Silicon Heist: (Ransom) Attacks for AMD Cloud FPGAs via Privilege Escalation

Simon Klix[1], Felix Hahn[1], Maik Ender[1], Nils Albartus[1], Christof Paar[1], Russell Tessier[2]

[1]*Max Planck Institute for Security and Privacy (MPI-SP)*
[2]*University of Massachusetts*

Abstract

Cloud-based FPGAs have become a billion-dollar industry, allowing users to deploy custom hardware designs with the scalability and flexibility of cloud infrastructure. Running user designs on hardware owned by the cloud service provider (CSP) introduces risks, including intentional hardware damage and Denial-of-Service (DoS) attacks against the host. To mitigate these risks, CSPs enforce security mechanisms that restrict user designs and prevent unauthorized behavior. We present a novel privilege escalation path on AMD FPGAs using (i) the Internal Configuration Access Port (ICAP) to circumvent provider defenses, (ii) incrementally escalate attacker capabilities to remote JTAG access, and (iii) investigate the resulting threat vectors.

Any typical cloud customer can maliciously acquire such ICAP access to reconfigure parts of the FPGA fabric without restrictions – re-enabling traditional cloud FPGA attacks. Through the ICAP, a user can ultimately gain remote control of the hardware's low-level JTAG interface, which enables access to the device's eFuses. This access, in turn, allows attackers to irreversibly program encryption settings, thereby disabling future reconfiguration and locking the CSPs out of their own devices. An attacker could leverage such escalated privileges for a ransomware attack in which cloud providers must pay a ransom for decryption keys to regain control of their devices – effectively introducing the first ransomware for FPGAs. Following the investigation of this novel privilege escalation path, we demonstrate its feasibility on Amazon's EC2 F1 and F2 instances and explore the impact of enabled attack vectors. We thereby expose the neglected threat of unsecured low-level hardware components in cloud environments.

1 Introduction

Following the path of traditional digital hardware, such as CPUs and storage, Field Programmable Gate Arrays (FPGAs) entered the cloud computing market and grew into a billion-dollar industry, with continued growth expected [18]. FPGAs are reconfigurable hardware devices that can be (re)programmed with custom designs to execute specific computations efficiently. They are widely used in fields such as artificial intelligence, medical imaging, signal processing, defense, and aerospace [12, 13, 23, 35]. In general, FPGAs are ideal for applications with low production volumes that make the development of an Application Specific Integrated Circuit (ASIC) cost-prohibitive. Cloud-based FPGAs enable users to deploy custom designs remotely on rented hardware, benefiting from the cloud's scalability, flexibility, and low entry costs, especially since high-performance FPGAs can be prohibitively expensive to purchase outright.

Granting direct hardware access in cloud environments introduces risks to Cloud Service Providers (CSPs) such as Denial-of-Service (DoS) attacks, intentional hardware damage, and harm to other customers in hypothetical multi-tenant scenarios. Hence CSPs, like Amazon and Microsoft, adopted security measures to safeguard legitimate users and their cloud infrastructure [24, 28, 33], building upon the foundational principles proposed by Trimberger et al. [40]. To this end, the CSPs introduce an abstraction layer that controls direct hardware access, ensuring user designs are appropriately constrained. For example, users are restricted from accessing privileged modes and resources, raw network traffic, or other tenants' data. A verification pipeline enforces security and functional compliance by analyzing each customer's design for potential violations and adherence to Design Rule Checks (DRCs) before deployment.

Despite these safeguards, it remains challenging to eliminate all risks associated with user design execution on provider-controlled hardware. Researchers have repeatedly demonstrated vulnerabilities bypassing such security measures. For instance, previous work has identified power-hammering attacks bypassing Amazon Web Services (AWS) security measures, leading to DoS attacks or even permanent hardware damage [15, 24, 25, 36]. Additionally, Albartus et al. [1] have shown the capabilities and risks associated with Internal Configuration Access Port (ICAP) control in AMD FPGAs. This ICAP interface provides unrestricted ac-

cess to the FPGA configuration engine from within a running design and enables runtime reconfiguration. In this work, we uncover a novel privilege-escalation path on a cloud FPGA, which lets a regular customer gain unauthorized access to (i) the ICAP and (ii) the Joint Test Action Group (JTAG) interfaces. To this end, (i) we first seize control over an ICAP, allowing us to arbitrarily reconfigure parts of the FPGA and bypass security and safety checks employed by the CSP to suppress malicious designs. Second, (ii) we further escalate our privileges to gain internal JTAG access, unlocking capabilities such as the ability to read from and write to the FPGA's eFuse array. Attackers could exploit this access to mandate FPGA configuration with encrypted and authenticated bitstreams, using attacker-controlled AES and RSA keys. Such a permanent, malicious configuration could lead to a ransomware attack where cloud providers must pay a ransom for the keys to regain control of their FPGAs, potentially resulting in millions of dollars in damages.

Based on the identified threats, we conduct real-world experiments on Amazon's EC2 F1 and F2 FPGA cloud instances to assess the feasibility of our privilege escalation path. We begin by outlining the challenges and security mechanisms encountered while attempting to access the ICAP and evaluate whether control over the configuration engine can be leveraged to bypass CSP defenses. Contrary to AMD's documentation, which states that an ICAP must remain in a static design region [42], we demonstrate that using the ICAP within a dynamically configured customer design is possible. Furthermore, we found a lack of effective provider security measures in their verification pipeline that would prevent the instantiation of an ICAP interface. Our findings confirm that typical users can escalate their privileges up to the JTAG port and subsequently access eFuses, demonstrating the realistic threat potential of our proposed ransomware scenario. We thereby expose the neglected threat of unsecured low-level hardware components in cloud environments.

Contributions

- We present a novel privilege escalation path in cloud FPGAs (Section 3), along with a discussion of the technical challenges that must be overcome to enable exploitation (Section 4).
- As evaluation we conduct multiple case studies on Amazon's cloud FPGA instances, demonstrating: (i) the practical implementation of our privilege escalation path using permissions of a regular cloud user; (ii) the feasibility of existing cloud attacks enabled by illegitimate configuration engine control; and (iii) the viability of a ransomware attack through JTAG access. (Section 5).
- We propose possible mitigations on multiple levels to prevent these attack vectors and harden the cloud environment against future threats. Furthermore, we discuss the security shortcomings facilitating this privilege escalation path. To raise awareness among FPGA owners of the risks associated with ICAP and JTAG access, we explore other scenarios where configuration capabilities and debugging interfaces could be exploited (Section 6).

2 Technical Background

This section provides the technical background for our work. We outline the fundamental primitives and architecture of the target cloud FPGAs. In the following, we adopt the nomenclature of AMD UltraScale(+) FPGAs, as these devices constitute the primary target of our case studies.

2.1 FPGA Fundamentals

An FPGA derives its flexibility and reconfigurability from an array of configurable elements, including Look-Up Tables (LUTs) for implementing Boolean logic, Flip Flops (FFs), and more complex blocks such as digital signal processing (DSP) blocks and block-RAMs (BRAMs), all interconnected by programmable routing. Together, these components form the reconfigurable FPGA *fabric*, capable of implementing almost any digital hardware design.

2.1.1 Development for FPGAs

To create a hardware design for an FPGA, a user typically begins with a design description, such as register transfer level (RTL) code written in a Hardware Description Language (HDL). This source code is processed through the FPGA vendor's toolchain in multiple stages. The flow begins with synthesis and technology mapping, where the behavioral description is translated into a netlist composed of elements available on the target FPGA. The resulting netlist then undergoes implementation, during which logic blocks and routing resources are mapped onto specific physical locations on the chip. AMD's tools support the generation of Design Checkpoints (DCPs) at various points in the flow. These checkpoints preserve the current state of the design, supporting analysis, reuse, and incremental synthesis. In the final stage, the implemented design is converted into a bitstream, a proprietary file that contains all the information needed to configure all elements on the FPGA with the finalized design.

2.1.2 Configuration Process

An FPGA is configured by loading a bitstream into its fabric, thereby programming its reconfigurable elements, i.e., LUTs, FFs, and routing. The bitstream is transferred to the FPGA via a dedicated configuration interface, such as SelectMAP, ICAP, or JTAG. Configuration data is organized into discrete units called *frames*, representing the smallest addressable segment

of the fabric. The size of a frame varies across FPGA families; for AMD's UltraScale(+) devices, each frame consists of 93 words, each 32 bits in length [43]. In addition to raw configuration data, the bitstream contains control commands that orchestrate the FPGA's internal configuration engine.

2.1.3 Partial Reconfiguration

AMD UltraScale(+) FPGAs offer partial reconfiguration capabilities, allowing users to dynamically modify specific regions of the design while the device continues to operate. This feature allows for dynamic function updates, resource sharing, and power optimization without requiring a full system reset. Partial reconfiguration is performed by loading a specialized bitstream containing configuration commands instructing the FPGA's configuration engine to update selected components. At the hardware level, reconfiguration is achieved by overwriting specific frames in the fabric, thereby modifying the configuration of basic elements and routing.

Partial reconfiguration can be performed through the same configuration ports as the initial device configuration. The ICAP assumes a special role because it allows access to the configuration engine from within the design, i.e., the FPGA can change its own functionality. On AMD UltraScale(+) device access to the ICAP is facilitated by an `ICAPE3` primitive that is physically implemented at specific locations on the FPGA. As with any other element in the fabric, the `ICAPE3` must be deliberately instantiated in the netlist and is subsequently enabled by the bitstream. However, according to documentation, the `ICAPE3` can only be instantiated in the *static* part of a design [42]. Specifically, this part can supposedly not be changed through partial reconfiguration.

2.2 FPGA Security

FPGAs are often employed in mission-critical operations or used to process highly sensitive data. In such cases, the FPGA design may contain valuable intellectual property (e.g., algorithms) or secret data (e.g., cryptographic keys). Consequently, bitstream protection is a crucial element in the security of an FPGA system. Many FPGA vendors provide bitstream encryption and authentication mechanisms to ensure the confidentiality and authenticity of the running design. In the AMD UltraScale(+) series, AES is used for bitstream encryption, while RSA handles bitstream authentication.

2.2.1 eFuses

Critical security settings on AMD UltraScale(+) devices are enforced through eFuses, which are non-volatile, one-time-programmable (from 0 to 1) memory cells organized into registers. These eFuses are essential for securing the FPGA, as they store cryptographic keys and define security policies. Additionally, each FPGA has a unique, factory-programmed 96-bit device identifier, called Device DNA, stored in eFuses.

Among the most critical security features controlled by the `FUSE_SEC` eFuse register is the enforcement of AES-encrypted and/or RSA-authenticated bitstreams [43]. In this configuration, the FPGA will only start booting if the bitstream is encrypted and/or authenticated. For maximum security, both encryption and authentication must be enforced simultaneously, mitigating vulnerabilities associated with each method individually [9–11].

Additional security settings enabled by eFuses include disabling external JTAG access, restricting AMD test access, and permanently disabling the AES decryptor [43]. These measures strengthen security by preventing unauthorized modifications, debugging, or decryption attempts. To safeguard against accidental or malicious modifications, the `FUSE_CNTL` register includes control bits that can be programmed to disable readback or further modification of eFuse settings. However, since eFuses are one-time programmable, careful planning and validation are essential because misconfigurations can result in irreversible consequences. For instance, disabling the AES decryptor while enforcing encrypted bitstreams would render the FPGA incapable of processing any valid configuration, effectively making it permanently unusable.

The eFuses can exclusively be programmed or read back via JTAG. Beyond an external JTAG port, there is also an internal `MASTER_JTAG` element enabling JTAG access from within the design, allowing the user to change security settings at runtime. According to documentation, the `MASTER_JTAG`, like the `ICAPE3`, must be instantiated in the static region of the netlist and cannot be programmed via partial reconfiguration [42].

2.3 SSI Devices

In our case studies, we target Virtex UltraScale+ FPGAs, which incorporate AMD's SSI technology [22]. This technology allows up to four FPGA chiplets to be combined into a single virtual device, interconnected via a high-speed silicon interposer. From a user perspective, this aggregation of multiple FPGAs into a single device is mostly invisible. The entire device appears as a cohesive unit divided into multiple Super Logic Regions (SLRs), each corresponding to one of the integrated chiplets, as illustrated in Figure 1. AMD's toolchain

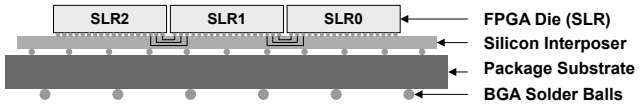

Figure 1: Example of AMD's SSI device layout consisting of three individual FPGA dies connected via a silicon interposer resulting in one FPGA with three Super Logic Regions (SLRs).

automatically handles routing and communication between these SLRs, ensuring seamless integration and usability.

Despite the unified abstraction, each SLR retains independent hardware components, including its own configuration engine, ICAP, and JTAG circuitry. Each SLR contains a dedicated configuration engine that is daisy-chained with the others to facilitate unified configuration. Configuration is initiated through the primary[1] SLR, which propagates configuration data sequentially down the chain. The primary configuration engine writes to the secondary SLR, which in turn configures the next SLR. Notably, this creates a unidirectional dependency: an SLR can access all downstream SLRs, but cannot access the configuration engine of an SLR higher up in the chain. Our investigation of the SLR architecture aligns with previous findings by Manev et al. [27].

2.4 Cloud FPGAs

In recent years, CSPs have begun offering compute instances equipped with one or more FPGAs, allowing customers to configure them for a wide range of applications. Common use cases for such cloud-based FPGA instances include video transcoding, genomic research, and machine learning [32].

The CSP we examined in our case studies follows a common workflow for customer design development and deployment [33]. Specifically, they utilize AMD UltraScale+ FPGAs featuring SSI technology and leverage partial reconfiguration capabilities. A designated portion of the FPGA is reserved for a *shell* supplied by the CSP, which interfaces with the device's PCIe connection and manages both customer logic configuration and off-chip communication to the host system. The customer's design is passed to this shell, which uses an ICAP within the primary SLR to configure the remaining reconfigurable regions of the FPGA.

Importantly, resources occupied by the shell, as well as those situated within its reserved region, remain inaccessible to the customer and cannot be instantiated within their design. This isolation model applies not only to the host and the customers but also to all users in a hypothetical multi-tenant scenario. Notably, the shell must utilize at least the ICAP located in the primary SLR, as this is the only ICAP capable of configuring all other SLRs in the device. An example of the resource allocation for the Amazon EC2 F1 and F2 instances is illustrated in Figure 2.

For the development of a hardware design destined for deployment in the cloud, CSPs provide users with a Hardware Development Kit (HDK). This HDK includes the encrypted Intellectual Property (IP) core of the shell and the necessary scripts to synthesize the shell alongside the customer logic and link them into one unified design – ultimately producing a partial reconfiguration bitstream that enables dynamic swapping of user designs. The development process can be carried out in cloud-based instances preconfigured with AMD's tooling and

[1] AMD uses master/slave terminology in their documentation.

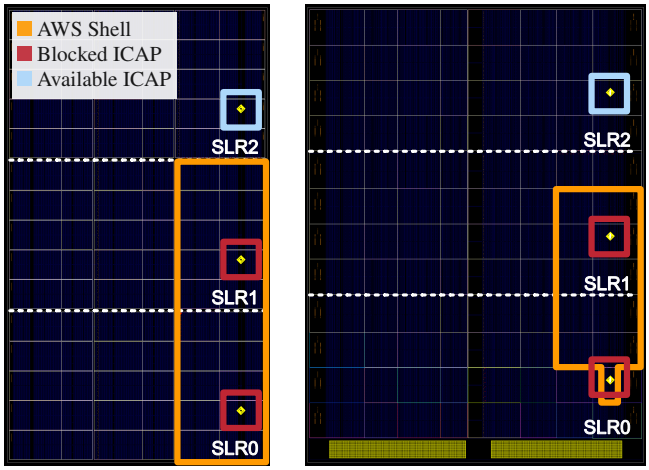

Figure 2: This figure shows the physical layout of the FPGAs utilized for Amazon F1 (left) and F2 (right) instances. Both FPGAs comprise three SLRs with SLR 1 being the primary SLR. The provider shell for both instances occupies resources on the two lower SLRs, including the respective ICAP locations. This leaves only the ICAP on SLR 2 available to the user.

the required licenses – available for rent from the CSP – or locally in a bring-your-own-license setup. While the CSP provides a functional development environment, customers retain substantial control over the process, apart from the encrypted shell. Once implementation is completed, the resulting DCP is uploaded to a cloud provider-controlled server. This server performs multiple verification checks (as discussed below) and creates an *FPGA image*, including the final bitstream, which can then be used to program the FPGAs in the cloud.

2.4.1 Cloud FPGA Defences

We adopt the terminology of La et al. [24], who categorized Amazon's cloud FPGA defenses into four levels of security fences. *Fence 1* performs multiple verification checks on the customer design, ensuring both security and functional compliance, for example, detecting combinatorial loops, unrouted nets, and preventing access to certain basic elements. *Fence 2* ensures that the bitstream generation is done exclusively by the CSP. Programming of this bitstream is only allowed via the hypervisor of the CSP (*Fence 3*). Thus, the user never has direct access to the generated bitstream, FPGA image, or to the programming of the FPGA. *Fence 4* completes the security mechanism by monitoring the FPGA during runtime, e.g., monitoring power consumption or temperature to halt the customer clock when certain thresholds are triggered (clock gating) or shut down the FPGA.

2.4.2 Cloud FPGA Attacks

Previously identified threats to cloud FPGAs include malicious electrical circuitry, ICAP attacks, and fingerprinting attacks.

Malicious Electrical Circuitry An often discussed attack vector is the insertion of malicious electrical circuitry into cloud FPGAs, which can either be used as power wasters or as sensing circuitry to measure voltage levels. The main primitives used as malicious electrical circuits include combinatorial loops (e.g., ring oscillators) and high fan-out nets.

Previous research identified DoS attacks using such power wasters capable of triggering thermal or voltage-induced shutdowns [24, 36]. The same primitives used for DoS can artificially age the FPGA, leading to a reduced device lifetime [7, 36]. With both attacks, the goal is to inflict economic damage on the CSP through downtime or required maintenance.

Other targets in usually non-reachable FPGA regions can include the CSP host system or other users in a multi-tenant scenario. Fault-injection attacks also rely on using power wasters to induce voltage spikes. Electrical faults can result in erroneous computation, allowing the extraction of cryptographic keys [30] or the manipulation of machine learning operations [31]. Another attack vector uses the potential sensing capabilities of ring oscillators or high fan-out nets to collect side-channel information either from other tenants' designs or system components. Such attacks can also leak sensitive data like cryptographic keys [45] or machine learning models [44] and their input [29].

ICAP Attacks Albartus et al. [1] have presented a set of attacks utilizing control over an ICAP that are applicable to cloud FPGAs.

Because of the ICAP's unrestricted connection to all configuration data inside an SLR, it can read back all configuration and user data. Besides granting readback and debug capabilities to the user's own region, an attacker could exploit multi-tenant environments or third-party IP core deployments. In multi-tenant scenarios, attackers can read back other users' designs to infringe on their IP and gain valuable insights. Even reading back their own design can result in IP infringements if the CSP configures third-party IP. Moreover, ICAP readback can be extended to capture internal FF states, potentially exposing sensitive or security-critical data (e.g., cryptographic keys or input data) belonging to co-located tenants.

ICAP access also allows design reconfiguration outside the assigned user region on a reachable SLR (see Section 2.3). This effort may include the insertion of hardware Trojans into the provider's shell or other tenants' designs. Classical hardware Trojans may degrade the security of cryptographic ciphers [5] or manipulate AI models for misclassification [6, 17]. These threats compromise the CSP's intended isolation both between users in multi-tenant environments and between users and the host shell.

Fingerprinting Attacks Tian et al. [37] identified FPGA fingerprinting as a prerequisite for targeted attacks, in both multi-tenant environments and time-shared scenarios, where an attacker's design is deployed after the victim's on the same hardware. Previous FPGA fingerprinting approaches use Physical Unclonable Functions (PUFs) inside DDR Random Access Memory (RAM) [24, 26, 37] to identify devices.

3 Privilege Escalation Overview

In this section, we provide an overview of a novel privilege escalation path applicable to AMD cloud FPGAs and illustrated in Figure 3. We begin by defining our attacker model. Subsequently, we present an overview of how attackers can incrementally increase privilege levels and the resulting capabilities, starting from the limited permissions of regular cloud users.

Attacker Model As an attacker, we assume a remote user on a cloud FPGA with access to standard vendor-provided tools for creating and deploying designs on cloud FPGAs. The CSP requires only minimal customer verification, limited to registration with a valid credit card and an email address. Notably, no identity verification documents are required. We assume that the attacker possesses basic knowledge of the FPGA bitstream structure, specifically the high-level aspects such as command sequences and bitstream frame structure. This information is accessible via official AMD documentation [43] and from existing literature [27]. Our attacker model requires no additional privileges, resembling any cloud FPGA user, and therefore represents a low-power adversary.

Level 0 – Regular Access Before users can configure their design, the CSP initializes the FPGA by programming their shell (❶ in Figure 3). At this initial stage, regular cloud users are restricted by design security checks and verification processes enforced by the provider. Thus, only the instantiation of benign designs and attacks involving design components and primitives passing CSP verification are feasible at this level (❷ in Figure 3).

Level 1 – Configuration Engine Access Our proposed privilege escalation begins when an attacker obtains control over an ICAP (❸ in Figure 3). The practical feasibility of instantiating and using an `ICAPE3` primitive is discussed in Section 4.1.

Deploying an FPGA image containing an `ICAPE3` escalates an attacker's privileges, as it enables injection of arbitrary configuration commands directly to the FPGA's configuration

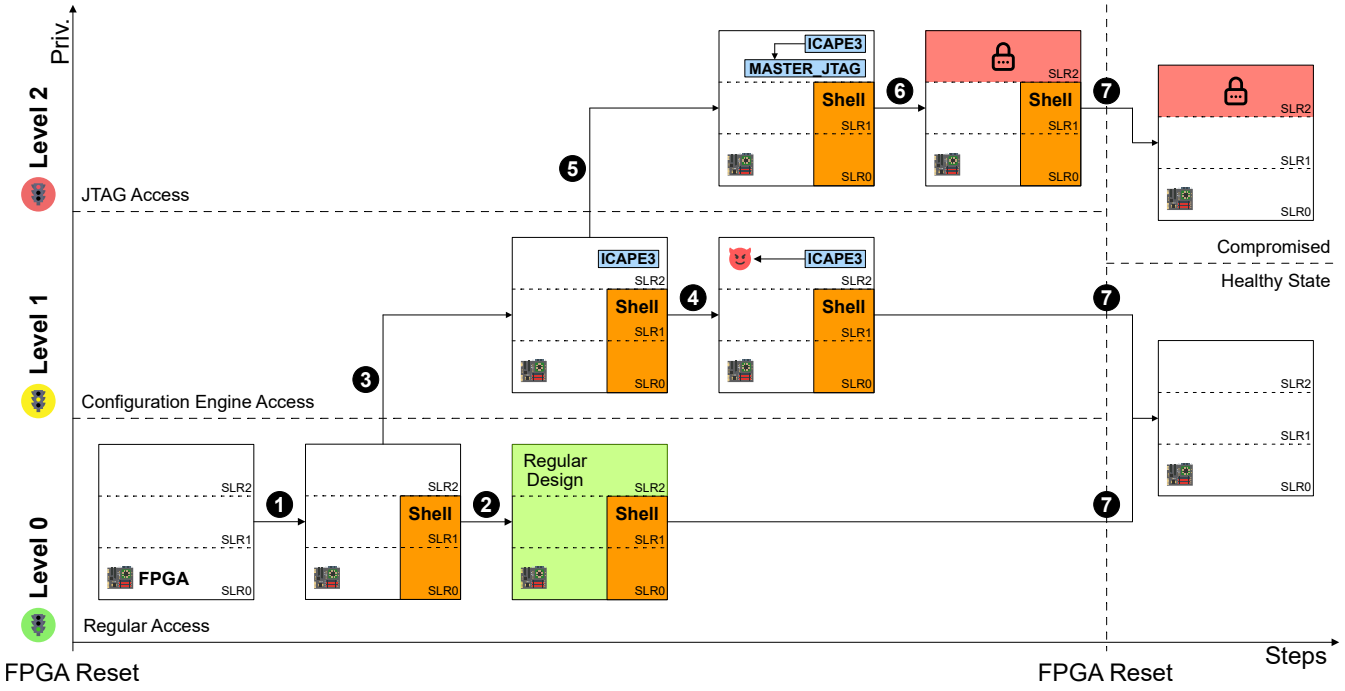

Figure 3: This diagram shows our privilege escalation path on cloud FPGAs. First, the CSP loads the shell onto the empty FPGA (❶). Afterward, a user configures their design through the shell (❷). To escalate privileges, an attacker could instantiate an `ICAPE3` primitive (❸). With access to the ICAP, arbitrary malicious circuits can be configured (❹). Instantiating a `MASTER_JTAG` primitive further escalates an attacker's privilege (❺). With access to this internal JTAG interface, eFuses of the device could be programmed to lock the CSP out of the SLR (❻). This attack results in a persistent compromise, even after a reset (❼).

engine. With ICAP *write* access, attackers can circumvent most provider-imposed verification (Fence 1-3 [24]) and instantiate prohibited circuitry, effectively bypassing security policies (❹ in Figure 3 and Figure 4). Verification on the provider backend and countermeasures relying on the analysis of the customer design are currently unable to prevent such an attack [25, 40]. This is the case because the malicious circuitry would be configured only after the user design is deployed to the device. Only runtime checks (Fence 4) might warn the CSP about malicious actions, but our results and recent studies [24, 46] suggest these defenses are insufficient under certain sophisticated scenarios. With ICAP *read* access, the attacker can potentially extract configuration and (private) user data via a readback mechanism. Both read and write access breach the intended isolation model, especially in multi-tenant scenarios.

Level 2 – JTAG Access The second stage in our privilege escalation flow involves gaining remote access to an internal JTAG interface. Since CSP verification explicitly blocks direct instantiation of the `MASTER_JTAG` primitive, attackers must leverage their previously gained unrestricted reconfiguration capabilities to instantiate this component (❺ in Figure 3).

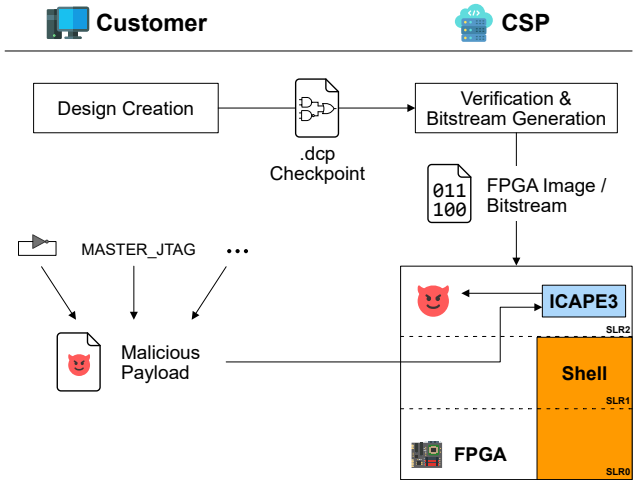

Figure 4: This figure shows the deployment flow of a customer's design to a cloud FPGA and how ICAP access allows bypassing vendor-sided verification pipelines.

We discuss the technical feasibility and the challenges we encountered when instantiating a `MASTER_JTAG` in Section 4.2. Gaining remote JTAG access provides attackers with exten-

sive privileges, including the critical ability to read from and write to an FPGA's eFuse memory (❻ in Figure 3). This elevated access enables the extraction of sensitive device information and introduces the risk of persistent compromise (❼ in Figure 3).

4 Implementation and Technical Challenges

This section assesses the feasibility and technical implementation of our proposed privilege escalation in a real-world scenario. Based on the attacker model from Section 3, we illustrate how the investigated privilege escalation attacks can be executed in Amazon EC2 F1 and F2 cloud instances. The targeted AWS cloud instances use AMD Virtex UltraScale+ FPGAs with designated part numbers `VU9P` for F1 and `VU47P` for F2. Although minor differences exist among our target platforms, the technical challenges remain consistently applicable. Initially, we explore the implementation of our Level 1 privilege escalation via ICAP access. Secondly, we demonstrate the feasibility of escalating to the JTAG interface (Level 2).

4.1 Configuration Engine Access (Level 1)

To escalate privileges to Level 1, we must gain control of the configuration engine. For this purpose, we instantiate an `ICAPE3` primitive in a user design. In that regard, AMD explicitly states:

> *RECOMMENDED: Only one* `ICAPE3` *resource should be instantiated for an UltraScale FPGA, and it should be automatically placed by the tools.* [43]

> *Only configuration components must remain in the static part of the design. These components are:* `[...]`, `ICAP`, `MASTER_JTAG`, `[...]` [42]

Since the AWS shell already uses an ICAP in the primary SLR and the user design is dynamically reconfigured into the FPGA, it first seemed questionable whether a customer can instantiate a functioning `ICAPE3` primitive. Nevertheless, vendor tools do not impose hard limitations that prevent placing an additional ICAP in a secondary SLR that is not occupied by the shell. Also, AWS does not perform checks in its verification pipeline, prohibiting the instantiation of an `ICAPE3` primitive. However, as the vendor documentation cautions against additional `ICAPE3` instantiation, it is to be expected that our implementation would face several challenges. In the following sections, we will first provide a high-level description of each challenge and its solution, followed by a detailed discussion.

4.1.1 Loss of Daisy Chain Connection

Challenge

Placing an `ICAPE3` into a secondary SLR causes failure of the daisy chain connection between configuration engines on different SLRs.

Solution

Manually constraining the routing and restarting the SLR to recover from the interrupted configuration process enables a usable connection to the newly instantiated ICAP.

Our design contains an ICAP controller serving as an intermediary between the Amazon shell and the `ICAPE3`. With this setup, we can forward user input, such as configuration data or control commands, to the configuration engine. An implementation utilizing the automated placement and routing of the vendor tools does not work and leaves parts of the implemented design unresponsive. Based on our observations, we assume that as soon as the ICAP on the primary SLR processes bitstream frames, which configure and activate a previously unused ICAP on a secondary SLR, configuration daisy-chaining is broken (see Section 2.3). From that point on, only the ICAP within the targeted SLR can configure the fabric in that SLR, and the ICAP in the primary SLR no longer has control. Hence, all bitstream configuration frames destined for that SLR and located after the `ICAPE3` frame within the bitstream are ineffective, as they are no longer relayed to the configuration engine on the secondary SLR.

This has two unwanted side effects:

- All circuitry that is located after the `ICAPE3` frames in the bitstream is not configured.
- The secondary SLR is halted during reconfiguration. As the restart commands at the end of the bitstream also do not reach their target, the SLR remains in a frozen state.

To solve these issues, we first had to avoid using any gates or wire connections that are configured with frames located after `ICAPE3` frames (❶ in Figure 5). The order of the bitstream frames maps to the physical design of the FPGA, as shown in Figure 6. This means that frames are configured in the order they appear in the bitstream, starting from the bottom-left corner of the device, proceeding left-to-right, then bottom-to-top [27]. To ensure that we used no frames after the `ICAPE3` instantiation, we restricted all wires and gates to be physically placed in a region below or to the left of the `ICAPE3`.

Second, the missing restart sequence for the secondary SLR leaves all Flip Flops (FFs) in the secondary SLR frozen after configuration. As a consequence, FFs and other elements cannot be used in the initially configured design of the attacked

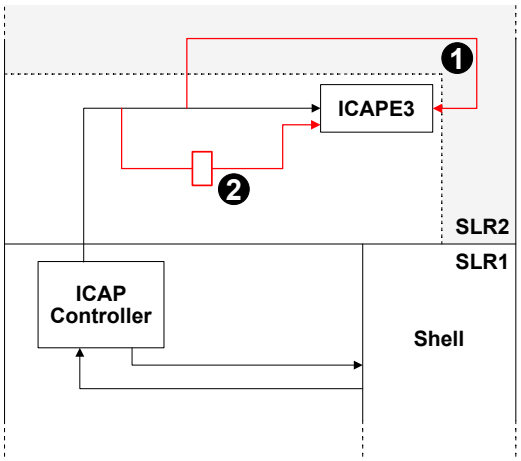

Figure 5: Schematic of our ICAP controller and its connection to the shell and ICAPE3 primitive. The red connections show dysfunctional routing (❶) and placement of elements (❷).

SLR (❷ in Figure 5). However, the routing and ICAP remain unaffected. Thus, control circuitry for ICAP communication must be placed on a different SLR that is configured earlier in the configuration daisy chain and, therefore, properly restarted. This placement, resulting in only wires running through the secondary SLR and connecting to the ICAP, can be achieved through placement constraints in the design process. In our experiments, the HDK always provides the required space for the control circuitry in an alternate SLR. Based on the small size of 1931 gates of our control circuitry, we assume that SLR placement of this circuitry is a noncritical constraint.

With such a setup, we can communicate with the configuration engine of the secondary SLR and write frames to configure its circuitry. Finally, by sending the usual sequence of startup commands to the ICAP, we restarted the secondary SLR. This restart fully resumes SLR operation, providing unrestricted autonomy of its configuration, even beyond the frames of the ICAPE3 (❶ in Figure 5, Figure 6). However, we cannot gain access to previous SLRs in the daisy chain.

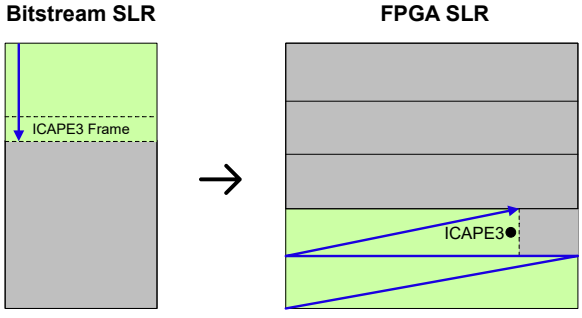

Figure 6: Illustration of how frame positions in the bitstream file map to the physical location on the device fabric.

4.1.2 Generating Payload Frames

Challenge

Vendor tooling does not support generating minimal frame manipulations that instantiate a MASTER_JTAG primitive. Assembling such bitstream commands is challenging due to the proprietary bitstream file format.

Solution

By locally generating bitstream variants with and without the MASTER_JTAG primitive and extracting their frame differences, we can build the required payload frames.

We want to send arbitrary configuration frames to the ICAP that we previously gained access to, allowing the FPGA to be maliciously reconfigured. A key challenge is the obscurity of the bitstream format, a consequence of its proprietary nature. While the bitstream format for older AMD series is known (e.g., by Project X-Ray [14]), only parts of the UltraScale(+) bitstream format have been reverse engineered [20]. Therefore, we must craft configuration frames defining specific circuitry on the FPGA without knowing the exact mapping of bitstream bits to hardware components.

However, this challenge can be overcome using AMD tools. We create two bitstreams – one containing a malicious payload on the target SLR, and another one without it. To identify the exact frames that program the malicious circuitry, we compare the two bitstreams and locate the differences. Since the only variation between the two is the presence of the payload circuitry, we can determine which frames to dynamically reconfigure via the ICAP to deploy the malicious modification. It is important to note that since the ICAP only has access to the secondary SLR (and potentially any SLRs further down the chain), the payload must be located in this/these SLRs.

For this process, we created a separate hardware design with only a placeholder for the shell and implemented our payload in a local environment. First, this allows for easier placement control than using the Amazon HDK and still generates the exact payload frames required for an attack on the cloud FPGAs. Second, for Amazon F2, the HDK restricts full bitstream generation in favor of partial bitstreams, which remain difficult to analyze. Partial bitstreams cannot be generated as debug bitstreams, which include explicit frame addresses and would allow for easier analysis. Further, existing open-source tooling cannot reliably handle partial bitstreams.

For the analysis of the two bitstreams, we utilized the open-source bitstream analysis framework Bitfiltrator [20, 21], which enables the extraction of frame content. We select all frames in the two bitstreams that differ, and build a manipulation bitstream, overwriting all selected frames with the frames from the bitstream variant containing our payload. An example bitstream for frame manipulation is shown in Listing 1.

4.2 JTAG Access (Level 2)

With control over the configuration engine of a secondary SLR, we can further escalate our privileges on the FPGA. Figure 7 shows the JTAG controller we implemented, which allows communication from the user to the MASTER_JTAG via the shell. By connecting to the MASTER_JTAG primitive, we can use the JTAG port from within the user design. Typically, the instantiation of the MASTER_JTAG primitive inside a reconfigurable region is blocked in the AWS verification pipeline, as it raises a DRC error preventing bitstream generation. However, using our escalated privileges to Level 1, we can instantiate and connect the MASTER_JTAG by generating the frames locally and sending them to the configuration engine unchecked. This instantiation is shown with the dotted lines in Figure 7 (❶), everything else is part of the user design.

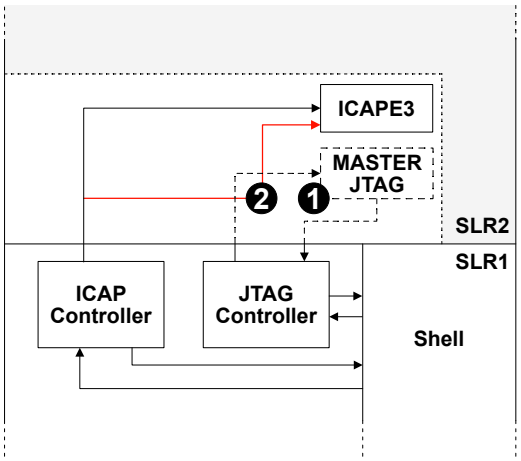

Figure 7: Schematic of our JTAG controller and its connection to the shell and MASTER_JTAG primitive. The dotted lines show the elements reconfigured via the ICAP (❶), and the red line depicts a problematic routing overlap with the ICAP routing (❷).

4.2.1 Routing Constraints Around the ICAP

Challenge

Reconfiguring the routing leading to the MASTER_JTAG might break the ICAP connection when routes running to the ICAPE3 and the MASTER_JTAG are configured inside the same frames.

Solution

Using manual constraints that restrict the two sets of routes to different frames allows for the safe reconfiguration of the MASTER_JTAG routing.

The MASTER_JTAG is configured within the same frames as the ICAPE3. Still, the ICAP is able to write and modify its own configuration frames, activating the neighboring MASTER_JTAG without permanently disabling itself. However, care was needed when placing the routes to the MASTER_JTAG as they run in close proximity to the routes connecting the ICAPE3 primitive to our control circuitry. The ICAP on the secondary SLR can modify the very SLR that it is placed on and can therefore also manipulate the wires connecting our controlling circuitry to the ICAP itself. We had to ensure that the routes to the ICAPE3 and MASTER_JTAG were not configured within the same configuration frames, as otherwise, the ICAP could lose connectivity by interfering with its own routing. In Figure 7, we illustrate an invalid routing with the intersection of the red wire leading towards the ICAPE3 and the dotted wire towards the MASTER_JTAG (❷). A solution for this is illustrated by the black wire, which leads to the ICAPE3 while maintaining physical distance from the dotted wire and not intersecting with the MASTER_JTAG routing. Using routing constraints, we separated the two sets of wires into different configuration frames. We implemented a controller that sends arbitrary command sequences to the JTAG interface and captures the output. By establishing a connection between our controller and the MASTER_JTAG, we are now in remote control of the security-critical infrastructure of the secondary SLR, including its eFuses. This opens up the novel attack vectors of reading and writing eFuses.

5 Evaluation

In this section, we verify the technical feasibility of the presented privilege escalation path first on a local test setup and then on the AWS EC2 cloud FPGA instances F1 and F2. Additionally, we discuss and evaluate the potential threat vectors that each capability opens up.

Local Test Setup For our local test setup, we use a VCU118 development board equipped with the same FPGA as the AWS F1 instances. To develop and evaluate our privilege escalation path, we implemented a minimal partial reconfiguration design similar to the partial reconfiguration component of the AWS shell. In this design, an ICAP on SLR 1 programs a reconfigurable partition that contains a second ICAP on SLR 2. This setup enables faster attack prototyping, identifying working placement and routing constraints, as well as obtaining debugging information via the JTAG interface. Additionally, we locally prototyped both the ICAP and JTAG controllers, as well as the JTAG command sequences used to read out the configuration engine and eFuse registers.

5.1 Level 1 – Configuration Engine Access

We begin by evaluating the attack vectors emerging from the first privilege level and conduct case studies to verify their real-world potential.

Attack Vector – Read Data Because of the ICAP's unrestricted connection to all configuration data inside an SLR, it can read back all configuration and user data, enabling an attacker to potentially exploit multi-tenant environments or third-party IP core deployments (see Section 2.4.2). The readback scenario was introduced by Albartus et al. [1] and already demonstrated on a test setup. The only thing missing from their practical evaluation was a demonstration showcasing a working ICAP instantiation in a reconfigurable flow. Through the following case studies, we demonstrate that it is possible to gain full control over an ICAP and complete their work.

Attack Vector – Write Data With ICAP write privileges, we can configure arbitrary circuitry on reachable SLRs, circumventing any provider design checks and breaking isolation models. This fulfills the prerequisites necessary to re-enable the hardware Trojan attacks and the insertion of malicious electrical circuitry, mentioned in Section 2.4.2. These attacks have already been explored in previous literature, but their execution through ICAP is novel.

Furthermore, reconfiguration capabilities on reachable SLRs enable the circumvention of some vendor defenses. A defense mechanism of AWS in Fence 4 is clock gating [24], which allows the shell to control or stop the user's clock. However, a customer can instantiate ring oscillators that can be used as an internal clock source. This renders the design independent of an external clock source, thereby reducing the efficacy of Fence 4. An especially noteworthy modification in light of the previously discussed electrical attacks would involve the manipulation of a system monitor; for example, La et al. mention that Amazon uses a monitor to observe voltage and temperature levels [24]. If such defenses reside in a reachable region, manipulation could allow an attacker to remove countermeasures, weakening Fence 4.

To demonstrate the feasibility of bypassing the verification pipeline via the ICAP, we developed a small-scale, non-destructive ring oscillator example to showcase the injection of malicious circuitry usually prohibited by provider checks. For this purpose, we created two bitstreams as described in Section 4.1.2 to extract the malicious payload frames. One (benign) variant employed a clock wire permanently connected to logic zero, while the other (malicious) utilized a 15-LUT ring oscillator to drive this wire. The clock wire's activity was externally observable through an attached counter. To synthesize the oscillator-based variant locally, we disabled the DRC rule `LUTLP-1`, which normally prohibits combinational loops in LUT configurations. Deploying the (benign) oscillator-free design on a cloud FPGA and subsequently reconfiguring the malicious frames via the ICAP allowed us to remotely activate and observe the ring oscillator.

5.2 Level 2 – JTAG Access

We continue our evaluation with a discussion on additional attack vectors emerging from the second privilege level and conduct further case studies.

Attack Vector – Read eFuses To investigate the attack vector of eFuse reading, we used our JTAG controller to scan the whole eFuse memory of the SLR. We could read out all registers except for the AES key, which is unreadable. This effort allows for a novel fingerprinting attack by reading the device's internal DNA value. The value is a unique and immutable device ID that is factory-programmed into the eFuse register `FUSE_DNA` by AMD. AWS keeps this value secret from their users and explicitly blocks access to it [24]. By reading the `FUSE_DNA` register, we uniquely identified the physical devices on which our designs were running.

An attacker could profile the cloud FPGA fleet by starting many instances repeatedly and reading out the DNA of each device, tracking how many unique devices there are in total. This capability may also enable usage estimations for competitor analysis. AMD also offers a service where each DNA value is mapped to an exact device type, which could leak information about Amazon's supply chain. Lastly, knowing an FPGA's DNA value would allow for lifecycle analysis, potentially allowing an FPGA to be traced back to its service in Amazon's fleet. By enabling direct access to the unique device DNA value, our approach makes such fingerprinting attacks (Section 2.4.2) trivial and 100% accurate. Additionally, read access to the eFuses allows an attacker to read out security-relevant settings located in the `FUSE_SEC` and `FUSE_CNTL` registers [43].

Attack Vector – Write eFuses The attack vector of writing eFuses bits enables a novel ransomware attack in which an adversary takes over parts of the FPGA by programming cryptographic keys into the eFuse array.

To successfully carry out this attack, an adversary must program several eFuses. Specifically, cryptographic keys must be written into the `AES_KEY` and `RSA` registers. Additionally, the attacker must configure the `FUSE_SEC` registers to enforce authenticated bitstreams encrypted with the programmed eFuse keys. The adversary must disable further eFuse programming by setting the corresponding bits in the `FUSE_CNTL` register to ensure AWS stays locked out. Otherwise, Amazon could attempt to hack their own devices by programming all unprogrammed eFuses, for example, setting the AES key to all ones. Both encryption and authentication are necessary, as previous research on FPGA security has demonstrated that relying on only one of these protections leaves the system vulnerable to attacks such as *starbleed* [10] and *JustSTART* [9].

As programming eFuses causes permanent damage to the devices, we did not test this on Amazon's FPGAs. Instead, we verified the attack in a controlled local environment. However, we did inspect the `FUSE_SEC` and `FUSE_CNTL` registers on the cloud devices to confirm that AWS did not prevent the programming of eFuses. Amazon confirmed that they do not foresee any obstacles in programming eFuses using our approach.

When such an attack is successful, Amazon loses the ability to reprogram the affected SLRs, as they cannot produce valid bitstreams without knowledge of the keys. Key programming effectively functions as a ransomware attack. Instead of encrypting user data, the attacker encrypts hardware functionality, effectively reducing the compute power provided by the FPGA. Such an attack would lead to AWS losing all available resources in the vulnerable SLRs. Amazon's F1 and F2 cloud FPGAs would lose one of three SLRs, not accounting for the space required by the shell. The attackers could subsequently demand ransom payments from Amazon in exchange for providing the keys required to restore regular operation. If Amazon does not want to pay the ransom, they could mitigate the impact by excluding affected SLRs from their bitstream generation process. However, this workaround causes significant overhead, as all customer designs must be newly implemented for the affected FPGAs, as they can no longer use the encrypted SLR. Furthermore, depending on the FPGA model, vital I/O-ports or functional blocks physically located in the compromised SLRs become permanently unusable.

5.3 Escalation to Flash Access

During responsible disclosure, AMD identified an alternative privilege escalation enabled by the demonstrated ICAP access. Instead of reconfiguring a `MASTER_JTAG` primitive, an attacker could also instantiate a `STARTUP` primitive, which grants read and write access to the non-volatile on-board flash memory storing the FPGA boot image. An attacker could therefore overwrite the boot image or erase the flash memory, requiring the device to be re-provisioned. While we did not experimentally validate this behavior, it is confirmed by the vendor and documented in an official design advisory [4].

5.4 Limitations

We acknowledge several limitations of the practical impact of the presented attack vectors and our evaluation.

Although our privilege escalation enables arbitrary configuration access on reachable SLRs, it does not necessarily grant control over any shell components. On the evaluated AWS F1 and F2 instances, critical shell logic is confined to SLRs 0 and 1, while an attacker is only able to compromise the ICAP on SLR 2. As a result, our practical attacks are limited to resources within or reachable from SLR 2.

Further, several attack scenarios assume multi-tenant FPGA deployments, which, to the best of our knowledge, are not offered by major CSPs, rendering these threats hypothetical in today's commercial cloud environments. Nevertheless, multi-tenant FPGA sharing remains a relevant consideration due to its potential efficiency benefits.

Finally, our evaluation is limited to AMD Virtex UltraScale+ FPGAs as deployed in AWS F1 and F2 instances, and therefore, the applicability of our findings to other devices or vendors may be limited. Nevertheless, AMD's design advisory contains a list of 19 affected FPGA devices [4].

6 Discussion

This section first discusses the impact of the presented ransomware attack, the most significant consequence of the novel privilege escalation path. We propose countermeasures on multiple levels to prevent such privilege escalation and harden cloud FPGA environments against future threats. Furthermore, we discuss security shortcomings. Lastly, we explore other FPGA privilege escalation scenarios in which configuration capabilities and debugging interfaces might be accessible. In doing so, we raise awareness among FPGA owners of the previously underexplored risks of ICAP and JTAG access.

6.1 Impact of Ransomware Attack on Cloud FPGAs

We hereafter estimate the potential impact of a ransomware attack on Amazon cloud FPGAs where the devices are encrypted by an adversary and the CSP loses access to an SLR. Considering the consequences, it is reasonable to assume that Amazon would replace FPGA instances that fell victim to a ransomware attack. A single Amazon F1 instance can host up to eight FPGAs, each retailing at roughly $40,000 [8], totaling $320,000 per instance. While CSPs likely receive volume discounts, the financial impact remains significant.

To assess the potential financial impact, we work on assumptions regarding the amount of FPGAs and the attacker's ability to instantiate them. The amount of FPGAs owned by Amazon is unknown to the public. However, Mahmod et al. [26] reported the instantiation of 300 unique devices. In the absence of more recent or official data, we adopt this figure as a conservative lower bound for our calculations. Multiplying the device count estimate by their retail pricing, the potential financial impact for F1 instances alone presumably could reach millions of USD.

Inflicting the maximum financial damage is only possible under the assumption that an attacker can instantiate an unrestricted number of instances. However, CSPs do have restrictions in place preventing a trivial scaling of such an attack. For example, users get assigned a quota of instances that they are allowed to start simultaneously. Additionally, CSPs may have implemented further safeguards to mitigate

malicious behavior. These might include restricting users to designated FPGA clusters or deploying monitoring systems that flag irregular usage patterns and a high number of non-starting FPGA instances. However, a determined attacker might employ various strategies to bypass these restrictions. For example, operating multiple accounts simultaneously, a strategy used in the area of free-trial-abuse [38] or targeted side channel attacks [41]. Another method is to leverage social engineering tactics to obtain increased contingents by posing as a legitimate large-scale customer [16].

As described in Section 3, we assume a low-power attacker model and have no requirements about the state of the target system. Finding the vulnerability and developing an exploit took hundreds of working hours. In comparison, starting an instance and deploying the malicious image takes minutes, while the reconfiguration and eFuse programming only take seconds. This process can be automated, and similar to a zero-click exploit, no interaction from a potential victim is required. This combination makes the threat particularly severe.

6.2 Countermeasures

Our presented privilege escalation path consists of multiple interdependent stages, in which each step relies on the successful completion of the previous one. There are several straightforward opportunities to either fully mitigate the threat or significantly limit its impact.

Banning the ICAPE3 primitive The most effective way to prevent our privilege escalation is to terminate it at the lowest level and forbid the instantiation of the `ICAPE3` primitive. This can be achieved by throwing a DRC error in the AMD tooling, as the documentation suggests [42]. Alternatively, CSPs do not have to rely on AMD for countermeasures and can instead block instantiation by extending verification checks on the customer design (Fence 1). Overall, banning the `ICAPE3` primitive in user designs can be implemented as a simple type check of instantiated primitives in the one-time executed verification pipeline. In response to our findings, Amazon blocked the usage of the `ICAPE3` primitive in customer designs.

ICAP Firewall The drastic measure of banning the `ICAPE3` primitive would prohibit users from deploying possible legitimate ICAP use cases, for example, the use of the Soft Error Mitigation (SEM) IP core [2]. To restrict ICAP functionality while giving users the freedom to use reconfiguration functionality, one could implement an *ICAP firewall* [1] as part of the provider shell. For example, this firewall could prevent the activation of powerful primitives, such as the `MASTER_JTAG`, and only allow harmless updates to the configuration engine. However, developing such a firewall is associated with substantial costs. Its implementation needs to be carefully integrated into the provider shell and designed to avoid exploitable bugs. Therefore, it is only appropriate in niche situations.

Preemptive eFuse programming An effective countermeasure to mitigate the catastrophic consequences of unauthorized eFuse programming is the proactive configuration of the eFuses by the CSP. Multiple eFuses can be programmed to lock down specific functionalities and prevent any further modifications to the eFuse memory. For example, it is possible to prohibit writing critical security settings, such as cryptographic key material, and ensure the enforcement of authenticated or encrypted bitstreams. This approach is highly effective, as it prevents a persistent impact. However, since programming the eFuses is irreversible, it can be disadvantageous for the CSPs to restrict their ability to update security settings or key material.

Infrastructure-based As discussed in Section 6.1, another limiting factor in launching the ransomware attack at scale is the attacker's ability to reach a large number of physical devices. For this purpose, CSPs could flag users who quickly start and stop instances and restrict their access to minimize the number of potentially affected FPGAs. In addition, an alarm could be triggered if too many instances fail a wellness check.

Pitfalls Implementing countermeasures can be challenging, as modern FPGAs are highly complex systems and navigating security documentation and deploying secure infrastructure is non-trivial. Moreover, the documentation can be out-of-date or misleading [1, 3]. It is imperative for users to exercise caution, as numerous options exist that possess ambiguous security implications. Some of these settings correspond to bits in the configuration data and register values in the configuration engine that an attacker with control over an ICAP can manipulate.

In summary, it is advisable to implement as many security measures as possible throughout the deployment stack of cloud FPGAs. Deciding which countermeasures to implement is highly dependent on the cloud provider's particular use case and long-term strategy; hence, there is no one-size-fits-all solution.

6.3 Security Shortcomings

In the following, we examine the security shortcomings identified during our case studies.

Forbidden Primitive Instantiation At the heart of this privilege escalation lies the control over an ICAP. There exists a mismatch between intended design guidelines and actual hardware capabilities. For example, UG909 [42] explicitly forbids the instantiation of certain elements in reconfigurable regions. Therefore, security designers might assume that these primitives pose no viable attack vector. However, we showcased a

successful instantiation of an ICAPE3 inside a dynamic user region, proving this assumption wrong. Similarly, the requirement for placing MASTER_JTAG in the static region is enforced only as a software-sided design rule check, which can be disabled by the user. This weakness indicates an overreliance on non-binding guidelines rather than hardware-level enforcement. Our findings confirm that no physical limitations prevent the exploitation of the MASTER_JTAG, showcasing that software-based barriers are insufficient for protecting against determined adversaries.

Remote eFuse Access Although cloud customers typically lack direct physical access to the underlying infrastructure, the ability to deploy custom hardware designs can introduce similar risks. Even though the device is not physically accessible, the internal JTAG port is available remotely from within the user design, opening up a novel attack vector. Therefore, cloud FPGAs require additional security consideration regarding JTAG capabilities, including eFuse programming.

Negligence for the Attacker Model While AMD's user guides mention restrictions regarding configuration components, e.g., ICAPE3 and MASTER_JTAG in reconfigurable regions, these limitations are based on technical feasibility and best practices – not mentioning any security impact in cloud environments [42]. Additionally, despite the existence of multiple countermeasures against privilege escalation and emerging attack vectors, none appear to have been implemented by the investigated CSP. In the past, AMD was aware of ICAP security risks; however, they only ever considered an external adversary in their threat model, not considering attackers located on the device [39]. This suggests that the prevailing threat model for cloud FPGAs was not mature enough to accurately reflect the realities of adversarial users.

Our findings reveal a meaningful parallel between traditional embedded system attacker models and those emerging in cloud FPGA environments. In particular, there is conceptual overlap with the man-at-the-end (MATE) attacker model, which assumes that the adversary has extensive control over a system's execution environment – often including physical access or the ability to manipulate low-level hardware features. Our work illustrates how attackers could gain a surprising degree of low-level control.

6.4 More FPGA Privilege Escalation Scenarios

While our case studies focus on a concrete, real-world privilege escalation scenario in cloud-based FPGAs, the implications of our findings extend beyond the cloud context. Any scenario in which an attacker gains access to the configuration engine or the JTAG interface introduces attack vectors investigated in Section 3. For JTAG access, this is not limited to eFuse-based attacks, but also enables the attack vectors on Level 1, as the JTAG also interfaces the configuration engine.

Third-Party IP Core A malicious third-party IP core designer could trojanize an IP core by secretly including an ICAPE3 or MASTER_JTAG primitive. As Albartus et al. [1] have demonstrated, these capabilities can be used for stealthy hardware Trojan insertion. Our work demonstrates that the insertion of such primitives elevates an attacker's privileges on the target device by gaining access to the configuration engine and JTAG. Again, this enables the adversary to read and program eFuses – unless explicitly prevented by the respective eFuses in the FUSE_CNTL register. Similar to our proposed ransom scenario, this would enable malicious IP designers to take over a user's FPGA. This fundamentally undermines assumptions about the isolation and security of third-party IP in many deployment scenarios. Users should be aware of the possible privilege escalation even from IP cores with a legitimate interest in using an ICAP and consider the same countermeasures discussed in Section 6.2.

Bitstream Manipulation Another potential vector for privilege escalation is bitstream manipulation, which can occur during bitstream generation, storage in memory, or transmission to the FPGA. An attacker with such access could inject malicious primitives directly into the bitstream, enabling the execution of previously described attacks. Moreover, this approach can bypass verification methods based on netlist inspection.

6.5 Related Work

In this section, we contextualize our work within the broader academic landscape of cloud FPGA security research. A comprehensive overview of known attack vectors was previously presented in Section 2.4.2. Our proposed privilege escalation and subsequent ransomware attack present a novel threat, distinct from existing research. Specifically, we demonstrate a persistent compromise of CSP resources by targeting a previously unexplored attack vector: the eFuses of the FPGA. Although the malicious potential of ICAP usage was initially theorized by Albartus et al. [1], we advance their work by demonstrating its real-world feasibility in cloud environments and its utility for privilege escalation. Furthermore, our research is the first to include a successful attack on Amazon EC2 F2 instances, which have been available since December 2024 [34].

In the security fence model introduced by La et al. [24], our privilege escalation attack bypasses Fences 1 through 3, and the ransomware scenario even circumvents Fence 4, as it does not rely on electrical-level attacks.

Finally, Jin et al. [19] survey the academic work on cloud FPGA security and categorize attackers into four groups. These are malicious cloud providers, malicious co-tenants, malicious IP providers, and malicious FPGA tools. However, they do not consider a malicious user targeting the CSP.

6.6 Future Work

Our findings underscore the need to reevaluate threat models in cloud FPGA environments, particularly with respect to low-level hardware access, as well as reconfiguration or hardware debugging features exposed to tenant designs. Similar attack surfaces may exist in emerging reconfigurable and accelerated cloud platforms, such as FPGA-based smart network cards and domain-specific accelerators (e.g., for machine learning), which likewise integrate sensitive low-level reconfiguration capabilities. Privilege escalation might also pose a risk in other scenarios (see Section 6.4), where refining threat models might be advisable.

In addition to mapping exposed attack surfaces, future work could include the development and evaluation of defenses beyond the ones discussed in Section 6.2. This could include hardware restrictions that prevent the instantiation or use of certain critical primitives. If low-level access to reconfiguration is necessary, a promising direction is the implementation of an ICAP firewall (see Section 6.2). Further research should also investigate the feasibility of integrating lightweight, real-time monitoring of security-critical components – potentially embedded in the CSP shell. These monitors could provide anomaly detection, runtime policy enforcement, or telemetry logging to enable post-incident forensics and accountability.

7 Conclusion

In this work, we expose a critical and previously unconsidered threat within the cloud FPGA ecosystem by demonstrating a privilege escalation path via ICAP that culminates in remote JTAG access. Thus, every regular cloud user could gain control over this powerful debug interface, permanently program encryption settings, prevent future reconfiguration, and lock the CSP out of their own devices. This persistent compromise of cloud infrastructure enables a novel form of attack: ransomware for FPGAs, where the CSP would have to pay a ransom for cryptographic keys to restore control over their hardware. We validate the feasibility of our privilege escalation as well as the enabled attack vectors through real-world case studies on Amazon's EC2 F1 and F2 cloud FPGAs instances. Although countermeasures are in place, we found that they do not consider the threat posed by low-level configuration and debugging interfaces. While the use of certain primitives is discouraged, restrictions on these components are non-binding and only enforced through software. The lack of effective countermeasures suggests that our novel threat vector was not considered during the security development process – possibly due to the atypical nature of remote JTAG access. As the adoption of reconfigurable hardware in the cloud accelerates, it is imperative that threat models evolve accordingly and that low-level configuration features are treated as high-impact, security-critical components.

Acknowledgments

We thank Julian Speith and Marc Fyrbiak for fruitful discussions. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy - EXC 2092 CASA – 390781972

Ethical Considerations

Our research involved several stakeholders, including Amazon, AMD, cloud customers, and the security community at large. We carefully considered the potential impact on these groups throughout the project. All experiments were conducted on local hardware to minimize the risk of accidental damage to the CSP's hardware. Only after safely reading eFuse registers locally did we move to a cloud setting, where we strictly limited our actions to non-destructive read operations. At no point did we attempt to irreversibly program eFuses on Amazon's devices.

After confirming that the issue extended to cloud-hosted FPGAs, we initiated a coordinated disclosure with AWS and AMD in April 2025. Since then, AWS has verified the privilege-escalation vulnerability, deployed mitigations to ensure it could no longer be exploited, and confirmed that the vulnerability could previously have been used to program eFuses in the cloud. These mitigations eliminated any risk of harm to CSPs and their customers before publication. AMD, in turn, notified its customers operating similar FPGA devices in other cloud environments and published a design advisory [4]. Throughout this process, we discussed our findings with both AWS and AMD.

Since the vulnerability has been mitigated and is no longer exploitable, we believe that publishing our findings is both ethically justified and in the public interest, as it promotes transparency and enhances the security of programmable hardware and cloud computing.

Open Science

In support of open science, we have released our local test designs and the Amazon HDK projects used in our cloud experiments. The published materials include FPGA bitstreams, source code, build scripts, and usage instructions. Because Amazon has patched the vulnerability, these artifacts cannot reproduce the exploit in a cloud environment. However, they enable researchers to validate and reproduce our methodology. All artifacts are available at: https://doi.org/10.5281/zenodo.16950451.

References

[1] Nils Albartus, Maik Ender, Jan-Niklas Möller, Marc Fyrbiak, Christof Paar, and Russell Tessier. On the mali-

cious potential of xilinx's internal configuration access port (ICAP). *ACM Trans. Reconfigurable Technol. Syst.*, 17(2):26:1–26:28, 2024.

[2] AMD. Reliability estimation • soft error mitigation controller product guide (pg036). [Online]. Available: https://docs.amd.com/r/en-US/pg036_sem/Reliability-Estimation, 2023. [Accessed 2025-04-13].

[3] AMD. 000037190 - ug570 - clarification on disable_jtag usage. [Online]. Available: https://adaptivesupport.amd.com/s/article/000037190?language=en_US, 2024. [Accessed 2025-04-13].

[4] AMD. 000038693 - design advisory for ultrascale/ultrascale+ ssit fpga as a service (faas) systems - preventing unauthorized access to fpga resources. [Online]. Available: https://docs.amd.com/r/en-US/000038693/000038693-Design-Advisory-for-UltraScale/UltraScale-SSIT-FPGA-as-a-Service-FaaS-Systems-Preventing-Unauthorized-Access-to-FPGA-Resources, 2025. [Accessed 2025-12-10].

[5] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware trojan attacks: Threat analysis and countermeasures. *Proc. IEEE*, 102(8):1229–1247, 2014.

[6] Joseph Clements and Yingjie Lao. Hardware trojan design on neural networks. In *IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, Japan, May 26-29, 2019*, pages 1–5, Sapporo, Japan, 2019. IEEE.

[7] Hayden Cook, Jonathan Thompson, Zephram Tripp, Brad L. Hutchings, and Jeffrey Goeders. Cloning the unclonable: Physically cloning an FPGA ring-oscillator PUF. In *International Conference on Field-Programmable Technology, (IC)FPT 2022, Hong Kong, December 5-9, 2022*, pages 1–10, Hong Kong, 2022. IEEE.

[8] Mouser Electronics. Amd virtex ultrascale+ fpga - field programmable gate array – mouser europe. [Online]. Available: https://eu.mouser.com/c/semiconductors/programmable-logic-ics/fpga-field-programmable-gate-array/?m=AMD&series=XCVU9P&tradename=Virtex%20UltraScale%2B, 2025. [Accessed 2025-04-09].

[9] Maik Ender, Felix Hahn, Marc Fyrbiak, Amir Moradi, and Christof Paar. Juststart: How to find an RSA authentication bypass on xilinx ultrascale(+) with fuzzing. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):426–450, 2024.

[10] Maik Ender, Gregor Leander, Amir Moradi, and Christof Paar. A cautionary note on protecting xilinx' ultrascale(+) bitstream encryption and authentication engine. In *30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2022, New York City, NY, USA, May 15-18, 2022*, pages 1–9, New York City, NY, USA, 2022. IEEE.

[11] Maik Ender, Amir Moradi, and Christof Paar. The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series fpgas. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1803–1819, Virtual, 2020. USENIX Association.

[12] Nuvation Engineering. Low-cost portable ventilator. https://www.nuvation.com/low-cost-portable-ventilator, 2025. [Accessed 2025-04-14].

[13] Defense Express. Encryption microchip from aliexpress found inside russian portable "azart" transceivers. [Online]. Available: https://en.defence-ua.com/weapon_and_tech/encryption_microchip_from_aliexpress_found_inside_russian_portable_azart_transceivers-4907.html, 2022. [Accessed 2025-04-14].

[14] F4PGA. Project X-Ray. [Online]. Available: https://github.com/f4pga/prjxray, 2017. [Accessed 2025-04-14].

[15] Dennis R. E. Gnad, Fabian Oboril, and Mehdi Baradaran Tahoori. Voltage drop-based fault attacks on fpgas using valid bitstreams. In Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi, editors, *27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017*, pages 1–7, Ghent, Belgium, 2017. IEEE.

[16] Google Cloud Threat Intelligence Team. Threat horizons: Cloud threat intelligence. Report Issue 1, Google Cloud, November 2021. Available at: https://services.google.com/fh/files/misc/gcat_threathorizons_full_nov2021.pdf.

[17] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[18] Business Research Insights. FPGA Cloud Server Market Size, Share, Growth, and Industry Analysis, By Type (16G, 32G, 64G, and Other), By Application (Face Recognition, Smart Home, Intelligent Transportation, Genetic Sequencing, and Other), Regional Insights and Forecast From 2025 To 2033. [Online]. Available: http

s://www.businessresearchinsights.com/market-reports/fpga-cloud-server-market-112875, 2025. [Accessed 2025-04-07].

[19] Chenglu Jin, Vasudev Gohil, Ramesh Karri, and Jeyavijayan Rajendran. Security of cloud fpgas: A survey. *CoRR*, abs/2005.04867:32, 2020.

[20] Sahand Kashani, Mahyar Emami, and James R. Larus. Bitfiltrator: A general approach for reverse-engineering xilinx bitstream formats. In *32nd International Conference on Field-Programmable Logic and Applications, FPL 2022, Belfast, United Kingdom, August 29 - Sept. 2, 2022*, pages 1–8. IEEE, 2022.

[21] Sahand Kashani, Mahyar Emami, and James R. Larus. Bitfiltrator: A general approach for reverse-engineering Xilinx bitstream formats. [Online]. Available: https://github.com/epfl-vlsc/bitfiltrator, 2022. [Accessed 2025-04-14].

[22] Xilinx Kirk Saban. Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency. https://docs.xilinx.com/v/u/en-US/wp380_Stacked_Silicon_Interconnect_Technology, 2012. [Accessed 2025-04-14].

[23] Simon Klix, Nils Albartus, Julian Speith, Paul Staat, Alice Verstege, Annika Wilde, Daniel Lammers, Jörn Langheinrich, Christian Kison, Sebastian Sester-Wehle, Daniel E. Holcomb, and Christof Paar. Stealing maggie's secrets-on the challenges of IP theft through FPGA reverse engineering. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 3391–3405. ACM, 2024.

[24] Tuan La, Khoa Dang Pham, Joseph Powell, and Dirk Koch. Denial-of-service on fpga-based cloud infrastructures - attack and defense. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):441–464, 2021.

[25] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale + fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 13(3):15:1–15:31, 2020.

[26] Jubayer Mahmod and Matthew Hicks. *PhasePrint:* exposing cloud FPGA fingerprints by inducing timing faults at runtime. In Lieven Eeckhout, Georgios Smaragdakis, Katai Liang, Adrian Sampson, Martha A. Kim, and Christopher J. Rossbach, editors, *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2025, Rotterdam, Netherlands, 30 March 2025 - 3 April 2025*, pages 831–844. ACM, 2025.

[27] Kristiyan Manev, Joseph Powell, Kaspar Matas, and Dirk Koch. byteman: A bitstream manipulation framework. In *International Conference on Field-Programmable Technology, (IC)FPT 2022, Hong Kong, December 5-9, 2022*, pages 1–9, Hong Kong, 2022. IEEE.

[28] Microsoft. Azure FPGA Attestation Service - Azure Virtual Machines — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/virtual-machines/field-programmable-gate-arrays-attestation, 2024. [Accessed 2025-04-07].

[29] Shayan Moini, Shanquan Tian, Daniel E. Holcomb, Jakub Szefer, and Russell Tessier. Remote power side-channel attacks on BNN accelerators in fpgas. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 1639–1644, Grenoble, France, 2021. IEEE.

[30] George Provelengios, Daniel E. Holcomb, and Russell Tessier. Power distribution attacks in multitenant fpgas. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(12):2685–2698, 2020.

[31] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant FPGA. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1919–1936, Virtual, 2021. USENIX Association.

[32] Amazon Web Services. Amazon web services amazon ec2 f2 instances. https://aws.amazon.com/ec2/instance-types/f2/. [Accessed 2025-04-02].

[33] Amazon Web Services. Official repository of the AWS EC2 FPGA Hardware and Software Development Kit. [Online]. Available: https://github.com/aws/aws-fpga, 2016. [Accessed 2025-04-07].

[34] Amazon Web Services. Now available – second-generation fpga-powered amazon ec2 instances (f2) | aws news blog. [Online]. Available: https://aws.amazon.com/de/blogs/aws/now-available-second-generation-fpga-powered-amazon-ec2-instances-f2/, 12 2024. [Accessed 2025-04-13].

[35] Arthur Shi, Manuel Haeussermann, CChin, and Carsten Frauenheim. Apple vision pro chip id. https://www.ifixit.com/Guide/Apple+Vision+Pro+Chip+ID/169813, 2024. [Accessed 2025-04-14].

[36] Mirjana Stojilovic, Kasper Rasmussen, Francesco Regazzoni, Mehdi B. Tahoori, and Russell Tessier. A visionary look at the security of reconfigurable cloud computing. *Proc. IEEE*, 111(12):1548–1571, 2023.

[37] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. Fingerprinting cloud fpga infrastructures. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 58–64, New York, NY, USA, 2020. Association for Computing Machinery.

[38] Raúl Gracia Tinedo, Marc Sánchez Artigas, and Pedro García López. Cloud-as-a-gift: Effectively exploiting personal cloud free accounts via REST apis. In *2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, June 28 - July 3, 2013*, pages 621–628, Santa Clara, CA, USA, 2013. IEEE Computer Society.

[39] Stephen Trimberger and Jason Moore. FPGA security: Motivations, features, and applications. *Proc. IEEE*, 102(8):1248–1265, 2014.

[40] Steve Trimberger and Steve McNeil. Security of fpgas in data centers. In *IEEE 2nd International Verification and Security Workshop, IVSW 2017, Thessaloniki, Greece, July 3-5, 2017*, pages 117–122, Thessaloniki, Greece, 2017. IEEE.

[41] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael M. Swift. A placement vulnerability study in multi-tenant public clouds. *CoRR*, abs/1507.03114:913–928, 2015.

[42] Xilinx. Vivado design suite user guide: Dynamic function exchange (ug909). https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration, 2024. [Accessed 2025-04-14].

[43] Xilinx. Ultrascale architecture configuration user guide (ug570). https://docs.amd.com/r/en-US/ug570-ultrascale-configuration, 2025. [Accessed 2025-04-14].

[44] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. Stealing neural network structure through remote FPGA side-channel analysis. *IEEE Trans. Inf. Forensics Secur.*, 16:4377–4388, 2021.

[45] Mark Zhao and G. Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 229–244, San Francisco, CA, USA, 2018. IEEE Computer Society.

[46] Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. Sensing nanosecond-scale voltage attacks and natural transients in fpgas. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, page 101–104, New York, NY, USA, 2013. Association for Computing Machinery.

A Frame Manipulation Bitstream

```
AA995566 # SYNC WORD
20000000 # NOP
30008001 # WRITE TO CMD
00000007 # RCRC
20000000 # NOP [2x]
30018001 # WRITE TO IDCODE
04B24093 # IDCODE
30008001 # WRITE TO CMD
00000000 # NULL
3000C001 # WRITE TO MASK
00000500 # MASK
3000A001 # WRITE TO CTL0
00000400 # CONFIGFALLBACK + GLUTMASK_B
3000C001 # WRITE TO MASK
00020000 # MASK
30030001 # WRITE TO CTL1
00020000 # BIT 17 (not documented)
30002001 # WRITE TO FAR
00052E05 # FRAME ADDRESS
30008001 # WRITE TO CMD
00000001 # WCFG
20000000 # NOP
300040BA # WRITE TO FDRI (186 WORDS)
XXXXXXXX # FRAME WORD [93x]
00000000 # DUMMY WORD [93x]
20000000 # NOP [2x]
30008001 # WRITE TO CMD
0000000A # GRESTORE
20000000 # NOP [2x]
30008001 # WRITE TO CMD
00000003 # DGHIGH/LFRM
20000000 # NOP [20x]
30008001 # WRITE TO CMD
00000005 # START
20000000 # NOP [8x]
30008001 # WRITE TO CMD
00000007 # RCRC
20000000 # NOP [2x]
30008001 # WRITE TO CMD
0000000D # DESNYC
20000000 # NOP [3x]
```

Listing 1: Example configuration bitstream for AMD UltraScale(+) FPGAs in hexadecimal notation. This bitstream configures one frame by writing 93 32-bit data words followed by 93 mandatory dummy words to the specified frame address. The write of frame data is embedded between all necessary configuration commands to process the write and resume operation. The commands are described in [43].