하드웨어 기반 시스템 감시 기법 분석

양명훈, 정선화, 박준모, 황동일, 백윤흥*
*서울대학교 전기정보공학부
e-mail:{mhyang, shjung, jmpark}@sor.snu.ac.kr, ypaek@snu.ac.kr

An Analysis on Hardware-based System Monitoring

MyongHoon Yang, Seonhwa Jung, Junmo Park, Dongil Hwang, Yunheung Paek*
*Dept of Electrical and Computer Engineering, Seoul National University

요 약

소프트웨어 방식의 시스템 감시 기법이 가지는 태생적인 단점으로 인해 최근에는 별도의 하드웨어를 추가하여 시스템을 감시하는 여러 기법들이 소개되고 있다. 본 논문에서는 하드웨어 기반의 감시 기법이 소프트웨어 기반의 감시 기법에 대해 가지는 이점에 대해 언급하고, 여러 가지 하드웨어 기반 감시 기법에 대한 동작 방법 및 특징들에 대해 살펴보도록 한다.

1. 서론

공격자들의 시스템 공격 패턴이 다양해짐에 따라 민감한 정보를 처리하는 시스템이 적절하게 동작하는지를 감시하는 방법에 대한 여러 연구가 진행되고 있다. 비교적 최근까지도 소프트웨어적으로 시스템의 공격을 탐지하는 방법이 연구되어 오고 있지만 감시 동작을 위한 추가적인 동작으로 인해 필연적으로 성능 저하가 발생하게 된다. 그래서 최근에는 보안 동작을 위한 별도의 하드웨어를 탑재하는 방식으로 시스템의 동작을 감시하는 연구가 활발히 진행되고 있다.

본 논문에서는 기존의 소프트웨어적인 방법이 가지고 있는 한계점 극복을 위해 연구되고 있는 하드웨어 기반의 감시 시스템에 대한 특성을 먼저 언급한다. 그리고 하드웨어 기반 감시 방법 4가지[3][4][5][6]를 소개하며, 감시 방법별 하드웨어의 역할과 특징들을 살펴본다.

2. 하드웨어 기반 감시 시스템의 특성

시스템 감시 분야는 1980년 중반부터 차츰 논의되기 시작하여[1], 시스템 아키텍처의 변천에 따라 지금까지도 활발히 논의되고 있는 분야이다. 2010년도 초반까지는 소프트웨어적으로 규칙 기반(rule-based) 방법[2]과 같은 정적인(static) 방식으로 시스템을 감시하고자 했다. 하지만 소프트웨어적인 감시 방법은 감시 동작을 위한 부가적 연산으로 인해 성능 오버헤드를 필연적으로 발생시킬 뿐만 아니라, 시스템의 정상 상태의 정보 혹은 시스템 공격에 해당하는 시그니처 정보를 모두 인지하고 있어야하기 때문에 메모리 자원 낭비도 심하다. 무엇보다도 가장 치명적인 단점은 보안을 위한 코드가 공격을 당한 시스템에서 독립적으로 수행되지 않기 때문에 정상적인 감시가 어려워 질 수 있다는 점이다.

그래서 최근에는 감시를 위한 전용 하드웨어 모듈에 필요한 정보들을 여러 방법으로 호스트 시스템에서 추출해내서, 그 정보를 바탕으로 시스템을 감시하는 연구가 활발히 진행 중이다. 하드웨어 기반 감시 방법은 별도의 전용 하드웨어를 통해 이루어지기 때문에 피감시 시스템의 동작과 성능에 경미하게 영향을 주거나 전혀 영향을 주지 않는다. 그래서 이러한 특성을 가진 하드웨어 기반 감시 기법은 소프트웨어 방식의 감시 기법에 대한 적절한 대안으로 부상하고 있다.

하드웨어 기반 보안 시스템 설계에는 공통적으로 요구되는 몇 가지 조건이 있다. 첫 번째는 감시하는 대상과의 독립성이 요구된다. 만약 메모리나 연산 등을 위한 자원을 피감시 대상과 공유할 경우, 공격을 당한 피감시 대상으로 인해 감시 장치의 내부 정보들이 유출될 가능성이 있다. 또는 이와 반대로 감시 동작이 호스트 프로세서의 동작에 영향을 주어 의도치 않은 동작이 발생할 수 있다. 두 번째는 감시 동작을 위해 필요한 충분한 자원이 존재해야 한다. 감시 시스템은 감시 내용이 누락되지 않아야 하고, 호스트 프로세서가 동작하는 한 지속적으로 동작해야 하므로, 이를 위한 충분한 메모리 용량과 전력이 확보되어야 한다.

3. 하드웨어 기반 시스템 감시 기법

본 장에서는 추가적인 하드웨어 기반의 전용 감시 장치를 탑재하여 시스템을 감시하는 4가지 방법에 대해 간략히 서술하고, 각 감시 방법별 특징들을 소개한다.

3.1 스냅샷(snap-shot)을 통한 커널 무결성 감시

2장에서 언급한 것과 같이 소프트웨어 기반의 보안 방법에는 여러 한계점이 있지만, 특히 루트킷(rootkit)을 활용

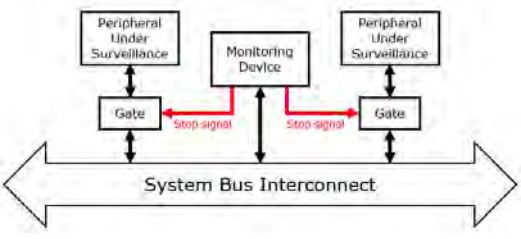
한 공격 방법에 취약하다. 루트킷이란 공격자가 이미 알려져 있는 시스템의 취약점을 찾거나, 암호를 크래킹하는 방법으로, 사용자용 접근 권한을 획득하기 위한 도구이다. 루트킷으로 인해 공격자가 커널의 내부를 수정할 수 있기 때문에 소프트웨어 방식의 감시 기법에 많은 제약 조건들이 따랐다. 이러한 문제를 해결하기 위해 Copilot[3]은 별도의 하드웨어 도움을 받아 커널을 감시하였다. 이 방식은 PCI 버스를 통해 메인 메모리의 커널의 상태를 일정 주기마다 스냅샷하는 방식으로 커널의 무결성을 확인하는 방법이다.

Copilot은 호스트의 PCI 버스에 PCI 애드인(add-in) 카드를 삽입하고, 메인 메모리에서 루트킷의 속성들에 대한 정보를 주기적으로 스캐닝한 뒤, 기존 정보들과 비교하여 커널의 수정 여부를 판단한다. 이러한 방법을 통해 시스템 호출 및 함수 테이블, 인터럽트 서술자 테이블(Interrupt Descriptor Table, IDT), 커널 텍스트, 숨겨진 프로세스 및 소켓 등에 대한 수정 여부를 확인한다.

하지만 감시 동작이 일종의 샘플링 작업인 스냅샷 동작 때만 이뤄진다는 점이 한계점으로 지목된다. 만약 공격자가 스냅샷 방식의 감시 기법을 사용하고 있다는 것을 안다면, 다음 주기적인 스냅샷 동작 사이에 커널에 수정을 가할 수 있는 멀웨어를 통해 시스템을 조작할 수 있다. 또한 스냅샷 동작이 일어날 때, 감시를 위한 메모리 접근 동작이 일어나 버스에 대한 자원 경합이 발생하게 되므로 기본적으로 약간의 성능 저하를 동반하게 된다.

3.2 버스 기반 실시간 동작 패턴 감시

이번에 언급할 감시 방법은 프로세서 및 주변 장치들이 연결된 시스템 버스의 트래픽을 감시하는 BusMOP[4]이다. 이 감시 구조는 그림 1에서 보다시피, 감시 장치 또한 시스템의 다른 주변 장치들과 마찬가지로 데이터 버스에 연결되어 버스를 지나는 데이터를 수신받게 된다. 이 때, 주변 장치 동작을 위해 특정 주소에 접근하는 패턴(메모리 읽기/쓰기 동작)이 해당 주변 장치 사용하기 위한 접근 패턴과 다를 경우, 중대한 오작동이라 판단하고 인터럽트가 발생함과 동시에 적절한 시스템 복원 작업을 수행한다.



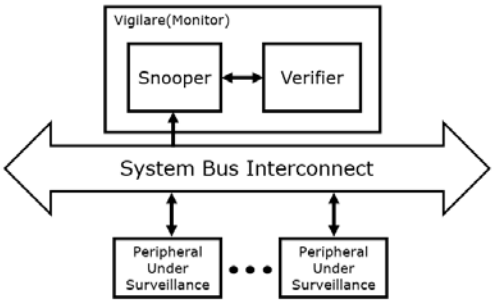
(그림 1) BusMOP의 감시 시스템 구조

추가적으로 비정상적인 접근이 발생한 주소에 대해, 해당 주소를 점유하고 있는 주변 장치가 다른 장치에 영향을 줄 수 있는 가능성이 있다. 그래서 해당 논문에서는 모든 주변 장치와 데이터 버스 사이에 게이트를 두는 방식을 사용하고 있다. 이 게이트는 시스템에 이상이 없는 평상시에는 각 주변 장치를 동작하는데 필요한 데이터들이 지나갈 수 있도록 개방해둔다. 만약 감시 장치에 의해 특정 주소의 접근이 해당 주변 장치를 사용하기 위한 패턴과 다르면 감시 장치는 게이트로 '정지 신호'를 보내어 게이트를 닫는다. 이를 통해 오류가 발생한 해당 주변 장치는 데이터 버스와의 교류가 차단되어 시스템 전체로 영향이 전파되는 것을 방지한다.

이 방식은 시스템이 동작하는 한 지속적으로 감시할 수 있어 실시간성을 가지기 때문에 3.1에서 언급한 스냅샷 기반 커널 감시의 취약점을 보완하고 있다. 또한 감시 하드웨어는 데이터의 패턴을 분석하기 위한 정도의 가벼운 사양만 갖추면 되기 때문에 최소한의 하드웨어 추가만으로 본 감시를 수행할 수 있다는 장점이 있다.

3.3 버스 기반 커널 무결성 감시

이번에 언급할 감시 방법은 데이터 버스의 트래픽을 통해 커널의 무결성을 감시하는 Vigilare[5]이다. Vigilare는 호스트 시스템 버스를 지나는 데이터 스트림을 얻어낼 수 있도록 설계되었다. 그리고 커널의 정보들이 저장된 특정 주소 영역을 불변 영역(immutable region)으로 설정하여, 부팅 이후 런타임시에 이 영역의 데이터를 수정하는 모든 동작을 시스템의 무결성을 해치는 행위로 간주한다. 이러한 방식으로 Vigilare가 감시하는 대상은 커널 코드 영역, 시스템 호출 테이블, 인터럽트 서술자 테이블(Interrupt Descriptor Table, IDT)이다.



(그림 2) Vigilare의 감시 시스템 구조

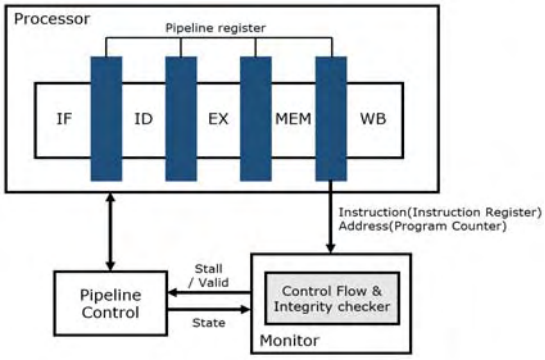
Vigilare의 감시 구조는 그림 2와 같다. Vigilare는 시스템의 트래픽을 버스로부터 받아들여 감시기(snooper)를 통해 필요한 트래픽만을 필터링해서 내보내고, 검증기(verifier)는 커널 영역의 수정을 판단하고 호스트 프로세서로 인터럽트를 내보낸다. 이 감시 구조의 특징은 커널의 수정점 검출을 위한 Vigilare 내부의 프로세서, 메모리, 시스템 버스 등이 여러 하드웨어 요소들이 감시 시스템과 완전히 별도로 마련되어 있다는 점이다. 그래서 호스트 시스템은 Vigilare의 시스템에 접근할 수 있는 방법이 없으며, 커널 수정이 일어났을 때 검증기에서 발생되는 인터럽트에 간섭할 수 없다.

이러한 방식은 주기적인 스냅샷을 통해 커널의 무결성을 감시했던 3.1과는 다르게 버스의 트래픽을 감시하기 때문

에 실시간성을 가질 수 있어 3.1의 취약점인 시간차 공격을 방어할 수 있다. 또한 호스트 시스템과는 완전히 독립적인 별도의 시스템으로 운용되기 때문에 호스트 시스템이 감시 시스템을 간섭할 수 있는 가능성이 원천 차단된다는 장점이 있다.

3.4 프로세서 내부 제어 흐름 무결성 감시

지금까지 언급한 감시 기법과는 다르게 본 감시 기법은 프로세서 내부의 파이프라인으로부터 필요한 정보를 직접 받아 별도의 감시 하드웨어에서 시스템을 감시한다[6]. 이 감시 기법은 프로그램의 동작이 원래의 의도에 맞게 동작하는지를 확인하는 것에 그 목적이 있다. 이 구조는 프로그램의 속성에 따라 1) 함수 호출 그래프로 나타낼 수 있는 프로그램의 절차간 제어 흐름(the inter-procedural control flow of a program), 2) 기본 블록 제어 흐름 그래프로 나타낼 수 있는 각 함수에 대한 제어 흐름(the intra-procedural control flow for each function), 3) 각 기본 블록 내에서의 명령어 스트림의 무결성(the integrity of the instruction stream) 세 가지 방식으로 식별이 가능하다.



(그림 4) 프로세서 내부 동작 감시 시스템 구조

다음과 같은 속성을 식별하기 위해 본 논문에서는 그림 4과 같은 시스템 구조를 설계하였다. 감시 하드웨어는 Memory access 단계와 Write back 단계 사이의 파이프라인 레지스터에 저장되어 있는 명령어와 그 주소를 각 사이클마다 넘겨받는다. 이 주소들은 미리 각 함수의 시작 주소와 반환 주소들이 저장된 해시 테이블과 비교되어 제어 흐름이 허용된 범위 내에서 일어나는지를 판단하게 된다. 만약 허용 범위 밖의 제어 흐름이라 판단되면 *invalid* 신호를 발생시키고, 프로그램 종료와 같은 방어 동작이 일어날 수 있도록 프로세서에 마스크될 수 없는 인터럽트(Non-Maskable Interrupt, NMI)를 발생시킨다. 또한 아주 드문 경우로 감시 하드웨어가 프로세서의 동작을 따라갈 수 없을 때는 *stall* 신호를 넘겨받게 된다. 이 경우, *stall* 신호가 다시 해제될 때까지 모든 파이프라인 단은 사용 불가 상태가 된다.

이러한 방식의 감시 시스템은 프로세서 내부에서 명령어와 주소를 직접 추출해낼 수 있는 환경에서는 쉽게 이 모듈을 붙일 수 있기 때문에 범용성이 좋다. 하지만 프로세서에 직접적으로 하드웨어 수정을 가할 수 없다면 본 감시 모듈을 탑재시킬 수 있다는 단점이 있다.

4. 결론

본 논문에서는 성능 저하가 필연적인 소프트웨어 방법의 한계점을 극복하는 하드웨어 기반 감시 시스템의 여러 사례들에 대해 살펴보았다. 시스템 커널, 주변 장치, 특정 프로그램 등 피감시 대상에 따라 감시에 필요한 정보가 달라짐을 보였다. 그리고 이에 따라 감시 하드웨어가 효율적으로 필요한 정보를 추출하기 위해 여러 지점에 탑재되었다. 그러므로 하드웨어 기반의 보안 모듈을 설계하고자 할 때는 감시 대상과 감시 동작을 위해 필요한 정보의 위치를 먼저 파악한 뒤 호스트 시스템과는 독립적으로 패턴이나 수정점 파악 같은 탐지 처리가 효율적으로 동작할 수 있는 하드웨어 구조를 고안해야 할 것이다.

5. Acknowledgement

이 논문은 2017년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원(NRF-2017R1A2A1A17069478), 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 결과로 수행되었으며 (IITP-2017-2015- 0-00403), 2017년도 정부(미래창조과학부)의 재원으로 정보 통신 기술 진흥 센터의 지원(No.2016-0-00078, 맞춤형 보안 서비스제공을위한클라우드기반지능형보안기술개발) 및 2017년도 두뇌 한국 21 플러스 사업에 의하여 지원되었고, 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(2015-0-00573,시스템 침입 탐지를 위한 프로세서 모니터링 기술 및 주요HW/SW 모듈 개발)을 밝힙니다.

참고문헌

[1] D. E. Denning and P. G Nuemann, “Requirements and model for IDES-A real-time intrusion detection system” Comput. Sci. Lab, SRI International, Menlo Park, CA, Tech. Rep., 1985.

[2] Howard Barringer, David Rydeheard, and Klaus Havelund ″Rule systems for run-time monitoring : from Eagle to RuleR″ Journal of Logic and Computation, Volume 20, Issue 3, 2008.

[3] N. L. Petroni, Jr., T. Faser, J. Monila, and W. A Arbaugh, “Copilot - a coprocessor-based kernel runtime integrity monitor,” in Proceedings of the 13th Conference on USENIX Security Symposium, Volume 13, ser. SSYM’04, 2004, pp. 13 - 13.

[4] R.Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu. “Hardware runtime monitoring for dependable cots-based real-time embedded

systems," In RTSS'08: Proceedings of the 29th IEEE Real-Time System Symposium, pages 481~491, 2008.

[5] H Moon, H Lee, J Lee, K Kim, Y Paek and B Kang, "Vigilare: toward snoop-based kernel integrity monitor," In CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security Pages 28-37, 2012.

[6] Divya Arora, Srivaths Ravi, Anand Raghunathan and Niraj K. Jha, "Secure Embedded Processing through Hardware-assisted Run-time Monitoring," In DATE '05 Proceedings of the conference on Design, Automation and Test in Europe - Volume 1 Pages 178-183, 2005