트램폴린 코드 기반의 난독화 기법을 위한 역난독화 시스템

이광열*, 김민호*, 조해현**, 이정현**

*숭실대학교 (대학원생) **숭실대학교 (교수)

Deobfuscated Scheme for Obfuscation Techniques based on Trampoline Code

Gwangyeol Lee*, Minho Kim*, Haehyun Cho**, Jeonghyun Yi**

*Soongsil University (Graduate student)

**Soongsil University (Professor)

요 약

악성코드 분석가들은 악성코드가 다양한 경로로 배포되는 것을 분석하고 대응하기 위해 많은 노력을 기울이고 있다. 그러나 악성코드 개발자들은 악성코드가 분석되는 것을 회피하기 위해 다양한 시도를 하고 있다. 이 중에서 패킹과 난독화 기법은 대표적인 방법이다. 패킹 및 난독화 기법이 적용된 악성코드를 분석하는 것은 많은 어려움을 겪게 된다. 기존의 연구들은 보편적인 프로그램 언패킹 방법을 제안했지만, Original Entry Point 난독화, API 난독화와 같은 기법들을 해결하지 못해 언패킹이 실패하는 사례가 여전히 발견되고 있다.

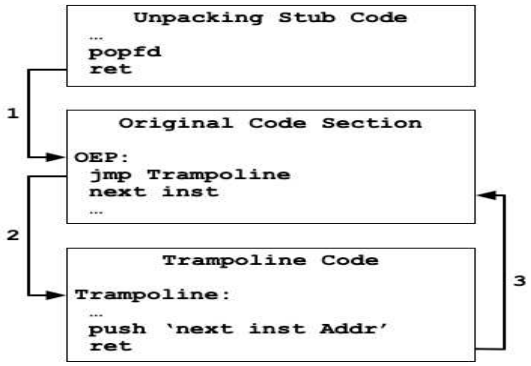
본 논문에서는 다양한 프로텍터들이 제공하는 OEP와 API 난독화 기법을 분석했다. 그리고 이러한 난독화 기법들을 자동으로 역난독화하는 시스템을 설계하고 구현했다. 제안된 시스템은 패킹된 프로그램의 메모리를 덤프하여 OEP와 API 난독화에 사용되는 트램폴린 코드를 탐지한다. 트램폴린 코드의 실행을 모니터링하여 난독화된 정보를 식별하고, 언패킹 과정에서 프로그램을 재구성하는데 활용한다. 실험을 통해 제안된 시스템이 OEP와 API 난독화 기법이 적용된 프로그램을 대상으로 효과적으로 언패킹 및 역난독화가 가능함을 증명했다.

I. 서론

보안 분야에서는 계속해서 다양한 문제들이 발견되고 있으며, 이에 대응하기 위해 많은 전문가들이 해결책을 모색하고 있다. 대표적인 문제점은 악성코드로 서버부터 개인용 컴퓨터, 모바일 기기까지 다양한 장치들을 감염시켜 심각한 피해를 초래하고 있다 [4]. 악성코드를 분석하기 위한 다양한 방법이 제안되었지만, 악성코드 개발자들은 분석 과정을 회피하기 위해 다양한 방법을 시도하고 있다. 이러한 시도 중에는 패커의 분석 방지 기법과 난독화 기법을 적용하는 방법이 있다. 이를 통해, 코드와 데이터를 보호하는 프로그램으로 악성코드에서도 널리 사용되고 있다. 악성코드는 패킹, 암호화, 분석방지 기법 등을 통해 기존의 디버거나 샌드박스 기반의 분석 환경에서의 실행을 피하고 있다 [8].

악성코드를 보호하는 다양한 기술에 대응하기 위해 언패킹과 역난독화 연구가 진행되고 있다. 기존 연구들은 임포트 테이블을 난독화하는 일부 기법을 해결하고 재구성하는데 초점을 두고 있다 [5]. 그러나 패커는 트램폴린 코드를 삽입하여 OEP 및 임포트 테이블과 같은 중요한 정보를 보호하고 있다 [8]. 따라서, 이 논문에서는 최신 패커가 OEP 및 임포트 테이블을 보호하기 위해 사용하는 난독화 기법을 분석하고, 이를 자동화하여 언패킹 및 역난독화를 효과적으로 수행할 수 있는 시스템을 제안한다.

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2017-0-00168, 사이버 위협 대응을 위한 Deep Malware 자동 분석 기술 개발)



[그림 1] OEP 난독화된 프로그램의 실행 흐름

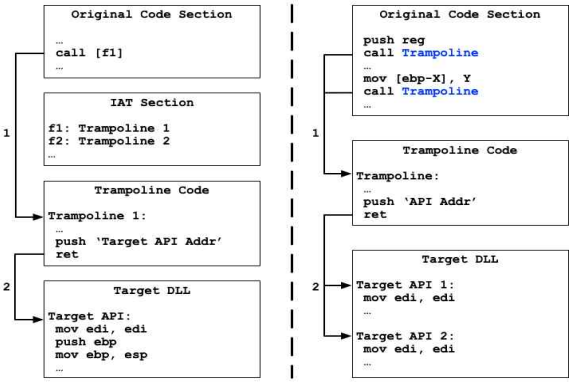
II. 배경 지식

2.1 OEP 난독화 기법

패커는 원본 프로그램에 언패킹 스텁 코드를 삽입하고, PE 헤더에서 엔트리 포인터 위치 등을 수정한다. 이로 인해, 패킹된 프로그램은 먼저 패커의 언패킹 스텁 코드를 실행하고 메모리에 원본 코드와 데이터를 복원한다 [9]. OEP는 원본 프로그램의 엔트리 포인트로, 복원된 원본 코드에서 처음 실행되는 명령어를 의미한다. OEP 난독화 기법은 패커가 메모리에 원본 코드를 복원할 때, OEP 명령어를 트램폴린 코드로 분기하도록 패치한다. 트램폴린 코드는 런타임에 다음 원본 코드의 주소를 스택 메모리에 저장하고 복귀 명령어를 호출한다. 복원된 코드의 실행 흐름이 복잡해지고 기능적으로는 동일한 코드를 실행한다. 그 결과, 기존의 언패커에 의해 언패킹된 프로그램의 OEP에는 트램폴린 코드 분기 명령어가 남게 된다. 언패킹된 프로그램은 [그림 1]과 같이 실행 즉시 잘못된 메모리 영역으로 분기하므로 정상 실행이 불가능한 문제가 발생한다.

2.2 API 난독화 기법

언패커는 패킹된 프로그램의 메모리 내에서 IAT 위치를 탐색하고, API의 주소를 식별하여 임포트 테이블을 복구합니다. 그러나, API 난독화 기법은 IAT 위치에 API 주소 대신 트램폴린 코드의 주소를 기록한다. 트램폴린 코드는 패커에 따라 언패킹 스텁 코드 영역이나 힙 메모리 영역에 위치한다. 트램폴린 코드는 분석 방해를 위한 의미 없는 코드나 데드 코드 등이



[그림 2] API 난독화된 프로그램의 실행 흐름

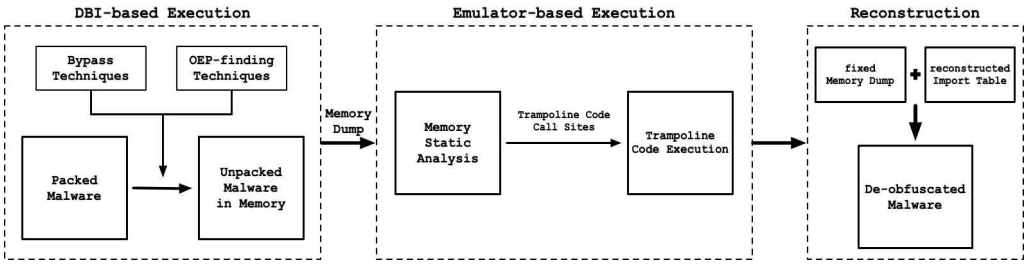
삽입되어 있으며, 실행 중에 난독화된 API 주소를 복호화하여 스택 메모리에 기록하고 복귀 명령어를 실행하여 API 코드를 호출한다 [5]. API 난독화 기법은 [그림 2]와 같이 트램폴린 코드와 난독화된 API가 일대일 대응하는 전달 인자에 민감하지 않은 형태와 일대다 대응하는 전달 인자에 민감한 형태로 구분된다. 일대일로 대응하는 경우는 Themida [1], VMProtect [2] 등의 프로텍터에서 사용하며, 모든 트램폴린 코드의 실행 흐름을 모니터링하면 난독화된 API를 식별할 수 있다 [5]. 반면에, 일대다로 대응하는 경우는 ASProtect [3] 등에서 사용하며, 다수의 난독화된 API가 공통의 트램폴린 코드를 사용한다. 트램폴린 코드 호출 이전에 실행되는 스택 메모리에 영향을 주는 명령어에 따라 복호화하는 API 주소가 다르다. 트램폴린 코드만을 식별하고 실행을 모니터링하는 기존의 역난독화 방식은 전달 인자에 민감한 형태의 API 난독화 기법을 해결하지 못하는 한계점이 있다.

III. 제안 기법

본 논문에서는 동적 바이너리 계측 도구인 Intel Pin [6]과 Unicorn CPU 에뮬레이터 [7]를 활용하여 OEP 및 API 난독화 기법을 역난독화하는 시스템을 제안한다. 본 논문에서 제안하는 시스템은 [그림 3]과 동일하다.

3.1 프로세스 메모리 덤프

OEP 및 API 난독화에 사용되는 트램폴린 코드는 원본 코드와 데이터는 메모리에 복원된



[그림 3] 제안 시스템 개요

후에 식별 가능하다. 원본 코드와 데이터가 복원되는 시점을 탐지하기 위해 패킹된 프로그램을 실행한다. 원본 코드와 데이터가 복원되는 시점은 언패킹 스텁 코드에 의해 메모리 쓰기가 이루어지는 영역의 명령어가 실행되는 시점(Written-then-Execute)을 의미한다. 원본 코드와 데이터가 모두 복원되면 프로세스 메모리를 전체적으로 덤프한다.

3.2 트램폴린 코드 실행 분석

메모리 덤프 파일은 원본 코드와 데이터가 복원된 상태이며, 패커의 OEP와 API 난독화 기법이 적용된 프로그램에서는 트램폴린 코드를 호출하는 분기 명령어가 존재한다. 트램폴린 코드는 원본 코드 섹션의 다양한 영역에 존재하며, 실행할 코드의 주소를 스택 메모리에 저장한다. 트램폴린 코드를 복귀 명령어까지 실행 후 스택 메모리의 최상단에 저장된 값을 통해 원본 코드의 주소인지 아니면 API인지를 구분하여 난독화된 정보를 식별할 수 있다.

3.3 원본 코드 및 데이터 복원

트램폴린 코드의 실행을 분석하여 난독화된 정보를 모두 식별할 수 있다. 이러한 식별된 정보를 기반으로, PE 헤더를 수정하고 임포트 테이블을 재구축하여 언패킹된 프로그램을 생성할 수 있다.

IV. 실험

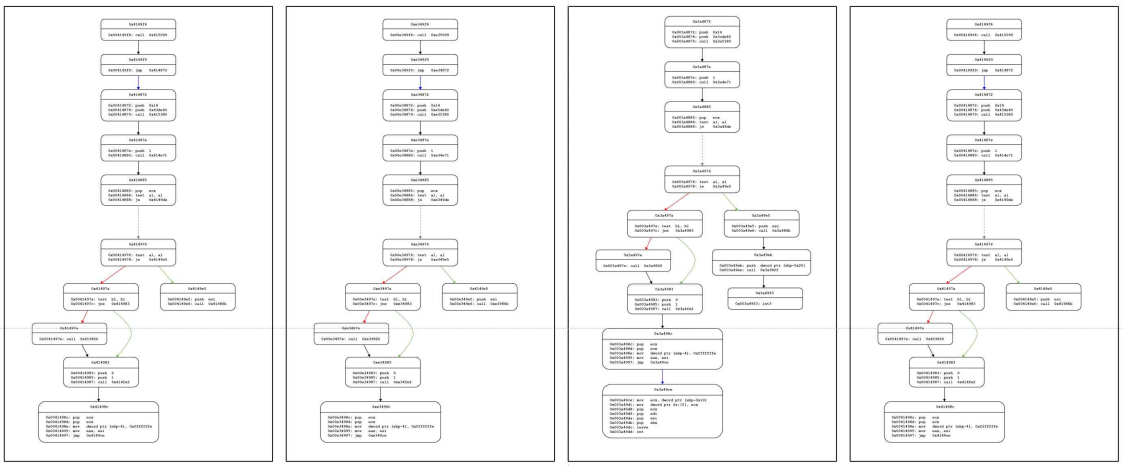
본 논문에서는 다양한 악성코드에서 사용되는 패커의 OEP와 API 난독화 기법에 대한 역난독화 시스템의 성능을 검증하기 위한 실험을 진행한다. 실험에 사용한 패커는 Themida v3.0.7, VMProtect v3.0.9, ASProtect v2.78으로 Juilet Test Suite 데이터셋에서 임의로 선택한 샘플 프로그램을 대상으로 OEP 및 API 난독화 기법을 적용하고 역난독화를 진행했다.

[표 1] API 난독화 기법에 대한 역난독화 결과

패커	난독화된 API 갯수	역난독화된 API 갯수
Themida v3.0.7	69/83	69/69
VMProtect v3.0.9	83/83	83/83
ASProtect v2.78	26/83	26/26

기존의 언패커로 OEP 난독화가 적용된 프로그램을 언패킹하면 OEP에 트램폴린 코드 분기 명령어가 존재하기 때문에 실행을 시도하면 에러가 발생한다. [그림 4]는 Themida, VMProtect, ASProtect에 의해 난독화된 프로그램을 OEP의 트램폴린 코드를 분석과 패치를 하여 재구성된 프로그램의 Control Flow Graph (CFG)를 추출하여 원본 프로그램과 비교한 모습이다. 이를 통해, 역난독화된 프로그램의 CFG가 원본 프로그램과 동일한 것을 확인했다.

[표 1]은 실험에 사용한 프로그램을 대상으로 패커에서 제공하는 API 난독화 기법을 각각 적용하고, 제안 시스템을 통해 역난독화한 결과를 보여준다. Themida, VMProtect, ASProtect는 모두 다른 방식으로 API 난독화를 하는 경우로, 83개의 API 중 69개, 83개, 26개를 대상으로 난독화했다. 제안 시스템은 원본 코드 섹션에 존재하는 모든 트램폴린 코드 호출 명령어가 포함된 기본 블록을 기준으로 분석한다. 이로 인해, 기존에 일대일 형태의 트램폴린 코드뿐만 아니라 ASProtect와 같이 다대일 형태의 트램폴린 코드를 사용하는 경우도 역난독화가 가능해진다.



[그림 4] 프로그램의 CFG 추출 결과 비교:
원본, Themida, VMProtect, ASProtect

V. 결론

최신의 역난독화 연구에서는 임포트 테이블을 재구성을 방해하는 API 난독화 기법에 대한 연구가 진행되었다. 그러나 다대일 형태의 트램폴린 코드를 사용하는 API 난독화 기법이 난독화한 정보를 식별하지 못하고, OEP 난독화가 적용된 악성코드의 언패킹이 실패한 한계점이 있다. 이에 본 논문에서는 다양한 프로텍터에서 사용되는 OEP 난독화 기법과 API 난독화 기법에 대해 분석하고, 자동으로 역난독화하는 시스템을 제안했다.

본 논문의 제안 시스템은 OEP 및 API 난독화 기법이 적용된 악성코드를 동적 바이너리 계측 도구와 에뮬레이터를 사용하여 트램폴린 코드가 난독화한 정보를 분석하여 다대일 형태의 트램폴린 코드를 사용하는 난독화 기법도 무력화되는 것을 확인했다. 이는 트램폴린 코드를 통해 코드 및 데이터를 보호하는 악성코드도 역난독화 및 언패킹이 가능하다는 것을 의미한다. 그러나 프로텍터는 OEP와 임포트 테이블 외에 다양한 정보를 난독화하는 기법을 제공하기 때문에 여전히 언패킹에 실패하는 악성코드가 발견될 수 있다. 추후 연구는 다양한 난독화 기법을 분석하고 이를 역난독화하기 위한 연구를 진행할 예정이다.

[참고문헌]

[1] Themida[Website]. (2023 Feb 06). https://www.oreans.com/Themida.php/.

[2] VMProtect[Website]. (2023 Feb 06). https://vmpsoft.com/.

[3] ASProtect[Website]. (2023 Feb 06). http://www.aspack.com/asprotect32.html.

[4] AV-TEST Malware Statistics[Website]. (2023 Apr 24). https://www.av-test.org/en/statistics/malware/.

[5] Cheng, Binlin, et al. "Obfuscation-Resilient Executable Payload Extraction From Packed Malware." 30th USENIX Security Symposium (USENIX Security 21). 2021.

[6] Pin[Website]. (2023 Jan 25). https://software.intel.com/sites/landingpage/pintool/docs/98579/Pin/doc/html/index.html.

[7] Unicorn[Website]. (2023 Jan 29). https://www.unicorn-engine.org/.

[8] Kevin A. Roundy and Barton P. Miller. “Binary-code Obfuscations in Prevalent Packer Tools.” ACM Computing Surveys, 46(1), 2013.