악성코드 분석을 위한 “실행압축” 해제 기법

NCSC-TR04025

국가사이버안전센터
National Cyber Security Center

악성코드 분석을 위한 "실행압축" 해제 기법

1. 개 요

이 제목을 보고 "실행압축이 뭐야?" 하는 이도 있을 테고, "실행하면 자동으로 압축이 풀리는 ZIP 파일과 비슷한 거 아냐?" 하고 떠올리는 이도 있을 것이다. 그러나 여기서 설명하는 실행압축은 그 대상이 다르다. 흔히 말하는 ZIP, RAR처럼 데이터들을 하나로 묶어 놓는 압축과는 달리 그 대상이 notepad.exe처럼 실행할 수 있는 파일을 압축한 것으로, 실행압축 된 notepad.exe는 압축을 푸는 과정없이 바로 프로그램을 실행할 수 있다.

그럼 이런 실행압축을 왜 하는 것일까? 예전처럼 하드디스크 용량이 적었을때에는 자주 쓰지 않는 파일을 압축해 놓아 디스크 사용량을 늘렸던 적이 있었지만 오늘날에는 하드 디스크 용량이 넉넉하기 때문에 그럴만한 이유가 없을 것이라고 생각할 수도 있다.

그러나 인터넷 인프라가 잘 갖춰진 우리나라의 경우는 예외겠지만, 압축을 사용할 경우 짧은 시간 동안 프로그램을 다운받을 수 있기 때문에 요즘에도 온라인상에 있는 파일들은 대부분 압축이 되어 있다. 특히 실행 파일의 경우 ZIP과 같은 범용 데이터 압축보다는 실행압축 방식을 사용하는 쪽이 더 용량이 적게 사용되기 때문이다.[8]

악성코드도 이런 점을 이용해 짧은 시간 안에 많은 곳으로 전파되도록 실행압축을 사용하고 있으며, 백신 제작자들로 하여금 악성코드를 분석하기 어렵도록 하는데도 사용된다. 분석하는데 시간이 걸리는 동안 악성코드 전파 시간을 늘릴 수 있기 때문이다. 이와 반대로 만약 실행압축을 빨리 해제하여 분석할 수 있다면, 그 피해규모 또한 현저히 줄일 수 있을 것이다.

그렇다면 실제 실행압축을 이용하는 악성코드(웜, 바이러스로 한정)에는 어떤 것들이 있을까? 대표적인 것으로 MyDoom, Netsky, Bagle, Agobot, Welchia, Sasser, Sobig 변종 등을 들 수 있다. 바이러스 분야에 어느

8. 한글 2002 Hwp.exe 원본 크기는 1,626,112바이트이고, 압축프로그램으로 압축된 ZIP 파일은 670,508바이트, UPX 방식의 실행압축을 사용할 경우 563,200바이트가 되기 때문에 통신 상에서 판매하는 쉐어웨어의 경우 실행파일 압축이 되어 있는 경우가 많다.

정도 관심이 있다면 쉽게 알만한 큰 피해를 주었던 웜 · 바이러스들이다. 이와 같은 악성코드의 분석을 위해 보다 빠른 실행압축 해제 방법을 습득하는 것은 앞으로 출현할 악성코드에 대비하는데도 큰 도움이 된다.

2. PE(Portable Executable) 구조 및 실행압축 기본 원리

실제 실행압축을 분석하기 전에, 알아야 할 PE 구조 및 그 실행압축 기본 원리를 알아보자.

[그림 30] PE 구조

Dos Stub 부분은 Dos에서 실행할 때 "This program cannot be run in DOS mode"를 출력해주는 부분과 파일이 윈도우 실행파일 형식인 PE 포맷이라는 걸 알려주는 부분이 들어있다.

File Header에는 이 실행 파일이 어느 시스템(i386, mips, alpha, powerpc)에서 실행될 수 있는지를 알 수 있는 정보가 들어있다. 다음 Optional Header에는 실행 정보에 대한 기본 정보들이 들어있다.(이후로 나오는 데이터 구조들은 Visual C++ 프로그램에 포함된 Winnt.h 파일 내에 포함된 내용중 관련 부분을 표시한 것이다.)

```
// Standard fields.
  DWORD   SizeOfCode;                 // 코드 크기
  DWORD   SizeOfInitializedData;
  DWORD   SizeOfUninitializedData;
  DWORD   AddressOfEntryPoint;        // 프로그램이 코드가 시작할 주소
  DWORD   BaseOfCode;
  DWORD   BaseOfData;

  // NT additional fields.
  DWORD   ImageBase;                  // 프로그램이 상주할 기본 주소
  DWORD   SectionAlignment;           // 메모리 상 Section 기본 단위
  DWORD   FileAlignment;              // 파일 상 Section 기본 단위
  DWORD   SizeOfImage;
  DWORD   SizeOfHeaders;              // 다음에 설명할 Section Headers의 수
  WORD    Subsystem;                  // Windows Dos 창 프로그램, 또는 윈도우 프로그램

  IMAGE_DATA_DIRECTORY DataDirectory[15];
```

제일 아래에는 IMAGE_DATA_DIRECTORY 배열이 있는데, 그 중 첫 번째 요소인 실행파일에서 외부에서 사용할 수 있는 함수 목록을 가리키는 Export Directory, 두 번째 요소인 내부에서 사용하는 함수 목록을 가리키는 Import Directory, 열세 번째 요소인 내부에서 사용하는 함수 목록의 주소를 가리키는 Import Address Table 주소값을 담고 있다.

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT      0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT      1   // Import Directory
#define IMAGE_DIRECTORY_ENTRY_IAT         12  // Import Address Table
```

다음 Section Headers에는 실행 코드나 데이터를 담은 Section에 대한 위치 정보가 들어있다. 이 Section Header는 프로그램에 따라 Section 수가 다르기 때문에 그 수가 가변적이다.

```
// Section Header
  BYTE    Name[8];                  // Section 이름
  DWORD   VirtualSize;              // 메모리 상에서 Section 크기
  DWORD   VirtualAddress;           // 메모리 상에서 Section 위치
  DWORD   SizeOfRawData;            // 파일 상에서 Section 크기
  DWORD   PointerToRawData;         // 파일 상에서 Section 위치
```

위와 같은 Section Header 하나가 다음으로 오는 Section 하나 하나에 대응한다. 이 후의 Section 1, 2, .. , n 내부 데이터는 특별한 형식이 없으며 실행 코드 자체나 데이터 그 자체의 형태이다. 간단하게 그림을 그려 보면 [그림 31]과 같다.

[그림 31] Section Headers 설명 그림

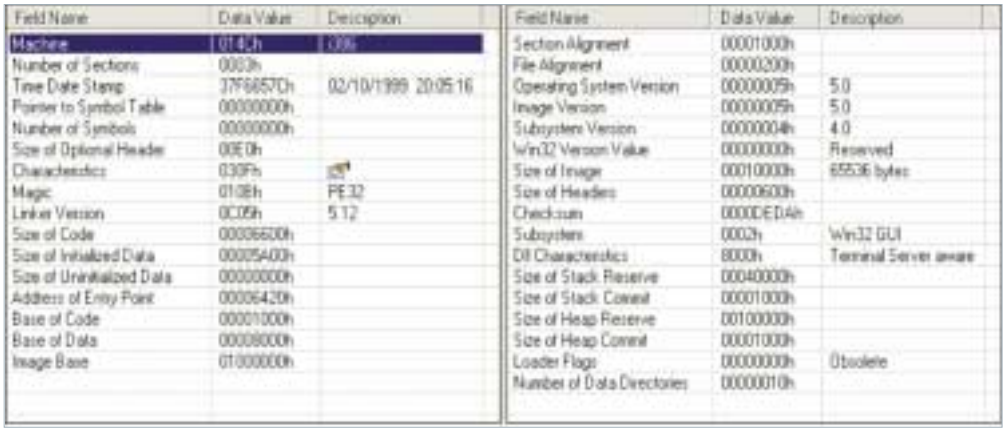
이러한 Section Data 들이 여러 Header 정보를 바탕으로 파일에서 메모리로 복사된 후 몇 가지 중간 과정을 거쳐 실행되는 것이다. 이 정도로 PE 구조에 대한 간단한 설명을 마치고 실행압축 프로그램 원리를 알아보자.

실행압축을 하는 프로그램(Packer)은 아래 그림처럼 왼쪽과 같은 실행파일을 오른쪽과 같은 형태로 만든다. 즉 프로그램의 실제 Code 및 Data를 프로그램 상의 다른 곳에 압축 저장해 두고 실행압축 해제 루틴(Unpacker)이 먼저 실행되어 압축 저장한 부분들을 풀어낸 다음 시작하는 방법으로 동작한다.

[그림 32] 실행압축 되기 전과 압축후의 파일 구조

3. PE 파일 분석으로 보는 실행압축

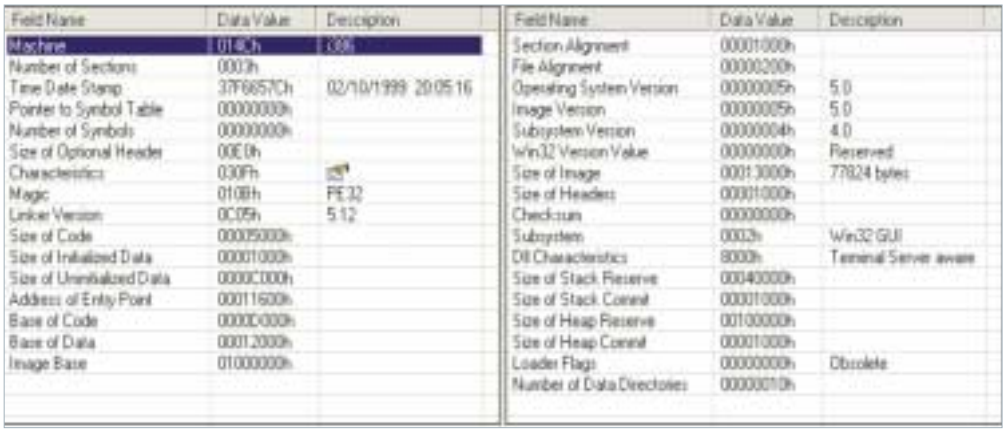
그럼 실제 파일을 실행압축 하여 어떤 변화가 있는지 알아보자. 사용 툴은 PE 포맷을 보는데 유용한 툴인 PE Explorer이다. 실행압축 방법은 UPX를 사용했다.



Field Name	Data Value	Description
Machine	014Ch	i386
Number of Sections	0003h	
Time Date Stamp	37F6857Dh	02/10/1999 20:05:16
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	030Fh	
Magic	010Bh	PE32
Linker Version	0C05h	5.12
Size of Code	00006600h	
Size of Initialized Data	00005400h	
Size of Uninitialized Data	00000000h	
Address of Entry Point	00006420h	
Base of Code	00001000h	
Base of Data	00008000h	
Image Base	01000000h	

Field Name	Data Value	Description
Section Alignment	00001000h	
File Alignment	00000200h	
Operating System Version	00000005h	5.0
Image Version	00000005h	5.0
Subsystem Version	00000004h	4.0
Win32 Version Value	00000000h	Reserved
Size of Image	00010000h	65536 bytes
Size of Headers	00000600h	
Checksum	0000DEDAh	
Subsystem	0002h	Win32 GUI
Dll Characteristics	8000h	Terminal Server aware
Size of Stack Reserve	00040000h	
Size of Stack Commit	00001000h	
Size of Heap Reserve	00100000h	
Size of Heap Commit	00001000h	
Loader Flags	00000000h	Obsolete
Number of Data Directories	00000010h	

[그림 33] 실행압축 되기 전 프로그램 값

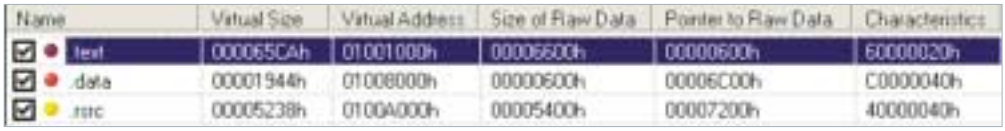


Field Name	Data Value	Description
Machine	014Ch	i386
Number of Sections	0003h	
Time Date Stamp	37F6857Dh	02/10/1999 20:05:16
Pointer to Symbol Table	00000000h	
Number of Symbols	00000000h	
Size of Optional Header	00E0h	
Characteristics	030Fh	
Magic	010Bh	PE32
Linker Version	0C05h	5.12
Size of Code	00005000h	
Size of Initialized Data	00001000h	
Size of Uninitialized Data	0000C000h	
Address of Entry Point	00011600h	
Base of Code	0000C000h	
Base of Data	00012000h	
Image Base	01000000h	

Field Name	Data Value	Description
Section Alignment	00001000h	
File Alignment	00000200h	
Operating System Version	00000005h	5.0
Image Version	00000005h	5.0
Subsystem Version	00000004h	4.0
Win32 Version Value	00000000h	Reserved
Size of Image	00013000h	77824 bytes
Size of Headers	00001000h	
Checksum	00000000h	
Subsystem	0002h	Win32 GUI
Dll Characteristics	8000h	Terminal Server aware
Size of Stack Reserve	00040000h	
Size of Stack Commit	00001000h	
Size of Heap Reserve	00100000h	
Size of Heap Commit	00001000h	
Loader Flags	00000000h	Obsolete
Number of Data Directories	00000010h	

[그림 34] UPX로 실행압축 된 후 프로그램 값

[그림 33]과 [그림 34]에서, Size of Code, Base of Data, Size of Image 등이 압축전과 비교했을 때 바뀐 것을 알 수 있으며, 이에 따라 코드 시작 부분을 가리키는 Entry Point 부분도 바뀌었다. 또한 Data Directories의 Import Table과 Resource Table 내용도 바뀌었지만 그림에서 나타내지는 않았다. 그럼 실제 코드와 데이터가 들어있는 Section 내용을 살펴보자.



Name	Virtual Size	Virtual Address	Size of Raw Data	Pointer to Raw Data	Characteristics
.text	000065CAh	01001000h	00006600h	00000600h	60000020h
.data	00001944h	01008000h	00000600h	00006C00h	C0000040h
.rsrc	00005238h	0100A000h	00005400h	00007200h	40000040h

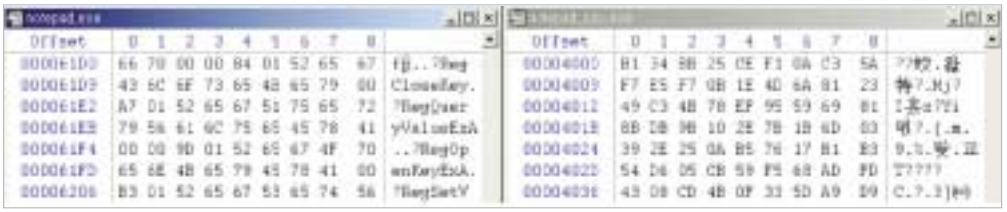
[그림 35] 실행압축 되기 전 Section 정보

Name	Virtual Size	Virtual Address	Size of Raw Data	Pointer to Raw Data	Characteristics
UPX0	0000C000h	01001000h	00000000h	00000400h	E0000080h
UPX1	00005000h	0100D000h	00004800h	00000400h	E0000040h
.rsrc	00001000h	01012000h	00001000h	00004C00h	C0000040h

[그림 36] 실행압축 후의 Section 정보

[그림 35]와 [그림 36]을 비교해 보면, .rsrc란 Section 이름 하나 빼곤 모든 내용이 다 바뀌었다. [그림 35]와 [그림 36]은 notepad.exe를 예로 든 것으로 Section 이름이 .text , .data, .rsrc였지만, UPX0, UPX1, .rsrc로 바뀌었으며, 그 여러 위치 정보에 관련된 부분은 같은 것이 하나도 없다.

실행압축 여부를 확인하기 위해 꼭 이렇게 도구를 써서 확인해야 하는 것은 아니며, WinHex와 같은 바이너리 형태의 파일내용을 텍스트로 볼 수 있는 도구를 이용해도 가능하다. [그림 36]과 같이 압축이 안된 경우 알아볼 수 있는 글자들이 많은 데 반해 압축된 경우에는 알아볼 수 있는 글자들이 얼마 나오지 않는다.



[그림 37] WinHex 로 본 정상 파일과 실행압축 파일 비교

4. 파일 보호 기법

대부분의 실행압축은 파일크기를 작게 하기 위한 것이 큰 이유이다. 앞에서도 언급했지만 악성코드가 압축을 하는 목적은 파일크기를 작게 하여 빠르게 전파되도록 하는 한편 분석을 어렵게 하기 위한 것으로 실제 악성코드에서 사용되고 있는 실행압축 해제를 어렵게 하는 방법에 대해서 알아보자.

통상적으로 분석을 어렵게 하기 위해 파일보호 프로그램을 사용하는데 이를 Protector(편의상 Crypter와 Protector들이 있지만 합쳐 Protector로 한다.)라 한다. 자주 사용되는 것으로는 yoda Crypter, PE Crypt가 있다. 이 툴이 사용하는 기법으로는 암호화, 코드치환, 디버거 무력화, API Redirection 등 몇몇 기술이 있다.

간단하게 위 기법들에 대해 설명해 보면, 암호화는 간단한 XOR 연산을 이용해 해석 불가능한 코드를 만드는 기법이고, 코드치환은 실제 같은 일을 하는 다른 코드로 치환하는 방법으로 패턴을 사용해 어셈블리어 수준이 아닌 좀 더 높은 수준으로 분석해 주는 툴을 무력화시키며, 디버거 무력화는 글자 그래도 디버거로 분석할

경우 프로그램 자체가 비정상 종료되어 버리며, API Redirection은 함수 호출을 몇 단계 거치도록 해 분석을 어렵게 한다. 실행 압축을 한 파일에 위와 같은 기법을 추가로 사용한 파일보호를 실행하면 [그림 38]과 같은 형태를 지닌다.

[그림 38] 실행압축과 파일보호를 했을 때 파일 구조

이 경우 실행압축된 파일에 파일보호 기능까지 되어 있으며, 이를 위해 PEiD라는 프로그램을 이용하였다. [그림 39]처럼 어떤 툴로 보호가 되었는지를 알 수 있으며, [그림 39]는 yoda's crypter 1.2라는 방식으로 파일보호 기능이 되어있음을 나타내고 있다. 그러나, 파일보호 방식이 최신의 방식이라면 확인이 불가능할 것이다.

[그림 39] PEiD를 이용한 파일 검사

5. 실행압축 해제의 기본원리

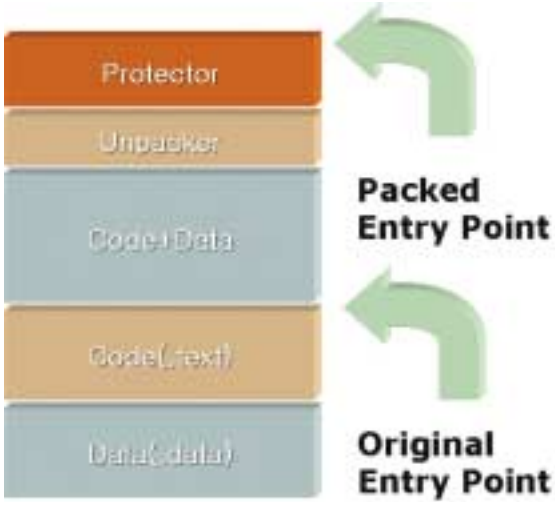
그럼 이제 가장 중요한 실행압축을 해제하기 위한 기본 원리를 알아보자. 무엇보다 중요한 건 "결과적으로 프로그램은 실행돼야 한다는 점이다." 아무리 압축을 하고, 암호화를 하더라도 작동하지 않으면 아무런 쓸모가 없다. 그리고 다음으로 중요한 건 "실행 대상이 될 프로그램 기본구조 자체는 바꿀 수 없다." 즉, 줄 하나를

이리 꼬고 저리 꼬더라도 줄은 풀 수 있으며, 그 줄이 두 개나 세 개의 줄로 되지는 않는다는 점이다.

이 내용을 기술적인 내용으로 바꾸면, 기본원리는 실행압축 프로그램이 실행되는 어느 시점에서는 압축되기 전 상태로 돌아가야 하며, 그 때 메모리 상태를 Dump(메모리 내용을 그대로 파일로 저장)하고, Entry Point를 압축하기 전 위치로 설정해주면 된다는 점이다.

특히 어느 시점을 OEP(Original Entry Point)라고 하며, PEP(Packed Entry Point, 실행압축된 프로그램에서 처음 시작하는 코드위치)와 비교한다. 이 OEP만 찾을 수 있다면 실행압축 해제는 아주 쉽게 이루어 질 수 있다.

[그림 40]은 실행압축 프로그램을 실행해서 압축하기 전 상태로 돌아간 어느 시점의 메모리 상태이다. 이때 실행코드 위치는 OEP 부분[9]이다.



[그림 40] 파일 보호 및 실행압축 시 PEP, OEP 위치

위의 예는 약간 일반적이기 때문에, 실제 UPX 실행압축을 사용한 파일을 예로 들어 설명해 보겠다. 먼저 UPX로 실행 압축된 프로그램의 경우 처음 시작할 때 UPX1에 PEP가 있어 UPX1이 실행[10]된다.

9. 앞의 예도 마찬가지로 이건 그림 상 Entry Point가 해당 Section 처음을 가리키는 것으로 나와 있을 뿐이지 대부분 Section 내의 다른 위치를 가리킨다.

10. UPX1 Section에 Unpacker와 Code+Data가 있다고 의문을 가질 수 있겠지만, 이처럼 한 Section에 Code와 Data 모두 들어갈 수 있다. 앞에서도 나왔지만 Section Data에 대한 제한사항은 없다.

[그림 41] UPX로 실행 압축된 파일과 실행압축이 풀린 후 파일 구조

UPX1의 Unpacker 루틴이 끝나면 EIP가 UPX0의 어느 부분을 가리키게 된다. 즉 이 부분이 OEP이다. 그래서 EIP 값을 검사하다가 UPX1에서 UPX0로 넘어갈 때 프로그램 작동을 멈추면 된다.

yoda Crypter로 보호된 프로그램의 경우, 처음 시작을 yC라는 Section에서 시작한다. yC에서 .text와 .data의 암호화를 해제한 다음 수행 코드부분을 .text로 넘긴다. 그렇지만 yoda Crypter의 경우 디버깅하는 걸 알아내서 전혀 다른 곳으로 분기해 버린다.

[그림 42] Yoda Crypter로 파일보호 된 파일 구조

그 후에 Access Violation에러를 만나 프로그램이 종료 돼 버린다. 그래서 다른 방법으로 Stack을 이용한다. 처음 시작할 때 Stack위치와 보호가 해제 되었을 때 Stack위치가 같다는 점을 이용한다. 즉, Protector루틴을 하나의 함수라고 하면, 함수가 종료하면 다시 실행되기 이전상태로 돌려 놓기 때문에 Stack위치도 이전 상태로 돌아올 때가 Protector가 종료한 시점이며, 이때 보호가 해제된 때라고 예측하는 것이다.

6. 실행압축 해제 실습

그럼 실제 실행압축한 파일을 해제해 보자. 먼저 파일보호를 한 경우의 해제와 이후 실행압축을 푸는 과정을 알아보자. 파일보호와 실행압축을 둘 다 적용한 경우는 아래 설명한 방법대로 하나씩 풀어내면 된다. 사용한 파일은 Windows 2000에 포함된 notepad.exe를 yoda Crypter로 보호한 파일 notepad_yoda.exe와 notepad.exe를 UPX로 실행압축한 notepad_upx.exe이다.

그럼 UPX로 실행 압축한 notepad_upx.exe 파일을 해제해 보자. PEiD로 이 파일을 검사하면 UPX 0.89.6 - 1.02 - 1.24라는 메시지를 출력하며 PE Explorer로 보면 Section 이름들이 UPX0, UPX1으로 되어 있어 UPX로 압축한 것임을 쉽게 알아낼 수 있다. Ollydbg를 사용해 디버깅을 시작하자.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapp
01000000	00001000	notepad_		PE header	Imag	R	RWE	
01001000	0000C000	notepad_	UPX0		Imag	R	RWE	
0100D000	00005000	notepad_	UPX1	code	Imag	R	RWE	
01012000	00001000	notepad_	.rsrc	data, import	Imag	R	RWE	

[그림 43] UPX로 실행 압축된 notepad_upx.exe Section 및 메모리 위치

[그림 44] Trace 시 브레이크 조건 설정

시작하면 파일이 압축되어 있는 거 같다며, 에러를 출력한다. 이때 '아니오'를 선택하고 Alt-M을 눌러 메모리 구조에서 UPX0 위치를 판단하고 Ctrl-T를 눌러 EIP가 UPX0로 가면 멈추도록 설정해 놓는다. 그리고 Trace(Ctrl-F12)한 후 얼마간 시간이 지난 뒤 멈추면 Push EBP 부분이 보이는데 이 위치가 압축이 다 풀린 OEP이다. 이 메모리 내용을 Dump 프로그램으로 파일 형태로 만든 다음 시작 위치를 PEP가 아닌 OEP 위치로 파일의 PE 내용 중 Address of Entry Point 부분으로 고쳐 놓으면 된다.

[그림 45] 실행이 OEP에서 멈춰있는 화면

[그림 46] Dump할 때 EP를 PEP에서 OEP로 변경하는 화면

다음으로 yoda Cryter로 보호한 파일을 해제해 보자. notepad_yoda.exe의 경우 Section 이름 중에 yC가 있으므로 쉽게 알 수 있다. Step Over(F8)를 몇 번 실행한 다음 처음 만나는 Call 루틴에서 멈춰 ESP를 가리키는 Stack 영역에 이 메모리 영역을 읽으면 멈추도록 브레이크를 걸어 놓는다. 그리고 실행시키면 예외가 발생하는데 무시하고, 실행시키면 예외 처리 루틴에서 OEP로 이동시킨다. 그렇지만, 자세히 보면 대부분의 함수는 PUSH EBP, MOV EBP ESP로 시작하는데 MOV EBP ESP에서 멈추었으므로 OEP 주소를 -1 수정해 주어야 한다. 이를 메모리 Dump 프로그램을 사용해서 파일형태로 만들고 찾은 OEP 값으로 PE 파일의 Entry Point 부분을 고쳐 놓으면 된다.

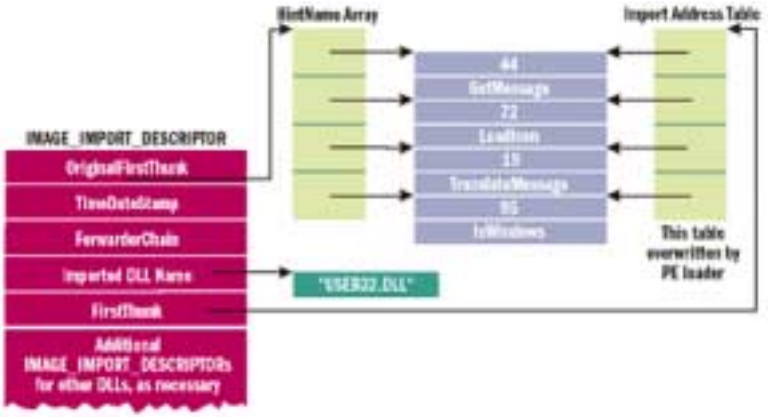
[그림 47] 실행이 OEP에서 멈춘 화면

[그림 48] Dump할 때 EP를 PEP에서 OEP로 변경하는 화면

이렇게 했지만 실제 Dump한 파일은 실행이 되지 않는다. 왜냐하면, 메모리 내용과 실제 실행파일과는 몇 가지 차이점이 존재하기 때문이다. 파일에서 A위치와 B위치라고 할 경우, 메모리 상에는 각 Section마다 다른 위치에 로드될 수 있기 때문에 위치가 달라진다. 그래서 파일 내에 메모리 관련 부분은 메모리 상에서 위치를 가진 RVA(Relative Virtual Address) 값을 사용한다.

또 다른 문제는 IAT(Import Address Table)에 있다. Dump할 때는 그 시스템의 메모리 위치를 포함해 Dump가 된다. 그렇지만 각 시스템 마다 함수들의 메모리 주소가 다르고, Packer나 Protector에서 IAT를 변경해 버리기 때문에 다른 곳에서도 프로그램을 실행할 수 있게 IAT내용을 재구성해 주어야 한다. 메모리 주

소로만 나와 있는 부분을 아래처럼 함수 이름으로 찾아 줄 수 있도록 재구성해주어야 한다. Import Address Table에 대한 데이터 구조와 간단한 그림을 그려보면 다음과 같다.



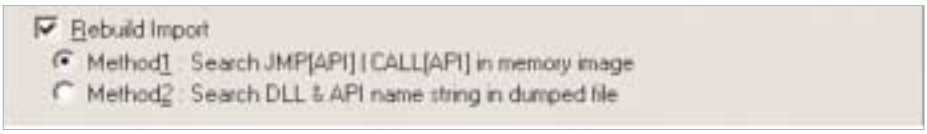
[그림 49] Import Address Table 관련 자료 구조 설명

이러한 IMAGE_IMPORT_DESCRIPTOR가 각 dll마다 하나씩 있고, OriginalFirstThunk와 FirstThunk는 IMAGE_IMPORT_BY_NAME으로 구성된 배열을 가리키고 있다가 실제 메모리로 프로그램이 로드되면, FirstThunk 부분은 실제 각 함수들의 주소를 가진 배열로 변경된다.

```
IMAGE_IMPORT_DESCRIPTOR; // [그림 49]의 빨간색 부분의 데이터 구조
   DWORD   OriginalFirstThunk;
   ..
   DWORD   Name;
   DWORD   FirstThunk;

 IMAGE_IMPORT_BY_NAME    // [그림 49]의 파란색 부분의 데이터 구조
   WORD    Hint;
   BYTE    Name[1];
IMAGE_THUNK_DATA32       // [그림 49]의 연녹색 부분의 데이터 구조
   union {
      PDWORD Function;
      DWORD Ordinal;
      PIMAGE_IMPORT_BY_NAME  AddressOfData;
   } u1;
```

이 경우, 사람 손으로 하기에는 귀찮은 작업이므로, 대부분 툴에서 알아서 함수이름을 재구성해 준다. Ollydbg의 플러그인 OllyDump를 보면 다음과 같은 부분을 볼 수 있다. 이 부분을 설정하면 IAT 부분까지 Dump와 함께 처리해 준다.



[그림 50] OllyDump 화면 중 Import 관련 부분

이렇게 IAT를 수정하면 파일을 실행할 수 있다. 실행 원리는 복잡해 보이지만, 현재는 툴이 잘 나와 있어서 이런 복잡한 과정을 거칠 필요 없이 실행압축 및 파일보호를 해제할 수 있다. 참고로 Ollydbg는 OEP를 찾는 스크립트, IAT Rebuilding Dump 등을 플러그인 상태로 제공해주고 있기 때문에 PE 파일구조를 모르더라도 쉽게 할 수 있다.

7. 맺 음 말

이처럼 실행압축을 해제하는 것은 그다지 어렵지 않으며, 인터넷에 해제 툴들이 있다. 물론 새로 나온 형태의 경우에는 디버거로 일일이 조사해 가면서 실행압축을 해제하겠지만, 그런 경우는 많지 않다.

본 내용에 대한 궁금점이나 추가적인 정보를 얻기 위해서는 아래에 소개하는 참고사이트를 참조토록 하고, 이 글이 앞으로 나올 실행압축 된 악성코드로 인한 피해 예방에 조금이나마 도움이 되기를 바란다.

참고사이트

▶ oPEN rEVERSE fORUMS
국내 Reverse Engineering 하는 사람들이 주로 찾는 곳
https://ampm.ddns.co.kr/~reverse

▶ codeDiver
개인으로 Asprotect, PE , Soft-ICE에 대한 내용을 주로 다룬다.
http://codediver.gg.ro/

- OllyDbg
  사용자 편의성이 뛰어나고, 플러그인 기능으로 기능확장이 충실한 디버거
  http://home.t-online.de/home/Ollydbg/

- OllyDbg User Forum
  OllyDbg 관련 여러 디버깅 FAQ와 플러그인 사이트 링크 포함
  http://ollydbg.win32asmcommunity.net/

- ASPack , AsProtect
  많은 쉐어웨어에서 사용하는 실행압축 프로그램 ASPack을 개발하는 사이트
  http://www.aspack.com/

- Packers/Crypters/Protectors
  수많은 Packers/Crypters/Protectors에 대한 설명이 있는 사이트
  http://www.programmerstools.org/packers.htm

- Unpackers/decrypters/unprotectors
  수많은 Unpackers/decrypters/unprotectors에 대한 설명이 있는 사이트
  http://protools.anticrack.de/unpackers.htm

- Yoda Crypter
  Yoda Crypter를 개발하는 사이트
  http://yodap.cjb.net

- Learn to Crack
  초보자를 위한 설명들이 많은 사이트
  http://www.learn2crack.com