#BHUSA @BlackHatEvents

Damn Vulnerable UEFI

Stanislav (Stas) Lyakhov and Mickey Shkatov

UEFI Security: why care?

- **Highly privileged systems**
  - Attackers can execute privileged code and gain persistence
  - Difficult to detect and remove UEFI Malware
- **Critical attack surface**
  - Confluence of platform initializes, drivers, bootloaders, etc.
  - Complex code + minimal protections
- **Sophisticated malware already targets pre-boot environments**
  - Lojax
  - BlackLotus

BlackLotus Mitigation Guide

Executive summary

BlackLotus is a recently publicized malware product garnering significant attention within tech media. Similar to 2020's BootHole (CVE-2020-10713), BlackLotus takes advantage of a boot loader flaw—specifically CVE-2022-21894 Secure Boot bypass known as "Baton Drop"—to take control of an endpoint from the earliest phase of software boot. Microsoft® issued patches for supported versions of Windows to correct boot loader logic. However, patches were not issued to revoke trust in unpatched boot loaders via the Secure Boot Deny List Database (DBX). Administrators should not consider the threat fully remediated as boot loaders vulnerable to Baton Drop are still trusted by Secure Boot.

As described in this Cybersecurity Information Sheet (CSI), NSA recommends infrastructure owners take action by hardening user executable policies and monitoring the integrity of the boot partition. An optional advanced mitigation is to customize Secure Boot policy by adding DBX records to Windows® endpoints or removing the Windows Production CA certificate from Linux® endpoints.

BlackLotus boot security threat

NSA recognizes significant confusion regarding the threat posed by BlackLotus. Some organizations use terms like "unstoppable," "unkillable," and "unpatchable" to describe the threat. Other organizations believe there is no threat due to patches that Microsoft released in January 2022 and early 2023 for supported versions of Windows. [1] The risk exists somewhere between both extremes.

BlackLotus shares some characteristics with Boot Hole (CVE-2020-10713). [2] Instead of breaking the Linux boot security chain, BlackLotus targets Windows boot by exploiting a flaw in older boot loaders—also called boot managers—to set off a chain of malicious actions that compromise endpoint security. Exploitation of Baton Drop (CVE-2022-21894) allows BlackLotus to strip the Secure Boot policy and prevent its enforcement. Unlike Boot Hole, the vulnerable boot loaders have not been added to the Secure Boot DBX revocation list. Because the vulnerable boot loaders are not listed within the DBX, attackers can substitute fully patched boot loaders with vulnerable versions to execute BlackLotus.

NSA recommends system administrators within DoD and other networks take action. BlackLotus is not a firmware threat, but instead targets the earliest software stage of boot.

U/OO/167397-23 | PP-23-1628 | JUN 2023 Ver. 1.0 1

DVUEFI Motivation

- **Learning UEFI hacking: high barrier to entry**
  - Complex environment (how to interact with BIOS?)
  - Scattered documentation
  - Lots of folklore techniques and tools
  - No definitive “exploitation guide” (see OWASP Vulnerable App)
- **Difficult to reproduce vulnerabilities**
  - “Is my environment vulnerable to this?”
  - “Is my DBX too new?”
  - Exploits that work with one environment are unlikely to work on another without modification

UEFI doesn't have to be difficult!

1. **Little to no binary protections make exploitations easier!**
   - Stack canaries disabled by default!
   - No ASLR!
2. **Lots of strong options for virtualization freely available**
   - QEMU (with or without KVM)
   - VMWare

DVUEFI In a Nutshell

- **Intentionally vulnerable UEFI platform**
  - Guided challenges that increase in difficult and complexity
  - Covers a diverse set of vulnerabilities within UEFI
- **Easy-to-setup UEFI exploitation Environment**
  - Thorough instructions for setting up full exploitation environment
  - Multiplatform support (both Linux and Windows)
  - Consistent setup across users/installations
- **Community contribution encouraged!**
  - Easy to add new custom vulnerabilities and challenges
  - Growing body of knowledge: information sharing and discussion

Who is DVUEFI for?

- **Everyone interested in learning about UEFI hacking**
  - No prior firmware experience required
  - Environment setup made simple
  - Guides explain vulnerability concepts
- **UEFI/Firmware Developers**
  - Learn how to identify and prevent vulnerabilities during development
  - Learn about existing mitigation techniques: https://github.com/jyao1/SecurityEx
- **Experienced Firmware Researchers**
  - Community contribution! New challenges and guides welcome
  - Publish research and demonstrate how to exploit in the DVUEFI environment!

Talk Outline

0) The DVUEFI blueprint

1) Level 1 Challenges: Simple UEFI Apps

2) Level 2 Challenges: Real-world UEFI Apps (with SecureBoot!)

3) Level 3 Challenges: Hacking on UEFI booting Windows

4) Plans for the future: Level 4 and Beyond

Level 0: DVUEFI Environment

1) DVUEFI is divided into **Environments/Levels** with increasing complexity

2) Each level follows the **same blueprint**

    1) Start with a *base firmware* image (e.g. EDK II)

    2) Insert *vulnerable "challenges"* into the environment

    3) Provide instructions for *running* (and debugging) the challenges

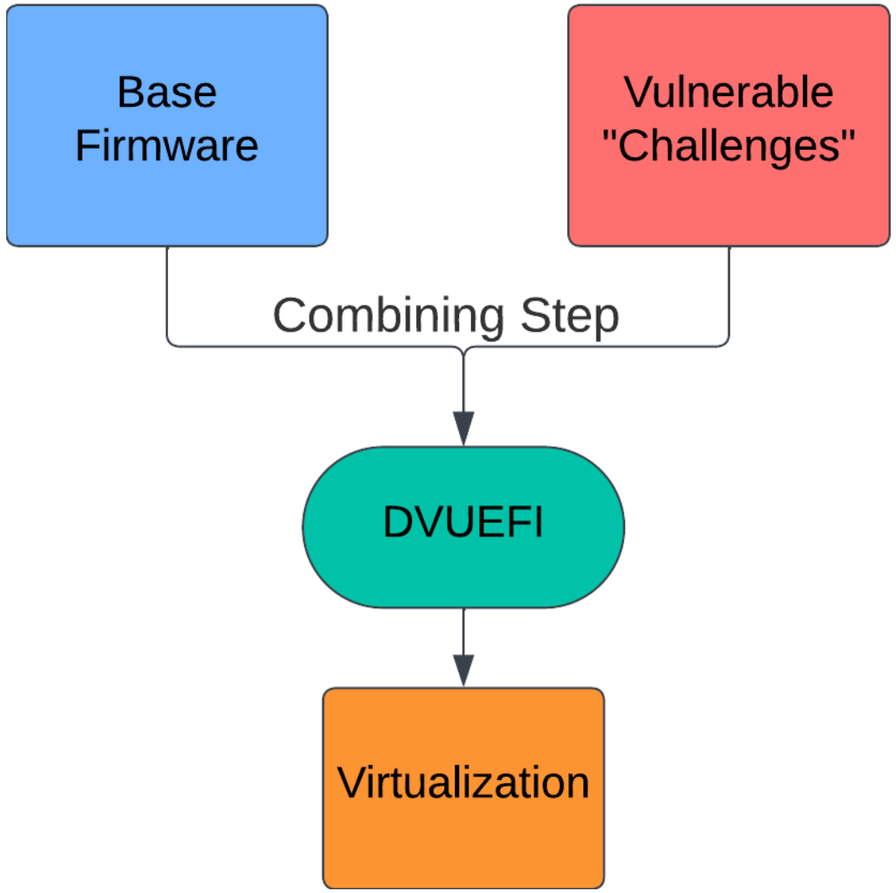
    4) Guidance on *triggering vulnerabilities* (no spoilers for full exploit!)

3) **General Philosophy:**

    1) Environment setup should be as simple and automatic as possible

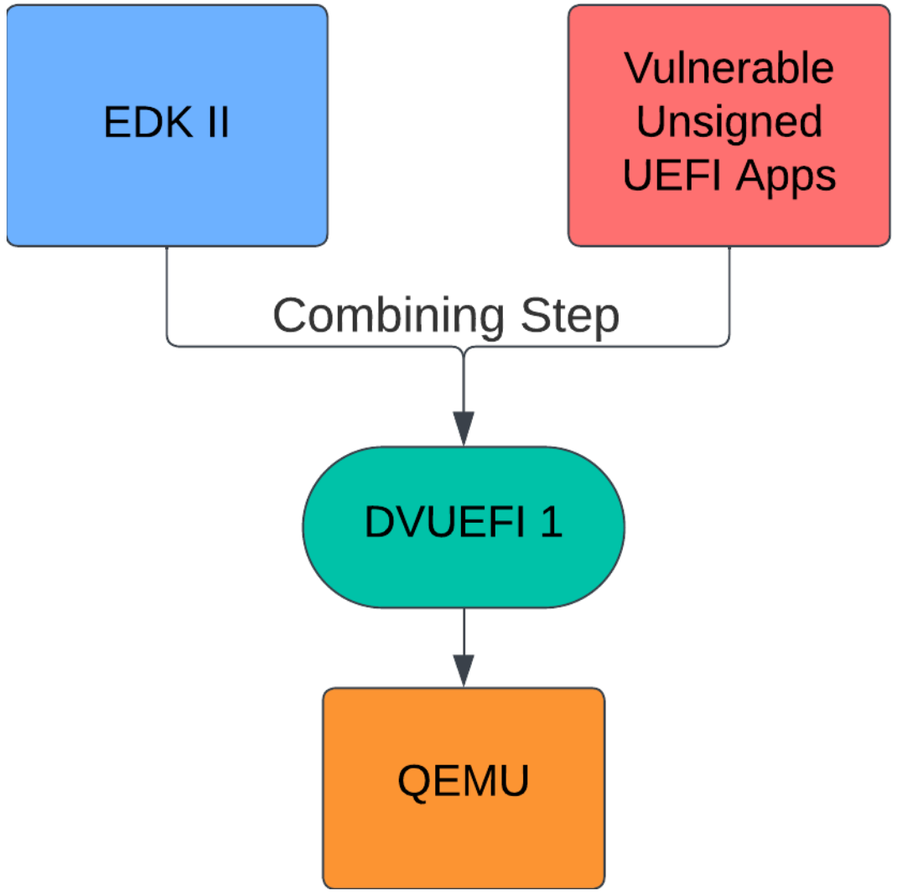
    2) Adding more challenges (and modifying existing ones) should be easy

    3) Keep a consistent tool-chain



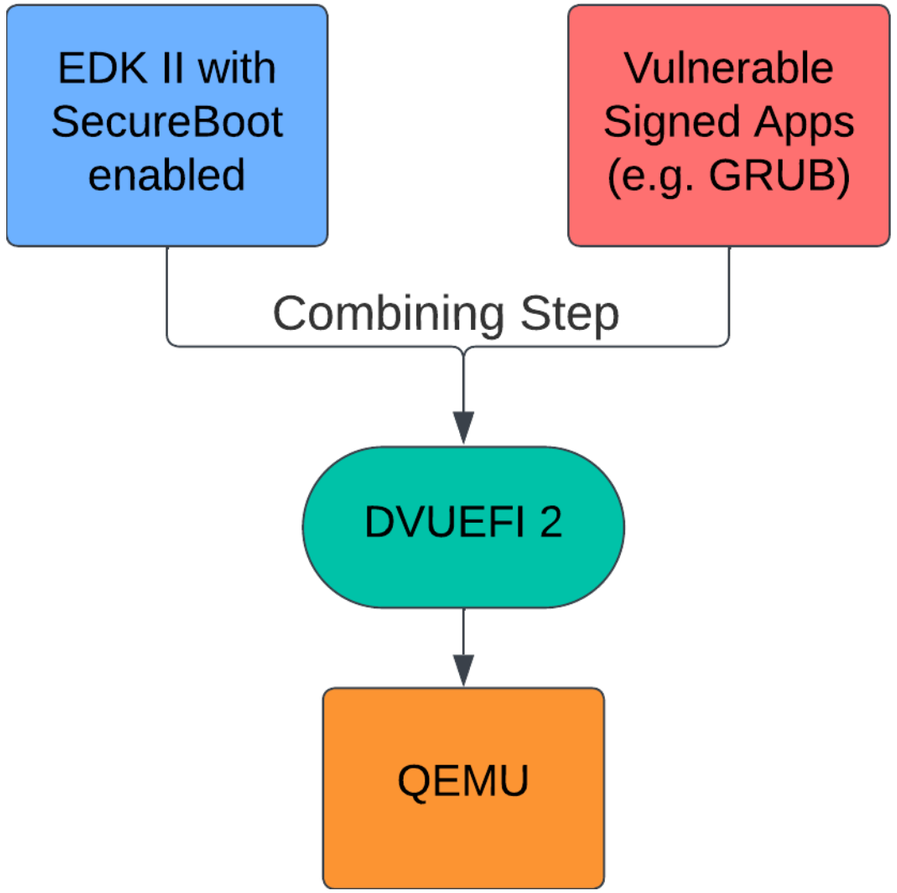
Level 1: Simple UEFI Apps

1) **Compilation and Setup**
    1) Full docker environment to reliably compile EDK II
    2) Guide on setting up and lunching challenges
2) Start exploring: Interact with simple application in **UEFI Shell**
    1) Making sure everything is properly compiled and DVUEFI is setup
    2) Getting comfortable with QEMU environment and UEFI Shell
3) **Apps** increase in complexity, but vulnerabilities remain simple
    1) Start with simple buffer overflow
    2) Move on to exploiting vulnerable network apps, image parsers, etc.



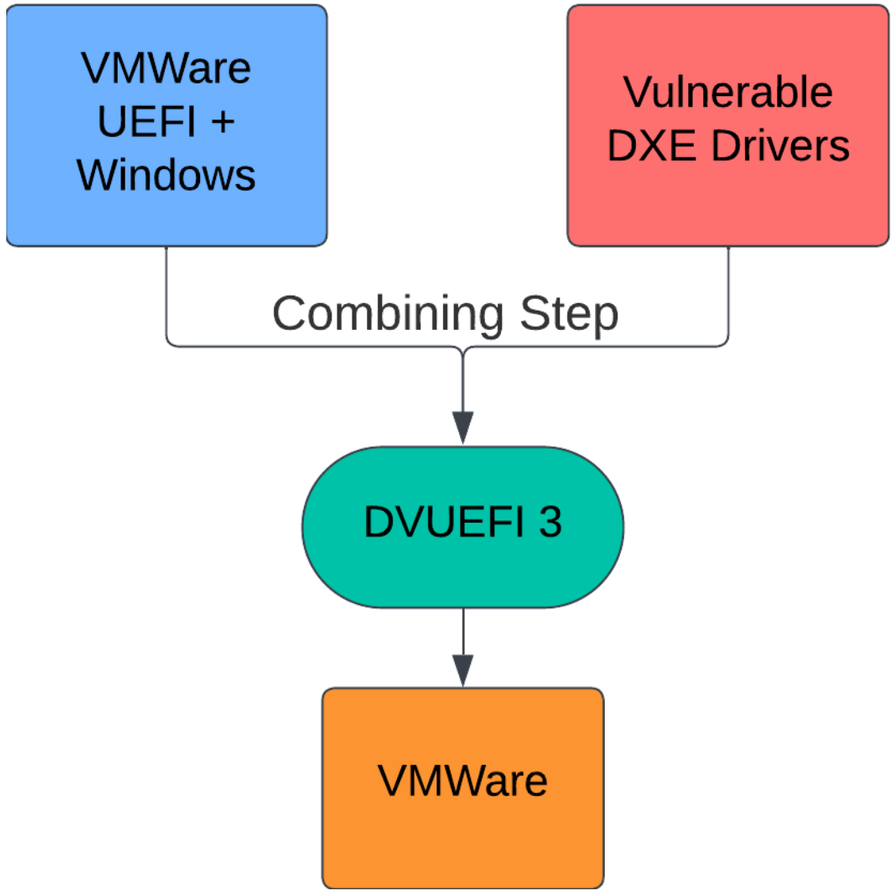
Level 2: Signed UEFI Apps

1) **SecureBoot Bypass Required**
    1) More realistic scenarios
    2) Can't run any off-the-shelf vulnerable app! Must be signed
2) **More Complex Applications**
    1) Exploiting Bootloaders
    2) Boothole, BatonDrop, HTTP Boot etc.
3) **Automatic Environment Setup**
    1) Compile EDK II
    2) Enroll keys automatically
    3) Load pre-signed vulnerable apps



Level 3: UEFI + Guest OS

1) **End-to-end Exploitation Environment**
   1) UEFI Firmware
   2) Bootloader
   3) Windows OS
2) **Setup using VMWare for Virtualization**
   1) Graphics at last!
   2) Need better performance than QEMU
3) **Base firmware image provided by VMware**
   1) Custom techniques for automatically modding firmware/injecting drivers
   2) Teaching how to modify firmware (without compiling from source)



Roadmap: Level 4 and Beyond

1) **Short-term**
    1) Integrate latest binary protection into challenges (togglable)
    2) Increase catalog of real-world vulnerabilities
    3) Increase diversity of vulnerable apps to teach more vulnerability vectors
2) **Longer-term**
    1) SMI Emulation environment to showcase exploits for SMM drivers
    2) Bare-metal deployment (no virtualization needed!)

Hope to see you at DVUEFI soon!

https://github.com/hacking-support/DVUEFI