임베디드 기기에서의 소프트웨어 보안 기법 근황

김성수 [1],조규원 [1],김지훈 [1],이호준 [2]
[1] 성균관대학교 소프트웨어학과 대학원생
[2] 성균관대학교 소프트웨어학과 교수

sskim71@skku.edu , kyuwon.cho@skku.edu, rhgus862@g.skku.edu , hojoon.lee@skku.edu

A Survey on Software Defenses for Embedded System

Sung Soo Kim[1], Kyuwon Cho[1], Hojoon Lee[1]
[1]Dept. of Computer Science and Engineering, Sungkyunkwan University

요 약

임베디드 시스템이 급속도로 고도화됨에 따라 임베디드 기기에서의 어플리케이션 또한 공격자에게 넓은 공격 표면으로 작용하게 되었다. 임베디드 기기의 역할이 이전보다 다양해짐에 따라 공격자가 민감한 개인정보를 탈취하거나 온디바이스 AI 모델을 탈취하는 등 임베디드 기기에 대한 공격 가치가 올라가고 있고, 그에 따라 임베디드 기기에 대한 보안 중요도 또한 이전보다 증가하고 있다. 본 연구에서는 임베디드 기기에서의 소프트웨어 보안 기법 연구 동향을 종합적으로 조사하고 분석하여 이들에 유효성을 평가하고 향후 새로운 임베디드 기기에서의 소프트웨어 보안 기법 연구 전략에 대한 제언을 제시하고자 한다.

1. 서론

현재 임베디드 시스템은 단순히 산업제어 환경에서만 쓰이는 것이 아니라 우리 사회 곳곳에 녹아 들어 있다. 헬스케어, 자율주행 등 많은 곳에서 임베디드 시스템을 활용하고 있다. 임베디드 시스템의 활용도가 높아짐에 따라 시스템에 대한 가치가 높아지게 되었다. 예를 들어, 헬스케어 시스템은 민감한 개인정보를 저장하고 있고, 온디바이스 추론 시스템은 AI 모델을 포함하고 있다. 이러한 고가치의 정보들은 악의적인 공격자들에게 임베디드 시스템을 더욱 매력적인 공격 타겟으로 여겨지게 하며, 따라서 기존보다 더 많은 공격 위험에 노출되게 한다.

그러나 임베디드 시스템은 상용 시스템보다 더 많은 제약조건 때문에 기존의 소프트웨어 보안 기법을 그대로 사용할 수 없다. 예시로, 메모리 레이아웃을 랜덤화 하여 공격 난이도를 증가시키는 ASLR(Address Space Layout Randomization)과 같은 기법은 고정된 물리 메모리 맵핑을 가지고 있는 임베디드 시스템에서는 사용할 수 없다. 또한 커널 없이 베어메탈로 동작하는 임베디드 시스템은 커널에 의한 접근 관리를 사용할 수 없다. 이러한 상용 시스템과 임베디드 시스템의 차이는 임베디드 시스템을 위한 소프트웨어 방어 기법에게 새로운 접근 방식을 취하도록 강제한다.

최근 활발히 연구되고 있는 임베디드 시스템의 소프트웨어 방어 기법은 크게 CFI(Control-flow Integrity), data integrity, 구획화(compartmentalization)으로 나눌 수 있다. 본 논문에서는 각 기법들의 대표적인 연구들을 소개하려 한다.

2. 배경지식

임베디드 시스템의 보호 기법들은 기존의 일반적인 컴퓨팅 시스템과의 차이에서 구현방식과 충족요건을 달리한다.

대표적인 것들로 예시를 들자면 첫번째는 메모리 가상화(memory virtualization)의 부재이다. 데스크톱 시스템에서는 메모리 관리 장치가 메모리 가상화를 제공해 프로세스들을 격리해 서로간의 메모리 접근을 불가한다. 반면, 임베디드 시스템에서는 메모리 관리 장치가 존재하지 않아 각 태스크(task)들과 운영체제가 모두 같은 메모리 주소 공간을 공유한다.

두번째는 실시간 시스템의 요구조건이다. 임베디드 시스템은 무인이동체, 헬스케어, IoT 기기 등의 영역에서 time-critical 한 동작들을 수행해야만 한다. 따라서 이러한 기기들을 대상으로 하는 보호기법들에선 데스크톱 시스템보다 성능 오버헤드를 최소화하는 것이 더 강조된다. 또한 대부분의 마이크로컨트롤러(MCU)는 32-bit 아키텍처(architecture)에서 수행되기 때문에 전체 메모리 주소 공간이 4GB 에 불과하고,

사용가능한 플래시와 SRAM 공간은 이보다 훨씬 적다. 따라서 보안기법에 사용되는 메모리 오버헤드도 최소화돼야만 한다.

임베디드 시스템의 보호기법으로는 대표적으로 네 가지가 있다. 첫번째는 Control-Flow Integrity(CFI)이다. CFI 란 프로그램의 실행 흐름이 의도하지 않은 곳으로 가는 것을 막는 기법이다. CFI 에는 forward-edge integrity 와 backward-edge integrity 가 있다. Forward-edge integrity 는 함수 호출이나 indirect branch 같은 indirect control-flow 의 무결성을 확인하는 것이고, backward-edge integrity 는 스택에 있는 return address 의 무결성을 확인하는 것이다.

두번째는 Data Integrity 로 CFI 가 감지하지 못하는 의도된 실행 흐름 안에서의 변질 수 있는 데이터의 무결성을 확인하는 방법이다.

마지막 보호기법은 구획화(Compartmentalization)이다. 구획화란 하나의 프로그램을 여러 개로 분할하고 각 섹션 간의 교류를 통제해 한 섹션이 공격자에게 취약점 공격 등을 통해 오염이 되어도 다른 섹션들에게는 영향이 가지 않도록 하는 방법이다.

3. 대표적인 보안 기법들

1) Silhouette

Silhouette [1]은 shadow memory 를 통해 backward-edge 의 무결성을 검사한다. 네 개의 컴파일러 패스들을 통해 어플리케이션 코드를 변형하는데 먼저 첫번째, 함수들의 프롤로그(prologue)와 에필로그(epilogue)에 shadow stack 으로부터 return address 를 저장하거나 읽어오는 코드를 instrumentation 한다. 두번째, 어플리케이션 코드에 있는 모든 store 명령어들을 ARMv7-M 아키텍처에서 제공하는 동일한 unprivileged instruction 으로 변환함으로써 프롤로그에 추가된 코드 외 다른 코드가 shadow stack 에 쓰는 것을 방지한다. 세번째, 공격자가 forward-edge control-flow hijacking 을 통해 실행 흐름을 마음대로 가져가면 shadow stack 이 무색해지는 것을 막기위해 가능한 branch target 에만 label 을 추가해 coarse-grained 하게 forward-edge integrity 를 검사한다. 마지막으로 어플리케이션 코드를 스캔해 Silhouette 의 설정을 수정할 수 있는 명령어들을 찾아 개발자에게 해당 정보를 전달한다. Silhouette 의 보호 대상은 베어메탈 프로그램이다.

2) Kage

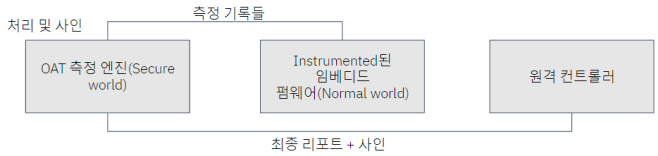
Kage [2]는 Silhouette 에서 더 나아가서 실시간 운영체제(Real-time OS)를 지원한다. Silhouette 에서 사용하는 네가지 컴파일러 패스를 동일하게 사용하는데, 차이점은 커널과 각 태스크마다 shadow stack 을 관리한다는 것이다. 또한 커널과 태스크들 간의 context switch 나 예외처리가 일어날 때의 프로세서 상태도 shadow stack 에 보관하고 Memory Protection Unit(MPU)를 이용해 context switch 이전 태스크 메모리 영역으로의 접근을 제한한다. Silhouette 과 달리 여러 태스크들을 지원한다는 장점이 있지만, 각 태스크 스택의 크기가 제한적이며, 협소한 메모리 공간으로 인해 지원가능한 태스크 수도 많지 않다.

3) SHERLOC

SHERLOC [3]은 트레이싱(tracing)을 기반으로 한 control-flow 위반 감지 기법이다. 베어메탈이나 태스크들이 있는 실시간 운영체제 모두 지원한다. SHERLOC 은 비순차적 프로그램 카운터(PC) 변화들을 기록해 두었다가, 워터마크만큼 기록이 차면 프로그램의 실행을 멈추고 exception 을 발생시킨다. Exception handler 는 기록들을 처리하면서 위반한 사례가 있는지 검사한다. Forward-edge 로는, 바이너리 분석 도구와 동적학습을 통해 indirect branch table(IBT)을 생성해 exception handler 에서 참조한다. Backward-edge 로는, 함수 호출이 있을 때마다 리턴 주소를 reconstructed call stack(RCS)라는 스택에 삽입한다. 실시간 운영체제 위에 실행되는 프로그램의 경우, 커널과 각 태스크마다 RCS 를 관리한다. SHERLOC 은 비동기 인터럽트를 포함한 control-flow 무결성을 제공한다는 점에서 총체적인 솔루션이라고 볼 수 있다. 그러나 기록들을 처리하는 오버헤드가 상당하고, SHERLOC 을 사용하는 임베디드 시스템은 하드웨어 트레이싱 유닛, 데이터 왓치포인트와 트레이스 유닛(DWT), 그리고 신뢰실행환경(TEE) 기능들을 지원해야 한다는 제약이 있다.

4) InsectACIDE

InsectACIDE [4]는 SHERLOC 에 기반해서 제안된 또 하나의 트레이싱 기반 control-flow 위반 감지 기법이다. 트레이스들을 처리하는 알고리즘은 SHERLOC 과 동일하지만, 차이점이 크게 두가지가 있다. 첫번째, InsectACIDE 에서는 기록들을 태스크 간의 idle 시간에 처리하여 오버헤드를 감소시켰다. 페리페럴(peripheral) 아웃풋과 같은 중요한 작업을 해야 하는 경우에는 SHERLOC 과 같이 exception 을 발생시켜 남은 기록들을 처리한다. 두번째, 프로그램 코드안에서 control-flow 무결성과는 무관한 코드 섹션들을 선택해 트레이싱 대상에서 제외한다. 이 또한 임베디드 시스템의 요구사항인 성능 오버헤드 최소화를 위해 InsectACIDE 에서 구현한 최적화 방법이다. 마지막으로 InsectACIDE 는 개별의 control-flow 변화만 보지 않고 일련의 변화들까지 확인하는 context-sensitive CFI 를 적용한다. InsectACIDE 는 SHERLOC 보단 높은 성



(그림 1) OAT 디자인.

능을 보이지만, 여전히 높은 오버헤드를 갖고 있고, 요구되는 하드웨어 기능도 InsectACIDE 를 도입가능한 임베디드 시스템에 제약을 준다.

5) OAT

OAT [5]는 말단 임베디드 기기들(프론트엔드)의 CFI 와 data integrity 여부를 원격 컨트롤러(백엔드)가 직접 검증할 수 있도록 해주는 attestation 기법이다. Attestation 이란 앞선 기법들 같이 위반사례들을 실시간으로 감지하거나 미리 예방하기 보다는 실행이 끝난 뒤에 실행이 무결성 위반 없이 무사히 끝났는지를 확인하는 행위이다. Control-flow 무결성 여부는 조건분기와 비간접 점프나 호출의 경우 조건 성립여부와 메모리 주소를 하나의 비트 스트림으로 만들고 리턴하는 경우 주소를 전 단계의 해시에 추가해 다시 해시하여 비트 스트림과 해시를 결과물로 벡엔드에 보냄으로 확인이 이루어진다. 벡엔드는 이를 참조해 추상적으로 프로그램을 실행하여 위반사례가 있었는지 판단한다. Data integrity 여부는 개발자가 명시한 변수들에 대한 해시맵을 관리해 현재 사용하려는 변수의 값이 변수에 대한 마지막 store 명령어가 변경한 값이랑 동일한지 확인하여 이루어진다. OAT 는 비교적 더 제약적인 기기들에 적용할 수 있을 정도로 가벼운 솔루션이지만, 개발자들이 직접 필요한 변수들을 명시해야 하는 노력이 요구된다.

6) MINION

MINION [6]은 임베디드 프로그램을 구획화하는 기법 중 하나이다. MINION 은 스레드 단위로 펌웨어를 나누어 스레드 간의 context switch 가 일어날 때마다 MPU 의 설정을 바꾸어 해당 스레드가 필요한 메모리 영역만 접근할 수 있도록 한다. 다만, MPU 가 한번에 지원하는 메모리 영역은 대부분 8 개이하로 제한적이기 때문에, 필요한 메모리 영역이 그 이상이면 더한만큼 영역을 합친다. 이때, k-means clustering 알고리즘을 이용해 영역을 합칠 때 생기는 잉여 공간을 최소화한다. MINION 은 실시간 운영체제가 MPU 설정에 접근하는 것과 스레드 간의 mode switch 오버헤드를 최소화하기 위해 운영체제와 스레드들 모두 unprivileged mode 에 두고 참조 모니터를 privileged mode 에 두어 context switch 마다의 MPU 설정변경을 수행한다.

7) CRT-C

CRT-C [7]는 컴파일 타임 구획화 기법이다. 기존의 구획화 방식들은 런타임에 실행되는 코드에 따라 메모리 접근가능 영역이 바뀌는 반면, CRT-C 는 컴파일 타임에 주어진 규칙을 펌웨어가 따르도록 강제함으로써 페리페럴, 커널, 스레드들 간의 구획화를 구현한다. 따라서 CRT-C 는 적용하기 위해 필요한 하드웨어 기능이 없고 오버헤드도 적다. 하지만, 커널 전체를 TCB 로 간주하기 때문에 커널이 가지고 있는 모든 취약점을 그대로 지니고 있다.

8) EC

EC [8]는 MINION 과 비슷한 MPU 를 이용한 구획화 기법을 사용한다. CRT-C 의 큰 TCB 문제를 해결하기 위해 TCB 를 참조 모니터 역할을 하는 마이크로 커널로만 하고 실시간 운영체제와 펌웨어와 같이 privileged mode 에 둔다. 같은 privileged mode 에 존재하기 때문에 펌웨어나 실시간 운영체제에서 마이크로 커널과 MPU 설정에 접근이 가능하다. 따라서 EC 에서는 DWT 를 이용해 관련 메모리로의 접근을 방지한다. EC 는 모든 코드가 privileged mode 에서 실행되기 때문에 mode switching 에서 나오는 오버헤드가 없다.

4. 결론

임베디드 시스템에서의 대표적인 보호기법에 대해서 정리하였다. Control-flow 무결성은 코드의 실행흐름이 의도한 것과 달리 되는 것을 방지하는 방식이다. 임베디드 시스템의 특성과 제약에 맞게 셰도우 메모리나 하드웨어 트레이싱 유닛을 사용한 방법들이 연구되었다. Control-flow 무결성과 Data 무결성이 위반되었는지 원격 컨트롤러에서 확인할 수 있는 방법인 attestation 기법도 있었다. 또한, 구획화를 달성하는 방식으로 MPU 설정을 변경하거나 컴파일 타임에 규칙을 강제하는 연구도 소개하였다. 이로써 향후 임베디드 시스템의 소프트웨어 보안 기법을 위한 현 기법들에 대한 분석과 통찰력을 제시하였다.

Acknowledgement

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원 (NRF-2022R1C1C11010494) 및 정보통신기획평가원의 지원 (RS-2022-II220688, RS-2022-II221199, RS-2024-00437306, RS-2024-00439819) 을 받아 수행된 연구임.

참고문헌

[1] Jie Zhou, et al. Silhouette: Efficient protected shadow stacks for embedded systems. In 29th USENIX Security Symposium (USENIX Security 20), pages 1219–1236.

[2] Yufei Du, et al. Holistic Control-Flow protection on Real-Time embedded systems with kage. In 31st USENIX Security Symposium (USENIX Security 22), pages

2281–2298.

[3] Xi Tan, et al. Sherloc: Secure and holistic control-flow violation detection on embedded systems. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS ’23, page 1332–1346.

[4] Yujie Wang, et al. Insectacide: Debugger-based holistic asynchronous cfi for embedded system. In 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS), page 360–372.

[5] Zhichuang Sun, et al. Oat: Attesting operation integrity of embedded devices. In 2020 IEEE Symposium on Security and Privacy (SP), page 1433–1449.

[6] Chung Hwan Kim, et al. Securing Real-Time Microcontroller Systems through Customized Memory View Switching. In Proceedings of the 25th Network and Distributed System Security Symposium (NDSS 2018), San Diego, CA.

[7] Arslan Khan, et al. Low-cost privilege separation with compile time compartmentalization for embedded systems. In 2023 IEEE Symposium on Security and Privacy (SP), page 3008–3025.

[8] Arslan Khan, et al. Ec: Embedded systems compartmentalization via intra-kernel isolation. In 2023 IEEE Symposium on Security and Privacy (SP), page 2990–3007.