ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

Review Article

Open Access

FRARE: FPGA Based Reconfigurable Attack Resilient Engine with Verified Proof-Of-Execution

Avani Dave* and Krunal Dave

USA

ABSTRACT

Modern society is getting accustomed to the Internet of Things (IoT) and Cyber-Physical Systems (CPS) for a variety of applications that involves security-critical user data and information transfers. In the lower end of the spectrum, these devices are resource-constrained with no attack protection. They become a soft target for malicious code modification attacks that steals and misuses device data in malicious activities. The resilient system requires continuous detection, prevention, and/or recovery and correct code execution (including in degraded mode). By end large, existing security primitives (e.g., secure-boot, Remote Attestation RA, Control Flow Attestation (CFA) and Data Flow Attestation (DFA)) focuses on detection and prevention, leaving the proof of code execution and recovery unanswered.

To this end, the proposed work presents lightweight FRARE: FPGA based Reconfigurable Attack Resilient Engine with Ver- ified Proof-of-Execution. It leverages a custom control register (Ctrl_register) based runtime memory modification attacks classification and detection technique. It uses the Proof Of Con- cept (POC) implementation of re-configurable use-case-specific attacks prevention and onboard recovery techniques. The proto- type implementation on Artix 7 Field Programmable Gate Array (FPGA) and state-of-the-art comparison demonstrates very low (2.5%) resource overhead and efficacy of the proposed solution.

***Corresponding author**
Avani Dave and Krunal Dave, USA.

Received: March 05, 2022; **Accepted:** March 15, 2022; **Published:** March 25, 2022

Keywords: Runtime Resilient Soc, Memory Modification attacks Resilient System

Introduction

Industry 4.0 has proliferated the use of connected small Internet of Things (*IoT*) and Cyber-Physical Systems (*CPS*) in applications ranging from home security systems, smart controllers, actuators, sensor nodes, activity trackers, and alarm systems [1]. Often these devices are used for security- critical user data and information transfers. A majority of them are resource-constrained, with no onboard security support, which makes them vulnerable to code modification attacks [2,3]. For example, the Electric Control Unit (*ECU*) of car measures various sensors (e.g., humidity, speed, temperature, speed) and performs different actuation tasks such as speed or heat controls. If an attacker modifies the temperature sensor code to give a low reading, it can overheat the car or damage other parts. Here are few more examples of such attacks [4-6].

The resilience of a system is defined as its ability to detect (including boot-time and continuous runtime) the presence of attacks, prevent adversarial effects and keep the device operational (including in degraded mode) before it can reach a fail-safe or recovery state. Fig 1 shows the resilient system operational timeline. The phases P1 and P5 depict the normal mode of operation. Phase P2 covers attack occurrence andruntime detection. The phases-3 and 4 (P3 & P4) represent the prevention and recovery operations.

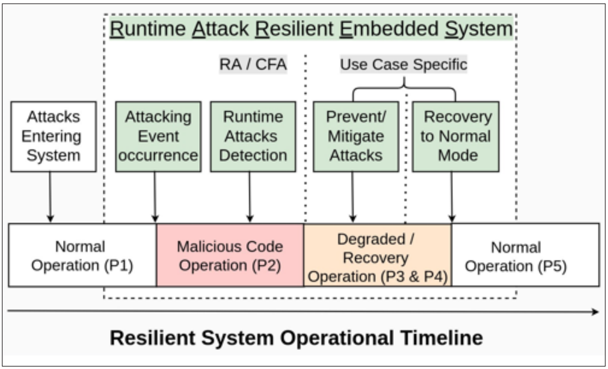
Runtime Attack Resilient Embedded System
RA / CFA
Use Case Specific
Attacks Entering System
Attacking Event occurrence
Runtime Attacks Detection
Prevent/ Mitigate Attacks
Recovery to Normal Mode
Normal Operation (P1)
Malicious Code Operation (P2)
Degraded / Recovery Operation (P3 & P4)
Normal Operation (P5)
Resilient System Operational Timeline

Figure 1: Depicts the resilient system operation flow timeline

From Figure 1, the resilient system requires a secure-boot like boot-time software integrity validation technique before the device enters in phase P1 [7-10]. Recent implementa- tions of *APEX* [11] presents lightweight continuous runtime attacks detection, prevention and verified proof of execution techniques (covering phase 2 and 3 from Fig 1). However, it resets the systems abruptly for attack prevention. Considering the wide applications spectrum of these devices (e.g., aircraft controllers, automotive Electronic Control Unit (*ECU*)), an abrupt system reset can result in adverse effects. These devices requires to operate (including degraded mode) until they can fail-safe or recover completely. Furthermore, they requires use- case-specific prevention and recovery techniques.

To this end, this paper presents *RARES*: a novel lightweight Runtime Attack Resilient Embedded System design using verified proof of execution.

Research Contributions: The design and implementation of RARES presents the following research contributions:

- **Runtime Attacks Classification & Detection:** It clas- sifies runtime memory modification attacks into three categories and presents a novel lightweight 16 bit control register (Ctrl_ register) based detection technique.
- **Prevention Technique:** It demonstrates novel use-case-specific re-configurable runtime attacks prevention tech- niques. It gives the control in the hands of system devel- opers to design use-case-specific prevention and recovery techniques on FPGA.

Related Work

As shown in Figure 1, the resilient system design involves various phases of detection, prevention, and recovery. Un- fortunately, RARES was unable to find a single state-of- art implementation supporting all of these. Therefore, we have studied and analyzed the state-of-the-art works in three main categories: 1) detection, 2) prevention, and 3) recovery techniques.

Detection Techniques: Remote Attestation (RA) is widely used client-server based security primitive that per- forms integrity verification of software state of the un-trusted prover device upon request from third party trusted verifier. Previous implementations of hardware-based, software-based and hybrid RAs can detect runtime memory modification attacks periodically. Control Flow Attestation (CFA) and Data Flow Attestation (DFA) techniques are used for continuous runtime attacks detection [12-25].

Prevention Techniques: The resource isolation is well- known technique to prevent / limit the adverse effect of attacks. The hardware-based techniques uses Trusted Platform Module (*TPM*), Arm TrustZone, Trusted Execution Environ- ment (*TEE*), or Physical Memory Protection (*PMP*) to isolate the shared resources and limit the attacking surface. By end large, these are resource-heavy techniques and not suit- able for targeted devices [12-14,26]. Recent lightweight implementation of *VRASED* (formally verified remote attestation) uses custom hardware module to detect different security property based attacks [27]. *APEX* extends *VRASED* to provide verified Proof Of eXecution (*POX*). They both resets the system to prevent the runtime attack. *RARES* advocates the development of use-case-specific prevention or recovery techniques to avoid adverse effects from abrupt system reset. The detailed system design is discussed in subsection III-B [11].

Recovery Techniques: The affected device can be re- covered by Over-The-Air (*OTA*) or manually code re-flash. Recent implementation of Healed [28] presents Merkle Hash Tree (*MHT*) based technique, which requires at least one node in the network to be untampered, and its firmware is used to re-flash the corrupted node. [29] keeps the software receiver-transmitter code in trusted *ROM* for connecting the affected device to a recovery server for re-flash. Recent implementations of *CARE* [30] presents lightweight secure- boot with onboard recovery technique for the system where manual or over-the-air code reflash are not possible. *SRACARE* extends *CARE* by adding secure communication and RA capabilities [26].

In summary, *RA* can only detect periodic runtime attacks and it suffers from *CWE* 367-Time-of-Check-Time-of-Use (TOCTOU) attacks [31]. Both *CFA* and *DFA* bloats the sys- tem memory by storing runtime execution flow logs, which makes them unsuitable for targeted low-end devices. The lightweight runtime attacks detection technique presented by *APEX* provides only one solution of resetting the system for preventing all different attacks. Furthermore, they do not cover the boot-time attacks detection or recovery techniques. In addition, *RARES* have presented lightweight control register based runtime attack detection, application specific prevention technique. The drawback of *RARES* is all the attack scenarios and its remedial flows needs to be known before the asic implementation.

Therefore, current work presents a FPGA based re- configurable recovery engine. Additionally, it presents the lightweight implementation of FPGA based recovery and automatic memory relocation/mapping for providing resiliency to future unknown attacks.

FRARE Overview

This section covers the details about the targeted platform, FRARE based system architecture, design, and operation.

Targeted Platform

The low-end microcontrollers (e.g., Texas Instrument’s MSP430 or Atmel AVR ATMega micro-controllers) are widely used in applications ranging from automotive ECU’s, indus- trial control systems, actuators, aviation, sensors, smart IoT, and Cyber-Physical System (*CPS*). These devices have very low hardware foot print with only a few KB of address and data memories. They do not have sophisticated hardware or OS support to detect and/ or prevent the malware attacks. Therefore, *FRARE* was designed targeting the OpenMSP430 platform as well-maintained open cores implementation of OpenMSP430 [2] was readily available. However, the pro- posed concept of custom control register (Ctrl register) based continuous runtime attacks classification and detection, use- case-specific prevention, and onboard recovery can be applied to other low-end devices such as Atmel AVR ATmega.

FRARE Design

Figure 2 Shows the high-level design architecture of FRARE.

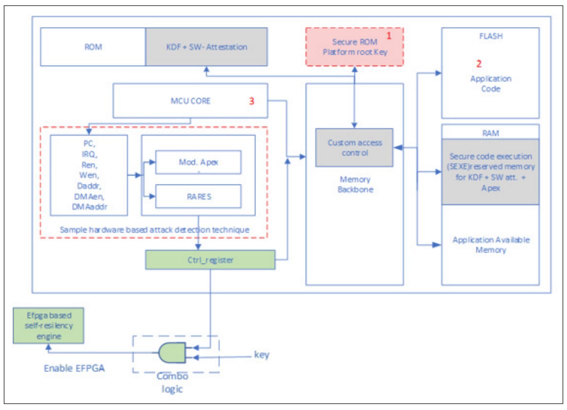
ROM
KDF + SW- Attestation
Secure ROM Platform root Key
1
FLASH
2
Application Code
MCU CORE
3
PC, IRQ, Ren, Wen, Daddr, DMAen, DMAaddr
Mod. Apex
RARES
Sample hardware based attack detection technique
Custom access control
Memory Backbone
RAM
Secure code execution (SEXE)reserved memory for KDF + SW att. + Apex
Application Available Memory
Ctrl_register
Efpga based self-resiliency engine
Enable EFPGA
Combo logic
key

Figure 2: Top-level design of lightweight runtime attacks resilient *FRARE* system. Highlighted are the key components of the proposed system.

FRARE was designed on top of *APEX* and *RARES*. It leverages underlying architecture to provide verified proof-of- execution (*POX*). *FRARE* tapes out the seven control signals

(*Pc, Irq, Ren, Wen, Daddr, DMAen, DMAaddr*) to its custom hardware module (Hw mod). It has carefully designed and modified the internal Finite State Machines (FSMs) of both *VRASED* and *APEX* for detecting different categories of attacks in only one machine clock cycle (mclk), as discussed in subsection §IV. It stores different attack bits in 16-bit control register Ctrl_register, which are discussed in detail in subsection §V. The Ctrl_register does not have high- level write Application Program Interface (*API*) access. The memory backbone acts as arbitration between the front end, *DMA*, and execution-unit interfaces for any system memory (e.g., program, data, and peripheral) accesses, and it is used by *FRARE* for attack prevention. *FRARE* includes separate secure recovery *ROM* and a small FPGA attached to the system to provide configurable mitigation and recovery.

FRARE Operation

Upon power-on the first-stage boot-loader (*FSBL*) code (from *ROM*) gets executed in reserved *RAM* memory and performs the secure-boot verification on flash image. It re- flashes the corrupted flash memory using recovery image upon integrity failure detection, else the system operates normally. FRARE satisfies all the security properties of *APEX* and uses the formally verified software HMC SHA256 code (SW-Att (HACL*) from *ROM* for secure-boot and RA function- ality. After that, the test application code (App Code) from flash gets executed in a specific region of (App. Avail. Mem.) *RAM* as shown in Figure 2 [32]. (Due to the page limitations here, interested readers are requested to refer for *RA* and *POX* operation). *FRARE* performs use-case-specific prevention and recovery operation as discussed in section ?? upon runtime attack detection [11].

Runtime Attacks Classification

Based on the seven control signals (*Pc, Irq, Ren, Wen, Daddr, DMAen, DMAaddr*) input to the custom hardware module and security properties of *APEX*, *FRARE* has classified memory modification attacks in three categories, namely: 1) *CPU* access violation, 2) *DMA* access violation, and 3)

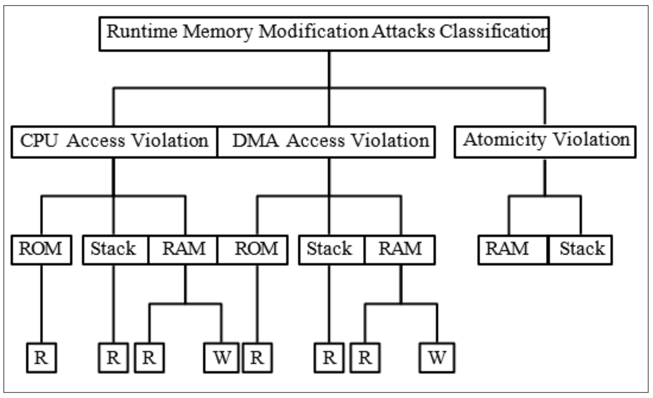

Figure 3: Runtime Memory Modification Attacks Classification

Atomicity violation as shown in Figure 3.

CPU Access Violation: For the system shown in Figure 2, while executing the program code from *RAM* the *CPU* can only read the data from reserved stack and *ROM* (Sw-Att code). However, it cannot access the device's secret key (*K*) from the key *ROM*. Similarly, The key (*K*) is only accessed by the *CPU* while it is executing the (Sw-Att) code inside the reserved stack. All other *ROM* and stack read accesses are detected as *CPU* access violation by the hardware FSM in Ctrl_register. Furthermore, any unauthorized RAM access (both read and write) violation during Sw-Att code execution are detected as *CPU* related *RAM* access violations. This sub-module focuses on (*Pc, Ren, Wen, Daddr*) control signals to detect any unauthorized memory read or write access request by the *CPU*. The corresponding detection bits are updated in Ctrl_register as discussed in subsection V.

DMA Access Violation: During the program code ex-ecution from *RAM*, direct memory access (*DMA*) read re- quest from the reserved stack and *ROM* (Sw-Att code) are allowed. However, *DMA* cannot access the device secret key (K), while executing the program code from *RAM*. Similarly, the *DMA* can access the key (*K*) only during *Sw-Att code* execution inside the reserved stack. All other *ROM* and stack related read accesses are identified as DMA access violation by the hardware FSMs and detected in Ctrl_register. Furthermore, unauthorized RAM access (both read and write) violations while running Sw-Att code are detected under DMA- related RAM access violation. This sub-module focuses on (*Pc, Ren, Wen, DMAen, DMAaddr*) control signals to detect any unauthorized memory read or write access request using *DMA*. The corresponding detection bits are updated in Ctrl_register as discussed in subsection V.

Atomicity Violation: This category detects any interrupt trigger violation during the code execution inside *RAM* and reserved stack (Sw-Att). The atomicity violation usually results in interrupt service routine (IRQ) code execution, intermittent data and secure key (*K*) leakage or loss. This sub-module detects mainly (*Irq*) IRQ during the code execution from the RAM and reserved stack (*Sw-Att*). The *POC* atomicity violation prevention technique is discussed in subsection ??.

Detection Technique

Based on attacks classification of section §IV, specific attack detection bits are updated in 16-bit Ctrl_register as depicted in Fig 4. Note that, at current stage FRARE has classified and detected total ten different types of memory modification attacks and stored them in bit positions D0-D9. The Ctrl_register bit (D0) and (D1) detects atomicity violations during RAM and stack code execution. The *DMA* related *RAM* write access violation is detected by flag bit (D2). The DMA read access violations for *RAM*, stack, and *ROM* are detected in bits (D3) (D4) and (D5), respectively. Similarly, CPU related RAM write access violation is detected by flag bit (D6). CPU read access violations for *RAM,stack*, and *ROM* are detected in bits (D7) (D8) and (D9), respectively.

Table I: State-of-The-Art (Qualitative) Comparison of Lightweight Attack Resilient Systems

Parameters	FRARE	RARES	Ref. [11]	Ref. [18]	Ref. [7]	ref. [22]	Ref. [21]	Ref. [20]	Ref. [26]	Ref. [28]
Design Type	Hybrid	Hybrid	Hybrid	Hybrid	HW	Hybrid	Hybrid	Hybrid	Hybrid	SW
Secure Communication	yes	yes	yes	yes	no	yes	yes	yes	yes	no
Lightweight	yes	yes	yes	yes	no	yes	yes	no	yes	no
Secure boot	yes	yes	no	no	yes	no	no	no	yes	no
Remote Attestation (periodic RA)	yes	yes	yes	yes	no	no	no	no	yes	yes
Runtime Attacks Detection	yes	yes	yes	yes	no	yes	yes	yes	no	no
System Reset for Attacks Prevention	no	no	yes	yes	yes	no	no	no	no	no
Memory Mod. Attacks Prevention	yes	yes	no	no	no	no	no	no	no	no
Recovery Techniques	yes	yes	no	no	no	no	no	no	yes	yes
Reconfigurable eFPGA	yes	no	no	no	no	no	no	no	no	no

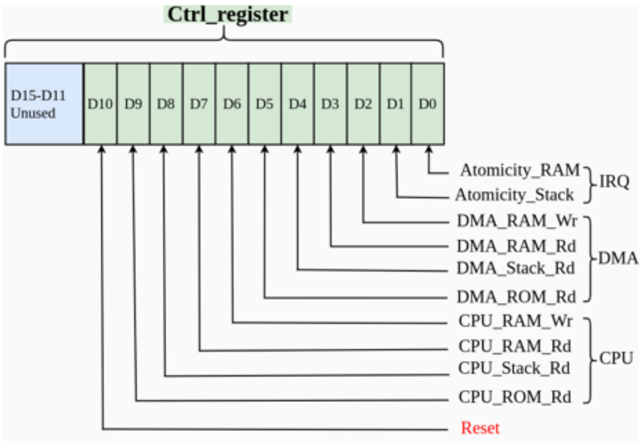

Figure 4: Depicts 16 bit Ctrl_register for each attacks detection by FRARE. Note that only 11-bits of the 16-bit Ctrl_register are used currently, and remaining D11-D15 bits are left for future development.

From this point the system developer can write use-case- specific runtime attack prevention or recovery technique.

Evaluation

This section performs qualitative and quantitative evaluation of FRARE based resilient system. The subsection §VI-A covers the resource utilization (and overheads) for quantitative analysis and subsection §VI-B performs the state-of-the-art comparison for qualitative analysis.

Resource Utilization - Quantitative Comparison

FRARE was implemented on top of *APEX* [11] and com- plete verilog Resistor Transistor Logic (*RTL*) was synthesized on Artix-7 Field Programmable Gate Array (*FPGA*) board using Xilinx Vivado 2018. As shown in Figure 2, the new control register (Ctrl register) was added into *APEX's METADATA* with only read access from the software. Therefore, *FRARE* increases the reserved memory of *APEX* by two bytes to store 16-bit Ctrl_register And the hardware FPGA module was attached to the system with SPI interface. The FPGA was loaded with the recovery image externally. In the even of update neded for never attact this method can be scaled to provide resiliency to the attack via *FPGA*. Table II shows the hardware and memory resource utilization for a FRARE based system and compares it with different state- of-the-art implementations. The baseline Openmsp430 has the lowest hardware resources and requires no reserved memory. *VRASED* uses approximately 2 KB of reserved stack memory for computing the RA (using SW-Att code) and storing results. *APEX* adds nine bytes to store the verified proof of execution. RARES-A extends it further by 2 bytes for storing 16-bit Ctrl_register at runtime.

Table 2: Resource Utilization- Quantitative Comparison

Architecture Details	Hardware Reg.	Resources LUT	Reserved Mem. RAM (bytes)	Verified # LTL # LTL
OpenMSP430 [2]	691	1904	0	-
V RASED [18]	729	1980	2332	10
APEX [11]	755	2290	2341	20
RARES − A	773	2330	2343	20
RARES − B	830	2572	2343	20
FRARE	830	2720	2343	25

FRARE based system and compares it with different state- of-the-art implementations. The baseline Openmsp430 has the lowest hardware resources and requires no reserved memory. VRASED uses approximately 2 KB of reserved stack memory for computing the RA (using SW-Att code) and storing results. *APEX* adds nine bytes to store the verified proof of execution. *RARES-A* extends it further by 2 bytes for storing 16-bit Ctrl_register at runtime.

This work has calculated hardware resource foot- print for two implementations, 1) *RARES-A* with 16-bit Ctrl_register and 2) *RARES-B* which includes the addi- tional onboard recovery *ROM. RARES-A* requires 2.3% more hardware registers and approximately 1.7% more LUT than *APEX. RARES-B* adds the recovery memory (as shown in Fig 2) and it requires 7.37% more hardware registers and approximately 10.3% more LUT than RARES-A (for 16KB ROM). *FRARE* maintains all twenty formal *LTL* specification verification of *APEX*.

State-of-the-Art Qualitative Comparison

FRARE was compared with state-of-the-art secure-boot, remote attestation, control flow attestation, and recovery-based resilience systems for qualitative analysis as shown in Table I. RAs provide periodic runtime software state verification. CFA and DFA provides continuous runtime attacks detection [20-22]. However, they are resource-heavy and bloats the system memory by logs storing. The lightweight implementa- tions of *APEX* [11], *VRASED,* and resets the systems to prevent the attacks. Only *FRARE* based system offers lightweight runtime attacks detection, use-case-specific pre- vention, secure-boot and onboard recovery techniques without an abrupt system reset [18,7].

Conclusion

The lightweight attack resilient system design requires runtime memory modification attacks detection, prevention, and/or recovery techniques. *FRARE* demonstrates the first implementation and efficacy of a novel *FPGA* based recovery and attack prevention engine along wirh lightweight control register (Ctrl register) based continuous runtime attacks detec- tion technique. This approach enables the system developers to design use-case-based prevention techniques.

References

1. Kagermann H, Wahlster W, Helbig J (2013) Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry. acatech – National Academy of Science and Engineering, Mu¨nchen, Final Report of the Industrie 4.0 Working Group, 2013 Available: http://forschungsunion.de/pdf/industrie 4 0 final report.pdf
2. Girard O (2009) Openmsp430. https://opencores.org/projects/openmsp430.
3. Furtak A, Bulygin Y, Bazhaniuk O, Loucaides J, Matrosov A et al. (2014) Bios and secure boot attacks uncovered.
4. Vijayan J (2010) Stuxnet renews power grid security concerns https://www.computerworld.com/article/2519574/stuxnet-renews-power-grid-security-concerns.html.
5. Schneider D (2015) Jeep hacking 101. http://spectrum.ieee.org/cars-that- think/transportation/systems/jeep-hacking-101.html.
6. Haj-Yahya J, Wong MM, Pudi V, Bhasin S, Chattopadhyay A (2019) Lightweight secure-boot architecture for risc-v system-on-chip. in 20th International Symposium on Quality Electronic Design (ISQED), March 216-223.
7. NSA Cyber Report (2017) UEFI DEFENSIVE PRACTICES GUIDANCE https://www.nsa.gov/Portals/70/documents/what-we-o/cybersecurity/professional-resources/ctr-uefi-defensive-practices-guidance.
8. Lebedev I, Hogan K, Devadas S (2018) Invited paper: Secure boot and remote attestation in the sanctum processor. in 2018 IEEE 31st Computer Security Foundations Symposium (CSF) 46-60.
9. Wong MM, Haj-Yahya J, Chattopadhyay A (2018) Smarts: secure memory assurance of risc-v trusted soc pp1-8.
10. Nunes IDO, Eldefrawy K, Rattanavipanon N, Tsudik G (2020) APEX: A verified architecture for proofs of execution on remote devices under full software compromise. in 29th USENIX Security Symposium (USENIX Security 20). USENIX Association 771–788.
11. (2010) Trusted Platform Module - Trusted Computing Group. https://trustedcomputinggroup.org/work-groups/trusted-platform-module/.
12. Jiang H, Chang R, Ren L, Dong W (2017) Implementing a arm-based secure boot scheme for the isolated execution environment. in 2017 13th International Conference on Computational Intelligence and Security (CIS) 336-340.
13. Sabt M, Achemlal M, Bouabdallah A (2015) Trusted execution envi- ronment: What it is, and what it is not in 2015 IEEE Trustcom/Big- DataSE/ISPA 1: 57-64.
14. Lee D, Kohlbrenner D, Shinde S, Song DX, Asanovic K (2019) Key- stone: A framework for architecting tees. ArXiv, vol. abs/1907.10119, 2019
15. Chakraborty D, Hanzlik L, Bugiel S (2019) simtpm: User-centric TPM for mobile devices. in 28th USENIX Security Symposium (USENIX Security 19). Santa Clara, CA: USENIX Association, Aug 533-550.
16. Goldman, Ken (2017) IBM-ACS. https://sourceforge.net/p/ibmtpm20acs/ activity/?page=0{\&}limit=100{\#}5cc3737aee24ca5b73320e9c.
17. 8/16-bit atmel xmega b3 microcontroller. http://ww1.microchip.com/downloads/en/DeviceDoc/.
18. Nunes IDO, Eldefrawy K, Rattanavipanon N, Steiner M, Tsudik G et al. (2019) VRASED: A verified hardware/software co-design for remote attestation. in 28th USENIX Security Symposium (USENIX Security 19). Santa Clara,

CA: USENIX Association 1429-1446.

19. Abera T, Asokan N, Davi L, Ekberg JE, Nyman T, et al. (2016) C-flat: Control-flow attestation for embedded systems software. in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '16. New York, NY, USA: Association for Computing Machinery 743-754.
20. Zeitouni S, Dessouky G, Arias O, Sullivan D, Ibrahim A, et al. (2017) Atrium: Runtime attestation resilient under memory attacks in 2017 IEEE/ACM International Conference on Computer- Aided Design (ICCAD) 384-391.
21. Dessouky G, Zeitouni S, Nyman T, Paverd A, Davi L, et al. (2017) Lo-fat: Low-overhead control flow attestation in hardware. in Proceedings of the 54th Annual Design Automation Conference 2017, ser. DAC '17. New York, NY, USA: Association for Computing Machinery https://doi.org/10.1145/3061639.3062276.
22. Dessouky G, Abera T, Ibrahim A, Sadeghi A (2018) Litehax: Lightweight hardware-assisted attestation of program execution. in 2018 IEEE/ACM International Conference on Computer-Aided Design pp 1-8.
23. Castro M, Costa M, Harris T (2006) Securing software by enforcing data-flow integrity. in Proceedings of the 7th Symposium on Operating Systems Design and Implementation, ser. OSDI '06. USA: USENIX Association 147-160.
24. Song C, Lee B, Lu K, Harris WR, Kim T, et al. (2016) Enforcing kernel security invariants with data flow integrity. in NDSS.
25. Rachala AR (2019) Evaluation of hardware-based data flow integrity," https://core.ac.uk/download/pdf/266596061.pdf.
26. Dave A, Banerjee N, Patel C (2020) Sracare: Secure remote attestation with code authentication and resilience engine. in 2020 IEEE Interna- tional Conference on Embedded Software and Systems (ICESS) pp 1-8.
27. Nunes IDO, Eldefrawy K, Rattanavipanon N, Steiner M, Tsudik G (2019) VRASED: A verified hardware/software co-design for remote attestation. in 28th USENIX Security Symposium (USENIX Security 19). Santa Clara, CA: USENIX Association pp1429-1446.
28. Ibrahim A, Sadeghi AR, Tsudik G (2019) Healed: Healing and attestation for low-end embedded devices. in Financial Cryptography.
29. Perito D, Tsudik G (2010) Secure code update for embedded devices via proofs of secure erasure. in ESORICS.
30. Dave A, Banerjee N, Patel C (2020) Care: Lightweight attack resilient secure boot architecture with onboard recovery for risc-v based soc https://arxiv.org/pdf/2101.06300.pdf.
31. Mitre (2006) Cwe-367: Time-of-check time-of-use (toctou) race condition. https://cwe.mitre.org/data/definitions/367.html.
32. Zinzindohoue JK, Bhargavan K, Protzenko J, Beurdouche B (2017) Hacl*: A verified modern cryptographic library. in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '17. New York, NY, USA: Association for Computing Machinery 1789-1806.