인텔 프로세서 트레이스를 이용한 제어 흐름 무결성에 대한 연구

백세현*, 서지원*, 양명훈*, 신장섭*, 백윤흥*
*서울대학교 전기정보공학부
e-mail : shbaek@sor.snu.ac.kr

A Study on Control-Flow Integrity Using Intel Processor Trace

Se-Hyun Baek*, Ji-Won Seo*, Myong-Hoon Yang*, Jang-Seop Shin, Yun-Heung Paek
*Dept. of Computer Science, Seoul National University

요 약

제어 흐름을 원래 의도와 다르게 만드는 공격을 제어 흐름 무결성을 보장하는 방법을 통해 보안성을 높여 왔다. 완벽한 보안성과 낮은 성능 저하를 만족시키기 위해 인텔 프로세서 트레이스 하드웨어를 이용한 제어 흐름 무결성을 보장하는 연구들에 대해 소개한다.

1. 서론

공격자가 코드를 삽입하는 공격 등을 막는 다양한 보안 기법이 제안되고 발달함에 따라 이미 메모리에 있는 코드들을 이용함으로써 원하는 공격이 이루지게 하는 코드 재사용 공격 (code reuse attack)에 초점을 맞추고 있다. 코드 재사용 공격과 같이 제어 흐름을 원래 의도와 다르게 만드는 공격을 제어 흐름 무결성 (Control-Flow Integrity)을 보장하는 방법을 통해 보안성을 높여왔다. 기본적으로 제어 흐름 무결성은 소프트웨어 실행이 반드시 제어 흐름 그래프 (Control-Flow Graph)의 한 경로를 따라 수행되어야 한다는 것을 뜻한다[1]. 제어 흐름 무결성을 보장하기 위해 Control-Flow Locking[2], CCFI[3] 등과 같은 연구들이 있었다. 그 중에서도 기존 소프트웨어 방식의 성능 저하와 한계점을 극복하기 위해 하드웨어의 도움을 받아 제어 흐름 무결성을 보장하는 연구들에는 ROPecker[4], Kbouncer[5] 등이 있다. 하지만 이들 역시 완벽한 보안성과 낮은 성능 저하 둘을 동시에 만족할 수 없었다. 최근 제어 흐름에 대한 로그를 완벽하게 얻을 수 있고 매우 효율적인 하드웨어인 인텔 프로세서 트레이스 (Intel Processor Trace)가 소개되었다. 본 연구에서는 인텔 프로세서 트레이스를 이용한 제어 흐름 무결성을 보장하는 연구들을 그 방법에 따라 소개한다. 나아가 이들 간의 특징을 비교하고 한계점을 분석하여 앞으로의 개선 방향을 제시한다.

2. 배경이론

2.1. 인텔 프로세서 트레이스

인텔 프로세서 트레이스[6]는 최근 인텔에서 소개한 하드웨어이며 적은 성능 저하로 제어 흐름들을 완벽하게 기록할 수 있는 것이 특징이다. 인텔 프로세서 트레이스는 시스템의 실행 흐름 정보를 트레이스 패킷 (trace packet)들의 흐름 형태로 추출해준다. 즉 시스템의 정해진 메모리 영역에 시스템이 동작함에 따라 발생하는 패킷들을 저장하는 기능을 수행한다. 이 때 패킷은 주요 레지스터의 업데이트나 분기 명령어의 수행 등 시스템의 동작에 따라 여러 종류가 있다.

<표 1> 인텔 프로세서 트레이스에서의 패킷과 그 설명

패킷 종류	패킷 설명
PSB	Packet Stream Boundary 패킷으로, 일종의 심장박동 기능을 하는 패킷이다. 즉 메모리 영역에 일정량의 패킷들이 쌓이면 발생하는 패킷이다.
TNT	Taken Not-Taken 패킷으로, 직접 조건 분기 명령어에서 분기의 방향을 나타내는 패킷이다.
TIP	Target IP 패킷으로, 간접 분기, 예외처리, 인터럽트가 발생할 때의 타겟 주소를 기록하는 패킷이다.
FUP	Flow Update 패킷으로, 인터럽트와 예외처리가 발생한 경우 발신 주소를 기록하는 패킷이다.

2.2. 제어 흐름 무결성

제어 흐름 무결성[1]은 작성자의 의도와 다른 경로로 프로그램이 실행되지 않도록 하는 것이 목적이다. 기존의 제어 흐름 무결성은 프로그램의 제어 흐름 그래프 (Control-Flow Graph)를 구성한 다음 실행 중에 각 간접 제어 흐름 전달의 타겟이 그래프와 맞는지 확인하는 형태이다. 제어 흐름 무결성 구현에는 파인 그레인드 (fined-grained)와 코스 그레인드 (coarse-grained)로 나뉜다. 파인 그레인드는 간접 제어 전달마다 자기만의 타겟 주소 집합이 있다. 반면에, 코스 그레인드는 간접 제어 전달과 그들의 타겟 주소를 분리시켜 놓는다.

3. 시스템 디자인

3.1. PT-CFI

PT-CFI[7]는 완벽한 후방 간선 (backward-edge) 제어 흐름 무결성을 목적으로 한다. 정적 바이너리 (binary) 다시 쓰기나 동적 수단을 이용하지 않고 그림자 스택 (shadow stack)을 이용한다. 해당 연구가 제안한 시스템은 네 단계로 이루어져있다. 네 단계는 각각 패킷 파싱 (Packet Parsing), TIP 그래프 부합 (TIP Graph Matching), 심층 검사 (Deep Inspection), 시스템호출 가로채기 (Syscall Hooking)로 이루어져 있다.

먼저, 패킷 파싱은 프로세서 트레이스의 다양한 패킷들을 가져오고 TIP 그래프를 만드는 단계입니다. 추가적으로 모든 가능한 입력들에 대해 훈련 (Training)을 시킴으로써 성능을 향상시킬 수 있습니다. 두 번째 단계에는 TIP 그래프와의 부합을 확인하여 알려지지 않은 경로에 대해서는 심층 검사를 거친다. 심층 검사에서는 해당 바이너리를 해체해서 경로의 옳고 그름을 더 자세히 판단한다. 심층 검사에서도 잘못된 경로임이 파악되면 실행 중인 프로세스를 종료해야 한다.

3.2. FLOWGUARD

FLOWGUARD[8]는 정적 바이너리 분석을 이용하여 제어 흐름 그래프를 구성합니다. 이를 인텔 프로세서 트레이스에 맞게 재구성한 다음 간선들에 라벨을 붙이는 훈련 단계를 거칩니다. 이를 통해 빠르거나 느린 경로로 나눈 혼합식 흐름 확인 방법을 구현할 수 있습니다.

먼저, 제어 흐름 그래프에서 직접 분기 간선을 제거함으로써 간접 타겟으로 연결된 제어 흐름 그래프 (ITC-CFG)를 구성한다. 만들어진 ITC-CFG 에서 프로그램 실행 중에 확인이 더 필요한 지에 따라 간선마다 라벨을 붙이는 훈련을 거친다. 이후 라벨을 기준으로 빠른 경로와 느린 경로를 분리시켜 혼합식으로 흐름을 확인한다. 확인된 모든 간선들이 확인이 더 필요한 경우가 아니라면 빠른 경로에 해당한다. 반대의 경우 느린 경로에 해당하여 실행중 문맥과 바이너리를 통해 더 정밀한 확인을 거치게 된다. 빠른 경로는 발신 주소 (source address)의 배열을 검색하고 해당 발신 주소가 가리키는 곳이 타겟 주소의 배열에 있는지를 확인한다. 느린 경로는 전체 실행 흐름을 모두 파악하고 문맥적 분석을 수행한다.

3.3. GRIFFIN

GRIFFIN[9]은 보안성과 성능 사이의 교환에 대한 유동적 대응을 위해 다양한 제어 흐름 무결성 정책을 지원한다. 정책에는 코스 그레인드, 파인 그레인드, 스테이트풀(stateful)이 있다. 코스 그레인드는 간접 제어 전달의 타겟이 옳은지 만을 판단하는 가장 단순하면서 가장 보안성이 낮은 정책이다. 파인 그레인드는 발신-목적지 쌍에 대해 타겟 주소가 옳은지를 판단하기 때문에 코스 그레인드보다 보안성이 높다. 스테이트풀은 그림자 스택을 이용해 프로그램 실행 상태로 타겟을 제한한다.

코스 그레인드는 각 코드 페이지마다 코드 위치가 옳은지를 저장하는 코스 그레인드 정책 페이지 (coarse-grained policy page)를 구성하여 옳으면 1 틀리면 0 으로 설정하여 이를 기준으로 판단한다. 파인 그레인드는 발신지를 행, 목적지를 열로 하는 행렬을 구성한 후 발신 주소를 알기 위해 바이너리를 해체하고 패킷을 추적한다. 스테이트풀은 제어 흐름

을 연속적으로 처리하기 위해 콜 (call)과 리턴 (return)을 실행 순서대로 처리한다.

4. 결론

이 논문은 인텔 프로세서 트레이스를 이용하여 제어 흐름 무결성을 보장하는 세 연구에 대해 소개했다. PT-CFI 는 패킷 파싱, TIP 그래프 부합, 심층 검사, 시스템 콜 가로채기의 네 단계로 이루어진 시스템을 제안했다. FLOWGUARD 는 ITC-CFG 를 구성하고 라벨을 붙이는 훈련을 거친 후 혼합식 흐름 확인을 통해 제어 흐름 무결성을 보장하고자 했다. 마지막으로 GRIFFIN 은 제어 흐름 무결성을 위해 코스 그레인드, 파인 그레인드, 스테이트풀 세 가지 정책을 제안했다. 세 연구 모두 벤치마크로 SPECCPU 2006 을 기준으로 PT-CFI, FLOWGUARD, GRIFFIN 각각 21%, 3.79%, 9.5% 의 성능저하가 있다.

인텔 프로세서 트레이스를 이용한 제어 흐름 무결성에 대한 연구에서 아직 바이너리 코드 수준의 전방 간선에 대한 완벽한 해결책이 없다. 뿐만 아니라 위의 연구들은 사용자 어플리케이션 수준에서의 제어 흐름 무결성에 대한 제안이지만 커널 단의 제어 흐름 무결성에 대한 연구는 아직 없다. 앞으로 이와 같은 한계점들에 대한 후속 연구를 기대할 수 있다.

5. ACKNOWLEDGEMENT

본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었으며 (IITP-2017-2015-0-00403), 2017 년도 정부(미래창조과학부)의 재원으로 정보 통신 기술진흥 센터의 지원을 받아 수행된 연구 (No.2016-0-00078, 맞춤형 보안 서비스제공을위한클라우드기반지능형보안기술개발) 및 2017 년도 두뇌 한국 21 플러스 사업에 의하여 지원되었음

참고문헌

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. ´ Control-flow integrity. In 12th CCS, pages 340–353, 2005.

[2] T. Bletsch, X. Jiang, and V. Freeh. Mitigating code-reuse attacks with control-flow locking. In Proceedings of the 27th Annual Computer Security Applications Conference, CSAC '11, pages 353–362. ACM, 2011.

[3] A. J. Mashtizadeh, A. Bittau, D. Boneh, and D. Mazières. CCFI: Cryptographically enforced control flow integrity. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, pages 941–951. ACM, 2015.

[4] Y. Cheng, Z. Zhou, M. Yu, X. Ding, and R. H. Deng. ROPecker: A generic and practical approach for defending against ROP attack. In Proceedings of the 2014 Network and Distributed System Security Symposium, NDSSâ˘A ´ Z14, 2014.

[5] V. Pappas, M. Polychronakis, and A. D. Keromytis. Transparent ROP exploit mitigation using indirect branch tracing. In Proceedings of the 22Nd USENIX Conference on Security, SEC'13, pages 447–462, Berkeley, CA, USA, 2013. USENIX Association.

[6] Intel 64 and IA-32 architectures software developer's manual. Volume 3 (3A, 3B, 3C & 3D): System Programming Guide, 2016.

[7] Y. Gu, Q. Zhao, Y. Zhang, and Z. Lin. PT-CFI: Transparent backward-edge control flow violation detection using intel processor trace. In Proceedings of the 7th ACM Conference on Data and Application Security and Privacy (CODASPY). ACM, 2017.

[8] Y. Liu, P. Shi, X. Wang, H. Chen, B. Zang, and H. Guan. Transparent and efficient cfi enforcement with intel Processor trace. In Proceedings of the 23rd IEEE Symposium on High Performance Computer Architecture (HPCA). IEEE, 2017.

[9] X. Ge, W. Cui, and T. Jaeger. Griffin: Guarding control flows using intel processor trace. In Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Apr. 2017.