System Side Channel Information 기반 Control-flow Attack 탐지 기법 설계 및 구현*

김선권*, 진홍주**, 이지원**, 이동훈**

*고려대학교 일반대학원 정보보안학과

**고려대학교 정보보호대학원 정보보호학과

Design and Implementation of A Control-flow Attack Detection Method Using System Side Channel Information

Seon Kwon Kim*, Hongjoo Jin**, Jiwon Lee**, Dong Hoon Lee**

*Graduate School of Cyber Security, Korea University

**Graduate School of Information Security, Korea University

요 약

최근 Control-flow attack 방어 기법 연구들은 실제 애플리케이션에 적용했을 때 발생하는 공격 분석 비용을 보완하기 위해 노력하고 있다. 그럼에도 불구하고, 기존 방어 기법들의 새로운 하드웨어 구현 및 알고리즘 기법은 실제 산업에서의 공격 분석 및 비용 문제에 있어 효용성이 여전히 떨어진다. System side channel information은 실행 중인 시스템 및 프로세스가 발생시키는 System utilization 정보이다. 본 논문은 기존 방어 기법들의 문제를 해결하기 위해 System side channel information을 활용한 경량화된 탐지 기법을 새롭게 제안한다. 새롭게 구현한 탐지 기법은 분석 프로그램의 프로세스가 실행되는 과정에 Thread를 통해 동적으로 프로그램의 프로세스 I/O, Memory, Context switch 정보를 Profiling한다. 특히, System side channel information이 정상 프로그램의 제어 흐름에서는 정상 패턴을 보이다가 Control-flow attack이 발생하게 되면 특이 변동 패턴을 보인다는 것을 약 1,600번의 Profiling 실험을 1,000번 수행하여 확인했다. 실제 System side channel information을 활용한 패턴 비교를 통해 특이 변동 구간을 분석하고 Control-flow attack을 탐지했다.

1. Introduction

Control-flow attack은 애플리케이션의 제어 흐름을 새로운 방향으로 바꿔 의도하지 않은 작업을 수행하게 한다. Control-flow attack 방어 기법들은 런타임에 애플리케이션을 모니터링하거나 미리 계산된 적절한 상태 집합과 비교함으로 애플리케이션의 제어 흐름을 적절한 실행 흐름으로 제한한다. 기존 Control-flow attack 방어 기법들은 성능 향상을 위해 빠른 알고리즘 기법과 추가적인 하드웨어 구현을 통해 해결하려 했다. 그러나, 기존 방어 기법들은 분석을 위해서 소스 코드 또는 Debug 정보를 필요로 하고, 적용에 많은 비용이 드는 문제로 인해 실제 산업에서 실효성이 떨어진다 [1].

최근 캐시 사이드 채널 공격 탐지 연구에서, 캐시 사이드 채널 공격과 System utilization의 관계성을 밝혔으며, 더 나아가 매 공격마다 발생하는 Cache set distribution of evictions의 특이 패턴을 기반으로 캐시 사이드 채널 공격 탐지 기법이 연구됐다 [2]. 캐시 사이드 채널 공격은 악성 소프트웨어를 통해 제어 흐름을 조작하여 Last Level Cache (LLC)에 데이터 충돌을 일으킨다는 점에서 Control-flow attack과 유사성을 보인다 [3, 4, 5]. 예를 들어, 프로그램이 시작되면서 운영체제가 실제 메모리에 적재하는 데이터의 양과 프로그램의 실행에 따른 실제 메모리 데이터 사용량을 분석하면 공격에 따른 메모리 데이터 I/O 및 사용량의 변화가 정상 흐름에 비해 큰 변동을 보임을 알 수 있다.

본 논문에서는 캐시 사이드 채널 공격 탐지 연구 방법을 Control-flow attack 탐지 연구 방법에 System side channel information을 기반으로 새롭게 적용하였다. 새롭게 적용한 기법의 기여는 다음과 같다.

- System side channel information을 활용한 Control-flow attack 탐지 기법은 기존의 분석 기법 보다 경량화된 동적 탐지 시스템으로, 실제 산업에서 필요로 되는 실시간 제어 환경에서 사용될 수 있다.
- 새롭게 System side channel information을 활용한 Control-flow attack 탐지 기법 프로토타입을 프로그램의 프로세스를 Profiling하는 Thread를 통해 구현했다.
- 약 1,600번의 Profiling 실험을 1,000번 수행하였다. 실험 결과로 발견된 System side channel information과 Control-flow attack 관계성을 활용하여, System side channel information 변동 패턴 비교 및 분석을 통하여 Control-flow attack을 탐지했다.

* 이 성과는 2021년도 과학기술정보통신부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.NRF-2021R1A2C2014428).

2. Background

2.1 Control-flow attack

Control-flow attack은 제어 흐름을 변경하기 위해 다양한 메모리 오류를 사용하여 프로그램 실행을 제어하는 공격이다 [6]. 특히, 섹션 4의 공격 시나리오인 Return Oriented Programming (ROP) [8]은 공격자가 프로그램 내에 코드를 삽입하지 않고 프로그램 주소 공간에 이미 존재하는 명령어들 중, Return 명령어로 끝나는 일련의 명령어들을 연결하여 프로그램의 제어 흐름을 조작하고 임의의 실행을 유도하는 공격이다.

2.2 System Side Channel Information

System side channel information은 컴퓨터 시스템과 프로세스에서 측정 가능한 System utilization 정보로, 시스템 Monitoring, Profiling을 통해 현재 실행 중인 시스템 및 프로세스의 CPU, 메모리, 디스크, 네트워크 사용 비율을 얻을 수 있다 [7]. 일반적으로, 시스템과 프로세스가 발생시키는 System side channel information은 수행 작업에 따라 변동을 보인다.

3. System Side Channel Information 기반 공격 탐지 기법

본 논문에서 제시하는 System side channel information 기반 Control-flow attack 공격 탐지 기법은 프로그램 실행에 발생하는 System utilization 중 사용자가 지정한 정보를 탐지하기위해 메인 Thread로부터 서브 Thread를 생성하여 주기적으로 프로세스를 Profiling한다. System utilization 탐지에는 시스템 Profiling 라이브러리를 사용하여 프로그램의 프로세스가 사용하는 메모리양, 프로세스에 발생하는 Context switch 수, 프로세스의 I/O 정보를 Profiling한다. System side channel information 패턴 비교를 통해 얻은 특이 변동 구간을 분석하여 Control-flow attack을 탐지한다.

3.1 System utilization

섹션 4에서 Profiling하는 System utilization 특성은 다음과 같다.

- Unique Set Size (USS): 프로세스에 할당된 고유한 메모리 공간으로, 프로세스가 종료되면 해제된다.
- Proportional Set Size (PSS): 다른 프로세스들과 공유된 메모리 공간으로, 공유 프로세스 간에 균등하게 메모리 공간이 나눠진다.
- Anonymous: 파일 시스템과 별개로 프로그램의 스택 및 힙에 대해 생성되는 가상 메모리 공간이다.
- Private clean: 프로세스에 할당된 메모리로, 디스크에서 적재된 이후 수정되지 않은 메모리이다.
- Private dirty: 프로세스에 할당된 메모리로, 디스크에서 적재된 이후 수정된 메모리이다.
- Shared clean: 여러 프로세스와 공유된 메모리로, 디스크에서 적재된 이후 수정되지 않은 메모리이다.
- Context switch - voluntary: 프로세스에서 수행한 자발적 Context switch 수이다. 프로세스가 필요로 하는 자원이 사용 불가능한 상황일 때 발생한다.
- Context switch - involuntary: 프로세스에서 수행한 비자발적인 Context switch 수이다. 프로세스가 정해진 Time slice를 넘었거나 우선순위가 높은 프로세스에 의해 전환이 생기면 발생한다.
- Read count: 프로세스가 수행한 누적된 읽기 작업 수로, 읽기 관련 System call 수를 포함한다.
- Write count: 프로세스가 수행한 누적된 쓰기 작업 수로, 쓰기 관련 System call 수를 포함한다.
- Read chars: 프로세스가 수행한 읽기 관련 총 바이트의 양이다.
- Write chars: 프로세스가 수행한 쓰기 관련 총 바이트의 양이다.

3.2 System utilization 탐지

System utilization 탐지를 위해 Python psutil library를 사용한다. 공격 시나리오 소스 코드는 Python process()를 통해 실행하며, 이를 Python threading을 통해 Profiling한다. 생성된 Thread는 psutil을 통해 공격 시나리오 소스 코드 프로세스의 System utilization을 Profiling한다. 시나리오 소스 코드를 공격하기 위해 미리 계산된 Payload를 시나리오 소스 코드 프로세스에 입력하여 공격 전, 후에 따른 System utilization을 출력한다. 출력 결과로 얻은 System side channel information의 패턴 비교를 통해 특이 변동 구간을 분석하고 Control-flow attack을 탐지한다.

4. Control-flow Attack Profiling

실험은 가상 환경의 Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-73-generic x86_64)에서 수행하였으며, Profiling 프로그램이 시나리오 소스 코드를 실행하는 동안 약 1,600번의 System utilization Profiling을 수행한다.

4.1 실험 내용

실험에서 사용한 공격 시나리오는 ROP에 따른 Control-flow attack을 고려한다. 시나리오 소스 코드 baby-rop는 C 언어로 작성됐으며, x86_64-linux-gnu의 libc-2.31.so를 포함한다. 코드 실행 시, 몇 가지 문구 출력과 함께 두 개의 입력을 받고, 이를 출력한 후 종료된다. 공격 시나리오는 Buffer over flow가 가능한 두 번째 입력에 ROP 가젯을 페이로드로 입력하고, 공격 성공에 따른 Flag 값이 출력되고 프로그램이 종료된다. 본 논문에서는 1,000번의 Profiling 실험을 수행하였으며, 각 실험에 따른 System utilization 결과가 공격 발생에 따라 어떤 연관성이 있는지 평가하고, 특이 패턴을 확인한다. 공격 발생 확인은 공격 성공 시에 Flag 파일 접근으로 생기는 프로세스의 File descriptor 수 증가 시점을 통해 알 수 있다. 또한, Profiling 출력을 통해 공격 시점의 Profiling Index를 확인함으로 알 수 있다.

4.2 실험 결과

사용자 프로그램 의도에 공격자 의도가 개입함에 따라 CPU의 필요 데이터에 메모리 적재가 증가 또는 감소하며, 15가지 System side channel information 변화를 발생시킴을 확인 했다. 그림 1은 System side channel information의 특이 변동의 변화량만을 명확히 나타내기 위해 Z-Score [2] 값에 절대값과 스케일링을 활용하여 사용 했다. Z-Score는 관찰치와 평균과의 차이를 표준 편차 단위로 나타낸 것으로, 특정 관찰치의

상대적인 위치를 나타낸다. 그림 1-a는 baby-rop의 Private clean이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 수정되지 않았던 전용 메모리 사이즈가 특이 변동했다. 그림 1-b는 baby-rop의 Private dirty이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 수정한 전용 메모리 사이즈가 특이 변동했다. 그림 1-c는 baby-rop의 Anonymous이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 가상 메모리 사이즈가 특이 변동했다.

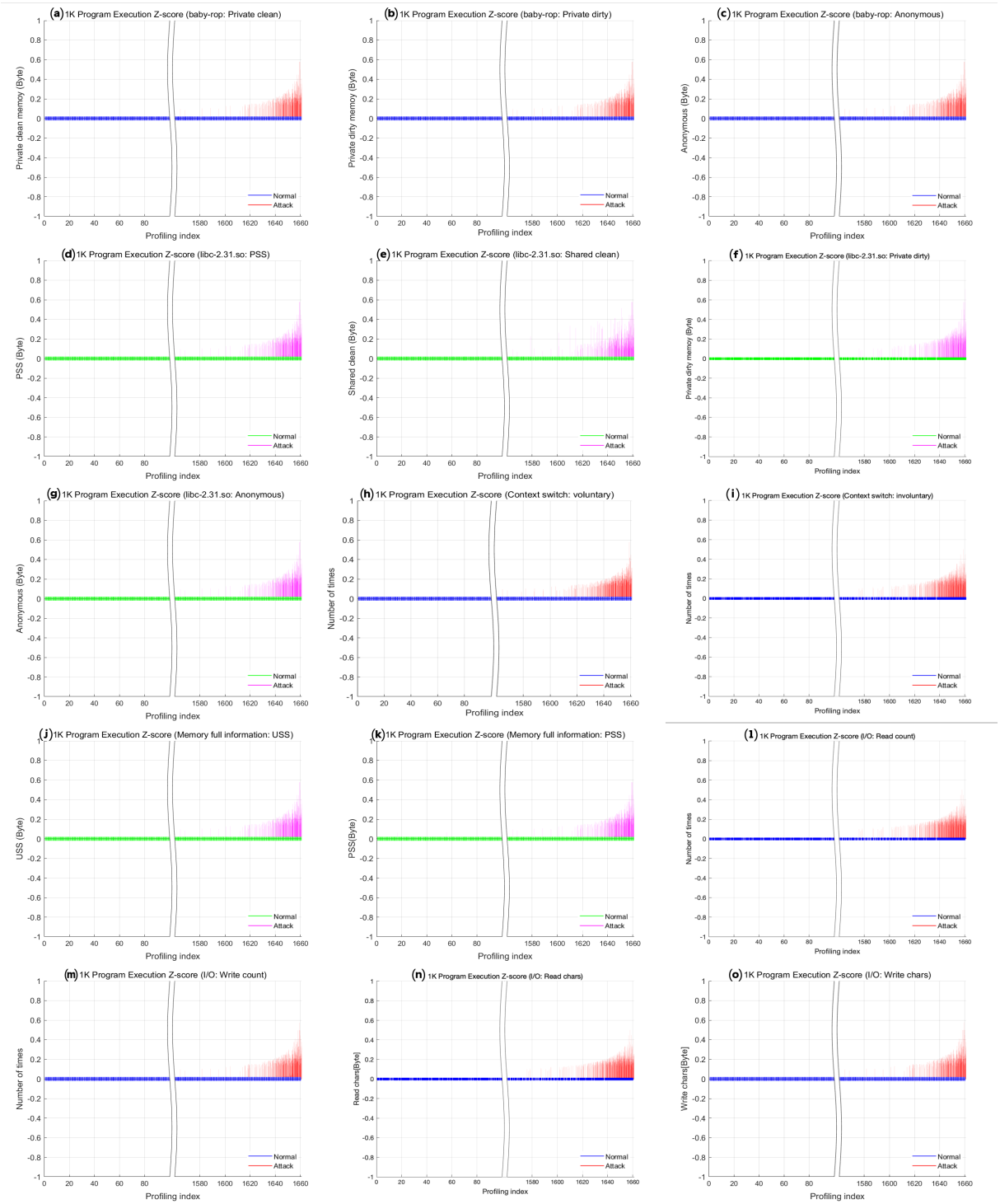

그림 1 System side channel information의 15가지 System utilization 항목에 따른 변동 추이

그림 1-d는 libc-2.31.so의 PSS이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 전체 공유 메모리 사이즈가 특이 변동했다. 그림 1-e는 libc-2.31.so의 Shared clean이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 수정되지 않았던 공유 메모리 사이즈가 특이 변동했다. 그림 1-f는 libc-2.31.so의 Private dirty이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 수정한 전용 메모리 사이즈가 특이 변동했다. 그림 1-g는 libc-2.31.so의 Anonymous이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 가상 메모리 사이즈가 특이 변동했다. 그림 1-h는 Context switch-voluntary이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스가 자체적으로 수행한 터미널과 메모리 I/O에 따른 Context switch 수가 특이 변동했다. 그림 1-i는 Context switch-involuntary이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스가 운영체제에 의해 강제적으로 수행한 터미널과 메모리 I/O에 따른 Context switch 수가 특이 변동했다. 그림 1-j는 전체 메모리 정보 가운데 USS이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스에 할당된 고유한 메모리 사이즈가 특이 변동했다. 그림 1-k는 전체 메모리 정보 가운데 PSS이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스에 할당된 공유 메모리 사이즈가 특이 변동했다. 그림 1-l는 프로세스 I/O 정보 가운데 Read count이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스의 메모리 I/O를 포함한 전체 I/O 읽기 수가 특이 변동했다. 그림 1-m은 프로세스 I/O 정보 가운데 Write count이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스의 메모리 I/O를 포함한 전체 I/O 쓰기 수가 특이 변동했다. 그림 1-n은 프로세스 I/O 정보 가운데 Read chars이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스의 메모리 I/O를 포함한 전체 I/O가 읽은 메모리양이 특이 변동했다. 그림 1-o는 프로세스 I/O 정보 가운데 Write chars이다. Profiling 후반부의 공격 발생 전, 후를 비교하면 정상 프로세스를 실행하는 동안 프로세스의 메모리 I/O를 포함한 전체 I/O가 쓴 메모리양이 특이 변동했다.

특이 변동 메모리 특성의 실험 결과 분석은 다음과 같다. 첫 번째, 실제 프로그램이 실행되는 동안 공격 발생으로 프로세스가 사용하는 메모리양이 증가함을 보인다. 실제 올라온 데이터 외에 추가적인 제어 흐름으로 인해 필요 데이터가 적재됐음을 알 수 있다. 두 번째, 공격 발생으로 프로세스의 Context switch가 증가함을 볼 수 있다. 정상 프로세스 수행에 비해 필요로 하는 메모리 I/O가 빈번히 발생하여 프로세스 전환이 많이 발생했음을 알 수 있다. 세 번째로, 메모리 I/O 카운터가 공격 발생으로 증가함을 보인다. 메모리에 대한 I/O 작업이 프로그램의 제어 흐름 변화로 인해 프로세스의 메모리 접근이 증가했음을 알 수 있다. 이러한 System side channel information의 특이 변동은 공격 패턴 탐지에 활용될 수 있으며, 이를 통해 Control-flow attack을 탐지하는 기법으로 사용될 것으로 기대한다.

5. Conclusion

본 논문에서는 Control-flow attack에 따른 System utilization profiling 결과인 System side channel information이 공격 발생 전, 후에 차이가 있음을 실험을 통해 보였다. 특히, 프로그램의 제어 흐름 변화는 프로그램의 System side channel information의 정상 패턴과 비교하여 특이 패턴을 나타냄을 확인했으며, 특이 패턴 분석을 통해 Control-flow attack을 탐지했다. 이를 통해, 본 논문은 Control-flow attack에 있어 System side channel information의 특이 패턴 기반 탐지 기법이 실제 산업에 효용성이 높을 것으로 기대한다.

향후, 본 논문은 시뮬레이션 플랫폼을 통해 보다 Atomic한 System utilization 측정 연구를 통해 보다 심도 깊은 공격과의 연관성을 연구할 것이며, 제안된 기법을 적용한 시스템을 개발하여 기존 시스템과 비교를 통해 제안 기법의 성능 및 오버헤드에 대한 면밀한 실험을 진행할 예정이다.

[참고문헌]

[1] Göktas, Enes, et al. "Out of control: Overcoming control-flow integrity." 2014 IEEE Symposium on Security and Privacy. IEEE, 2014.

[2] Song, Wei, et al. "Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It." arXiv preprint arXiv: 2008.01957 (2020).

[3] Percival, Colin. "Cache missing for fun and profit." (2005): 1-13.

[4] Osvik, Dag Arne, Adi Shamir, and Eran Tromer. "Cache attacks and countermeasures: the case of AES." Cryptographers' track at the RSA conference. Springer, Berlin, Heidelberg, 2006.

[5] Liu, Fangfei, et al. "Last-level cache side-channel attacks are practical." 2015 IEEE symposium on security and privacy. IEEE, 2015.

[6] Szekeres, Laszlo, et al. "Sok: Eternal war in memory." 2013 IEEE Symposium on Security and Privacy. IEEE, 2013.

[7] Preeth, E. N., et al. "Evaluation of Docker containers based on hardware utilization." 2015 international conference on control communication & computing India (ICCC). IEEE, 2015.

[8] Roemer, Ryan, et al. "Return-oriented programming: Systems, languages, and applications." ACM Transactions on Information and System Security (TISSEC) 15.1 (2012): 1-34.