CFI 기술의 연구 동향 및 모바일 기기에의 적용 가능성 고찰

이용제*, 이진용*, 허인구*, 문현곤*, 황동일*, 백윤흥*
*서울대학교 전기정보공학부
e-mail:yjlee@sor.snu.ac.kr

A Study on Researches for CFI enforcement and their Application to Mobile Devices

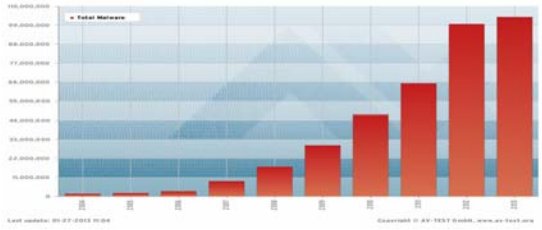
Yong-Je Lee*, Jinyong Lee*, Ingoo Heo*, Hyungon Moon*, Dongil Hwang*
Heo*
*Dept of Electrical and Computer Engineering, Seoul National University

요 약

최근 포스트 PC 시대가 시작되면서 다양한 모바일 기기가 상호 연결되는 트렌드가 나타나고 있다. 하지만 이로 인해 나타나는 보안상의 취약점 문제는 모바일 기기 상에서의 보다 강력한 보안성 요구하게 되었다. 이 논문에서는 보안 취약점에 대처하는 연구 중 *Control Flow Integrity* (CFI) 연구에 대한 동향을 살펴보고, 이를 모바일 기기에 적용하기 위해서 고려해야 할 점에 대하여 논하겠다.

1. 서론

최근 Microsoft와 Intel의 연합을 상징하는 윈텔(Wintel)이 컴퓨터 시장을 석권했던 시대를 지나 포스트 PC 시대가 시작되고 있다. 포스트 PC 시대는 데스크톱 PC 의 판매량이 감소하고 스마트폰이나 태블릿 등의 모바일 기기가 시장을 지배하게 되는 새로운 트렌드를 의미한다[1]. 포스트 PC 시대가 가지는 가장 핵심적인 특징은 연결성이다. 모바일 기기, 가전제품, 자동차, 클라우드 서버 등 모든 전자기기는 상호간에 연결됨으로써 이동성 및 휴대성을 보장받게 되며 이를 통해 다양하고 편리한 서비스가 제공될 것이다. 이처럼 장치 간 연결성이 부각되고 이를 통한 서비스가 시장에 등장할 때 놓치지 말아야 할 키워드 중 하나는 보안이다. 그림 1에서 볼 수 있듯이 세계적으로 다양한 목적을 가진 사이버 공격들이 증가하고 있으며, 우리나라도 이러한 사이버 공격으로부터 예외일 수는 없다. 특히, 현대 경제 연구원에 따르면, 2009년에 국내에서 발생한 7.7 DDoS공격의 경우 금전적 피해 규모가 최소 363억 원에서 최대 544억 원에 이른다고 한다[2].



(그림 1) 지난 10년간의 악성코드 추이

기존의 보안 공격자들은 자신들의 목적을 달성하기 위해서 컴퓨터 시스템의 취약점을 찾아내고 그 취약점을 이용하여 악성 코드를 수행시키려는 노력을 하였다. 이런 소프트웨어 취약점을 노린 공격들이 늘어나면서 반대로 이런 공격들에 대한 대처 방법에 대한 연구들도 활발히 진행되었다. 예를 들어 공격자들이 널리 사용하는 버퍼 오버플로(buffer overflow) 의 경우, 이를 활용한 공격으로 ROP(Return Oriented Programming)가 있는데, 이 공격은 버퍼 오버플로를 발생시켜 스택에 저장된 리턴 주소(Return Address)를 변경함으로써 공격자가 의도한 코드를 수행하도록 한다. 이를 막기 위해서 하드웨어로 구현된 스택에 저장을 하고, 리턴 시 이를 활용하게 함으로써 리턴 주소를 보호하고자 하였다[3]. 이 연구가 공격자가 삽입한 코드로의 점프를 막고자 하였다면, 이후 DEP (Data Execution Prevention) 기술을 구현하여 공격자가 데이터 영역에 삽입한 코드의 실행을 불가능하게 하는 기술도 연구되었으며, 이 기술은 상용화된 대부분의 CPU 에 구현되어 있다. DEP의 일반화로 인해 공격자들이 기존에 이미 존재하는 코드를 도구(gadget)로 재사용 및 연결하여 악성 코드를 만들어 실행하는 방법인 CRA(Code Reuse Attack)를 고안하였으며, 최근에는 이에 대응하기 위한 연구도 활발히 진행되고 있다.

이 논문에서는 이러한 보안 취약점에 대한 공격 대처 방법 중 하나인 *Control-Flow Integrity* (CFI) 를 보장하는 연구 동향을 알아보고 모바일 기기의 사실상의 표준인 ARM 프로세서 기반 시스템에서 CFI 보장을 효율적으로 구현하는 방법에 대해서 논하도록 하겠다.

	플랫폼	구현 방법	성능 저하	대상
CFI [4] (CCS'05)	x86	machine-code instrumentation ID-check for indirect calls	0~45% (SPEC 2000)	응용프로그램
SBCFI [5] (CCS'07)	x86	state-based CFI check kernel state periodically Xen VMM 상에 구현	3%~ (10s period) ~240% (1s period)	OS 커널
MoCFI [6] (NDSS'12)	ARM	정적분석과 실행 시간 분석으로 나눔 binary load 시에 수행한 정적분석의 결과를 이용하여 CFI 보장을 위해 binary rewriting 을 수행	1%~5x 성능 저하 (Gensystek Lite Benchsuite) 10000 개 Quick Sort 시 13x 성능 저하	응용프로그램
BR [7] (ISCA'12)	x86	CFG 를 그리는 대신 통상적인 branch 의 특성을 이용하여 CFI 를 보장 추가적인 하드웨어 구현	~2% 성능 저하 (SPEC 2006)	응용프로그램
CFIMon [8] (DSN'12)	x86	상용 프로세서의 BTS 와 PMU 활용. 기존의 H/W, S/W 에 대한 수정 없음.	2.3%~8.4% 성능 저하 (server applications)	응용프로그램
CCFIR [9] (S&P'13)	x86	Springboard 라는 새로운 코드 영역 도입 indirect control-flow transfer 가 Spring board 에 있는 코드로 분기하는지 검사.	3.6%/8.6% (평균/최대) 성능 저하 (SPEC CPU 2000)	응용프로그램

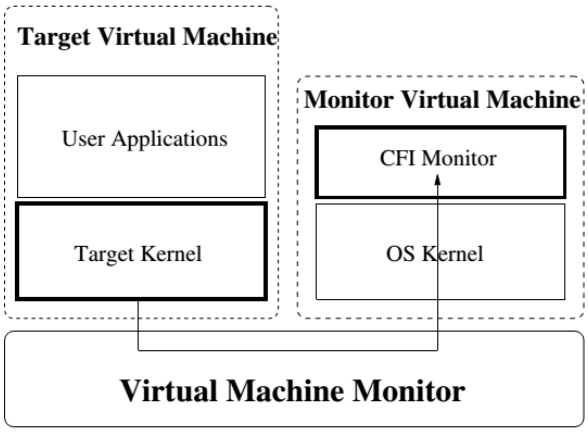
(표 1) CFI 보장 연구의 흐름

2. 기존의 CFI 연구 동향

CFI 보장은 2005년 Abadi [4] 에 의해서 처음으로 제안된 개념으로 프로그램 내의 모든 제어 흐름이 의도한 대로 흘러가는 것을 확인함으로써 현재 수행 중인 프로그램의 무결성을 보장하는 기법이다. 즉, 미리 분석해 놓은 Control-Flow Graph (CFG) 를 가지고 실제 프로그램 수행이 CFG 와 부합되는지를 확인하는 방식이다. 이는 많은 공격들이 자신들의 목적을 달성하기 위해 제어 흐름을 변경하는 것으로 미루어 보았을 때 효과적인 대처 방법이라고 하겠다.

Abadi 가 제안한 CFI 보장 기법은 코드의 instrumentation 을 통하여 indirect 분기의 source 주소와 destination 주소에 고유의 ID 를 부여하고 분기 시에 source 주소의 ID 와 destination 주소의 ID 를 비교하여 같으면 분기를 하고, 다르면 예외(exception) 을 발생시키는 방식이었다. 이 방법은 공격을 탐지하는 데는 효과적이었으나 매 분기에 대해서 고유의 ID 를 부여하고 그 ID를 비교함으로써 성능 저하가 뚜렷한 단점이 있었다.

이어서 2007년에는 Petroni 가 *state-based control-flow integrity* (SBCFI)[5] 라는 방법을 발표하였다. 이는 모든 분기문에 대한 감시를 수행하지 않고, 주기적으로 OS Kernel 의 상태, 즉 kernel에서 변하지 않아야만 하는 data struct 들이 변조되었는지를 확인하여 위반 여부를 판단하는 방법이다. 이는 루트킷(rootkit) 들이 CFG 에 지속되는(persistent) 수정을 가하게 된다는 점에서 착안한 것으로 감시 주기로 인한 성능 저하와 검사의 정확성을 트레이드오프(trade-off) 관계로 설정할 수 있다. 또한 CFI 모니터를 Xen VMM 상에서 외부 모니터 형태로 구현함으로써 tampering 으로부터 쉽게 보호될 수 있는 구조를 가진다는 장점이 있다. 하지만 이 방법 또한 검사 주기를 1초 정도로 설정했을 때 240% 정도의 성능 부하를 가지는 등 한계를 가지고 있다.

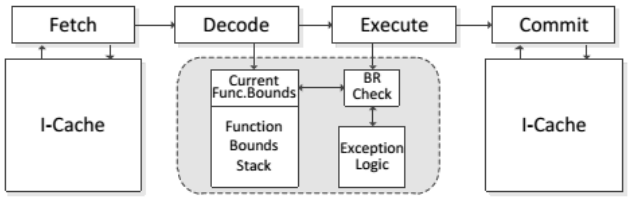


(그림 2) SBCFI 모니터 구조도

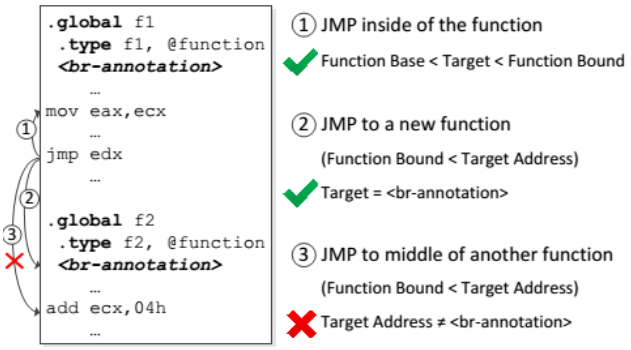
2012년에는 L. Davi 에 의해서 최초의 ARM 플랫폼 상에서 수행되는 CFI 방법[6]이 발표되었다. *MoCFI*(Mobile CFI) 라고 명명한 이 프레임웍은 기존의 연구가 x86 기반 플랫폼에 편중되었던 반면 최초로 ARM 프로세서 기반 스마트 기기에서 사용 가능한 CFI 보장 기법이라는 데에 그 의미가 있다. MoCFI 는 정적분석을 통해 binary 의 branch 및 CFG 정보를 얻게 되고, 이를 통해 binary 를 rewriting 하고 CFI 보장을 수행하게 된다. 특히 ARM 과 Intel x86 사이의 구조적인 차이로 인하여 CFI 의 구현을 어렵게 하는 요소를 제시하고 이에 대한 해결책을 제시했다는 것에 대해 높이 평가할 수 있지만, 바이너리 분석의 불완전함으로 인해 일부 calls/jumps 가 임의의 valid 한 function 시작 주소로 뛰는 것을 허용하는 단점이 있으며, 여전히 큰 성능 저하로 인해 널리 적용이 힘들었다.

역시 2012년 ISCA 에 발표된 *Branch Regulation* (BR)[7] 은 M. Kayaalp 에 의해 수행된 연구로서 CRA 를 막기 위한 연구이다. 이 연구에서는 branch 가 1) 호출

된 function 범위 내의 주소로 뛰는지, 2) 다른 function 의 시작 주소로 뛰는지 만을 검사하여 Return-Oriented programming (ROP) 이나 Jump-oriented programming (JOP) 등 CRA 를 막으려고 하였다. 또한 Processor 의 파이프라인에 Function Bound Stack (FBS) 를 구현하여 적법한 리턴 주소와 function 의 경계점을 저장하게 하고 Execute stage에서 계산된 분기 주소와 비교함으로써 그 적법성을 검사하게 된다. (그림 3) 과 (그림 4) 에 BR 의 추가 하드웨어 도식도와 BR 이 분기 주소의 적법성을 검사하는 정책을 도식화하였다.



(그림 3) BR 을 위한 프로세서 파이프라인 수정

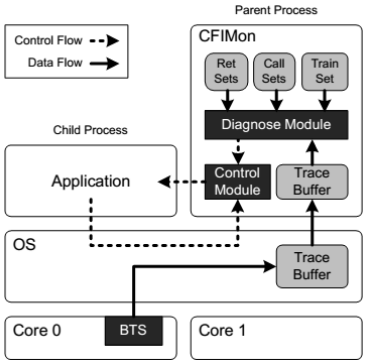


(그림 4) BR 분기주소 적법성 검사

그러나 BR은 (그림 3) 에 나타낸 대로 프로세서의 파이프라인을 수정하는 방식을 채용하였는데, 이는 추가적인 설계와 검증에 대한 부담으로 기존의 프로세서 제조사에서 쉽게 채용하기에는 어렵다는 단점이 있다.

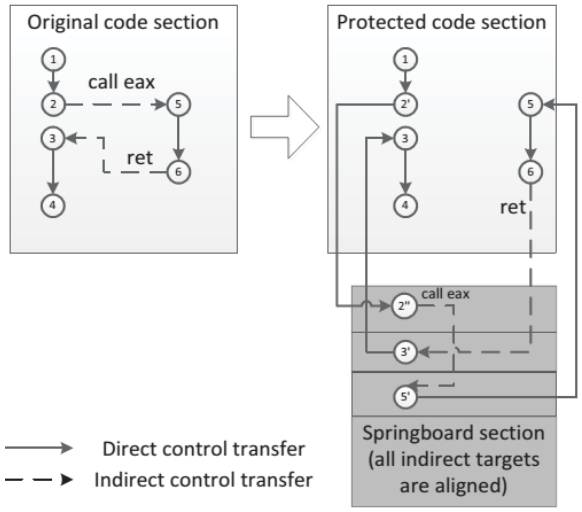
또한 같은 해 Y. Xia 등에 의해 *CFIMon[8]* 이라는 방법이 제안되었다. CFIMon 은 기존 상용 프로세서에 있는 *Branch Trace Store (BTS)*와 *Performance Monitoring Unit (PMU)*를 이용하여 CFI 를 보장하는 방법을 제시하였다. CFIMon 은 offline phase 와 online phase 로 나누어져서 offline phase 에는 프로파일링을 통해 응용 프로그램 분기 주소의 적법한 집합을 만들어내고, online phase 에는 분기 주소를 실시간으로 감시하면서 offline phase에서 미리 준비한 적법한 집합과 비교하여 공격 가능성을 판단하게 된다.

CFIMon 은 kernel 과 user 수준 도구로 크게 나누어진다. Kernel 수정은 user 수준 도구가 BTS 등에서 나오는 감시 신호를 볼 수 있도록 경로를 만들어준다. user 수준 도구는 Diagnose Module 과 Control Module 로 나누어지는데, Diagnose Module 은 call_set, ret_set 그리고 train_set 을 입력으로 받아서 CFI 를 검사하게 된다. Control module 은 CFIMon 환경을 초기화하고 응용프로그램과의 동기화를 하는데 사용된다. 이렇게 구성된 CFIMon 은 널리 쓰이는 server 응용 프로그램에 대해서 평균 6.1% 의 성능 저하를 보이게 된다. 그러나 이 방법은 code-injection attack 혹은 CRA 에 대해서 false positive 나 false negative 를 발생시킨다는 단점이 있다.



(그림 5) CFIMon 의 아키텍처

2013년 IEEE S&P 에 실린 CCFIR[9] 은 Springboard 라고 명명한 새로운 코드 영역을 정의하고 원래 코드에서 indirect jump 시에 반드시 Springboard 의 코드 영역을 지나가도록 코드에 수정을 가하여 이 성질을 가지고 CFI 위반 여부를 판단하는 방법이다.



(그림 6) CCFIR의 구조

(그림 6)에서 볼 수 있듯이, CCFIR 방법은 Protected code section 과 Springboard section 사이에 direct jump 를 추가하여 모든 indirect 분기의 타깃 주소 (그림 상의 노드 3과 5) 로 jump 할 때 Springboard 를 거치도록 강제하게 된다. 이 방법은 x86 플랫폼 상에 구현되어 SPEC CPU 2000 벤치마크를 수행하였을 때 3.6% 의 성능 저하

를 일으키는 것으로 보고되었다. 하지만 CFI 보장을 위해서 Springboard 라는 별도의 메모리 공간을 확보해야 한다는 점에서 메모리 양등에 제한을 가지는 모바일 환경에서는 적용이 쉽지 않다는 단점을 가진다.

3. 모바일 기기에의 적용 가능성

2장에서는 현재까지 진행된 CFI 관련 연구의 동향에 대해서 살펴보았다. 현재까지의 방법론은 소프트웨어를 통해 CFI 보장을 할 수 있도록 정적분석을 통해 CFG 를 추출하고 실행 파일 내에 제어 흐름을 테스트 할 수 있는 코드를 삽입하는 방법이 주를 이루었다. 여기서 주목할 점은 대부분의 연구가 x86 플랫폼 상에서 진행되었다는 점이다. MoCFI 가 ARM 플랫폼을 이용하기는 하지만 성능 저하가 매우 크기 때문에 널리 쓰이기는 힘든 실정이다. 그러므로 ARM 기반의 모바일 기기에 CFI 를 적용하기에는 추가적인 고려 사항들이 필요하다.

첫째는, 별도의 외부 하드웨어의 필요성이다. 모바일 기기는 데스크톱 기반의 컴퓨팅 환경과 달리 성능과 메모리 자원 측면에서 제한이 있기 때문에 소프트웨어로 구현하는 경우에는 프로세서의 부하가 가중되게 된다. 기존의 컴퓨터 시스템의 발전도 어떠한 연산 요구가 있을 때, 그래픽 처리기(GPU) 혹은 디지털 신호 처리기(DSP) 같은 특정 연산에 최적화된 하드웨어를 개발하여 그 처리를 전담하도록 발전해왔다. CFI 보장을 위해서도 외부의 하드웨어를 설계하여 붙이면 기존 프로세서의 연산 부하를 늘리지 않은 채로 보안성을 강화하는 효과를 낼 수 있을 것이다. 둘째는 외부 하드웨어에 프로그램의 제어 흐름을 알려줄 수 있도록 프로세서의 수정이 필요하다. 이 부분은 하드웨어의 수정이 프로세서의 검증에 최소한의 부담만을 주도록 내부의 분기 주소를 하드웨어 핀으로 뽑는 정도의 사소한 수정만을 가해야 한다. 이를 통해서 외부 하드웨어는 분석된 CFG 정보와 호스트로부터 받은 분기 주소 정보를 이용해 CFI 보장을 효율적으로 할 수 있을 것이다. 마지막으로, 외부 하드웨어를 이용하여 호스트 시스템으로부터 물리적으로 격리시키는 효과를 냄으로써 호스트 시스템의 무결성과 독립적인 안전한 보안 감시 시스템의 구축이 가능해 질 것이다.

4. 결론

본 논문에서는 지금까지의 CFI 보장 연구의 동향을 살펴보고 이를 모바일 기기에 적용하기 위한 조건 등을 논하였다. 앞으로 포스트 PC 시대에 걸맞은 다양한 모바일 기기들이 등장하고, 이들이 상호 연결되어 보안성에 대한 관심이 커질 때 전담 하드웨어의 도움을 받아 보안성을 강화하는 접근법은 모바일 기기의 성능 저하를 최소화하면서도 높은 보안성을 가질 수 있게 하는 유망한 기술이 될 것이며, 관련 연구 분야에 새로운 플랫폼과 연구 방향을 제시함으로 인해 보안 기술 연구의 새로운 전기를 마련할 수 있을 것으로 예상한다.

Acknowledgement

본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(과제번호 2012-0000470), 중소기업청에서 지원하는 2012년도 산학연공동기술개발사업(No. C0019562), 미래창조과학부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신) [No. 10047212, 1kB 이하 암호문 간의 연산을 지원하는 동형 암호 원천 기술 개발 및 응용 연구] 및 IDEC의 지원을 받아 수행하였습니다.

참고문헌

[1] en.wikipedia.org/wiki/Post-PC_era

[2] 한국과학기술단체총연합회, 사이버 냉전시대, kofst Issue paper, 2011-03

[3] S. Sinnadurai, Q. Zhao, and W. fai Wong. Transparent runtime shadow stack: Protection against malicious return address modifications, 2008

[4] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. “Control-flow integrity” in Proceedings of CCS, pages 340-353. ACM, 2005.

[5] N. L. Petroni, Jr. and M. Hicks. “Automated Detection of Persistent Kernel Control-Flow Attacks” in Proceeding of the 14th ACM conference on Computer and Communications Security, October 2007.

[6] L. Davi, A. Dmitrienko, M. Egele, T. Fischer, T. Holz, R. Hund, S. N¨ urnberger, and A.-R. Sadeghi, “MoCFI: A framework to mitigate control-flow attacks on smartphones,” in Network and Distributed System Security Symposium (NDSS), 2012.

[7] M. Kayaalp, M. Ozsoy, N. Abu-Ghazaleh, and D. Ponomarev, "Branch Regulation: Low-Overhead Protection from Code Reuse Attacks" in ACM/IEEE International Symposium on Computer Architecture (ISCA), 2012

[8] Y. Xia, Y. Liu, H. Chen, and B. Zang, “CFIMon: Detecting violation of control flow integrity using performance counters,” in IEEE/IFIP International Conference on Dependable Systems and Networks, 2012.

[9] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, and W. Zou, "Practical Control Flow Integrity & Randomization for Binary Executables" in IEEE Symposium on Security and Privacy (S&P), 2013