a trusted platform key.

2.6. UEFI variables

UEFI utilizes a key/value storage system that maintains volatile and non-volatile data inside UEFI variables. Volatile variables are maintained inside RAM and are lost during the device reboot. In contrast, non-volatile variables are typically stored inside flash or battery-backed NVRAM memory and persist even after the device loses external power. Thus, UEFI relies on non-volatile variables for storing important device configuration BIOS configuration (e.g. secure boot settings, boot device order, etc.).

During the UEFI boot process, both variable types are accessed by EFI module through a Variable Service that provides read, write, and deletion access to variables identified through their unique GUID used as the key. This service also extends access to the bootloader and OS to the UEFI variables that are either non-volatile or have a runtime access flag set through SMM modules. As a result, the OS and bootloader can read and write a subset of the UEFI variables exposed by UEFI firmware during boot. Importantly, the OS does not have a direct method of reading or writing the memory containing UEFI variables.

3. Problem

As depicted in Figure 1, there are several known attack vectors used by adversaries to deploy persistent rootkits and compromise the device boot process. Such attacks typically start by executing on the targeted system some malicious process that leverages, compromises, or tricks an OS driver into compromising ESP partition data, flashing malicious firmware updates, or altering non-volatile UEFI variables. Regardless of the infection method used, these attacks are designed to persist across system reboots and alter the boot process of a device to compromise the OS prior to its execution.

In recent years multiple instances of alarming rootkit infections have been discovered in the wild. Some compromise devices by targeting either UEFI firmware [1]–[4] as depicted in Figure 1(a), others the ESP disk partition [5]–[7], as shown in Figure 1(b). As a response to these attacks, device manufacturers have deployed an increasingly large number of solutions designed to harden the boot process of modern machines, such as Secure Boot, Intel Boot-Guard [19], and SPI write protection bits. However, an analysis [20] of commercial systems indicates that device vendors often fail to configure them correctly or unknowingly introduce vulnerabilities in their implementation, leaving their devices vulnerable. This is unsurprising given the long, convoluted supply chain involved in the construction of UEFI firmware.

Adversaries also routinely discover new vulnerabilities to exploit within the UEFI boot process, such as the loading of unverified firmware code from PCI Expansion ROMs [21] or vulnerable image processors within UEFI code that can execute attacker-provided image data as shown in Logo-Fail [22]. Rootkits that leverage such vulnerabilities bypass the existing state-of-the-art solutions that focus on code stored within the ESP partition or BIOS region of SPI chips.

Finally, existing malware detection systems (e.g. anti-viruses, intrusion detection systems, memory analysis tools, SPI flash scanners, etc.) might detect the known malicious

processes used to deploy rootkits or find traces of rootkit execution when scanning systems. However, as shown in Section 5 adversary-introduced malware can easily evade detection from such tools through the utilization of obfuscation and usage of legitimate firmware update tools. Further, reliable rootkit detection post-compromise is difficult because the UEFI execution gives the rootkit complete control over the system. This allows the rootkit to scan the system for malware detection tools and prevent them from executing correctly.

Due to the large attack surface and complicated UEFI supply chain of the modern devices UEFI is extremely difficult to deploy software systems that are capable of preventing all the various sophisticated rootkit attacks targeting them. *Thus, it is imperative to also deploy solutions that can reliably identify such attacks and collect their malicious payloads for analysis, before it can execute and completely compromise the monitoring device at the next boot.*

4. Bonanza

Bonanza introduces a novel rootkit collection framework that detects when adversaries compromise the UEFI boot process of virtual honeypots designed to mimic security-sensitive devices and automatically extracts any introduced malicious payload for analysis. This prototype will be made open-source to help improve all the virtual honeypots.

Figure 3 illustrates the components of Bonanza and the process through which Bonanza leverages VMs to automatically capture rootkits and forward them for analysis. This rootkit capturing process starts by Bonanza using the information collected from a real security-sensitive device to construct VMs designed to mimic the hardware (e.g. CPU, memory, storage, etc.) layout. Most importantly, in addition to the UEFI variables and disk storage, Bonanza also emulates and monitors the access to all the device's flash chips, regardless if they are located on the motherboard or PCI device Expansion ROMs as detailed in Section 4.2. Finally, once all non-volatile storage devices are emulated, Bonanza writes inside the UEFI information collected from the target device. This information includes the device-specific UEFI firmware, NVRAM variables, and contents of disk boot partitions.

The monitoring of non-volatile storage access allows Bonanza to immediately detect when the adversary compromises the data or code within. Once such a compromise is detected and logged, the Bonanza storage monitor executing inside the hypervisor waits until the next VM reboot is triggered to ensure all the compromise contents are forwarded to the external rootkit payload extractor described in Section 4.3 for collection. Once collected, the rootkits can be automatically inserted through the process described in Section 4.4 inside the respective UEFI firmware, ESP partitions or NVRAM variables of analysis sandboxes.

Importantly, the utilization of fake flash chips inside virtual honeypots allows Bonanza to easily distinguish between malicious flash reads/writes and the firmware updates that are always performed on the host physical device. There is no legitimate reason for users and legitimate software to modify the contents of the Bonanza-emulated chips.

4.1. Threat model

Bonanza operates under the following assumptions:

1) Adversaries have to infect either the UEFI firmware, Option ROMs, ESP partition, or UEFI variable non-volatile storage to compromise the boot process.
2) Rootkit deployment in non-volatile storage is performed in-band by malicious or compromised software.
3) Adversaries do not detect the hypervisor and Bonanza constructed virtualized components.

Due to the complexity of modern systems, there are numerous ways an adversary can compromise a system. They can exploit vulnerabilities within an OS, trick users into installing a malicious application with administrator privileges, or even introduce malware through supply chain attacks. Bonanza is not designed to handle all such possible attack vectors. Instead, as indicated by the first assumption listed above, the Bonanza framework is specialized in detecting attacks where adversaries attempt to obtain persistent control over a device through the compromise of NVRAM variables or UEFI data or code stored inside SPI chips, Option ROMs, and ESP partitions. While such attacks are more uncommon, they are also one of the most dangerous, as once they are used to deploy rootkits, it can take years to discover their presence, as demonstrated by the discovery of LoJax [1] and MossaicRegressor [2].

Malware that only compromises runtime software applications or the OS is not detected by Bonanza and requires orthogonal detection methods.

The second assumption clarifies that only rootkit injection from within the guest OS is supported, as VMs do not support out-of-band firmware updates from baseboard management controllers (BMCs) and MEs. Similarly, the introduction of rootkits through physical access is also out of scope.

Finally, the third assumption makes it clear that even though Bonanza allows honeypots to be built using real device UEFI information, sophisticated adversaries might detect the virtualization environment used and abort the rootkit deployment. Bonanza emulates crucial components required by honeypots to mimic real devices. However, they still only cover part of the hardware emulation. Ensuring that adversaries cannot identify in any way the virtualized environment is outside the scope of Bonanza.

4.2. Rootkit detection and collection

Bonanza utilizes virtualized high-interaction honeypots created using the Quick Emulator (QEMU) [23] to detect and capture rootkits. This virtualization enables Bonanza to monitor disk and UEFI variable access and provide a full emulation of various SPI flash chips and Option ROMs utilized by modern motherboards and PCI cards to store the UEFI firmware. As a result, Bonanza can efficiently

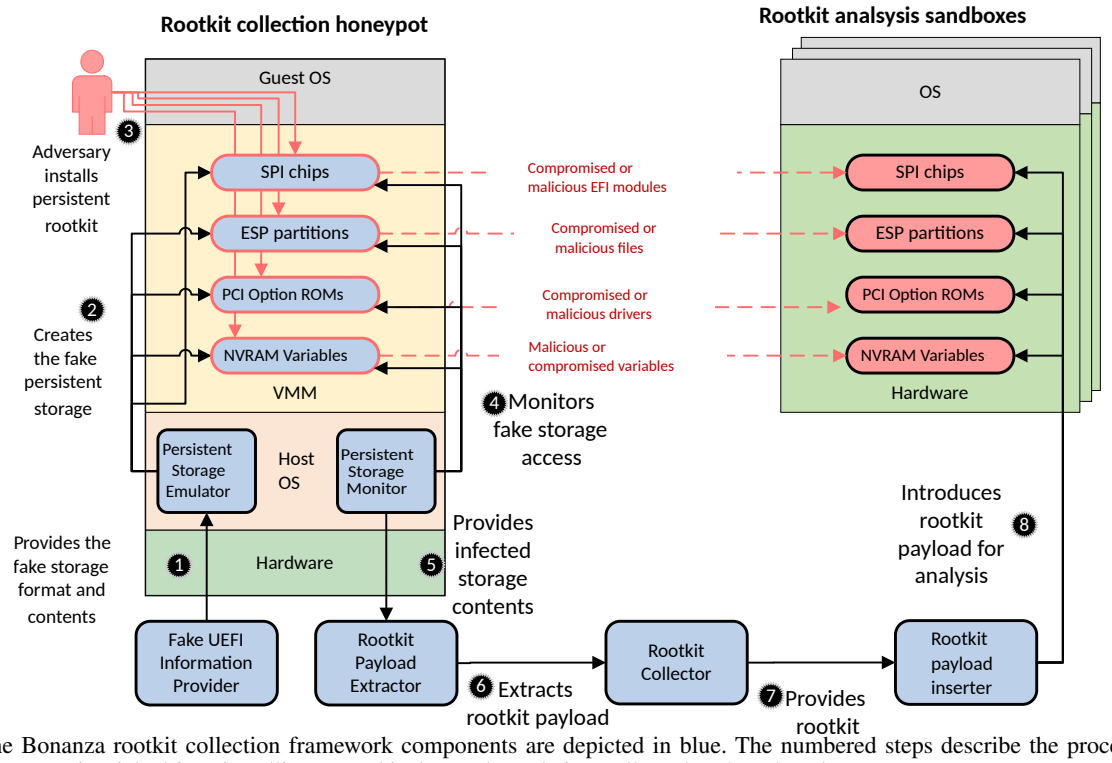

Figure 3: The Bonanza rootkit collection framework components are depicted in blue. The numbered steps describe the process through which an adversary is tricked into installing a rootkit that ends up being collected and analyzed.

deploy various enticing virtual machine (VM) honeypots that expose to adversaries any permutation of hardware configuration and OS.

SPI chip emulation and monitoring. A unique aspect of honeypots running Bonanza is the emulation of fake SPI flash chips. For this, Bonanza introduces in the QEMU hypervisor a virtual SPI device controller that implements the SPI protocol and routes all read/write operations intended for the SPI device to a host-controlled file. This device sits on the virtual system bus, making this configuration OS agnostic and allowing Bonanza to monitor all fake flash chip reads and writes, as well as efficiently instantiate each honeypot with an exact copy of the desired UEFI firmware image independent of the desired OS.

To read and write to an SPI flash from either the kernel or user space, software must first locate the SPI device's configuration registers. On x86 platforms, these registers are memory-mapped within the chipset's configuration space. QEMU implements the Q35 chipset emulation [24] which has a similar PCIe/PCI topology and configuration to many modern Intel chipsets. However, this emulation does not include a comprehensive emulation of typical configuration registers, the SPI device being one of them. As explained in Section 2, The SPI configuration registers are located at the *SPI Base Address Register* (SPIBAR) which is typically located through the *Root Complex Base Address Register* (RCBA) located in the device's memory-mapped chipset configuration space. Bonanza extends the Q35 emulation with these registers and configures the emulated SPI device controller to control the memory addresses corresponding the SPI configuration registers. This chipset-specific information required to correctly configure credible honeypots is extracted by Bonanza from Intel platforms.

Another important detail of SPI emulation in x86 systems is the automatic mapping of the top 16MB of memory to the UEFI firmware during boot before configuring the rest of the system memory. Accesses to this memory region are typically restricted by the operating system and only used during the boot process. However, to ensure the contents inside this memory also match the data exposed through the emulated SPI device, Bonanza also monitors all accesses to this memory region from the guest OS. Note that other SPI flash regions (e.g. ME region, option ROMs, etc.) are not mapped to a similar predefined region and are typically accessed through the previously mentioned configuration registers.

To entice adversaries to infect the Bonanza honeypot, Bonanza exposes the SPI protection registers covered in Section 2.2 and configures them to suggest an unprotected system. Specifically, the chipset configuration registers emulated include *BIOSWE* (BIOS Write Enable), *BLE* (BIOS Lock Enable), and *SMM_BWP*(SMM BIOS Write Protect Disabled), as well as the range protection registers. The register locations are determined depending on the SPIBAR provided and initialized at honeypot deployment. This ensures SPI device registers expected by rootkit droppers match those present on a physical device.

PCI Expansion ROM monitoring. In contrast to UEFI firmware SPI chips, PCI devices such as video and network cards are routinely virtualized inside VMs. Further, these emulated devices already contain Expansion ROMs with

contents loaded from files within the host that are readable from the virtual PCI interface and executed by the Open Virtual Machine Firmware (OVMF) during the VM boot. However, by default writing to these ROMs is not supported, as QEMU does not expect them to be updated from within the VM.

Under Bonanza the QEMU construction of Expansion ROMs is altered to trigger a special hypervisor handler that intercepts all Expansion ROM read and write access from within the VM. The introduced handler ensures that all write access to each ROM is correctly monitored and forwarded to a shadow copy of ROMs maintained inside dedicated Bonanza files on host storage. Further, this process works automatically for all PCI devices that contain Expansion ROMs and does not require additional engineering effort to support new devices.

Importantly, to ensure that any PCI device with Expansion ROMs can be emulated Bonanza also introduces logic inside the OVMF firmware that can identify through device ids the Expansion ROMs of fake PCI devices that are not designed inside the VM. This process allows the emulation of fake ROMs while also preventing their execution from breaking the VM boot process.

UEFI variable monitoring. OVMF already provides full emulation of both volatile and non-volatile UEFI variables and exposes them to the OS through SMM modules. However, it also utilizes its own set of UEFI variables during the OVMF boot process. Thus, the challenge for capturing rootkits targeting UEFI variables lies in replacing the VM-specific ones with those extracted from real devices without impacting the VM boot process.

To ensure only the UEFI variables expected by adversaries are seen inside the VM Bonanza introduces a DXE module inside OVMF that deletes the existing variables and introduces fake ones right before passing execution over to the bootloader. Further, this module also hooks into the variable get and set methods to log all access and save those modified at runtime.

Importantly, Bonanza can automatically process variable information provided by *dmptore*, an utility included in most devices and is accessible from the EFI Shell. Bonanza converts this information into binary data that is then saved inside the OVMF image. During boot, this data is transformed into fake UEFI variables. Since the OVMF image is not visible to adversaries, this process ensures that fake UEFI variables can be easily imported inside honeypots and that their existence remains hidden at runtime.

ESP partition monitoring. The QEMU hypervisor also already provides all the functionality for emulating ESP partitions and monitoring disk modifications. Further, the VM only accesses data inside this partition during boot or when updating the boot process at runtime. Thus, to detect rootkits targeting these partitions Bonanza simply updates the disk monitoring to retain only the disk accesses performed at runtime inside ESP partitions.

VM monitoring. To obtain crucial information required to identify the exact method used to install the rootkit, Bonanza monitors the honeypot's network activity using *Wireshark* [25] and takes a snapshot of the virtual machine every time a process writes to one of the monitored non-volatile storage described. This snapshot can be later analyzed using virtual machine introspection tools (e.g. libVMI [26], DECAF [27], etc.) to detect which process(es) were used by adversaries to compromise the UEFI firmware.

4.3. Rootkit payload extraction

Rootkit modifications to UEFI variable changes are automatically captured individually at runtime by the monitoring module and do not require any special processing. However, the rootkit payloads introduced inside other non-volatile storage require careful extraction to allow their introduction inside other systems for analysis.

For emulated SPI flash chips, Bonanza automatically captures all changes inside UEFI firmware. However, these changes can affect various SPI flash chip regions differently. Thus, to identify the changes made, Bonanza parses the UEFI firmware using the following standard techniques used by open-source tools such as UEFITool [28].

Flash descriptors. For Intel chips, Bonanza uses the flash descriptors included at the beginning of all Intel SPI flash to identify each region within the SPI flash chip and ensure any modifications within the region are correctly reported.

BIOS Region. To analyze the UEFI firmware region, Bonanza leverages the well-known modular structure of a UEFI image. The UEFI firmware region stores the UEFI firmware files in volumes using the Firmware File System (FFS) as shown in Figure 2. Typically, volumes maintain files related to a specific UEFI execution phase, where each file maintains one or more sections holding the data. There are several types of files. However, the most interesting ones from a rootkit perspective are the DXE drivers/applications and PEI modules, which contain one or more sections, including an executable one that runs during the boot process. Other file types include free-form and raw format where the structure is generally unknown and can be thought of as a binary blob.

Using this structure Bonanza converts the UEFI firmware contents into a tree structure. Each tree node corresponds to a volume or file in the firmware. All files, regardless of their type are added to the tree. This process ensures that Bonanza identifies all files affected by the rootkit and captures even the smallest modification that could lead to system compromise. As shown in LogoFail [22], a fine-grained identification of all firmware modifications is required for a correct rootkit payload collection.

Option ROMs. Besides the UEFI firmware, an SPI flash chip can contain several other types of regions that risk being modified, such as embedded Option ROMs. These regions are vendor-specific, and their format is not always well-known. In this case, Bonanza marks the entire region as compromised and reports the differing bytes to notify the user what modifications are present. By checking all regions, Bonanza can identify attacks, such as the one shown at DEF CON [29] where a proof-of-concept rootkit payload was hidden inside the Intel GbE Option ROM region. The same

process is also used for extracting the modified PCI device Option ROMs.

ESP partition. For automatic extraction of modification of files inside the ESP partition, Bonanza saves a copy of the disk partitions prior to VM boot and then compares their contents against those modified during VM execution. Any introduced or modified files are reported as potentially malicious and forwarded for analysis.

4.4. Rootkits preparation for analysis

After extraction, all saved rootkit modifications can be automatically introduced inside analysis sandboxes as depicted in Figure 3. The Bonanza transfer of ESP and UEFI variable modification to other systems is trivial and only involves writing the corresponding malicious or compromised files or UEFI variables or replacing the existing ones. Thus, this section focuses on the subtleties of firmware modification transfer.

For each rootkit extracted from within compromised UEFI firmware images, Bonanza leverages the UEFI format of the target image prepared for analysis to traverse and reproduce all rootkit modifications from within compromised UEFI firmware. For example, any rootkit DXE file that does not exist in the target is introduced in the corresponding DXE volume. Similarly, any file within the target that exists in both images is replaced with the one in the compromised UEFI firmware if it was modified by the rootkit. Importantly, several files with a well-known format (the Apriori and NVRAM variable files) are not replaced completely. These files are processed in more detail to replicate the modifications made by rootkits with greater accuracy. This process ensures that device-specific configuration NVRAM variables that are critical for the boot process and are not altered by rootkits remain intact. Finally, the modified target image can either be an OVMF image used to boot a virtual machine or real UEFI firmware that requires flashing inside a physical machine prepared for monitoring.

By default, the OVMF images are hardware-agnostic files that contain all the modules required to boot an OS. This enables them to emulate most of the UEFI boot process while also being easy to modify. However, any rootkit modifications inside hardware-specific UEFI modules might not be executed correctly, as we have found during the experiments detailed in Section 5.4. Thus, unless all the hardware components of the target physical machine are emulated, only the respective physical machine can ensure the correct execution of all malicious code.

In contrast to OVMF image modification, flashing rootkit-compromised firmware images on real physical devices targeted by the adversaries ensures they execute exactly as intended inside the monitored device. Thus, Bonanza can also automatically insert such firmware inside real physical SPI chips containing BIOS images. For this process Bonanza provides both an in-band flashing method through a Chipsec-inspired SPI flashing process and out-of-band method that can leverage the flashing interface provided either by a BMC or SPI programmer. By providing several methods of automatically flashing compromised firmware images, Bonanza facilitates the deployment of captured rootkit payloads inside various physical devices designed for rootkit analysis.

5. Evaluation

Malware researchers have discovered handful of rootkits which target the UEFI firmware (LoJax [1], MossaicRegressor [2], CosmicStrand [4], MoonBounce [3], VectorEDK [30]), or the ESP boot partition (FinSpy [7], ESPecter [5], BlackLotus [6], and Gluteba [8]), or both (Bootkitty [31]) in the wild through manual device scans. These rootkits have only been discovered post compromise and as a result for most of them the method of infection is not known. However, their EFI or ESP payloads have been analyzed and made public through malware databases, allowing us to leverage them for evaluating Bonanza's rootkit detection effectiveness.

After starting a set of Bonanza-modified QEMU VMs running Windows and Linux OSes, the payloads of all discovered rootkits were installed in the UEFI firmware or ESP boot partition using two rootkit droppers created from open-source tools. These droppers are detailed in Section 5.1. As shown in Table 1 and Table 2, all tested detection tools and Bonanza were successful in identifying the rootkit payloads and prevent their deployment.

A rootkit hardening tool (described in Section 5.2) that automatically creates encrypted versions of the rootkit payloads was also built to evaluate Bonanza detection of more obfuscated rootkits. As shown in Table 1 in contrast to the original payloads, the installation of encrypted payloads evaded detection from almost all tested malware detection tools with the exception of Bonanza. As detailed in Section 5.2, this result is not surprising because Bonanza detects rootkit payloads due to their non-volatile storage monitoring while the other tools tested rely on widely known and well-documented rootkit payload fingerprints for the detection process.

To demonstrate that automatically providing the captured rootkit payloads for analysis is feasible we briefly present in Section 5.3 how rootkit payloads can be automatically introduced inside Bonanza sandboxes that can monitor rootkit execution. In particular, for LoJax and MossaicRegressor rootkits detailed case studies are presented in Section 5.4. These case studies highlight important lessons learned from monitoring rootkit execution inside Bonanza physical and virtual sandboxes.

5.1. Rootkit installation and obfuscation

To compromise tested systems, the custom-built rootkit dropper covertly embeds the rootkit's DXE payload into the UEFI firmware by following these steps. It first reads the current firmware image from the SPI device using CHIPSEC [11]. Then it creates rootkit files by prepending the necessary header information to each DXE payload with the edk2 GenFfs [32] utility. Those prepared files are merged

Rootkit		UEFI firmware scanners					
Name	Version	Windows Defender	Kaspersky firmware scanner	FwHunt	Avast full scan	ESET firmware scanner	**Bonanza**
Lojax	Original	✓	✓	✗	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
MossaicRegressor	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
CosmicStrand	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
MoonBounce	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
VectorEDK	Original	✓	✓	✗	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
Bootkitty	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓

TABLE 1: UEFI rootkit detection

Bootkit		Runtime malware scanners					
Name	Version	Windows Defender	Kaspersky	BitDefender	Avast full scan	ESET	**Bonanza**
FinSpy	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
BlackLotus	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
Bootkitty	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
ESPecter	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
Gluteba	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓

TABLE 2: UEFI bootkit detection

into the extracted UEFI firmware image with UEFITool [28], a common utility for inspecting and editing UEFI images. Finally, the modified malicious firmware image is written back to the SPI device using CHIPSEC.

In addition to the open-source tools utilized, the created UEFI firmware rootkit dropper only contains 292 lines of Python code. This ensures the tool evades detection from malware detection tools and demonstrates that even modest adversaries can easily deploy rootkits through a similarly stealthy deployment mechanism. For example, a new LoJax dropper might avoid detection by replacing the well-known *RWEverything* [12] driver used to infect the SPI chip with CHIPSEC. Thus, it is not enough to simply fingerprint known droppers to prevent UEFI firmware infection.

Similarly, a generic dropper was created for rootkits targeting the ESP partition. After obtaining administrator privileges this dropper 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 [6], this limitation can be removed by exploiting Secure Boot vulnerabilities.

To demonstrate how trivial it is for rootkits to evade detection from existing tools, a simple generic rootkit hardening tool was also created. This tool can encrypt both DXE and ESP payloads using the AES CBC encryption algorithm and automatically create a complementary DXE driver or EFI application that decrypts and executes the payloads during the UEFI boot process. Note, the DXE and EFI decryption modules are not flagged as malicious by current tools because they simply decipher a specified DXE or EFI payload and trigger its execution. This is a legitimate process that could be used by manufacturers to obfuscate proprietary EFI modules and protect them against reverse engineering.

Finally, in the case of NVRAM variables, there is no need for a dedicated tool because both *dmpstore* and OS services directly provide read and write access to UEFI variables. The former was used to ensure Bonanza monitoring identifies and captures the modification of any UEFI variable inside the virtual honeypot.

5.2. Rootkit installation detection

There are several commercial malware scanners that provide tools designed to detect threats in the ROM BIOS, such as Kaspersky's FirmwareScanner [33], ESET firmware scanner [34], Avast [35], and Windows Defender [36].

These tools are proprietary and often incorporated into a larger suite of commercial products designed to work under Windows OS. Table 1 presents a comparison of rootkit detection between these tools, FwHunt [37], and Bonanza. As depicted, most of the scanners detect all original rootkit payloads. This is because all existing firmware scanners utilize a special SPI driver to dump and inspect the UEFI firmware from a booted system. However, only Bonanza can detect the same rootkit payload once encrypted. This is because Bonanza detects any unauthorized modification of firmware stored in SPI chisp and do not rely on proprietary heuristics to detect rootkit installation post compromise.

Bonanza's successful detection of encrypted rootkits is mainly due to its ability to immediately identify malicious changes to the fake UEFI firmware exposed from the monitored emulated SPI chip. In contrast to real systems, where the UEFI firmware could be updated by system administrators (e.g. to support new features, patch vulnerabilities, etc.) between UEFI firmware scans, there is no legitimate reason for users to update the fake firmware exposed inside the Bonanza monitored VMs. Thus, any firmware modifications are most likely malicious and false positives reported by Bonanza can be easily identified by tracking user actions. Both original and encrypted payloads were evident in the files extracted by Bonanza from the compromised firmware and logs produced.

Similarly, as depicted in Table 2 in the case of rootkits stored inside ESP partitions, also known as bootkits, Bonanza and all tested commercial malware scanners (Kaspersky [38], ESET [39], Avast, Windows Defender, and BitDefender [40]) have successfully detected the known bootkit payloads. However, the commercial tools have once again faile to detect any encrypted bootkit payloads. This is due to the same limitations of heuristics and signature scanning. In contrast, the continuous monitoring of the emulated ESP allowed Bonanza to detect all encrypted bootkit payloads.

Importantly, only bootkit infections and ESP logging operations have been observed to write changes to the ESP in our experiments. The latter were trivially flagged when setting up Bonanza monitoring to avoid mistakenly flagging ESP log data as malicious. Any legitimate writes to UEFI such as firmware updates can similarly be flagged as they would have to be approved by the honeypot administrators or performed during honeypot setup. As a result, Bonanza the rootkit detection has not yielded any false positives once Bonanza monitoring was setup.

Online rootkit payload scanning. To demonstrate the importance of analyzing rootkit payloads in Bonanza sandboxes we have also uploaded all the original and encrypted malicious DXE drivers of the six rootkits targeting UEFI firmware contained within SPI chips and four payloads of bootkits that only compromise bootloader files located inside ESP partitions were submitted to five of the most widely-used online file scanning tools (Hybrid Analysis [41], Virus Total [42], MetaDefender Cloud [43], Binarly Risk Hunt [44], and Jotti malware scan [45]). These services utilize dozens of malware detection tools, heuristics, known-bad signatures, and even attempt to execute the payloads inside sandboxes for analysis.

As shown in Table 3, similar to firmware scanners and runtime malware scanners, these tools successfully flag almost all files containing the known original rootkit payloads files as malicious but cannot identify these payloads once encrypted. The VectorEDK rootkit was the only one that did not have all of its DXE drivers flagged. The *NtfsDxe* driver of VectorEDK was not flagged as malicious by any tested online tool because it only implements a legitimate EFI protocol for reading and writing to an NTFS formatted disk. This driver itself does not contain explicitly malicious code but is typically leveraged by rootkits to install malicious files on Windows disks. In contrast, Bonanza still reports the unauthorized introduction of this *NtfsDxe* driver as part of the rootkit deployment attacks.

The evaluation of online file scanners shows that rootkit binaries can easily be fingerprinted and added to malware blacklists. However, the created rootkit encryption tool shows that existing fingerprinting methods can be easily bypassed through various malware obfuscation techniques such as compression, encryption, and obfuscation. Once a rootkit payload was altered through encryption, post compromise file scanning fails to identify introduced modules malicious. In contrast, Bonanza detected and collected all encrypted payloads the moment they were written to the SPI chip or ESP partition respectively. This highlights the importance of introducing Bonanza sandboxes as part of malware detection tests suits in order to be able to identify when even encrypted malicious files attempt to compromise persistent UEFI storage.

While existing malware detection tools could be improved to detect more advanced obfuscated rootkit payloads, the rootkits themselves can also evolve and utilize more advanced techniques to evade new detection methods. However, no UEFI-targeting rootkit can persist without compromising one of the UEFI storages monitored by Bonanza. As a result, Bonanza capturing method remains successful in the face of novel persistent rootkits because it detects the very process of writing payload data either to the fake-SPI chip, OptionROMs, ESP partition, or UEFI variables.

5.3. Automatic rootkit payload execution

To automatically analyze captured rootkits it is necessary to execute them inside sandboxed environments designed to monitor their actions. Thus, we have also investigated the feasibility of automatically introducing the collected rootkits inside virtual and physical sandboxes.

Physical analysis environments. The collected rootkit payloads were flashed inside physical machines from which the fake hardware layout and UEFI information were extracted, ensuring all corresponding hardware was initialized correctly during the boot process and rootkit execution was not altered by virtualization. However, the use of such environments has some obvious drawbacks, most notably much lower limits in terms of scalability and fine-grained rootkit behavior monitoring. For example, a dedicated physical machine has to be set up for each tested OS and SPI chip.

Rootkit/Bootkit		Online file scanners					
Name	Version	Virus Total	Binarly Risk Hunt	MetaDefender Cloud	Hybrid Analysis	Jotti Malware Scan	**Bonanza**
Lojax	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
MossaicRegressor	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
CosmicStrand	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
MoonBounce	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
VectorEDK	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
FinSpy	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
ESPecter	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
BlackLotus	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
Gluteba	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓
Bootkitty	Original	✓	✓	✓	✓	✓	✓
	Encrypted	✗	✗	✗	✗	✗	✓

TABLE 3: Rootkit/bootkit detection by online file scanners

Further, monitoring the rootkit behavior is much harder to track without the control and information readily available by a hypervisor. Finally, the firmware of most modern motherboards is protected against unauthorized alteration by hardware through systems such as Intel BootGuard. Thus, introducing rootkits inside such physical devices would require manufacturer cooperation.

Virtual analysis environments. At first glance executing boot firmware from a real UEFI firmware image in a virtual environment seems infeasible due to the tight coupling with physical hardware. However, a system typically performs all the hardware-specific initialization in the initial UEFI boot phases (SEC and PEI), allowing the later stages (DXE and BDS) to be more hardware-agnostic and rely instead on services provided by UEFI for their device interaction. This process makes the later phases much more enticing for malicious actors, as rootkits discovered in the wild affect only the hardware-agnostic DXE and BDS phases. Of course, nothing prevents a rootkit from targeting code executed in the earlier phases or option ROMs. Thus, for analysis inside virtual sandboxes, the extracted rootkit payloads were inserted inside an Open Virtual Machine Firmware (OVMF) image used to boot a UEFI-compliant OS in a QEMU VM.

By default, the OVMF image contains all the modules required to boot an OS and is hardware-agnostic, enabling the emulation of a UEFI boot process. However, any rootkit modifications inside hardware-specific UEFI modules might not be executed correctly, unless all the hardware components of the target physical machine are emulated. Further, as detailed in Section 5.4, executing a rootkit payload in an environment different from its intended usage can result in failed rootkit execution due to missing dependencies. Thus, it is important to execute each rootkit payload inside a multitude of various virtual and physical sandboxes.

5.4. Case Studies

To demonstrate the importance of using various virtual and physical environments for the capture and analysis of rootkits, we have conducted two case studies that detail the process by which Bonanza captured two existing rootkits for analysis. The first case study detailed in 5.4.1 focuses on the LoJax [1] rootkit, which is the first rootkit discovered in the wild and the only one whose malicious dropper was also discovered in the wild. The second case study focuses on a more recently discovered rootkit known as MossaicRegressor [2] and is presented in 5.4.2. Compared to LoJax, MossaicRegressor is more complex as it uses multiple malicious modules, some very similar to previously leaked VectorEDK components.

These two studies show that minor UEFI environment details can affect rootkit behavior, causing their malicious UEFI modules to even fail to execute on different hardware or software stacks. For example, the case studies detail important, previously unknown rootkit UEFI firmware and OS dependencies that have been discovered when executing the Bonanza collected rootkits. These dependencies demonstrate the importance of utilizing various device hardware and software configurations for monitoring and analyzing rootkit capture and execution.

For example, initially, the LoJax-introduced DXE module failed to execute completely inside VMs due to a previ-

OS \ Rootkit	LoJax	MossaicRegressor
Windows 7	✗	✓
Windows 8	✓	✗
Windows 10	✓	✗
Windows 11	✗	✗

TABLE 4: Rootkit OS Infection

ously unknown OVMF TPL incompatibility. The process of discovering this OVMF incompatibility is detailed in 5.4.1.

Once OVFM was patched to fix the LoJax incompatibility, another previously unknown important OS dependency was discovered when executing the LoJax payload in multiple virtual analysis sandboxes. As depicted in Table 4, LoJax successfully executed and infected the virtual sandboxes running Windows 8 and 10 after fixing the OVMF TPL incompatibility. However, the other sandboxes running Windows 7 and 11 were not successfully compromised due to file structure differences between the various OS versions. In short, LoJax was not designed to work on Windows 7 and 11 boot partitions. Similarly, it quickly became evident that the LoJax payload is designed only to modify NTFS-formatted disks and thus cannot be used to compromise Unix systems that typically utilize FAT partitions to store bootloaders.

To confirm the OVMF and OS compatibility issues described LoJax was also executed inside physical analysis sandboxes. Here, there were no TPL incompatibility issues, but LoJax still only successfully infected devices running Windows 8 and 10.

In the case of MossaicRegressor, no OVMF incompatibility issues were discovered. Once deployed, the MossaicRegressor was able to compromise the virtual Windows 7 sandbox. However, when tested on other OSes (Windows 8, 10, and 11), the introduction of MossaicRegressor malicious modules inside the UEFI firmware caused them to fail to boot. By analyzing the collected logs, it became evident that the *SmmAccessStub* module introduced by MossaicRegressor expects to modify an OS file structure specific to Windows 7. When VMs with a different file structure were utilized, the discrepancy caused a bug in the OVMF memory allocation. This bug introduced by MossaicRegressor, caused the VMs to halt execution when running Windows 8, 10, or 11. This behavior was also reproduced inside physical machines by booting the respective OSes from an MossaicRegressor-infected firmware and inspecting the logs. This demonstrates that the incompatibility observed is caused by different OS file structures and not an OVMF incompatibility problem.

Importantly, sophisticated adversaries can introduce other rootkit dependencies, similar to the ones discovered, which cause rootkits to not execute inside virtual and real devices that utilize a hardware or software layout that is different from their target. *Thus, it is imperative that both virtual and physical honeypots closely mimic both the hardware and software stack of security-sensitive machines to capture sophisticated rootkits.*

5.4.1. LoJax capture. To evaluate Bonanza detection of LoJax, a QEMU-based virtual rootkit collection honeypot and two types of rootkit analysis sandboxes (virtual and physical) were used. The LoJax rootkit dropper is a Windows Executable File, making a Windows OS the natural choice for the collection honeypot. For the analysis sandboxes, although the dropper was designed to target Windows OSes, Bonanza's automatic extraction of rootkit payloads made it easy to test its execution under different OSes. Thus, both Unix-based and Windows OSes sandboxes were instantiated by executing the captured LoJax payloads. Note, the virtual analysis sandboxes were QEMU-based, while the physical ones were set up on Supermicro X10SDV-4C-TLN2F devices.

The rootkit infection process was triggered by executing the LoJax dropper inside a Windows 8-based rootkit collection honeypot configured with a copy of the X10SDV-4C-TLN2F UEFI firmware image. To ensure the complete original LoJax infection process was monitored, the original LoJax dropper was chosen instead of the obfuscated version of LoJax presented in Section 5.2. Initially, this dropper was detected by Windows Defender and failed to execute. However, after disabling the antivirus, the dropper successfully introduced a malicious DXE driver labeled *SecDxe* inside the DXE volume of the UEFI firmware.

Once the dropper started writing the *SecDxe* driver, the Bonanza monitor detected a set of writes to the fake X10SDV-4C-TLN2F SPI chip. At this point, the monitor saved a snapshot of the honeypot for analysis. Once the SPI write process finished, the monitor forwarded the compromised UEFI firmware to the rootkit extractor. This process showed that the LoJax dropper was successfully tricked into compromising the fake X10SDV-4C-TLN2F SPI chip.

The rootkit extractor successfully identified the new malicious *SecDxe* driver by parsing the LoJax compromised UEFI firmware. Then, this driver was automatically inserted by the Bonanza rootkit payload inserter into the UEFI firmware (OVMF) images. The LoJax-infected OVMF images were then used to boot the VMs created for rootkit analysis. Interestingly, these VMs failed to execute from this OVMF image once it contained the malicious *SecDxe* driver. To understand why the infected boot process failed, the OMVF execution logs were collected and analyzed.

The logs revealed that the honeypot's boot process was broken by a callback registered by the *SecDxe* driver. During the DXE execution phase the execution order follows a depth-first tree traversal which results in *SecDxe* being invoked by the DXE Core using the *StartImage* protocol used during the standard DXE phase. Upon executing, *SecDxe* registered a callback function to be called during the `ReadyToBoot` event. This event occurs when UEFI runtime is ready to transfer control to the OS. Upon executing the callback function, DXE Core halted.

Further examination of the OMVF execution logs revealed that the incompatibility between the OVMF UEFI runtime and the *SecDxe* was caused by a misconfiguration in task priority level (TPL). Specifically, the OVMF code expected the TPL to be lower than the value present during

the execution of the LoJax introduced callback, which makes a call to *StartImage*. However, this value was too high, causing OVMF to halt the VM execution. Normally, this behavior would ensure that tasks with lower priority (lower TPL) cannot interrupt those with higher priority. However, in this case, OVMF was also preventing LoJax code from being executed because it would have interrupted a higher-priority task.

To fix the incompatibility issue, OVMF was patched to lower the TPL level before LoJax code execution. This trivial patch allowed *SecDxe* to execute and OVMF to complete the boot process. As a result, virtual rootkit analysis sandboxes could successfully execute the LoJax payload and monitor its execution at runtime.

The Bonanza collected information confirmed LoJax infection details covered in the discovery report [1] that details the results of detailed manual analysis. For example, the LoJax dropper was identified as the source of malicious SPI writes by using DECAF [27] and analyzing the honeypot snapshot containing the saved rootkit. Through binary reverse engineering, the LoJax dropper was confirmed to be using the RWEverything driver to bypass the software SPI write protection bits and read and write data to the SPI device. Similarly, the *SecDxe* introduction and its compromise of the boot process was evident in the logs collected from the VM.

5.4.2. MossaicRegressor capture. For the capture of MossaicRegressor, the infection of the UEFI firmware was activated by running the generic dropper detailed in Section 5.1 with the payload MossaicRegressor inside a honeypot for the collection of Windows 8 rootkits containing a copy of the X10SDV-4C-TLN2F. The generic dropper was used because the MossaicRegressor dropper has not yet been discovered. Similarly to LoJax, the known MossaicRegressor payload is flagged as malicious by most malware detection tools. Thus, Windows Defender had to be disabled to deploy the original, non-obfuscated MossaicRegressor payload.

Once the Windows Defender was disabled, the dropper successfully inserted the MossaicRegressor payload, consisting of two DXE drivers (*SmmInterfaceBase* and *Ntfs*) and two DXE applications (*SmmReset* and *SmmAccessStub*) into the fake SPI chip. The SPI infection was detected by the Bonanza monitor, allowing the automatic extraction of the payload through the same process described for LoJax.

Next, the malicious payload was automatically inserted into the OVMF image of the same virtual and physical rootkit analysis environments used for LoJax (detailed in 5.4.1). However, in contrast to LoJax, the malicious DXE modules introduced by MossaicRegressor did not violate the TPL task execution order and did not require virtual environment modifications to execute.

From the OVMF execution logs collected from the Windows 7 VM it was evident that MossaicRegressor utilized a *SmmInterfaceBase* driver that registered a new callback to the UEFI `ReadyToBoot` event. This callback then started the execution of a *SmmAccessStub* application. Once executed, this application used the *Ntfs* driver to locate the disk that contains the Windows installation and introduced a malicious file into its "\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup' directory. As detailed in Kaspersky's report [2], this file corresponds to a malicious "IntelUpdate.exe" binary that is embedded within the *SmmAccessStub* itself. At OS boot, this binary was automatically executed. Note, the *SmmReset* that is supposed to write a "fTA" NVRAM variable was not invoked and thus the respective NVRAM variable was also not written.

6. Discussion

This section explores limitations of the Bonanza framework prototype and shares the directions of future work.

Legacy BIOS. The current Bonanza prototype uses systems and honeypots that rely on UEFI, the predominant boot standard which has largely replaced the legacy BIOS. Honeypots that use a legacy BIOS can also be deployed to capture rootkits targeting this older boot mode, however, the non-standardized process makes this a significant engineering effort. Note, since 2011 more than 80% of all PCs and servers worldwide run UEFI firmware according to the UEFI Forum [46].

Physical rootkit collection honeypots. The hypervisor level present in virtual honeypots enables Bonanza to easily introduce hooks that monitor all SPI operations and detect a malicious process the moment it tries to infect the UEFI firmware. In a physical device, any monitoring from within the device can be easily circumvented by sophisticated malware that can compromise the OS. Thus, rootkit detection inside physical devices requires additional hardware that can access the SPI chip out-of-band. A possible direction of future work uses out-of-band monitors on physical machines such as BMCs or SPI programmers to detect UEFI firmware modifications.

Unsupported emulation. Sophisticated droppers may attempt to leverage system vulnerabilities inside hardware that are not emulated by QEMU or Bonanza (e.g., S3 boot attack). However, researchers who uncovered the Trickboot dropper confirm that the malware takes the easiest approach first method before trying more complex ways of writing to the SPI memory device [47]. Similarly, researchers at ESET Security [1] determined that the LoJax rootkit was installed using the RWEverything driver that first checks the flags to see if there is an easy path to install the rootkit before exploiting an SMM vulnerability. Therefore, Bonanza's approach to unlocking the SPI flash in a virtual environment is robust against adversaries attempting to install rootkits as quickly and efficiently as possible.

Mitigating Bonanza honeypot detection. A sophisticated malware may attempt to detect the presence of Bonanza components on their targeted systems and try to elude detection by not compromising the UEFI firmware of Bonanza honeypots. Thus, to remain stealthy, the Bonanza honeypot design has focused on minimizing its monitoring footprints and any discrepancies between real hardware components and their emulated counterparts. First Bonanza ensures that

OVMF details are removed and all the visible UEFI information is identical to the one provided by the emulated physical machines. Second, Bonanza minimizes monitoring detection by only performing it from within the hypervisor or UEFI firmware modules that are removed before OS boot. Third, Bonanza introduces fake SPI and PCIE hardware that emulates completely all registers and hardware operations provided by the real devices. These measures ensure that malware cannot utilize hardware or UEFI information to detect the honeypot presence. Additionally, other standard honeypot anti-detection techniques should be introduced in Bonanza honeypots to maximize their stealthiness. For example, Bonanza should emulate the entire hardware and software stack to closely resemble the desired security-sensitive device. Similarly, typical side-channels (e.g. processor execution time, network operations, etc.) used to detect execution inside a VM have to be eliminated.

Finally, hardening honeypots against detection represents a continuous battle against malware developers and requires a continuous effort to deploy solutions that mitigate novel honeypot detection vectors discovered. There are numerous other solutions and methods [48]–[53] that have been proposed and utilized to prevent detection of the virtualized environment.

Some of the methods to evade malware detection can be trivial to introduce, such as ensuring none of the known artifacts of VMs, such as the 100 flags listed by VWAware [54], are present in honeypots utilizing Bonanza. Others required more complex solutions to hide VM virtualization artifacts. For example, to evade detection from malware that utilizes execution timing techniques for VM detection [55] proposes the utilization of a virtual clock source that allows the acceleration of sandboxes such as Bonanza that run on top of a hypervisor.

To minimize the monitoring detection risk, additional standard honeypot anti-detection techniques should be introduced in deployed Bonanza honeypots. It is important to note that the discovered rootkit typically do not contain complex logic and simply use existing tools such as the RWEverything driver to test whether the UEFI firmware can be altered and perform the required read and write operations. Bonanza can detect such droppers as both the SPI flash operations and the bits used to protect it are completely emulated inside the virtual environments.

Non-persistent rootkit detection. Of course, not all malware operates through the attack vectors presented in Section 3 that include compromising the UEFI firmware or ESP partition. For example, non-persistent rootkits that only compromise the kernel memory are not detected by honeypots that only deploy the presented Bonanza framework and do not deploy additional malware detection components that monitor the guest OS memory. Such rootkits can be detected by deploying additional monitoring systems such as [56] inside the hypervisor. However, it is important to note that introducing additional monitoring components also increases the risk of honeypot discovery due to additional performance overhead.

7. Related Work

Rootkit detection. Malware scanners often fail to detect rootkits because rootkits are specifically designed to operate stealthily and evade standard detection mechanisms. Unlike typical malware, rootkits can manipulate the boot process and compromise the operating system in order to hide processes, files, or registry entries, and even intercept system calls before the conventional scanners start executing. Further, most malware scanners rely on signature-based detection, heuristics [57]–[59], or model-checking [60]–[62]. As shown in the Bonanza evaluation, such approaches can identify known threats but not sophisticated rootkits can alter their behavior or structure to avoid triggering these checks. Similar to [**?**], Bonanza can help identify new rootkits by deploying honeypots capable of running virtual machines that can immediately identify and track any unauthorized change done to the UEFI boot process in order to collect and analyze novel rootkit behavior.

Firmware integrity. Modern devices incorporate firmware integrity mechanisms such as Intel Boot Guard [19] and AMD Platform Secure Boot (PSB) [63], which rely on public keys embedded in one-time programmable fuses to block the execution of unauthorized firmware. Despite their widespread deployment, these protections are not foolproof. Prior work by Matrosov [64] has shown that errors and oversights in complex supply chains can leave firmware unprotected, undermining the intended security guarantees. In addition, multiple vulnerabilities have been identified in Intel Boot Guard [65]–[67] that allow attackers to run unsigned firmware, while the keys used to verify firmware signatures at boot are occasionally exposed [68].

Moreover, firmware loaded from OptionROMs is generally verified only during the boot process, and this process can be disabled or bypassed, creating unmonitored attack surfaces. Documented bypasses of secure-boot protections highlight additional weaknesses in current protection schemes [69]–[71]. These limitations point to a critical research gap: hardware enforced integrity-based mechanisms underscores the need for complementary approaches, such as Bonanza and other runtime monitoring and rootkit detection systems that can identify and mitigate firmware-level threats that evade boot-time protections.

Boot process verification. UEFI Secure Boot is a well-known and widely deployed mechanism designed to ensure that only trusted, signed bootloaders and OSes are executed on a security-sensitive device. The Secure Boot trust is always anchored in the Platform Key stored inside UEFI firmware, which verifies the valid bootloader hashes against a whitelist. Thus, Secure Boot does not protect UEFI firmware against compromise and requires trusted firmware components to verify the bootloader and OS. Note that savvy adversaries can also stealthily disable Secure Boot entirely or modify the keys and signatures to completely bypass its protection [72]. Further, verification of the firmware only takes place from the UEFI Boot Manager onwards, leaving much of the earlier boot firmware vulnerable. In contrast, Bonanza monitoring of both UEFI firmware and ESP files

ensures that rootkits such as BlackLotus are always captured and can help identify quickly how such malware bypasses Secure Boot protection.

Firmware analysis. Research has also focused on hardening the firmware code itself against compromise. For example, the following systems focused on discovering embedded and IoT devices firmware vulnerabilities have been proposed. First, Avatar [73] is a framework that enables dynamic analysis of embedded firmware by running the respective binary in an emulator and forwarding I/O accesses to the physical device, similar to Bonanza. Next, the Firmadyne [74] software-based firmware emulation framework can execute embedded firmware, collect runtime information, and help identify vulnerabilities within. FirmAE [75] further improves the Firmadyne proposed framework by eliminating some of the device configuration limitations required for successful firmware emulation. However, note that UEFI firmware has a completely different structure compared to the embedded firmware composed of a Linux kernel and file system. Each UEFI firmware vendor has its variant of core UEFI components (e.g., DXE Core, PEI Core, etc.) and contains additional proprietary components. Thus, embedded firmware emulation approaches are not suited to monitor the various attack vectors utilized by rootkits in fully-fledged systems.

Firmware honeypots. Naturally, Bonanza is not the first effort to employ specialized monitoring environments for uncovering low-level compromise attempts. Provos et al. [76] is one of the earliest work to introduce the concept of honeypots. However, unlike our approach that concentrates on detecting UEFI boot process manipulation through emulation of UEFI components, most prior work such as Briffaut et al. [77], Pauna [78], and Vetterl et al. [79] have focused on leverage virtualization to create high-interaction honeypots capable of observing compromises of runtime software. These approaches primarily target application and network layer attacks and are not designed to expose malware activity that only occurs during the UEFI boot process.

Comprehensive surveys such as Bringer et al. [80], Campbell et al. [81], and Franco et al. [82] provide in-depth overviews of honeypot deployments, including those in embedded ecosystems, further illustrating both the breadth of the space and the need for mechanisms capable of observing attacks at the firmware layer.

Open-source UEFI firmware. Finally, there has been a big push in recent years to replace all or a portion of firmware components with open-source alternatives such as Coreboot [83] and Linuxboot [84]. While such operations can help eliminate some vulnerabilities hidden inside proprietary code, they do not prevent the introduction of rootkits inside UEFI firmware. Vulnerabilities present inside the open-source code can provide additional attack vectors for adversaries. Thus, systems such as Bonanza are required to detect and mitigate such attacks.

8. Conclusion

Increasingly persistent and stealthy UEFI rootkits have been discovered on commodity machines. Most often their incubation periods and origin are completely unknown and they can remain undetected for years. It is essential to provide tools that defend against these new attack vectors.

Bonanza is one such tool. It lures rootkits into compromising the UEFI firmware of seemingly vulnerable honeypots and then automatically extracts their malicious payload. This payload can later be easily executed inside many sandboxed environments that are designed to automatically analyze how the rootkit affects different devices and operating systems. To detect and capture a variety of rootkits Bonanza exposes ”misconfigured” fully virtualized UEFI firmware flash chips, PCI Expansion ROMs and monitors access to all non-volatile storage used to store UEFI data, including UEFI variables and the disk ESP partitions. This monitoring enables Bonanza to detect rootkit deployments inside virtual environments with minimal modification to their hypervisor.

Experiments indicate that Bonanza can reliably detect even the deployment of rootkits hardened through encryption and can provide their malicious payload for analysis. Further, these experiments also show that invaluable rootkit information can be learned by capturing and analyzing the behavior of rootkits found in the wild.

9. Ethics considerations

All rootkit binaries evaluated are publicly available and have been widely analyzed and documented. As demonstrated in the Bonanza evaluation, all tested malware detection tools can detect them in the wild. Further, these tools also detect all the known rootkits if the generic rootkit dropper introduced in 5.1 is used.

The rootkit dropper itself simply automates the utilization of publicly available tools to modify and flash UEFI firmware. It does not exploit any vulnerabilities or introduce new risks to systems. In contrast, the rootkit hardening tool described in Section 5.1 could be used to avoid detection by malware detection tools. Thus, to not facilitate the deployment of rootkits in the wild, the creation of the decryptor module of the rootkit hardening tools will not be made public.

The rootkit hardening process is useless without the decrypting module, because the rootkits encrypted will never execute, even if they were introduced successfully inside a UEFI firmware. This ensures that anybody can reproduce the evaluation section in regards to encrypted rootkit payload detection, but they cannot easily utilize the presented tools to directly compromise systems. Note, adversaries that can construct the decryption modules themselves can easily construct their own tools or directly utilize the open source to perform attacks. They do not need the tools or information presented for malicious attacks.

References

[1] ESET, "Lojax first uefi rootkit found in the wild, courtesy of the sednit group," Tech. Rep., 2018.

[2] M. Lechtik, I. Kuznetsov, and Y. Parshin, "Mosaicregressor: Lurking in the shadows of uefi," Kaspersky, Tech. Rep., 2020.

[3] M. Lechtik, V. Berdnikov, D. Legezo, and I. Borisov, "Moonbounce: the dark side of uefi firmware," Kaspersky, Tech. Rep., 2022.

[4] G. R. . A. Team, "Cosmicstrand: the discovery of a sophisticated uefi firmware rootkit," Kaspersky, Tech. Rep., 2022.

[5] M. Smolár and A. 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.

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

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

[8] L. Rochberger and D. Yashnik, "Diving into glupteba's uefi bootkit," Palo Alto Networks, Tech. Rep., 2024.

[9] R. Wilkins and B. Richardson, "Uefi secure boot in modern computer security solutions," in *UEFI forum*, 2013, pp. 1–10.

[10] C. Sanders and J. Smith, "Chapter 12 - using canary honeypots for detection," in *Applied Network Security Monitoring*, C. Sanders and J. Smith, Eds. Boston: Syngress, 2014, pp. 317–338. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012417208100012X

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

[12] "Rweverything - read & write everything," http://rweverything.com/.

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

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

[15] X. Kovah and C. 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.

[16] C. Kallenberg and R. Wojtczuk, "Speed racer: Exploiting an intel flash protection race condition," *Bromium Labs (January 2015)*, 2015.

[17] *UEFI Platform Initialization Specification*, UEFI Forum, Inc., Mar 2023.

[18] A. Matrosov, E. Rodionov, and S. Bratus, *Rootkits and bootkits: reversing modern malware and next generation threats*. No Starch Press, 2019.

[19] J. Frazelle, "Securing the boot process: The hardware root of trust," *Queue*, vol. 17, no. 6, p. 5–21, feb 2020. [Online]. Available: https://doi.org/10.1145/3380774.3382016

[20] A. Marchand, Y. Imine, H. Ouarnoughi, T. Tarridec, and A. Gallais, "Firmware integrity protection: A survey," *IEEE Access*, 2023.

[21] T. Hudson and L. Rudolph, "Thunderstrike: Efi firmware bootkits for apple macbooks," in *Proceedings of the 8th ACM International Systems and Storage Conference*, 2015, pp. 1–10.

[22] F. Pagani and A. 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.

[23] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41. Califor-nia, USA, 2005, p. 46.

[24] M. Apfelbaum, "Q35 - qemu," https://wiki.qemu.org/images/4/4e/Q35.pdf.

[25] "Wireshark," https://www.wireshark.org.

[26] H. Xiong, Z. Liu, W. Xu, and S. Jiao, "Libvmi: A library for bridging the semantic gap between guest os and vmm," in *2012 IEEE 12th International Conference on Computer and Information Technology*, 2012.

[27] A. Henderson, L. K. Yan, X. Hu, A. Prakash, H. Yin, and S. McCamant, "Decaf: A platform-neutral whole-system dynamic binary analysis platform," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 164–184, 2016.

[28] N. Schlej, "Uefitool," https://github.com/LongSoft/UEFITool, 2023.

[29] M. Shkatov and J. Michael, "Bytes in disguise," https://www.youtube.com/watch?v=KDo3CExd8Ns, Aug 2020.

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

[31] J. Lee, J. Kwon, H. Seo, M. Lee, H. Seo, J. Jung, and H. Koo, "Bootkitty: A stealthy bootkit-rootkit against modern operating systems," in *19th USENIX WOOT Conference on Offensive Technologies (WOOT 25)*, 2025, pp. 303–320.

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

[33] Kaspersky, "Anti-rootkit and remediation technology," https://www.kaspersky.com/enterprise-security/wiki-section/products/anti-rootkit-and-remediation-technology.

[34] ESET, "Uefi scanner," https://help.eset.com/glossary/en-US/technology_uefi.html.

[35] Avast, "Scanning for viruses with avast one," https://support.avast.com/en-us/article/one-virus-scan/#pc.

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

[37] Binarly, "Binarly releases fwhunt.run project to scale firmware threat detection," https://www.binarly.io/news/binarly-releases-fwhunt-run-project-to-scale-firmware-threat-detection.

[38] Kaspersky, "Kaspersky antivirus," https://www.kaspersky.com.

[39] ESET, "Eset antivirus," https://www.eset.com.

[40] Bitdefender, "Bitdefender antivirus," https://www.bitdefender.com/en-us/consumer/antivirus.

[41] "Hybrid analysis," https://www.hybrid-analysis.com.

[42] Google, "Virustotal," https://www.virustotal.com.

[43] OPSWAT, "Metadefender cloud," https://www.hybrid-analysis.com.

[44] Binarly, "Binary risk hunt," https://risk.binarly.io/.

[45] "Jotti malware scan," https://virusscan.jotti.org.

[46] "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.

[47] I. Eclypsium, "Trickbot now offers 'trickboot': Persist, brick, profit," 2020.

[48] H. Shi, J. Mirkovic, and A. Alwabel, "Handling anti-virtual machine techniques in malicious software," *ACM Trans. Priv. Secur.*, vol. 21, no. 1, Dec. 2017. [Online]. Available: https://doi.org/10.1145/3139292

[49] Y. B. Lee, J. H. Suk, and D. H. Lee, "Bypassing anti-analysis of commercial protector methods using dbi tools," *IEEE Access*, vol. 9, pp. 7655–7673, 2021.

[50] M. Sato, H. Taniguchi, and R. Nakamura, "Virtual machine monitor-based hiding method for access to debug registers," in *2020 Eighth International Symposium on Computing and Networking (CANDAR)*, 2020, pp. 209–214.

[51] M. Mohammadzad and J. Karimpour, "Using rootkits hiding techniques to conceal honeypot functionality," *Journal of Network and Computer Applications*, vol. 214, p. 103606, 2023.

[52] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proceedings from the sixth annual IEEE SMC information assurance workshop*. IEEE, 2005, pp. 29–36.

[53] S. Mukkamala, K. Yendrapalli, R. Basnet, M. Shankarapani, and A. Sung, "Detection of virtual environments and low interaction honeypots," in *2007 IEEE SMC Information Assurance and Security Workshop*. IEEE, 2007, pp. 92–98.

[54] Louis, "Vmaware," https://github.com/kernelwernel/VMAware.

[55] C.-H. Lin, H.-K. Pao, and J.-W. Liao, "Efficient dynamic malware analysis using virtual time control mechanics," *Computers & Security*, vol. 73, pp. 359–373, 2018.

[56] I. Korkin and S. Tanda, "Detect kernel-mode rootkits via real time logging and controlling memory access," 05 2017.

[57] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, pp. 323–334, 11 2008.

[58] W. Arnold and G. Tesauro, "Automatically generated win32 heuristic virus detection," in *Proceedings of the 2000 international virus bulletin conference*, 2000.

[59] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. 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*. Springer, 2007, pp. 468–477.

[60] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Proactive detection of computer worms using model checking," *IEEE transactions on dependable and secure computing*, vol. 7, no. 4, pp. 424–438, 2008.

[61] ——, "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*. Springer, 2005, pp. 174–187.

[62] A. Holzer, J. Kinder, and H. Veith, "Using verification technology to specify and detect malware," in *International Conference on Computer Aided Systems Theory*. Springer, 2007, pp. 497–504.

[63] J. 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.

[64] A. 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.

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

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

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

[68] L. Abrams, "Intel investigating leak of intel boot guard private keys after msi breach," May 2023. [Online]. Available: https://www.bleepingcomputer.com/news/security/intel-investigating-leak-of-intel-boot-guard-private-keys-after-msi-breach/

[69] M. Smolár, "Blacklotus uefi bootkit: Myth confirmed," 2023.

[70] ——, "Under the cloak of UEFI Secure Boot: Introducing CVE-2024-7344," https://www.welivesecurity.com/en/eset-research/under-cloak-uefi-secure-boot-introducing-cve-2024-7344/, WeLiveSecurity, 2025.

[71] B. REsearch, "Another Crack in the Chain of Trust: Uncovering (Yet Another) Secure Boot Bypass," https://www.binarly.io/blog/another-crack-in-the-chain-of-trust, Binarly, 2025.

[72] R. Lakshmanan, "Researchers uncover uefi secure boot bypass in 3 microsoft signed boot loaders," https://thehackernews.com/2022/08/researchers-uncover-uefi-secure-boot.html, Aug 2022.

[73] J. Zaddach, L. Bruno, A. Francillon, D. Balzarotti *et al.*, "Avatar: A framework to support dynamic security analysis of embedded systems' firmwares." in *NDSS*, vol. 14, 2014, pp. 1–16.

[74] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for linux-based embedded firmware." in *NDSS*, vol. 1, 2016, pp. 1–1.

[75] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis," in *Annual computer security applications conference*, 2020, pp. 733–745.

[76] N. Provos *et al.*, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, no. 2004, 2004, pp. 1–14.

[77] J. Briffaut, J.-F. Lalande, and C. Toinard, "Security and results of a large-scale high-interaction honeypot." *J. Comput.*, vol. 4, no. 5, pp. 395–404, 2009.

[78] A. Pauna, "Improved self adaptive honeypots capable of detecting rootkit malware," in *2012 9th International Conference on Communications (COMM)*, 2012, pp. 281–284.

[79] A. Vetterl and R. Clayton, "Honware: A virtual honeypot framework for capturing cpe and iot zero days," in *2019 APWG symposium on electronic crime research (eCrime)*. IEEE, 2019, pp. 1–13.

[80] M. L. Bringer, C. A. Chelmecki, and H. Fujinoki, "A survey: Recent advances and future trends in honeypot research," *International Journal of Computer Network and Information Security*, vol. 4, no. 10, p. 63, 2012.

[81] R. M. Campbell, K. Padayachee, and T. Masombuka, "A survey of honeypot research: Trends and opportunities," in *2015 10th international conference for internet technology and secured transactions (ICITST)*. IEEE, 2015, pp. 208–212.

[82] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.

[83] R. Minnich, "coreboot," https://github.com/coreboot/coreboot.

[84] ——, "Linuxboot," https://github.com/linuxboot/linuxboot.