GGSEC RESEARCH · WHITEPAPER · JUNE 2026

UEFI Attack Surface Research

Bootkits · Secure Boot bypasses · Pre-OS exploitation vectors · Firmware rootkit detection · SMM · DXE

Author: Maciej Gojny **Organization:** GG Advanced IT Security **Web:** ggsec.de

Classification: TLP:WHITE – Public

01 Executive Summary

The Unified Extensible Firmware Interface (UEFI) has replaced legacy BIOS as the industry-standard firmware interface for x86, x64, and ARM platforms. While UEFI brings significant security improvements — Secure Boot, cryptographic verification, and a standardized architecture — it also introduces a dramatically expanded attack surface.

UEFI firmware executes before the operating system, before any security agent loads, and before most hardware-based attestation begins. A compromise at this layer provides the attacker with:

- **Persistence** that survives OS reinstalls, disk replacements, and factory resets
- **Stealth** invisible to all OS-level security tools (EDR, AV, HIDS)
- **Privilege** equivalent to hypervisor or System Management Mode (SMM)
- **Control** over boot process, Secure Boot policy, and hardware initialization

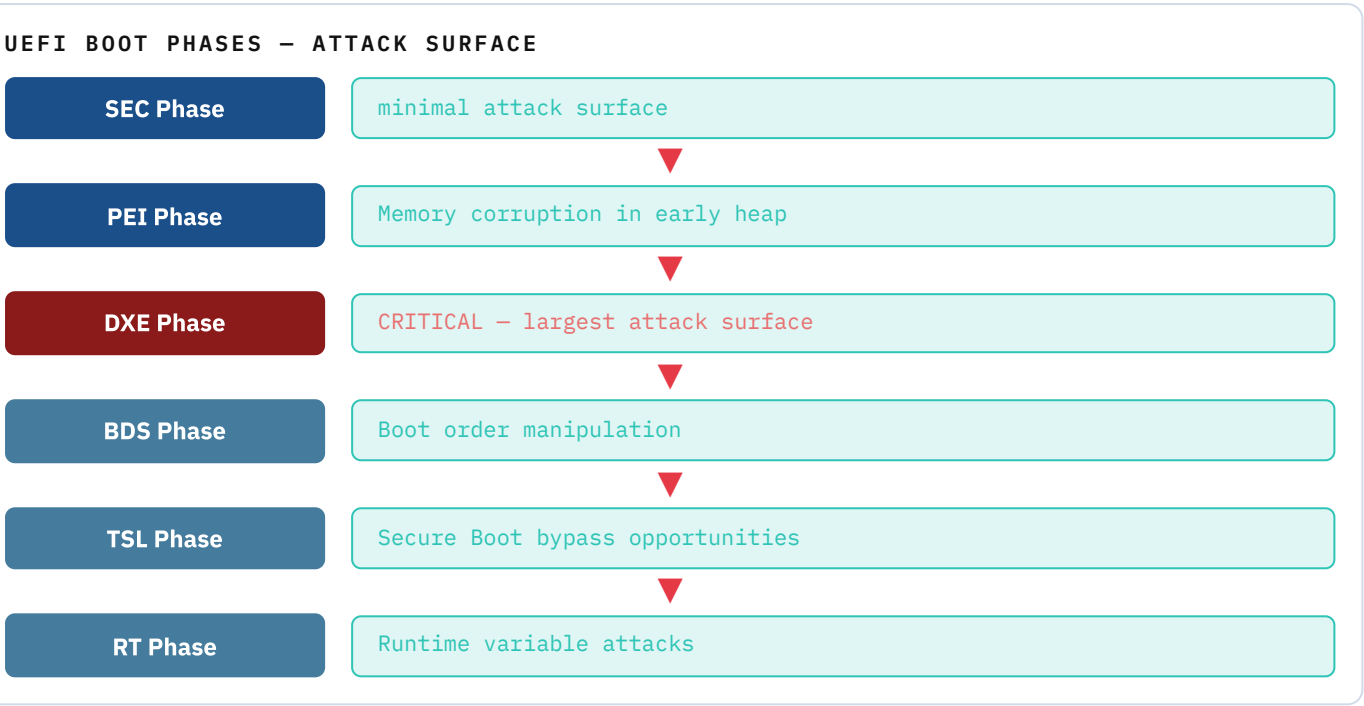
> **Key Insight: A UEFI bootkit is the cybersecurity equivalent of a building tenant changing the locks and then giving you a key that only works when they allow it. You never truly own your system again.**

1.1 Business Impact

Impact Area	Consequences
Operational	Persistent compromise across fleet; remediation requires physical reflashing or motherboard replacement
Financial	$50K–$500K per incident for forensic investigation and fleet remediation
Compliance	PCI DSS, HIPAA, FedRAMP violations due to inability to attest boot integrity
Reputational	Loss of customer trust when bootkits are discovered in production environments

02 UEFI Architecture Overview

2.1 UEFI Boot Phases



03 Bootkit Architecture & Capabilities

3.1 What is a UEFI Bootkit?

A UEFI bootkit is malicious code that resides in the SPI flash storage (firmware) and executes during the UEFI boot process, before the OS loads. Unlike traditional bootkits that resided on disk (MBR/VBR) and were removed during OS reinstallation, UEFI bootkits survive all software-based remediation.

3.2 Bootkit Insertion Vectors

Insertion Vector	Privilege Required	Persistence	Detectability
SPI flash write	Kernel/System (Ring 0)	Permanent until reflash	Medium
UEFI capsule update	Admin/Platform owner	Permanent	Low
Runtime variable	Kernel (Ring 0)	Survives reboot	High
Option ROM	PCIe device vendor	Per-device	Medium
SMM callout	Ring 0 + SMI handler	Firmware persistent	Low

3.3 Bootkit Capability Matrix

Capability	MBR Bootkit	UEFI Bootkit	UEFI Firmware Bootkit
Survives OS reinstall	✓	✓	✓
Persists after disk format	✗	✓	✓

Capability	MBR Bootkit	UEFI Bootkit	UEFI Firmware Bootkit
Survives firmware update	✗	✗	✓ (if not properly signed)
Invisible to EDR/AV	Partial	✓	✓
Can subvert Secure Boot	✗	✓	✓
Requires physical reflash	✗	✗	✓
OS-level detection possible	Partial	Limited	Near impossible

04 Secure Boot Bypass Techniques

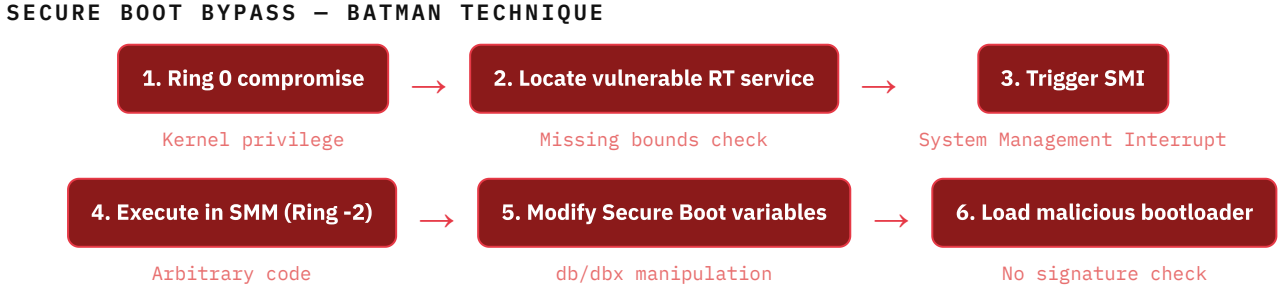
4.1 How Secure Boot Works

Secure Boot ensures that only cryptographically signed and trusted software executes during the boot process. The UEFI firmware contains Platform Key (PK) and Key Exchange Keys (KEK). The signature database (db) contains authorized signatures or hashes. The forbidden database (dbx) contains revoked signatures. Each binary (bootloader, driver, option ROM) is verified before execution.

4.2 Known Secure Boot Bypasses

Bypass Technique	CVE	Mechanism	Mitigation
Golden Key	CVE-2018-12181	Microsoft master key leakage	Revoke compromised keys via dbx
BootHole	CVE-2020-10713	GRUB2 integer overflow leading to arbitrary code	Update dbx with vulnerable GRUB hashes
PKFail	CVE-2023-24932	Multiple vendors used test PK keys in production	Revoke test PKs, enforce production keys
LogoFAIL	CVE-2023-40238	Image parser vulnerabilities in DXE drivers	Patch UEFI firmware, driver updates
Batman	CVE-2022-21894	SMM callout bypass via runtime service abuse	SMM isolation, runtime service validation

4.3 Secure Boot Bypass Attack Flow



4.4 Detection of Secure Boot Bypass

Indicator	Detection Method	Tooling
Secure Boot disabled unexpectedly	UEFI variable monitoring	sbctl, bootinfoscript
Unexpected certificates in db	Certificate inventory comparison	sbctl, mokutil
Missing expected certificates	Baseline comparison	Custom inventory scripts
Forged SetupMode variable	Variable integrity check	TPM attestation
SMI handler corruption	SMI handler validation	Chipsec, SMM detection tools

05 Pre-OS Exploitation Vectors

5.1 DXE Driver Injection

The Driver Execution Environment (DXE) loads hundreds of UEFI drivers during boot. Each driver represents a potential attack surface.

Attack Method: Compromise running OS (kernel privilege), write malicious DXE driver to SPI flash or EFI System Partition (ESP), modify boot order or driver load order, then malicious driver loads during next boot before any security software.

Detection: Firmware integrity scanning (e.g., CHIPSEC), driver hash verification against golden images, boot-time driver enumeration logging.

5.2 UEFI Runtime Variable Attacks

UEFI variables store configuration data in NVRAM. Attackers can modify BootOrder to load malicious bootloader first, set SetupMode=1 to allow unsigned code, inject Boot#### variables pointing to attacker-controlled binaries, or overwrite SecureBoot=Disabled in variable storage.

Forensic Artifacts: Unexpected BootOrder values, unknown Boot#### variables, SecureBoot state changes without authorized update, AuditMode or DeployedMode transitions.

5.3 Option ROM Attacks

PCIe devices (NICs, storage controllers, GPUs) can ship with Option ROMs containing UEFI drivers. Attackers can flash malicious Option ROM to PCIe device, leverage DMA capabilities to read/write system memory, and execute code in UEFI context during device initialization.

Detection: Option ROM scanning and hash verification against vendor database.

5.4 ACPI Table Manipulation

ACPI tables provide power management and hardware configuration. Malicious modifications can inject code via DSDT (Differentiated System Description Table), manipulate FACP (Fixed ACPI Description Table) for SMM abuse, or add malicious SSDT (Secondary System Description Table) entries.

Tooling: acpiexec, iasl for table inspection.

5.5 SMM (System Management Mode) Subversion

SMM is the highest privilege execution environment (Ring -2). SMM code executes in response to System Management Interrupts (SMIs) and can access all system memory.

Attack Vectors: SMI handler buffer overflow leading to SMM code execution, SMM callout via runtime service, SMM cache poisoning, and software-generated SMI with attacker-controlled parameters.

Detection: SMM integrity monitoring via CHIPSEC module common.smm, SMI handler enumeration and analysis, memory scanning for SMM-resident implants.

06 Known UEFI Firmware Rootkits & Malware

6.1 Comprehensive Malware Database

Malware	Discovered	Target	Mechanism	Persistence
BlackLotus	2022	Windows UEFI	Bypasses Secure Boot via CVE-2022-21894	SPI flash
MoonBounce	2021	Windows	Injects into AMI UEFI DXE driver	Firmware module
FinSpy (FinFish)	2011-2023	Windows/macOS	UEFI bootkit variant for surveillance	ESP + firmware
LoJax (Sednit)	2018	Windows	First UEFI rootkit in the wild	SPI flash (LoJack module)
ESPecter	2021	Windows	Malicious bootloader on ESP	EFI System Partition
Bootkitty	2024	Linux (Ubuntu)	Signed bootkit for UEFI Linux systems	Signed bootloader
Cuttlefish	2023	Windows	TPM manipulation + Secure Boot bypass	Variable manipulation

6.2 BlackLotus Deep Dive

BlackLotus (2022-2023) is the most sophisticated publicly documented UEFI bootkit.

Attribute	Detail
Discovery	ESET Research, October 2022
CVE	CVE-2022-21894 (Batman) – Secure Boot bypass
Target	Windows 10/11 x64 with Secure Boot enabled
Capabilities	Disables HVCI, BitLocker, Defender; loads unsigned kernel drivers
Persistence	Writes payload to SPI flash, survives OS reinstall

Attribute	Detail
Detection evasion	Patches boot-time integrity checks; removes itself from logs
Distribution	Underground forums (Dark Web), $5,000+ per license

> **Mitigation: Apply Microsoft KB5025885 (2023) and subsequent patches, update dbx with CVE-2022-21894 mitigations, enable HVCI and firmware TPM attestation.**

6.3 MoonBounce Deep Dive

MoonBounce (discovered by Kaspersky, 2021) targeted AMI UEFI firmware on enterprise laptops.

Attribute	Detail
Target	AMI UEFI firmware on enterprise laptops
Insertion	Infected AMI firmware module (AMITSE DXE driver)
Persistence	Firmware-level; survives disk replacement
Detection	Anomalous SMI handler, unusual SMM communication
Attribution	APT41 (suspected)

07 Detection Methodologies

7.1 Firmware Integrity Scanning Tools

Tool	Method	Strengths	Limitations
CHIPSEC	Platform analysis framework	Comprehensive SMM, variable, SPI checks	Requires technical expertise
BinWalk	Firmware binary analysis	Extracts and parses UEFI volumes	Labor-intensive, no automation
UEFITool	UEFI image manipulation/analysis	GUI for firmware exploration	Manual analysis only
Fwhunt	UEFI module analysis	Detects known malicious patterns	Signature-based

7.2 CHIPSEC Audit Modules

```
# Basic system audit
python chipsec_main.py -a -l chipsec.log

# Critical UEFI audit modules
python chipsec_util.py uefi scan
```

```
python chipsec_util.py spi dump firmware.bin
python chipsec_util.py smi scan
python chipsec_util.py vars list

# Secure Boot verification
python chipsec_util.py uefi secureboot

# SMM integrity check
python chipsec_util.py smm info
python chipsec_util.py smm compress
```

7.3 Detection Analytics (MITRE ATT&CK)

Technique	ID	Detection Method
System Firmware	T1542.001	Firmware version monitoring; SPI flash hash comparison
Bootkit	T1542.003	UEFI variable monitoring; boot order baseline
Secure Boot Bypass	T1553.001	Signature verification events; db/dbx modification audit
SMM Abuse	T1055 (variant)	SMI handler integrity; SMM communication monitoring
UEFI Variable Manipulation	T1612	Runtime variable audit logging

7.4 Enterprise Detection Controls

Control	Implementation	Coverage
Firmware inventory	Periodic SPI flash hash collection (PowerShell + WMI)	Baseline deviation
Secure Boot monitoring	Event ID 1035 (Secure Boot validation failure)	Windows Event Log
UEFI variable audit	Get-UEFIVariable PowerShell logging	Variable changes
Boot order baseline	Centralized boot order registry	Unexpected changes
TPM attestation	PCR 0-7 measurements; remote attestation	Hardware-backed integrity

7.5 Windows-Specific Detection

```
# Check Secure Boot status
Confirm-SecureBootUEFI
Get-SecureBootPolicy

# Enumerate UEFI variables
Get-UEFIVariable -Name "BootOrder"
Get-UEFIVariable -Name "SecureBoot"

# Check for unexpected boot entries
```

```
bcdedit /enum all

# Review Code Integrity events
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-
CodeIntegrity/Operational'; ID=3076}
```

7.6 Linux-Specific Detection

```
# Check Secure Boot status
sbctl status
mokutil --sb-state

# Enumerate UEFI variables
ls /sys/firmware/efi/efivars/
efivar -l

# Verify kernel signature
kmodsign --verify /boot/vmlinuz-$(uname -r)

# Firmware detection (fwupd)
fwupdmgr get-devices
fwupdmgr get-updates
```

08 Mitigation Strategies

8.1 Architectural Controls

Control	Description	Priority
Secure Boot enforcement	Mandatory signature verification; UEFI 2.3.1+	Critical
Firmware write protection	SPI flash write protect via hardware jumper or BIOS setting	Critical
Measured Boot (TPM)	Boot measurements in PCRs 0-7; remote attestation capability	Critical
UEFI capsule updates signed	Require manufacturer-signed firmware updates	High
SMM isolation	SMM page table protection; restricted SMI sources	High
Runtime variable protection	Critical variables locked after EndOfDxe	Medium
Option ROM verification	Signature check for PCIe Option ROMs	Medium

8.2 Microsoft Recommended Mitigations for BlackLotus/Batman

KB	Date	Mitigation	Deployment
KB5025885	Apr 2023	Initial BlackLotus mitigation	Windows 10/11
KB5027455	Jun 2023	Enhanced dbx update	Windows 10/11
KB5025885 (update)	Aug 2023	Final revocations; full BlackLotus blocking	All supported Windows

```
# Deployment verification
reg query HKLM\SYSTEM\CurrentControlSet\Control\SecureBoot\Changes
reg query HKLM\SYSTEM\CurrentControlSet\Control\SecureBoot\Updated
```

8.3 Vendor Firmware Update Best Practices

- Enable automatic firmware updates from trusted vendors for timely security patch deployment
- Verify checksums before flashing to prevent corrupted/malicious updates
- Use hardware-backed update verification to prevent rollback attacks
- Maintain recovery image for rapid restoration if compromise detected
- Disable legacy BIOS compatibility (CSM) to reduce attack surface

8.4 Incident Response for Bootkit Compromise

When bootkit is suspected, follow these steps:

1. **Isolate affected system** – Assume all data is compromised
2. **Capture forensic artifacts** – SPI flash dump (hardware programmer preferred), UEFI variable export, TPM event log, Secure Boot event logs
3. **Determine persistence mechanism** – SPI flash vs. ESP vs. Option ROM
4. **Remediate** – For SPI flash compromise: physical reflash with verified image or motherboard replacement. For ESP compromise: secure wipe and OS reinstall. For Option ROM compromise: replace affected PCIe device
5. **Verify remediation** – CHIPSEC audit, Secure Boot re-verification
6. **Root cause analysis** – How did initial compromise occur?

09 Forensic Analysis

9.1 SPI Flash Acquisition

Physical acquisition (preferred): Identify SPI flash chip (Winbond, Macronix, etc.), connect programmer (CH341A, Dediprog, Bus Pirate), dump full flash contents (8MB-32MB typical).

Software acquisition (limited): CHIPSEC (chipsec_main.py -a -l log.txt) and UEFITool for runtime extraction.

Analysis: Binary diff against known-good image, UEFITool volume extraction, disassemble suspicious DXE or SMM modules, check for unexpected PEI or DXE drivers.

9.2 Forensic Artifacts

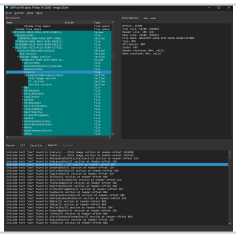
Artifact	Location	Forensic Value
SPI flash image	Physical flash chip	Complete firmware; all modifications visible
UEFI variables	NVRAM (part of SPI flash)	Boot order, Secure Boot state, db/dbx
EFI System Partition (ESP)	Disk partition (FAT32)	Bootloaders, boot applications, drivers
TPM event log	TPM hardware / OS log	Boot measurements; attestation failures
Secure Boot event log	Windows Event Log	Signature failures; revoked binaries

9.3 Signature-Based Detection (Blacklist)

Indicator	Type	Source
BlackLotus SPI patterns	Hash	ESET, Microsoft
MoonBounce DXE module	Hash	Kaspersky
Compromised GRUB versions (BootHole)	Hash list	Microsoft dbx, US-CERT
Test PK certificates	Certificate	CVE-2023-24932

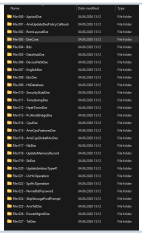
9.4 Practical Firmware Analysis – Real-World Artifacts

📸 UEFITOOL NE – VOLUME 000: DXE DRIVERS (DXECORE, APRIORIDXE, SECURITYSTUBDXE)



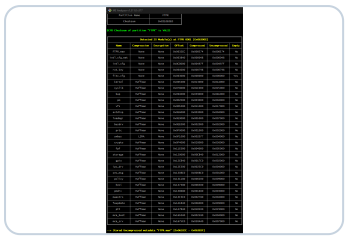
🔍 DxeCore, AprioriDxe, SecurityStubDxe – key modules for analysis

📸 UEFITOOL NE – COMPLETE DXE DRIVER LIST (AMI FIRMWARE)



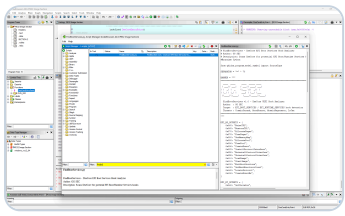
🔍 25+ drivers: AprioriDxe, DxeCore, Bds, CpuDxe, NbDxe, SbDxe

📸 ME ANALYZER – FTPR PARTITION: KERNEL, CRYPTO, PTT (PLATFORM TRUST TECHNOLOGY)



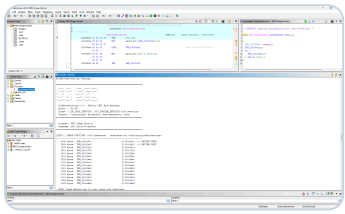
🔍 kernel (76KB), crypto (216KB), ptt (164KB – software TPM), fwupdate, heci

📸 GHIDRA – DXECOREENTRYPOINT + FINDBOOTSERVICES.PY (GG SEC HOOK DETECTOR)



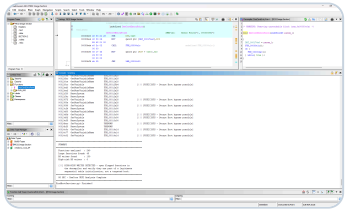
🔍 GG SEC custom script: FindBootServices – scans DxeCore for EFI_BOOT_SERVICES hooks

📸 GHIDRA – DXECOREENTRYPOINT (FINDBOOTSERVICES.PY SCAN RESULTS – OFFSETS)



🔍 FindBootServices.py output – detected potential hooks in EFI_BOOT_SERVICES table (offsets: RaiseTPL, AllocatePages, ExitBootServices, LocateProtocol, CreateEventEx, GetVariable)

📸 GHIDRA – DXECOREENTRYPOINT (FINDBOOTSERVICES.PY SCAN – FUNCTION ANALYSIS)



🔍 FindBootServices.py detailed scan – DxeCoreEntryPoint analysis, GetVariable, GetNextVariableName calls

10 DXE Driver Deep Dive – BiosGuard, Hooking & Boot Services Manipulation

10.1 DXE Driver Loading Order – AprioriDxe

Driver	Purpose	Security Relevance
DataHubDxe	Data hub for system information	Low – info gathering

Driver	Purpose	Security Relevance
CpuDxe	CPU initialization, SMM setup	HIGH – SMM entry point
PciHostBridgeDxe	PCI host bridge	Medium – DMA attack surface

10.2 BiosGuardDxe – Critical Security Driver

> **Historical Vulnerability (CVE-2017-5706): Bypass of Intel Boot Guard and BIOS Guard via removal of verification HOBs.**

10.3 Critical DXE Drivers for Security Research

Driver	Why Important
DxeCore	PRIME TARGET – CosmicStrand, MosaicRegressor hook DxeCore
SmmCore / *Smm*	HIGHEST PRIVILEGE – BlackLotus target
BiosGuardDxe	Bypass = unrestricted flash write

10.4 EFI Boot Services Hooking

Service	When Hooked	Known Malware
ExitBootServices	Execute payload before OS	CosmicStrand
AllocatePages	Allocate hidden memory	MoonBounce
CreateEventEx	Schedule malicious callback	MosaicRegressor

10.5 Indicators of Compromise (IoC) – DXE Level

IndicatorTypical of Write to gBS→ExitBootServices pointerCosmicStrand CreateEventEx with unknown GUIDMosaicRegressor Jump to address outside PE rangeBlackLotus

11 SMM Deep Dive – Ring -2 & System Management Mode

System Management Mode (SMM) is the highest privilege level in x86/x64 – **Ring -2**, beyond hypervisor (Ring -1) and kernel (Ring 0). SMM code executes in response to System Management Interrupts (SMIs) and can access all system memory, including hypervisor and kernel space.

> **Why SMM matters: If an attacker compromises SMM, no software-based detection (EDR, AV, hypervisor) can see or stop them. SMM is invisible from the OS.**

11.1 SMM Architecture – Attack Surface

Component	Attack Surface
SMI (System Management Interrupt)	Malicious SMI generation → forced entry
SMI Handler	Buffer overflow → arbitrary SMM code
SMRAM (SMM RAM)	SMRAM cache poisoning, lock bypass
SMM Communication Protocol	Buffer manipulation → SMM memory corruption
SW SMI (I/O port 0xB2)	Unprivileged SMI generation → DoS / code execution

11.2 BlackLotus SMM Techniques (CVE-2022-21894)

- Triggered vulnerable SMI handler via runtime service abuse
- Executed arbitrary code in SMM (Ring -2)
- Modified Secure Boot variables (db/dbx) from SMM – invisible to OS
- Patched TPM measurements to report false attestation

11.3 SMM Detection Methodology

```
# CHIPSEC SMM detection
python chipsec_util.py smi scan
python chipsec_util.py smm info
python chipsec_util.py smm compress
python chipsec_main.py -m common.smm
python chipsec_util.py smm smram_lock
```

Znane podatności w SMI handlerach:

- CVE-2022-21894 – Batman (BlackLotus) – SMM callout via runtime service
- CVE-2020-24489 – Intel – SMI handler buffer overflow
- CVE-2017-5706 – BIOS Guard bypass via HOB manipulation affecting SMM

11.4 SMM Indicators of Compromise (IoC)

Indicator	Detection
Unexpected SMI handler	SMI handler enumeration
SMRAM cache poisoning	SMM integrity check (CHIPSEC)
SW SMI from unprivileged code	I/O port 0xB2 monitoring
SMRAM lock bypass	Register check (SMRR, SMM_ACCESS)

SMM Mitigation:

- **Enable SMM page table protection (BIOS setting)**
- **Lock SMRAM after initialization (D_LCK bit)**

- **Limit SMI sources (USB, PCIe hotplug)**
- **Use CHIPSEC to verify SMM integrity regularly**

12 Enterprise Implementation Roadmap

12.1 Phase 1: Assessment (Week 1-2)

- Inventory UEFI versions across fleet – Firmware version report
- Secure Boot status audit – Compliance dashboard
- Identify non-Secure Boot systems – Remediation list
- CHIPSEC scan on high-value systems – Baseline report

12.2 Phase 2: Hardening (Week 3-8)

- Enable Secure Boot enterprise-wide – Critical
- Update dbx with all known revocations – Critical
- Enable TPM + BitLocker (with PCR 0-7) – Critical
- Configure firmware write protection – High
- Disable legacy boot (CSM) – High
- Implement UEFI variable monitoring – Medium

12.3 Phase 3: Continuous Monitoring (Ongoing)

- Firmware version drift detection – Monthly
- Secure Boot attestation – Continuous (via MDE/CrowdStrike)
- CHIPSEC full audit – Quarterly (critical systems)
- UEFI variable baseline comparison – Weekly
- Option ROM scanning – Upon new device enrollment

13 Why GGSEC – Threat Research & Firmware Specialization

GGSEC is a specialized security research laboratory focused exclusively on firmware-level threats, embedded systems auditing, and supply chain risk assessment.

13.1 GG SEC Internal Tooling

Tool	Purpose
FindBootServices.py (Ghidra)	Scans DxeCore for EFI Boot/Runtime Services hooks – detects CosmicStrand, MoonBounce, MosaicRegressor
UEFI Volume Extractor	DXE driver extraction, hash baseline generation, anomaly detection

Tool	Purpose
ME Analyzer Automation	ME region integrity validation, checksum verification, manifest signature validation
SMM Integrity Scanner	SMI handler enumeration, SMM memory analysis, BlackLotus detection

14 Appendix & References

14.1 Quick Reference Card – Windows Detection Commands

```
Confirm-SecureBootUEFI
Get-UEFIVariable -Name "BootOrder"
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-
CodeIntegrity/Operational'} | Where-Object {$_.Id -eq 3076}
Get-WmiObject -Class Win32_BIOS
```

14.2 Quick Reference Card – Linux Detection Commands

```
sbctl status
mokutil --sb-state
ls /sys/firmware/efi/efivars/
dmesg | grep -i "secure boot\|uefi"
```

14.3 Critical CVEs to Track

CVE	Description	Status
CVE-2022-21894	Batman (BlackLotus)	Patched (KB5025885)
CVE-2020-10713	BootHole (GRUB2)	Patched (dbx update)
CVE-2023-24932	PKFail	Patched (2023)
CVE-2023-40238	LogoFAIL	Patch pending (vendor dependent)

14.4 References

1. ESET Research. (2023). "BlackLotus UEFI bootkit: Myth or reality?"
2. Kaspersky. (2021). "MoonBounce: the dark side of UEFI firmware."
3. Microsoft Security Response Center. (2023). "CVE-2022-21894 Secure Boot Security Feature Bypass Vulnerability."
4. MITRE ATT&CK. (2024). "Bootkit (T1542.003)" and "System Firmware (T1542.001)."
5. Binarly Research. (2023). "LogoFAIL: Image parser vulnerabilities in UEFI DXE drivers."
6. Quarkslab. (2020). "The BootHole vulnerability: GRUB2 bootloader compromise."
7. Google Project Zero. (2020). "CVE-2020-10713: GRUB2 BootHole."

8. Intel Corporation. (2023). "CHIPSEC: Platform Security Assessment Framework."
9. National Institute of Standards and Technology. (2018). "NIST SP 800-193: Platform Firmware Resiliency Guidelines."
10. US-CERT. (2023). "AA23-165A: BlackLotus UEFI Bootkit."

Maciej Gojny · GG Advanced IT Security · ggsec.de

GGSEC Research · June 2026 · TLP:WHITE – Public Release

Document Version 1.0 · Last updated: June 2026