Achilles Heel in Secure Boot: Breaking RSA Authentication and Bitstream Recovery from Zynq-7000 SoC

Prasanna Ravi and Arpan Jati, *Temasek Laboratories, Nanyang Technological University, Singapore;* Shivam Bhasin, *National Integrated Centre for Evaluation* (*NiCE*), *Nanyang Technological University, Singapore*

https://www.usenix.org/conference/woot24/presentation/ravi

This paper is included in the Proceedings of the 18th USENIX WOOT Conference on Offensive Technologies.

August 12–13, 2024 • Philadelphia, PA, USA

ISBN 978-1-939133-43-4

Achilles Heel in Secure Boot: Breaking RSA Authentication and Bitstream Recovery from Zynq-7000 SoC

Prasanna Ravi[a] Arpan Jati[a] Shivam Bhasin[a,b]
[a] *Temasek Labs, Nanyang Technological University, Singapore*
[b] *National Integrated Centre for Evaluation (NiCE), Nanyang Technological University, Singapore*
Email: {prasanna.ravi, arpan.jati, sbhasin}@ntu.edu.sg

Abstract

Secure boot forms the backbone of trusted computing by ensuring that only authenticated software is executed on the designated platform. However, implementation of secure boot can have flaws leading to critical exploits. In this paper, we highlight a critical vulnerability in open source First Stage Boot Loader (FSBL) of AMD-Xilinx's flagship Zynq-7000 System on Chip (SoC) solution for embedded devices. The discovered vulnerability acts as a 'single point of failure' allowing complete bypass of the underlying bypass RSA authentication during secure boot. As a result, a malicious actor can take complete control of the device and run unauthenticated/malicious applications. We demonstrate an exploit using the discovered vulnerability in form of first practical 'Starbleed' attacks on Zynq-7000 devices to recover the decrypted bitstream from an encrypted (using AES-256) boot image. The identified flaw has existed in the secure-boot software for more than 10 years. The vulnerability was responsibly disclosed to the vendor under CVE 2022/23822. The vendor thereafter patched the FSBL software and issued a design advisory. Our work therefore motivates the need towards rigorous security evaluation tools to test for such trivial security vulnerabilities in software.

1 Introduction

Due to the demand of System on Chips in sensitive applications, they support various security features such as secure boot, device authentication, bitstream encryption, readback protection, etc. However, the robustness of these security features remains unclear due to a lack of proper documentation and third-party evaluation/scrutiny. In this work, we perform an in-depth analysis of the *RSA authentication feature* of the Zynq-7000 SoC from AMD-Xilinx. AMD-Xilinx Zynq-7000 SoCs have been a market leader in the integrated FPGA and processor market, with wide adoption across several industries such as automotive, aerospace, industrial, and healthcare sectors. We identified a critical *double fetch* security flaw in the RSA authentication feature within the First Stage Boot Loader (FSBL) provided by Xilinx. Its exploitation makes it possible to execute an unauthenticated software application on the Zynq-7000 SoC. The identified flaw is only present in the FSBL software and thus can be easily fixed through appropriate modification of the FSBL software.

Thus, the first contribution of our work is the *identification of a critical security flaw in the FSBL software to bypass RSA authentication.*

Upon bypassing RSA authentication, we utilize the unauthenticated software application to demonstrate a novel attack to recover the encrypted bitstream in the boot image, thereby subverting the bitstream encryption feature. To the best of our knowledge, there does not exist any prior work that has reported a bitstream recovery attack on the Zynq-7000 SoC. In this context, Ender et al. [3] in 2020 proposed the *Starbleed* attack, capable of breaking bitstream decryption on standalone Virtex-6 and 7-series Xilinx FPGAs. The design advisory from Xilinx as a response to the Starbleed attack claims that the Zynq-7000 SoC is resistant "due to the use of asymmetric and/or symmetric authentication in the boot/configuration process" [4]. Due to the security flaw found in the FSBL, we managed to identify a novel approach to mount the Starbleed attack on the Zynq-7000 device for full bitstream recovery.

Thus, as a second contribution of our work, *we present the first practical demonstration of the Starbleed attack on the Zynq-7000 SoC* with practical validation on PYNQ-Z1 platform

We have thus performed an end-to-end recovery of the bitstream exploiting the RSA bypass vulnerability and the Starbleed attack. We communicated our findings to Xilinx in a vulnerability disclosure on March 8, 2022.

Xilinx quickly confirmed the vulnerability on March 24, 2022, and also published a patch for the FSBL software on March 25, 2022 [6]. Information about the vulnerability was also published as a design advisory by Xilinx on April 28, 2022 [5]. Furthermore, we also investigated if the flaw in the FSBL software is also present in the BootROM code of the Zynq-7000 SoC. Analyzing the BootROM behavior presents significant challenges, since the BootROM code is unavailable or cannot be modified, as it is hard-coded within the SoC.

Thus, as a third contribution of our work, *we present a novel black-box analysis of the communication interface between the Zynq-7000 SoC and the NVM during BootROM execution.*

However, our analysis was able to positively confirm that the BootROM software does not suffer from the RSA vulnerability present in the FSBL.

Availability of Software

All the software used for this work is available in the public domain at the following link: https://github.com/PRASANNA-RAVI/RSA_Bypass_Vulnerability_Zynq_7000_SoC.

1.1 Threat Model

The boot image of the victim Zynq-7000 SoC device boots from a boot image stored in a Non-Volatile Memory (NVM) accessible to an attacker. The SoC typically consists of two components: (1) Programmable System (PS) which refers to the dual-core ARM Cortex-A9 processor and (2) Programmable Logic (PL) which refers to the FPGA fabric. The victim boot image has three partitions - FBSL, PL partition (bitstream to execute on the FPGA), and PS partition (software application to execute on the processor). The target device mandates RSA authentication of the boot image (i.e.) the RSA eFUSE is enabled, and all partitions in the victim boot image are encrypted as well as authenticated. This means that every partition has its corresponding RSA signature stored along with it, and is referred to as the Authentication Certificate (AC). Refer to Figure 1 for the structure of the victim boot image we consider for our attack. The attacker's goal is to execute an unauthenticated application on the Zynq-7000 SoC.

2 Background

We now provide a brief background on the secure boot feature of the Zynq-7000 SoC, to facilitate the understanding of our attack, described later in Sections 3-6.

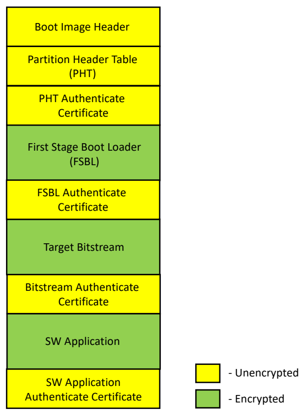
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
SW Application
SW Application Authenticate Certificate
- Unencrypted
- Encrypted

Figure 1: Authenticated victim boot image

2.1 Secure Boot of Zynq-7000 device

The central component of secure boot is the *secure boot image* which consists of various partitions that will be sequentially loaded securely into the appropriate locations within the Zynq-7000 SoC (either DDR, On-Chip Memory (OCM) or FPGA). The important components of a boot image are as follows:

- **Boot Image Header (BIH) and BootROM code**: The BootROM code is the first piece of software executed upon resetting the Zynq-7000 SoC. It is hard-coded onto the BootROM of the chip (and not part of the boot image), and cannot be modified. It initializes the device based on information in the BIH. Its main function is to retrieve the FSBL from the NVM, after which it authenticates the FSBL using its Authentication Certificate (AC that contains its RSA signature), and further decrypts the FSBL, before securely passing control to it.

- **Partition Header Table (PHT)**: The PHT is a critical component of the boot image, which contains metadata information about each PL and PS partition in the boot image. Each partition has an associated entry of 64 bytes in the PHT and contains information such as encrypted partition size, decrypted partition size, total partition size including its AC, destination address in the device, location within the boot image, authentication status, etc. The PHT is used by the FSBL to get information about each partition in the boot image. *We remark that the PHT is present unencrypted within the boot image allowing an attacker to gain information about the metadata of each partition in the secure boot image.*

- **First Stage Boot Loader (FSBL)**: The FSBL is responsible for loading each of the PS and PL partitions in the boot image into the appropriate locations within the device (i.e.) PL partition is loaded into the FPGA, and PS partition is loaded into the DDR memory. The FSBL first retrieves the PHT from the NVM and authenticates it using its AC. Upon successful authentication, then FSBL securely loads the PL and PS partitions individually in the same manner, from the boot image, based on information in the PHT. However, if PHT authentication fails, then the FSBL simply aborts the secure boot procedure. After loading all the PS and PL partitions, the FSBL transfers control to the last software application that was loaded from the boot image. In this work, we use the official FSBL code for the Zynq-7000 SoC provided by Xilinx (FSBL version 2018.1).

- **Programmable Logic (PL) or Programmable System (PS) Partition**: After the FSBL, the remaining portion of the boot image is occupied either by a PL partition or a PS partition. For an authenticated partition, there is an Authentication Certificate (AC), that contains its RSA signature, which is appended to it in the boot image.

2.2 RSA Authentication in Zynq-7000 SoC

The Zynq-7000 SoC uses the well-known RSA-2048-based signature scheme for authentication. It is done with two keys: the Primary Key and the Secondary Key. While the primary key is stored in the eFuse of the device (fixed for a given device), the secondary key is associated with each partition. The primary key is used to authenticate the secondary key of a partition, and the secondary key is used to authenticate the partition data itself, thereby forming an authentication chain. The authentication operation (i.e.) signature verification is carried out by a cryptographic software library, part of the BootROM and FSBL of the Zynq-7000 SoC. Since an understanding of the intricate details of RSA authentication is not required for our attack, we refer the reader to [10] for more details.

RSA authentication is an integral component of the FSBL. FSBL is an open-source and modifiable piece of software. We analyze the FSBL source code to understand how it authenticates various components of the boot image.

3 Analyzing the RSA Authentication Procedure within FSBL

We noticed that the PHT authentication serves as a single point of failure in the secure boot procedure. If an attacker can bypass PHT authentication, he/she can mount a tampered PHT that can be used to execute an unauthenticated application. We analyzed the PHT authentication procedure by FSBL (implemented within the `image_mover.c` source file in the *embeddedsw* project [9]). Refer to Figure 2 for a pictorial illustration of the authentication procedure of the PHT by the FSBL.

1. The FSBL first retrieves the PHT data from the NVM and stores it into a global variable denoted as `GVAR`. We denote the fetched PHT data from NVM as `PHT1`.

2. The FSBL then checks the status of the RSA eFUSE. If enabled, the FSBL again retrieves the PHT along with its AC. We denote the fetched PHT data as `PHT2` since it is retrieved at a different time than `PHT1`.

3. If verification of AC of `PHT2` is successful, then the FSBL uses the data in `GVAR` (`PHT1`) as the PHT to load the PS and PL partitions in the boot image.

In other words, the FSBL authenticates `PHT2` but uses the unauthenticated `PHT1` for secure boot. This is mainly because of the *double fetch* of the PHT data from the NVM which is external to the security boundary of the device. This is the critical vulnerability that we have identified that could be exploited to bypass PHT authentication.

We remark that our experiments were done on FSBL version 2018.1, they also applied to the latest FSBL version dated 23 Apr 2020[1], during the time of our research.

3.1 Related Works

Double fetch is a term referring to a bug that occurs when a process reads and uses the same value twice, expecting it to be identical while it is possible for an attacker to modify it between the two reads. This term was first coined by Serna [7], and there have been several works that have exploited double-fetch bugs in what is commonly referred to as Time-of-Check to Time-of-Use (TOCTTOU) attacks [1]. Well-known instances of such attacks include attacks on the Linux kernel [8], applications such as Firefox [11] and Intel BootGuard [2].

3.2 Exploiting the RSA Security Flaw in FSBL

We formulate an attack methodology to exploit the double fetch PHT, using an NVM emulator, which is configured to behave in the following manner during PHT authentication.

[1]https://github.com/Xilinx/embeddedsw/blob/master/lib/sw_apps/zynq_fsbl/src/image_mover.c

1. When the FSBL fetches the PHT for the first time (PHT1), the NVM emulator provides a tampered PHT, configured according to the attacker's requirements. Thus, the tampered PHT is stored in GVAR variable, within the DDR.

2. The FSBL then checks the status of RSA eFUSE and if enabled, again fetches the PHT (PHT2) along with its AC. This time, the NVM emulator provides the valid PHT along with its AC.

3. The FSBL successfully authenticates PHT2, but now uses the tampered PHT1 for secure boot present in GVAR. Based on the tampered PHT1, the FSBL loads an unauthenticated application on the target device, thereby bypassing RSA authentication.

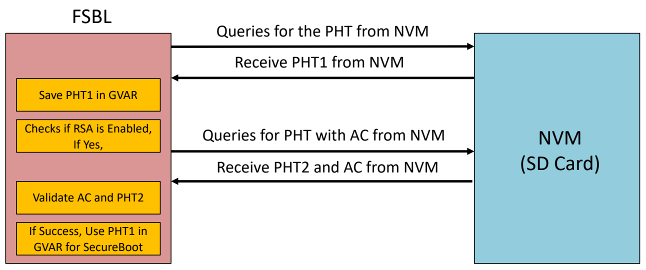
FSBL
Save PHT1 in GVAR
Checks if RSA is Enabled, If Yes,
Validate AC and PHT2
If Success, Use PHT1 in GVAR for SecureBoot
Queries for the PHT from NVM
Receive PHT1 from NVM
Queries for PHT with AC from NVM
Receive PHT2 and AC from NVM
NVM
(SD Card)

Figure 2: Authentication of the PHT by the FSBL

3.3 Proof of Concept (PoC) Attack Implementation

We started with a Proof of Concept (PoC) attack to demonstrate the presence of the double fetch vulnerability during PHT authentication. This was done not with an NVM emulator, but by performing manual modifications to the FSBL, to replicate the behavior of the emulator. We manually modified the PHT data in the GVAR variable after fetching PHT1, and the data is tampered with to load an unauthenticated PS partition (software application) from the boot image. This is the only modification done in the FSBL and does not aid the attack in any other manner. Within the boot image, the authenticated application is replaced with a malicious and unauthenticated application in the boot image. We ran repeated experiments using the tampered FSBL as well as the tampered boot image, and we were able to successfully load and execute the unauthenticated software application on the target device, which demonstrates the presence of the RSA bypass vulnerability.

However, this does not qualify as a real attack, since we made manual modifications to the FSBL that is encrypted within the boot image. Since the attacker does not know the encryption key, it is not possible in a real-attack scenario. In the following, we thus attempt to perform a practical real-world attack by building a low-cost NVM emulator, that does not require making modifications to the FSBL in the boot image.

4 Practical Attack using SD Card Switcher Board

One approach to carry out a practical attack would be to implement the NVM emulator on an FPGA/ASIC. However, designing the it requires significant engineering effort, and hence adopted a simpler approach. The basic requirement for our NVM emulator is to present a tampered PHT during the first fetch, and a valid PHT during the second fetch. To achieve this, we built an SD card switcher that can switch between two SD cards (SD Card 1 and SD Card 2) during the secure boot procedure.

The SD card switcher has two SD card slots, and we can choose the SD card to connect to the target, based on the logic level of a GPIO pin. The board also facilitates keeping the SD cards powered on from an external power source. This ensures that the SD card once initialized by the target device is powered on, even if the target device is powered off. In the following, we explain the proposed attack methodology using our SD card switcher board.

4.1 Attack Methodology

The two SD cards (SD cards 1, and 2) are loaded with two different attack boot images derived from the victim boot images. SD card 1 contains a boot image with the tampered PHT (i.e.) PHT1 (mapping to an unauthenticated attack application), while all the other contents match that of the victim SD card. SD card 2 contains a boot image with a valid PHT (PHT2) but with the authenticated software application replaced with the unauthenticated attack application. Refer to Figure 3 for a pictorial illustration of the boot images on both SD cards. We load both the SD cards onto the SD card switcher board and connect the SD card switcher to the Zynq-7000 SoC. The attack is carried out in the following manner:

1. The Zynq-7000 SoC first boots with SD card 2 mounted on the SD card switcher board. This is done to initialize SD card 2.

2. We now power off the Zynq-7000 SoC and switch to SD card 1. This is done while maintaining the power of both SD cards.

3. We now boot the Zynq-7000 SoC with SD card 1, which ensures that the tampered PHT (PHT1) during the first PHT fetch. After the first PHT fetch,

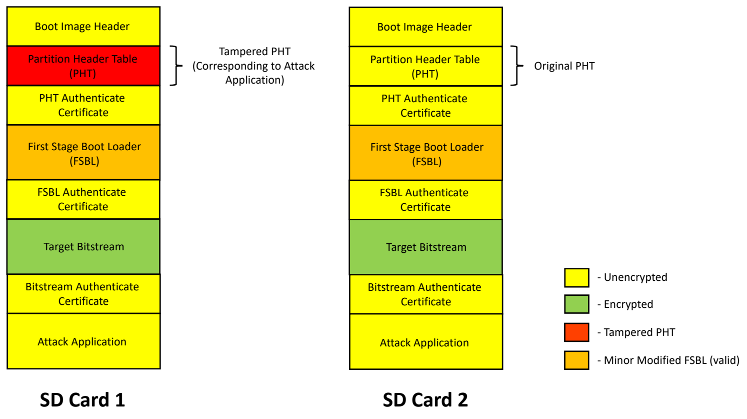
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
Attack Application
Tampered PHT (Corresponding to Attack Application)
SD Card 1
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
Attack Application
Original PHT
SD Card 2
- Unencrypted
- Encrypted
- Tampered PHT
- Minor Modified FSBL (valid)

Figure 3: Boot Images of SD card 1 and SD card 2 within the SD switcher board

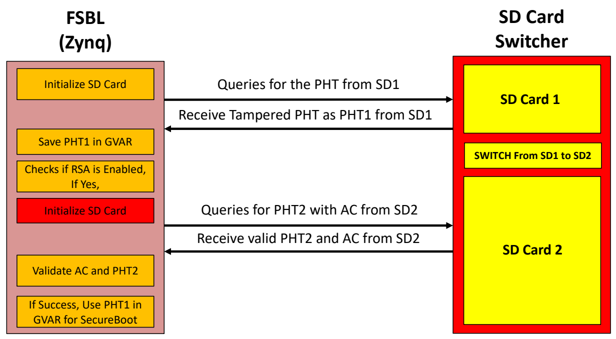
FSBL (Zynq)
Initialize SD Card
Save PHT1 in GVAR
Checks if RSA is Enabled, If Yes,
Initialize SD Card
Validate AC and PHT2
If Success, Use PHT1 in GVAR for SecureBoot
SD Card Switcher
SD Card 1
SWITCH From SD1 to SD2
SD Card 2
Queries for the PHT from SD1
Receive Tampered PHT as PHT1 from SD1
Queries for PHT2 with AC from SD2
Receive valid PHT2 and AC from SD2

Figure 4: Improved Attack on the Zynq-7000 SoC using SD card switcher, with the modification done to the FSBL denoted in red.

we switch from SD card 1 to SD card 2. For our experiments, we added a manual delay between the first and second PHT fetches. However, this can be automated as the timing of the switch is constant for the Zynq device upon power up.

4. After the switch, we expect that the Zynq-7000 SoC will retrieve the valid PHT from SD card 2 (which was already initialized), which should be authenticated successfully by the FSBL. This should also ensure that PHT1 is used for secure boot, and will therefore execute the unauthenticated on the Zynq-7000 SoC.

4.1.1 Experimental Observations of Attack using SD Card Switcher Board

Our experiments reveal that the Zynq-7000 SoC halts operation after switching from SD card 1 to SD card 2, after the first PHT fetch. The FSBL is unable to connect to SD card 2, even though it is initialized. We hypothesize that the SD card peripheral on the target device, which is oblivious to the switch tries to communicate with commands for SD card 1, while the switcher connects the device to SD card 2. To overcome this, we perform a minor modification to the FSBL, by adding to the InitSD function, to initialize SD card 2 after the switch. After this modification, we can successfully perform a bypass of the PHT authentication and load the unauthenticated application, demonstrating a successful RSA bypass. Refer to Figure 4 for a pictorial illustration of our improved attack using the SD card switcher.

Since our current setup still requires a modification to the FSBL, it does not qualify as a practical attack. We believe this limitation can be overcome using specialized hardware (FPGA/ASIC) to tamper the SD card interface at precise time instances. Nevertheless, our attack concretely exposes the flaw in the FSBL software to bypass RSA authentication.

4.2 Fixing the Flaw in PHT Authentication within FSBL

The vulnerability mainly arises from the retrieval of the same PHT data twice from the NVM and only using the data from the first fetch. This flaw can be patched by ensuring that PHT is only retrieved once from the NVM and authenticated immediately. This fix is implemented as part of the patched FSBL (dated March 25th, 2022) [6], and our manual analysis of the patched FSBL source code confirmed the removal of the double fetch vulnerability. AMD-Xilinx referred to our attack as a physical attack [6] and that the device "was not designed to be resistant to physical attacks". However, the identified vulnerability still exposes a critical flaw in the RSA authentication process, which enables a practical attack that enables it to completely bypass it. While we verified the BootROM of Zynq-7000 SoC for the same vulnerability, we have not analysed other devices from AMD-Xilinx, and we leave this analysis for future work. This is not the first time that such double fetches have been detected in secure software [1]. In the following section, we show that an attacker can use the unauthenticated application to perform a novel bitstream recovery attack.

5 Starbleed for Bitstream Recovery

Ender et al. [3] in 2020 exposed a critical security flaw in the bitstream decryption protocol of standalone Virtex-6 and 7-series Xilinx FPGAs, which enables recovery of bitstream data, now well-known as the *Starbleed* attack. The only requirement is that the attacker requires access to the configuration interface of the FPGA (PL). In this work, we adapt the Starbleed attack to the Zynq-7000 device for bitstream recovery.

5.1 Attack Methodology

The attacker makes malicious changes to the encrypted bitstream, such that upon decryption, a targeted decrypted bitstream word is written into the Warm Boot Status Address (WBSTAR) register of the configuration interface. The WBSTAR register retains its value even upon FPGA reset and thus an adversary can access the decrypted word from the WBSTAR register. Similarly, full bitstream recovery can be performed one word at a time. Since the attacker now has control of the PS (through the attack application), we designed an attack application to carry out the attack by accessing the PL through the PCAP (Processor Configuration Access Port) interface. The application programs the PL with the tampered bitstream, but it results in failure of HMAC integrity check, triggering a HMAC error. The reference manual claims that readback of any register (including WBSTAR) is not possible unless the PL is configured with a valid bitstream. Thus, it was evident that the Starbleed attack could not be performed as on the Zynq-7000 SoC.

5.1.1 Attack Execution using Workaround

We identified a workaround to ensure that register readback is possible, even after programming the PL with a tampered bitstream.

1. Program PL with a valid encrypted bitstream.

2. Without initializing the PL again, we push the tampered attack bitstream through the PCAP interface. We observe that FPGA stays programmed (DONE signal is high) even though the tampered bitstream trigger an HMAC error.

3. We then issue read command to successfully read the WBSTAR register containing the decrypted bitstream word.

This technique of programming bitstreams without initializing is not recommended practice. We typically expect that PL is not configured properly without initialization. We observe that the tampered bitstreams were able to write the decrypted bitstream word to the WBSTAR register while ensuring that readback is also possible. However, after readback, the PCAP interface becomes unresponsive, and only a Power-on Reset (PoR) of the device could bring it back to normal working condition. Thus, we can only recover one bitstream word per secure boot, and the attacker needs to power cycle the device to recover every bitstream word. We can recover the bitstream at a speed of 32 bits per second, and an estimated recovery time of 46 days for our experimental bitstream of size 3.85 MB. We observe that the secure boot time after every POR reset serves as a bottleneck for our attack. While reducing the attack time is possible, we consider performance acceleration out of the scope of our work. This attack would not be possible without bypassing RSA authentication, and thus using the patched version of the FSBL (dated March 25, 2022) [6] can serve as a strong mitigation against the bitstream recovery attack.

6 Conclusion

In this work, we have identified a critical double fetch security flaw in the FSBL software of AMD-Xilinx's Zynq-7000 SoC, which enables bypassing the RSA authentication procedure, to execute an unauthenticated application on the target device. We experimentally validated a potential exploit using a custom-built SD card switcher board. We also analyzed the BootROM code for a similar vulnerability, and confirm that the same bug is not present (Refer to Appendix A). We then proceeded to demonstrate the first successful bitstream recovery attack on the Zynq-7000 SoC using the Starbleed attack technique. In essence, our work uncovers a simple double fetch vulnerability in the secure boot software of Zynq-7000 SoC, but such vulnerabilities are not new. Our work demonstrates a serious need for automated tools for identifying such trivial bugs. While there have been proposals for such techniques [8], the applicability of these tools to embedded devices is to be studied and forms an interesting direction for future research.

Acknowledgement

This work was supported in part by the "National Integrated Centre of Evaluation" (NICE), and in part by a facility of the Cyber Security Agency (CSA), Singapore

References

[1] Atri Bhattacharyya, Uros Tesic, and Mathias Payer. Midas: Systematic kernel {TOCTTOU} protection. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 107–124, 2022.

[2] Peter Bosch and Trammell Hudson. Warpattack: bypassing cfi through compiler-introduced double-fetches. In *Hack in the Box Amsterdam*, 2019.

[3] Maik Ender, Amir Moradi, and Christof Paar. The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series fpgas. In *29th USENIX Security Symposium*, 2020.

[4] Xilinx Inc. Ar 73541-design advisory for 7 series/virtex-6 fpgas: Defeating bitstream encryption dated 04/27/2020.

[5] Xilinx Inc. Design advisory for zynq-7000: Fsbl authentication attack dated 04/28/2022. https://support.xilinx.com/s/article/76974?language=en_US.

[6] Xilinx Inc. Software patch of the fsbl software for zynq-7000 soc against fsbl authentication attack. https://github.com/Xilinx/embeddedsw/commit/28111bd377ec77e8cbb5492e5a0a4f4d37b6c5e3.

[7] Fermin J Serna. Ms08-061: The case of the kernel mode double-fetch, 2008.

[8] Pengfei Wang, Jens Krinke, Kai Lu, Gen Li, and Steve Dodier-Lazaro. How {Double-Fetch} situations turn into {Double-Fetch} vulnerabilities: A study of double fetches in the linux kernel. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1–16, 2017.

[9] Xilinx. Fsbl software for zynq-7000 soc (image_mover.c). https://github.com/Xilinx/embeddedsw/blob/master/lib/sw_apps/zynq_fsbl/src/image_mover.c.

[10] Xilinx. Zynq-7000 all programmable soc technical ref-erence manual-ug585 (vl. 13).

[11] Jianhao Xu, Luca Di Bartolomeo, Flavio Toffalini, Bing Mao, and Mathias Payer. Warpattack: bypassing cfi through compiler-introduced double-fetches. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1271–1288. IEEE, 2023.

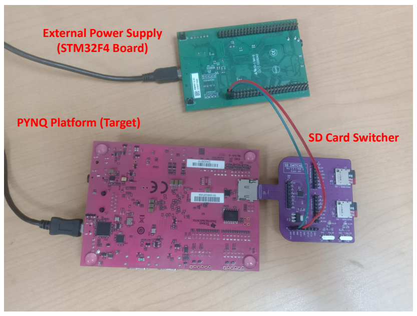

Figure 5: Attack Setup with SD Card Switcher

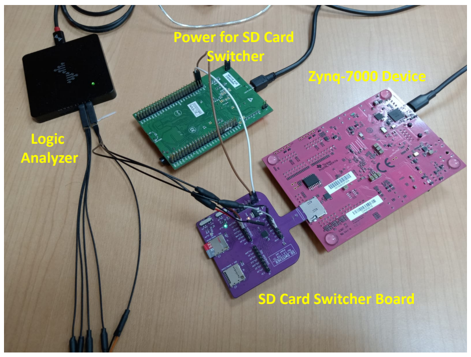

Figure 6: Experimental Setup: Logic Analyzer probing the SD Card interface to the Zynq-7000 device

A Assessing the RSA Authentication Procedure in BootROM

While we identified a flaw in the RSA authentication procedure in FSBL, we asked ourselves *whether the same flaw is also present in other operations during the secure boot*. We thus conducted a security analysis of the BootROM software, which also performs authentication of the FSBL itself, before FSBL starts execution. But, analysing the BootROM software is particularly challenging compared to the FSBL, since neither the BootROM source-code nor the binary is available. It is also hard-coded within the on-chip memory of the Zynq device, and hence cannot be modified. Thus, a code-analysis similar to that of FSBL is not possible. However, we observe that the BootROM loads data from the Non-Volatile Memory (NVM) (i.e.) SD card and thus monitoring the SD card interface during BootROM execution could provide critical information about its operation. In this respect, we utilize a logic analyzer to monitor the SD card communication during BootROM execution.

A.1 BootROM Analysis using SD Card Communication

In order to understand the data transfer between the SD Card (NVM) and the Zynq-7000 SoC during BootROM

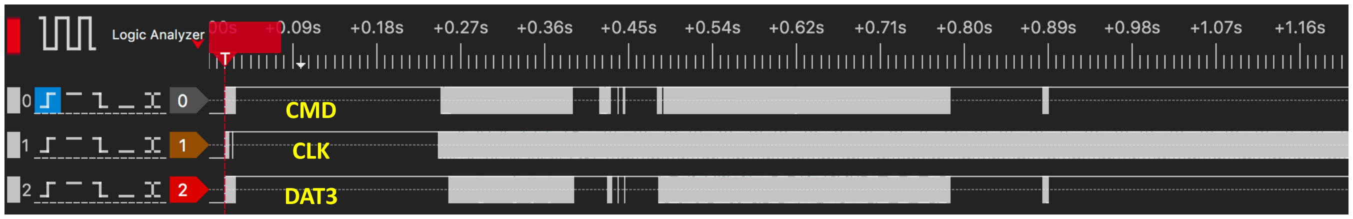

Figure 7: Data Transfer over SD interface covering the full boot-up of the Zynq device

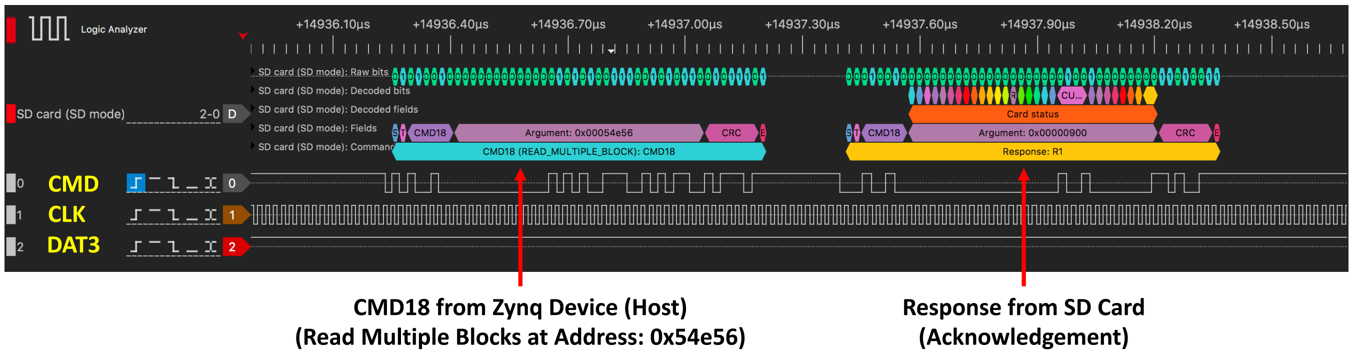
Logic Analyzer
SD card (SD mode)
SD card (SD mode): Raw bits
SD card (SD mode): Decoded bits
SD card (SD mode): Decoded fields
SD card (SD mode): Fields
SD card (SD mode): Command
CMD18
Argument: 0x00054e56
CRC
CMD18 (READ_MULTIPLE_BLOCK): CMD18
Card status
Argument: 0x00000900
Response: R1
CMD
CLK
DAT3
CMD18 from Zynq Device (Host)
(Read Multiple Blocks at Address: 0x54e56)
Response from SD Card
(Acknowledgement)

Figure 8: Visualization of packets transferred over the CMD line of the SD card interface

execution, we utilized a logic analyzer to analyze the communication between the SD card and the Zynq device, during FSBL execution. The reading/writing of data from/to the SD card occurs in blocks of 512 bytes, in a serial fashion, and in particular we monitored the commands CMD17 and CMD18, which can be used to read a single block and multiple blocks respectively.

We utilized the DSLogic Plus logic analyzer from DreamSourceLab to probe the SD card communication interface. Refer to Fig.6 for the picture of our experimental setup. We used to logic analyzer to probe the CMD, CLK and DAT3 lines (as a representative data line among DAT0-DAT3), and the captured signals can be viewed on the DSView software IDE. Please refer to Fig.7 for the data transfer over the SD interface during the entire boot-up phase of an authenticated and encrypted boot image. This captures the entire execution time of BootROM and FSBL. Channel 0 corresponds the CMD line, channel 1 corresponds to the CLK line and channel 3 corresponds to the DAT3 line. We only chose DAT3 as a reference for a data line, but any of the other data lines among DAT0-DAT2 can also be probed. Moroever, zooming into the data transfer allows us to visualize the packets within the DSView IDE, and refer to Fig.8 for the visualization of a command to read multiple blocks from the host (i.e.) CMD18 and the subsequent acknowledgement from the SD card.

A.2 Experimental Observations of BootROM Execution

We recall that the buggy software implementation of FSBL performs two PHT transfers (PHT1 and PHT2), when RSA authentication is enabled. While PHT1 only corresponds to retrieval of the PHT table, transfer of PHT2 corresponds to retrieval of the PHT along with the AC. If we identify such a redundant PHT transfer during BootROM execution, we can confirm that the vulnerability in the FSBL also exists in the BootROM. For our analysis, we considered three different types of boot images: (1) Non-secure (NSec), (2) Secure with only encryption (Sec_Encrypt) and (3) Secure with both encryption and authentication (Sec_Auth_Encrypt).

1. **Non-secure Image (NSec):** In this case, the BootROM is expected to only fetch the unencrypted FSBL. The size of the unencrypted FSBL in our boot image is ≈ 114.5 KB, which is equivalent to 225 blocks. Refer to Fig.9 for the data transfer corresponding to the retrieval of FSBL by the BootROM. We observe a total of 225 blocks being read from the SD card, using single block read commands (CMD17).

2. **Secure with only encryption (Sec_Encrypt):** When only encryption is enabled, the BootROM is expected to fetch the encrypted FSBL, whose

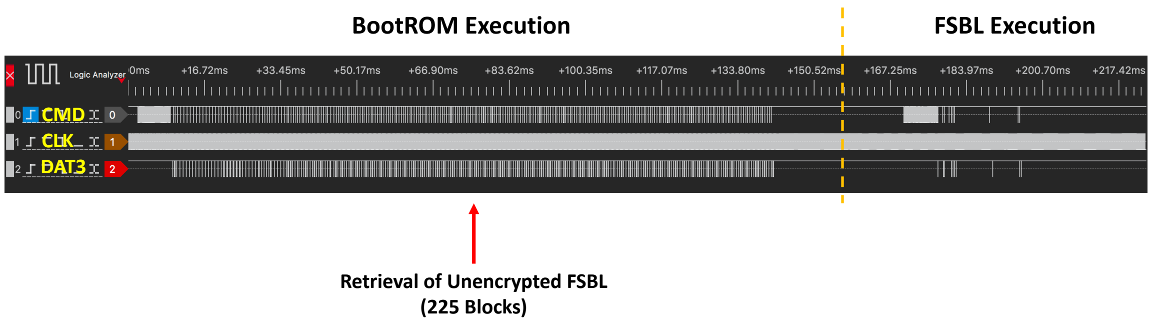

Figure 9: Retrieval of FSBL by BootROM in NSec image

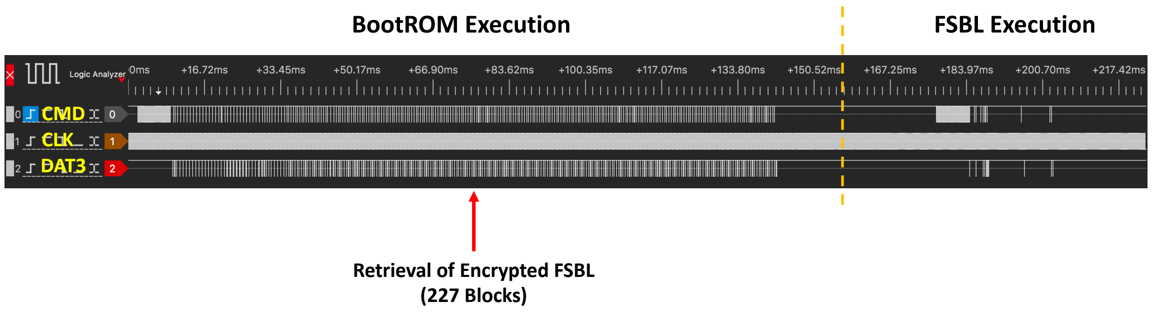

Figure 10: Retrieval of FSBL by BootROM in Sec_Encrypt image

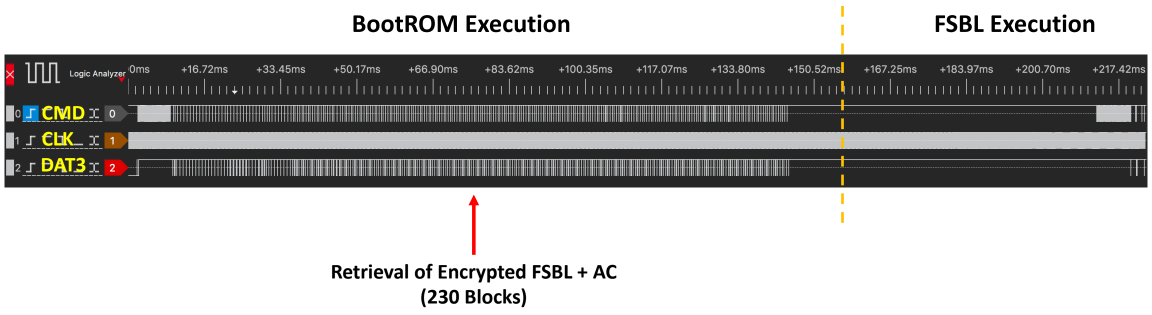

Figure 11: Retrieval of FSBL by BootROM in Sec_Auth_Encrypt image

size is roughly 115.5KB which is equivalent to 227 blocks. Refer to Fig.10 for the data transfer corresponding to the retrieval of the encrypted FSBL by the BootROM. We observe a total of 227 blocks being read from the SD card, using single block read commands (CMD17).

3. **Secure with both encryption and authentication (Sec_Auth_Encrypt):** When both authentication and encryption are enabled, the BootROM is expected to fetch the encrypted FSBL along with its Authentication Certificate (AC), whose size is roughly 116.8KB which is equivalent to 230 blocks. Refer to Fig.11 for the data transfer corresponding to the retrieval of the encrypted FSBL along with its AC by the BootROM. We observe a total of 230 blocks being read from the SD card, using single block read commands (CMD17).

If there were a duplicate transfer of the FSBL, similar to that of the PHT, then we should have observed roughly 455 blocks being transferred. However, the number of blocks transferred from the SD card tallies with the expected number of blocks to be read. From this observation, we can positively confirm that the flaw identified in the FSBL is not present in the BootROM software.

However, this does not rule out the possibility of other vulnerabilities within the BootROM, that could be exploited for RSA authentication bypass.