Unicorn 엔진 기반 자동 역난독화 시스템

이광열(학부생),* 김민호(대학원생),* 조해현,* 이정현*

*숭실대학교

Automated Deobfuscation System based on Unicorn Engine

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

*Soongsil University

요 약

악성코드는 수많은 기업과 사용자에게 개인정보 유출과 금전적 피해를 입히고 있다. 다양한 경로로 배포되는 악성코드를 대처하기 위해 악성코드 분석가는 엄청난 노력을 기울이고 있다. 마찬가지로, 악성코드 제작자는 다양한 분석 환경에서 탐지를 회피하기 위해 많은 기법을 사용한다. 대표적인 기법으로는 난독화 기법이 있다. 난독화 기술이 적용된 악성코드를 분석하는 것은 어렵다. 안타깝게도, 기존의 역난독화 방법들은 Original Entry Point (OEP) 난독화, API 난독화 등의 기술을 제대로 대응하지 못하고 자동화된 역난독화에 어려움을 겪고 있다.

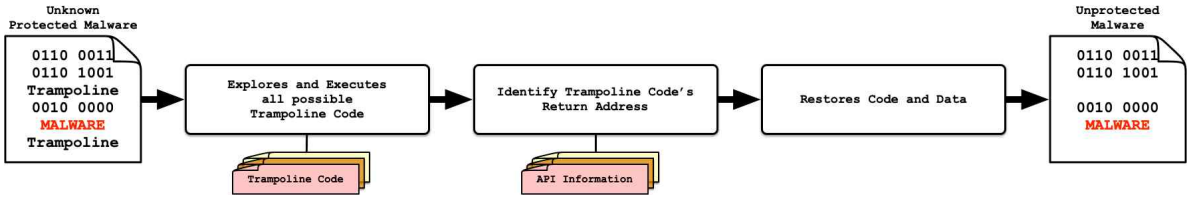
본 논문에서는 먼저 OEP 및 API 난독화 기술에 대해 분석했다. 그런 다음 OEP 및 API 난독화 기술을 역난독화하는 시스템을 설계 및 구현한다. 특히, 제안 시스템은 OEP 및 API 난독화에 사용되는 트램폴린 코드를 분석하고 처리한다. 실험을 통해 제안 시스템을 사용하여 다양한 프로텍터의 OEP 및 API 난독화 기술을 분석하고 역난독화할 수 있음을 입증했다.

I. 서론

최신의 악성코드는 기존의 악성코드 분석 방식을 회피하기 위해 다양한 분석 방지 및 난독화 기법을 사용하고 있다[1]. 이러한 기법을 사용하는 악성코드는 악성코드 분석가가 리버스 엔지니어링을 통해 분석하는 과정에서 분석을 회피하거나 어렵게 만들고 있다. 프로텍터는 대상 프로그램을 보호하기 위해 패킹, 분석 회피 기법, 그리고 다양한 난독화 기법을 제공해주는 프로그램으로, 악성코드에 널리 사용되고 있다. 대표적으로 Themida[2], VMProtect[3], ASProtect[4] 등이 있다. 각 프로텍터는 동일한 기법을 프로텍터 고유의 방식으로 악성코드에 적용하고 있다. 프로텍터에 의해 보호되는 악성코드를 분석하기 위해 악성코드를 언패킹 및 역난독화 연구가 많이 수행되고 있다[5,6,7]. 하지만 기존 연구는 프로세스 메모리 덤프를 통한 프로그램 복원과 API 난독화 기법 중 일부를 해결하는 방식에 치우쳐 있다[7,8]. 특히, 최신의 연구는 OEP 탐지를 전제로 언패킹 및 API 난독화 기법을 해결하려고 시도했으나 OEP 난독화 기법이 적용된 데이터셋으로 인해 실행이 불가능한 언팩된 프로그램을 생성하는 한계점을 보여주었다. 또한, API 난독화 기법 중 일부만을 대상으로 역난독화하기 때문에 임포트 테이블을 올바르게 복구하지 못하는 경우가 발생할 수 있다[7].

따라서, 본 논문에서는 악성코드 분석을 위해 다양한 프로텍터에서 제공하는 OEP 및 API 난독화 기법에 대해 분석하고 기존의 한계점을 개선하여 효과적으로 역난독화하는 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] 제안 시스템 개요

II. 배경 지식 및 관련 연구

2.1 OEP 난독화 기법 분석

패킹된 프로그램의 OEP는 원본 프로그램의 Entry Point (EP)로, 원본 프로그램의 처음 실행되는 명령어를 의미한다. OEP 난독화 기법은 OEP의 명령어를 프로텍터가 추가한 코드인 트램폴린 코드로 분기하도록 패치한다. 트램폴린 코드는 패킹 과정에서 추가된 프로텍터의 코드 섹션 혹은 동적으로 할당된 메모리에 위치한다. 기존의 언패킹 과정에서는 프로텍터의 코드 섹션을 제거하기 때문에 트램폴린 코드가 메모리에 존재하지 않는다. 이로 인해, OEP 난독화된 악성코드를 언패킹하면 [그림 2]와 같이 OEP에는 잘못된 메모리 접근 에러를 발생하는 명령어가 존재하고 정상적으로 실행되지 않는 문제가 발생한다.

2.2 API 난독화 기법 분석

프로그램의 임포트 테이블은 프로그램의 실행 과정에서 호출되는 API의 정보가 기록되어 있다. 기존 연구에서는 다양한 접근 방식을 통해 임포트 테이블을 복구하고 있다.

API 난독화 기법은 프로텍터가 임포트 테이블에 각 API의 호출을 난독화한 트램폴린 코드의 주소를 기록하는 방식을 사용한다. 트램폴린 코드는 OEP 난독화 기법과 동일한 형태로 프로텍터의 코드 섹션 혹은 동적 할당된 메모리 영역에 위치한다. 기존 언패킹 과정은 임포트 테이블에 기록된 API 주소들을 식별하고 패킹 과정에서 삭제된 정보를 복원하는 방식을 사용한다. 그러나 임포트 테이블에 기록된 트램폴린 코드로 인해 언패커는 잘못된 주소를 식별하고 올바른 임포트 테이블을 생성하지 못하는 문제가 발생한다. 또한, 임포트 테이블에 기록되는 트램폴린 코드는 크게 두 가지 형태로 구분할 수 있다. 난독화된 API마다 모두 다른 트램폴린 코드를 사용하는 전달 인자에 민감하지 않은 경우와 난독화된 API가 모두 같은 트램폴린 코드를 사용하는 전달 인자에 민감한 경우로 분류할 수 있다. 전달 인자에 민감하지 않은 트램폴린 코드는 각각 제어 흐름을 추적하면 난독화된 API를 식별할 수 있다. 그러나 전달 인자에 민감한 트램폴린 코드는 코드 자체의 제어 흐름 분석만으로는 난독화된 API 식별이 불가능하다.

```
 >>Exploring -- 0x00ec1023 jmp 0x20ad
 >>Exploring -- 0x00ec20d0 jmp 0x39e6e4
[Jmp to Other Section]  0x00ec20d0 -> 0x0125f7b4 push ebp
[Jmp to Origin Section] 0x00f13902 -> 0x00ec1d30 push ebp
----------------- [not Patched Instruction] -----------------
 >>Exploring -- 0x00ec1023 jmp 0x20ad
 >>Exploring -- 0x00ec20d0 jmp 0x39e6e4
ERROR: Invalid memory mapping (UC_ERR_MAP)
```

[그림 2] OEP 난독화 코드 패치 전 실행 결과

2.3 기존 역난독화 접근 방식

기존의 역난독화 연구는 API 난독화 기법에 대해 집중적으로 이루어졌다. 특히, 최신의 연구인 API-Xray[7]는 12개의 프로텍터에서 제공하는 API 난독화 기법을 5가지 유형으로 분류하고, Intel BTS를 기반으로 각각의 트램폴린 코드를 모두 실행하여 난독화된 API를 식별하고 올바른 임포트 테이블을 복원한다. API-Xray는 API 난독화 기법에 대한 가장 큰 규모의 역난독화 연구 중 하나로, 프로텍터 별로 제공하는 API 난독화 기법의 유형을 분류한 점에 의의가 있다. 하지만 트램폴린 코드의 시작 지점만을 탐색하고 실행하는 방식으로 인해, 전달인자에 민감한 트램폴린 코드가 난독화하는 API를 식별할 수 없는 한계점이 있다.

III. 제안 기법

본 논문에서는 CPU 에뮬레이터 중 하나인

Unicorn 엔진[8]을 기반으로 OEP 및 API 난독화 기법을 역난독화하는 시스템을 제안한다. 본 논문이 제안하는 역난독화된 프로그램을 생성하기 위한 시스템은 [그림 1]과 같다. 프로텍터가 제공하는 OEP 및 API 난독화 기법을 역난독화하기 위한 방법은 다음과 같다.

3.1 트램폴린 코드 탐색 및 실행

본 논문에서는 난독화된 프로그램을 동적으로 OEP에 도달한 시점의 프로세스 메모리를 덤프한 상황을 기반으로 분석한다. 먼저, 코드 섹션의 영역에 트램폴린 코드를 호출하는 명령어가 존재하는지 탐색한다. 트램폴린 코드는 프로텍터가 추가한 섹션 혹은 동적으로 할당된 섹션에 존재할 수 있다. 트램폴린 코드는 호출되는 시점에 따라 OEP 난독화 코드, API 난독화 코드로 분류할 수 있다. 특히, API 난독화 코드는 전달인자에 민감한 트램폴린 코드와 민감하지 않은 트램폴린 코드로 분류할 수 있다. 전달인자에 민감한 트램폴린 코드는 스택에 영향을 주는 명령어에 따라 호출되는 API가 다른 특징을 가지고 있다. 따라서 전달인자에 민감한 트램폴린 코드는 트램폴린 코드가 속한 베이직 블록 내에 스택에 영향을 주는 명령어까지 모두 탐색해야 한다. 트램폴린 코드를 모두 분류하고 유니콘 에뮬레이터를 사용하여 트램폴린 코드를 각각 실행한다. 트램폴린 코드를 에뮬레이트함으로써 트램폴린 코드의 실행 결과만을 분석하여 난독화된 정보만을 식별할 수 있다.

3.2 트램폴린 코드의 복귀 주소 식별

OEP 난독화 코드는 OEP에서 트램폴린 코드로 분기하고 많은 명령어를 실행한 후 원본 프로그램을 기준으로 OEP 다음 코드를 실행하는 주소로 복귀하는 특징을 가지고 있다.

API 난독화 코드는 API를 호출하는 과정에서 트램폴린 코드로 직접 분기하거나 임포트 테이블에 기록된 트램폴린 코드의 주소로 간접 분기한다. 분기한 트램폴린 코드는 최종적으로 난독화된 API의 주소를 계산하고 스택 영역에 API 주소를 기록하고 리턴함으로써 난독화된 API를 직접 호출하는 특징을 가지고 있다. 이처럼 OEP 및 API 난독화에 사용되는 트램폴린 코드의 복귀 주소를 식별함으로써 트램폴린 코드가 난독화한 정보를 식별할 수 있다.

3.3 원본 코드 및 데이터 복원

OEP 트램폴린 코드에서 식별한 복귀 주소는 OEP 이후에 실행되는 정상적인 실행 흐름을 의미한다. OEP의 명령어는 트램폴린 코드로 분기하기 때문에 다시 정상적인 실행 흐름으로 분기할 수 있도록 패치해야 한다. 이 과정에서 OEP 트램폴린 코드를 실행하여 식별한 복귀 주소로 분기하도록 패치한다.

API 트램폴린 코드에서 식별한 복귀 주소는 프로그램의 임포트 테이블에 기록되는 API의 주소를 의미한다. 패킹 및 API 난독화 과정에서 일부 호출 명령어는 임포트 테이블을 거치지 않고 직접 호출하는 명령어로 조작되거나 임포트 테이블에 트램폴린 코드가 기록되어 있기 때문에 이를 올바르게 식별하고 패치해야 한다. 이 과정에서 새로운 임포트 테이블을 생성하고 API 호출 명령어들이 임포트 테이블을 참조하도록 패치한다.

IV. 실험

본 논문에서는 다양한 악성코드에서 사용하는 프로텍터의 OEP 난독화 기법과 API 난독화 기법에 대한 역난독화 시스템의 성능을 검증하기 위한 실험을 진행한다. 실험에 사용한 프로텍터는 Themida (v2.4.5, v3.0.7), VMProtect (v3.0.9, v3.4.0), ASProtect (v2.78)로 실험에 사용한 프로그램에 OEP 난독화 기법과 API 난독화 기법을 모두 적용했다.

기존의 언패커는 프로텍터가 추가한 코드 섹션을 제거하기 때문에 트램폴린 코드가 코드 섹션 혹은 동적 할당된 메모리에 존재하지 않는다. 이로 인해, OEP 난독화가 적용된 프로그램을 언패킹하면 [그림 2]와 같이 매핑되지 않은 영역의 명령어를 실행을 시도하여 에러가 발생한다. [그림 3]의 실행 로그는 제안 시스템에서 프로텍터의 코드 섹션과 동적 할당된 섹션을 모두 매핑 후 실행한 결과와 코드 패치

```
 >>Exploring -- 0x00ec1023 jmp 0x20ad
 >>Exploring -- 0x00ec20d0 jmp 0x39e6e4
[Jmp to Other Section]  0x00ec20d0 -> 0x0125f7b4 push ebp
[Jmp to Origin Section] 0x00f13902 -> 0x00ec1d30 push ebp
 >>Exploring -- 0x00ec1d30 push ebp
 >>Exploring -- 0x00ec139d jmp 0x2eb3
 >>Exploring -- 0x00ec3250 push ebp
 >>Exploring -- 0x00ec327c call 0xf44
 >>Exploring -- 0x00ec31c0 push ebp
 >>Exploring -- 0x7607f390 jmp dword ptr [0x760e1878]
[Call API] 0x7607f390 GetSystemTimeAsFileTime
------------------ [Patch Instruction] ----------------------
 >>Exploring -- 0x00ec1023 jmp 0x20ad
 >>Exploring -- 0x00ec20d0 jmp 0xc60
 >>Exploring -- 0x00ec1d30 push ebp
 >>Exploring -- 0x00ec139d jmp 0x2eb3
 >>Exploring -- 0x00ec3250 push ebp
 >>Exploring -- 0x00ec327c call 0xf44
 >>Exploring -- 0x00ec31c0 push ebp
 >>Exploring -- 0x7607f390 jmp dword ptr [0x760e1878]
[Call API] 0x7607f390 GetSystemTimeAsFileTime
```

[그림 3] OEP 난독화 코드 패치 후 실행 결과

Address	Hex															
0041B000	70	5E	40	77	6C	C2	B9	89	B0	05	6C	76	80	DF	6B	76
0041B010	F0	DF	6B	76	02	6D	F4	7C	73	8D	08	BC	00	7D	D1	89
0041B020	33	25	6C	84	D2	5A	8A	E0	50	F5	6B	76	1A	AC	55	E8
0041B030	70	F5	6B	76	80	F3	6B	76	60	DF	6B	76	10	DF	6B	76
0041B040	10	E0	6B	76	50	0E	6C	76	BE	DC	D0	86	50	15	6C	76
0041B050	20	17	6C	76	20	4F	6D	76	00	87	42	77	00	00	00	00

[그림 4] ASProtect의 API 난독화가 적용된 프로그램의 임포트 테이블

후 실행한 결과를 비교한 모습이다. 이를 통해, OEP 난독화 코드 패치 후에 정상적으로 실행하는 것을 확인했다.

[표 1]은 각각의 프로텍터의 API 난독화 기법을 프로그램에 적용했을 때, 전체 API 69개 중에 얼마만큼 API 난독화가 적용되는지와 난독화된 API 중 식별한 API의 수를 나타낸다. 그 중에서 Themida와 ASProtect는 일부 API만을 대상으로 난독화가 적용된다. 원본 프로그램과 비교했을 때, [그림 4]과 같이 kernel32.dll의 API 중 일부 API만 난독화가 되는 것으로 분석했다. 제안 시스템을 이용하여 난독화된 API의 트램폴린 코드를 실행함으로써 트램폴린 코드의 복귀 주소를 모두 식별했다.

V. 결론

악성코드를 보호하기 위한 기법 중 난독화 기법이 차지하는 비중은 매우 크다. 최신의 연구에서는 API 난독화 기법을 상세하게 분석하고 해결하고자 했다. 그러나 전달인자에 민감한 트램폴린 코드로부터 난독화된 API 정보를 식별하지 못하는 한계점이 있다. 또한 약 10%의 악성코드에 OEP 난독화 기법이 적용되어 있어 언패킹에 실패하였다. 이에 본 논문에서는 다양한 프로텍터에서 제공하는 OEP 난독화 기법과

[표 1] API 난독화 기법에 대한 역난독화 결과

Protector	난독화된 API 개수	역난독화된 API 개수
Themida v2.4.5	18/69	18/18
Themida v3.0.7	67/69	67/67
VMProtect v3.0.9	69/69	69/69
VMProtect v3.4.0	69/69	69/69
ASProtect v2.78	8/69	8/8

API 난독화 기법에 대해 분석하고 자동으로 역난독화하는 시스템을 제안했다.

본 논문의 제안 시스템을 통해 프로텍터가 제공하는 OEP 및 API 난독화 기법을 효과적으로 무력화하는 것을 확인할 수 있다. 그러나 패킹 및 난독화된 프로그램의 OEP에 도달한 시점에 메모리 덤프한 바이너리를 분석하기 때문에 패킹된 프로그램이 메모리 덤프가 되지 않으면 분석 시스템의 적용이 불가능한 단점을 가지고 있다. 추후 연구를 통해 다양한 프로텍터가 제공하는 다양한 난독화 기법을 분석하고 역난독화하는 연구를 진행할 예정이다.

[참고문헌]

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

[2] Oreans Technology, “Themida”, https://www.oreans.com/Themida.php/.

[3] VMProtect Software, “VMProtect”, https://vmpsoft.com/.

[4] ASPACK software, “ASProtect”, http://www.aspack.com/asprotect32.html.

[5] J. H. Suk, J. Lee, H. Jin, I. S. Kim and D. H. Lee, "UnThemida: Commercial obfuscation technique analysis with a fully obfuscated program," Software: Practice and Experience, 48(12), pp. 2331-2349 2018.

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

[8] Osnat Levi. “Unicorn - the ultimate cpu emulator”, 2015-2022. https://www.unicorn-engine.org/.