Themida의 API 난독화 복구 자동화에 관한 연구

A Study on the Automation of Unpacking API Wrapping in Themida

저자
(Authors)

이재휘, 이상진
Jaehwi Lee, Sangjin Lee

출처
(Source)

디지털포렌식연구 11(1), 2017.06, 37-46 (10 pages)
Journal of Digital Forensics 11(1), 2017.06, 37-46 (10 pages)

발행처
(Publisher)

한국디지털포렌식학회

URL

http://www.dbpia.co.kr/Article/NODE07217922

APA Style

이재휘, 이상진 (2017). Themida의 API 난독화 복구 자동화에 관한 연구. 디지털포렌식연구, 11(1), 37-46.

ISSN : 1976-5304
http://www.kdfs.or.kr

Themida의 API 난독화 복구 자동화에 관한 연구

이 재 휘*, 이 상 진**
고려대학교 정보보호대학원

A Study on the Automation of Unpacking API Wrapping in Themida

Jaehwi Lee*, Sangjin Lee**
Center for Information Security Technologies, Korea University

요 약

프로텍터는 실행파일에 분석방해 기술을 적용해 실행파일 내부 알고리즘을 파악하기 어렵게 만드는 프로그램이다. 프로텍터는 전문지식이 없어도 손쉽게 높은 수준의 분석방해 기술을 적용할 수 있게 해주는 유용한 프로그램이지만, 범죄에 악용되어 범죄행위 입증을 방해하는 경우도 발생한다. 프로텍터가 적용된 실행파일은 높은 수준의 기술을 가진 분석가가 오랜 시간 분석을 진행해야 한다. 만약 프로텍터가 적용된 실행파일이 발견될 때 마다 분석가가 분석방해 기술을 우회하는 것은 비효율적이므로, 프로텍터 해제를 자동화 해 분석시간을 단축할 필요가 있다. 본 논문에서는 상용 프로텍터인 Themida 프로텍터가 적용된 실행파일에서 분석방해 기술인 API 난독화를 해제하는 자동화 알고리즘을 제안해 프로텍터 해제 작업의 자동화 가능성을 살펴본다.

주제어 : 자동화, 언패킹, 더미다, API 난독화

ABSTRACT

Protector is a program that makes it difficult to understand the algorithm inside the executable file by applying the analysis interruption technique to the executable file. Protector is an useful program that allows you to easily apply high-level analysis interruption techniques without expert knowledge, but it can also be misused to disturb the investigation on the criminal activity. Executables files that applied the protector should be analyzed by a high-level analyst for a long time. It is inefficient to analyze the protected executables everytime, so it is necessary to shorten analysis time by automation. In this paper, we propose an automation algorithm that removes the API wrapping technique in the executable protected by Themida protector.

Key Words : Automation, Unpacking, Themida, API wrapping

▪ Received 03 March 2013, Revised 04 April 2013, Accepted 5 May 2013
▪ 제1저자(First Author) : Jaehwi Lee (Email : slimv.dfrc@gmail.com)
▪ 교신저자(Corresponding Author) : Sangjin Lee (Email : sangjin@korea.ac.kr)

Ⅰ. 서 론

프로텍터는 실행파일에 분석방해 기술을 적용해 실행파일 내부 알고리즘을 파악하기 어렵게 만드는 프로그램이다. 상용 프로텍터의 경우 버튼 클릭만으로 손쉽게 프로텍터를 적용할 수 있으며, 주로 개인 또는 회사가 개발한 소프트웨어의 핵심 알고리즘이 역공학을 통해 드러나는 것을 막고 싶을 때 사용한다. 프로텍터는 전문지식이 없어도 손쉽게 높은 수준의 분석방해 기술을 적용할 수 있게 해주는 유용한 프로그램이지만, 범죄자가 악용해 프로텍터를 적용한 실행파일을 범죄에 사용하는 경우도 존재한다. 만약 범죄에 활용된 실행파일에 프로텍터가 적용된 경우, 범죄 행위를 밝히기 위해서는 프로텍터를 우선적으로 해제해야 한다. 프로텍터가 적용된 실행파일은 높은 수준의 기술을 가진 분석가가 오랜 시간 분석을 진행해야 하므로, 프로텍터가 적용된 실행파일이 발견될 때 마다 분석가가 분석방해 기술을 우회하는 것은 비효율적이다. 따라서 프로텍터 해제 자동화를 통해 분석시간을 단축할 필요가 있다. 본 논문에서는 상용 프로텍터인 Themida[1]가 적용된 실행파일에서 분석방해 기술을 해제하는 자동화 알고리즘을 제안하고, 프로텍터가 적용된 실행파일에서 프로텍터를 해제하는 작업의 자동화 가능성을 살펴본다.

본 논문의 Ⅱ장에서는 기존에 진행된 Themida가 사용하는 분석방해 기술들을 분석하고 우회하는 방안을 제시한 연구에 대해 알아보고, Ⅲ장에서는 Themida의 가지는 분석방해 기술에서 기존의 연구가 해결하지 못한 부분에 대해 설명하고, Ⅱ장에서 소개한 우회 방안을 발전시킨 알고리즘을 제안한다. Ⅳ장에서는 다양한 버전의 Themida가 적용된 실행파일들에 제안한 알고리즘을 적용하고, 그 결과를 확인한다. 마지막 Ⅴ장에서는 본 연구의 결론 및 향후 연구에 대해 기술한다.

Ⅱ. 관련연구

프로텍터 적용이 보편화 되며 이에 대응하기 위한 연구도 진행되고 있다. 그중에 프로텍터 해제에 관한 연구로는 상용 프로텍터인 Themida가 가지는 분석방해 기술인 API 난독화의 작동 과정을 분석해 원본 API와 난독화 된 함수주소의 관계를 파악하는 연구가 진행되었다[2]. 이를 이용해 IAT(Import Address Table)[3]와 코드영역에서 사용하는 난독화 된 함수주소를 찾아 원본 API를 사용하도록 복구하면 Themida의 분석방해 기술들이 해제된 원본 실행파일을 얻을 수 있다. 그러나 앞선 연구에서는 원본 API와 난독화 된 함수주소의 관계를 찾기 위한 알고리즘만이 제시되어 있어 코드영역에서 난독화 된 함수주소를 사용하는 위치를 탐색하고 복구하는 작업은 분석가가 직접 수행해야 하는 단점이 있다. 따라서 코드영역에서 난독화 된 함수를 사용하는 부분을 찾아내고 복구하는 알고리즘이 필요하다.

Ⅲ. Themida의 API 난독화 해제 방안

자동화 된 Themida 프로텍터 해제 시스템을 만들기 위해서는 Ⅱ장에서 살펴본 API 난독화를 해제하는 연구에서 추가적으로 난독화 된 함수를 사용하는 코드영역을 찾아내고 복구하는 알고리즘이 필요하다. 이를 마련하기 위해 우선적으로 코드영역에서 어떤 방식으로 난독화 된 함수를 사용할 수 있도록 변경하는지를 살펴보고, 기존의 알고리즘에서 이를 해결하는 알고리즘을 추가해 제안한다.

3.1 API 난독화에서의 코드영역 변경

Themida는 난독화 대상이 되는 API에 접근해 새로운 메모리 영역에 복사해 난독화하고, 난독화 된 함수의 주소를 IAT에 저장한다. 그 후 코드영역에서 해당 함수를 사용해야 하는 위치를 모두 찾아 난독화 된 함수를 호출하는 CALL 명령어를 쓰게 된다. 위의 작업을 난독화 대상이 되는 모든 API에 대해 작업하며, 이는 [그림 1]으로 나타낼 수 있다. 코드 수정 과정에서는 [그림 2]와 같이 아무 작업도 하지 않는 명령어인 NOP(No Operation)[4]으로 초기화 되어있는 영역을 난독화 된 함수를 호출하는 CALL 명령어로 변경하게 된다.

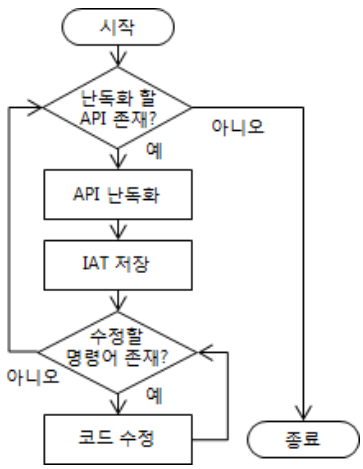

그림 1. Themida의 API 난독화 과정

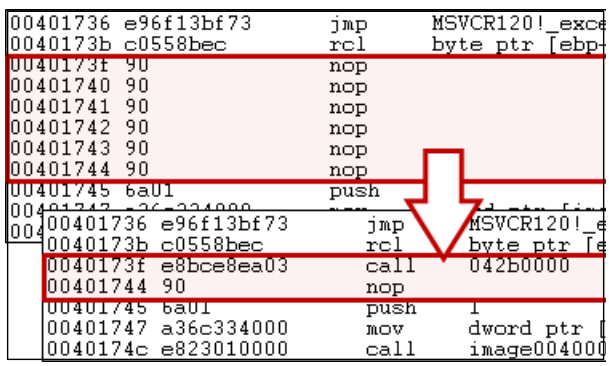

그림 2. 난독화 함수를 호출하는 CALL 명령어로 수정

3.2 제안하는 알고리즘

API 난독화 과정에서 코드영역을 수정하는 작업은 IAT에 난독화 된 함수의 주소를 저장하는 작업을 기준으로 나누어지므로, IAT에 난독화 된 함수의 주소를 저장하는 작업 다음으로 코드영역에 속하는 주소에서 메모리 쓰기 작업이 발생하면 API 난독화의 코드영역 수정 작업으로

생각할 수 있다. [그림 3]은 API 난독화 과정에서 원본 API와 난독화 된 함수의 주소 관계를 파악하고, 코드영역을 수정하는 작업의 주소를 찾아 복구하는 알고리즘을 나타낸다. 우선 원본 API와 난독화 된 함수의 주소 관계를 파악하는 단계로 메모리 할당을 관찰해 난독화에 사용될 메모리를 탐지한다. 난독화에 사용하기 위해서는 메모리의 접근권한이 읽기, 쓰기, 실행 모두 가지도록 설정되므로 이를 이용해 난독화에 사용되는 메모리를 탐지할 수 있다. 그리고 난독화를 위해 할당된 메모리에 메모리 복사가 발생하고, 복사 대상의 주소가 API로 확인되면 해당 메모리는 확인된 API가 난독화되어 저장된다고 판단한다. 그리고 메모리 쓰기가 발생한 경우 쓰기 목적지의 주소가 IAT 영역에 속한다면, 난독화 된 함수 주소를 IAT에 저장하는 작업으로 판단하고 메모리에 쓰는 값과 IAT상의 주소를 기록한다. 만약 코드영역에 속하는 경우 메모리에 쓰는 값이 난독화 된 함수의 주소인 경우 API 난독화 과정에서 코드영역을 수정하는 작업으로 판단하여 해당 난독화 된 함수의 원본 API 주소를 가지는 IAT를 참조하는 CALL 명령어로 복구한다.

[그림 4]는 제안하는 알고리즘을 이용해 IAT에 저장되는 난독화 된 함수의 주소의 원본 API 주소 정보와 코드영역에서 난독화 된 함수를 사용하는 CALL 명령어의 위치정보를 찾아낸 결과이다. 찾아낸 정보를 활용해 IAT가 원본 API의 주소를 가지도록 복구하고, 코드영역을 IAT를 참조해 함수를 호출하도록 변경하면 실행파일에 적용된 API 난독화를 무력화 할 수 있다.

```
lastEvt := 최근에 발생한 이벤트          lMem := 메모리 영역을 저장하는 리스트
newMem := 최근에 할당받은 메모리         dictWrap := 메모리 주소와 API의 관계를 저장하는 사전
copySrc := 복사 대상의 주소              dictIat := IAT 상의 주소와 API의 관계를 저장하는 사전
copyDst := 복사 목적지의 주소            GetLastEvent() : 가장 최근에 발상한 이벤트 반환
writeDst := 쓰기 목적지의 주소           GetApiName() : 주어진 주소의 API 이름 반환
writeVal := 메모리에 쓸 데이터           GetCallDest() : 주어진 주소의 CALL 명령어가 호출하는
nameApi := API의 이름                    주소 반환
sIat := IAT 테이블의 범위                FixCall() : 주어진 주소의 함수 호출 명령어를 IAT를 참조하
sCode := 코드영역의 범위                 도록 변경

while lastEvt := GetLastEvent()
  if lastEvt = EVT_ALLOC_MEMORY then
    if newMem.protect = READ | WRITE | EXECUTE then
      lMem.append(newMem)
  else if lastEvt = EVT_MEM_COPY then
    if copyDst ∈ lMem then
      nameApi := GetApiName(copySrc)
      if nameApi != API_UNKNOWN then
        dictWrap[copyDst] := copySrc
  else if lastEvt = EVT_MEM_WRITE then
    if writeDst >= sIat.base and writeDst < sIat.end then
      dictIat[writeVal] := writeDst
    else if writeDst >= sCode.base and writeDst < sCode.end then
      if GetCallDest(writeDst) ∈ dictWrap then
        FixCall(writeDst, dictIat[dictWrap[GetCallDest(writeDst)]])
```

그림 3. API 난독화 및 코드영역 수정 정보 파악 및 복구 방법

```
[IAT] 0x0040206c <= 0x7000480c ( 0x7000480c : MSVCR120!__crtUnhandledException )
 - 0x0040187a
 - 0x0040187b
 - 0x0040187c
[IAT] 0x00402070 <= 0x700047f7 ( 0x700047f7 : MSVCR120!__crtTerminateProcess )
 - 0x00401880
 - 0x00401881
 - 0x00401882
[IAT] 0x00402074 <= 0x6ffdbbb8 ( 0x6ffdbbb8 : MSVCR120!_exit )
 - 0x0040129f
 - 0x004012a0
 - 0x004012a1
```

그림 4. IAT의 원본 API 및 코드영역 수정 정보 파악 결과

Ⅳ. Themida의 API 난독화 해제

4.1 실행파일 복구 실험환경

하나의 실행파일을 다양한 버전의 Themida를 이용해 API 난독화를 적용한 후, 제안하는 알고리즘을 이용해 실행파일을 복구하였다. 실험환경은 Windows 7 SP1 x86이고, 사용한 Themdia의 버전은 [표 1]와 같다. 실험대상 실행파일은 Visual Studio 2013으로 컴파일 해 사용하였다. 또한 제안하는 알고리즘을 구현하기 위해 Microsoft사가 제공하는 디버거인 WinDbg[5]를 기반으로 Python을 사용할 수 있는 기능을 제공하는 PyKd[6]를 사용하였다. 또한 마지막으로 실행중인 프로세스에서 IAT의 정보를 기반으로 임포트 테이블[7]을 생성해주는 ImpREC[8]를 사용해 복구한 실행파일에 임포트 테이블을 추가하였다.

표 1. 사용한 Themida 버전

Themida 버전
v2.2.6.0
v2.4.5.0
v2.4.6.0

4.2 난독화 해제 절차

Themida의 API 난독화가 적용된 실행파일에서 제안하는 알고리즘을 이용해 자동으로 원본 실행파일을 복구하는 과정에 대해 설명한다. [표 2]은 실행파일을 복구하는 전체 과정을 단계별로 나타낸 것이다. 제안한 알고리즘을 이용해 API 난독화를 해제하고, 코드 난독화가 해제 된 상태의 메모리를 획득하기 위해 OEP(Original Entry Point)에 도달한 상태에서 메모리를 덤프한다. 덤프

한 메모리 파일들을 윈도우즈 실행파일 PE(Portable Executable)[9] 구조에 맞게 재조합하면 API 난독화가 해제된 실행파일을 획득할 수 있다. 실행파일을 복구하는 과정을 자동으로 처리해 주는 시스템의 구조는 [그림 5]와 같다. 해제 절차의 1단계와 2단계를 난독화 해제 프로세스에서 처리하고, 실행파일 구조 획득 프로세스에서 3단계로 OEP에 도달한 상태에서 메모리를 덤프해 실행파일의 구조를 획득한다. 그리고 실행파일 재조합 프로세스에서 앞서 덤프한 메모리 파일들을 실행파일 구조에 맞게 재조합하는 4단계 절차를 진행한다. 생성된 파일에는 아직 임포트 테이블이 존재하지 않으므로, WinDbg가 연결된 프로세스를 ImpREC가 읽어 임포트 테이블을 생성한 후 생성된 파일에 추가해 5단계를 처리한다. 이 모든 과정을 거치고 나면 실행 가능한 형태로 복구된 실행파일이 완성된다.

표 2. Themida의 API 난독화 해제 절차

#	단계	방법
1	API 난독화 해제	원본 API 복구
		코드영역 복구
2	코드 난독화 해제	OEP 탐지
3	실행파일 구조 획득	메모리 덤프
4	실행파일 재조합	메모리 덤프 조합
5	임포트 테이블 복구	IAT 정보 기반 임포트 테이블 복구

그림 5. Themida의 API 난독화 해제 시스템 구조

4.3 난독화 해제 결과

다양한 버전의 Themdia가 적용된 실행파일을 난독화 해제 시스템에 입력해 복구하였다. 복구한 모든 실행파일은 정상적으로 작동하였다. [표 3]는 실행파일별 크기를 보여준다. 실행파일에서 Themida가 생성한 보호기술이 제거되었으므로 실행파일의 크기가 줄어들었다. 원본 실행파일의 크기와 복구한 실행파일의 크기가 차이나는 원인은 메모리 단위로 덤프를 떠 페이지의 최소단위인 SYSTEM_INFO.dwPageSize[10]에 맞춰 메모리를 획득한 것과 추가한 임포트 테이블의 크기이다. [표 4]는 Themdia가 적용된 실행파일의 코드영역 해시 값과 복구한 실행파일에서의 코드영역 해시 값을 보여준다. API 난독화의 영향으로 세 파일 모두 복구 이전에는 서로 다른 해시 값을

가지고 있었지만, 복구 후에는 API 난독화로 인해 난독화 된 함수를 호출하는 CALL 명령어가 모두 복구되어 원본 실행파일의 코드영역 해시 값과 동일한 값을 가지는 것을 알 수 있다. 실제로 [그림 6]을 통해 원본 실행파일과 복구한 실행파일의 코드를 보면 동일한 명령어를 가지고 있는 것을 알 수 있다.

표 3. 실행파일 크기 비교

파일 유형		실행파일 크기 (KB)	
원본 실행파일		7	
Themida		복구 전	복구 후
	v2.2.6.0	1,129	24
	v2.4.5.0	1,803	24
	v2.4.6.0	1,845	24

표 4. 실행파일 복구 전과 후 해시 값 비교

파일 유형	복구 전 코드영역 해시 값 (MD5)
	복구 후 코드영역 해시 값 (MD5)
원본 파일	7875b054b988666c770b476c6f847007
Themida v2.2.6.0	a19cff4c207707d3dad548acd3ad2933
	7875b054b988666c770b476c6f847007
Themida v2.4.5.0	444f5feedfb9570864ba135e4682c21f
	7875b054b988666c770b476c6f847007
Themida v2.4.6.0	3e7acad44f0e77e1fe2f8aeb605848c2
	7875b054b988666c770b476c6f847007

원본 실행파일

```
0:000> u 401000 L10
HelloThemida!printVal [c:\users\slimv0x00\documents\visual studio 2013\proje
00401000 55              push    ebp
00401001 8bec            mov     ebp,esp
00401003 8b4508          mov     eax,dword ptr [ebp+8]
00401006 50              push    eax
00401007 6800214000      push    offset HelloThemida!GS_ExceptionPointers+0x
0040100c ff158c204000    call    dword ptr [HelloThemida!_imp__printf (00402
00401012 83c408          add     esp,8
00401015 5d              pop     ebp
00401016 c3              ret
```

복구한 실행파일

```
0:000> u 401000 L10
image00400000+0x1000:
00401000 55              push    ebp
00401001 8bec            mov     ebp,esp
00401003 8b4508          mov     eax,dword ptr [ebp+8]
00401006 50              push    eax
00401007 6800214000      push    offset image00400000+0x2100 (00402100)
0040100c ff158c204000    call    dword ptr [image00400000+0x208c (0040208c)]
00401012 83c408          add     esp,8
00401015 5d              pop     ebp
00401016 c3              ret
```

그림 6. 원본 실행파일과 복구한 실행파일의 코드

Ⅴ. 결 론

Themida 프로텍터가 사용하는 분석방해 기술을 우회하는 기존 연구의 알고리즘으로는 해결하지 못하는 코드영역 복구 부분에 대해 연구하고, 이를 해결할 수 있는 알고리즘을 추가적으로 제안하였다. 만약 프로텍터가 적용된 실행파일을 분석해야 하는 상황이 발생할 때 분석가가 프로텍터를 해제하는 작업을 하지 않고, 자동화 된 시스템의 도움을 받을 수 있다면 분석 작업이 훨씬 간단해 질 것으로 기대한다. 본 논문에서는 제안하는 알고리즘을 이용해 분석가의 개입 없이 다양한 버전의 Themida가 적용된 실행파일에서 원본 실행파일을 복구하는 자동화 된 시스템을 개발하는데 성공하였다. 현재는 Themida에 한정되어 알고리즘 적용이 가능하다는 한계가 있지만, 앞으로의 연구를 통해 더 많은 프로텍터에 대응 가능한 알고리즘을 찾아내어 자동화 할 것이다.

[투고일] 2017.06.13 [심사일] 2017.06.14 [게재확정일] 2017.06.24

참 고 문 헌 (References)

[1] Oreans Technologies, Themida, Advanced Windows Software Protection System (also see Themida, https://www.oreans.com/themida.php)

[2] Jaehwi Lee, et al. “A Study on API Wrapping in Themida and Unpacking Technique”, Journal of The Korea Institute of Information Security and Cryptology, Vol. 27, No. 1, pp. 67-77, Feb. 2017

[3] 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/CodeStuff/Understanding_imports.html)

[4] Intel, “NOP-No Operation”, Intel® 64 and IA-32 architectures software developer’s manual volume 2B: Instruction set reference, M-U, pp. 163, Jun. 2016 (also see Intel® 64 and IA-32 Architectures Software Developer Manuals, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html)

[5] Microsoft, Debugging tools for Windows (also see Download Debugging tools for Windows, https://developer.microsoft.com/en-us/windows/hardware/download-windbg)

[6] Microsoft Public License, Python Extension for WInDbg (also see, Python Extension for WinDbg, https://pykd.codeplex.com/license)

[7] Matt Pietrek, Microsoft, “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format”, Mar. 1994 (also see https://msdn.microsoft.com/en-us/library/ms809762.aspx)

[8] MackT, IAT and Portable Executable Rebuilding (also see, Import REConstructor 1.7e FINAL, https://tuts4you.com/download.php?view.415)

[9] Microsoft, Microsoft PE and COFF Specification, Revision 8.3, Feb. 2013 (also see Microsoft PE and COFF Specification, https://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx)

[10] Microsoft, Microsoft MSDN: SYSTEM_INFO structure (also see SYSTEM_INFO structure, https://msdn.microsoft.com/en-us/library/windows/desktop/ms724958(v=vs.85).aspx)

著者紹介

이 재 휘 (Jaehwi Lee)

학생회원
2015년 2월 : 경북대학교 컴퓨터학부 공학사
2015년 3월 ~ 현재 : 고려대학교 정보보호대학원 석사과정
<관심분야> 디지털 포렌식, 역공학

이 상 진 (Sangjin Lee)

정회원
1989년 10월 ~ 1999년 2월 : ETRI 선임 연구원
1999년 3월 ~ 2001년 8월 : 고려대학교 자연과학대학 조교수
2001년 9월 ~ 현재 : 고려대학교 정보보호대학원 교수
2008년 3월 ~ 현재 : 고려대학교 디지털포렌식연구센터 센터장
2017년 3월 ~ 현재 : 고려대학교 정보보호대학원 원장
<관심분야> 디지털 포렌식, 심층 암호, 해쉬 암호