Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.6, Dec. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.6.1043

트램폴린 코드 기반의 난독화 기법을 위한 역난독화 시스템*

김 민 호,[1†] 이 정 현,[2] 조 해 현[2‡]
[1,2]숭실대학교 (대학원생, 교수)

De-Obfuscated Scheme for Obfuscation Techniques Based on Trampoline Code*

Minho Kim,[1†] Jeong Hyun Yi,[2] Haehyun Cho[2‡]
[1,2]Soongsil University (Graduate student, Professor)

요 약

악성코드 분석가들은 다양한 경로로 배포되는 악성코드를 분석하고 대응하기 위해 많은 노력을 기울이고 있다. 그러나 악성코드 개발자들은 분석을 회피하기 위해 다양한 시도를 하고 있다. 대표적인 방법으로는 패킹과 난독화 기법 등이 있다. 기존 연구들은 일반적인 프로그램 언패킹 방법을 제안했으나, 최근의 패커들이 사용하는 OEP 난독화나 API 난독화 기법 등에 대한 대응이 부족하여, 언패킹 과정에서 실패하는 경우가 있다.

본 논문에서는 다양한 패커들이 사용하는 OEP 및 API 난독화 기법을 분석하고, 이를 자동으로 역난독화하는 시스템을 제안한다. 제안 시스템은 패킹된 프로그램의 메모리를 덤프하여 OEP와 API 난독화에 사용되는 트램폴린 코드를 탐지한다. 이후 트램폴린 코드의 패턴을 분석하여 난독화된 정보를 탐지하고, 언패킹된 프로그램으로 재구성한다. 실험 결과, 제안 시스템이 다양한 패커에 의해 OEP와 API 난독화 기법이 적용된 프로그램을 효과적으로 역난독화할 수 있음을 확인하였다.

ABSTRACT

Malware analysts work diligently to analyze and counteract malware, while developers persistently devise evasion tactics, notably through packing and obfuscation techniques. Although previous works have proposed general unpacking approaches, they inadequately address techniques like OEP obfuscation and API obfuscation employed by modern packers, leading to occasional failures during the unpacking process.

This paper examines the OEP and API obfuscation techniques utilized by various packers and introduces a system designed to automatically de-obfuscate them. The system analyzes the memory of packed programs, detects trampoline codes, and identifies obfuscated information, for program reconstruction. Experimental results demonstrate the effectiveness of our system in de-obfuscating programs that have undergone OEP and API obfuscation techniques.

Keywords: OEP obfuscation, API obfuscation, De-obfuscation

Received(10. 16. 2023), Modified(11. 27. 2023), Accepted(11. 27. 2023)

* 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)

† 주저자, mhkim37@soongsil.ac.kr
‡ 교신저자, haehyun@ssu.ac.kr(Corresponding author)

I. 서 론

컴퓨터 보안 분야는 지속적으로 다양한 문제를 직면하고 있으며, 이에 대응하기 위해 전문가들이 다양한 해결책을 모색하고 있다. 악성코드는 이 중 대표적인 문제로, 서버부터 개인용 컴퓨터, 모바일 기기에 이르기까지 다양한 장치를 감염시켜 심각한 피해를 야기한다[1]. 악성코드 분석 방법들이 다양하게 제안되고 있지만, 악성코드 개발자들은 분석 과정을 회피하기 위해 다양한 방법을 도입하고 있다. 이중 패커의 난독화 기법은 프로그램의 코드와 데이터의 일부를 보호하는 기법으로, 악성코드에서도 널리 활용되고 있다[2].

언패킹과 역난독화 연구는 악성코드에 적용된 다양한 보호 기술에 대응하기 위해 진행되고 있다. 기존 연구들은 주로 OEP 탐지와 임포트 테이블 재구성에 초점을 맞추고 있으나[6][7][18][19], 패커는 난독화 기법을 사용하여 프로그램에 트램폴린 코드를 삽입하여 OEP(Original Entry Point) 및 API와 같은 중요한 정보를 보호하고 있다[3][5]. 본 논문에서는 최신 패커가 OEP 및 API를 보호하기 위해 사용하는 난독화 기법을 분석하고, 이를 자동화하여 언패킹 및 역난독화를 효과적으로 수행할 수 있는 시스템을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 패킹된 프로그램 분석의 어려움, 그리고 OEP 및 API 난독화 기법에 대해 설명한다. 3장에서는 제안 시스템에 대해 설명하며, 4장에서는 역난독화 성능 평가를 진행한다. 마지막으로, 5장에서는 결론으로 마무리한다.

II. 배경지식

패커는 프로그램의 엔트리 포인트와 임포트 테이블을 제거한 다음, 원본 코드와 데이터 섹션을 압축한다. 기존의 언패킹 관련 연구는 대체로 패킹된 악성코드 샘플의 OEP를 탐지하고 임포트 테이블을 재구성하는데 중점을 둔다[6]. 그러나 많은 패커들은 다양한 분석 방지 기법 및 난독화 기법을 사용하여 패킹된 프로그램에 대한 분석이나 언패킹을 방해하고 있다. 이 분야에서 최신 연구인 API-Xray[3]는 패커의 API 난독화 기법을 분석하고, 난독화된 악성코드를 대상으로 임포트 테이블의 재구성을 성공했다. 그러나 임포트 테이블 재구성에 실패한 악성코드 중 9.5%는 OEP 난독화 기법으로 인해 실패했다고 보고했다.

비록 관련 연구들이 API 난독화 기법을 분석하고 해결하기 위한 방식을 제안했지만[3][4], 아직도 해결되지 않은 API 난독화 기법이 적용된 악성코드는 임포트 테이블 재구성을 복잡하게 만들고 있다. 더욱이, 패킹에 의해 숨겨진 OEP를 찾는 방법에 대해 많은 연구가 있었지만[7][8][13], OEP 난독화 기법에 대한 분석은 충분히 이루어지지 않고 있다.

본 논문은 악성코드 분석가가 패킹된 악성코드를 언패킹하는데 있어 OEP 및 API 난독화 기법이 초래하는 문제를 다룬다. 먼저, 패커가 프로그램을 패킹하는 과정에서 엔트리 포인트와 임포트 테이블에 미치는 영향에 대해 설명한다. 그 다음으로, 다양한 패커들의 API 및 OEP 난독화 기법에 대한 분석 결과를 제시한다.

2.1 일반적인 패커가 바이너리에 미치는 영향

Fig. 1.은 일반적인 PE 파일의 실행 흐름을 나타낸다. 윈도우 로더는 프로그램을 메모리에 로드하고 실행할 수 있도록 일련의 작업을 수행한다. 이 과정에서 주요 작업은 IAT(Import Address Table)에 실제 API 주소를 바인딩하고, 프로세스의 실행 흐름을 엔트리 포인트부터 시작하도록 만드

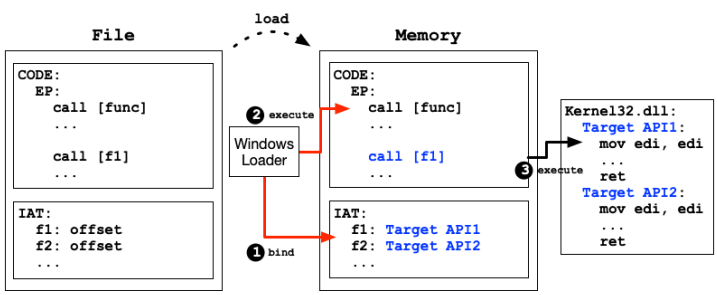

Fig. 1. The control flow of normal program.

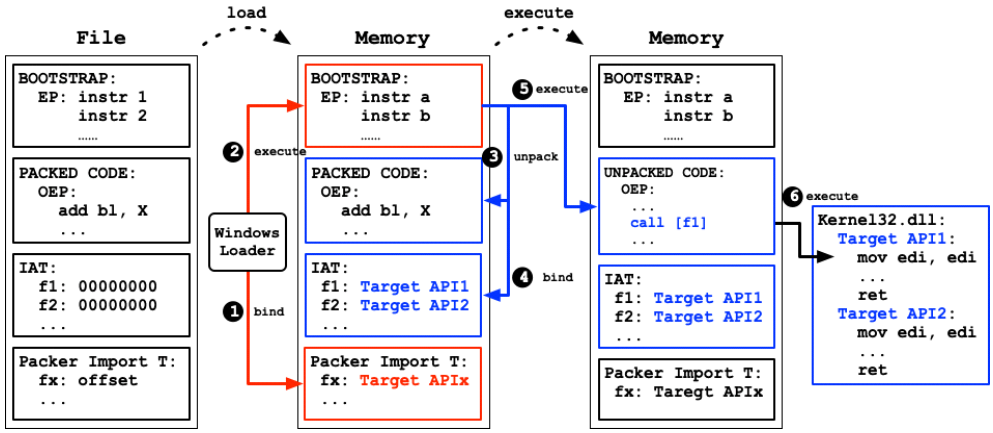

Fig. 2. The control flow of packed program.

는 것이다.

PE 파일은 여러 동적 라이브러리의 API를 호출하여 다양한 기능을 수행한다. API를 사용하기 위해 윈도우 로더는 임포트 테이블에 기록된 API 목록을 읽고 API 주소를 IAT에 기록해야 한다. 그런 다음, 프로세스가 API를 호출하면, IAT를 참조하여 실제 API 주소로 분기한다.

윈도우 로더는 프로그램 실행에 필요한 모든 작업을 완료하면, 프로그램의 실행 시작 지점을 엔트리 포인트로 지정한다. 엔트리 포인트는 프로그램의 실행이 시작되는 지점으로, 프로그램 실행에 필요한 초기화를 수행하고 여러 함수를 호출하여 프로그래머가 작성한 코드를 실행한다. 많은 패커들은 Fig. 2.와 같이 프로그램을 패킹할 때, 엔트리 포인트를 자신들의 부트스트랩 코드를 가리키도록 변경한다. 패킹된 프로그램을 실행하면, 먼저 부트스트랩 코드가 실행된다. 이 코드는 원본 프로그램을 메모리에 압축 해제하고, 라이브러리를 로드하며, 시스템 리소스를 할당한다. 이러한 초기 작업이 완료되면, 부트스트랩 코드는 압축 해제된 프로그램의 OEP를 호출한다.

2.2 API 난독화 기법

일반적으로 PE 프로그램을 API를 사용하기 위해 런타임시 API 주소를 IAT에 바인딩한다. 대다수의 패커는 패킹 과정에서 임포트 테이블의 정보를 제거하고, 런타임에만 IAT를 복원하여 사용한다. 패킹된 프로그램은 일반적으로 정적 분석 도구로는 임포트 테이블을 분석할 수 없다. 그러나 패킹된 프로그램이 메모리에 로드된 상태에서 바인딩 과정 이후 동적 분석 도구를 통해 분석이 가능하다.

임포트 테이블을 재구성하여 언패킹된 프로그램을 만드는 것을 방지하기 위해, 다양한 패커들은 API 난독화 기법을 도입한다. API 난독화 기법은 난독화된 프로그램의 실행 과정에서 원본 코드의 API 호출 명령어 혹은 호출 명령어가 참조하는 IAT의 정보를 변조하여 API 호출 과정을 복잡하게 만든다. 이 중 패커들은 실행 과정을 복잡하게 만들기 위해 트램폴린 코드를 활용하고 있다.

트램폴린 코드를 활용한 난독화 기법은 실행 로직을 복잡하게 만들어 분석 과정이 더 어렵고 시간이 오래 걸리게 하려는 목표를 가지고 있다. 'Dead Code Insertion', 'Constant Propagation', 'Constant Folding Obfuscation', 'Fake Constants' 및 'Indirect Jump'와 같은 기술들이 이 복잡성을 더욱 증가시킨다[9]. 트램폴린 코드는 주로 함수 호출을 링크하거나 리디렉션하는 역할로, API 호출 뿐만 아니라 OEP에 위치한 호출 명령어를 난독화하는데도 활용되고 있다. 그 중 API 호출 대신 트램폴린 코드를 호출하기 위한 방법은 크게 두 가지로, IAT를 통한 트램폴린 코드 간접 호출과 코드 패치를 통한 트램폴린 코드 직접 호출로 구분할 수 있다.

먼저, IAT를 통한 트램폴린 코드 간접 호출 방법은 Themida[10] 등의 패커에서 주로 사용되며, 메모리에 임시로 압축 해제된 원본 코드 명령어의 조작 없이, 부트스트랩 코드가 IAT에 트램폴린 코드의 주소를 바인딩한다. 압축 해제된 원본 코드에서 API를 호출하면 IAT에 기록된 트램폴린 코드를 호출한다.

다음으로, 코드 패치를 통한 트램폴린 코드 직접

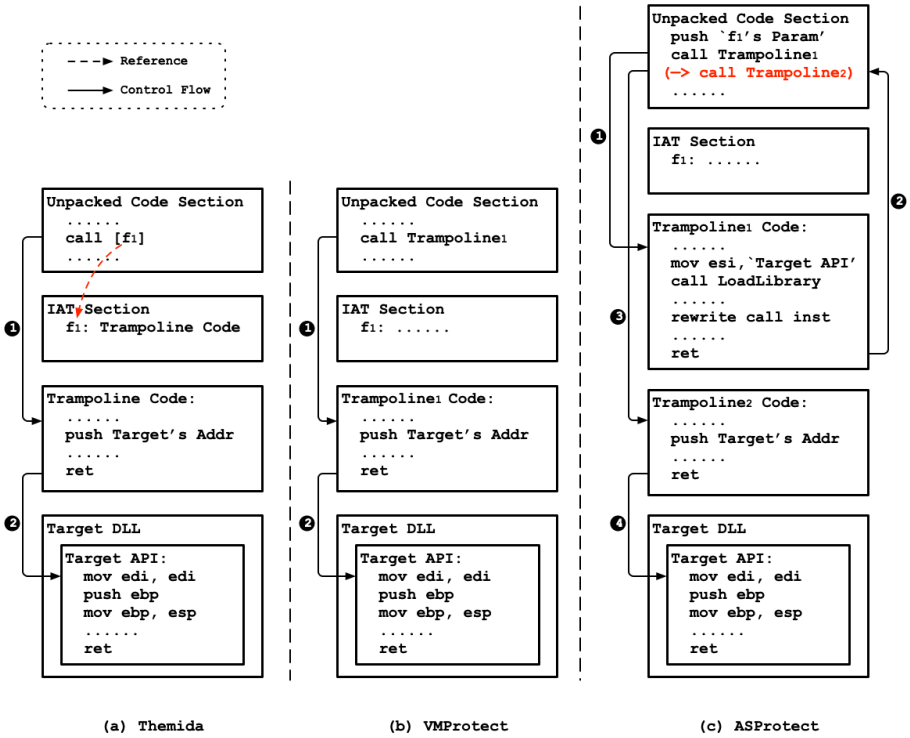

Fig. 3. The analysis results of API obfuscation techniques.

호출 방법은 주로 VMProtect[11]와 ASProtect[12]에서 사용되며, API 호출 명령어를 트램폴린 코드로 직접 분기하도록 패치한다. 난독화된 API는 IAT를 참조하지 않기 때문에 난독화되지 않은 일부 API 만이 IAT에 바인딩된다. 따라서, 압축 해제된 코드 내에 존재하는 트램폴린 코드 호출 명령어를 API를 호출하는 명령어로 복원하는 과정이 필요하다.

트램폴린 코드는 호출 방법에 관계없이, API 난독화에 사용되는 트램폴린 코드는 복잡한 실행 과정 끝에 각각의 API를 호출하여 원본 코드와 동일한 실행 결과를 보여준다. 트램폴린 코드는 분석을 방해하기 위한 여러 기법들과 복잡한 실행 과정 도중에 난독화된 API 주소를 연산하고, 트램폴린 코드 종료 직전에 API를 호출하게 만든다. 여기서 API 주소 연산 과정은 트램폴린 코드 실행 직전의 API와 관련된 전달인자에 영향을 받는 경우와 받지 않는 경우로 나뉜다.

Themida, VMProtect과 같은 대부분의 패커들은 API의 전달인자에 영향을 받지 않는 트램폴린 코드를 활용한다. 이 방식에서의 주소 계산은 독립적이며, 이는 난독화된 API마다 각각 대응하는 트램폴린를 가지고 있음을 의미한다. 이는 역난독화기가 모든 트램폴린 코드를 실행하여 역난독화된 API 목록을 만들어야 되는 어려움이 있다.

반면에, ASProtect와 같은 일부 패커들은 API의 전달인자에 영향을 받는 트램폴린 코드를 활용한다. 이 때, 난독화된 API의 호출 명령어는 모두 동일한 트램폴린 코드 호출 명령어로 수정된다. 이는 직전에 실행된 전달인자에 따라 주소 계산의 결과가 달라짐을 의미한다. 이 후, 트램폴린 코드는 실행된 호출 명령어를 전달인자에 영향을 받지 않은 트램폴린 코드로 수정한 후 해당 트램폴린 코드를 호출한다. 이런 특성 때문에, 기존의 OEP 이후의 메모리 덤프 파일을 정적 분석하는 방식으로는 전달 인자에 민감한 트램폴린 코드를 사용하는 API 난독화 기법을 역난독화하기 어려운 한계점이 있다[3].

2.3 OEP 난독화 기법

엔트리 포인트는 로더가 프로그램 실행 준비를 마친 후, 실제 프로그램 코드가 처음으로 실행되는 지

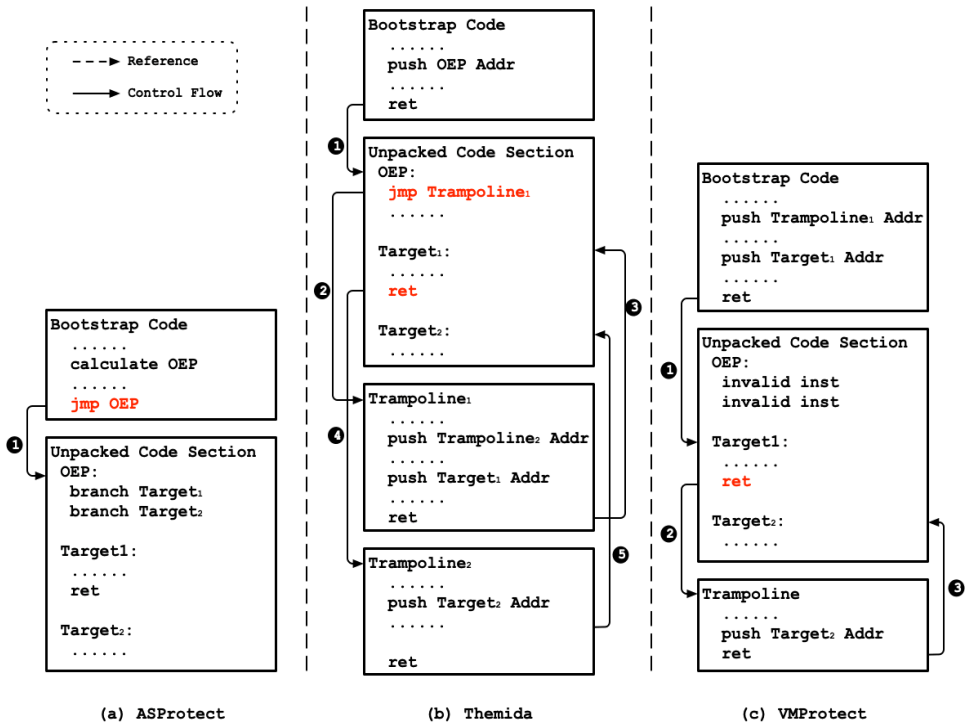

Fig. 4. The analysis results of OEP obfuscation techniques.

점이다. 대부분의 패커는 부트스트랩 코드 섹션을 추가하고, 엔트리 포인트를 부트스트랩 코드 섹션으로 조작한다. 결과적으로, 패킹된 프로그램은 부트스트랩 코드를 먼저 실행하며, 패킹된 원본 프로그램은 런타임에서만 메모리에 임시로 압축 해제된다. 이후, 원본 프로그램이 실행 가능한 상태가 되면, 그제서야 부트스트랩 코드에서 OEP로 분기한다.

패킹된 프로그램을 성공적으로 언패킹하기 위해, 메모리 덤프된 파일을 바이너리 생성에 활용할 뿐만 아니라, 임포트 테이블을 재구성하고, 프로그램 헤더의 엔트리 포인트를 OEP로 복원해야 한다. 관련 연구들은 'Written-then-Execute' 라고 부르는 방식으로 메모리에 복원되고 실행된 명령어의 패턴을 분석하여 OEP를 탐지하여 프로그램을 언패킹했다 [7][8][13][19].

기존의 언패킹 방식을 무력화하기 위해 많은 패커들은 OEP 난독화 기법을 제공하고 있다. OEP 난독화 기법은 트램폴린 코드를 활용하는 방법과 그렇지 않는 방법으로 구분할 수 있다. 먼저, 트램폴린 코드를 활용하는 OEP 난독화 기법은 OEP 주변의 분기 명령어를 삭제한다. 삭제된 분기 명령어는 런타임에 부트스트랩 코드 혹은 트램폴린 코드에 의해 대신 실행된다. Fig. 4.는 다양한 패커에서 사용하는 OEP 난독화 기법의 예시를 보여준다. Themida와 같이 OEP에서 실행되는 트램폴린 코드는 삭제된 첫 번째 분기 *Target1*와 두 번째 분기 *Target2*를 실행하기 위한 트램폴린 코드를 차례대로 실행하도록 만든다. 이 경우는 OEP에 진입한 후 다시 부트스트랩 코드가 속한 섹션으로 분기함으로써 'Jump Outer Section'과 같은 일반적인 OEP 탐지 알고리즘을 효과적으로 방해하여, 가능한 OEP 후보의 수를 증가시킨다. 또한, OEP 난독화 기법을 해결하지 않고 언패킹한 프로그램은 트램폴린 코드 분기 명령어가 OEP에 존재하기 때문에, 프로그램이 실행되자마자 유효하지 않은 메모리 영역으로 분기, 종료된다.

또 다른 예로, VMProtect은 부트스트랩 코드가 *Target1*으로 직접 분기하는 방식을 사용한다. 이 경우는 압축 해제된 코드 섹션에서 호출되는 트램폴린 코드의 수가 비교적 적다. 또한, 부트스트랩 코드가 OEP 명령어를 의도적으로 복원하지 않기 때문에 정확한 위치를 탐지할 수 없다는 어려움이 있다. 이

로 인해, 기존 OEP 탐지 알고리즘은 OEP가 아닌 다른 명령어를 OEP로 탐지할 가능성이 있다. 또한, 트램폴린 코드 호출 명령어가 임시 복원된 코드 섹션에 존재하지 않기 때문에 분석의 난이도를 높일 수 있다.

마지막으로, ASProtect과 같이 트램폴린 코드를 활용하지 않는 OEP 난독화 기법은 부트스트랩 코드에서 OEP 주소 연산을 더욱 복잡하게 만들거나 OEP에 도달하기까지의 분기 명령어를 추가한다. 그러나 한번 OEP에 도달하면, 원본 프로그램과 동일한 실행 결과를 보여준다. 이러한 점으로 인해, 기존의 OEP 탐지 알고리즘만으로도 충분히 OEP를 탐지할 수 있다.

III. 제안 시스템

본 논문의 목표는 자동화된 역난독화 기법을 설계하고 구현하는 것으로, 트램폴린 코드를 탐지하고, OEP 및 API 난독화에 사용된 트램폴린 코드를 분석하고, 역난독화하기 위한 시스템을 제안한다.

3.1 해결 방안

3.1.1 OEP 난독화 기법

다수의 패커들은 부트스트랩 코드 내에서 OEP로의 분기 과정을 복잡하게 설계했다. 관련 연구들은 패커의 종류와 상관없이, 런타임에 원본 코드와 데이터를 복원하는 특성을 활용하여 보편적인 프로그램 언패킹 방법을 연구했다. 그 예로, 'Written-then-Execute' 패턴을 분석하여 OEP 후보를 찾고, 언패킹된 프로그램을 재구성했다. 혹은, 휴리스틱한 방법을 사용하여 OEP를 탐지하고 언패킹을 위해 노력했다. 이 과정에서 몇몇 악성코드에서는 OEP 난독화 기법이 적용되어 언패킹을 실패한 사례가 발견되었다[3]. OEP 난독화 기법을 해결하기 위해, 먼저 상용 패커들의 기법을 분석했다. OEP 난독화 기법은 원본 코드의 OEP 부분을 변형하며 실행 결과를 복잡하게 만들어, 언패킹 과정에서 원본 프로그램의 복원을 어렵게 만든다. 그 중 Themida와 같은 패커들은 프로그램의 실행 흐름을 복잡하게 만들며 OEP 탐지 난이도를 높이기 위해 트램폴린 코드를 활용하고 있다. 트램폴린 코드가 OEP에 삽입된 상태로 언패킹된 프로그램은 실행 즉시 유효하지 않은 메모리 영역에 접근하게 되어 에러가 발생시키고 종료된다. 이러한 문제를 해결하기 위해, 기존의 OEP 탐지 알고리즘을 활용하여, 많은 OEP 후보의 위치를 확인하고, 트램폴린 코드 호출 여부를 분석했다. 이후 트램폴린 코드에서 OEP 다음의 원본 코드로 이어지는 실행 흐름을 모니터링 했다. 트램폴린 코드 호출 여부를 통해 난독화 기법의 적용 여부와 삭제된 OEP 명령어를 파악할 수 있다.

3.1.2 API 난독화 기법

다수의 패커들은 원본 프로그램의 임포트 테이블을 삭제하여 패킹된 프로그램을 생성한다. 이런 접근 방식으로 인해, 패킹된 프로그램이 사용하는 API 식별이 어려워진다. 관련 연구들은 패킹된 프로그램이 실행하는 동안 원본 코드와 데이터를 메모리에 임시 복원하는 과정을 관찰하고, IAT의 위치를 탐지했다. 이때 IAT에 기록된 정보를 활용하여 원본 프로그램의 임포트 테이블에 등록된 API 목록을 파악하고, 임포트 테이블을 재구성했다. 그러나 최신의 패커들은 이러한 방식을 무력화하기 위해 API 난독화 기법을 제공하고 있다. API 난독화 기법은 API의 전달인자에 민감한 트램폴린 코드를 사용하는 경우와 그렇지 않은 경우로 분류할 수 있다. 기존 역난독화 연구는 'API마다 호출되는 트램폴린 코드가 다르다'는 전제 하에, 전달인자에 민감하지 않은 API 난독화 기법만을 해결하려 했다. 그러나 ASProtect와 같은 일부 패커들은 난독화된 API들이 모두 동일한 트램폴린 코드를 호출하는 방식으로, 전달인자에 민감한 API 난독화 기법을 사용한다. 기존에 트램폴린 코드만을 실행하고 분석하는 접근 방식으로는 난독화된 API를 식별하는 것이 불가능했다. 이러한 문제를 해결하기 위해, 트램폴린 코드 호출 명령어가 포함된 베이직 블록을 탐색한다. 이후, 트램폴린 코드를 호출하는 베이직 블록부터 실행 과정을 분석함으로써, 난독화된 API를 모두 식별할 수 있다.

3.2 제안 시스템 상세 설명

본 논문은 동적 바이너리 계측 도구인 Intel Pin[14]과 CPU 에뮬레이터인 Unicorn[15]을 활용하여 OEP 및 API 난독화 기법을 역난독화하는 방안을 제시한다. Fig. 5.는 본 논문에서 제안하는 역난독화 방법의 동작 흐름도를 나타낸다.

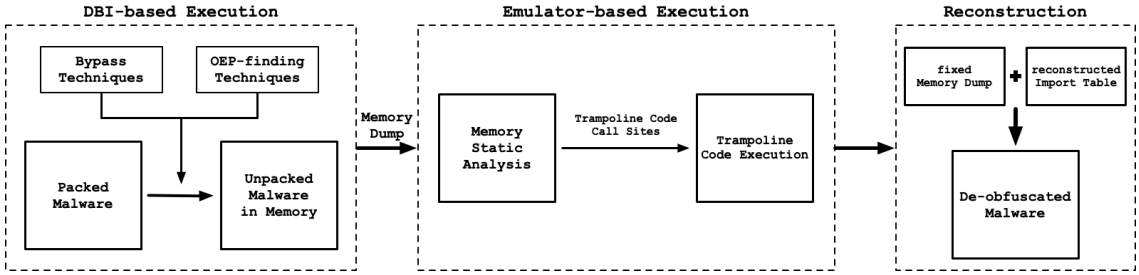

Fig. 5. The overview of Proposed Scheme.

제안 방법은 난독화된 악성코드를 계측하여 가상 메모리에 일시적으로 복원된 프로그램을 덤프하며, 덤프된 파일 내의 모든 트램폴린 코드를 탐색한다. 이후, 어떤 데이터가 트램폴린 코드에 의해 난독화되었는지를 분석하기 위해, 트램폴린 코드를 독립적으로 실행한다. 최종적으로, 트램폴린 코드 분석을 통해 얻은 정보를 활용하여 프로그램 헤더의 OEP와 임포트 테이블을 재구성한다.

3.2.1 프로세스 메모리 덤프

트램폴린 코드, 즉 OEP 및 API 난독화 기법은 난독화된 프로그램이 메모리에 임시로 복원되는 런타임 상태에서 식별이 가능하다. DBI(Dynamic Binary Instrumentation)를 활용하여 난독화된 프로그램을 실행하면, 원본 프로그램이 메모리에 복원되는 시점을 탐지할 수 있다. 이 복원 시점은 메모리 쓰기가 이루어진 영역의 명령어가 부트스트랩 코드에 의해 실행되는 시점, 즉 'Written-then-Execute'로, 기존 언패커에서는 이를 OEP 후보로 식별했다. 또한, 관련 연구들은 OEP 후보는 줄이고 정확도를 높이기 위한 다양한 방안을 제시했다[7][8][13]. 프로세스 메모리 덤프 단계에서는 Intel Pin 기반의 언패커인 PinDemonium[17]의 OEP 탐지 알고리즘을 사용하여 OEP 후보를 식별하고, 해당 시점의 메모리를 덤프한다.

3.2.2 트램폴린 코드 실행 분석

메모리 덤프 파일은 원본 프로그램이 복원된 상태를 포함하며, OEP와 API 난독화 기법이 적용된 프로그램의 경우, 트램폴린 코드 호출 명령어가 덤프 파일 내에 존재한다. OEP 난독화 기법은 OEP의 위치에 트램폴린 코드 호출 명령어를 배치하거나 런타임에 트램폴린 코드의 주소가 스택에 저장하여 실행할 수 있다. API 난독화 기법은 IAT에 트램폴린 코드의 주소가 기록하거나 API 호출 명령어를 트램폴린 코드 호출 명령어로 수정하여 실행할 수 있다.

트램폴린 코드들은 난독화된 주소를 연산하고, 복호화된 주소를 스택에 저장한다. 이들 코드들은 복호화된 주소가 스택의 최상단에 위치할 때, 리턴 명령어를 실행하여 복호화된 주소로 분기하는 패턴을 보인다.

트램폴린 코드 실행 분석 단계에서는 덤프 파일 내에 모든 트램폴린 코드를 식별한다. 앞서 언급한 바와 같이, 이 코드들은 스택에 복호화된 주소를 저장하고 리턴 명령어를 실행하는 패턴을 가지고 있다. 에뮬레이터는 식별된 트램폴린 코드를 순차적으로 실행하며, 이 패턴을 탐지하는 역할을 한다. 이 과정에서 탐지된 주소를 통해 OEP 또는 API 난독화 기법에 사용된 트램폴린 코드인지 구분할 수 있다.

3.2.3 원본 프로그램 재구성

원본 프로그램 재구성 단계는 패킹과 난독화 기법에 의해 변형된 프로그램의 정보를 복원하는 과정을 포함한다. 기존의 언패커와 유사하게, 덤프 파일의 프로그램 헤더에서 엔트리 포인트 위치에 OEP 주소를 기록하고, 임포트 테이블을 재구성함으로써, 언패킹 및 역난독화된 프로그램을 생성한다.

IV. 역난독화 성능 평가

이 섹션에서는 악성코드에서 사용되는 다양한 패커의 OEP 및 API 난독화 기법을 역난독화하는 제안 기법의 성능을 검증하기 위한 실험을 진행한다.

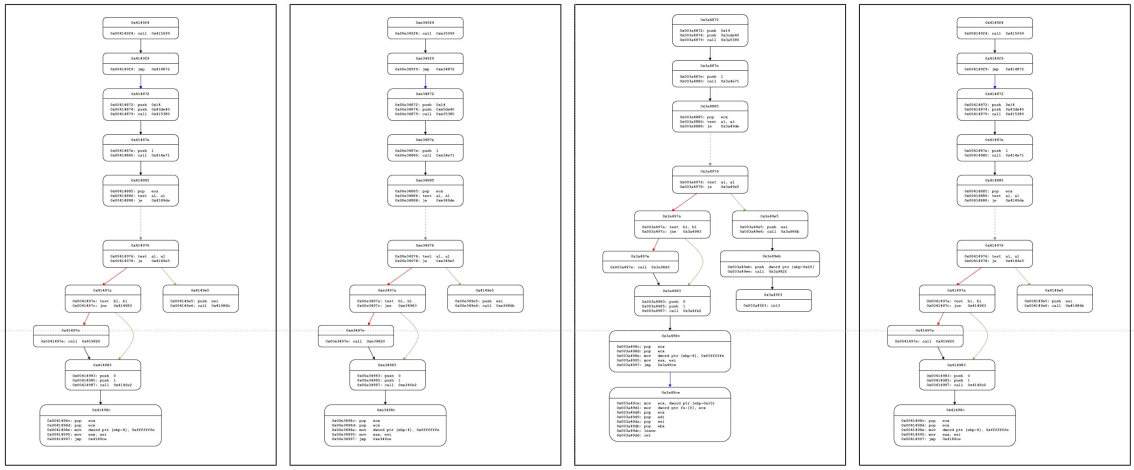

Fig. 6. CFG Extraction Comparison: Original vs. Themida vs. VMProtect vs. ASProtect.

4.1 실험 데이터셋

실험에 사용한 패커는 Themida v3.0.7, VMProtect v3.0.9, ASProtect v2.78이다. 패커를 적용할 샘플 프로그램은 NIST에서 제공하는 Juliet Test Suite C/C++ 데이터셋[16]을 컴파일하여 생성된 프로그램으로 임의로 선정한 테스트 프로그램에 OEP 및 API 난독화 기법을 적용했다.

4.2 실험 환경

본 논문에서 제안하는 시스템은 총 15개의 Windows 10 가상머신에서 실행된다. 각 가상머신은 4개의 CPU 코어와 8GB의 메모리를 갖추고 있다. DBI 기반 실행 중, 각 샘플 프로그램에 대해 30분의 실행 시간 제한을 설정했다.

4.3 실험 결과

Table. 1은 실험에 사용한 프로그램에 대해 각 패커가 제공하는 API 난독화 기법을 적용한 후, 제안 시스템으로 역난독화한 결과를 보여준다. Themida, VMProtect, ASProtect는 각각 고유한 API 난독화 기법을 사용한다. 실험에서, 테스트 프로그램에 난독화를 적용한 결과, 83개의 API 중 Themida는 69개, VMProtect는 83개, ASProtect는 26개가 난독화했다. 이 차이는 각 패커가 난독화할 수 있는 DLL (Dynamic Loading Library), 부트스트랩 코드 내에서 사용하는 API, 트램폴린 코드의 방식에 따라 난독화가 가능한 범위가 다르기 때문이다. 제안 시스템은 메모리 덤프 파일 내에 있는 트램폴린 코드 호출 명령어가 포함된 베이직 블록을 분석의 기준으로 삼아 역난독화를 수행한다. 결과적으로, Themida와 VMProtect처럼 API의 전달인자에 민감하지 않은 경우 뿐만 아니라, ASProtect와 같이 API의 전달인자에 민감한 경우에도 역난독화가 성공적으로 이루어짐을 확인했다.

Fig. 6은 실험에 사용된 프로그램에 각 패커가 제공하는 OEP 난독화 기법을 적용한 후, 제안 시스템으로 역난독화한 결과를 CFG (Control Flow Graph)로 비교한 것이다. CFG는 원본, Themida, VMProtect, ASProtect 순서로 추출되었는데, VMProtect의 CFG에서는 약간의 차이점이 관찰되었다. 그러나 이 차이는 OEP 명령어 주변의 일부에 국한되며, 프로그램의 전체 실행 결과는 모두 동일함을 확인했다.

Table 1. The de-obfuscation result of test program's API obfuscation techniques

Packer	Number of Obfuscated API	Number of De-obfuscated API
Themida	69/83	69/69
VMProtect	83/83	83/83
ASProtect	26/83	26/26

V. 결 론

최신의 역난독화 연구에서는 임포트 테이블의 재구성을 방해하는 API 난독화 기법에 대한 연구가 진행되었다. 그러나 API의 전달인자에 민감한 트램폴린 코드를 사용하는 API 난독화 기법에 의해 숨겨진 임포트 테이블을 재구성하는데 실패했다. 또한, OEP 난독화 기법이 적용된 악성코드를 언패킹하는데 실패했다. 이에 본 논문에서는 다양한 패커에서 사용되는 OEP 난독화 기법과 API 난독화 기법에 대해 분석하고, 효과적으로 역난독화하는 방법을 제안했다.

본 논문의 제안 방법은 OEP 및 API 난독화 기법이 적용된 악성코드를 동적 바이너리 계측 도구와 에뮬레이터를 사용하여 트램폴린 코드가 난독화한 정보를 분석하여 API의 전달인자에 민감한 트램폴린 코드를 사용하는 기법이 난독화한 정보를 식별할 수 있다. 또한, OEP 후보에서 트램폴린 코드 호출 유무를 탐지하여, 삭제된 OEP 명령어를 식별할 수 있다. 이는 트램폴린 코드를 통해 난독화된 악성코드를 역난독화 및 언패킹이 가능하다는 것을 의미한다. 그러나 알려지지 않은 패커들은 OEP와 API를 난독화하기 위한 또 다른 기법을 사용할 가능성이 있으며, OEP와 API 외에 다양한 정보를 난독화하는 것이 가능하기 때문에 여전히 악성코드 분석에 어려움을 겪을 수 있다.

추후 연구는 다양한 악성코드를 분석하여 알려지지 않은 패커의 난독화 기법을 분석하고, 이를 역난독화하는 방향으로 진행할 예정이다.

References

[1] AV-TEST "Malware Statistics", https://www.av-test.org/en/statistics/malware/", accessed Apr. 2023.

[2] FireEye, "M-Trends 2020", https://content.fireeye.com/m-trends/rpt-m-trends-2020, accessed Apr. 2022.

[3] Binlin Cheng, Jiang Ming, Erika A Leal, Haotian Zhang, Jianming Fu, Guojun Peng and Jean-Yves Marion, "Obfuscation-resilient executable payload extraction from packed malware," Proceeding of the 30th USENIX Security Symposium, pp. 3451-3468, Aug. 2021.

[4] Bang Cheol-ho, Jae Hyuk Suk, and Sang-jin Lee, "Vmprotect operation principle analysis and automatic deobfuscation implementation," Journal of the Korea Institute of Information Security & Cryptology, 30(4), pp.605-616, Aug. 2020

[5] Kevin A. Roundy and Barton P. Miller, "Binary-code obfuscations in prevalent packer tools," ACM Computing Surveys, vol. 46, no. 1, pp. 1-32, Oct. 2013.

[6] Jae Hyuk Suk, Jae-Yung Lee, Hongjoo Jin, In Seok Kim and Dong Hoon Lee, "Unthemida: commercial obfuscation technique analysis with a fully obfuscated program," Software: Practice and Experience, vol. 48, no. 12, pp. 2331-2349, Jul, 2018.

[7] Ryoichi Isawa, 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, vol. 98, no. 4, pp. 883-893, Apr. 2015.

[8] Min Gyung Kang, Pongsin Poosankam and Heng Yin, "Renovo: a hidden code extractor for packed executables," Proceedings of the 2007 ACM workshop on Recurring malcode, pp. 46-53, Nov. 2007.

[9] Seokwoo Choi, Taejoo Chang, Changhyun Kim and Yongsu Park, "X64unpack: hybrid emulation unpacker for 64-bit windows environments and detailed analysis results on vmprotect 3.4," IEEE Access, vol. 8, pp. 127939-127953, Jul. 2020.

[10] Oreans Technologies, "Themida:

Advanced Windows Software Protection System," https://www.oreans.com/Themida.php, accessed Apr. 2022.
[11] VMProtect, "VMProtect Software," https://vmpsoft.com/, accessed Apr. 2022.
[12] ASProtect, "ASProtect 32 - packer for 32bit software with protection functions," http://www.aspack.com/asprotect32.html, accessed Apr. 2022.
[13] Gyeong Min Kim and Yong Su Park, "Improved original entry point detection method based on pindemonium," KIPS Transactions on Computer and Communication Systems, 7(6), pp. 155-164, May. 2018
[14] Pin, "Pin - A Dynamic Binary Instrumentation Tool, " https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html, accessed Apr. 2022.
[15] Unicorn, "Unicorn - The Ultimate CPU emulator," https://www.unicorn-engine.org/, accessed Apr. 2022.
[16] Juliet Test Suite, "Juliet C/C++ 1.3 - NIST Software Assurance Reference Dataset", https://samate.nist.gov/SARD/test-suites/112, accessed Apr. 2022.
[17] Sebastiano Mariani, Lorenzo Fontana, Fabio Gritti and Stefano D'Alessio, "Pindemonium: a dbi-based generic unpacker for windows executables," 2016.
[18] Binlin Cheng, Jiang Ming, Jianming Fu, Guojun Peng, Ting Chen, Xiaosong Zhang and Jean-Yves Marion, "Towards paving the way for large-scale windows malware analysis: generic binary unpacking with orders-of-magnitude performance boost," Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 395-411, Oct. 2018.
[19] Binlin Cheng, Erika A Leal, Haotian Zhang and Jiang Ming, "On the feasibility of malware unpacking via hardware-assisted loop profiling," Proceeding of the 32nd USENIX Security Symposium. pp. 7481-7498, Aug. 2023.

〈저자소개〉

김 민 호 (Minho Kim) 학생회원
2020년 2월: 숭실대학교 전자정보공학부 학사
2022년 2월: 숭실대학교 소프트웨어학과 석사
2022년 3월~현재: 숭실대학교 소프트웨어학과 박사과정
〈관심분야〉 시스템 보안, 프로그램 분석, 모바일 보안

이 정 현 (Jeong Hyun Yi) 종신회원
1993년 2월: 숭실대학교 전자계산학과 학사
1995년 2월: 숭실대학교 컴퓨터학과 석사
2005년 8월: University of California at Irvine, Computer Science 박사
1995년 2월~2001년 7월: 한국전자통신연구원(ETRI) 연구원
2000년 4월~2001년 3월: 미국표준기술연구소(NIST) 객원연구원
2005년 10월~2008년 8월: 삼성종합기술원 수석연구원
2008년 9월~현재: 숭실대학교 IT대학 소프트웨어학부 교수
〈관심분야〉 모바일 보안, 컴퓨터 보안, 네트워크 보안

조 해 현 (Haehyun Cho) 종신회원
2013년 2월: 숭실대학교 컴퓨터학부 학사
2015년 2월: 숭실대학교 컴퓨터학과 석사
2021년 2월: Arizona State University, Computer Science 박사
2021년 3월~현재: 숭실대학교 소프트웨어학부 조교수
〈관심분야〉 시스템 보안, 취약점 탐지, 프로그램 분석, AI 보안