언패킹을 위한 OEP 및 API 난독화 기법에 대한 역난독화 방법 연구

이광열*, 김민호*, 이정현**, 조해현**

*,**숭실대학교 (대학원생, 교수)

Research on OEP and API De-obfuscation Methods for Unpacking

Gwangyeol Lee*, Minho Kim*, Jeong Hyun Yi**, Haehyun Cho**

*,**Soongsil University (Graduate student, Professor)

요 약

악성코드 분석가들은 다양한 경로로 유포되는 악성코드를 분석하고 대응하는 데 많은 노력을 기울이고 있다. 하지만, 악성코드 제작자들은 분석을 회피하기 위해 여러 방법을 시도하고 있다. 이 중 패킹과 난독화 기법이 대표적이다. 기존의 연구들은 다양한 언패킹 방법을 제시했지만, 패커들이 사용하는 OEP 난독화 및 API 난독화 기법에 대한 대응이 부족해 언패킹 과정에서 실패하는 경우가 종종 발생하고 있다.

본 논문에서는 다양한 패커들이 사용하는 OEP 난독화와 API 난독화 기법을 분석하고, 이를 자동으로 역난독화하는 방법을 제시한다. 이 방법은 패킹된 프로그램을 메모리에서 덤프한 후, OEP 난독화와 API 난독화에 사용되는 트램폴린 코드를 식별한다. 이어서 트램폴린 코드의 패턴을 분석하여 난독화된 정보를 탐지하고, 이를 바탕으로 언패킹 및 역난독화된 프로그램을 제구성한다. 실험을 통해, 제안된 방법이 OEP 난독화와 API 난독화 기법을 적용한 프로그램에 대해 효과적으로 언패킹 및 역난독화를 수행할 수 있음을 확인했다.

I. 서론

컴퓨터 보안 분야는 끊임없이 변화하는 도전 과제들에 직면해 있으며, 이를 해결하기 위해 보안 전문가들은 지속해서 새로운 대처방안을 모색하고 있다. 많은 문제들 중에서도 악성코드는 네트워크의 서버부터 일반 가정의 컴퓨터, 스마트폰에 이르기까지 광범위한 기기들에 침투해 큰 피해를 일으키는 주된 위협 요소로 자리 잡고 있다 [1]. 악성코드 탐지 및 분석을 위한 다양한 방법론이 제시되고 있지만, 악성코드 제작자들은 분석을 회피하기 위해 더욱 정교한 기술들을 사용하고 있다 [2]. 특히, 패커를 이용한 난독화 기법은 프로그램의 중요 코드나 데이터를 숨기는 데에 자주 사용되는 방법으로 악성코드에서도 널리 사용되고 있다.

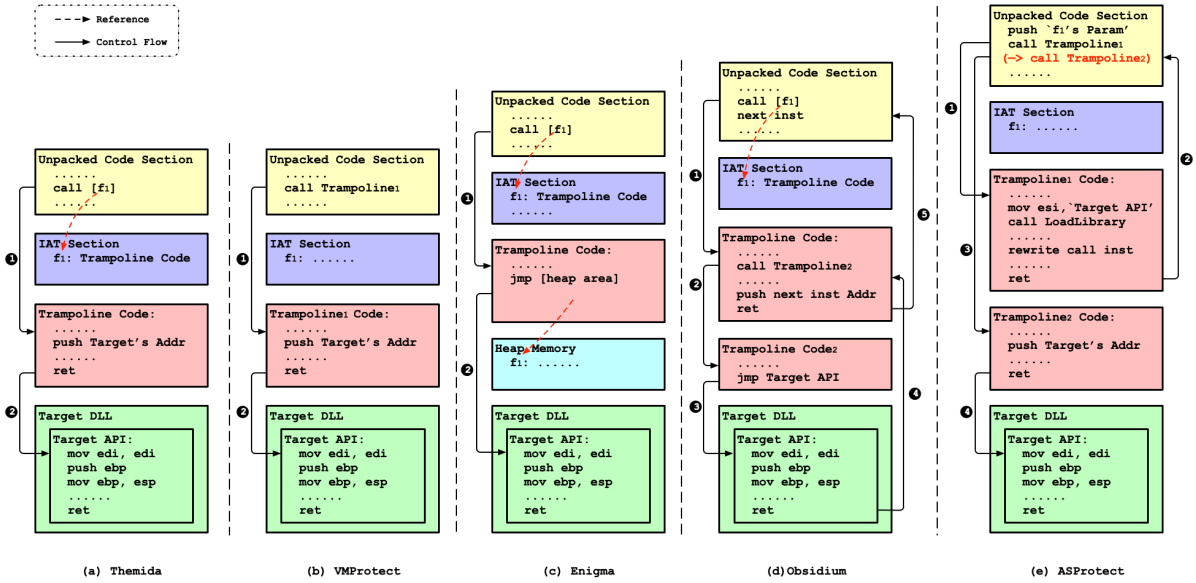
역난독화와 언패킹에 관한 연구는 악성코드가 다양한 보호 기술에 대응하여 진행되고 있다. 기존의 연구들은 주로 원본 진입점(Original Entry Point, OEP)을 찾아내고 임포트 테이블을 재구성하는 데 초점을 두고 있으나 [6, 7], 패커들은 난독화 기법을 이용해 중요 정보인 OEP와 임포트 테이블을 숨기기 위해 트램폴린 코드를 프로그램 안에 삽입한다. 본 논문에서는 최신 패커들이 어떻게 OEP와 임포트 테이블을 난독화하여 보호하는지 분석하고, 이 과정을 자동화하여 효율적으로 역난독화 및 언패킹을 할 수 있는 시스템을 제안한다.

II. 배경지식

2.1 API 난독화 기법

API 난독화 기법은 기존의 임포트 테이블 재구성 방식을 방해한다. 이 기법은 프로그램이 실행되는 동안 API 호출 명령어나 IAT 내의 API 주소를 변경하여 API 호출 과정을 복잡하

* 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)

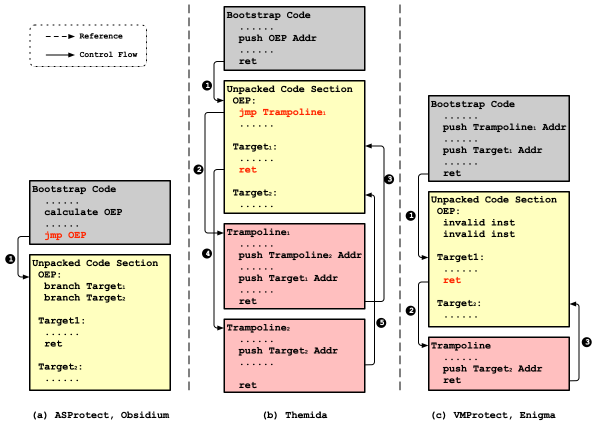


[그림 1] API 난독화 기법 분석 결과

게 만들기 위해 트램폴린 코드를 활용한다. 트램폴린 코드는 분석을 어렵게 만들기 위해 복잡한 실행 흐름을 생성하고, 난독화된 API 주소를 계산하고 트램폴린 코드의 실행이 끝나면 해당 API를 호출하도록 설계한다. API 주소 계산 과정은 트램폴린 코드 호출 직전에 제공된 인자에 의해 영향을 받거나 받지 않을 수 있다. [그림 1]에 나타난 Themida, VMProtect, Enigma, Obsidium과 같은 대부분의 패커들은 인자에 영향을 받지 않는 트램폴린 코드를 사용한다. API 주소 계산은 독립적으로 이루어지며, 각 난독화된 API는 고유한 트램폴린 코드를 호출한다. 따라서, 모든 트램폴린 코드를 분석하여 역난독화된 API 목록을 추출해야 한다. 반면, [그림 1]에 나타난 ASProtect와 같은 일부 패커들은 인자에 영향을 받는 트램폴린 코드를 사용한다. 단일 트램폴린 코드가 모든 난독화된 API의 호출을 담당한다. API 주소 계산은 제공된 인자에 따라 달라진다. 그런 다음, 트램폴린 코드는 인자에 영향을 받지 않는 트램폴린 코드를 실행한다. 이러한 특성 때문에, 인자에 민감한 트램폴린 코드를 단순히 강제 실행하여 모니터링하는 것만으로는 API 주소 계산이 제대로 이루어지지 않아, 기존 역난독화 방식으로는 한계가 있다 [3].

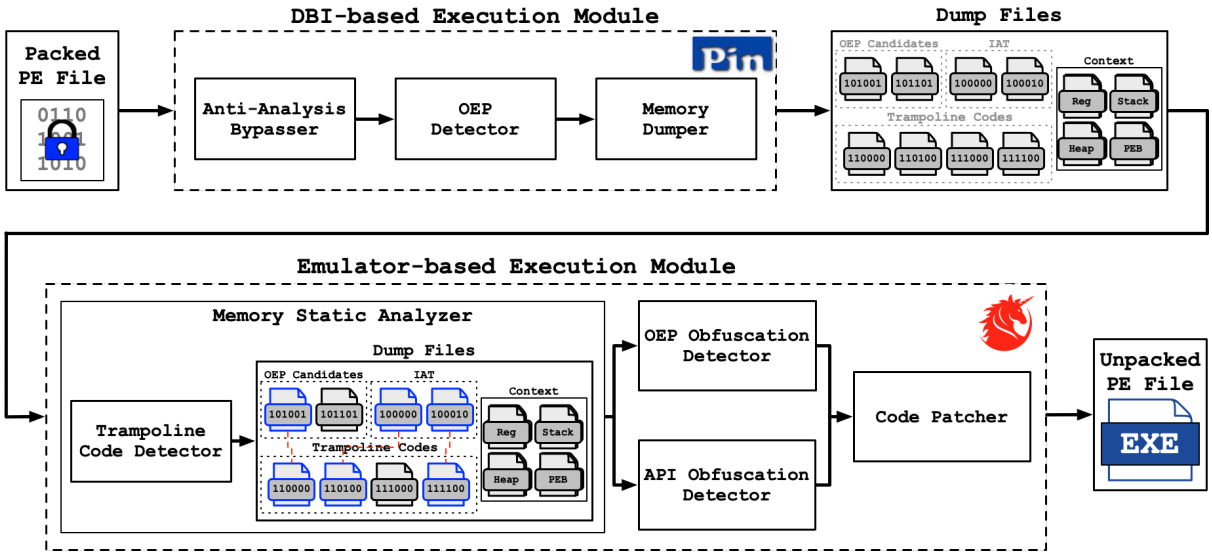
2.2 OEP 난독화 기법

기존의 언패킹 방식을 무력화하기 위해, 많은



[그림 2] OEP 난독화 기법 분석 결과

패커들은 OEP 난독화 기법을 사용한다. 이 기법은 트램폴린 코드를 활용하는 방식과 그렇지 않은 방식으로 나뉜다. 트램폴린 코드를 활용하는 방식은 OEP 주변의 명령어들이 제거되고, 이러한 명령어들은 실행 시간에 패커의 부트스트랩 코드나 트램폴린 코드에 의해 실행된다. [그림 2]에 볼 수 있듯이, Themida는 OEP에서 트램폴린 코드를 호출하여 실행되며, VMProtect와 Enigma는 부트스트랩 코드가 *Target 1*을 실행시킨 후, 실행 시간에 트램폴린 코드를 호출해 정상적인 실행 흐름인 *Target 2*로 분기한다. 이러한 실행 과정은 기존의 언패커들이 정확한 OEP 주소를 찾는 데 있어 한계를 드러낼 수 있다 [4, 5]. 결과적으로, 언패킹된 프로그램이 부적절한 진입점 주소를 가짐으로



[그림 3] 제안 시스템 개요

써 실행 오류가 발생할 위험이 있다.

III. 제안 시스템

본 논문에서는 동적 바이너리 계측 도구인 Intel Pin과 CPU 에뮬레이터인 Unicorn을 사용하여 API 난독화와 OEP 난독화 기법을 역난독화하는 방법을 제안한다. [그림 3]은 제안하는 역난독화 시스템의 동작 흐름을 도식화한 것이다. 제안 방법은 난독화된 악성코드를 계측하여 메모리에 일시적으로 복원된 프로그램을 덤프하고, 트램폴린 코드를 통해 난독화된 정보를 분석한다. 이를 위해 트램폴린 코드를 독립적으로 에뮬레이트하고, 그 분석 결과를 바탕으로 언패킹된 프로그램을 재구성한다.

3.1 프로세스 메모리 덤프

OEP 난독화 및 API 난독화 기법은 프로그램이 메모리에 일시적으로 복원되는 실행 시간 시점에 탐지 가능하다. Intel Pin을 사용하여 난독화된 프로그램을 실행시키면, 원본 프로그램이 메모리에 복원되는 순간을 포착하고 해당 시점의 메모리를 덤프한다.

3.2 트램폴린 코드 실행 분석

OEP 및 API 난독화를 적용한 프로그램은 원본 코드와 트램폴린 코드 호출 명령어를 덤프 파일 안에 포함한다. OEP 난독화는 OEP 위치에 있는 트램폴린 코드가 실행되거나, 실행 시간에 스택에 저장된 트램폴린 코드 주소가 실행되도록 한다. API 난독화는 트램폴린 코드를 통해 API 호출을 수행한다. 트램폴린 코드는 난독화된 주소를 계산하고, 복호화된 주소를 스택에 저장한 후, 복호화된 주소가 스택 최상단에 도달하면 리턴 명령어를 실행하여 해당 주소로 분기한다.

트램폴린 코드 실행 단계에서는 덤프 파일에 있는 모든 트램폴린 코드를 에뮬레이트하여, 트램폴린 코드가 최종적으로 분기하는 주소를 찾아낸다. 이렇게 탐지된 주소를 통해 OEP 난독화 또는 API 난독화에 사용된 트램폴린 코드를 구분할 수 있다.

3.3 원본 프로그램 재구성

원본 프로그램 재구성 단계는 패킹과 난독화 기법으로 변형된 프로그램의 정보를 복원하는 과정이다. 기존 언패커들과 유사하게 덤프 파일의 프로그램 헤더에 있는 진입점에 OEP 주소를 기록하고, 재구성된 임포트 테이블을 추가하여 언패킹 된 프로그램을 생성한다.

IV. 실험

본 실험에서 사용된 패커는 Themida, VMProtect, ASProtect, Enigma, 그리고 Obsidium이며, OEP 난독화 및 API 난독화 기법을 적용한 샘플 프로그램들을 역난독화하는 과정을 진행했다. [표 1]에는 각 패커별 역난독화 결과가 요약되어 있다.

ASProtect와 Obsidium은 OEP 난독화에 트램폴린 코드를 사용하지 않는다 (–). 반면에 Themida는 OEP 부분에서 트램폴린 코드 호출을 사용한다 (✓). VMProtect와 Enigma는 원본 프로그램의 코드에 진입한 이후, 런타임에 스택에 저장된 트램폴린 코드 주소를 통해 해당 코드를 호출하는 방식을 사용한다 (△). API 난독화와 관련해서는, 모든 패커들이 트램폴린 코드 호출을 이용하고 있다 (✓). 특히, 인자에 민감한 트램폴린 코드를 사용하는 ASProtect의 경우에도, 본 논문에서 제안된 시스템이 난독화된 API를 정확히 식별하는 데 성공했다.

[표 1] 역난독화 결과 비교

	OEP obfuscation	API obfuscation
Themida	✓	✓
VMProtect	△	✓
ASProtect	–	✓
Enigma	△	✓
Obsidium	–	✓

V. 결론

최근의 역난독화 연구는 임포트 테이블의 재구성을 방해하는 API 난독화 기법에 초점을 맞추었으나, 인자에 민감한 트램폴린 코드를 활용하는 API 난독화 기법으로 인해 임포트 테이블의 재구성에 어려움을 겪었다. 또한, OEP 난독화 기법에 대한 연구는 상대적으로 부족한 상황이다. 이러한 문제를 해결하기 위해, 본 논문은 다양한 패커들이 사용하는 OEP 난독화 및 API 난독화 기법을 분석하고, 이를 효과적으로 역난독화하는 방법을 제안한다.

본 연구에서 제안하는 방법은 동적 바이너리 계측 도구와 에뮬레이터를 활용하여, OEP 난독화 및 API 난독화 기법이 적용된 프로그램에서 트램폴린 코드로 난독화된 정보를 분석한다. 이를 통해, 인자에 민감한 트램폴린 코드를 이용한 API 난독화 기법을 역난독화할 수 있으며, OEP 후보군 중 트램폴린 코드 호출의 존재 여부를 탐지하여 삭제된 OEP 명령어가 지시하는 분기 주소를 식별할 수 있다. 이는 트램폴린 코드를 활용하여 난독화된 프로그램을 역난독화하고 언패킹이 가능함을 시사한다. 그러나 아직 알려지지 않은 패커들의 다양한 난독화 기법이 존재하고, 이에 따라 악성코드 분석에는 여전히 도전이 따른다. 향후 연구에서는 다양한 악성코드 샘플을 분석하여 알려지지 않은 패커들의 난독화 기법을 파악하고, 이를 역난독화하는 방안을 모색할 계획이다.

[참고문헌]

[1] AV-TEST “Malware Statistics”, https://www.av-test.org/en/statistics/malware/“

[2] FireEye, “M-Trends 2020”, https://content.fireeye.com/m-trends/rpt-m-trends-2020

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

[4] Isawa, Ryoichi, Daisuke Inoue, and Koji Nakao. "An original entry point detection method with candidate-sorting for more effective generic unpacking." IEICE TRANSACTIONS on Information and Systems 98.4 (2015): 883-893.

[5] Kang, Min Gyung, Pongsin Poosankam, and Heng Yin. "Renovo: A hidden code extractor for packed executables." Proceedings of the 2007 ACM workshop on Recurring malcode. 2007.

[6] Kim, Gyeong Min, and Yong Su Park. "Improved Original Entry Point Detection Method Based on PinDemonium." KIPS Transactions on Computer and Communication Systems 7.6 (2018): 155-164.

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