Breaking Thunderbolt Protocol Security: Vulnerability Report

Björn Ruytenberg
Eindhoven University of Technology
bjorn@bjornweb.nl

April 17, 2020

Changelog

Date	Description
Feb 10, 2020	First released version.
Mar 8, 2020	Added vulnerability 6 and exploitation scenarios 3.3.1–3.3.3.
Apr 17, 2020	Added vulnerability 7 and exploitation scenarios 3.4.1, 3.4.2.

Abstract

Thunderbolt is a proprietary I/O protocol promoted by Intel and included in a number of laptops, desktops, and other systems. As an external interconnect, it allows exposing the system's internal PCI Express (PCIe) domain to external devices. This enables high-bandwidth, low-latency use cases, such as external graphics cards. Being PCIe-based, Thunderbolt devices possess Direct Memory Access-enabled I/O, allowing complete access to the state of a PC and the ability to read and write all of system memory.

In recent years, the former characteristic has prompted research into attacks collectively known as evil maid, which require an attacker-controlled device and only seconds of physical access to the computer. Industry response has been two-fold. First, hardware and OS vendors incorporated support for DMA remapping using Input-Output Memory Management Units (IOMMUs), which imposes memory protections on DMA. However, following various implementation issues, OS vendors classified DMA remapping as an optional countermeasure requiring driver support. Second, revised Thunderbolt controllers introduced a software-based access control measure enabling users to authorize trusted devices only. As a result, unidentified devices should be barred from system access without prior user interaction.

In the context of Thunderbolt, studies have primarily focused on employing DMA and IOMMU attacks on the PCIe level. We therefore investigate the feasibility of breaking Thunderbolt protocol security, by analyzing the protocol and its software and hardware stack, as well as associated PCIe-based technology.

In our study, we have found and experimentally confirmed multiple vulnerabilities that break all primary Thunderbolt 3 security claims. Based on our on-going research, in this report, we disclose the following vulnerabilities: (i) inadequate firmware verification schemes, (ii) weak device authentication scheme, (iii) use of unauthenticated device metadata, (iv) backwards compatibility with legacy protocol versions, (v) use of unauthenticated controller configurations, (vi) SPI flash interface deficiencies, and (vii) no Thunderbolt security on Boot Camp. Finally, we present nine practical exploitation scenarios. Given an "evil maid" threat model and varying Security Levels, we demonstrate the ability to create arbitrary Thunderbolt device identities, clone user-authorized Thunderbolt devices, and finally obtain PCIe connectivity to perform DMA attacks. In addition, we show unauthenticated overriding of Security Level configurations, including the ability to disable Thunderbolt security entirely, and restoring Thunderbolt connectivity if the system is restricted to exclusively passing through USB and/or DisplayPort. We conclude this report by demonstrating the ability to permanently disable Thunderbolt security and block all future firmware updates.

Exploitation Scenario	Applicable Vulnerabilities	CVSS Score[1]
3.1.1	1–4	7.7
3.1.2	1–4	7.7
3.1.3	1–4	7.7
3.2.1	5	7.7
3.3.1	5–6	7.7
3.3.2	5–6	7.7
3.3.3	5–6	7.7
3.4.1	2–3	-
3.4.2	1–4, 7	7.7

Table 1: Exploitation scenarios corresponding to the disclosed vulnerabilities.

1 Introduction

In Thunderbolt, devices authenticate to the system using a collection of strings and numerical identifiers. Figure 1 shows device metadata enumerated on MacOS, Linux and Windows. As can be seen, device identification information includes [11]:

- **Device ID.** A 16-bit device identifier.

[1] CVSS v3.1. Calculated using AV:P/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H.

- **Device name.** An ASCII string representing the device name.
- **Vendor ID.** A 16-bit vendor identifier.
- **Vendor name.** An ASCII string representing the vendor name.
- **Unique ID / UUID.** A 64-bit numerical string uniquely identifying the device.

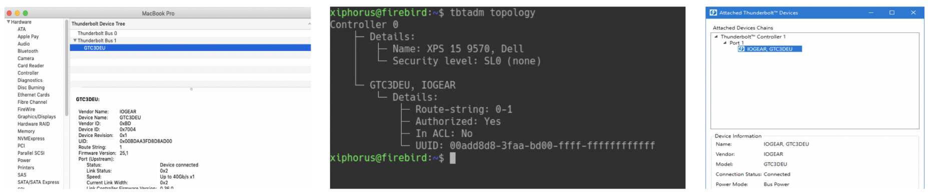


Figure 1: Thunderbolt device metadata enumeration.

Several studies have shown Thunderbolt to be a viable entry point in DMA attacks [3][6][7]. In response, Intel has introduced various countermeasures. A whitepaper on Thunderbolt security [4] details their threat model, and outlines how new security measures aim to prevent the former attacks given this model. In this section, we briefly summarize its scope.

In the whitepaper, Intel recognizes that Thunderbolt "(..) potentially allows access to system memory from a physical IO device utilizing the PCIe protocol". Countermeasures introduced in Thunderbolt therefore "(..) prevent unauthorized Thunderbolt PCIe-based devices from connecting without user authorization". To this end, Intel introduces software-based device authorization, using a user-configurable multi-step system referred to as Security Levels (SL). The whitepaper proceeds by defining the following Security Levels:

- **SL0 - None.** Similar to Thunderbolt 1, all attached devices are authorized without user interaction.
- **SL1 - User.** Access control based on device UUID. This is the default setting for all systems supporting Thunderbolt 2 onward.
- **SL2 - Secure.** Access control based on device UUID and challenge-response device authentication.
- **SL3 - No PCIe tunneling.** Switches the PHY to either DP-only, USB-only or mixed USB/DisplayPort mode.

To further strengthen device authentication, the *secure* security level provides "cryptographic authentication of connection[s] to help prevent a peripheral device to be spoofed to masquerade as an approved device to the user" [4].

As an extension to Security Levels, *pre-boot protection* is defined to ensure devices "(..) are allowed to be enumerated and connected at boot time only if they have been approved by the user before" [4]. Specifically, it is intended to prevent malicious devices from compromising sensitive UEFI data structures, including Security Levels configuration. This protection is disabled for devices that have been authorized by the user.

2 Vulnerability details

2.1 Overview

In our study, we have found and experimentally confirmed multiple vulnerabilities related to Thunderbolt protocol security. In this report, we disclose the following vulnerabilities:

1. **Inadequate firmware verification schemes.** Thunderbolt host and device controllers operate using updatable firmware stored in its SPI flash. Using this feature, Thunderbolt hardware vendors occasionally provide firmware updates online to address product issues post-release. To ensure firmware authenticity, upon writing the image to the flash, Thunderbolt controllers verify the firmware's embedded signature against Intel's public key stored in silicon. However, we have found authenticity is not verified at boot time, upon connecting the device, or at any later point. During our experiments, using a SPI programmer, we have written arbitrary, unsigned firmware directly onto the SPI flash. Subsequently, we have been able to verify Thunderbolt controller operation using our modified firmware.

2. **Weak device authentication scheme.** As noted in Section 1, device identification comprises several strings and numerical identifiers. However, we have found none of the identifiers are linked to the Thunderbolt PHY or one another, cryptographically or otherwise.

3. **Use of unauthenticated device metadata.** Thunderbolt controllers store device metadata in a firmware section referred to as *Device ROM (DROM)*. We have found that the DROM is not cryptographically verified. Following from the first issue, this vulnerability enables constructing forged Thunderbolt device identities. In addition, when combined with the second issue, forged identities may partially or fully comprise arbitrary data. Figure 2 demonstrates a device passing authentication while presenting a forged DROM to the host.

4. **Backwards compatibility.** Thunderbolt 3 host controllers support Thunderbolt 2 device connectivity, irrespective of Security Levels. Such backwards compatibility subjects Thunderbolt 3-equipped systems to vulnerabilities introduced by Thunderbolt 2 hardware.

5. **Use of unauthenticated controller configurations.** In UEFI, users may choose to employ a Security Level different to the default value (SL1) as listed in Section 1. In storing Security Level state, we have determined

that Thunderbolt employs two state machines, with one instance being present in UEFI, and another residing in host controller firmware. However, we have found firmware configuration authenticity is not verified at boot time, upon resuming from sleep, or at any later point. In addition, we have found these states machines may be subjected to desynchronization, with controller firmware overriding UEFI state without being reflected in the latter. As such, this vulnerability subjects the Thunderbolt host controller to unauthenticated, covert overriding of Security Levels configuration.

6. **SPI flash interface deficiencies.** As noted before, Thunderbolt systems rely on SPI flash to store controller firmware (vulnerability 1) and maintain their Security Level state (vulnerability 5). In our study, we have found Thunderbolt controllers lack handling hardware error conditions when interacting with flash devices. Specifically, we have determined enabling flash write protection (i) prevents changing the Security Level configuration in UEFI, again without being reflected in the latter, and (ii) prevents controller firmware from being updated, without such failures being reflected in Thunderbolt firmware update applications. As such, when combined with the fifth issue, this vulnerability allows to covertly, and permanently, disable Thunderbolt security and block all future firmware updates.

7. **No Thunderbolt security on Boot Camp.** Apple supports running Windows on Mac systems using the Boot Camp utility [2]. Aside from Windows, this utility may also be used to install Linux. When running either operating system, Mac UEFI disables all Thunderbolt security by employing the Security Level "None" (SL0). As such, this vulnerability subjects the Mac system to trivial Thunderbolt-based DMA attacks.

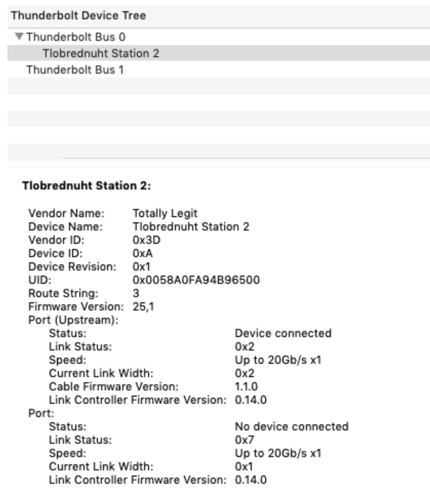


Figure 2: A Thunderbolt device presenting a forged DROM.

2.2 Affected configurations

2.2.1 PC systems

- **Hardware:** All laptop and desktop systems equipped with a Thunderbolt 2 and/or Thunderbolt 3 family host controller employing Security Levels[2].
- **Operating systems:**

  Microsoft Windows: 7, 8.1, and all Windows 10 builds currently available. As of writing, this includes the latest 1909 release.

  Linux: All kernel releases from 4.13 onward [11][12]. Previous versions, starting from 3.17, are trivially affected; these provide Thunderbolt support without implementing Security Levels [9], and hence require SL0 to operate.

2.2.2 Apple Mac systems

- **Hardware:** All Apple Mac systems equipped with a Thunderbolt 2 and/or Thunderbolt 3 family host controller.
- **Operating systems:**

  Microsoft Windows (Boot Camp): As PC systems.

  Linux (Boot Camp): As PC systems.

[2] Systems supporting Kernel DMA Protection [8] in place of Security Levels, released from 2019 onward, are currently subject to further investigation.

MacOS: Regarding Thunderbolt security, MacOS employs (i) an Apple-curated whitelist in place of Security Levels [7], and (ii) IOMMU virtualization when hardware and driver support is available [1][3]. Vulnerabilities 2–3 enable bypassing the first protection measure, and fully compromising authenticity of Thunderbolt device metadata in MacOS "System Information". However, the second protection measure remains functioning and hence prevents any further impact on victim system security via DMA. The system becomes vulnerable to attacks similar to BadUSB [10]. Therefore, MacOS is partially affected.

Vulnerability ID	Affects MacOS	Affects Windows	Affects Linux
1	No	Yes	Yes
2	Partially	Yes	Yes
3	Partially	Yes	Yes
4	No	Yes	Yes
5	No	No	No
6	No	No	No
7	No	Yes	Yes

Table 2: Vulnerabilities affecting Apple Mac systems when running MacOS, as well as Windows and Linux when using Boot Camp.

2.3 Partial mitigations for PC systems

As vulnerabilities 5 and 6 enable (permanent) unauthenticated overriding of Security Levels configuration, it is not sufficient to modify the Thunderbolt host controller Security Level to disable PCIe tunneling (SL3). Until a fix has been made available, we recommend disabling all Thunderbolt 2 and 3 host controllers in UEFI (BIOS) entirely.

Once a fix has been issued for the former vulnerabilities, we strongly encourage users to follow the instructions below to mitigate vulnerabilities 1 to 4.

On Thunderbolt 3 systems, we recommend changing the Thunderbolt controller Security Level to disable PCIe tunneling (SL3). Affected users may choose one of the following options in UEFI:

- **USB passthrough mode.** The Thunderbolt port operates as a USB 3.1 Type C interface, that is, without DisplayPort (DP) alt-mode support.
- **DisplayPort only mode.** In this mode, the Thunderbolt 3 port behaves identical to the video-only DP interface.
- **Mixed USB/DisplayPort mode.** Enables support for mixed mode devices, such as USB Type C hubs with DisplayPort sinks. In this mode, the Thunderbolt 3 port operates as a USB 3.1 Type C interface supporting DP alt-mode.

Similarly, on Thunderbolt 2 systems, we recommend changing the Thunderbolt Security Level to "DisplayPort only" (SL3).

Alternatively, users willing to permanently disable PCIe tunneling may use vulnerability 6 to render SL3 permanent.

Please note that disabling PCIe tunneling will prevent native Thunderbolt devices from operating. Additionally, we would like to emphasize the former suggestions are provided exclusively under the assumption that Intel's Thunderbolt Security Levels specifications are accurate.

2.4 Partial mitigations for Apple Mac systems

Vulnerability 7 disables all Thunderbolt security on Apple Mac systems when running Windows or Linux using the Boot Camp utility. Therefore, users are strongly encouraged to avoid using either operating system until a fix has been issued to address this vulnerability.

3 Exploitation

This section outlines a number of exemplary scenarios in which the identified vulnerabilities can be exploited. Given varying real-world threat models, we present four categories of exploitation scenarios. For vulnerabilities 1 to 4, the first category demonstrates cloning the identity of a user-authorized Thunderbolt 3 device to an arbitrary, attacker-controlled device. For vulnerability 5, the second category demonstrates unauthenticated overriding of Security Levels configuration. After completing a procedure from either category, the attacker-controlled device will obtain DMA access to the victim system without prior user authorization. For vulnerability 6, the third category outlines three methods to render overriding Security Levels permanent, as well as block all future firmware updates. Finally, for vulnerabilities 1–3 and 7, the fourth category demonstrates trivially obtaining DMA access to an Apple Mac victim system running Windows or Linux. It also demonstrates their limited impact on MacOS. We define the following keywords:

SPI programmer A SPI programmer that can be used to interface with a Thunderbolt controller's SPI flash.

Victim system The victim's system equipped with a Thunderbolt 3 interface.

Victim device A Thunderbolt 3 device authorized by the user to connect to the victim system.

Attacker device For scenarios 3.1.1–3.1.3, an arbitrary Thunderbolt 2 device which will spoof the victim device's identity. For scenarios 3.2.1-3.4.2, an arbitrary Thunderbolt 2 or 3 device.

Attacker system An attacker-controlled system equipped with a Thunderbolt 3 interface.

Tcfp Thunderbolt 3 Controller Firmware Patcher. A Python-based script we have developed that enables parsing and patching host controller firmware images.

SPIblock A Python-based script we have developed that enables configuring on-flash write protection.

3.1 Exploitation scenarios for vulnerabilities 1–4

3.1.1 Cloning victim devices with physical device access (SL1)

Threat model

We assume an "evil maid" threat model, in which the attacker has physical access to a victim device as well as the victim system. The system is in a locked (S0) or sleep state (S3). The Thunderbolt controller is set to employ the default Security Level "User".

Procedure

1. Connect the victim device to the attacker system. On the attacker system, using e.g. `tbtadm` on Linux, obtain the UUID of the victim device.
2. Disassemble the attacker device enclosure. Obtain the firmware image from the Thunderbolt controller's SPI flash of the attacker device.
3. Locate the DROM section by searching for the string `DROM` in the attacker device firmware image. Figure 6 depicts the DROM data structure. Using the figure as a reference, locate the appropriate offset and replicate the victim device UUID.
4. Compute `uid_crc8` and replicate the value at the appropriate offset.
5. Write the image to the attacker device SPI flash.

Verification

1. Connect the attacker device to the victim system. Observe that the victim system identifies the attacker device as being the victim device, and enables PCIe tunneling as a result. Figure 3 and Figure 4 demonstrate an example scenario, cloning the device identity of a NetStor NVMe enclosure to a CalDigit Thunderbolt dock.

3.1.2 Cloning victim devices without physical device access (SL1)

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has access to the victim system. More specifically, the attacker is not in possession of any victim devices. Furthermore, we assume the system is in a sleep state (S3), while the Thunderbolt controller is set to employ the default Security Level "User".

Procedure

1. Disassemble the case of the victim system. Obtain the firmware image from the Thunderbolt host controller's SPI flash.

2. Locate the Thunderbolt device ACL section in the firmware image between the offsets 0x0000–0x2000. The exact location will depend on the controller model, firmware revision, and the number of authorized devices. The indicated region primarily stores the ACL.

3. In the ACL data structure, identify a target victim device's UUID.

4. Disassemble the attacker device enclosure. Obtain the firmware image from the Thunderbolt controller's SPI flash of the attacker device.

5. Locate the DROM section by searching for the string `DROM` in the attacker device firmware image. Figure 6 depicts the DROM data structure. Using the figure as a reference, locate the appropriate offset and replicate the victim device UUID.

6. Compute `uid_crc8` and replicate the value at the appropriate offset.

7. Write the image to the attacker device SPI flash.

Verification

1. Connect the attacker device to the victim system. Observe that the victim system identifies the attacker device as being the victim device, and enables PCIe tunneling as a result.

3.1.3 Cloning victim device including challenge-response keys (SL2)

Threat model

We assume an "evil maid" threat model, in which the attacker has physical access to a victim device as well as the victim system. The system is in a locked (S0) or sleep state (S3). The Thunderbolt controller is set to employ the Security Level "Secure".

Procedure

1. Connect the victim device to the attacker system. On the attacker system, using e.g. `tbtadm` on Linux, obtain the UUID of the victim device.

2. Disassemble the victim device enclosure. Obtain the firmware image from the Thunderbolt controller's SPI flash of the victim device.

3. Locate the challenge-response key section in the firmware image between the offsets 0x1910–0x2000. The exact location will depend on the controller model, firmware revision, and the number of previously connected hosts. The indicated region exclusively stores the challenge-response keys in a dictionary data structure, comprising (i) the victim host controller UUID, and (ii) the corresponding challenge-response key.

4. Obtain the firmware image from the Thunderbolt controller's SPI flash of the attacker device.

5. In the attacker device firmware image, replicate the content from step 3 at the first available host–key slot between the offsets 0x1910–0x2000.

6. Locate the `DROM` section in the attacker device firmware image by searching for the string `DROM`. Figure 6 depicts the DROM data structure. Using the figure as a reference, locate the appropriate offset and replicate the victim device UUID.

7. Compute `uid_crc8` and store the value at the appropriate offset.

8. Write the image to the attacker device SPI flash.

Verification

1. Connect the attacker device to the victim system. Observe that the victim system identifies the attacker device as being the victim device, and enables PCIe tunneling as a result. Figure 5 illustrates spoofing the device identity of an IOGEAR eSATA-to-Thunderbolt dongle on a CalDigit Thunderbolt dock.

3.2 Exploitation scenario for vulnerability 5

3.2.1 Disabling Thunderbolt security (SL1/SL2), or restoring Thunderbolt connectivity when disabled (SL3)

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has access to the victim system. More specifically, the attacker is not in possession of any victim devices. Furthermore, we assume the system is in a sleep state (S3), while the Thunderbolt controller is set to employ either the Security Level "User", "Secure" or "No PCIe tunneling".

Procedure

1. Disassemble the case of the victim system. Obtain the firmware image from the Thunderbolt host controller's SPI flash.

2. On the attacker system, open a terminal window. Using *tcfp*, verify the image obtained in step 1 parses correctly:
   `$ python3 tcfp.py parse image.bin`

   Figure 7 shows example output for various valid firmware images.

3. When verified successfully, use *tcfp* to patch the firmware image to override the host controller Security Level to "None" (SL0):
   `$ python3 tcfp.py patch image.bin`

   Figure 8 shows example output for a host controller originally configured to employ the default Security Level "User" (SL1). Similarly, Figure 9 lists example output for a host controller initially configured to disable Thunderbolt connectivity, that is, exclusively pass through DisplayPort and USB (SL3).

4. Write the image to the victim system SPI flash.

Verification

1. Resume the victim system from sleep and connect the attacker device. If the victim system employed SL1 or SL2, observe that Thunderbolt security has been disabled. If the victim system employed SL3, notice that Thunderbolt connectivity has been restored and its security has been disabled. In both cases, observe that the victim system enables PCIe tunneling as a result.

2. Optionally, in UEFI, verify that the indicated Security Level does not reflect the actual Security Level state, that is, "None" (SL0).

   If the victim system is using Windows, verify the actual state is listed under the "About" section of the Thunderbolt Software device authorization UI. Similarly, on Linux, verify the actual state is listed when invoking `tbtadm`.

3.3 Exploitation scenarios for vulnerability 6

3.3.1 Soft-block method: Rendering Security Level "None" (SL0) permanent and blocking future firmware updates

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has access to the victim system. More specifically, the attacker is not in possession of any victim devices. Furthermore, we assume the system is in a sleep state

(S3), while the Thunderbolt controller is set to employ either the Security Level "User", "Secure" or "No PCIe tunneling".

Procedure

1. Execute procedure 3.2.1 to override the victim system's Security Level to "None" (SL0).

2. On the attacker system, open a terminal window. Using *SPIblock*, probe for the flash device:
   `$ python3 spiblock.py -p`

   Figure 10 shows example output for various flash devices.

3. Using *SPIblock*, observe `Status Register` returns a clean state (`0x00`), implying block protection is disabled:
   `$ python3 spiblock.py -s`

   Figure 10 shows example output for various flash devices.

4. When device connection and state have been verified successfully, enable on-flash block protection `BPx`:
   `$ python3 spiblock.py -b 1`

5. Observe block protection `BPx` has been enabled:
   `$ python3 spiblock.py -s`

   Figure 11 shows example output for a Winbond W25Q80.V.

Verification

1. On the victim system, in UEFI, modify the host controller Security Level to any value other than "None" (SL0).

2. Reboot to UEFI, and verify the host controller Security Level is set to the desired value.

3. Continue booting to the OS, and connect the attacker device. Observe that the host controller Security Level has not been set to the chosen value from step 1. Instead, observe the actual state is "None" (SL0), and that the victim system enables PCIe tunneling as a result.

4. From the victim system's vendor website, obtain a Thunderbolt host controller firmware update.

5. If the victim system is using Windows, record the current "NVM version" listed under the "About" section of the Thunderbolt Software device authorization UI.
   If the victim system is using Linux, record the current "NVM version"

using:
`# cat /sys/bus/thunderbolt/devices/0-0/nvm_version`

6. Install the Thunderbolt host controller firmware update obtained in step 4.

7. Observe the former utility reports the Thunderbolt firmware update was updated successfully. For reference, an Intel whitepaper [5] shows example output for Thunderbolt 3-equipped NUC systems.

8. If the victim system is using Windows, open the "About" section of the Thunderbolt Software device authorization UI. Notice that the "NVM version" remains unchanged.
   Similarly, on Linux, notice the value returned by `nvm_version` has not changed.

3.3.2 Hard-block method: Rendering Security Level "None" (SL0) permanent and blocking future firmware updates

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has access to the victim system. More specifically, the attacker is not in possession of any victim devices. Furthermore, we assume the system is in a sleep state (S3), while the Thunderbolt controller is set to employ either the Security Level "User", "Secure" or "No PCIe tunneling".

Procedure

1. Execute procedure 3.3.1 to override the victim system's Security Level to "None" (SL0) and enable block protection.

2. On the attacker system, using *SPIblock*, enable WP pin control:
   `$ python3 spiblock.py -w 1`

3. Observe Status Register Protect `SRP0` has been enabled:
   `$ python3 spiblock.py -s`

   Figure 12 shows example output for a Winbond W25Q80.V.

4. On the SPI flash device, solder a wire from the `GND` pin to the `WP` pin.

Verification

1. Proceed executing the verification steps from procedure 3.3.1 to observe the same results.

2. On the attacker system, using *SPIblock*, observe neither block protection nor WP pin control can be disabled. Figure 12 shows example output for a Winbond W25Q80.V.

3.3.3 One Time Program method: Rendering Security Level "None" (SL0) permanent and blocking future firmware updates

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has access to the victim system. More specifically, the attacker is not in possession of any victim devices. Furthermore, we assume the system is in a sleep state (S3), while the Thunderbolt controller is set to employ either the Security Level "User", "Secure" or "No PCIe tunneling".

In addition, we assume the victim system's host controller SPI flash supports One Time Program (OTP) write protection or similar technologies[3]. Once enabled, this feature programs the SPI flash device to an irrevocable read-only state.

Procedure

1. Execute procedure 3.3.1 to override the victim system's Security Level to "None" (SL0) and enable block protection.
2. Obtain the SPI flash datasheet from the vendor website.
3. On the attacker system, follow the instructions from the datasheet to enable OTP write protection. For example, for Winbond W25Q80.V [13], perform the following procedure:
   (a) Assert Write Enable Latch (`WEL`).
   (b) Write the value `1` to Status Register Protect 0 and 1 (`SRP0`, `SRP1`).
   (c) Disable Write Enable Latch (`WEL`).

Verification

1. Proceed executing the verification steps from procedure 3.3.1 to observe the same results.
2. On the attacker system, using *SPIblock*, observe neither block protection nor WP pin control can be disabled.

[3] In our study, we have found various recent and legacy Thunderbolt 3 implementations use a variant of Winbond W25Q80.V, which supports OTP write protection [13].

3.4 Exploitation scenarios for vulnerabilities 2-3, 7 on Apple Mac systems

3.4.1 Cloning an Apple-whitelisted device identity to an attacker device (MacOS) [4]

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has physical access to a victim system. The system is in a locked (S0) or sleep (S3) state, while running MacOS.

Preparation

1. Acquire a MacOS-certified Thunderbolt device.
2. Disassemble the MacOS-certified device enclosure. Obtain the firmware image from the Thunderbolt controller's SPI flash of the MacOS-certified device.
3. Disassemble the attacker device enclosure. Obtain the firmware image from the Thunderbolt controller's SPI flash of the attacker device.
4. Connect the MacOS-certified device to the attacker system. On the attacker system, using e.g. `tbtadm` on Linux, obtain the UUID of the MacOS-certified device.
5. Locate the DROM section by searching for the string `DROM` in the attacker device firmware image. Figure 6 depicts the DROM data structure. Using the figure as a reference, locate the appropriate offsets and replicate the MacOS-certified device UUID.
6. Compute `uid_crc8` and replicate the value at the appropriate offset.
7. Write the image to the attacker device SPI flash.

Procedure

1. Connect the attacker device to the victim system.

Verification

1. Observe that the victim system identifies the attacker device as being a MacOS-certified device. Figure 2 demonstrates an example scenario, showing a forged Thunderbolt device identity in the MacOS "System Information" application.

[4] This exploitation scenario is exclusively provided to demonstrate that, on MacOS, vulnerabilities 2–3 (i) bypass Thunderbolt device whitelist protection as detailed in Section 2.2.1, (ii) fully compromise authenticity of Thunderbolt device metadata in MacOS "System Information", and (iii) prevents impact on victim system security via DMA, but enables attacks similar to BadUSB [10].

3.4.2 Connecting an attacker device to DMA attack Apple Mac systems (Windows, Linux)

Threat model

We assume an "evil maid" threat model, in which the attacker *exclusively* has physical access to a victim system. The system is in a locked (S0) or sleep state (S3), while running Windows or Linux.

Procedure

1. Connect the attacker device to the victim system. Observe that the system does not provide any Thunderbolt security, and enables PCIe tunneling as a result.

Verification

1. If the victim system is using Windows, open the "About" section of the Thunderbolt Software device authorization UI. Notice that the Thunderbolt controller is set to employ Security Level "None" (SL0).
   Similarly, on Linux, notice the Security Level returned by `tbtadm topology` states "None" (SL0).

References

[1] Apple. Debugging VT-d I/O MMU Virtualization – Thunderbolt Device Drivers, 2013. URL https://developer.apple.com/library/archive/documentation/HardwareDrivers/Conceptual/ThunderboltDevGuide/DebuggingThunderboltDrivers/DebuggingThunderboltDrivers.html#//apple_ref/doc/uid/TP40011138-CH6-SW2.

[2] Apple. Boot Camp - Official Apple Support, 2020. URL https://support.apple.com/boot-camp.

[3] Ulf Frisk. Rise of the Machines: Direct Memory Attack the Kernel – DefCon 24. URL https://raw.githubusercontent.com/ufrisk/presentations/master/DEFCON-24-Ulf-Frisk-Direct-Memory-Attack-the-Kernel-Final.pdf.

[4] Intel Corporation. Thunderbolt 3 and Security on Microsoft Windows 10 Operating System, September 2018. URL https://thunderbolttechnology.net/security/Thunderbolt%203%20and%20Security.pdf. Whitepaper.

[5] Intel Corporation. Thunderbolt 3 Firmware Update Guide, July 2019. URL https://www.intel.com/content/dam/support/us/en/documents/mini-pcs/Thunderbolt-FW-Update-Guide.pdf. Whitepaper.

[6] Carsten Maartmann-Moe. Inception, 2014. URL https://github.com/carmaa/inception. GitHub repository.

[7] A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson. Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019. URL http://thunderclap.io/thunderclap-paper-ndss2019.pdf.

[8] Microsoft. Kernel DMA Protection for Thunderbolt 3, 2019. URL https://docs.microsoft.com/en-us/windows/security/information-protection/kernel-dma-protection-for-thunderbolt. Technical report documenting Windows 10 Kernel DMA Protection support for recently released systems.

[9] Andreas Noever. thunderbolt: Add initial cactus ridge NHI support, June 2014. URL https://github.com/torvalds/linux/commit/16603153666d22df544ae9f9b3764fd18da28eeb. Linux kernel patch adding initial Thunderbolt support.

[10] SRLabs. BadUSB - On accessories that turn evil, July 2014. URL https://srlabs.de/wp-content/uploads/2014/11/SRLabs-BadUSB-Pacsec-v2.pdf. PacSec.

[11] Mika Westerberg. Thunderbolt security levels and NVM firmware upgrade, June 2017. URL https://lwn.net/Articles/724465/. Details on initial Linux Thunderbolt security levels support.

[12] Mika Westerberg and Greg Kroah-Hartman. thunderbolt: Add support for Internal Connection Manager (ICM), June 2017. URL https://github.com/torvalds/linux/commit/f67cf491175a315ca86c9b349708bfed7b1f40c1. Linux kernel patch adding Thunderbolt security levels support.

[13] Winbond. W25Q80DV-DL 2.5V and 3V 8M-bit Serial Flash Memory with Dual and Quad SPI, October 2015. URL https://www.winbond.com/resource-files/w25q80dv%20dl_revh_10022015.pdf. Datasheet.

Appendix: Exploitation scenarios

This appendix demonstrates some of the exploitation scenarios, as outlined in Section 3.

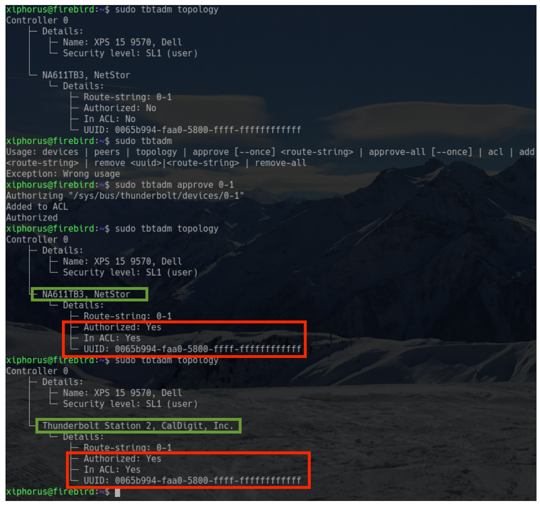


Figure 3: An attacker-controlled device spoofing the identity of a victim device on Linux.

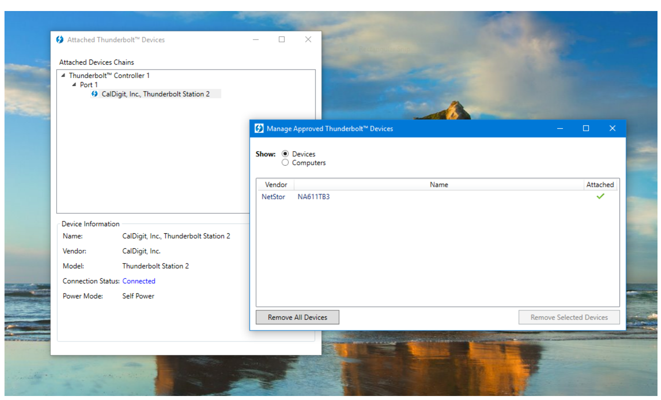


Figure 4: An attacker-controlled device cloning the identity of a victim device on Windows.

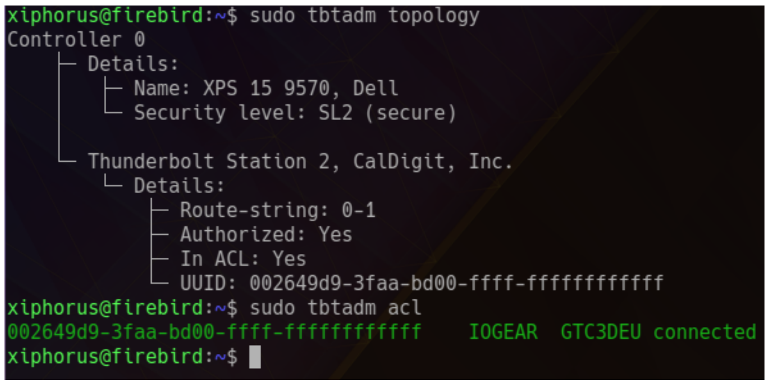


Figure 5: Spoofing a victim device while employing the “Secure” Security Level.

```
struct tb_drom_header {
        /* BYTE 0 */
        u8 uid_crc8; /* checksum for uid */
        /* BYTES 1-8 */
        u64 uid;
        /* BYTES 9-12 */
        u32 data_crc32; /* checksum for data_len bytes starting at byte 13 */
        /* BYTE 13 */
        u8 device_rom_revision; /* should be <= 1 */
        u16 data_len:10;
        u8 __unknown1:6;
        /* BYTES 16-21 */
        u16 vendor_id;
        u16 model_id;
        u8 model_rev;
        u8 eeprom_rev;
} __packed;
```

Figure 6: Device ROM data structure.

```
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$ python3 tcfp.py parse samples/intel-nuc8i3beh-M45PE80-nvm33-user.bin
Vendor ID : 0x8086
PCI ID : 0x15da
PCI Device Name : JHL6340 Thunderbolt 3 Bridge (C step) [Alpine Ridge 2C 2016]
Model ID : 0x6357
NVM version : 1 (0x1)
Vendor : Intel Corporation
Device : NUC8BEB
Security Level : SL1

0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$ python3 tcfp.py parse samples/hp-zbook-studio-g4-W25Q80.V-nvm41-secure.bin
Vendor ID : 0xf0
PCI ID : 0x15d3
PCI Device Name : JHL6540 Thunderbolt 3 Bridge (C step) [Alpine Ridge 4C 2016]
Model ID : 0x826b
NVM version : 1 (0x1)
Vendor : HP, Inc.
Device : HP ZBook Studio G4
Security Level : SL2

0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$ python3 tcfp.py parse samples/lenovo-p1-new-MX25L8005-nvm36-dp-usb.bin
Vendor ID : 0x109
PCI ID : 0x15ea
PCI Device Name : JHL7540 Thunderbolt 3 Bridge [Titan Ridge 4C 2018]
Model ID : 0x1711
NVM version : 36 (0x24)
Vendor : Lenovo
Device : ThinkPad P1
Security Level : SL3
```

Figure 7: Parsing Thunderbolt 3 host controller firmware images.

```
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$ python3 tcfp.py patch hp-zbook-studio-g4-W25Q80.V-nvm41-user.bin
Vendor ID : 0xf0
PCI ID : 0x15d3
PCI Device Name : JHL6540 Thunderbolt 3 Bridge (C step) [Alpine Ridge 4C 2016]
Model ID : 0x826b
NVM version : 1 (0x1)
Vendor : HP, Inc.
Device : HP ZBook Studio G4
Security Level : SL1

Image patched succesfully.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$ python3 tcfp.py parse hp-zbook-studio-g4-W25Q80.V-nvm41-user.bin
Vendor ID : 0xf0
PCI ID : 0x15d3
PCI Device Name : JHL6540 Thunderbolt 3 Bridge (C step) [Alpine Ridge 4C 2016]
Model ID : 0x826b
NVM version : 1 (0x1)
Vendor : HP, Inc.
Device : HP ZBook Studio G4
Security Level : SL0

0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/tcfp$
```

Figure 8: Altering firmware image to patch Security Level configuration from "User" (SL1) to "None" (SL0).

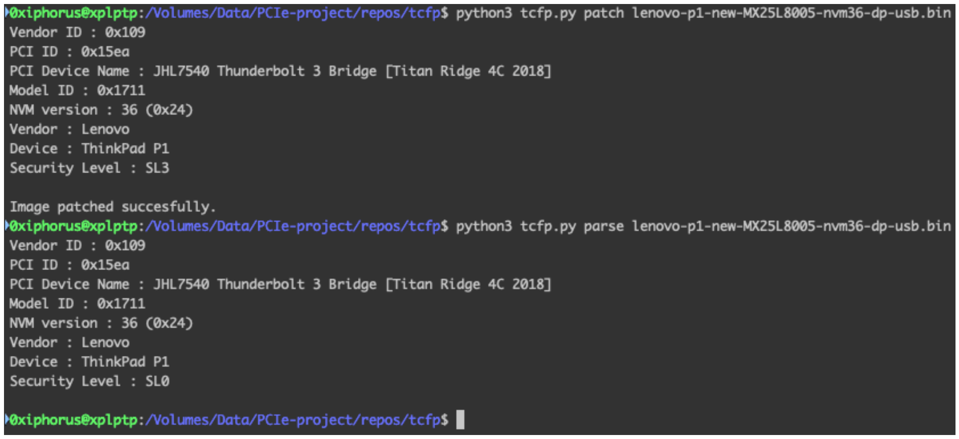


Figure 9: Altering firmware image to patch Security Level configuration from “No PCIe tunneling” (SL3) to “None” (SL0).

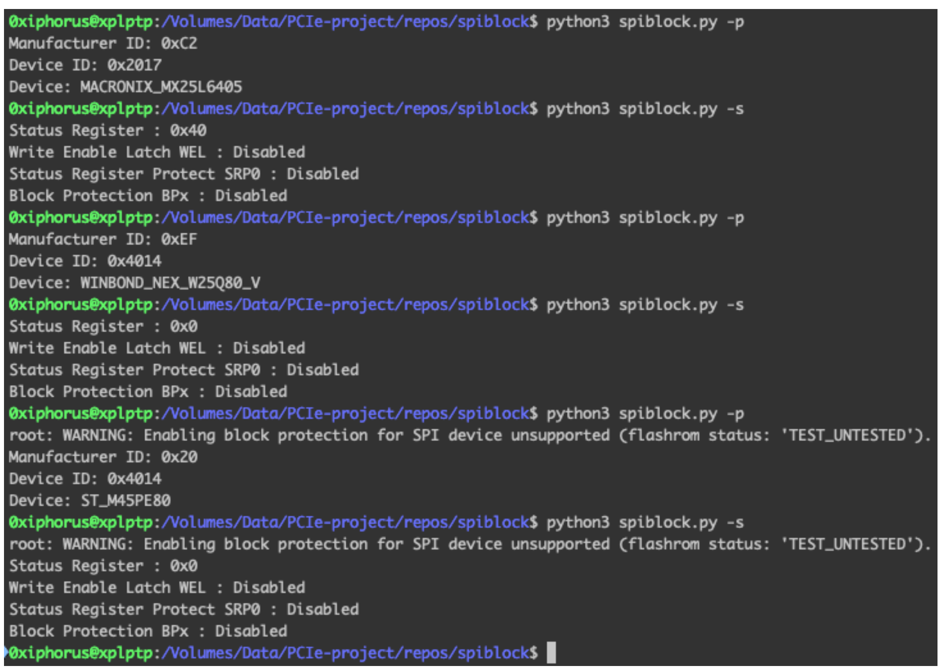
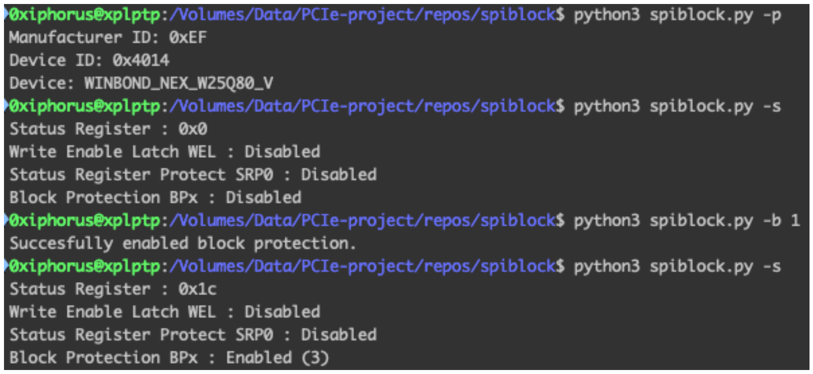
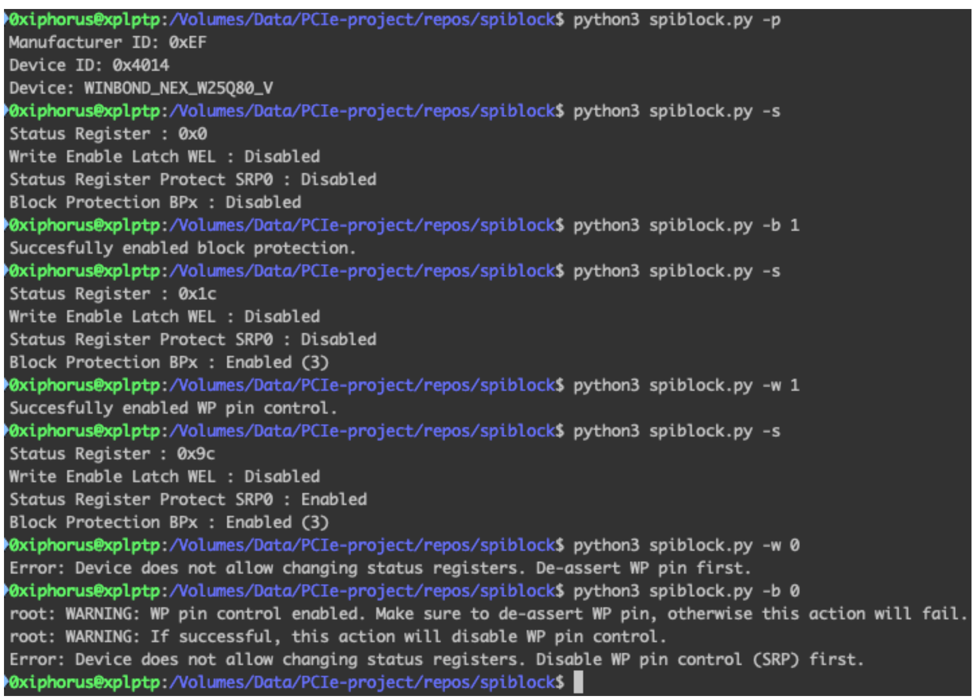


Figure 10: Verifying SPI flash connection and state.



```
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -p
Manufacturer ID: 0xEF
Device ID: 0x4014
Device: WINBOND_NEX_W25Q80_V
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -s
Status Register : 0x0
Write Enable Latch WEL : Disabled
Status Register Protect SRP0 : Disabled
Block Protection BPx : Disabled
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -b 1
Succesfully enabled block protection.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -s
Status Register : 0x1c
Write Enable Latch WEL : Disabled
Status Register Protect SRP0 : Disabled
Block Protection BPx : Enabled (3)
```

Figure 11: Enabling on-flash block protection.



```
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -p
Manufacturer ID: 0xEF
Device ID: 0x4014
Device: WINBOND_NEX_W25Q80_V
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -s
Status Register : 0x0
Write Enable Latch WEL : Disabled
Status Register Protect SRP0 : Disabled
Block Protection BPx : Disabled
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -b 1
Succesfully enabled block protection.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -s
Status Register : 0x1c
Write Enable Latch WEL : Disabled
Status Register Protect SRP0 : Disabled
Block Protection BPx : Enabled (3)
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -w 1
Succesfully enabled WP pin control.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -s
Status Register : 0x9c
Write Enable Latch WEL : Disabled
Status Register Protect SRP0 : Enabled
Block Protection BPx : Enabled (3)
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -w 0
Error: Device does not allow changing status registers. De-assert WP pin first.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$ python3 spiblock.py -b 0
root: WARNING: WP pin control enabled. Make sure to de-assert WP pin, otherwise this action will fail.
root: WARNING: If successful, this action will disable WP pin control.
Error: Device does not allow changing status registers. Disable WP pin control (SRP) first.
0xiphorus@xplptp:/Volumes/Data/PCIe-project/repos/spiblock$
```

Figure 12: Enabling on-flash block protection and WP pin control.