Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.6, Dec. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.6.1375

최신 버전의 Themida가 보이는 정규화가 어려운 API 난독화 분석방안 연구

이 재 휘,† 이 병 희, 조 상 현‡
NAVER Corporation

A Study on the Analysis Method to API Wrapping that Difficult to Normalize in the Latest Version of Themida

Jae-hwi Lee,† Byung-hee Lee, Sang-hyun Cho‡
NAVER Corporation

요 약

최근 상용 프로텍터인 Themida의 최신 버전이 업데이트되면서, 추적할 초기 데이터를 제공하는 가상 메모리 할당을 사용하지 않도록 변경해 기존 연구의 정규화된 대응방안을 적용할 수 없게 되었다. 또한, 실행 중에 결정되는 값이 많아 동적으로 추적하기 수월했던 기존의 버전에 비해, 프로텍터를 적용하는 시점에 결정된 고정값이 많아 동적으로 추적하는데 어려움이 생겼다. API 난독화 과정을 정규화하기 어려워지도록 최신 버전의 Themida가 어떤 방식을 채용했는지 알아보고, 이를 해제하는 자동화된 시스템을 추후 개발하기 위해 어떠한 기술을 적용할 수 있을지 가능성을 검토해 본다.

ABSTRACT

The latest version of commercial protector, Themida, has been updated, it is impossible to apply a normalized unpacking mechanism from previous studies by disable the use of a virtual memory allocation that provides initial data to be tracked. In addition, compared to the previous version, which had many values that determined during execution and easy to track dynamically, it is difficult to track dynamically due to values determined at the time of applying the protector. We will look at how the latest version of Themida make it difficult to normalize the API wrapping process by adopted techniques and examine the possibilities of applying the unpacking techniques to further develop an automated unpacking system.

Keywords: Themida, API wrapping, Unpacking

I. 서 론

프로텍터는 실행 파일을 압축하거나 난독화하는 프로그램인 패커의 한 종류로, 실행 파일의 설계를 파악하기 어려워지도록 다양한 분석방해 기술을 적용한다[1]. 그래서 실행 파일의 설계가 노출되지 않기를 원하는 개발자는 프로텍터를 실행 파일에 적용하기도 한다. 하지만 분석을 어렵게 만드는 특성 때문에 자신의 행위를 감추려 하는 악성코드나 게임 핵, 어뷰징 프로그램 역시 프로텍터를 적용하고 있다. 만약 분석해야 하는 프로그램에 프로텍터가 적용되어 있다면 적용된 분석방해 기술을 제거해가며 분석해야 하므로, 분석 시간이 늘어나게 된다. 따라서 분석 시간을 효과적으로 줄이기 위해서는 프로텍터가 사용하는 분석방해 기술의 작동원리를 파악하고, 각 분석방해 기술을 우회하거나 해제하는 방안을 찾아 정규화

Received(10. 02. 2019), Modified(11. 14. 2019), Accepted(11. 18. 2019)
† 주저자, jaehwi.lee@navercorp.com
‡ 교신저자, super.bg@navercorp.com(Corresponding author)

하고, 그에 맞는 자동화된 해제 시스템 개발이 필요하다.

최근 상용 프로텍터인 Themida[2]는 2019년 중순에 새로운 메이저 버전인 3.0을 출시했다. 최신 버전에서는 이전의 2.x 버전에 비해 속도가 느려지고 실행 파일의 크기가 커지는 대신, 더 복잡하고 어려운 분석방해 기술이 적용됐다. 기존에 진행된 연구에서는 Themida가 적용된 실행 파일의 실행과정에서 가상 메모리를 할당해 사용하는 것을 이용해 추적할 초기 데이터를 획득하고, 동적으로 결정되는 다양한 값을 추적하는 방법을 정규화해 API 난독화 해제 시스템을 개발했다. 그러나 최신 버전의 API 난독화는 가상 메모리를 할당하지 않아 기존 연구의 정규화된 분석방안으로는 대응할 수 없다. 또한, 이전 버전에서는 실행 중에 동적으로 결정되던 값이 최신 버전에서는 실행 파일에 프로텍터를 적용하는 시점에 결정되고, 고정되어 있어 동적으로 추적하는 과정을 정규화하기 훨씬 까다로워졌다.

API는 프로그램 분석시 각 함수의 작업 내용을 유추할 수 있는 중요한 요소이므로, 효율적인 분석 진행을 위해서는 반드시 API 정보를 복구해주어야 한다. 따라서 본 논문에서는 최신 버전의 Themida가 API 난독화 과정의 정규화를 어렵게 만들기 위해 어떤 방식을 채용했는지 알아보고, 난독화 과정에서 사용하는 데이터가 어떤 식으로 이동하며 변화하는지 파악한다. 그리고 이를 해제하는 자동화 시스템을 개발하기 위해 어떠한 기술을 적용할 수 있을지 알아본다. 또한, 자동화된 해제 시스템 개발 가능성을 검토해 본다.

논문의 구성은 다음과 같다. 2장에서는 기존 연구 대상이었던 Themida v2.x 버전의 API 난독화 과정을 분석한 결과와 이를 토대로 개발한 자동화된 API 난독화 해제 시스템에 대해 알아본다. 3장에서는 새로운 메이저 버전인 Themida v3.x 버전의 API 난독화 과정에 대해 분석한 결과를 설명하고, 최신 버전의 API 난독화를 어떻게 해제할 수 있는지를 보인다. 또한, 어떤 이유로 인해 정규화가 어렵게 변하였는지 설명하고, 이를 해결할 방안에 대해 알아본다. 그리고 분석한 결과 토대로 API 난독화가 적용된 실행 파일을 수작업으로 복구해보고, 그 결과를 보인다. 마지막 4장에서는 본 연구의 한계점과 향후 자동화된 시스템을 구축하기 위한 연구 방향을 기술한다.

II. 기존 연구

2.1 이전 버전의 Themida API 난독화 과정 연구

Themida의 이전 버전인 v2.x의 API 난독화를 해제하기 위한 연구에서는, 실행과정을 분석하고 관찰해 필요한 정보들을 모아 API 난독화를 해제하였다[3]. 이전 버전의 API 난독화는 실행 중에 결정되는 요소가 많으며, 변화를 관찰할 대상의 위치가 고정되어 있다. 이전 버전의 API 난독화를 해제하기 위해 관찰해야 하는 요소는 Table 1.과 같다.

먼저 API 난독화 과정에서 가상 메모리를 할당해 사용하므로, 할당받은 메모리에 난독화가 적용된 코드를 생성하는 과정을 관찰해 원본 API를 찾아냈다. 찾아낸 할당받은 메모리와 원본 API의 관계를 이용해 IAT(Import Address Table)[4]에 저장된 값을 복구해주고, 코드 영역에서 난독화 된 API를 직접 호출하는 명령어를 복구해 실행 파일에 적용된 API 난독화를 해제하였다. 마지막으로 앞의 과정에서 파악한 내용을 토대로 원본 API와 난독화가 적용된 API의 관계를 파악하는 알고리즘을 제시하였다.

Table 1. Objects to watch to figure out the API wrapping process in Themida v2.x

#	Object	Purpose
1	Memory allocation	To find a memory for API wrapping
2	Allocated memory	To find an API to obfuscate
3	IAT	To find a position of API on IAT
4	Code section	To find a patched CALL instruction

2.2 이전 버전의 Themida API 난독화 해제 자동화

앞서 알아본 연구에서 제안한 알고리즘은 이전 버전의 Themida API 난독화에서 원본 API와 난독화 된 API의 관계를 파악하는 것은 가능하지만, 난독화 된 API를 직접 호출하도록 코드 영역의 명령어를 패치하는 과정을 관찰하지 못한다.

본 장에서 알아보는 연구에서는 하나의 난독화 된 API를 생성한 후 코드 영역의 명령어를 패치하는 것을 알아냈으며, 이전 연구의 알고리즘을 발전시켜

코드 영역을 패치하는 과정까지 관찰이 가능한 알고리즘을 제시하였다. 또한, 제시한 알고리즘을 토대로 개발한 시스템이 잘 작동하는 것을 보였다[5].

이전 버전의 API 난독화를 해제하는 시스템은 Table 2.와 같은 단계를 거친다. 먼저 API 난독화를 위해 할당받은 메모리를 확인하고, 해당 메모리에 난독화 해 저장하려는 원본 API가 어떤 것인지를 찾는다. 원본 API를 찾으면 IAT에서 난독화 된 API가 저장된 위치를 원본 API의 주소로 복구한다. 마지막으로 코드 영역에서 난독화 된 API를 직접 호출하도록 패치된 것을 IAT를 참조해 호출하도록 복구한 후 각 섹션의 메모리를 덤프해 PE (Portable Executable)[6] 구조에 맞게 재조립하면 API 난독화가 해제된 실행 파일을 얻을 수 있다.

Fig. 1.은 위의 과정을 따라 실행 파일에 적용된 이전 버전의 Themida API 난독화를 자동으로 해제하는 시스템의 구조를 나타낸다. WinDbg와 PyKD를 이용해 API 난독화를 해제하고, 실행 파일을 재조합하며, ImpREC를 이용해 재조합한 실행 파일이 정상적으로 작동할 수 있도록 새로운 IT(Import Table)[7]를 생성해 준다. 해당 시스템을 거친 실행 파일은 API 난독화가 해제된 상태로 복구된다.

Table 2. Process to clear API wrapping on Themida v2.x

#	Process
1	Find memory for API wrapping
2	Match original API
3	Restore address on IAT table
4	Restore CALL
5	Rebuild PE

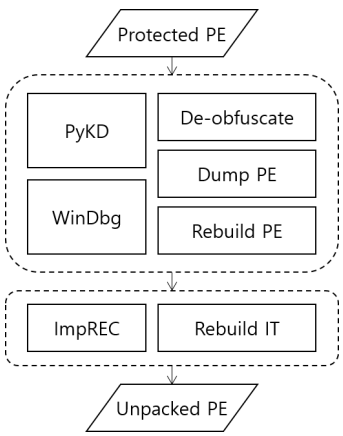

Fig. 1. Automated system to de-obfuscate Themida v2.x API wrapping

III. 가상 메모리 할당을 이용하지 않는 API 난독화 분석

3.1 API 난독화 작동 분석

최신 버전인 Themida v3.x의 API 난독화는 가상 메모리를 할당하지 않고, 실행 파일 내부의 영역을 사용한다. 난독화 된 API를 생성하는 대신 API의 주소를 난독화 해 저장하고, 함수 호출 시 이를 해석해 API를 호출한다. 그러므로 이전 연구의 해제 방안을 적용하더라도 가상 메모리 할당 함수에서 읽기, 쓰기, 실행 권한이 적용된 메모리를 할당하는 것은 관찰되지 않는다. 따라서 API 난독화의 작동을 분석하기 위해서는 새로운 접근 방식이 필요하다.

API의 주소는 라이브러리가 할당된 메모리에 직접 접근해 PE 구조를 따라가 읽는다. 라이브러리의 PE 구조를 직접 읽어서 사용하는 경우는 흔치 않으므로, 각 라이브러리가 할당된 메모리에 접근하는 것을 관찰하면, API를 찾기 위해 EXPORT 테이블[8]을 읽는 순간을 찾을 수 있다. Fig. 2.는 시스템 라이브러리인 USER32의 API를 사용하기 위해 라이브러리의 PE 구조를 직접 읽어 NT header[9]의 Data Directories[10]에 접근해 EXPORT 테이블의 RVA를 가져오는 것을 보여준다.

원하는 API를 찾으면 주소를 난독화 한 후 실행 파일 내부 영역에 저장한다. 이 과정에서 저장한 난독화 된 주소는 추후 API를 호출하는 과정에서 해석되어 원본 API의 주소로 나타나게 된다. 이때 API의 주소를 난독화 해 저장하는 영역은 실행 파일에 난독화를 적용하는 시점에 결정되므로, 동적으로 위치를 할당받는 작업은 존재하지 않는다. 따라서 난독화 된 주소를 저장하는 위치는 실행 파일을 다시 실행하더라도 변하지 않고 고정되어 있다.

실행 파일이 사용하는 모든 API의 주소를 난독화 해 저장하고 나면, 난독화 된 API 주소를 해석해 호출하는 함수를 IAT에 등록한다. 이 과정에서 등록하는 함수들 역시 실행 파일에 난독화를 적용하는 시점에 생성되어 주소가 고정되어 있으며, 분석이 어렵도

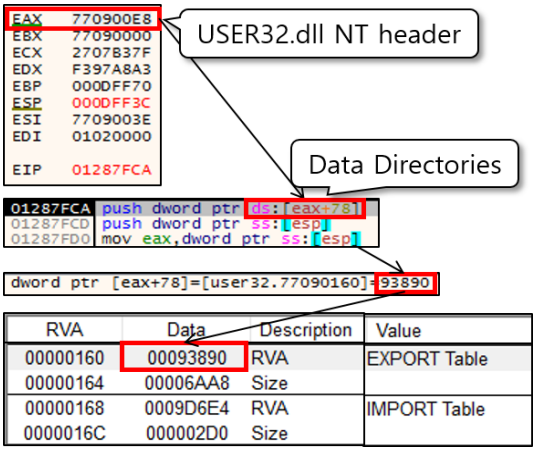

Fig. 2. Access to Data Directories in USER32.dll

록 난독화가 적용된 코드로 이루어져 있다. IAT에 함수 등록이 끝나면, OEP(Original Entry Point)에 도달해 원본 실행 파일의 코드가 실행된다. 이후 원본 실행 파일의 코드에서 API를 호출하게 되면 IAT를 읽어 난독화 된 API 주소를 해석해 API의 시작 주소로 이동한다. Fig. 3.은 IAT에 등록된 난독화 된 API 주소를 해석해 호출하는 함수에서 원본 API가 나타나는 것을 보여준다. 난독화 된 API 주소를 해석해 호출하는 함수의 주소 0x01024000이 IAT에 저장되어 있고, 해당 함수에서 난독화 된 주소인 0xA173C2B3을 해석해 시스템 라이브러리 USER32의 IsChild[11] API를 호출한다. 이 과정에서 주소 0x01024000의 함수는 USER32.IsChild와 관계가 있는 것을 파악할 수 있다.

최신 버전에서는 이전 버전과 달리 코드 영역에서 직접 난독화 된 API를 호출하도록 패치하는 작업이 존재하지 않는다. 따라서 원본 실행 파일과 최신 버

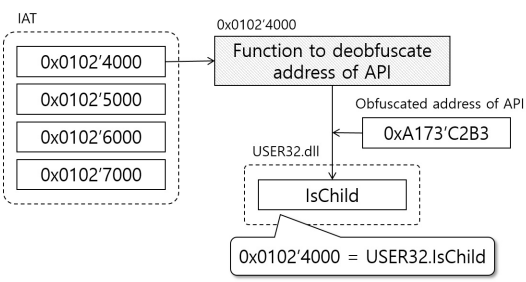

Fig. 3. Matching original API with function to deobfuscate address of API

전의 API 난독화가 적용된 실행 파일의 코드 영역은 차이가 나타나지 않는다. Table 3.은 원본 실행 파일과 최신 버전의 API 난독화가 적용된 실행 파일의 코드 영역 메모리를 덤프한 후 해시를 비교한 결과이다. IT가 가지고 있던 정보를 Themida가 삭제했기 때문에 해시가 차이가 나지만, IT를 제외한 코드 영역의 해시는 동일한 것을 알 수 있다.

Table 3. Comparison of MD5 hashes of CODE section between original file and file that protected by Themida v3.x

Section	Original
	Protected by Themida v3.0
CODE	95f86ffacbc0c10360db2a72698d9a3b
	bc06c1524e57f723437eafdfe08cfbb4
CODE (No IT)	f72f29b3813fa86472e0e9656c42d748
	f72f29b3813fa86472e0e9656c42d748

3.2 API 난독화 해제 방안

제안하는 가상 메모리 할당을 이용하지 않는 최신 버전인 Themida v3.x의 API 난독화를 해제 방안은 Table 4.와 같다. 먼저 시스템 라이브러리의 PE 구조를 읽어 EXPORT 테이블에 접근하는 순간을 찾는다. 그리고 API의 주소를 찾아 난독화 한 후, 난독화 된 주소를 어디에 저장하는지를 찾는다. Fig. 4.는 API의 주소를 찾아 해당 API의 주소를 난독화 한 값을 저장하는 과정을 추적해 기록한 결과이다. 원본 API의 주소가 알아볼 수 없는 형태의 값으로 변형되어 저장되는 것을 확인할 수 있다.

실행 파일이 사용하는 모든 API의 주소를 난독화해 저장하는 과정이 끝나면, IAT에 등록된 함수를

Table 4. Process to clear API wrapping on Themida v3.x

#	Process
1	Find accessing to EXPORT directory
2	Check where obfuscated addresses are stored
3	Analyze functions for call API on IAT
4	Match original API with obfuscated address
5	Restore address on IAT table
6	Rebuild PE

분석하며 따라가 난독화 된 주소에 접근하는 순간을 찾는다. 이때 호출하려는 원본 API의 주소에 접근하는 것을 찾지 않고, 함수의 작동 흐름을 분석해 난독화 된 주소에 접근하는 것을 찾는 이유는, 단순히 시스템 라이브러리의 API에 접근하는 순간을 찾으면 IAT를 거쳐서 들어온 호출인지, 일반적인 API 호출인지를 확인하는 과정이 추가로 필요하기 때문이다. Fig. 5.는 IAT에 등록된 함수에서 난독화 된 API 주소를 해석해 API의 주소를 알아내는 과정을 추적한 결과를 보여준다. 가장 먼저 읽어온 값 0xA173C2B3은 Fig. 4.에서 시스템 라이브러리인 user32의 API인 IsChild 주소를 난독화 해 저장한 값이다. 따라서 난독화 된 주소를 해석하는 과정의 마지막에 user32.IsChild의 주소가 나타난 것을 알 수 있다. 따라서 앞의 과정에서 찾은 정보들을 토대로 모든 난독화 된 주소와 원본 API의 관계를 찾을 수 있으며, 이를 활용해 IAT에 저장된 주소를 원본 API의 주소로 복구할 수 있다. IAT 복구가 완료된 상태에서 메모리를 덤프해 재조립하면 API 난독화가 해제된 실행 파일을 얻을 수 있다.

최신 버전인 Themida v3.x의 API 난독화는 이전 버전과 달리 API 난독화를 적용하는 시점에 결정되는 요소가 많아졌으므로, 추적할 요소를 정규화하기 어려워졌다. 또한, 관찰할 대상의 위치도 고정되어 있지 않기 때문에, 동적으로 난독화 과정을 추적하기도 어려워졌다. 최신 버전의 API 난독화를 해제하기 위해 관찰해야 하는 요소는 Table 5.와 같다. 시스템 라이브러리의 헤더와 IAT는 고정된 메모리 주소를 가지므로 정규화에 어려움이 없다. 하지만 난독화 된 API 주소를 저장하는 주소는 실행 파일에 난독화를 적용하는 시점에 저장되므로, 코드상에서 고정된 값으로 존재해 위치를 파악하기가 어렵다. 또한, IAT에 저장된 함수는 난독화 된 API에 직접 접근하는 방식이 아닌, 난독화 된 API 주소를 읽어오는 형태이므로 관련된 API를 파악하기가 어렵다. 따라서 자동화된 난독화 해제 시스템을 개발하기 위해서는, 난독화 된 API의 주소와 관련된 코드의 흐름을 파악하기 위한 고민이 필요하다.

Fig. 4. Save obfuscated address of API

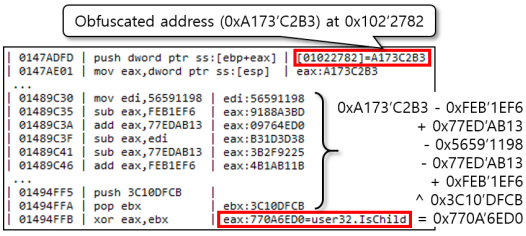

Fig. 5. Deobfuscation for get address of API

난독화 과정에서 실행되는 코드는 코드의 양을 늘려 분석을 방해할 목적으로 생성된 의미 없는 더미(dummy) 코드와 데이터를 난독화하는 코드로 나뉜다. 분석에 걸리는 시간을 효율적으로 줄이기 위해서는 난독화하는 코드를 식별해 더미 코드를 제거해야 하는데, 난독화하는 코드는 데이터를 사용한다는 점을 이용하면 식별할 수 있다. 여기서 데이터를 사용한다는 것은 데이터를 읽고, 어떤 연산을 하고, 저장하는 것을 의미한다. 이는 데이터의 이동을 추적하는 taint analysis[12]를 적용해 해결할 수 있을 것이다. 따라서 taint analysis 적용에 성공한다면, 자동화된 난독화 해제 시스템을 개발할 수 있을 것으로 생각된다.

Table 5. Objects to watch to figure out the API wrapping process in Themida v3.x

#	Object	Purpose
1	PE header in library	To find an API to obfuscate
2	Whole memory on PE	To find a memory for write an obfuscated API address
3	IAT	To find a function for de-obfuscating and calling an API
4	Function call via IAT	To figure out a relation between API and function taht on IAT

3.3 API 난독화 해제 결과

최신 버전의 Themida로 실행 파일을 보호한 후,

제안하는 해제 방안을 적용해 API 난독화를 해제하였다. 실험환경은 Windows 10 x64이고, 사용한 Themida의 버전은 3.0이다. 실험은 Windows의 기본 실행 파일인 계산기(calc.exe)와 메모장(notepad.exe) 그리고 x86 PE 구조를 보여주는 프로그램인 PE view(peview.exe)를 보호한 후 API를 복구하였다. 제안하는 해제 방안을 적용한 복구 결과는 Table 6.과 같다. 실행 파일이 사용하는 API 중 일부 또는 전체가 난독화되었으며, 난독화가 적용된 API는 모두 정상적으로 복구되었다.

Fig. 6.은 API 난독화가 적용된 상태와 복구 후의 CALL 명령어를 보여준다. 위의 API 난독화가 적용된 상태에서는 호출하는 함수를 파악할 수 없어 아무런 정보를 보여주지 않지만, 복구 후에는 원본 API의 정보를 정상적으로 표시해주는 것을 확인할 수 있다. 복구 후에는 별다른 분석 없이 API만으로도 다이얼로그의 텍스트를 표시해주는 작업임을 간단히 파악할 수 있다.

Table 6. Result of restoring Obfuscated API

File (.exe)	Number of API	Obfuscated	Restored
calc	132	82	82
notepad	201	145	145
peview	115	115	115

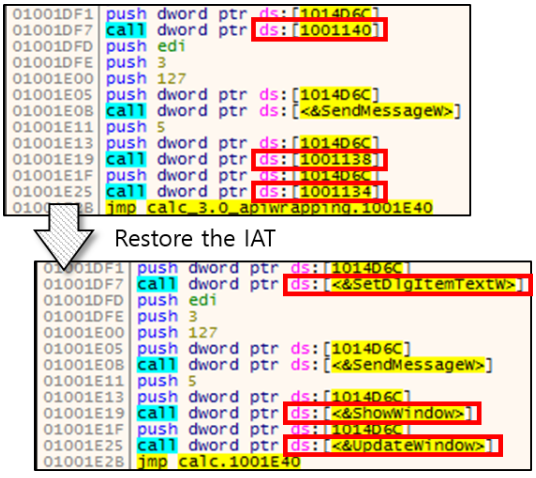

Fig. 6. Restored IAT points the original API

IV. 결 론

API는 프로그램 분석시 활용할 수 있는 유용한 정보이므로, API 정보를 복구하면 분석을 효율적으로 진행할 수 있게 된다. 따라서 Themida의 새로운 메이저 버전인 3.0의 API 난독화를 분석하고, 이에 대응할 방안을 기술하였다. 최신 버전의 API 난독화는 가상 메모리 할당을 이용하지 않고, 실행 파일에 난독화를 적용하는 시점에 결정되는 요소가 많아져 추적할 요소를 정규화하기 까다로워졌다. 그러므로 본 논문에서 기술한 내용만으로는 분석가가 직접 난독화 과정을 추적할 수는 있지만, 정규화된 과정이 없어 자동화된 시스템을 구축하기에는 무리가 있다. 그러나 API 난독화 추적을 시작할 수 있는 요소인 원본 API의 주소를 읽어오는 부분을 찾는 것이 가능하므로, 데이터의 이동을 추적하는 taint analysis를 이용한다면 정규화 없이도 저장하는 주소를 찾을 수 있을 것으로 생각된다. 따라서 앞으로는 본 연구를 발전시켜 최신 버전의 API 난독화를 자동으로 해제할 수 있도록 시스템을 구축하는 연구를 진행할 예정이다.

References

[1] Yan, Wei, Zheng Zhang, and Nirwan Ansari, "Revealing packed malware", IEEE seCurity & PrivaCy, Vol. 6.5 pp. 65-69, Oct. 2008

[2] Oreans Technologies, Themida, Advanced Windows software protection system, Revision 3.0, Aug. 2019 (also see Themida, https://www.oreans.com/Themida.php)

[3] Jae-hwi Lee, et al, "A Study on API Wrapping in Themida and Unpacking Technique", Journal of The Korea Institute of Information Security & Cryptology, Vol. 27, No. 1, pp. 67-77, Feb. 2017

[4] Microsoft, Microsoft Portable Executable and Common Object File Format Specification, Revision 8.2, Sep. 2010 (also see Understanding the Import Address Table, http://sandsprite.com/C

odeStuff/Understanding_imports.html)
〔5〕 Jae-hwi Lee, et al, "A Study on the Automation of Unpacking API Wrapping in Themida", Journal of Digital Forensics, 11(1), pp. 37-46, Jun. 2017
〔6〕 Microsoft, Microsoft Windows Dev Center, PE Format, Mar. 2019 (also see PE Format, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format)
〔7〕 Microsoft, Microsoft Docs, Peering Inside the PE: A Tour of the Win32 Portable Executable File Format: PE File Imports, Jun. 2010 (also see Microsoft Docs, Peering Inside the PE: A Tour of the Win32 Portable Executable File Format: PE File Imports, https://docs.microsoft.com/en-us/previous-versions/ms809762(v=msdn.10)#pe-file-imports)
〔8〕 Microsoft, Microsoft Docs, Peering Inside the PE: A Tour of the Win32 Portable Executable File Format: PE File Exports, Jun. 2010 (also see Microsoft Docs, Peering Inside the PE: A Tour of the Win32 Portable Executable File Format: PE File Exports, https://docs.microsoft.com/en-us/previous-versions/ms809762(v=msdn.10)#pe-file-exports)
〔9〕 Microsoft, Microsoft Windows Dev Center, PE Format: File Headers, Mar. 2019 (also see Microsoft Windows Dev Center, PE Format: File Headers, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#file-headers)
〔10〕 Microsoft, Microsoft Windows Dev Center, PE Format: Optional Header Data Directories, Mar. 2019 (also see Microsoft Windows Dev Center, PE Format: Optional Header Data Directories, https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#optional-header-data-directories-image-only)
〔11〕 Microsoft, Microsoft Windows Dev Center, IsChild function, May 2018 (also see IsChild function, https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-ischild)
〔12〕 Schwartz, Edward J., Thanassis Avgerinos, and David Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask).", 2010 IEEE symposium on Security and privacy, pp. 317-331, May 2010

〈저자소개〉

이 재 휘 (Jae-hwi Lee) 정회원
2015년 2월: 경북대학교 컴퓨터학부 공학사 졸업
2017년 8월: 고려대학교 정보보호대학원 석사 졸업
2018년 3월~현재: NAVER Corp. 보안부서 근무
〈관심분야〉 정보보호, 역공학

이 병 희 (Byung-hee Lee) 정회원
2002년 2월: 동국대학교 컴퓨터공학과 학사 졸업
2004년 8월: 한국과학기술원 전산학과 석사 졸업
2004년 8월~2008년 6월: 삼성전자 정보통신 총괄 근무
2008년 6월~현재: NAVER Corp. 보안부서 근무
〈관심분야〉 정보보호, 네트워크 보안, 침입 탐지

조 상 현 (Sang-hyun Cho) 정회원
1997년 2월: 고려대학교 컴퓨터학과 학사 졸업
1999년 2월: 한국과학기술원 전산학과 석사 졸업
2005년 2월: 한국과학기술원 전산학과 박사 졸업
2007년~현재: NAVER Corp. 정보보호최고책임자(CISO)
2014년~현재: 고려대학교 정보보호대학원 겸임교수
〈관심분야〉 정보보호, 네트워크 보안, 침입 탐지, 어뷰즈 탐지