연산 효율적인 API 호출 핑거프린팅 기법 고안을 위한 인터럽트 연산의 미시적 프로파일링

하영빈[1], 안성규[1], 장우현[2], 김연재[2], 허재원[2], 박기웅[3†]
[1]세종대학교 SysCore Lab.
[2]LIG넥스원
[3]세종대학교 정보보호학과
hybin0615@naver.com, yiimfn@gmail.com, woohyun.jang2@lignex1.com,
yeonjae.kim@lignex1.com, jaewon.huh@lignex1.com, woongbak@sejong.ac.kr

Micro-Profiling of Interrupt Operations to Design an Efficient API Call Fingerprinting Techniques

Young-Bin Ha[1], Sung-Kyu Ahn[1], Woo-Hyun Jang[2], Yeon-Jae Kim[2], Jae-Won Heo[2], Ki-Woong Park[3†]
[1]SysCore Lab., Sejong University
[2]LIG Nex1
[3]Dept. of Computer and Information Security, Sejong University

요 약

국방·항공 분야에서 활용되는 임베디드 시스템은 정해진 시간 내에 반드시 동작을 처리해야 하는 실시간성과 운영 안정성이 핵심 요소이지만, 물리적으로 노출되었다는 특성으로 인해 내부 정보 유출 및 정상적인 동작을 수행하지 못하는 상태가 되는 등의 보안 위협이 공존한다. 이러한 위협을 탐지하기 위한 여러 연구가 진행되어 왔으며, 그 중 디바이스 드라이버를 모니터링하는 방식이 있다. 그러나 이러한 방식은 실제 RTOS 환경에서 적용되면 안정성과 성능적인 측면에서 문제점이 발생한다. 따라서, 본 연구에서는 디바이스 드라이버 API 호출 모니터링 과정에서 보다 효율적인 연산 기법을 마련하기에 앞서, RTOS 내부 동작을 프로파일링하는 실험을 통해, 드라이버 API 호출 및 인터럽트 처리 과정에서 나타나는 실행 시간과 호출 패턴을 분석한다.

1. 서론

RTOS 기반의 임베디드 시스템은 단순한 전자 장치를 넘어 항공, 위성, 무기 체계 등 국가 안보와 직결되는 분야에서 핵심적인 역할을 수행한다. 그러나 이러한 시스템은 임무 수행의 특성상 장치가 공격자에 의해 탈취될 경우, 시스템 내부의 펌웨어 변조나 제어 흐름 탈취를 통해 임무 수행을 방해하거나 오작동을 유발할 수 있다. 이러한 상황에서 전원을 차단하거나 시스템을 업데이트 및 재부팅하여 복구하는 것은 현실적으로 불가능한 경우가 다수 발생하며, 그 결과 임무 실패나 기술 유출이 동반될 수 있다.[1] 따라서 원활한 임무 수행과 기술 유출 방지를 위해서는 시스템에 대한 탈취 시도나 펌웨어 변조 등을 실시간으로 신속하게 탐지하고 대응할 수 있는 능력이 필수적이다.[2] 이와 같은 필요성으로 시스템에서 발생하는 위협을 탐지하는 다양한 연구가 수행되어 왔다. 대표적인 방법으로는 전류, 전압, 진동, 통신 버스 등 물리적인 데이터를 활용하는 방식이나 실행 로그 기반 모니터링이 있다.[3, 4, 5, 6, 7] 그러나 무기 체계에서 센서 데이터 기반 탐지는 노이즈나 환경 등의 요인으로 인해 오탐, 미탐이 발생하기 쉬우며, 실행 로그 기반의 방식은 리소스 부족, 로그 조작의 문제가 있다. 이러한 방식 외에 펌웨어 내부의 디바이스 드라이버 API 호출을 모니터링하여 탐지를 수행하는 방법도 제안되었다. 이 방식은 외부 요인에 영향을 덜 받는 장점이 있으나, 성능 저하 및 안정성 리스크라는 문제를 유발할 수 있다는 점에서 추가적인 연구가 필요하다.

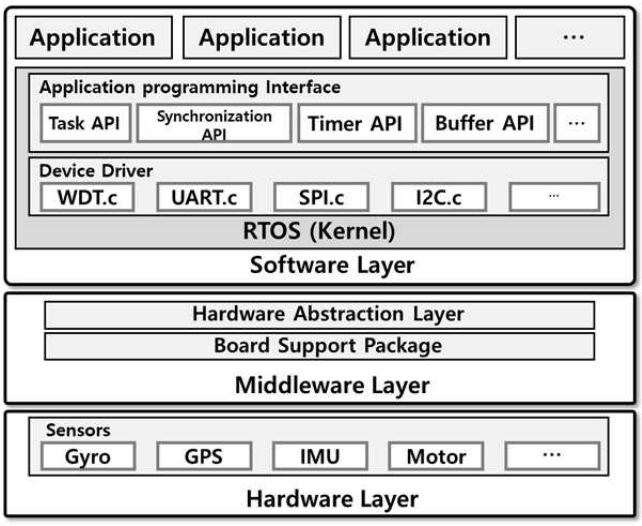
따라서, 본 연구는 제한된 자원을 갖는 RTOS 환경의 임베디드 시스템에서, 드라이버·모듈별 API 호출을 효율적으로 모니터링하여 행동 지문(Behavioral fingerprint)을 생성하고 이를 기반으로 시스템에서 발생하는 이상을 탐지하는 기법을 제안하기에 앞서, 시스템 내부에서 발생하는 실행 시간, 호출 패턴 등 동작 특성을 프로파일링 하고자 한다. 본 논문의 구성은 2장에서 RTOS 구조에 대한 배경 지식 및 드라이버 API를 이용한 이상 탐지 연구

에 관하여 기술하고, 3장에서 시스템 내부의 동작에 대한 프로파일링 과정을 서술한다. 끝으로 4장에서는 결론 및 향후 연구에 대해 언급하며 마무리하고자 한다.

2. 배경 지식 및 관련 연구

2.1 RTOS에서 하드웨어 추상화 계층 구조

RTOS 환경의 시스템은 주로 다양한 하드웨어 모듈이 결합되어 동작하는 특성을 가진다. 그러나 이러한 구조는 특정 하드웨어 플랫폼에 종속되기 쉬우며, 장치가 변경될 경우 코드 수정이나 재설계가 필요하다. 이러한 문제를 해결하기 위해 HAL(Hardware Abstraction Layer) 구조를 사용하곤 한다. HAL 구조는 디바이스 드라이버를 기반으로 하드웨어와 상위 계층을 연결하는 추상화 계층으로 구성되어 있다. 이러한 구조는 공용 API를 통해 다양한 장치를 일관된 방식으로 제어할 수 있기에, 하드웨어에 대한 의존성을 낮출 수 있으며, 시스템은 하드웨어 변경 시에도 최소한의 수정을 통해 높은 이식성과 유지보수성을 확보하게 된다. 그림 1은 RTOS 환경에서의 시스템 계층적 구조 아키텍처를 표현한다.



(그림 1) RTOS 기반 시스템 계층적 구조 아키텍처

2.2 Driver 계층 데이터를 이용한 이상 탐지

RTOS에서 드라이버 API 호출은 응용 프로그램과 하드웨어 사이의 주요 인터페이스로, 이는 시스템 내부 동작을 모니터링하는 핵심 지표로 활용될 수 있다. 드라이버 API는 단순한 하드웨어 제어 명령을 넘어, 태스크가 특정 시점에 어떤 방식으로 장치에 접근하는지를 드러내는 행위의 흔적이며, 이를 통해 행동지문을 생성할 수 있다.

이와 관련해 Prashanth Krishnamurthy 외 2인 은 드라이버 및 시스템 호출 계층에서의 호출 패턴이 사용자의 동작이나 시스템의 상황에 따라 달라진다는 점에 주목하였다.[8] 정상적인 실행 환경에서 나타나는 API 호출들의 분포와 시퀀스를 학습하고, 이를 기준으로 예상치 못한 호출 조합이나 순서 변화를 이상으로 식별하였다.

Fucci는 드라이버의 올바른 동작을 형식화하고 심볼릭 실행을 통해 디바이스 드라이버의 코드가 규칙을 따르는지를 검증하는 방식을 통해 결함을 탐지하는 방식을 제안했다.[9]

3. 시스템 내부 동작 프로파일링

3.1 실험 환경 구성

실험은 ARM Cortex-M7 마이크로컨트롤러를 탑재한 보드에서 진행되었으며, 해당 보드에 RTOS를 포팅하여 실험 환경을 구성하였다. ARM Cortex-M 시리즈는 저전력·경량 설계와 함께 실시간 처리가 중요한 임베디드 응용 분야에서 많이 활용되고 있다.

3.2 드라이버 및 인터럽트 계층 모니터링

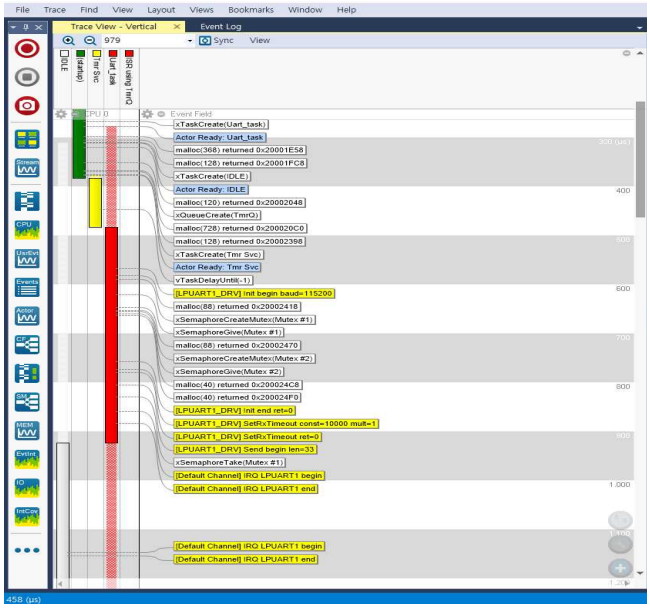
본 절에서는 디바이스 드라이버와 인터럽트 경로에서의 동작을 프로파일링하기 위해 적용한 후킹 기반의 모니터링 절차와 그에 대한 실험 결과를 기술한다. 우선, 드라이버 레벨에서는 디바이스 드라이버 API의 진입 및 복귀 지점에 로깅을 하도록 구현하였다. 이를 통해 API 호출 직후 발생하는 큐 송/수신, 세마포어 획득 및 해제, 태스크 전환 시점 등이 연속적으로 기록되며, API 호출과 커널 서비스간의 상관관계를 시계열 관점에서 관찰할 수 있다.

인터럽트 레벨에서는 스타트업 코드에 정의되어있는 IRQ 핸들러를 래핑하여 후킹 지점을 삽입하였다. 실제 디바이스 드라이버 핸들러가 호출되기 전과 후에 이벤트를 남기도록 구성함으로써, 인터럽트가 발생했을 시 ISR 실행 구간을 기록하고, 세마포어 신호처리 및 커널 스케줄러에 의해 수행되는 태스크 재개 과정 등을 추적할 수 있도록 하였다.

로깅은 최대한 시스템 실행을 불필요하게 지연시키지 않으며 이벤트를 기록할 수 있도록 하기 위해 Percepio Tacealyzer의 TraceRecorder 모듈에서 제

공하는 링 버퍼 기반 논블로킹(Non-Blocking) 방식으로 수행되었다.

수집된 트레이스는 Tracealyzer 상에서 가시화를 하였으며, 아래의 그림 2는 수집한 실행 트레이스의 결과를 나타낸다. 그림 2에서 확인할 수 있듯이 관련 API 호출이 반복적으로 발생하고, 그 이후 메모리 할당, 타이머, 인터럽트 진입 그리고 커널 내부에서 발생한 세마포어 처리 및 태스크 재개 등이 수행되는 과정을 볼 수 있다. 이를 통해 API 호출이 드라이버 계층과 인터럽트 처리, 커널 서비스까지 어떠한 경로로 연결되는지를 추적할 수 있었으며, RTOS 내부 동작을 프로파일링을 수행하였다.

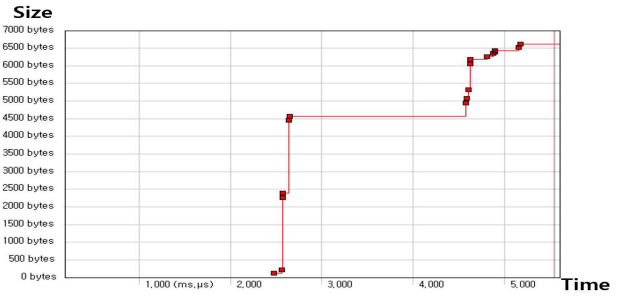


(그림 2) Tracealyzer를 이용한 동작 추적 결과

그림 3은 앞서 언급한 방법으로 수집된 트레이스 데이터에 대한 시간별 힙 메모리 사용량을 나타낸 결과이며, 해당 그림 3의 X축은 프로그램 실행 후 경과된 시간을 의미하고, Y축은 특정 시점에 할당된 메모리 사용량을 나타낸다.

후킹을 삽입하는 것은 단순히 로그 문자열만 남기는 것이 아니라, 내부적으로 추가적인 구조체 생성, 버퍼 관리 같은 작업을 필요로 하며, 또한 세마포어 및 뮤텍스와 같은 객체도 적지 않은 메모리 블록을 차지하기에 메모리 사용이 증가할 수 있다.

실제 실험 결과 단일 디바이스 드라이버 요소를 모니터링할 경우, API 호출이 발생하는 시점부터 μs 단위의 매우 짧은 시간에 약 2.6 KB 정도의 메모리를 사용하는 것을 확인하였으며, 그림 3과 같이 모니터링 범위를 늘렸을 경우 메모리 사용량이 7 KB 정도로 증가하는 것을 확인할 수 있다. 이러한 결과는 후킹 기반 모니터링이 시스템 내부 동작 특성을 관찰할 수 있음을 보여주지만, 동시에 후킹 지점을 과도하게 적용하거나 모니터링 범위를 확장하게 되면 자원이 제한적인 환경에서는 연산 성능에 부정적인 영향을 주게된다는 것을 나타낸다. 나아가 디바이스 드라이버 코드에 접근하거나 커버하는 방식의 후킹은 장치의 타이밍, 상태에 변화를 초래해 오작동을 유발할 수 있고, 후킹으로 인해 내부 상태가 노출되면 공격자가 이를 표적으로 삼아 시스템을 악용할 가능성도 커진다.



(그림 3) 힙 메모리 사용량 변화

4. 결론 및 향후 연구

본 연구에서는 실제 보드에 RTOS 환경을 구축하고, Tracealyzer를 활용하여 태스크 실행 흐름과 디바이스 드라이버의 동작을 추적하는 실험을 수행하였다. 이를 위해 드라이버 API 호출 지점과 인터럽트 핸들러 진입 및 종료 지점 등에 후킹 코드를 삽입하여 API 호출 및 인터럽트 처리 과정을 기록하였다. 실험 결과, 후킹 과정에서 핸들 등록과 구조체 초기화 등이 수행되는데 각 동작들이 수 μs 단위로 수행되는 동안 수 KB 규모의 메모리 사용이 발생하는 것을 확인하였으며, 이러한 동작이 고빈도로 반복될 경우 지연이 누적되어 실시간성이 중요한 RTOS 환경에서 동작하는 시스템의 성능 저하로 이어질 수 있음을 확인하였다. 해당 실험을 통해 디바이스 드라이버 계층까지 모니터링이 가능함을 입증함과 동시에, 후킹 기반 모니터링이 가지는 한계 또한 파악할 수 있었다. 이를 바탕으로 향후 연구에서는 RTOS의 철학을 위배하지 않으며, 효율적이고 경량화된 연산 처리를 통해 디바이스 드라이버를 모니터링 하는 시스템에 대한 연구를 수행하고, 이를 통해 시스템 내에서 발생하는 이상을 탐지하는 방향

을 제시하고자 한다.

Acknowledgement

이 논문은 2022년 정부(방위사업청)의 재원으로 국방기술진흥연구소의 지원을 받아 수행된 연구임 (KRIT-CT-22-051)

참고문헌

[1] B. Kim, “Analysis of Satellite RTOS Security Requirements”, Korea Institute of Information Security and Cryptology, vol. 35(2), 25-30, 2025

[2] S. Kong et al., “GRAND: GAN-based software runtime anomaly detection method using trace information” vol. 169, pp. 365 - 377, 2024

[3] Jose Paulo G. de Oliveira et al., “Non-invasive embedded system hardware/firmware anomaly detection based on the electric current signature”, Advanced Engineering Informatics, vol.51, pp. 101519, 2022

[4] H. J. Im et al., “Anomaly Detections Model of Aviation System by CNN”, Journal of Aerospace System Engineering, vol.17, no. 4, pp. 67-72, 2023

[5] R. Hou et al., “Unsupervised graph anomaly detection with discriminative embedding similarity for viscoelastic sandwich cylindrical structures”, ISA transactions, vol. 147, pp. 36 - 54, 2024

[6] E. Levy et al., “AnoMili: Spoofing Prevention and Explainable Anomaly Detection for the 1553 Military Avionic Bus“, arXiv preprint arXiv:2202.06870, 2022

[7] Anton Landor, “Runtime Monitoring on a Real-Time Embedded System”, Master’s thesis, Dept. of Computer and Information Science, Linköping University, Sweden, LIU-IDA/LITH-EX-A-21/003-SE, 2021

[8] Prashanth Krishnamurthy et al., “Enabling Deep Visibility into VxWorks-Based Embedded Controllers in Cyber-Physical Systems for Anomaly Detection,” arXiv preprint arXiv:2504.17875, 2025

[9] F. Fucci, “Model-Based Verification of Operating Systems Device Drivers,” Ph.D. thesis, Università degli Studi di Napoli Federico II, Italy, 2015