Creative Commons Attribution 4.0 International (CC BY 4.0)
https://creativecommons.org/licenses/by/4.0/

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

arXiv:2305.03266v1 [cs.CR] 5 May 2023

RARES: Runtime Attack Resilient Embedded System Design Using Verified Proof-of-Execution.

Avani Dave Nilanjan Banerjee Chintan Patel
Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County

***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 *RARES* - Runtime Attack Resilient Embedded System design using verified Proof-of-Execution. It presents first custom hardware control register (`Ctrl_register`) based runtime memory modification attacks classification and detection technique. It further demonstrates the Proof Of Concept (*POC*) implementation of use-case-specific attacks prevention and onboard recovery techniques. The prototype implementation on Artix 7 Field Programmable Gate Array (*FPGA*) and state-of-the-art comparison demonstrates very low (2.3%) resource overhead and efficacy of the proposed solution.

***Index Terms*—runtime resilient soc, memory modification attacks resilient system**

I. Introduction

Industry 4.0 [1] 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. Often these devices are used for security-critical user data and information transfers. A majority of them are resource-constrained [2], [3], with no onboard security support, which makes them vulnerable to code modification attacks. 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 [4]–[6] of such attacks.

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 and runtime detection. The phases-3 and 4 (P3 & P4) represent the prevention and recovery operations.

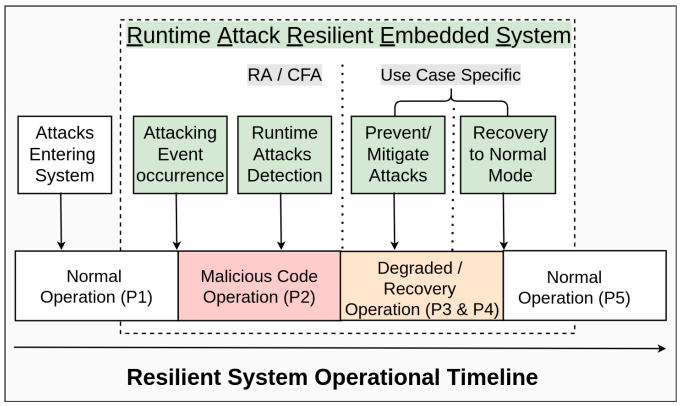

Fig. 1. Depicts the resilient system operation flow timeline.

From Fig 1, the resilient system requires a secure-boot [7]–[10] like boot-time software integrity validation technique before the device enters in phase P1. Recent implementations 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 classifies 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 two novel use-case-specific runtime attacks prevention techniques. It gives the control in the hands of system developers to design use-case-specific prevention and recovery techniques.

- **Secureboot and Recovery Technique:** It presents the lightweight implementation of secure-boot and onboard recovery architecture for the resilient system.

II. Related work

As shown in Fig 1, the resilient system design involves various phases of detection, prevention, and recovery. Unfortunately, *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.

1) Detection Techniques: Remote Attestation (*RA*) is widely used client-server based security primitive that performs integrity verification of software state of the un-trusted prover device upon request from third party trusted verifier. Previous implementations of hardware-based ([12], [13], [14], [15]), software-based ([16], [17]) and hybrid ([10], [18]) *RA*s can detect runtime memory modification attacks periodically. Control Flow Attestation (*CFA*) [19]–[22] and Data Flow Attestation (*DFA*) [23]–[25] techniques are used for continuous runtime attacks detection.

2) 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*) [12], Arm TrustZone [13], Trusted Execution Environment (*TEE*) [14], or Physical Memory Protection (*PMP*) [26] to isolate the shared resources and limit the attacking surface. By end large, these are resource-heavy techniques and not suitable for targeted devices. Recent lightweight implementation of *VRASED* [27] (formally verified remote attestation) uses custom hardware module to detect different security property based attacks. *APEX* [11] 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.

3) Recovery Techniques: The affected device can be recovered 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* [26] extends *CARE* by adding secure communication and *RA* capabilities.

In summary, *RA* can only detect periodic runtime attacks and it suffers from CWE 367-Time-of-Check-Time-of-Use (TOCTOU) [31] attacks. Both *CFA* and *DFA* bloats the system 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.

Therefore, *RARES* proposes the first implementation of the lightweight novel control register (`Ctrl_register`) based runtime attacks detection, application-specific prevention techniques. Additionally, it presents the lightweight implementation of onboard recovery and secure-boot for resilient system design.

III. *RARES* Overview

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

A. 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, industrial 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, *RARES* was designed targeting the OpenMSP430 platform as well-maintained open cores implementation of OpenMSP430 [2] was readily available. However, the proposed 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.

B. RARES Design

Fig 2 shows the high-level design architecture of *RARES*.

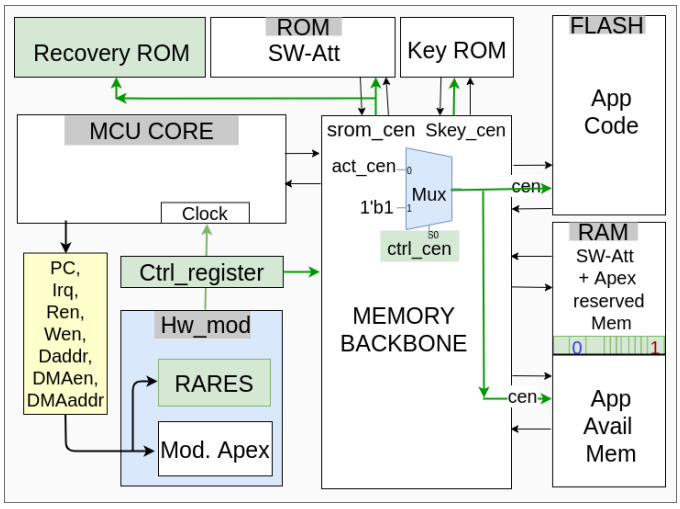

Fig. 2. Top-level design of lightweight runtime attacks resilient *RARES* system. Highlighted are the key components of the proposed system.

RARES was designed on top of *APEX*. It leverages underlying architecture to provide verified proof-of-execution (*POX*).

RARES 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 *RARES* for attack prevention. *RARES* includes separate secure recovery *ROM* for onboard recovery technique implementation as discussed in subsection VI-B.

C. RARES 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. *RARES* satisfies all the security properties of *APEX* and uses the formally verified software HMC_SHA256 code (*SW-Att* (HACL*) [32]) from *ROM* for secure-boot and *RA* functionality. After that, the test application code (App Code) from flash gets executed in a specific region of (App. Avail. Mem.) *RAM* as shown in Fig 2. (Due to the page limitations here, interested readers are requested to refer [11] for *RA* and *POX* operation). *RARES* performs use-case-specific prevention and recovery operation as discussed in section VI upon runtime attack detection.

IV. 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*, *RARES* has classified memory modification attacks in three categories, namely: 1) *CPU* access violation, 2) *DMA* access violation, and 3) Atomicity violation as shown in Fig 3.

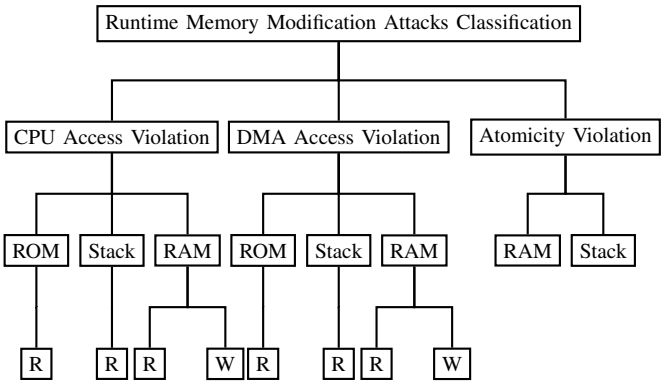

Fig. 3. Runtime Memory Modification Attacks Classification

1) CPU Access Violation: For the system shown in Fig 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.

2) DMA Access Violation: During the program code execution from *RAM*, direct memory access (*DMA*) read request 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.

3) 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 VI-B.

V. 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 *RARES* 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. From this

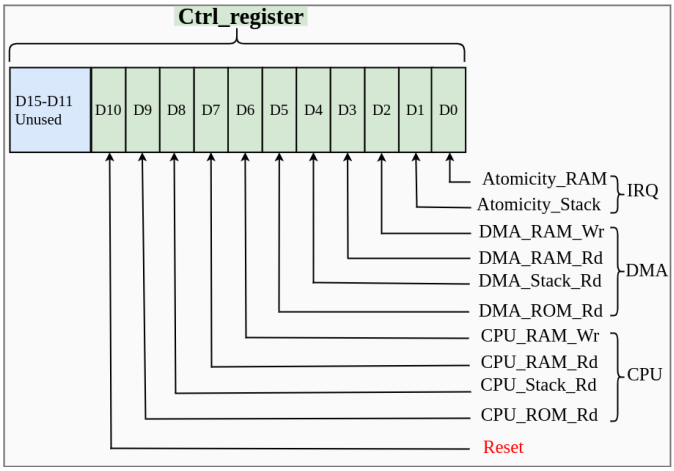

Fig. 4. Depicts 16 bit `Ctrl_register` for each attacks detection by *RARES*. Note that only 11-bits of the 16-bit `Ctrl_register` are used currently, and remaining D11-D15 bits are left for future development.

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

VI. Use-case Specific Prevention Techniques

RARES detects ten types of memory modification attacks in by setting corresponding flag bit high in *Ctrl_register*. It enables the system developer to implement multiple use-case-specific attacks prevention and recovery solutions instead of abrupt system reset like in *APEX*. This section presents two use-case-specific runtime attack prevention techniques, and onboard recovery to demonstrate the efficacy of the proposed solution.

The test application performs *RA* feature for the integrity verification of the flash memory region. It uses hmac_sha256 (*Sw-Att*) for the digest computation. The key K is only accessed by the system during *Sw-Att* code execution. *RARES* has implemented two different attacks and their prevention techniques as follows.

A. ROM Key (K) Read Attack Prevention

RARES has implemented key (*ROM*) read attack, by making *CPU* read key (K) location while executing code from *RAM* memory. It gets detected by bit D9 in Ctrl_register (CPU_ROM_Rd). *RARES* has implemented one hardware based and one software based prevention techniques.

1. Software based Prevention: *RARES* has identified that, the underlying OpenMSP430 micro-controller has six different

```
1  void prev()
2  {
3    __asm__ volatile("bis  #240, r2" "\n\t");
4  }
5
```

Fig. 5. Depicts software-based *RARES* based mode switching technique for attack prevention

Low Power Modes (e.g., $LPM0$, $LPM1$, $LMP2$, $LPM3$, $LPM4$, $LPM5$) for mainly power conservation. The system can be switched to operate in any of the available mode based on the value in r2 register. *RARES* has leveraged this mode switching capabilities of the targeted devices to prevent runtime attacks. It has implemented and validated software-based mode switching upon ROM access violation detected by bit D9. *RARES* was switched to *LPM0* to prevent the read *ROM* attack. Fig 5 shows the code snippet of software-based mode-switching.

2. Hardware based Prevention: *RARES* detects the ROM access violation by setting bit D9 high in *Ctrl_register*. *RARES* has identified that the underlying OpenMSP430 microcontrollers has a hardware pin called *CPUOFF*, which makes the *CPU* goes into the idle state (not off) while keeping mclk (peripherals) and *DMA* ON. For the *POC* of runtime attack prevention, *RARES* has ORed (bitwise logic OR operation) the D9 bit with the *CPUOFF* bit ON selection logic in hardware.

Another use-case for this solution could be, consider a sample application in car *ECU*, where the device is continuously reading data from attached sensor node using *DMA* and *CPU* is performing arithmetic computation on different data. In this case, the runtime *CPU* related *ROM* access violation is detected by *Ctrl_register*. The proposed technique becomes very useful as it keeps the connected peripherals and *DMA* ON, while keeping *CPU* in the idle state.

B. RAM write Access Prevention

To prevent runtime *RAM* access (read/ write) violations, Fig 6 demonstrates the first implementation of hardware-based prevention. This solution was only possible because,

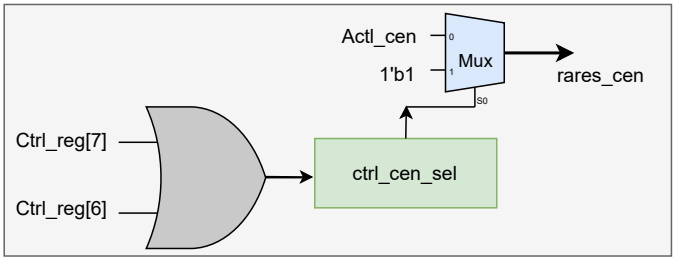

Fig. 6. Depicts unauthorized memory read/write attack prevention technique

RARES was able to identified that underlying OpenMSP430 has individual (active low) chip enable signal for each memory modules (e.g., `srom_cen`, `skey_cen`, `pmem_cen`, `dmem_cen`) to trigger the data transfer, which are controlled by the memory backbone. *RARES* uses the attack detection flag bits as chip enable selection switch. *RARES* performs bitwise OR operation on corresponding memory access violations bits to generate the `ctrl_cen_sel` signal. *RARES* makes the chip enable (chip_en) signal high upon attack detection by `ctrl_cen_sel`. By doing this, *RARES* inserts a wait states in current memory read/write instruction to pause the operation

TABLE I
STATE-OF-THE-ART (QUALITATIVE) COMPARISON OF LIGHTWEIGHT ATTACK RESILIENT SYSTEMS

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

by hardware (this covers phase P3 in Fig 1), while keeping other operations ON. Inserting wait state pauses the unauthorized code execution. However, to bring the device back to normal operation, *RARES* triggers a subroutine call to performs the code reflash using un-tempered (golden) recovery (*ROM*) image. For the atomicity violations *RARES* triggers system reset by enabling D10 reset signal.

Note that, the above prevention and recovery techniques are implemented for *POC* only. The goal of *RARES* design is to demonstrate runtime detection using the 16-bit *Ctrl_register*, and to give system developers an opportunity to design their use-case-specific prevention and recovery solution.

VII. EVALUATION

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

A. Resource Utilization - Quantitative Comparison

RARES was implemented on top of *APEX* [11] and complete verilog Resistor Transistor Logic (*RTL*) was synthesized on Artix-7 Field Programmable Gate Array (*FPGA*) board using Xilinx Vivado 2018. As shown in Fig 2, the new control register (Ctrl_register) was added into *APEX*'s *METADATA* with only read access from the software. Therefore, *RARES* increases the reserved memory of *APEX* by two bytes to store 16-bit `Ctrl_register`. Table II shows the hardware and

TABLE II
RESOURCE UTILIZATION- QUANTITATIVE COMPARISON

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

memory resource utilization for a *RARES* 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 footprint for two implementations, 1) *RARES-A* with 16-bit `Ctrl_register` and 2) *RARES-B* which includes the additional 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*). *RARES* maintains all twenty formal *LTL* specification verification of *APEX*.

B. State-of-the-art Qualitative Comparison

RARES 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. *RA*s provide periodic runtime software state verification. *CFA* and *DFA* ([20]–[22]) provides continuous runtime attacks detection. However, they are resource-heavy and bloats the system memory by logs storing. The lightweight implementations of *APEX* [11], *VRASED* [18], and [7] resets the systems to prevent the attacks. Only *RARES* based system offers lightweight runtime attacks detection, use-case-specific prevention, secure-boot and onboard recovery techniques without an abrupt system reset.

VIII. DISCUSSION

RARES is the first implementation of runtime memory modification attacks detection using *Ctrl_register*. It opens a broad possibilities of use-case-specific attack prevention or recovery-based system design. Since *RARES* is designed on top of *APEX*, all the security properties and formal verification proofs are maintained, along with the adversarial model and limitations. The proposed solution can be generalized and applied to any lightweight micro-controllers by attaching the custom hardware module to them. While porting to the new platform will require the system developer to identify and implement the (new) platform specific suitable prevention and recovery techniques. The solution can be used with different test applications as well by storing the correct recovery image in *ROM*. *RARES* uses onboard recovery to cover broad

application areas where over-the-air or manual recovery is not possible. The system designer can store only the critical section of flash code instead of the full image to reduce the resource utilization in *RARES-B* implementation.

IX. Conclusion

The lightweight attack resilient system design requires runtime memory modification attacks detection, prevention, and/or recovery techniques. *RARES* demonstrates the first implementation and efficacy of a novel lightweight control register (Ctrl_register) based continuous runtime attacks detection technique. This approach enables the system developers to design use-case-based prevention techniques. It further showcases two runtime memory modification attack prevention and onboard recovery techniques. It requires very little resource overhead when compared with the state-of-the-art techniques.

References

[1] H. Kagermann, W. Wahlster, and J. Helbig, "Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry," acatech – National Academy of Science and Engineering, München, Final Report of the Industrie 4.0 Working Group, 2013. [Online]. Available: \url{http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf}

[2] O. Girard, "Openmsp430," https://opencores.org/projects/openmsp430, 2009.

[3] "8/16-bit atmel xmega b3 microcontroller," http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8074-8-and-16-bit-AVR-Microcontroller-XMEGA-B-ATxmega64B3-ATxmega128B3_Datasheet.pdf, 2014.

[4] A. Furtak, Y. Bulygin, O. Bazhaniuk, J. Loucaides, A. Matrosov, and M. Gorobet, "Bios and secure boot attacks uncovered," 2014.

[5] J. Vijayan, "Stuxnet renews power grid security concerns," https://www.computerworld.com/article/2519574/stuxnet-renews-power-grid-security-concerns.html, June 2010.

[6] D. Schneider, "Jeep hacking 101," http://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101.html, 2015.

[7] J. Haj-Yahya, M. M. Wong, V. Pudi, S. Bhasin, and A. Chattopadhyay, "Lightweight secure-boot architecture for risc-v system-on-chip," in *20th International Symposium on Quality Electronic Design (ISQED)*, March 2019, pp. 216–223.

[8] NSA Cyber Report, "UEFI DEFENSIVE PRACTICES GUIDANCE," https://www.nsa.gov/Portals/70/documents/what-we-do/cybersecurity/professional-resources/ctr-uefi-defensive-practices-guidance.pdf, 2017.

[9] I. Lebedev, K. Hogan, and S. Devadas, "Invited paper: Secure boot and remote attestation in the sanctum processor," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, July 2018, pp. 46–60.

[10] M. M. Wong, J. Haj-Yahya, and A. Chattopadhyay, "Smarts: secure memory assurance of risc-v trusted soc," 06 2018, pp. 1–8.

[11] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "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, Aug. 2020, pp. 771–788. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/nunes

[12] "Trusted Platform Module - Trusted Computing Group," https://trustedcomputinggroup.org/work-groups/trusted-platform-module/, 2010.

[13] H. Jiang, R. Chang, L. Ren, and W. Dong, "Implementing a arm-based secure boot scheme for the isolated execution environment," in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, Dec 2017, pp. 336–340.

[14] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug 2015, pp. 57–64.

[15] D. Lee, D. Kohlbrenner, S. Shinde, D. X. Song, and K. Asanovic, "Keystone: A framework for architecting tees," *ArXiv*, vol. abs/1907.10119, 2019.

[16] D. Chakraborty, L. Hanzlik, and S. Bugiel, "simtpm: User-centric TPM for mobile devices," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 533–550. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/chakraborty

[17] Goldman, Ken, "IBM-ACS," https://sourceforge.net/p/ibmtpm20acs/activity/?page=0{\&}limit=100{\#}5cc3737aee24ca5b73320e9c, 2017.

[18] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1429–1446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/de-oliveira-nunes

[19] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "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, 2016, p. 743–754. [Online]. Available: https://doi.org/10.1145/2976749.2978358

[20] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 384–391.

[21] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "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, 2017. [Online]. Available: https://doi.org/10.1145/3061639.3062276

[22] G. Dessouky, T. Abera, A. Ibrahim, and A. Sadeghi, "Litehax: Lightweight hardware-assisted attestation of program execution," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[23] M. Castro, M. Costa, and T. Harris, "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, 2006, p. 147–160.

[24] C. Song, B. Lee, K. Lu, W. R. Harris, T. Kim, and W. Lee, "Enforcing kernel security invariants with data flow integrity." in *NDSS 2016*, 2016.

[25] A. R. RACHALA, "Evaluation of hardware-based data flow integrity," https://core.ac.uk/download/pdf/266596061.pdf, 2019.

[26] A. Dave, N. Banerjee, and C. Patel, "Sracare: Secure remote attestation with code authentication and resilience engine," in *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2020, pp. 1–8.

[27] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1429–1446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/de-oliveira-nunes

[28] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, "Healed: Healing and attestation for low-end embedded devices," in *Financial Cryptography*, 2019.

[29] D. Perito and G. Tsudik, "Secure code update for embedded devices via proofs of secure erasure," in *ESORICS*, 2010.

[30] A. Dave, N. Banerjee, and C. Patel, "Care: Lightweight attack resilient secure boot architecture with onboard recovery for risc-v based soc," https://arxiv.org/pdf/2101.06300.pdf, 2020.

[31] Mitre, "Cwe-367: Time-of-check time-of-use (toctou) race condition," https://cwe.mitre.org/data/definitions/367.html, 2006.

[32] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "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, 2017, p. 1789–1806. [Online]. Available: https://doi.org/10.1145/3133956.3134043