PAC 기반 CFI 동향 분석

Analysis of CFI Trends Based on a PAC

이정민*
정보보호학과
세종대학교
서울특별시, 대한민국
neutrinox4b1@gmail.com

요 약

4차산업혁명의 도래와 함께 사물인터넷 기술의 발전은 다양한 분야의 연결성과 효율성을 증대시키며, 우리의 삶을 윤택하게 하고 있다. 하지만, 사물인터넷의 발전에 따라 악의적인 공격자는 제한된 환경에서 낮은 컴퓨팅 파워로 동작하는 임베디드 장비에 대한 지능화되고 고도화된 공격을 수행하여 국가핵심기반시설을 파괴하거나 동작 불능의 상태에 빠지게 하는 공격 사례가 지속적으로 등장하고 있다. 이에 임베디드 장비를 대상으로 한 주요 공격이 코드 재사용 공격과 같은 실행 흐름 하이재킹 공격이 사용되고 있음으로 분석된 바 이를 보안하기 위한 제어 흐름 무결성과 관련된 연구가 이루어지고 있는 실정이다. 특히 최근에는 포인터 인증 코드를 기반으로 하여 낮은 오버헤드와 높은 보안성을 가지는 제어 흐름 무결성 기법들이 연구 및 적용되고 있는 추세임에 따라 본 논문에서는 임베디드 시스템의 안전성을 제고하기 위한 포인터 인증 코드 기반 제어 흐름 무결성 기술 연구/개발 동향을 분석하여 임베디드 장비 보안을 위한 경량화되고 최적화된 제어 흐름 무결성 기술의 초석을 다지고자 한다.

키워드 : 제어 흐름 무결성, PAC, 임베디드 보안, 시스템 보안, IoT 보안

1. 서론

IoT(Internet Of Things) 기술의 발전은 의료, 가전, 교통 등 다양한 분야에서 연결성과 효율성을 증대시켰으나, 동시에 IoT 장치를 대상으로 하는 보안 위협도 가져왔다. SonicWall에서 2023년 및 2024년에 발표한 '사이버 위협 보고서'에 따르면 IoT 장치에 대한 공격 시도는 2023년에 2022년 대비 15%가 증가하였으며, 2022년에 2021년 대비 IoT 멀웨어는 87% 증가한 바 있다 [1][2].

또한, 2016년 우크라이나 대규모 정전 사태에서 해커들은 내부에 침투하여 산업제어시스템 디바이스를 감염시킴으로써, 키예프 북쪽 지역 전반을 정전시켜 피해를 발생시킨 바 있다. 이와 같은 사례는 IoT 장비를 통한 사이버 공격이 국가 핵심 인프라에 심각한 피해를 야기할 수 있음을 시사한다 [3].

일반적으로 임베디드 시스템을 대상으로 하는 공격은 디바이스의 펌웨어나 운영체제의 취약점을 이용한다. 공격 방식으로는 'Memory Corruption을 통한 코드 재사용 공격(Code Reuse Attack, CRA)'이 대표적이며 'ROP(Return-Oriented Programming)', 'JOP(Jump-Oriented Programming)'와 같이 스택이나 점프 테이블 등을 활용하여 실행 흐름을 제어한다.

제어 흐름 무결성(Control-Flow Integrity, CFI)은 CRA같은 실행 흐름 하이재킹 공격에 효과적인 대응책이다. CFI는 프로그램이 개발자가 설계한 방향과 같이 정상적이고 예상 가능한 프로세스로 실행될 수 있도록 보장하는 기술이다. 즉, 이는 실행 흐름을 모니터링하여, 코드 블록 간의 관계를 표시한 제어 흐름 그래프(Control-Flow Graph, CFG)에 부합하는지를 검증한다. CFI는 Abadi에 의해 제안된 이후, 정적 분석뿐만 아니라 동적 분석 기법을 활용하여 정교한 CFG를 생성함으로써 코드 커버리지를 향상시키는 효과적인 보안 기법에 관한 연구가 지속되고 있다 [4].

특히 산업 환경에서 주요 사례를 살펴보면 2016년 ARM에서는 하드웨어 기반 CFI 중 하나로, PAC(Pointer Authentication Code)를 도입한 바 있다. 이후, PAC를 활용하는 새로운 CFI가 등장하고 있으며 낮은 오버헤드를 가지는 강화된 보안 기법들이 적용되고 있는 추세이다 [5]. 이에 본 논문에서는 임베디드 환경에서 널리 사용되는 AArch64 아키텍처의 PAC와 PAC를 기반으로 하는 CFI 기술 연구를 분석함으로써 임베디드 시스템의 안전성을 제고하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 PAC의 정의와 특징에 대해 분석하며, 3장에서는 PAC를 활용한 CFI 연구 동향을 기술한다. 4장에서는 본 논문의 결론을 맺는다.

2. PAC(Pointer Authentication Code)

본 장에서는 PAC 기술의 정의와 원리를 이해하고, 그에 따른 특징을 기술한다. 그림 1을 통해 PAC 생성 및 적용 과정을 살펴볼 수 있다.

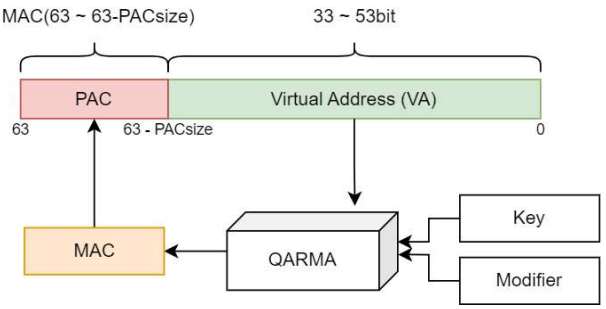

그림 1. PAC 생성 및 적용

PAC는 포인터에 메타데이터인 인증 코드를 추가하여 포인터의 무결성을 검증하고 유지하는 기법으로, ARMv8.3-A부터 도입되었다. PAC의 개념은 64bit 아키텍처에서 실제로 사용하는 주소 공간의 크기가 64bit보다 작다는 것에서 시작한다. 실제 사용되는 가상 주소 공간의 크기는 33bit에서 53bit 사이로 구성되며, PAC는 주소 지정 방식과 태깅 사용 여부에 따라 나머지 3bit에서 31bit의 크기를 가진다 [6]. 이처럼 PAC는 변동적인 크기를 가짐에 따라 고정적인 크기를 가지는 MAC(Message Authentication Code)을 일부 잘라 사용한다. MAC은 64bit 포인터, 64bit 수정자 그리고 128bit 키를 통해 생성된다. 생성 알고리즘의 경우, 제조업체별로 선택할 수 있으나, 일반적으로 QARMA 블록 암호 알고리즘을 사용한다 [7].

MAC 생성에 사용되는 키는 총 5개로 'APIAKey_EL1', 'APIBKey_EL1', 'APDAKey_EL1', 'APDBKey_EL1', 'APGAKey_EL1'로 구성된다 [8]. I-Key는 명령어 포인터, D-Key는 데이터 포인터를 위해 사용되며 각각의 키들은 모두 EL1 이상에서만 'Read/Write' 권한을 가진다. 해당 키들을 사용하여 PAC를 서명, 검증하기 위해 AArch64 명령어 집합이 확장되었다.

PAC* 명령어는 포인터를 서명하기 위해서 사용하는 명령어이다. 예를 들어, PACIAZ 명령어는 '명령어 포인터(I)를 A-Key와 0값의 수정자(Z)를 사용하여 서명(PAC)한다'는 의미를 갖는다.

AUT* 명령어는 서명된 포인터를 검증하기 위해서 사용하는 명령어이다. PAC* 명령어와 동일한 구조를 가지며 PAC를 검증하기 위한 명령어이다.

이외에도 BLRAAZ, RETAA 등의 명령어를 통해 함수의 forward edge, backward edge를 보호할 수 있다.

앞서 언급한 바와 같이, PAC는 실제 사용하는 주소 크기를 활용함에 따라 추가적인 메모리 오버헤드를 감소시킨다. 또한, EL1 권한의 키를 사용하여 일반 사용자가 PAC 값을 계산할 수 없도록 함으로써 기존 메모리 보호 기법의 Memory Leak 공격에 대한 취약점을 보완하였다. 더불어, 서명 및 검증 단계에서 명령어 집합을 확장한 하드웨어 기반 CFI로써 소프트웨어 기반 CFI에 비해 무시할 수 있는 오버헤드를 가진다. 이와 같은 특징을 기반으로 기존 보호 기법 대비 높은 보안성, 낮은 오버헤드를 달성하기 위해 PAC를 기반으로 한 CFI 연구가 활발하게 진행되고 있다.

3. PAC 기반 CFI 동향

본 장에서는 PAC를 기반으로 한 CFI 연구들을 제시한다. 연구의 핵심 아이디어와 이점에 관해 기술한다.

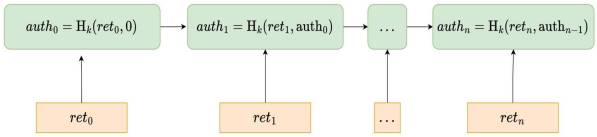

그림 2. ACS의 PAC 체인 형성 과정

Liljestrand, H. et al.은 PAC 기능을 활용하여 호출 스택의 무결성을 보장하는 ACS(Authenticated Call Stack)를 제안하였다. ACS는 함수의 반환 주소를 연쇄적으로 인증함으로써 PAC 체인을 형성하고 이를 통해 ROP 공격을 효과적으로 방어하는 기법이다 [9]. 이는 기존의 SCS (Shadow Call Stack) 보안 기법을 표방한다. 그림 2는 PAC 체인 형성 과정을 나타낸다. 그림 2에서 H_k는 MAC 생성 함수, $auth$는 인증 토큰, ret는 스택에 저장된 반환 주소를 의미한다.

ACS는 LLVM 9.0 및 AArch64 백엔드와의 통합을 통해 pacia 와 autia 명령어를 사용하여 인증된 반환 주소를 검증하도록 구현되었으며, SCS 전용 하드웨어가 아닌 ARM 범용 하드웨어 환경에서 SCS와 동등한 보안 성능을 구현하였다는 데 의의가 있다. ACS는 SPEC CPU 2017 벤치마크를 사용한 성능 평가에서 약 3.0%의 오버헤드를 나타내며 실용적임을 증명하였으나, 약 0.8%의 오버헤드를 가지는 하드웨어 기반 SCS에 비해 상대적으로 높다는 한계점을 가진다.

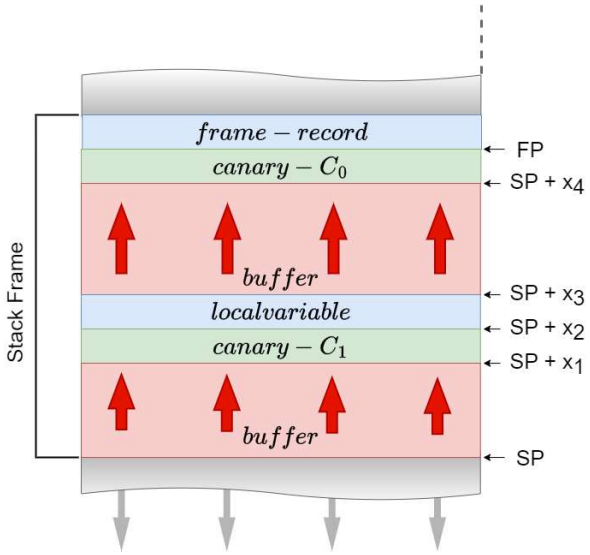

그림 3. 모든 버퍼 오버플로우를 탐지하기 위한 PCan

또한, Liljestrand, H. et al. 은 PAC를 기반으로 기존의 Canary 보호 기법을 보완한 PCan을 제안하였으며, 이는 그림 3과 같다 [10]. PCan은 동적으로 생성된 Canary를 사용하여, Memeroy Leak 공격 및 Brute Force 공격에 대한 취약성을 감소시켰으며, 함수 내의 각 버퍼에 동적으로 생성된 Canary를 배치함으로써 이를 가능하게 한다. 이는 PAC* 명령어를 통해 생성되며, 수정자는 SP(Stack Pointer) 하위 48bit와 함수 식별자(function-id) 16bit로 구성된다. 즉, 수정자 $m = \mathrm{SP} * 2^{16} + (\mathrm{function} - \mathrm{id})$를 취한다. 이와 같은 설계를 기반으로 PCan은 함수 호출에 따라 고유한 값을 가지며 공격자는 다른 함수의 Canary를 이용하여 보호 기법을 우회할 수 없게 된다. 또한, 각 버퍼에 Canary를 배치하여 반환 주소뿐 아니라 지역 변수를 대상으로 하는 공격도 탐지할 수 있다는 장점을 가진다.

PCan은 다른 반환 주소 보호 기법과 통합하여 적용이 가능함에 따라 앞선 ACS와 같은 기법을 함께 사용하였을 때 보다 더 높은 보안성을 제공할 수 있다. 성능 평가에서 PCan은 평균 0.03%의 오버헤드로 기존 Canary 방식의 - 0.08%와 비교하여 미미한 차이를 가졌으나, 동일한 함수 호출이 같은 메모리 주소에서 호출되는 특수한 경우 Memory Leak에 대한 보안성이 감소한다는 한계를 지닌다.

Kim, Y. et al.은 forward edge CFI를 강화하기 위해 CPT(Code Pointer Tagging)라는 동적 CFI 솔루션을 제안하였다 [11]. CPT는 포인터의 MAC을 CFI 라벨과 연결하여 공간 효율적인 CFG 저장소와 CFI 검증을 구현하기 위해, 하드웨어 기반 EMT(Edge Metadata Table) 해시 테이블과 추가적인 명령어 집합을 설계하였다. EMT는 MAC과 포인터의 edge metadata를 매핑하는 데이터 구조로써, 제어 흐름 검증 과정에서 신속하고 정확한 검증이 이루어지도록 하며, 제안된 명령어에 의해 접근된다. 제안된 명령어로는 tagc, estr, eact, echk가 있다. 각각의 명령어는 QARMA 블록 암호 알고리즘을 통해 생성된 MAC를 상위 비트에 배치하며(tagc) 계산된 edge metadata를 EMT에 저장하고(estr) 입력된 함수 주소를 EMT에서 활성화함으로써(eact) 활성화 여부와 CFI 라벨을 통해 CFI를 검증한다(echk). 이와 같은 메커니즘을 통해 CPT는 평균 1.2%의 오버헤드를 바탕으로 forward edge CFI를 제공한다. 이는 보안성 대비 적은 오버헤드를 가지지만 EMT 또한 공격자에 의해 무결성이 훼손될 수 있기 때문에 EMT 무결성을 위한 오버헤드 또한 고려해야 한다. 또한, 해당 연구에서는 EMT 무결성 보호를 위해 하드웨어 기반 SCS를 제안하였으며, 이는 앞선 ACS를 활용한 EMT 무결성 보호에서도 논의될 수 있을 것으로 판단된다.

Schilling, R. et al.은 Software Attack 및 Fault Attack으로부터 제어 흐름을 보호하기 위해 FIPAC라는 솔루션을 제안하였다 [12]. 기존 CFI는 Software Attack과 Fault Attack 중 하나에만 효과적이며, 큰 런타임 오버헤드를 가지고 광범위한 하드웨어 수정이 필요하였다. 해당 연구에서는 Software Attack과 Fault Attack으로부터 시스템을 보호하고 대규모 배포를 위해 4가지 요구 사항을 제시하고 있다. 우선적으로 CFI 보호는 미세한 세분화가 이루어져야 한다. 또한, CFI 상태 업데이트 함수는 적절하게 선택되어야 하며 대규모 배포를 위해 하드웨어 변경이 필요하지 않고 컴파일 중에 자동으로 적용되어야 한다. 이에, FIPAC는 AArch64

표 1. PAC 기반 CFI 기술 비교

제안 시스템	특징	한계점	보호 대상	평균 오버헤드
ACS [9]	• 반환 주소를 연쇄적으로 인증하여 PAC 체인을 형성함	• 하드웨어 기반 SCS에 비해 상대적으로 높은 오버헤드를 가짐	Backward edge	약 3.0%
PCan [10]	• 함수마다 다른 Canary를 사용하며 각 버퍼에 Canary를 배치함	• 동일한 함수가 같은 메모리에서 호출되는 경우 보안성이 감소함	Backward edge, Local variable	0.03%
CPT [11]	• EMT 테이블과 제안된 명령어로 CFI 검증을 증속함	• EMT 테이블의 무결성 또한 보호되어야 하며, 이에 대한 오버헤드도 존재함	Forward edge	1.2%
FIPAC [12]	• Software Attack과 Fault Attack 모두에 대한 보호를 제공함	• 타 기법들에 비해 상대적으로 높은 오버헤드를 가짐	Forward edge, Backward edge	35 ~ 105%

환경에서 PAC를 활용함으로써 요구 사항을 만족시켰다. 소프트웨어 기반 설계인 점을 고려하여 CFI 상태 세분도를 basic block 수준으로 낮추었으며, CFI 상태 업데이트 함수에 PAC를 사용하였다. 이를 기반으로 PAC를 사용함으로써 보안성과 오버헤드의 이점을 얻을 수 있음을 확인하였다. PAC를 통해 생성된 CFI 상태는 이전 상태와 XOR(eXeclusive OR)되어 basic block들 사이에 종속성을 가지도록 한다. 이렇듯, FIPAC는 기존 CFI가 가지는 보안성을 향상시켰으나 SPEC 2017 성능 평가에서 평균 35~105%의 오버헤드를 가지는 한계점이 존재한다. 이에, 추후 연구에서는 해당 기법의 오버헤드를 감소시키는 방향으로 연구가 진행될 것으로 사료된다.

4. 결론

본 논문에서는 임베디드 시스템의 안전성을 제고하기 위해 PAC 기반 CFI 기술의 동향을 분석하였다. PAC의 생성 및 적용 과정과 그에 따른 특징을 확인하였으며, 이를 기반으로 하는 CFI 동향 분석을 수행하였다. 이후, 동향 분석에서는 각 시스템의 핵심 원리 및 보호 대상과 평균 오버헤드를 조사, 비교/분석하였다. 이에 Canary를 보완한 PCan의 경우 backward edge와 Local variable뿐 아니라 기법을 응용하여 heap 영역 보호에도 적용될 수 있을 것으로 사료된다. 향후, 현재까지 분석한 연구 내용을 토대로 CFI 구현에서 취약점을 분석하고 보완함으로써 임베디드 시스템 환경에 최적화된 CFI 기법을 연구/개발하고자 한다.

참고문헌

[1] “2024 SonicWall Cyber Threat Report”. [Online]. Available: https://www.sonicwall.com/medialibrary/en/white-paper/2024-cyber-threat-report.pdf [Accessed: 2024-03-13].

[2] “2023 SonicWall Cyber Threat Report”. [Online]. Available: https://www.sonicwall.com/medialibrary/en/white-paper/2023-cyber-threat-report.pdf [Accessed: 2024-03-13].

[3] “Large-scale blackouts in Ukraine”. [Online]. Available: https://www.ciokorea.com/news/32445 [Accessed: 2024-03-13].

[4] Abadi, Martín, et al. "Control-flow integrity principles, implementations, and applications." In proc of: *ACM Transactions on Information and System Security (TISSEC)* 13.1, pp. 1-40, 2009.

[5] “Examining Pointer Authentication on the iPhone XS”, [Online]. Available: https://googleprojectzero.blogspot.com/2019/02/examining-pointer-authentication-on.html [Accessed: 2024-03-13].

[6] “Pointer authentication on armv8.3”, [Online]. Available: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/pointer-auth-v7.pdf [Accessed: 2024-03-13].

[7] Avanzi, Roberto. "The QARMA block cipher family. Almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes." In proc of: *IACR Transactions on Symmetric Cryptology,* pp. 4-44, 2017.

[8] “Arm Architecture Reference Manual for A-profile architecture”, [Online]. Available: https://yurichev.com/mirrors/ARMv8-A_Architecture_Reference_Manual_(Issue_A.a).pdf [Accessed: 2024-03-13].

[9] Liljestrand, Hans, et al. "Authenticated call stack." In proc of: *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1-2, 2019.

[10] Liljestrand, Hans, et al. "Protecting the stack with PACed canaries”. In proc of: *Proceedings of the 4th Workshop on System Software for Trusted Execution*, pp.1-6, 2019.

[11] Kim, Yonghae, et al. "Hardware-Assisted Code-Pointer Tagging for Forward-Edge Control-Flow Integrity”. In proc of: *IEEE Computer Architecture Letters,* pp. 117-120, 2023.

[12] Schilling, R., Nasahl, P., & Mangard, S. “FIPAC: Thwarting Fault-and Software-Induced Control-Flow Attacks with ARM Pointer Authentication”. In proc of: *In International Workshop on Constructive Side-Channel Analysis and Secure Design,* pp. 100-124, 2022.