

A Practical Analysis of UEFI threats against Windows 11

Joshua Machauer

June 21, 2022
Version: Draft 1.0

Technische Universität Berlin



Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Security in Telecommunications (SecT)

Bachelor's Thesis

A Practical Analysis of UEFI threats against Windows 11

Joshua Machauer

1. Reviewer **Prof. Dr. Jean-Pierre Seifert**
Electrical Engineering and Computer Science
Technische Universität Berlin

2. Reviewer **Prof. Dr.-Ing. Friedel Gerfers**
Electrical Engineering and Computer Science
Technische Universität Berlin

Supervisors Jane Doe and John Smith

June 21, 2022

Joshua Machauer

A Practical Analysis of UEFI threats against Windows 11

Bachelor's Thesis, June 21, 2022

Reviewers: Prof. Dr. Jean-Pierre Seifert and Prof. Dr.-Ing. Friedel Gerfers

Supervisors: Jane Doe and John Smith

Technische Universität Berlin

Security in Telecommunications (SecT)

Institute of Software Engineering and Theoretical Computer Science

Electrical Engineering and Computer Science

Ernst-Reuter-Platz 7

10587 and Berlin

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 10. Juli 2022

Joshua Machauer

Abstract

In Computer Security one of the most feared threats is a bootkit, executing at the beginning of a computers boot chain, before the operating system and accompanying antivirus programs. With the widespread adaption of standardized UEFI firmware these threats have become less machine dependent and can now target a host of systems at once. Past analyses about bootkits have been case studies of their appearances in the wild, this thesis instead aims to be a more practical approach by developing a bootkit and analyzing the challenges doing so. We restrict our analysis by assuming an attacker has already gained read and write access to the BIOS image and is thus only facing security mechanisms involved during and with execution of the bootkit. Our bootkit was able to achieve elevated execution on Windows 11 by exploiting unrestricted hard drive access to edit Windows Registries, this was also possible on BitLocker encrypted hard drives by keylogging the Recovery Key. UEFI makes it very easy for an attacker who has gained access to the System Firmware to leverage its powers and gain full control over the system.

Abstract (different language)

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Acknowledgement

Contents

1	Introduction	1
2	Background	3
2.1	UEFI	3
2.1.1	Boot Sequence	3
2.1.2	UEFI/PI Firmware Images	5
2.1.3	UEFI Images	5
2.1.4	Firmware Core	6
2.1.5	edk2	6
2.1.6	Security	6
2.2	Windows	6
2.2.1	User Access Control (UAC)	6
2.2.2	Signing	6
2.2.3	Bitlocker	6
3	Related Work	7
4	Attacks	9
4.1	Neither Secure Boot nor Bitlocker	9
4.2	Secure Boot	10
4.3	Secure Boot and Bitlocker	10
5	Discussion	13
5.1	Rootkit classification	13
5.2	Mitigations	13
5.2.1	User awareness	14
6	Conclusion	15
6.1	Achieved Goals	15
6.2	Future Work	15
A	Appendix	23

Introduction

definition of rootkit/bootkit persistence goals 1/3 - 1/2 pages

Background

The following introduces the background information necessary to understand the employment of a Unified Extensible Firmware Interface (UEFI) rootkit. This includes the general workings of the Platform Initialization (PI) and UEFI, the UEFI programming model and interface itself; as well as its security mechanisms. It is also necessary to understand our target's defenses, for this, we briefly describe the Windows security mechanisms faced when performing our attacks.

2.1 UEFI

Unified Extensible Firmware Interface (UEFI) is a specification that defines an abstract interface between an Operating System (OS) and the Platform Firmware (PF). It was designed to replace the legacy Boot Firmware Basic Input/Output System (BIOS), while also often offering a backwards compatible mode with the Compatibility Support Module (CSM). The specification is a pure interface specification thus merely states what interfaces and structures a PF has to offer and what an OS may use. how it is implemented by PF what is used by OS boot- and runtime service functions for the bootloader and os to call datatables containing platform-related information - complete solution describing all features and capabilities - abstract interfaces to support a range of processors without the need for knowledge about underlying hardware for the bootloader - sharable persistent storage for platform support code security

2.1.1 Boot Sequence

focus will be on dxef and transient system load

1. Security (SEC)

establishment of root of trust in the system inductive security design integrity of subsequent module is checked by the previous module first module is root of trust

handles all platform restart events applying power to system from unpowered state restarting from active state receiving exception conditions

creates temporary memory store possibly CPU Cache as RAM (CAR) cache behaves as linear store of memory no evictions mode every memory access is a hit eviction not supported as main memory is not set up yet and would lead to platform failure

final step Pass handoff information to the Pre-EFI Initialization (PEI) Foundation

- state of platform
- location and size of the Boot Firmware Volume (BFV)
- location and size of the temporary RAM
- location and size of the stack
- optionally one or more Hand-off Blocks (HOBs) via the SEC HOB Data PEIM-to-PEIM Interface (PPI)

SEC Platform Information PPI information about the health of the processor

SEC HOB Data PPI

2. Pre-EFI Initialization (PEI)

initialize stuff

3. Driver Execution Environment (DXE)

dxs core dxs dispatcher depex dxs drivers

4. Boot Device Selection (BDS)

5. Transient System Load (TSL)

boottime and runtime services/driver bootloader ExitBootServices()

6. Runtime (RT)

runtime services/driver

7. Afterlife (AL)

hibernation sleep

2.1.2 UEFI/PI Firmware Images

flash device flash volume flash file system file sections depex

2.1.3 UEFI Images

executable subset of PE32+ file format with modified header signature to distinguish from normal PE32 Images + stands addition of 64-bit relocation fix-up extension fixed and dynamic address loading relocatable boot and runtime memory application vs os loader vs driver loaded fully into memory and relocation fix ups memory marked as code and data jump to entry point what is the boot manager

UEFI Applications

example efi shell loaded by boot manager or other applications return or calling exit specifically always unloaded from memory

UEFI OS Loaders

example windows boot manager normally take over control from the firmware upon load behaves like a normal UEFI application - only use memory allocated from the firmware - only use services/protocols to access devices that the firmware exposes - conform to driver specifications to access hardware on error can return allocated resources with Exit boot service with error specific information given in ExitData on success take full control with ExitBootServices boot service all boot services in the system are terminated, including memory management UEFI OS loader now responsible

UEFI Drivers

loaded by boot manager, UEFI firmware (DXE foundation), or other applications example payload unloaded only when returning error code persistent on success boot and runtime drivers only difference is that runtime are available after ExitBootServices was called boottime drivers are terminated and memory is released runtime drivers are fixed up with virtual mappings upon SetVirtualAddressMap call has to convert its allocated memory

2.1.4 Firmware Core

boot and runtime services boot service table guides handles and protocols protocols

2.1.5 edk2

build system

2.1.6 Security

Secure Boot

Signed Capsule Update

SMM

TPM

SEC starts by measuring PEI

2.2 Windows

2.2.1 User Access Control (UAC)

2.2.2 Signing

2.2.3 Bitlocker

TPM

how does it work explain TPM

Related Work

scholar ranking

Attacks

Our different attacks face three escalating levels of security mechanisms. The first is with Secure Boot and Bitlocker disabled, the second is just Secure Boot enabled and the third is both Secure Boot and Bitlocker enabled with the focus of the study on Bitlocker. All attacks share the requirement of being able to add DXE Drivers to the DXE Volume. This can be achieved by having read/write access to the SPI flash or using the Signed Capsule Update. Gaining read/write access to the SPI Flash is possible either through physical access to the device by using an SPI clamp on the chip itself or through exploits like for example the . Signed Capsule Updates can be leveraged with access to private vendor information by signing the payload to make it appear legitimate or by intercepting the distribution process and employing infected firmware.

4.1 Neither Secure Boot nor Bitlocker

use read access to dump image since it an FV with FFS we can open with UEFITool
remove previous NTFS driver if present, for full control, might be read only etc in
UEFITool search and remove add in NTFS driver use write access

try in EFI shell navigate to Windows folder create folder

how does one compile uefi application with edk2 it's open source so we can look up
examples for most stuff

try in code compile dx driver within ovmf to receive .ffs file with version depex
user interface section SimpleFileSystem Protocol iteration write failed on hibernated
file patch to allow write on hibernated drives

pack executable binary as uefi module iterate over firmware volume protocols search
for payload guid check size match override notepad works

but no automatic execution nor elevated privileges dll proxying dll hijacking registry
editing

Task Scheduler defined in xml cached in registry edit with start cmd.exe and trigger manually whoami

chntpw and reged port to uefi edit Task in machine under Control maybe look if just adding a key would have also worked export target registry key modify so that registry key can differ and found via matching values import and override registry key on target machine payload whoami localsystem

4.2 Secure Boot

how does one enable it mostly comes with default keys OEM expectation: not to boot observation: no difference secure boot default policy snippet option roms and bootloader instead relies on Signed Capsule Updates assumes integrity

4.3 Secure Boot and Bitlocker

assumptions: secure boot or not bitlocker enabled with TPM auto decryption

observation: boot execution differs from executing rootkit tpm values different bitlocker auto decryption fails recovery key prompt

what is the reaction of the average user (ask admin for recovery password) type in recovery password alternative would be to remove drive and insert into safe device

prompt is done by the OS Loader ergo still during transient system load phase required to use protocol services therefor uses uefi services for IO such as SimpleTextInputEx Protocol go over the two different input protocols find out which one is used

explain more in depth how protocols are returned to the end user one instance per controller/handle

explain basic hooking explain how we retain information of the hook in question map protocol pointer to hook information keylog recovery key key input advancement is weird and makes tracking tricky

alternatively screen shot still need hook to find when enter is pressed explain how screenshotting works some basic compression wait for recovery key send recovery key on enter press

on real hardware network stack wasn't installed onto handles when boot over ip was disabled compared loaded dx drivers between both configurations with efi shell Realtek Family driver not loaded load manually reinstall all handle to controllers to enable network stack regardless

sending key out is only good for physical access attack vector dislocker linux utility mount encrypted drive with decryption mean read and write access dual boot in vm enter password and it works port to uefi bitlocker encrypts block-wise uefi protocol stack hook block io again hook data mapping dislocker validate block solves recovery key advancement issue

hook ExitBootServices enable hook write payload import registry key disable hook

next boot would require to input tpm values again update tpm values in payload caveat pin? look into this

persistence when part of root of trust fresh install / tpm update values hook Trusted Computing Group 2 (TCG2) Protocol TPM communication receive bitlocker vmk key and send to dislocker

Discussion

attack assumption reflected to real world aplicability

social engineering aspekt

driver vorhanden und was mitbringen, debloating

5.1 Rootkit classification

statistiken zu bilocker und secureboot auf systemen

industrie standard zur system security in firmen

5.2 Mitigations

hardware validated boot

inaccessible spi flash

tpm + pin detectability

googeln wie legitime recovery key prompt reaktion aussieht

enterprise policy recovery key einschraenkbar?

enterprise policy on recovery key loss

5.2.1 User awareness

vermitteln was das prompt bedeuten koennte

aber kann man einfach nicht anzeigen lassen

Security Flaw of entering a Recovery Password in an inheritly unsafe System

enterprise doesnt hand out recovery keys and instead receives hard drive

!!!!!!!!!!!!!!!!!!!!!! without hardware chain of trust a compromised system can patch/change any software and fixes are impossible

phishing prompts on their own

Conclusion

6.1 Achieved Goals

when we are already in the image we can gain full control over the system system cant be trusted anymore e.g. uefi services full file access escalate it to local system level execution bitlocker has the flaw of allowing to enter critical information into an inherently untrustable system on the other hand one could force such a prompt themselves mere existence of a recovery key is a security flaw

6.2 Future Work

tpm and pin capsule update

List of Figures

List of Tables

List of Listings

Appendix

A

Acronyms

AL	Afterlife
BDS	Boot Device Selection
BF	Boot Firmware
BFV	Boot Firmware Volume
BIOS	Basic Input/Output System
CAR	Cache as RAM
CSM	Compatibility Support Module
DXE	Driver Execution Environment
EFI	Extensible Firmware Interface
HOB	Hand-off Block
OS	Operating System
PEI	Pre-EFI Initialization
PEIM	Pre-EFI Initialization Module
PF	Platform Firmware
PI	Platform Initialization
PPI	PEIM-to-PEIM Interface
RT	Runtime
SEC	Security
TSL	Transient System Load
UEFI	Unified Extensible Firmware Interface

