SecBoot - Lightweight secure boot mechanism for Linux-based embedded systems on FPGAs

Peter Rouget §¶, Benoît Badrignans §, Pascal Benoit ¶, Lionel Torres ¶
¶ LIRMM UMR5506, University of Montpellier, 161 rue Ada, 34095 Montpellier, France
§ Seclab Fr, 40 av. Théroigne de Méricourt, 34000 Montpellier, France

***Abstract*—In recent years, the need in security for embedded devices and data centers has increased sharply. The possible consequences of attacks on these equipments make them privileged targets. In these fields, FPGA are increasingly used because of their flexibility and constantly decreasing power consumption and cost: they can embed several hard/soft processors running Linux enhancing system integration. This paper discusses the security issues related to operating system boot security on FPGAs. We show how the software early boot stages can be protected using FPGA built-in security mechanisms and user logic. We consider that external memories can be tampered by software attacks or board level attacks. By using open source elements and standard tools, we present and implement a lightweight solution. We show that the dynamic reconfiguration has nearly no impact on usable resources of the FPGA matrix at the end of the boot process.**

I. Introduction

In past years, the usage of embedded devices by industrials has increased a lot. Among circuits used for embedded systems, FPGAs became popular thanks to their flexibility and constantly decreasing power consumption and cost. In this context, security is a major challenge, as malfunction of industrial target might lead to heavy repercussions (human and financial). Furthermore [1] has shown that security has become a concern for embedded devices, particularly since their proximity to the user, they offer easy physical access. This paper discusses the issues related to operating system boot security on FPGAs. Literature addresses authenticity, integrity and confidentiality of boot sequence elements but most of the proposed solutions are expensive and complex: they generally require user resources, the modification of the configuration logic, or supplementary devices (Crypto Processor, Trusted Platform Module) to operate. In this paper we show how to protect software early boot stages using FPGA security mechanisms and configurable logic in a cost effective way.

The remainder of the paper is organized as follows:

Chapter II presents the threat model and addresses an overview of existing solutions proposed in the literature to secure the bitstream and the operating system. Chapter III exposes our contribution and the results of our implementation are displayed in Chapter IV. Chapter V discusses our approach against prior works and the expected behavior facing an example of attack. Finally, a conclusion with suggestions for future research is provided in Chapter VI.

II. Problematic

A. Threat model

The boot process of an FPGA-based embedded system starts with a step specific to this type of programmable circuit, the loading of the bitstream. It is loaded to the configuration logic from an external memory, generally a flash memory. Then a bootloader handles the configuration of physical resources (clocks setting, serial interfaces, network interfaces, …). Finally the operating system (OS) starts the boot process: for Linux, the kernel allocates resources (memories, cache, …) and loads the rootfs. All the software elements are loaded from an external memory to the RAM where they are executed.

The update of the bitstream or software elements can be done remotely through the network or locally with a physical access. In both cases, an attacker can either send malicious elements, or during a remote update, he can intercept and modify the data. The flash storage on the device is a vulnerable location: a malicious entity might replace or alter memories content through logical or physical access. For example, if at runtime an attacker modifies the content of the Flash memory containing kernel or rootfs, he is then able to install a backdoor that can persist after reboot. Even the RAM or the communications between parts of the system can be probed to retrieve or alter informations.

Our threat model (Fig.1) considers that the FPGA is secured. We use the bitstream encryption and we check the freshness with the manufacturer mechanisms, if available, otherwise the solution described in [9] and [8] can be employed. Physical and side channel attacks are not considered in the scope of this paper. The update of the bitstream is done by a certified entity and in a secured environment. The security of the external RAM is not considered at runtime but approaches based on Merkel-tree [23] concept or through encryption and checksum [24] may be considered. We mainly address the security of the kernel and other boot elements stored in the external flash.

978-1-5386-3344-1/17/$31.00 ©2017 IEEE

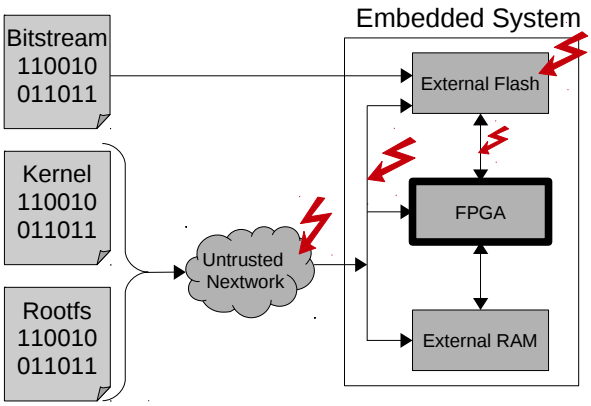

Fig 1: Threat model

B. Related Works

Compared to software only programmable systems, the reconfiguration capabilities offered by FPGAs present additional risks. FPGA manufacturers (Altera [2], Microsemi [3], Xillinx [4]) offer possibilities such as encrypting the bitstream generated with their tools. A dedicated hardware engine handles the decryption of the firmware on the device. [5] proposed to add a CRC check for the bitstream to ensure the integrity and [3] achieves the same purpose with their AES-based MAC. When the bitstream check fails, [6] and [7] propose to fall back on a golden firmware stored in a dedicated read-only flash memory within the FPGA. To ensure the freshness of the bitstream and to prevent replay attacks, [7] offers the possibly to add a version number which is compared and incremented at each update. [8] and [9] suggest a similar approach by adding TAGS to the bitstream. With the SecReCon architecture, [10] submits a method based on a Root of Trust (RoT) and requiring a trusted authority, but the implementation is heavy. Finally to address the problematic of authenticity [8] introduces a process of challenge response between the FPGA and the programmer.

The growing transistors integration enables FPGA to embed processor to run software elements. Their integrity and authenticity are fundamental to ensure the security of the embedded device, and to prevent advanced persistent threats. By using Hash algorithm [11] and [12] propose to verify the integrity of the kernel loaded on the processor. Then signing the hash and verifying it on the device allows ensuring the authenticity of the kernel like described in [13], [14] and [9]. [16] uses a similar mechanism to check the kernel and performs additional memory measurement to ensure that the binary Image matches the expected size and memory location. To allow each boot stage to verify the next one, [15] offers a boot chain using a Trusted Platform Module (TPM) to ensure the authenticity and integrity from the lowest elements to the kernel.

III. Contribution

Embedded systems require highly optimized designs, and to achieve the desired security level, this contribution aims at developing a new method that strikes the balance between both area and boot sequence integrity.

A. General overview

We extend the approach in [9] using Open source tools to perform the kernel verification and taking advantage of FPGA dynamic reconfiguration as proposed in [20] to reduce the resources overhead.

B. Embedded early boot stage

Rather than checking each element of the boot chain, all the vulnerable elements are embedded in the bitstream. In the presently proposed boot sequence, the first and second bootloader stages are stored in read-only BRAMs located in the user logic of the matrix. This allows them taking advantage of security features made available by FPGA vendors.

For FPGA's embedded CPU, booting on an embedded bootloader is generally natively supported. The first bootloader needs to be modified to get the correct memory mapping of the second bootloader. Uboot (Universal Bootloader) is an open source bootloader which natively supports kernel verification. It needs an Image Tree Source (ITS) file, which contains the information of the kernel images (path, compression, Image format, load address, …) that will be verified, and the algorithms chosen for hash and sign. This file is compiled with a Device Tree Compiler and with the given set of private keys to generate a Flattened Image Tree (FIT) file. It contains the binary of all the given images and their signature. Then the bootloader is compiled to embed public keys. By default Uboot supports up to RSA 2048 for signature and SHA 256 for hashing, but to harden the signature and improve the security level support for RSA 4096 and SHA 512 is added. The generated FIT file is stored in the external flash in place of the Kernel. The reader may refer to [17] for further precision about Uboot verification feature. Finally to set up a secure boot chain from FPGA to OS, the support for signature verification to the initramfs [18] embedded in the kernel is added. It ensures that the rootfs is authentic and not corrupted by verifying its GNU Private Guard (GPG) [19] signature. This verification is not in the scope of this paper as the focus is on FPGA to kernel security.

C. Reconfiguration

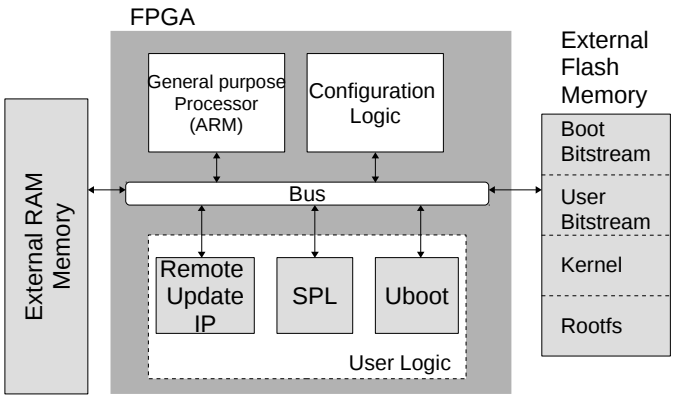

Fig 2: FPGA boot elements locations

978-1-5386-3344-1/17/$31.00 ©2017 IEEE

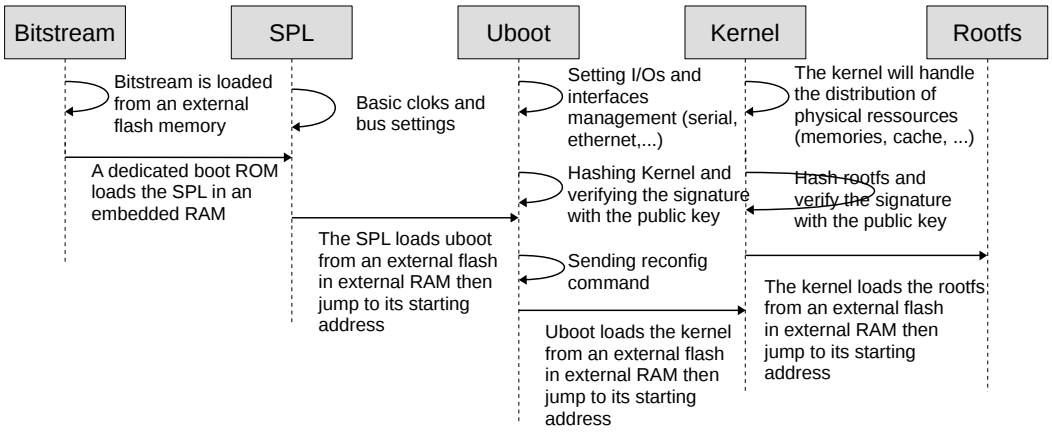

Fig 3: Boot chain

Embedding early boot elements in the FPGA matrix consumes lots of RAM blocks. It reserves a part of the resources normally made available to the user and thus limits the possibility of implementation for the rest of the design. To avoid this overhead we take advantage of the reconfiguration capabilities of the FPGA. We use two bitstreams, the first (Boot bitstream) with the RAM blocks for boot and the second (User bitstream) without these RAMs. Fig.2 depicts the system overview with the location of the boot elements. On FPGA's power up the first bitstream is loaded from the default address. During the boot process after kernel verification we reconfigure the FPGA matrix with the second bitstream. The reconfiguration is triggered by Uboot through a gpio. A dedicated IP access FPGA's configuration register, rewrites the bitstream source address and then launch the reloading (on an Intel FPGA this IP is the Remote Update, on a Xilinx it is the Partial Reconfiguration Controller). The reconfiguration only impacts the user logic, the embedded processor will pursue system's boot during this time. The IP used for dynamic reconfiguration must be present in both bitstreams. In case of a shutdown request from the processor, we want to rewrite the default address in the FPGA configuration registers before shutdown to be able to load the first bitstream on the next power-up. Fig.3 summarizes the new boot sequence with the actions of each element. Another way to freed the Ram blocks used for the boot is to remap them, to make temp re-usable by other design elements. This involves making these memories writable and raises a security question. Moreover the feasibility depends entirely on the design and the FPGA. For example, on a small FPGA, the size of the memory that embarks the Uboot imposes heavy constraints over the timing of the design and greatly complicates the addition of other IPs. On the contrary, on larger FPGA, the constraints imposed by this memory are negligible and remapping is a feasible option.

IV. Experiment

To demonstrate the occupied resources and time cost of the proposed boot chain we use an Intel Cyclone V FPGA with an EPCQ memory for bitstream storage and an EMMC memory for kernel and rootfs storage. The simple preloader (SPL), based on Uboot, is generated through Intel tools kit SoC EDS, for the second bootlloader we use Uboot as mentioned previously. For the dynamic reconfiguration of the FPGA we use an Intel IP, the Remote Update (RU), coupled with a custom state machine.

The FPGA resources consumption of each element needed to implement the proposed boot chain is displayed in Table 1. At the end of the boot, when the OS is operating, only the RU and his state machine remain on the matrix. For the FPGA used in the experiment the resource cost represents a bit more than 0,3% of the available LUTs which may be considered negligible. In comparison with prior work which relies on the use of FPGA configuration logic in order to perform security check over the boot chain, this work still has an overhead on the user logic. On the contrary, compared to asymmetric cryptography and hardware acceleration, the resources cost is lowered, we move the resource overhead from the FPGA to the flash, as we need to store 2 bitstreams instead of one. But nowadays flash memories are large and cheap, much less expensive than FPGA resources.

The boot time of the overall system from FPGA to rootfs, using a light kernel, takes several seconds. The verification of a RSA 4096 signature takes at most 1 or 2 milli-seconds which is negligible in comparison. Furthermore the FPGA reconfiguration, in addition to being extremely fast, is done in discrete time while the kernel is booting so it will not impact the boot delay.

978-1-5386-3344-1/17/$31.00 ©2017 IEEE

TABLE I. RESOURCES USAGE

		Block Memory Bits	M10Ks	Registers	LUTs
Before Reconfiguration	SPL	224,352	28	3	30
	Uboot	1,349,960	168	6	213
	RU*	0	0	74	125
	Total	1,574,312	196	83	368
After Reconfiguration	RU*	0	0	74	125

(*) - Remote Update IP and control state machine

V. DISCUSSION

Even if the experimentation was carried using an Intel FPGA, it remains applicable with any manufacturer supporting dynamic reconfiguration. For example on a Xilinx target: [21] and [22] shows how to embark the early boot stages. Furthermore by using the Partial Reconfiguration Controller IP it should be possible to achieve a similar work as this contribution. RAM blocks are basic IP and are embedded in all FPGA families. The use of an Open source software allows a free customization of boot elements. All these points allow a great flexibility in the choices of the system elements. This contribution is focused on reducing the resources reserved by security mechanism as describes in [9]. Using FPGA reconfiguration optimize the area used by the boot elements to a barely negligible level. It does not require any supplementary device or processor to perform security mechanism.

The Kernel integrity and authenticity is checked by the Uboot. If an attacker manages to modify the flash content and alter the Kernel, the signature will differ and Uboot will stop the boot process. At this moment the circuit is locked and wait for a reboot or a physical intervention to update the kernel. We can imagine falling back on a golden Kernel stored in a Read Only memory to perform security checks while waiting for an intervention. In case of a replay-attack, an attacker could be able to retrieve an older version in order to downgrade the Kernel. Uboot verification will end successfully and the boot process will continue on the downgraded Kernel. In order to limit the risk it is necessary to change the key used by Uboot to verify the signature and therefore to update the bitstream. This solution is recommended for security updates, each Kernel version is signed with a different key to prevent downgrade.

All of this depends on the anti-replay protection of the bitstream. An attacker able to load a previous version of the bitstream with the Uboot verification key corresponding to the downgraded Kernel signature will successfully perform a replay-attack on the device. For example with the used Cyclone V FPGA and the previously described threat model we are vulnerable to replay-attacks. It is necessary to use other FPGA families or to implement solutions as suggested in [9] and [8]. Table II contains the criteria discussed above in relation to related works.

TABLE II. RELATED WORKS COMPARISON

	Kernel Integrity (1)	Area optimized (2)	Flexible (3)	FPGA oriented (4)	Anti-replay (5)
Discretix 2006 [11]	Yes	Yes	Yes	No	Yes
Atmel 2006 (12]	Yes	Yes	No	No	No
ARM 2009 [14]	Yes	No	Yes	No	No
Apple 2009 [13]	Yes	Yes	Yes	No	No
Devic 2011 [9]	Yes	No	Yes	Yes	Yes
Kai et al. 2012 [16]	Yes	Yes	No	No	Yes
Microsemi 2014 [15]	Yes	Yes	No	Yes	No
SecBoot 2017	Yes	Yes	Yes	Yes	FPGA dependent

(1) - Protect against unauthorized alteration of data.
(2) - Once the boot is done, is there some user resources (memory space, FPGA logic, CPU bandwidth) reserved and unusable because of a boot element.
(3) - Not linked to a specific physical platform, a manufacturer or a specific tool. Does it allow different cryptographic algorithms, adding or deleting boot stages.
(4) - Already implemented or usable on FPGA.
(5) - Ensure that an old version of a boot element can not be maliciously reloaded on device.

VI. CONCLUSION

Secure boot is fundamental to ensure the security of embedded devices. In this paper,we have presented a quick and easy way to secure a Kernel by capitalizing on already existing FPGA security, using Uboot. Thanks to the reconfiguration capabilities we achieved nearly negligible resources and boot time overheads. The presented solution is compliant with other security features as the protection over early boot stage increase with the security of the bitstream. It allows building a stronger boot chain, with reduced development time over boot security.

REFERENCES

[1] Paul Kocher, Ruby Lee, Gary McGraw and Anand Raghunathan, “Security as a new dimension in embedded system design”, Proceedings of the 41st *Design Automation Conference*, 2004, New York, USA, DAC ’04, pp 753–760.

[2] Altera Corporation, *White Paper - Design Security in Stratix III Devices*, 2009. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01010.pdf

[3] Microsemi Corporation, *Introduction to Implementing Design Security with Microsemi SmartFusion2 and IGLOO2 FPGAs*, 2013. Available: https://www.microsemi.com/document-portal/doc_view/132860-introduction-to-implementing-design-security-with-microsemi-smartfusion2-and-igloo2-fpgas

[4] Kyle Wilkinson, “Using Encryption to Secure a 7 Series FPGA Bitstream”, in *Application Note: 7 Series FPGAs*, 2015, Xilinx.

978-1-5386-3344-1/17/$31.00 ©2017 IEEE

[5] Kirti Chawla and Taniya Siddiqua, *Mutation Resistant Runtime Code using Kernel Attestation*, 2009, OSSEC project.

[6] Saar Drimer and Markus Kuhn, “A Protocol for Secure Remote Updates of FPGA Configurations”, in *Reconfigurable Computing: Architectures, Tools and Applications*, 2009, volume 5453 of Lecture Notes in Computer Science, pages 50–61.

[7] Lattice corporation, *LatticeXP2 Family Handbook*, January 2012. Available: https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwih5ZOc-dPUAhURIlAKHQfGB9gQFggjMAA&url=http%3A%2F%2Fwww.latticesemi.com%2F~%2Fmedia%2FLatticeSemi%2FDocuments%2FHandbooks%2FLatticeXP2FamilyHandbook.PDF%3Fdocument_id%3D24315&usg=AFQjCNEbKohiQ6bJDkCOLbrBXL_bE9jw4g

[8] Benoît Badrignans, *Using FPGAs for security-sensitive applications*, PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, December 2009

[9] Florian Devic, *Securing embedded systems based on FPGA technologies*, Université Montpellier II - Sciences et Techniques du Languedoc, June 2012

[10] Krzysztof Kepa, Fearghal Morgan, Krzysztof Kosciuszkiewicz and Tomasz Surmacz, “SeReCon: a secure reconfiguration controller for self-reconfigurable systems”, in *Comput.-Based Syst*, February 2010, vol. 1, pp. 86–103.

[11] Discretix corporation, *Discretix Secure Boot (DxSB)*. Available: http://www.discretix.com/secureboot/index.html.

[12] Atmel corporation, *Safe and Secure Bootloader Implementation*, December 2006. Available: http://www.atmel.com/dyn/resources/prod_documents/doc6253.pdf.

[13] Joshua de Cesare, October 2009, US 2009/0257595, Apple corporation. Available: http://www.freepatentsonline.com/20090257595.pdf.

[14] ARM corporation, *Building a Secure System using TrustZone Technology*, April 2009. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[15] Microsemi Corporation, *Microsemi secure boot reference design – White Paper*, June 2014. Available: https://www.microsemi.com/document-portal/doc_view/133604-microsemi-secure-boot-reference-design-white-paper.

[16] Tang Kai, Xu Xin, Chunxia Guo, “The Secure Boot of Embedded System Based on Mobile Trusted Module”, in *Intelligent System Design and Engineering Application*, January 2012, Hainan, China, ISDEA.

[17] Marek Vaˇsut, *Secure and flexible boot with U-Boot bootloader*, October 2014. Available: http://events.linuxfoundation.org/sites/events/files/slides/elce-2014.pdf

[18] Gentoo Fundation, *Initramfs/Guide*, 2016. Available: https://wiki.gentoo.org/wiki/Initramfs/Guide/fr

[19] *The GNU Privacy* Guard, 2015. Available: https://www.gnupg.org/

[20] L. Bossuet, G. Gogniat, W. Burleson, “Dynamically Configurable Security for SRAM FPGA Bitstreams”, in *International Journal of Embedded Systems*, 2016, pp.73-83.

[21] *Secure Boot with SecBus.* Available: https://secbus.telecom-paristech.fr/wiki/SecureBoot

[22] JagannadhaSutradharudu Teki, “U-Boot: Verified RSA – Boot on ARM target”, in *U-boot Mini Summit*, Edinburgh, October 2013.

[23] Reouven Elbaz, Lionel Torres, Gilles Sassatelli, Pierre Guillemin, Michel Bardouillet and Albert Martinez, “A parallelized way to provide data encryption and integrity checking on a processor-memory bus”, in *Proceedings of the 43rd annual Design Automation Conference*, New York, USA, 2006, DAC, pp. 506–509.

[24] Jérémie Crenne, Romain Vaslin, Guy Gogniat, Jean-philippe Diguet, Russell Tessier and Deepak Unnikrishnan, “Configurable Memory Security In Embedded Systems”, in *ACM Transactions on Embedded Computing Systems - Volume 12 Issue 3*, March 2013, pp. 23.

978-1-5386-3344-1/17/$31.00 ©2017 IEEE