# 악성코드 탐지, 암호화 및 패키징 프로젝트

Protector(프로텍터)에 대한 보고서



**목**

**차**

1. **Protector의 소개**
2. **Protctor에서 사용되는 기술**
   1. **패킹(Packing)**
   2. **암호화(Encryption)**
   3. **난독화(Obfuscation)**
   4. **안티 디버깅(Anti-Debugging)**
3. **Protector가 리버스 엔지니어링에 미치는 영향**
4. **법적 및 윤리적 측면 Protector로 인해 미치는 영향**
5. **결론**
6. **참고 문헌**

## Ⅰ. Protector의 소개

### 정의

보호기(Protector)는 소프트웨어의 불법 복제, 변조, 역공학을 방지하기 위해 설계된 기술 또는 방법이다. 소프트웨어 보안에서 보호기는 매우 중요한 역할을 하며, 소프트웨어 개발자와 기업이 자산을 안전하게 보호할 수 있도록 돕는다. 보호기는 소프트웨어가 제대로 작동하면서도 외부의 위협으로부터 안전하게 유지되도록 설계된다.

### 역사

소프트웨어 보호 기술은 초기의 \*\*CRC(Cyclic Redundancy Check)\*\*와 같은 간단한 오류 검출 기법에서 시작되었다. 1980년대에는 초기의 라이센스 관리 시스템이 등장하였고, 1990년대에는 패킹 기술이 부각되었다. 2000년대에는 암호화와 난독화 기술이 널리 사용되기 시작하며, 최근에는 안티 디버깅과 안티 탬퍼링 기술이 추가되어 더욱 강력한 보호가 가능해졌다. 이처럼 소프트웨어 보호 분야는 지속적으로 발전해 왔다.

### 목적

보호기의 주요 목적은 소프트웨어의 역공학을 방지하고, 불법 복제 및 변조를 차단하는 것이다. 이를 통해 개발자는 자신의 지적 재산을 보호하고, 수익을 극대화할 수 있다.

### 사용 사례

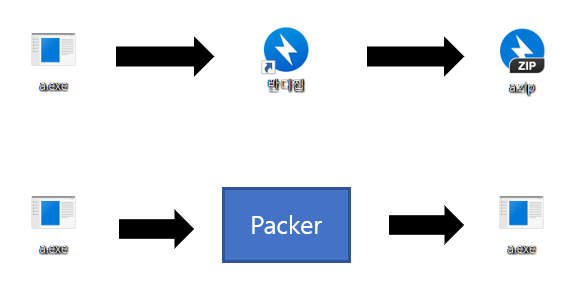
보호기는 다양한 산업에서 널리 사용된다. 예를 들어, 비디오 게임 산업에서는 DRM(Digital Rights Management) 기술이 사용되며, 기업용 소프트웨어에서는 라이센스 관리와 같은 보호 기술이 적용된다. 또한, 지적 재산 보호를 위한 기술도 필수적이다.

## Ⅱ. Protector에서 사용되는 기술

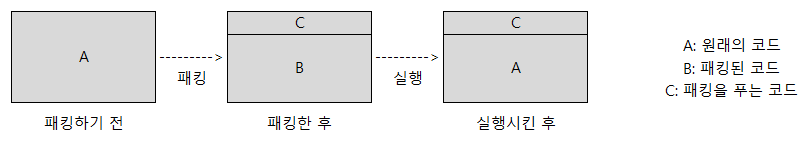
## 패킹(Packing)

패킹(Packing)은 실행압축이라는 뜻을 가지고 있다.

실행압축이란 말 그대로 압축을 푸는 과정 없이 바로 실행시킬 수 있는 압축방법이다.실행압축은 일반적인 압축인 ZIP, 7Z와는 다르다. 일반적인 압축은 파일의 확장자가 바뀌고 압축을 풀어야만 안에 있는 것을 실행시킬 수 있다.



그 이유는 패킹된 파일은 파일 내부에 패킹 해제 코드를 가지고 있기 때문이다. 패킹된 파일을 실행하는 순간 패킹 해제 코드를 실행시켜 프로그램 스스로 패킹을 해제하고 프로그램이 실행된다.



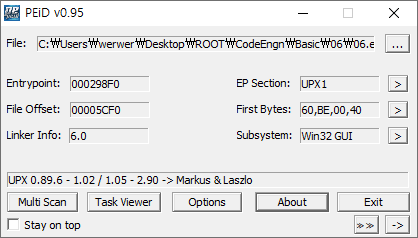
**패킹을 하는 이유**

첫 번째로 프로그램 분석을 어렵게 할 수 있다. 패킹을 하면 코드를 분석하기 전에 먼저 패킹을 풀어야 하므로 코드를 분석하기가 어려워진다. 예로 악성코드 같은 경우에는 패킹을 해서 분석하기 어렵게 해서 백신 개발이 늦어지게 된다.또한 코드에 중요한 정보가 있는 경우에도 사용할 수 있다. 두 번째로 데이터를 압축해서 프로그램의 크기를 줄일 수 있다.

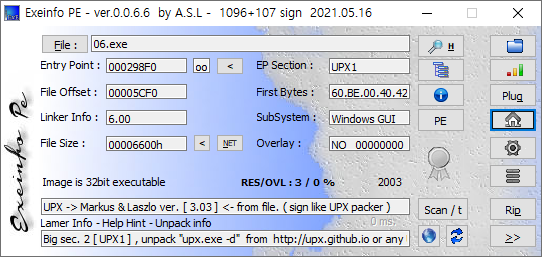
**패킹 관련 도구**

예시: UPX, ASPack, Themida. PEiD, Exeinfo PE

이 도구들은 PE 파일에 대한 다양한 정보를 제공하는 정적 분석 도구로,프로그램이 작성된 언어를 알 수 있고 패킹, 프로텍팅 여부와 컴파일러 등을 확인할 수 있다. 둘의 차이점은 PEiD는 64비트 파일 분석을 지원하지 않고 개발과 지원이 중단되었으며, Exeinfo PE를 통해서 얻을 수 있는 정보가 더 많다.



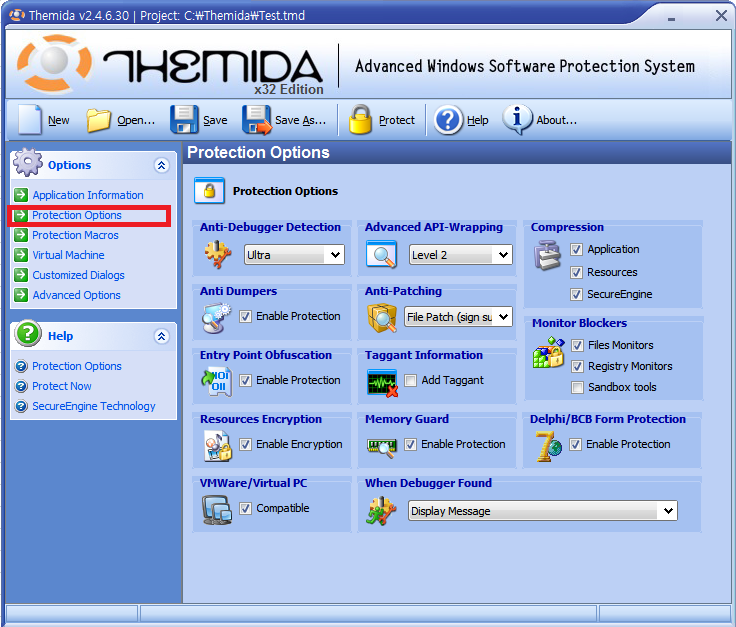
PEiD 실행 화면 https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml



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

**Themida**

Themida 툴은 유명한 패킹툴이자 안티 디버깅툴로 지적재산권을 보호하기 위해 상용 애플리케이션을 패킹하거나 안티 디버깅하는데 주로 쓰인다. Themida 툴로 보호된 프로그램은 역공학을 아예 못하게 한다 기 보다는 분석을 어렵게 하고자 목적을 둔 툴이다.



**Themida 기능**

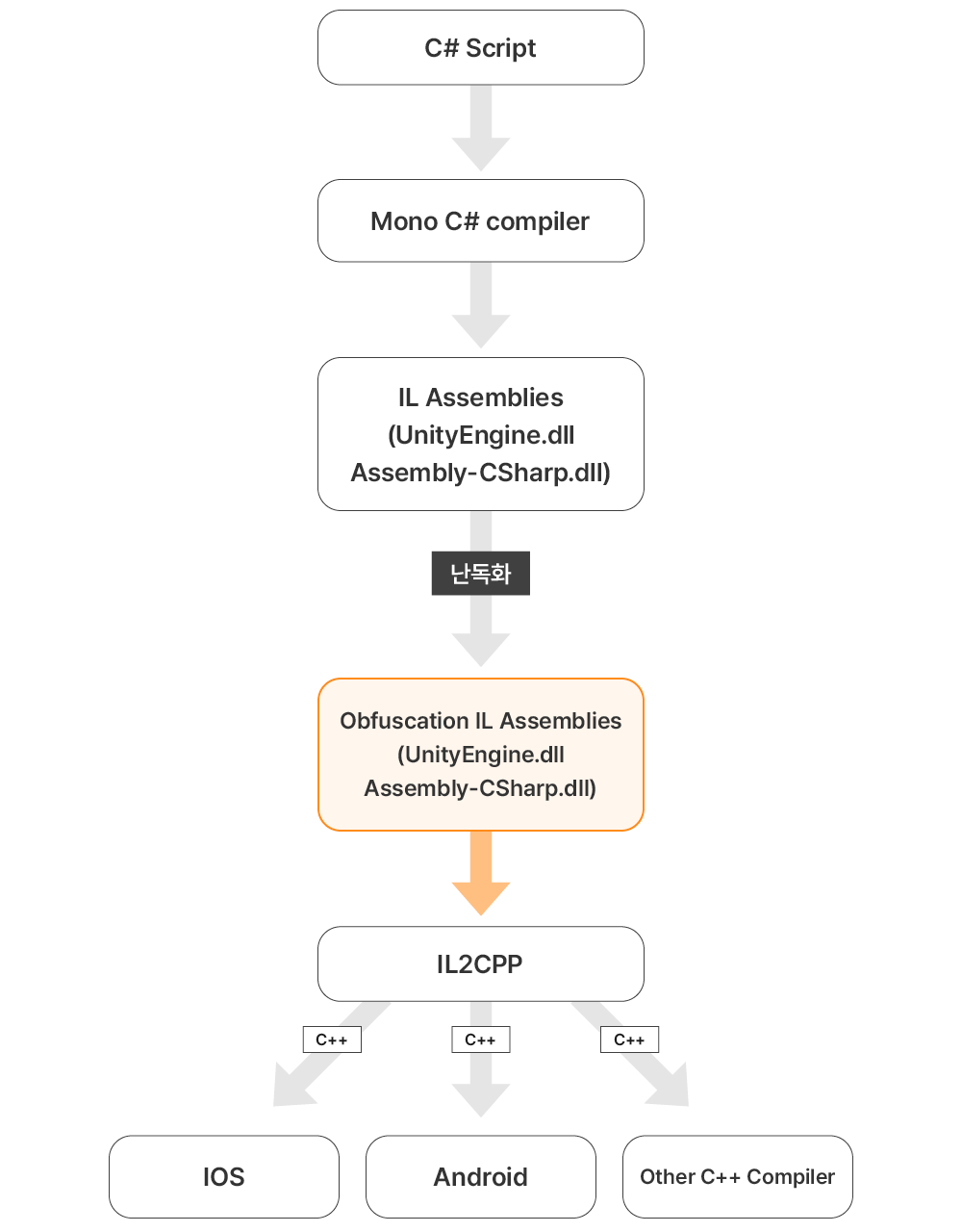
1. Anti-Debugger Detection
2. Anti Dumper
3. EntryPoint Obfuscation
4. Resource Encryption
5. Anti-Patching

## 암호화(Encryption)

암호화는 코드 섹션(예: .text)을 암호화하여 실행 시점에 동적으로 복호화되는 방식이다. 이를 통해 코드 분석을 방지한다.

## 난독화(Obfuscation)

난독화는 변수명과 함수명을 의미 없는 이름으로 변경하고, 코드의 복잡성을 높여 분석을 어렵게 만드는 기술이다. 이를 통해 공격자가 코드를 이해하기 힘들게 한다.



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

[Token(Token = "0x2000002")]

public class GoldManager

{

[Token(Token = "0x4000001")]

[FieldOffset(Offset = "0x8")]

private int \_gold;

[Token(Token = "0x6000001")]

[Address(Offset = "0x1E2C14", RVA = "0x1E2C14", VA = "0x1E2C14")]

public bool UseGold(int gold) => new bool();

[Token(Token = "0x6000002")]

[Address(Offset = "0x1E2C34", RVA = "0x1E2C34", VA = "0x1E2C34")]

public GoldManager() {}

}

// 난독화 후

[Token(Token = "0x2000002")]

public class CCOOFEENFCA

{

[Token(Token = "0x4000001")]

[FieldOffset(Offset = "0x8")]

private int JODFLMBMOHC;

[Token(Token = "0x6000001")]

[Address(Offset = "0x1E2C64", RVA = "0x1E2C64", VA = "0x1E2C64")]

public bool JMPNPBAGNFF(int BEMMJODKCFC) => new bool();

[Token(Token = "0x6000002")]

[Address(Offset = "0x1E2C84", RVA = "0x1E2C84", VA = "0x1E2C84")]

public CCOOFEENFCA() {}

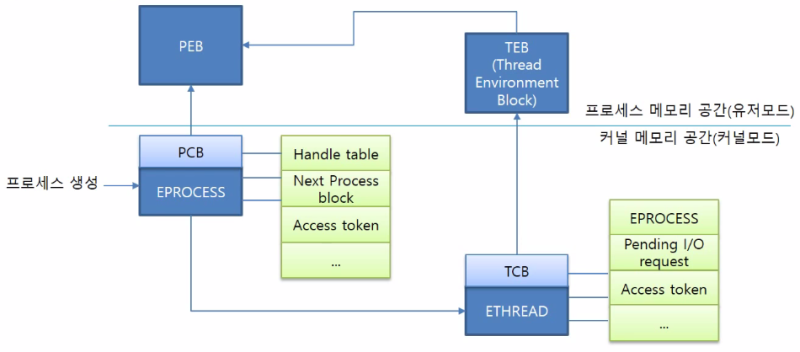
}

###### 난독화 전(위)과 후(아래) decompile 비교

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

## 안티 디버깅(Anti-Debugging)

디버거 감지 및 차단 기법으로, OllyDbg, x64dbg와 같은 도구를 감지하여 프로그램의 실행을 중단시키거나 변조를 방지한다.



## Ⅲ. Protector가 리버스 엔지니어링에 미치는 영향

## 코드 난독화

## Protector는 프로그램의 코드를 난독화하여, 역분석을 수행하는 사람이 코드를 쉽게 이해하지 못하게 합니다. 코드의 흐름을 복잡하게 하거나 변수명, 함수명 등을 모호하게 바꾸어 해독을 어렵게 만듭니다.

## 디버깅 방해

## Protector는 디버거를 감지하고 이를 방해하는 기능을 포함할 수 있습니다. 리버스 엔지니어가 프로그램을 분석할 때 주로 사용하는 디버깅 도구를 무력화하거나, 실행을 중단시키는 방식으로 분석을 방해합니다.

## 안티 디스어셈블리

## Protector는 코드가 정적 분석(디스어셈블리)되는 것을 어렵게 만듭니다. 이를 통해 리버스 엔지니어가 프로그램의 어셈블리 코드를 추출하거나 분석하기 어렵도록 설계됩니다.

## 암호화 및 압축

## Protector는 프로그램의 중요한 코드나 데이터를 암호화하거나 압축하여 분석을 방해합니다. 실행 중에만 해당 코드가 복호화되기 때문에 리버스 엔지니어가 코드를 정적 분석하기 힘듭니다.

## 엔트로피 증가

## Protector는 프로그램의 구조를 복잡하게 만들어, 엔트로피(무작위성)를 높입니다. 이는 자동 분석 도구나 패턴 기반 분석이 잘 작동하지 않게 만듭니다.

## 자체 수정 코드 (Self-Modifying Code)

## Protector는 프로그램이 실행 중일 때 코드 자체를 동적으로 수정하는 기능을 추가할 수 있습니다. 이는 역분석자가 코드의 원래 모습을 정확히 파악하기 어렵게 만듭니다.

## Ⅳ. 법적 및 윤리적 측면 Protector로 인해 미치는 영향

## 법적 측면

## 저작권 보호

## 긍정적 영향: Protector는 소프트웨어의 소스 코드나 구조를 보호하여 불법 복제 및 저작권 침해를 막는 데 기여할 수 있습니다. 이로 인해 소프트웨어 개발자나 회사는 자산을 보호하고 정당한 수익을 확보할 수 있습니다.

## 부정적 영향: 일부 Protector는 너무 강력하게 보호된 나머지, 사용자가 합법적으로 구매한 소프트웨어에 대한 정당한 사용권을 제한할 수 있습니다. 예를 들어, 소프트웨어를 백업하거나 분석하는 것이 불가능할 수 있으며, 이에 대해 일부 법적 분쟁이 발생할 수 있습니다.

## 리버스 엔지니어링과 법적 허용성

## 여러 국가에서 리버스 엔지니어링은 특정 조건하에서 합법적일 수 있습니다. 예를 들어, 소프트웨어 상호 운용성을 확보하기 위한 목적이거나 보안 결함을 찾기 위한 리버스 엔지니어링은 허용됩니다. 그러나 Protector는 이를 방해할 수 있으며, 사용자가 법적으로 허용된 목적을 달성하는 데 어려움을 줄 수 있습니다.

## 일부 법적 시스템에서는 너무 강력한 보호 기법을 사용하는 것이 소비자의 권리를 침해할 수 있다고 보고, 이에 대한 규제를 제정할 수 있습니다.

## 디지털 밀레니엄 저작권법(DMCA)및 유사 법률

## 미국의 DMCA나 유럽의 유사한 법률은 소프트웨어의 무단 복제 방지를 목적으로 하는 기술적 보호 조치를 우회하는 행위를 불법으로 규정합니다. Protector는 이러한 법률 하에서 강력한 법적 보호를 받을 수 있지만, 연구 목적이나 보안 감사 등의 합법적인 리버스 엔지니어링 시도까지 차단할 수 있어 논란이 될 수 있습니다.

## 윤리적 측면

## 사용자 권리 침해

## Protector는 소프트웨어의 보안을 강화하지만, 이는 정당한 사용자의 권리를 지나치게 제한할 수 있습니다. 예를 들어, 사용자는 자신이 구매한 소프트웨어를 합법적으로 백업하거나 수정할 권리가 있을 수 있는데, Protector는 이러한 행위를 어렵게 만듭니다.

## 사용자가 소프트웨어의 문제를 고치거나 맞춤 설정을 하려는 합법적인 시도를 억제할 수 있는 윤리적 논란이 있습니다.

## 보안 연구 제한

## Protector는 소프트웨어의 보안 취약점을 분석하려는 윤리적 해커나 보안 연구자의 활동을 제한할 수 있습니다. 보안 연구자들은 소프트웨어의 결함을 발견하고 이를 고쳐주는 역할을 하지만, Protector는 이러한 분석을 방해하여 보안 문제를 감추는 결과를 초래할 수 있습니다.

## 보호된 소프트웨어가 심각한 보안 취약점을 가지고 있는 경우에도, Protector는 해당 문제를 조사하고 해결하는 데 걸림돌이 될 수 있습니다.

## 소프트웨어의 장기 보존

## 소프트웨어의 장기적인 보존을 위한 아카이빙 작업에도 Protector는 윤리적 논란을 일으킬 수 있습니다. 소프트웨어가 시간이 지나면 더 이상 지원되지 않거나 구형이 될 수 있는데, 이때 보호 기술이 너무 강력하면 후에 소프트웨어를 분석하고 보존하려는 시도를 어렵게 만들 수 있습니다.

## 투명성 부족

## Protector는 사용자가 소프트웨어가 실제로 어떤 행동을 하는지 정확히 알기 어렵게 만듭니다. 특히, 악성 소프트웨어나 스파이웨어가 Protector를 사용하여 자신의 활동을 감추는 경우, 이는 윤리적으로 문제가 될 수 있습니다. 사용자는 자신이 설치한 소프트웨어가 어떻게 동작하는지 알 권리가 있지만, Protector는 이 권리를 침해할 수 있습니다.

## Ⅴ. 결론

## Protector는 소프트웨어 보안 및 저작권 보호에 필수적인 도구이지만, 그 사용에는 법적, 윤리적 고려가 필요합니다. Protector는 리버스 엔지니어링을 방해하고, 소프트웨어의 무단 복제 및 변조를 막아 소프트웨어 개발자와 기업의 자산을 보호합니다. 그러나 동시에 사용자의 합법적인 권리와 보안 연구 활동을 방해할 수 있는 문제도 존재합니다. Protector 기술을 적용할 때에는 법적, 윤리적 균형을 고려하여, 소프트웨어의 보안을 강화하면서도 정당한 사용자의 권리를 보호할 수 있는 방식으로 사용해야 합니다.

.

## 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