Breaking RSA Authentication on Zynq-7000 SoC and Beyond Identification of Critical Security Flaw in FSBL Software

Prasanna Ravi
PRASANNA.RAVI@ntu.edu.sg

Arpan Jati
arpan.jati@ntu.edu.sg

Shivam Bhasin
sbhasin@ntu.edu.sg

Temasek Labs@NTU, Singapore

December 13, 2023

1 Introduction

In this report, we perform an in-depth analysis of the RSA authentication feature used in the secure boot procedure of Xilinx Zynq-7000 SoC device. The First Stage Boot Loader (FSBL) is a critical piece of software executed during secure boot, which utilizes the RSA authentication feature to validate all the hardware and software partitions to be mounted on the device. We analyzed the implementation of FSBL (provided by Xilinx) for the Zynq-7000 SoC and identified a *critical security flaw*, whose exploitation makes it possible to load an unauthenticated application onto the Zynq device, thereby bypassing RSA authentication [5]. We also experimentally validated the presence of the vulnerability through a Proof of Concept (PoC) attack to successfully mount an unauthenticated software application on an RSA authenticated Zynq device. 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, an attacker can mount any unauthenticated software application on the target device to mount a variety of attacks. Among the several possible attacks, we are interested to perform recovery of the encrypted bitstream in the target boot image of the Zynq-7000 device. To the best of our knowledge, there does not exist any prior work that has reported a practical bitstream recovery attack on the Zynq-7000 device. In the context of bitstream recovery, Ender et al. [1] in 2020 proposed the *Starbleed* attack that is applicable to standalone Virtex-6 and 7-series Xilinx FPGAs. The design advisory provided by 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 that ensures configuration is authenticated prior to use" [2]. 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*. We perform experimental validation of our proposed attacks on the PYNQ-Z1 platform based on the Zynq-7000 SoC.

While Xilinx's design advisory claims that the Zynq-7000 SoC is "resistant" to the Starbleed attack [2], the critical security flaw identified in the FSBL software (*first contribution*) allows to bypass RSA authentication and subsequently use our novel approach (*second contribution*) allows to mount the Starbleed attack for full bitstream recovery on an RSA authenticated Zynq-7000 device.

We communicated our findings to Xilinx in a vulnerability disclosure on March 8, 2022 and started cooperating on the issue. Xilinx quickly confirmed the vulnerability on March 24, 2022 and also published a patch for the FSBL software on March 25, 2022 [4]. Information about the vulnerability was also published as a design advisory by Xilinx on April 28, 2022 [3].

1.1 Threat Model

We consider a victim Zynq-7000 SoC device securely booting using a secure boot image stored in a Non-Volatile Memory (NVM) accessible to an attacker. In our experiments with the PYNQ-Z1 platform, we store the victim boot image in the SD card. For the sake of explanation, we consider the case where the victim boot image has three partitions - FBSL, PL partition and PS partition. The target device mandates RSA authentication of the boot image (i.e.) the RSA eFUSE is enabled within the device and all the partitions are encrypted as well as authenticated. The hash of the RSA PPK is programmed into the corresponding eFUSE registers. The AES key used for the secure boot is also programmed into the eFUSE of the Zynq SoC. However, we note that the key source (eFUSE or BBRAM) does not affect the attack methodology in any way. Please refer Fig.1 for the structure of the victim boot image we consider for our attack.

The attacker's objective is to mount an unauthorized application (unauthenticated) onto the Zynq device. We assume that the attacker has access to the boot image as it is stored on the NVM (SD card). We also assume that the attacker neither has access to the RSA key nor the AES key.

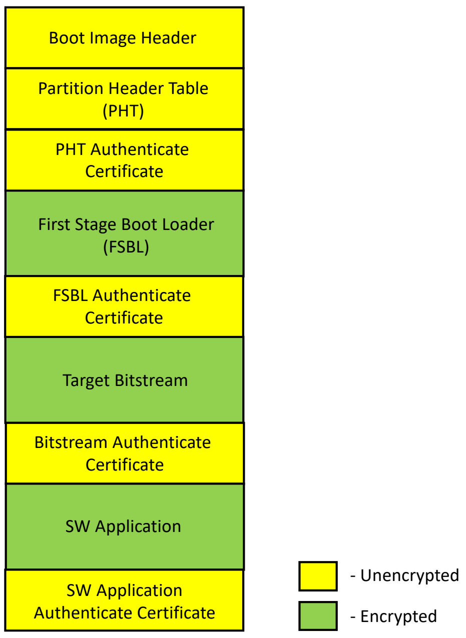
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
SW Application
SW Application Authenticate Certificate
- Unencrypted
- Encrypted

Figure 1: Authenticated victim boot image

2 Background

In this section, we will provide a brief background on secure boot as well as the RSA authentication feature of the Zynq-7000 SoC device, to establish a few necessary concepts which will facilitate the understanding of our attack, described later in Sections 3-6.

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 in a secure manner into the appropriate locations within the Zynq device (either DDR, On-Chip Memory (OCM) or FPGA). The important components of a boot image are as follows:

1. Boot Image header (BIH)
2. Partition Header Table (PHT)
3. First Stage Boot Loader (FSBL)
4. Programmable Logic (PL) partition/s (bitstream)
5. Programmable System (PS) partition/s (Standalone Application or Operating System)

We now explain the main components of the boot image, which are relevant to carry out our RSA bypass attack on the Zynq-7000 device.

2.1.1 Partition Header Table (PHT)

The partition harder table (PHT) is a critical component which contains information about the characteristics of the individual partitions of the boot image. Each partition is associated with an entry in the PHT, which contains metadata information about the corresponding partition. Some of the important fields in a PHT entry are encrypted partition size (at offset 0x00), decrypted partition size (0x04), total partition size including the RSA signature (0x08), address to be loaded and executed (0x0C), location of partition in the boot image (0x14) and attribute bits (0x18) which contain the encryption and authentication status of the partition.

The PHT is utilized by the FSBL to get information about each partition during the boot procedure. *It is important to note that the PHT is present unencrypted within the boot image. So, an attacker can gain information about the metadata of each partition as well as the entire boot image.*

2.1.2 First Stage Boot Loader (FSBL)

The FSBL is the first software executable to be run on the Zynq-7000 device, which loads some or all partitions within the boot image into the appropriate locations within the device. The first significant operation performed by the FSBL is to retrieve the PHT from the boot image and authenticate the PHT. If successful, then the FSBL starts loading the different PS and PL partitions based on information present in the PHT. Upon failure, the FSBL simply aborts the secure boot procedure.

The FSBL reads the PHT one entry at a time and for each authenticated partition, there is an Authentication Certificate (AC) appended right next to the partition in the boot image. The AC contains the partition signature which is verified by the FSBL. Upon successful verification, the FSBL decrypts the partition data (if required) and loads it in the appropriate location before moving onto the next partition. It is important to note that mandating RSA authentication through the eFUSE only requires the FSBL to be authenticated. All the other PS/PL executable partitions need not be authenticated in an RSA authenticated boot image. The authentication status of each partition is determined from the authentication status bit in the PHT entry.

After loading the required number of partitions, the FSBL can either transfer control to a Second Stage Boot Loader (SSBL) which further takes care of the loading process or handover control to a final application executable. While one can use a custom FSBL, we in this work utilize the official FSBL code for the Zynq-7000 SoC provided by Xilinx (FSBL version 2018.1).

2.2 RSA Authentication in Zynq-7000 SoC

The Zynq device uses the well-known RSA-2048 based signature scheme for authentication. It is carried out using two keys: Primary Key and Secondary Key. They form an authentication chain wherein the primary keys authenticate the secondary keys, and then the secondary keys authenticate the partition. The primary key consists of the Primary Public Key (PPK) and a Primary Secret Key (PSK) and the secondary key consists of the Secondary Public Key (SPK) and a Secondary Secret Key (SSK).

The primary key remains fixed for a given device and a SHA-256 hash value of the PPK is burnt in the eFUSE of the Zynq device. The BootROM upon reset, loads the PPK from the boot image and then calculates a SHA-256 signature of the PPK and matches it with the hash value stored in the eFuse. The boot only continues it the match is successful. However upon failure, the Zynq device goes into a secure lockdown. While the primary key is fixed for a given device, each partition can have its own secondary key, which of course needs to be signed by the primary key. Since the understanding of the intricate details about RSA authentication is not required for our attack, we refer the reader to [6] for more details.

The RSA authentication process is carried out by the BootROM code as well as the FSBL. While the BootROM code authenticates the FSBL, the FSBL authenticates the PHT along with all the listed partitions within the PHT. The BootROM code is present in the secure on-chip BootROM and is not accessible. However, the FSBL is open-source and is a modifiable piece of software. In the following section, we therefore analyze the FSBL source code to understand the implementation of the authentication procedure of all partitions by the FSBL.

3 Analyzing the RSA Authentication Procedure within FSBL

The FSBL starts by authenticating the PHT. If an attacker is able to bypass PHT authentication, he/she can mount a tampered PHT that can be used to execute an unauthenticated application on the Zynq device. Thus, we identify PHT authentication as a single point of failure to defeat RSA authentication. We analyzed the PHT authentication by FSBL, which is done in the following manner. This is implemented within the image_mover.c source file in the *embeddedsw* project which can be found in [5]. Refer Fig.2 for a pictorial illustration of the authentication procedure of the PHT by the FSBL.

1. The FSBL first retrieves the PHT (without its AC) from the NVM and stores it into a global variable denoted as GVAR. The GVAR variable is a *structure* that can house different partitions as well as the fields within each entry of the PHT. We denote the retreived PHT as PHT1. We note that this data transfer is performed between the NVM and the Zynq device since the PHT is present in the NVM within the boot image.

2. The FSBL then checks the status of the RSA eFUSE. If enabled, the FSBL again queries the NVM to retrieve the PHT along with its AC, which contains the PHT signature. The retrieved PHT signature is verified by the FSBL. We denote the PHT retrieved in the second data exchange as PHT2. Though the PHT retrieved in the first and second exchange are the same (i.e.) PHT1 = PHT2, they are retrieved at two different time instances and thus we refer to them as two different data sets.

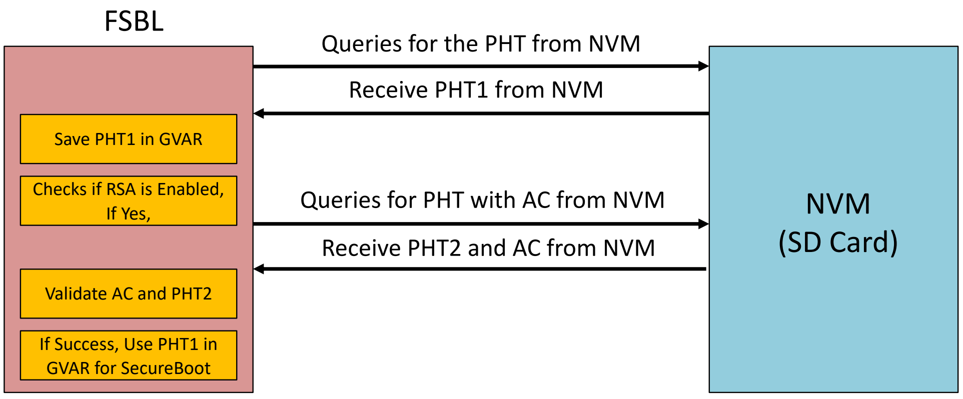
FSBL
Save PHT1 in GVAR
Checks if RSA is Enabled, If Yes,
Validate AC and PHT2
If Success, Use PHT1 in GVAR for SecureBoot
Queries for the PHT from NVM
Receive PHT1 from NVM
Queries for PHT with AC from NVM
Receive PHT2 and AC from NVM
NVM
(SD Card)

Figure 2: Authentication of the PHT by the FSBL

3. If the signature of PHT2 is successfully authenticated, the FSBL then uses the PHT data contained in the GVAR variable (corresponding to PHT1), to load the other PS/PL partitions in the boot image.

In other words, the FSBL authenticates PHT2, but uses the unauthenticated PHT1 within the GVAR variable for secure boot. Thus, we observe in the FSBL source code that, the unauthenticated PHT1 is used for secure boot. Both PHT1 and PHT2 are retrieved from the NVM (in our case, SD card) and can therefore be actively tampered by an attacker. In the following section, we demonstrate a simple and practical Proof of Concept (PoC) attack which tampers this repeated PHT data transfer to boot the Zynq device with a tampered PHT, which allows us to execute an unauthenticated application on the target device. While our observations were made on the FSBL version 2018.1, we confirm that our observations also apply to the current FSBL version dated 23 Apr 2020 available in the GitHub page[1].

4 Exploiting the RSA Security Flaw in FSBL

4.1 Attack Methodology

We formulate an attack methodology whose central component is an NVM emulator, for instance, an SD card emulator implemented using FPGA or a dedicated ASIC. We replace the SD card with an SD card emulator and thus the Zynq device communicates with an SD card emulator through the SD card interface. The NVM emulator is configured to behave like a typical NVM most of the time, providing the valid boot image contents. We consider NVM emulator as a black box component and the designer/adversary is free to choose how to implement it. The NVM emulator is configured to behave in the following manner during PHT authentication.

1. When the FSBL tries to retrieve PHT1, the NVM emulator provides a tampered PHT. This tampered PHT contains an entry corresponding to a unauthenticated PS partition. So, the tampered PHT1 is loaded into the GVAR variable, stored in the DDR of the Zynq device.

2. The FSBL then checks for the status of RSA eFUSE and if enabled, again retrieves the PHT along with its AC. The NVM emulator now provides the valid PHT (PHT2) and its signature, as present in the victim boot image.

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 queries the NVM for a unauthenticated application. The NVM emulator can simply hand over the unauthenticated application to the Zynq device, which will be successfully executed on the Zynq device after secure boot.

Refer Fig.3 for a pictorial illustration of our attack carried out using the NVM emulator. Thus, we can utilize an NVM emulator to externally tamper the PHT data and successfully execute an unauthenticated application on an RSA authenticated Zynq-7000 device.

4.2 Proof of Concept (PoC) Implementation

We validated our attack methodology using a Proof of Concept (PoC) attack to demonstrate bypass of RSA authentication on the Zynq-7000 device. We did not implement an SD card emulator for our PoC, however made manual modifications to the FSBL, to replicate the behaviour of an SD card emulator. We manually modify

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

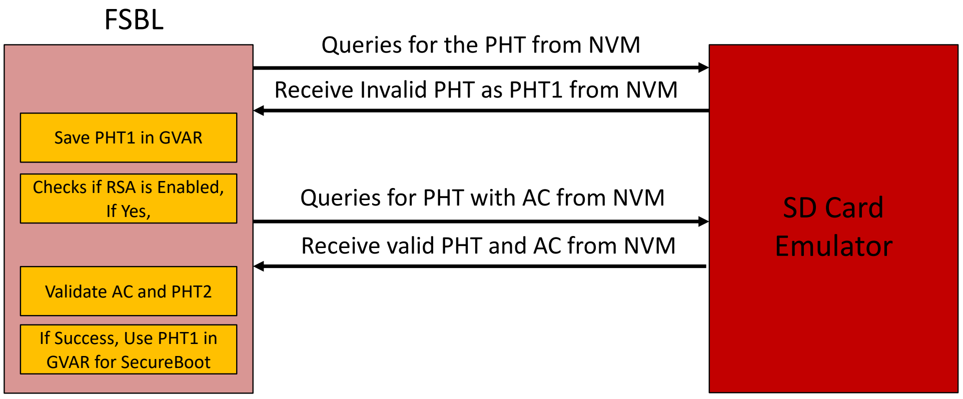
FSBL
Save PHT1 in GVAR
Checks if RSA is Enabled, If Yes,
Validate AC and PHT2
If Success, Use PHT1 in GVAR for SecureBoot
Queries for the PHT from NVM
Receive Invalid PHT as PHT1 from NVM
Queries for PHT with AC from NVM
Receive valid PHT and AC from NVM
SD Card Emulator

Figure 3: Attack of the PHT authentication procedure using an NVM emulator

the PHT data in the global variable GVAR which is equivalent to sending a tampered PHT using an SD card emulator. The tampered PHT contains a PHT entry corresponding to an unauthenticated and unencrypted PS partition (malicious attack application) on the Zynq device. However, the valid PHT in the victim boot image corresponds to all authenticated partitions. The unauthenticated PS partition requires to be unencrypted as well, since the attacker does not have access to the AES key. We reiterate that that the modifications done to the FSBL only tamper the first PHT retrieval to replicate the behaviour of an SD card emulator. Thus, it does not aid our attack is any other manner. In the presence of an SD card emulator, this modification of the FSBL would not be required.

We create a malicious software application and replace it with the authenticated application in the boot image. We denote the modified boot image as our attack boot image. So, the attack boot image is essentially the victim boot image with a modified application partition. Please refer Fig.4 for the attack boot image. It is important to note that the FSBL in the attack boot image is same as one in the victim boot image.

4.2.1 Experimental Observations on PoC Implementation

Upon reset, the FSBL is executed and loads the valid PHT in the global variable GVAR from the SD card. However, the modification in FSBL ensures that the PHT data in GVAR is corrupted. Upon checking the status of the RSA eFUSE, the FSBL again retrieves the valid PHT and its AC from the attack boot image, which is successfully authenticated. Subsequently, the tampered PHT present within the GVAR variable is used for secure boot. We were then able to successfully execute our malicious unauthenticated attack application on the Zynq device.

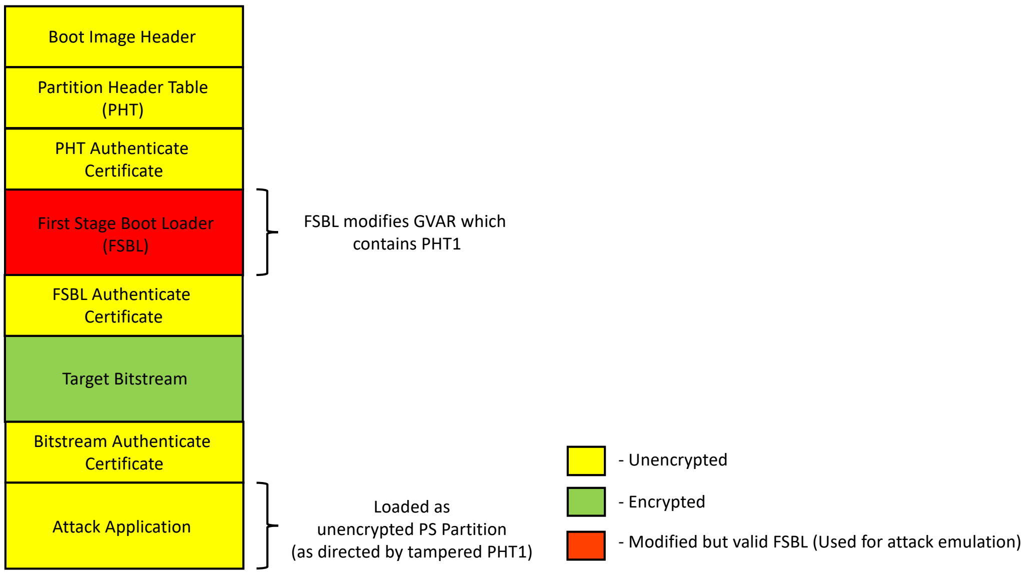
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
Attack Application
FSBL modifies GVAR which contains PHT1
Loaded as unencrypted PS Partition (as directed by tampered PHT1)
- Unencrypted
- Encrypted
- Modified but valid FSBL (Used for attack emulation)

Figure 4: Attack boot image (with modified FSBL) used to replicate the behaviour of the NVM emulator

Though the attack exposes the flaw in the PHT authentication by the FSBL, it does not qualify as a real attack for the following reasons: (1) The PHT data is manually tampered within the FSBL (2) Any modification to the FSBL is not possible, since the FSBL is encrypted using an unknown key and signed within the target boot image. So, a real attack should be ideally performed without any modifications to the FSBL. In the following, our objective is to mount a practical attack to bypass RSA, which does not require any modifications to the FSBL software. In the following section, we will present an almost practical instantiation of our attack using our custom SD card switcher board.

4.3 Improved Attack using SD-Card Switcher Board

Upon analysing the flow of the PoC attack, we observed that the PHT data is sent twice to the Zynq device from the NVM. To realize the attack, we want to tamper only the first PHT transfer (PHT1), while the second PHT data transfer (PHT2) and all the other data transfers from then on, need not be corrupted. One trivial technique to achieve this would be to simply implement the entire SD card on an FPGA/ASIC, and designing the required logic to tamper the first PHT data transfer. However, implementing the entire SD card on the FPGA/ASIC is an arduous task that would require a significant engineering effort, considering the complex intricacies of the SD card protocol.

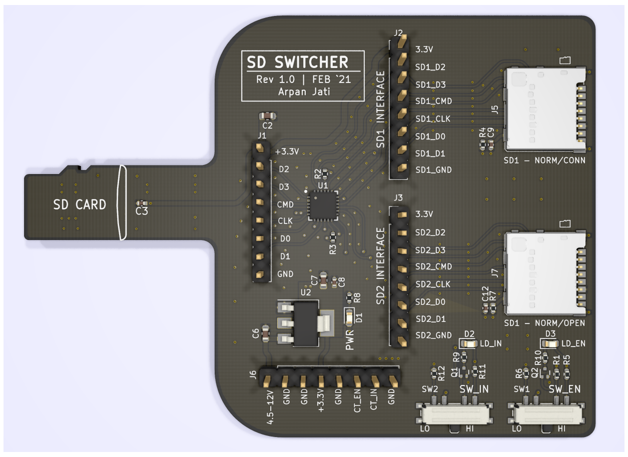

Figure 5: SD Card Switcher Board

We however attempted to utilize a much simpler approach, which relies on existing hardware to implement the SD card emulator. The basic idea is to build a *multiplexer* which can switch between multiple SD cards to communicate with the Zynq device. If it can be realized, then we can utilize say SD Card 1, to sent the tampered PHT as PHT1, while another SD Card 2, can be used to send the correct PHT as PHT2 in the second PHT transfer. However, this should be done in a manner that is oblivious to the target Zynq device, as the target assumes that it only communicates with a single SD card.

Refer Fig.5 for the designed SD card switcher board. It has slots for two SD cards, and there are manual switches (at the bottom of the board) to select the SD card to be connected to the target. The board however also allows to choose the SD card based on the logic level of a GPIO pin. The switching action is performed by simply switching the 4 DATA lines (DAT0-DAT3) and the CMD lines of the SD card interface from one SD card to another. The other lines such as the CLK, VDD and GND lines are common for the two SD cards. The board also facilitates to keep 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.

4.3.1 Attack Methodology

To carry out our attack, we utilize two SD cards. The first SD card contains a boot image with a tampered PHT, while the second SD card contains a boot image with a valid PHT. Please refer to Fig.6 for the boot images present in the two SD cards denoted as SD card 1 and SD card 2. The SD card 1 image, contains the tampered PHT, corresponding to an unauthenticated attack application. The unauthenticated attack application is present as the PS partition, after the bitstream in the boot image. The SD card 2 image, contains a valid PHT, similar to the one present in the victim boot image (Fig.1). All the other partitions are same as the boot image of SD card 1. We load both the SD cards onto the SD card switcher board, and connect the SD card switcher to the Zynq device. The switcher is also powered externally using an external power supply. This is done to ensure power to the SD cards even when the target is switched off. This keeps the SD cards in a good state, once initialized. Please refer to Fig.7 for the attack setup. The attack is carried out in the following manner:

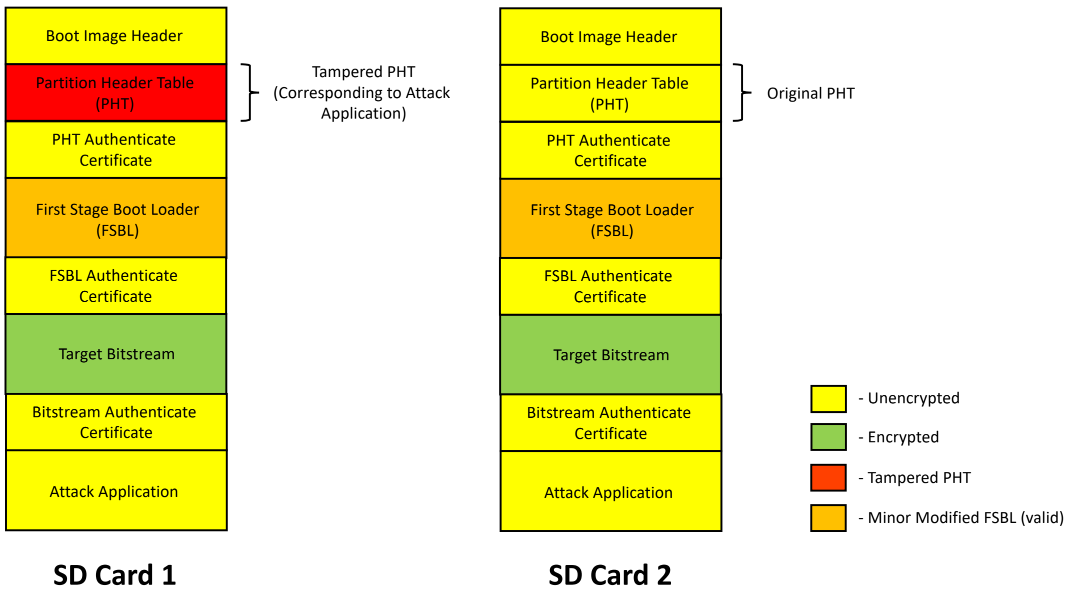
Boot Image Header
Partition Header Table (PHT)
PHT Authenticate Certificate
First Stage Boot Loader (FSBL)
FSBL Authenticate Certificate
Target Bitstream
Bitstream Authenticate Certificate
Attack Application
Tampered PHT (Corresponding to Attack Application)
SD Card 1
Boot Image Header
Partition Header 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 6: Boot Images of SD card 1 and SD card 2 within the SD switcher board

1. The switcher initially connects SD card 2 to the target Zynq device. The Zynq device is booted up using the SD Card 2 image. This is done to initialize SD card 2.

2. Once SD card 2 is initialized, we now power off the target Zynq device and switch to SD card 1. This is done while maintaining power to both the SD cards in the switcher.

3. Now, we boot the Zynq device with the image in SD card 1. Once the FSBL retrieves the tampered PHT (PHT1) from SD card 1, we turn off the Zynq device and use the manual switch on the board to switch from SD card 1 to SD card 2. For our demonstration, we add a suitable delay in the FSBL to enable the manual switch. However, this can be automated as the timing of switch is constant for the Zynq device upon power up.

4. The FSBL now checks the RSA eFuse value and if enabled, retrieves the PHT again (PHT2) along with its signature. But, since the Zynq device is connected to SD card 2 with a valid PHT, the PHT should be authenticated successfully by the FSBL. The flaw in the FSBL software ensures that the tampered PHT1 is used for secure boot. This should ensure that the attack application is mounted in an unauthenticated manner on the Zynq device.

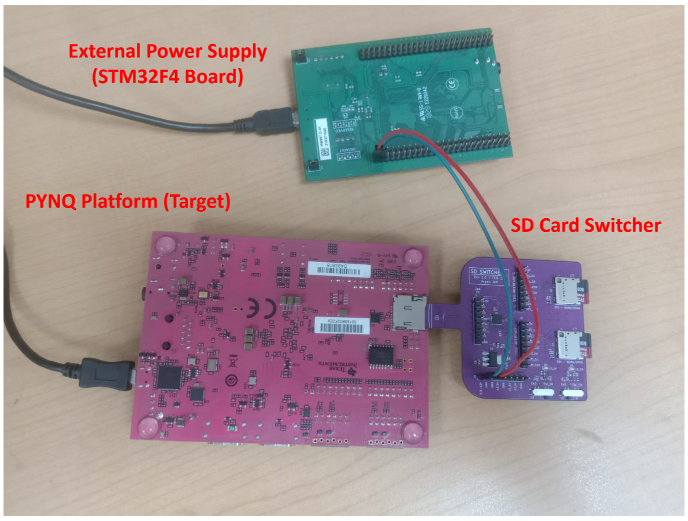

Figure 7: Attack Setup with SD Card Switcher

4.3.2 Experimental Observations of Attack using SD Card Switcher Board

Upon performing the practical experiments, we observe that the Zynq device halts operation after the switch from SD card 1 to SD card 2 in Step 3, during the operation of the FSBL. We use our knowledge of the FSBL operation to hypothesize reasons for the observed behaviour. The FSBL typically initializes the SD card using a certain InitSD function. At the end of Step-2, the SD card 2 is initialized and powered up. In the following Step 3, the target device now boots from SD card 1, and the FSBL thus initializes SD card 1. After retrieval of PHT1 from SD card 1, we switch to SD card 2, but the FSBL is unable to retrieve data from SD card 2, though it is intialized.

We hypothesize that the SD card controller on Zynq, which is oblivious to the switch, tries to communicate with commands pertaining to SD card 1, while SD card 2 is connected. Every connected SD card has a Unique Identifier provided by the Zynq device, and we hypothesize that the Zynq device uses the UID of SD card 1 to communicate with SD card 2, which fails. We can overcome this behaviour by simply modifying the memory read command from the controller using an external hardware, realized using FPGA/ASIC. But, for our attack demonstration, we modify the FSBL to add an additional call to the InitSD function, to initialize SD card 2 after the switch (i.e.) before the second read of PHT with its signature from the NVM. We however note that such modification to the signed FSBL is not possible in a real attack scneario. Nevertheless, we present results of our attack with this minor modification in the FSBL.

Using the modified FSBL with just a single additional InitSD function, we are able to successfully demonstrate bypass of PHT authentication. This enabled us to mount an unauthenticated application on the target Zynq device. While our PoC attack (presented in SecNext-D4) involved manual tampering of the PHT data within the FSBL, our improved attack using the SD card switcher board enables manipulation of PHT data, fully external to the Zynq device. Please refer to Fig.8 for a pictorial illustration of our improved attack using the SD card switcher. The normal operations carried out by the FSBL are colored in orange, while the additional call of the InitSD function in the FSBL (added by us) is colored in red.

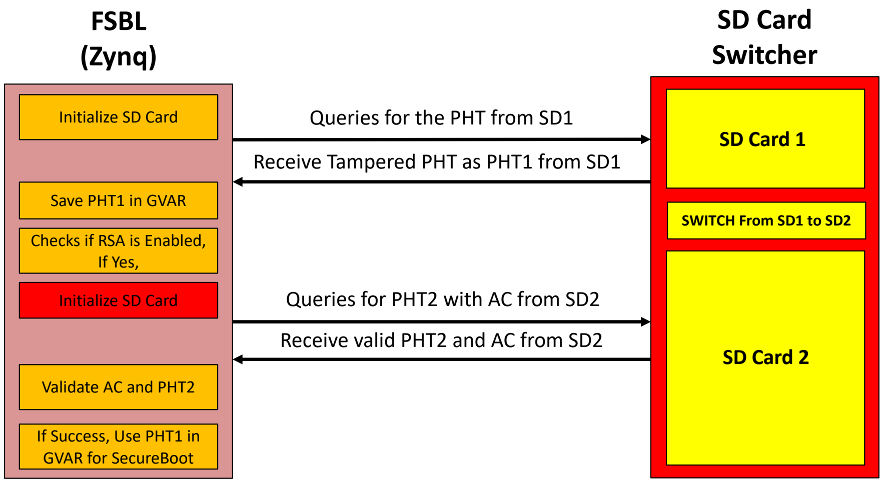
FSBL (Zynq)
SD Card Switcher
Initialize SD Card
Save PHT1 in GVAR
Checks if RSA is Enabled, If Yes,
Initialize SD Card
Validate AC and PHT2
If Success, Use PHT1 in GVAR for SecureBoot
Queries for the PHT from SD1
Receive Tampered PHT as PHT1 from SD1
Queries for PHT2 with AC from SD2
Receive valid PHT2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

Figure 8: Improved Attack on the Zynq device using SD card switcher. The operations of the FSBL are indicated in orange, while the modifications done to the FSBL to facilitate the attack are denoted in red.

Our current attack setup still cannot be used for a practical attack, and demonstration of a practical attack would require a specialized hardware to initialize the SD card 2 after the switch. Our improved attack however concretely exposes the flaw in the FSBL software, since manipulated data from an external NVM can successfully bypass the RSA authentication procedure. We therefore concretely believe that this limitation can be easily overcome using a additional specialized hardware (FPGA/ASIC) capable to tampering the data transfer over the SD card interface at precise time instances.

4.4 Fixing the Flaw in PHT Authentication within FSBL

The vulnerability mainly arises due to retrieval of the same PHT data twice from the NVM and subsequently utilizing the unauthenticated PHT data. Thus, the flaw can be easily fixed by changing the FSBL implementation such that the PHT is only retrieved once from the NVM and that the authenticated PHT data is used for the secure boot. This fix is in fact implemented as part of the patched FSBL (dated March 25th, 2022) [4], and our manual analysis of the patched FSBL source code reveals that the redundant fetch of the PHT has been removed, and thus the aforementioned attack is not possible to be mounted.

Upon bypass of RSA authentication, an attacker can mount a malicious software or hardware application to mount a variety of attacks. We analyze the possibility of performing bitstream recovery on such a compromized Zynq-7000 device. The attacker's main intention is to decrypt the encrypted bitstream stored in the target victim boot image. In the following section, we also demonstrate a practical bitstream recovery attack using a malicious and unauthenticated software application on the Zynq-7000 device.

5 Starbleed Attack for Bitstream Recovery

Ender et al. [1] in 2020 proposed the *Starbleed* attack that is applicable to standalone Virtex-6 and 7-series Xilinx FPGAs. The attack does not require use of side-channels and merely exploits a design flaw in the Xilinx FPGAs for bitstream recovery. The only requirement is that the attacker requires access to the configuration interface of the FPGA (PL). Since this requirement is satisfied in our scenario, due to the presence of the internal PCAP interface, we attempt to adapt the Starbleed attack to the Zynq-7000 device for bitstream recovery.

5.1 Attack Methodology

The adversary makes malicious changes to the encrypted bitstream (as proposed in the Starbleed attack), such that upon decryption, a targeted decrypted bitstream word is written into the WBSTAR register. The WBSTAR register retains its value even upon FPGA reset and thus an adversary can access the decrypted word from the WBSTAR register. In a similar manner, full bitstream recovery can be performed one word at a time. We therefore utilized the internally available PCAP (Processor Configuration Access Port) interface accessible to the PS to feed the AES/HMAC module with the malicious tampered bitstreams.

We observed that our attack application was able to push the tampered bitstreams through the PCAP interface in a straightforward manner and trigger an HMAC error. However, the reference manual of the Zynq-7000 SoC claims that readback of the WBSTAR register through the PCAP interface is not possible unless the PL is configured with a valid bitstream (i.e.) configuration DONE signal is high. Thus, readback of the WBSTAR register containing the decrypted bitstream word was not trivially possible. However, we overcame the limitation using the following technique.

5.1.1 PL readback through PCAP Interface:

As stated earlier, the main limitation with respect to performing PL readback through PCAP is to ensure that the configuration DONE signal stays high. We identified a hack to ensure that the configuration DONE signal can be artificially maintained high in the following manner:

1. Program PL with a valid encrypted bitstream (target bitstream) which raises DONE signal to high.
2. Without initializing the PL, we push the tampered Starbleed attack bitstream through the PCAP interface. We observe that the DONE signal still stays high even though HMAC error is triggered due to the tampered bitstream.
3. We provide commands through the PCAP interface to readback the WBSTAR register. We were able to successfully readback the decrypted bitstream word from the WBSTAR register.

This technique of pushing bitstreams through the PCAP interface without initializing the PL (step 2) is not recommended practice. Moreover, we expect that the FPGA does not accept bitstreams for configuration unless the FPGA is properly initialized (i.e.) pulsing the PROG_B pin and waiting for INIT_B signal to indicate that PL is ready for programming. But, we observed that the handcrafted bitstreams were still decrypted by the AES/HMAC unit in the PL ensuring that the decrypted bitstream word is present in the WBSTAR register.

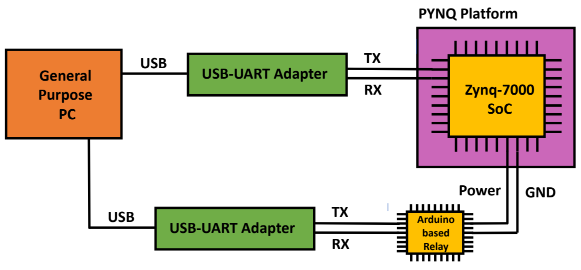

Figure 9: Attack Setup for Automation of Starbleed Attack through PCAP readback on Zynq-7000 SoC

After readback of the bitstream word from the WBSTAR register, the PCAP interface becomes unresponsive and only a Power On Reset (POR) could revive the PCAP interface to normal working condition. In particular, it is the final PCAP readback operation to retrieve the WBSTAR register, that pushes the PCAP into an unresponsive state. Hence, in a single boot, it is only possible to recover a single bitstream word with our current attack. Thus, we need to do a power cycle of the Zynq device to recover every bitstream word when only using the PCAP interface for the attack.

Refer Fig.9 for our attack setup to perform the Starbleed attack on the Zynq-7000 SoC which relies on PL readback using the PCAP interface. We use an arduino controlled relay shield to do an automatic POR reset of the Zynq device. The UART interface on the Zynq device is used to receive the handcrafted Starbleed bitstreams from the PC, as well as communicate the decrypted bitstream word from the Zynq device to the PC.

With our current setup, we are able to perform full bitstream recovery at the speed of 32-bits per second. Thus, recovery of a sample bitstream of size 3.85 MB takes roughly about 46 days. Performing repeated POR reset and going through the secure boot for recovery of every bitstream word serves as the main bottleneck with respect to speeding up bitstream recovery. We see that it is trivially possible to automate it and reduce the overall time with some off the shelf low-cost hardware. However, performance acceleration is out of scope.

5.1.2 PL readback through JTAG Interface:

While PCAP readback is possible, we observed that PL readback is also possible through the JTAG interface. We were able to successfully validate recovery of the decrypted bitstream through a JTAG readback. Unlike PCAP readback, we observe that the PCAP interface remains in a working condition even after reading the WBSTAR register through the JTAG interface. Thus, a POR reset is not required after recovery of every bitstream word. This can significantly improve the speed of bitstream recovery compared to our earlier attack which only relies on the internal PCAP interface. Thus, much faster bitstream recovery is possible (potentially in a matter of few hours) when the JTAG interface is exposed on the target device.

However, there are scenarios where JTAG interface might not be exposed to an adversary. This is possible when JTAG is intentionally disabled by blowing the appropriate eFUSE bits in the PS or PL. As long as the JTAG is not permanently disabled, it is possible for an adversary to rely on JTAG readback for full bitstream recovery.

5.2 Countermeasure against Bitstream Recovery

We were able to carry out the Starbleed attack using a malicious and unauthorized software partition, mainly through exploitation of the RSA authentication flaw in the FSBL. Thus, using the patched version of the FSBL (dated March 25, 2022) [4], that is devoid of this vulnerability can serve as a strong mitigation against the bitstream recovery attack. Evaluating whether Starbleed could be applied successfully to other interfaces is an area of future research but is out of scope for our work.

6 Conclusion

In this work, we have identified a *critical security flaw* in the FSBL software whose exploitation makes it possible to bypass RSA authentication and load an unauthenticated and unencrypted application onto the Zynq device. The vulnerability is present in the FSBL software, and mainly arises due to mishandling of the PHT data. We also experimentally validated the presence of the vulnerability through a Proof of Concept (PoC) attack to successfully mount an unauthenticated software application on an authenticated Zynq-7000 device. The identified flaw is only present in the FSBL software and thus can be easily fixed. We also demonstrated the first successful bitstream recovery attack on the Zynq-7000 SoC by mounting the Starbleed attack using a malicious software application.

References

[1] 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 ({USENIX} Security 20)*, 2020.

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

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

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

[5] 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.

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