악성코드 분석, 암호화 및 패키징 프로젝트

Protector(프로텍터)에 대한 보고서

문서 번호 : SRP-003

목 차

I. 패킹(Packing)의 소개

II. UPX

III. 프로텍터(Protector)의 소개

IV. Themida

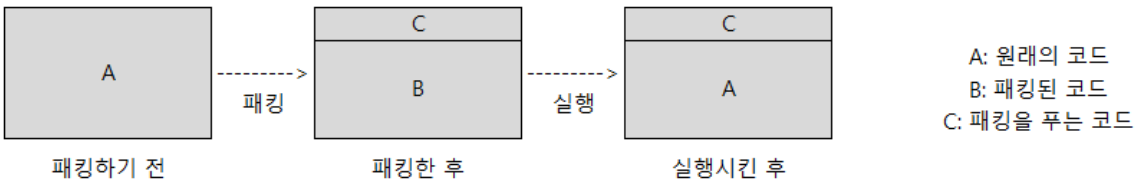
V. 결론

VI. 참고 문헌

Pre-Vision

I. 패킹의 소개

패킹은 프로그램을 압축하는 방법 중 하나로, 압축을 푸는 과정 없이도 바로 실행이 가능한 압축 방식을 의미합니다. 흔히 사용하는 ZIP, 7Z 같은 압축 파일은 압축을 풀어야 실행할 수 있지만, 패킹된 파일은 그럴 필요가 없습니다. 이는 파일 내부에 패킹을 푸는 코드가 포함되어 있기 때문에, 파일을 실행하면 자동으로 압축이 풀리고 프로그램이 실행됩니다.

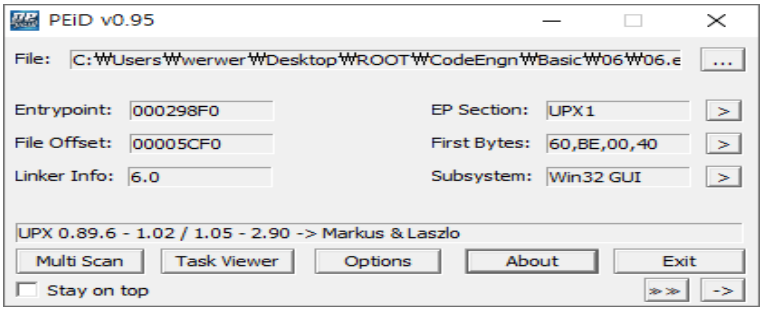


패킹을 사용하는 이유

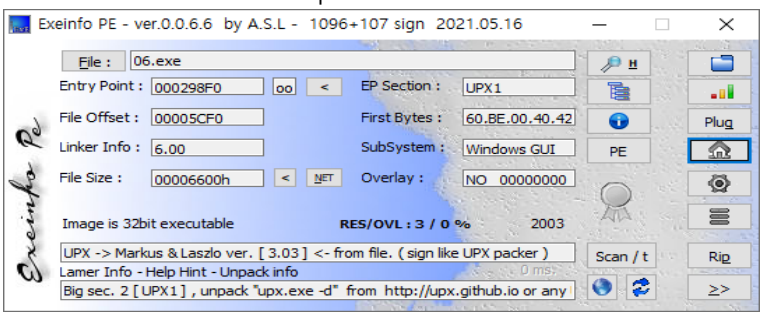
패킹을 하는 이유는 크게 두 가지입니다. 첫 번째로, 프로그램을 분석하기 어렵게 만들 수 있습니다. 패킹을 하면 코드를 분석하기 전에 먼저 압축을 풀어야 하므로 악성코드처럼 분석이 어려워져 백신 개발이 지연될 수 있습니다. 두 번째 이유는 프로그램 크기를 줄이기 위해서입니다. 패킹을 통해 데이터를 압축하여 용량을 줄일 수 있습니다.

패킹 관련 도구

대표적인 패킹 도구로는 UPX, ASPack, Themida 등이 있으며, PEiD나 Exeinfo PE 같은 도구는 프로그램이 패킹되었는지 여부를 확인하는 데 사용됩니다. PEiD는 64비트 파일 분석을 지원하지 않지만, Exeinfo PE는 다양한 정보를 제공하여 더 많은 분석이 가능합니다.



PEiD 실행 화면 https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml



Exeinfo PE 실행 화면 http://exeinfo.booomhost.com/?i=1

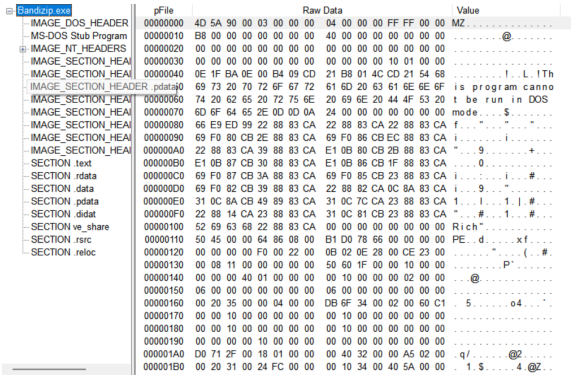
II. UPX

UPX(Universal Packer for Executables)는 실행 파일을 압축하여 크기를 줄이는 무료 도구로, 안전하고 휴대성이 뛰어나며 성능이 우수합니다. 주로 소프트웨어 개발자들이 파일 크기를 최적화하고 배포를 효율적으로 하기 위해 사용됩니다.

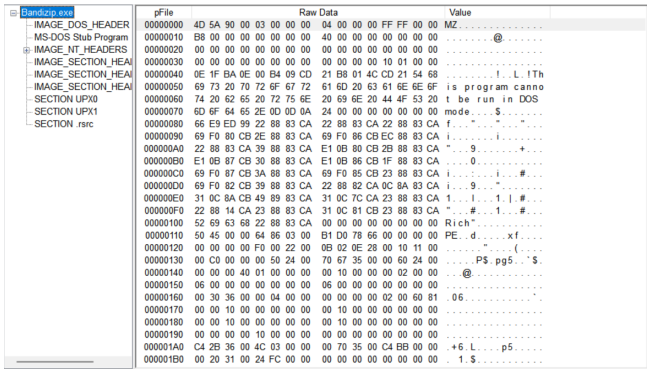
UPX는 프로그램 실행 파일이나 DLL 파일의 크기를 약 50%에서 70%까지 줄여주며, 이로 인해 배포 파일 크기를 줄이고 다운로드 속도를 높이는 데 매우 유용합니다. UPX로 압축된 파일은 기존과 동일한 방식으로 동작하며, 추가적인 메모리나 성능 상의 불이익 없이 압축 해제가 이루어집니다.

또한 Windows, macOS, Linux 등의 다양한 운영체제에서 실행 파일을 압축할 수 있으며, GNU General Public License(GPL) v2+ 하에 배포되는 오픈 소스 프로젝트입니다. 이를 통해 누구나 소스 코드를 열람하고 수정할 수 있으며, 상업적 용도로도 자유롭게 사용할 수 있습니다.

UPX로 Bandizip.exe를 압축하는 모습



기존 반디집 PE



UPX 압축 후 반디집 PE

기능과 특징

1. 뛰어난 압축 성능

UPX는 다른 일반적인 압축 도구보다 더 높은 압축률을 자랑합니다. 실행 파일 크기를 절반 이상 줄여 파일을 빠르게 다운로드하고, 네트워크 대역폭을 절감하는 데 도움이 됩니다. UPX는 ZIP과 같은 포맷보다 더 나은 압축률을 제공하며, 여러 상황에서 효율적인 파일 관리가 가능합니다.

2. 빠른 압축 해제

UPX로 패킹된 파일은 실행 시 자동으로 해제되며, 성능 저하 없이 즉각적인 실행이 가능합니다. 최신 시스템에서 UPX로 압축된 파일은 초당 500MB 이상의 속도로 해제되며, 실행 파일 크기를 줄이는 데도 불구하고 프로그램의 성능에 영향을 미치지 않습니다.

3. 다양한 파일 형식 지원

UPX는 Windows의 PE 파일, Linux의 ELF 파일, macOS의 Mach-O 파일을 포함한 여러 운영체제의 실행 파일 형식을 지원합니다. 이처럼 다양한 형식의 파일을 압축하고 관리할 수 있기 때문에 개발자들은 여러 환경에서 UPX를 사용해 소프트웨어를 최적화할 수 있습니다.

4. 오픈 소스 및 자유로운 배포

UPX는 오픈 소스로 제공되며, 누구나 소스 코드를 열람하고 수정할 수 있습니다. 또한 상업용 응용 프로그램에서도 자유롭게 사용할 수 있어, 배포 및 활용 측면에서 유연성이 높습니다. GPL v2+ 라이선스 하에 배포되어 있어 상업용 소프트웨어에서도 무제한적으로 사용할 수 있습니다.

5. 안전성과 투명성

UPX는 오픈 소스이기 때문에 보안 소프트웨어에서 패킹된 파일의 내부를 분석할 수

있으며, 이를 통해 악성코드 여부를 판별할 수 있습니다. 또한 UPX는 파일 무결성을 유지하는 체크섬 기능을 제공하여, 압축 전후의 파일이 손상되지 않았음을 확인할 수 있습니다.

III. 프로텍터(Protector)의 소개

소프트웨어 프로텍터(Protector)는 프로그램의 무단 복제, 역공학(리버스 엔지니어링), 디버깅 등을 방지하기 위해 다양한 보안 기술을 사용하는 도구입니다. 이는 프로그램의 코드 흐름 분석을 방해하고, 악의적인 공격자가 프로그램을 분석하거나 변조하는 것을 막기 위한 목적으로 사용됩니다. 주로 상용 소프트웨어, 게임, 민감한 데이터를 다루는 프로그램에서 소프트웨어 보호 및 라이선스 관리를 위해 사용되며, 악성코드에서도 악용될 수 있는 기술입니다.

프로텍터는 기본적인 파일 패커 기술을 바탕으로 다양한 분석 방해 기술을 추가하여 프로그램을 보호합니다. 알려진 대표적인 프로텍터로는 ASProtect, Enigma, Obsidium, Themida, VMProtect 등이 있으며, 이들 모두 프로그램 분석을 지연시키고 역공학을 어렵게 만드는 것을 목표로 합니다.

프로텍터의 주요 기능

1. 코드 난독화 (Code Obfuscation)

코드 난독화는 프로그램의 코드를 알아보기 어렵게 변환하여 공격자가 역공학을 시도할 때 이를 이해하기 어렵게 만드는 기술입니다. 변수명, 함수명, 제어 흐름 등이 복잡하게 바뀌며, 이는 코드 분석 툴을 사용하더라도 원래의 의미를 파악하기 어렵게 만듭니다.

2. 안티 디버깅 (Anti-Debugging)

안티 디버깅 기술은 프로그램이 실행되는 동안 디버거를 사용하여 프로그램을 분석하려는 시도를 탐지하고 차단하는 기능입니다. 디버거가 감지되면 프로그램이 비정상적으로 종료되거나 오작동을 유도할 수 있습니다.

3. 무결성 검증 (Integrity Check)

무결성 검증은 프로그램의 데이터를 변조하거나 수정하려는 시도를 탐지하는 기술입니다. 분석 도구를 사용해 프로그램의 코드를 패치하거나 소프트웨어 브레이크포인트를 삽입하는 등의 시도를 방지합니다. 이를 통해 프로그램의 무결성을 보장하고, 변조된 경우 프로그램이 정상적으로 작동하지 않도록 합니다.

.

4. API 난독화 (API Obfuscation)

프로그램이 사용하는 API 호출 정보를 숨기거나 난독화하여 API 기반의 분석을 어렵게 만듭니다. 공격자는 보통 프로그램이 사용하는 외부 라이브러리나 API를 통해 프로그램의 동작을 분석하려 하기 때문에, 이를 난독화하여 분석이 어렵게 만듭니다.

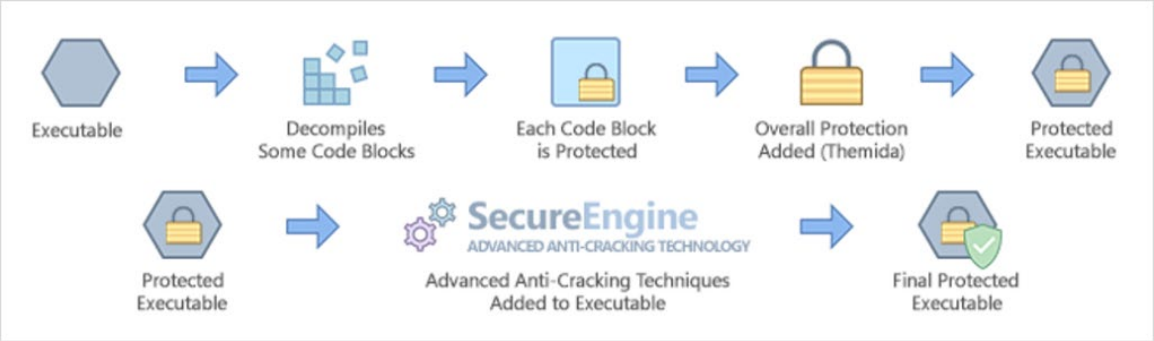
5. 코드 가상화 (Code Virtualization)

코드 가상화는 프로그램의 중요한 코드 부분을 가상의 CPU에서 실행되도록 변환하는 기술입니다. 이는 원래의 코드를 바이너리 코드로 변환한 후, 가상 CPU가 이 코드를 해석하여 실행하도록 만들어 일반적인 역공학 도구로 분석하기 매우 어렵게 만듭니다.

IV. Themida

Themida는 소프트웨어를 크랙과 역공학으로부터 보호하기 위해 개발된 상용 소프트웨어 보호 도구입니다. 주로 게임과 상업용 소프트웨어에서 사용되며, 다양한 난독화 및 역공학 방지 기법을 제공하여 프로그램의 무결성을 유지하고 분석을 어렵게 만듭니다. Themida는 원본 코드의 제어 흐름을 보호하는 동시에 분석을 방해하는 코드를 함께 실행해 역공학을 방지하는 방식을 채택하고 있습니다. 이로 인해 파일 크기가 늘어나고 실행 속도가 다소 느려질 수 있지만, 강력한 보호 성능 덕분에 소프트웨어 보안에 널리 사용되고 있습니다.

Themida 화면



Themida 작동 방식

Themida의 주요 기능

Themida는 소프트웨어 보호를 위해 다양한 보호 옵션을 제공합니다. 이 옵션들은 소프트웨어를 역공학, 크랙, 디버깅, 코드 패치 등 다양한 공격으로부터 보호하며, 각 보호 옵션을 독립적으로 적용하거나 결합해 사용 가능합니다.

1. 난독화 (Obfuscation)

Themida의 핵심 기능 중 하나는 코드 난독화입니다. 난독화는 프로그램의 원본 코드가 분석자에게 쉽게 파악되지 않도록 복잡하게 변경하는 기술입니다. Themida는 원본 코드의 제어 흐름을 유지하면서도, 역공학을 방해하는 다양한 난독화 기술을 적용해 분석을 어렵게 만듭니다. 대표적인 난독화 기법으로는 Entrypoint Obfuscation이 있으며, 이를 통해 역공학 도구가 프로그램의 진입점을 찾기 어렵게 합니다.

Entrypoint Obfuscation: 프로그램이 실행되는 진입점을 난독화하여 역공학 도구가 이를 추적하기 어렵게 만듭니다.

API Wrapping: 프로그램이 외부 API를 호출하는 방식을 숨겨, 역공학 도구가 API 호출을 분석하지 못하도록 만듭니다.

2. 암호화 및 압축 (Encryption & Compression)

Themida는 프로그램의 중요한 코드와 데이터를 암호화하고 압축하는 기능을 제공합니다. 이를 통해 프로그램이 실행되지 않는 동안에는 코드가 보호되며, 실행 시점에만 복호화되어 메모리에서만 실행되기 때문에 역공학 도구로 분석하기 매우 어렵습니다. 또한, 암호화와 압축을 통해 파일 크기를 줄이거나 보호 레벨을 강화할 수 있습니다.

Encryption: 중요한 코드와 데이터를 암호화하여 프로그램이 실행될 때에만 해당 코드가 복호화되도록 합니다.

Compression: 실행 파일 크기를 줄이고 보호 레벨을 향상시키기 위해 코드와

리소스를 압축합니다.

3. 안티 디버깅 및 안티 분석 (Anti-Debugging & Anti-Analysis)

Themida는 소프트웨어가 디버거와 같은 분석 도구로부터 보호될 수 있도록 다양한 안티 디버깅 및 안티 분석 기술을 제공합니다. 이러한 보호 옵션들은 디버깅 시도가 감지되면 프로그램을 종료하거나 알람 메시지를 띄워주며, 해커나 분석자가 소프트웨어를 수정하거나 분석하는 것을 방지합니다.

Anti-Debugging: 디버거가 감지되면 프로그램이 종료되거나 오류가 발생하도록 합니다.

Anti-Analysis: 모니터링 도구나 분석 도구가 감지될 때 프로그램을 종료하거나 방해합니다.

VM 탐지: Themida는 프로그램이 가상 머신 환경(VMware, VirtualPC)에서 실행 중인지 탐지하여 가상 머신에서의 분석 시도를 방지합니다.

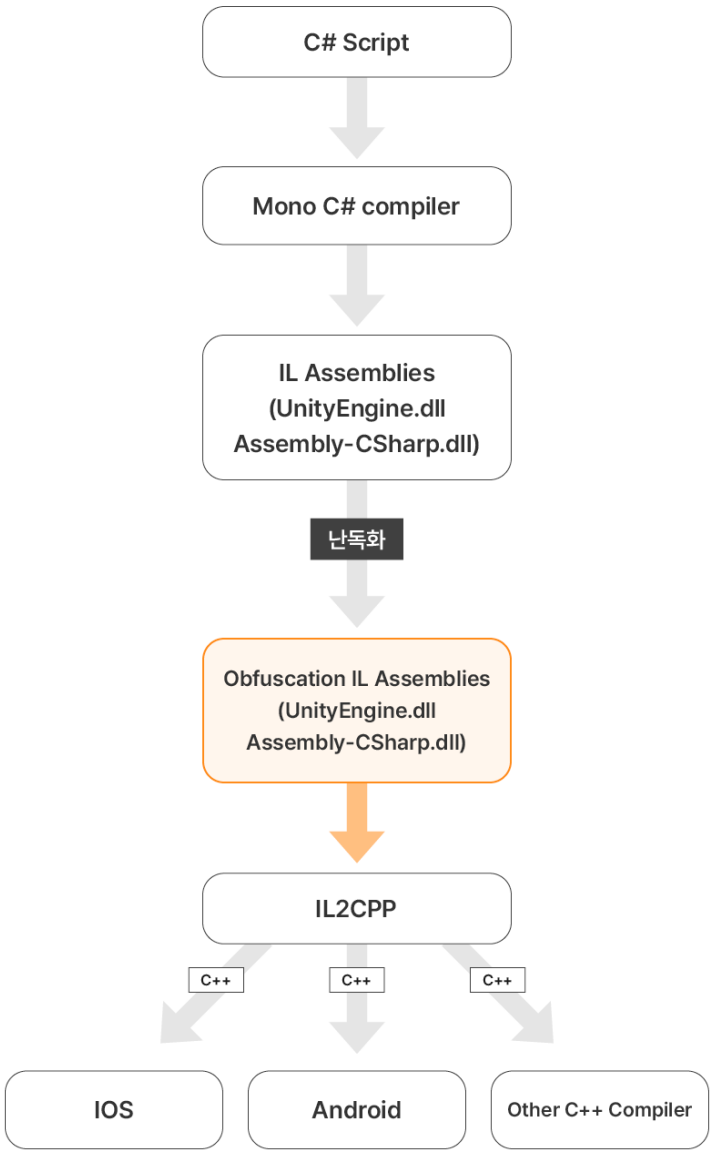
4. IAT 제거 및 API 보호 (IAT Removal & API Wrapping)

Themida는 소프트웨어 보호를 위해 Import Address Table(IAT)을 제거하는 기능을 제공합니다. IAT는 프로그램이 실행 중에 사용하는 API 목록을 관리하는데, 이를 제거함으로써 공격자가 API 호출을 추적하는 것을 어렵게 만듭니다. 또한, API 호출을 숨기는 API Wrapping 기능도 제공하여 역공학 도구가 API 기반으로 분석하지 못하게 합니다.

IAT Removal: Import Address Table을 제거해 프로그램이 호출하는 API 정보를 숨깁니다.

API Wrapping: 외부 API 호출을 감춰 API 분석을 방해합니다.

Pre-Vision



난독화 과정

출처 : https://story.cookapps.com/articles/189

1. 컴파일러에서 Assembly-CSharp.dll, UnityEngine.dll 등이 생성
2. 생성된 .dll을 Mono Cecil을 이용하여 수정
3. 수정된 .dll을 il2cpp를 통해 C++로 변환되어 빌드

// 난독화 전

```
1. [Token(Token = "0x2000002")]
2. public class GoldManager
3. {
4.   [Token(Token = "0x4000001")]
5.   [FieldOffset(Offset = "0x8")]
6.   private int _gold;
7.   [Token(Token = "0x6000001")]
8.   [Address(Offset = "0x1E2C14", RVA = "0x1E2C14", VA = "0x1E2C14")]
9.   public bool UseGold(int gold) => new bool();
10.  [Token(Token = "0x6000002")]
11.  [Address(Offset = "0x1E2C34", RVA = "0x1E2C34", VA = "0x1E2C34")]
12.  public GoldManager() {}
13. }
```

```
14.
15. // 난독화 후
16. [Token(Token = "0x2000002")]
17. public class CCOOFEENFCA
18. {
19.   [Token(Token = "0x4000001")]
20.   [FieldOffset(Offset = "0x8")]
21.   private int JODFLMBMOHC;
22.   [Token(Token = "0x6000001")]
23.   [Address(Offset = "0x1E2C64", RVA = "0x1E2C64", VA = "0x1E2C64")]
24.   public bool JMPNPBAGNFF(int BEMMJODKCFC) => new bool();
25.   [Token(Token = "0x6000002")]
26.   [Address(Offset = "0x1E2C84", RVA = "0x1E2C84", VA = "0x1E2C84")]
27.   public CCOOFEENFCA() {}
28. }
29.
```

난독화 전(위)과 후(아래) decompile 비교

출처 : https://story.cookapps.com/articles/189

V. 결론

프로텍터는 소프트웨어 보호에서 매우 중요한 역할을 담당합니다. 특히, Themida와 같은 고급 보호 도구는 코드 난독화, 안티 디버깅, 암호화 등의 다양한 기술을 활용하여 소프트웨어를 역공학 및 크랙 시도로부터 보호합니다. Themida는 상업용 소프트웨어와 게임에서 자주 사용되며, 이러한 난독화 및 보호 기법을 통해 해커가 프로그램의 내부 구조를 파악하거나 악의적으로 변경하는 것을 방지합니다.

문서에서 설명한 바와 같이, Themida는 보호의 수준이 매우 높지만, 그만큼 파일 크기가 증가하거나 실행 속도가 저하될 수 있다는 단점도 존재합니다. 또한, 여러 보호 옵션을 조합해 사용하는 과정이 복잡할 수 있습니다. 하지만, 소프트웨어의 무결성을 보장하고 역공학을 방지하는 데 있어 중요한 역할을 한다는 점에서 Themida는 소프트웨어 보호를 위한 매우 유용한 도구라 할 수 있습니다.

VI. 참고 문헌

"프로텍터 유형의 분석방해 기술 우회방안에 관한 연구" (2017.06),
https://dcollection.korea.ac.kr/srch/srchDetail/000000076259
"실행 파일 형태로 복원하기 위한 Themida 자동 역난독화 도구 구현" (2017.08),
https://scienceon.kisti.re.kr/srch/selectPORSrchArticle.do?cn=JAKO201711656578275
"가상화 난독화 기법이 적용된 실행 파일 분석 및 자동화 분석 도구 구현", (2013.08) from
https://koreascience.or.kr/article/JAKO201326940560978.page
"Themida의 API 난독화 분석과 복구방안 연구", (2017.02) from
https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002200969