# 악성코드 탐지, 암호화 및 패키징 프로젝트

Protector(프로텍터)에 대한 보고서



**목**

**차**

1. **Protector의 소개**
2. **Protctor에서 사용되는 기술**

**2.1. 패킹(Packing)**

**2.2. 암호화(Encryption)**

**2.3. 난독화(Obfuscation)**

**2.4. 안티 디버깅(Anti-Debugging)**

1. **Protector가 리버스 엔지니어링에 미치는 영향**
2. **법적 및 윤리적 측면 Protector로 인해 미치는 영향**
3. **결론**
4. **참고 문헌**

## Ⅰ. Protector의 소개

정의

Protector 는 소프트웨어를 불법 복제, 변조, 역공학으로부터 지키기 위해 만들어진 기술이나 방법을 말합니다. 소프트웨어 보안에서 핵심적인 역할을 하는 보호기는 개발자와 기업들이 자신들의 자산을 안전하게 지킬 수 있도록 도와줍니다. 이는 소프트웨어가 정상적으로 작동하면서도 외부 위협으로부터 안전을 유지할 수 있게 설계됩니다.

역사

소프트웨어 보호 기술은 \*\*CRC(Cyclic Redundancy Check)\*\* 같은 단순한 오류 검출 방식에서 출발했습니다. 80년대에 초기 라이센스 관리 시스템이 등장했고, 90년대에는 패킹 기술이 주목받았죠. 2000년대 들어 암호화와 난독화 기술이 널리 쓰이기 시작했고, 최근에는 안티 디버깅과 안티 탬퍼링 기술까지 더해져 한층 강력한 보호가 가능해졌습니다. 이렇게 소프트웨어 보호 분야는 꾸준히 발전해 왔습니다.

목적

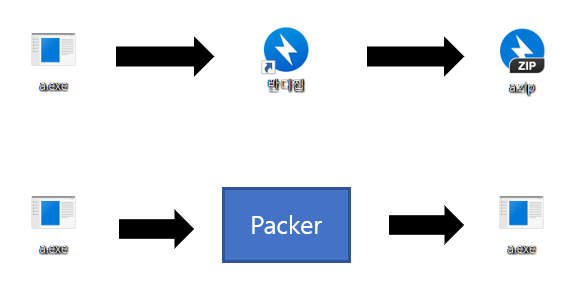
Protector의 주된 목적은 소프트웨어의 역공학을 막고, 불법 복제와 변조를 차단하는 것입니다. 이를 통해 개발자들은 자신의 지적 재산을 지키고 수익을 최대화할 수 있게 됩니다.

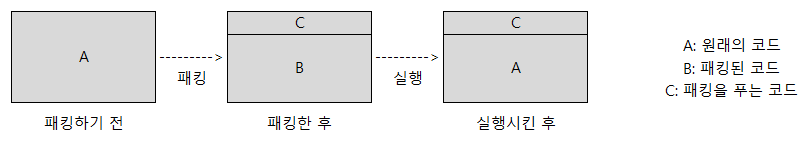
사용 사례

Protector는 여러 산업 분야에서 폭넓게 활용됩니다. 예를 들어, 게임 업계에서는 DRM(Digital Rights Management) 기술을 사용하고, 기업용 소프트웨어에서는 라이센스 관리 같은 보호 기술을 적용합니다. 또한, 지적 재산을 보호하기 위한 기술도 필수적으로 사용되고 있죠.

## Ⅱ. Protector에서 사용되는 기술

## 패킹(Packing)

**패킹**은 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.

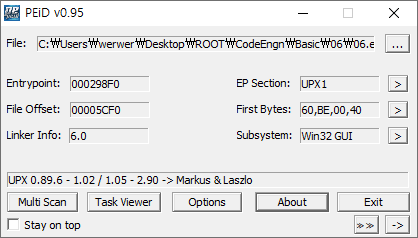


**패킹을 사용하는 이유**

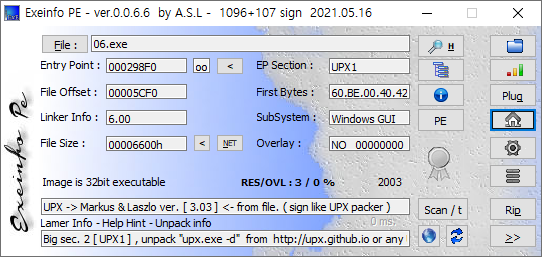
패킹을 하는 이유는 크게 두 가지입니다. 첫 번째로, 프로그램을 **분석하기 어렵게** 만들 수 있습니다. 패킹을 하면 코드를 분석하기 전에 먼저 압축을 풀어야 하므로 악성코드처럼 분석이 어려워져 백신 개발이 지연될 수 있습니다. 두 번째 이유는 **프로그램 크기를 줄이기** 위해서입니다. 패킹을 통해 데이터를 압축하여 용량을 줄일 수 있습니다.

**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.



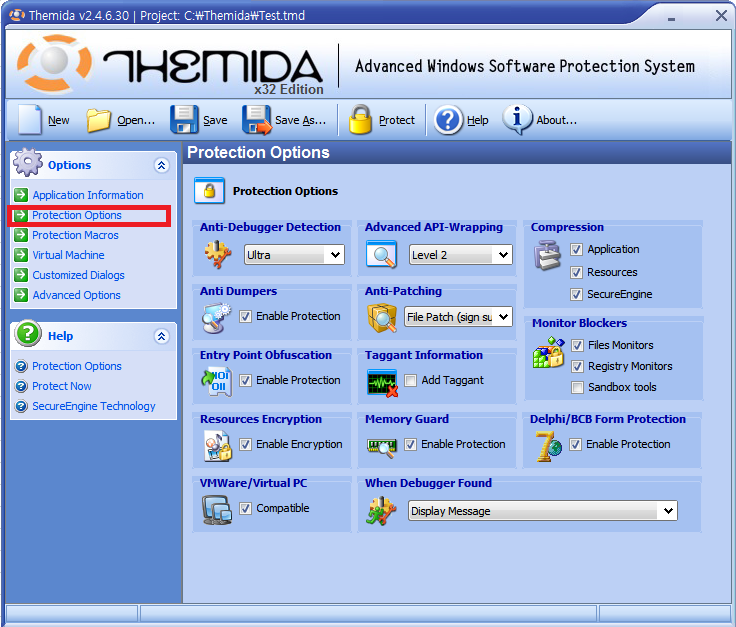
PEiD 실행 화면 https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml



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

**Themida**

**Themida**는 잘 알려진 패킹 및 안티 디버깅 도구로, 주로 상용 소프트웨어를 보호하기 위해 사용됩니다. Themida는 프로그램을 완전히 분석하지 못하게 하는 것이 아니라, 분석을 더욱 어렵게 만들어 지적 재산을 보호하는 데 목적이 있습니다.



**Themida 주요기능**

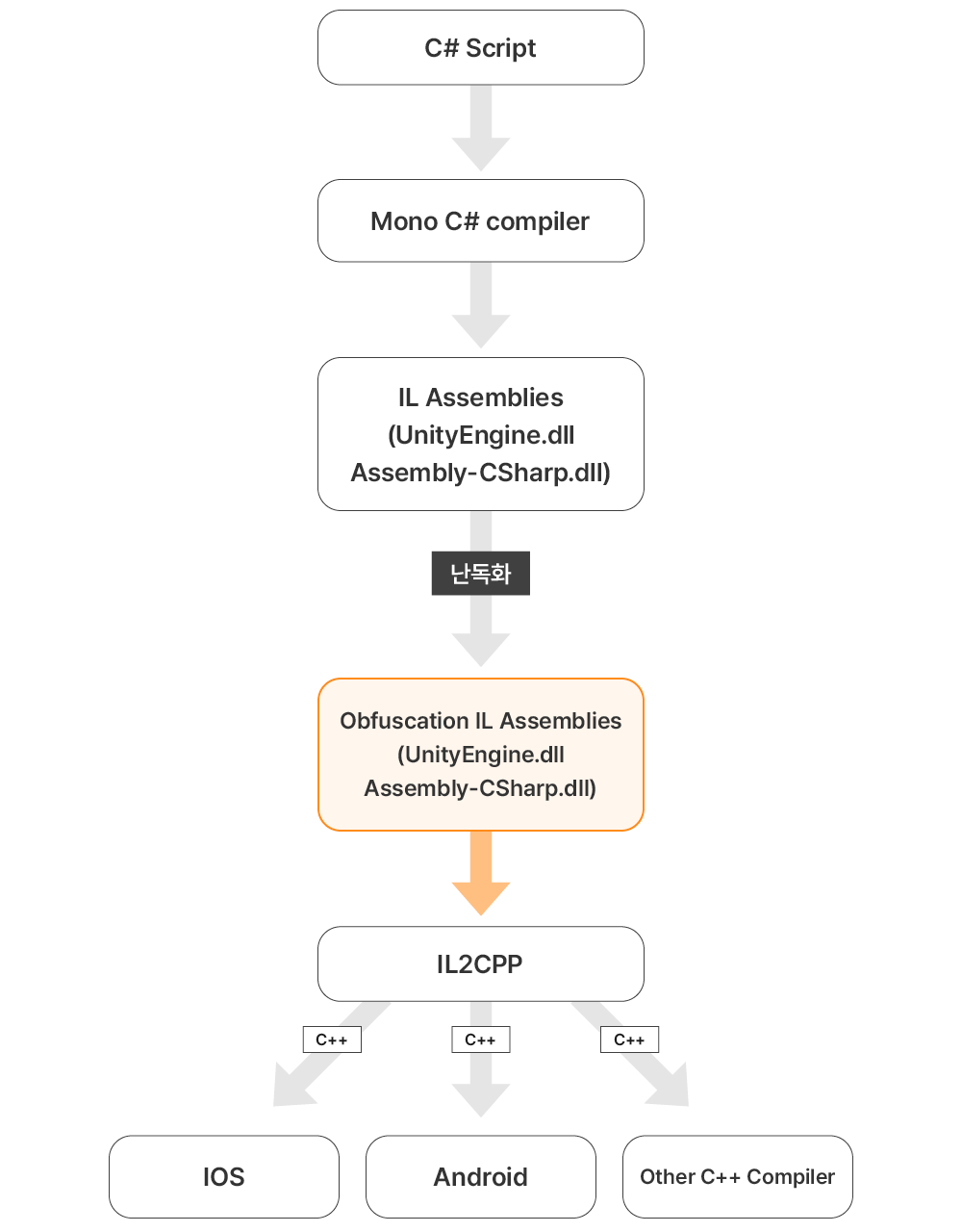
1. Anti-Debugger Detection
2. Anti Dumper
3. EntryPoint Obfuscation
4. Resource Encryption
5. Anti-Patching

## 암호화(Encryption)

암호화는 프로그램의 중요한 코드 섹션을 암호화하여, 실행 시점에만 복호화되는 방식입니다. 이로 인해 코드 분석이 어렵게 되어 악성코드나 중요 데이터를 보호하는 데 유용합니다.

## 난독화(Obfuscation)

난독화는 변수명과 함수명을 의미 없는 이름으로 변경하고, 코드를 복잡하게 만들어 **코드 분석을 어렵게** 하는 기술입니다. 이렇게 하면 공격자가 코드를 이해하기 힘들어지며, 프로그램 보호에 효과적입니다.



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() {}

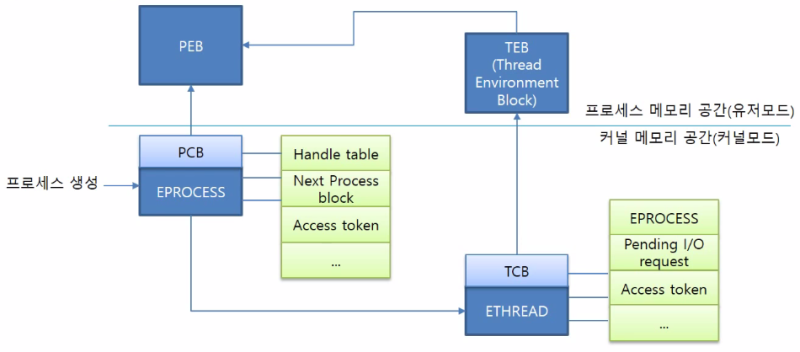
}

###### 난독화 전(위)과 후(아래) decompile 비교

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

## 안티 디버깅(Anti-Debugging)

안티 디버깅은 프로그램이 실행될 때 **디버거**를 감지하고, **OllyDbg**나 **x64dbg**와 같은 디버깅 도구의 사용을 차단하는 기술입니다. 이를 통해 프로그램의 변조나 분석을 방지할 수 있습니다.



## Ⅲ. Protector가 리버스 엔지니어링에 미치는 영향

## 코드 난독화

## Protector는 프로그램의 코드를 난독화하여, 역분석을 시도하는 사람이 코드를 쉽게 이해하지 못하게 만듭니다. 이는 변수명이나 함수명을 의미 없는 이름으로 바꾸거나, 코드의 흐름을 복잡하게 구성함으로써 이루어집니다. 이러한 난독화는 코드를 읽는 것을 어렵게 만들고, 프로그램의 의도를 파악하기 힘들게 합니다.

## 디버깅 방해

## Protector는 디버거 감지 및 방해 기능을 가지고 있어, 리버스 엔지니어가 프로그램을 분석할 때 사용하는 디버깅 도구를 무력화합니다. 프로그램이 디버깅되면 실행을 중단시키거나 예상치 못한 행동을 유발하여 분석을 어렵게 만들죠.

## 안티 디스어셈블리

## Protector는 정적 분석을 방해하는 기능을 제공하여, 프로그램이 디스어셈블리되는 것을 어렵게 합니다. 이로 인해 리버스 엔지니어가 어셈블리 코드를 추출해 분석하는 과정이 복잡해지고, 프로그램 구조를 이해하는 데 큰 장애물이 됩니다.

## 암호화 및 압축

## Protector는 프로그램의 중요한 코드나 데이터를 암호화하거나 압축합니다. 이 기술은 프로그램이 실행되는 동안에만 해당 코드가 복호화되기 때문에, 리버스 엔지니어는 정적 분석만으로는 실제 코드를 확인하기 힘듭니다. 실행 시점에만 복호화되는 코드는 프로그램의 중요한 로직을 보호하는 데 큰 역할을 합니다.

## 엔트로피 증가

## Protector는 프로그램의 구조를 복잡하게 만들어 \*\*엔트로피(무작위성)\*\*를 증가시킵니다. 이는 자동 분석 도구나 패턴 기반 분석 기술을 적용하기 어렵게 만듭니다. 엔트로피가 높은 프로그램은 일관된 패턴이 적어 분석 도구가 제대로 작동하지 않을 가능성이 큽니다.

## 자체 수정 코드 (Self-Modifying Code)

## Protector는 프로그램이 실행 중에 코드 자체를 수정하는 기능을 추가할 수 있습니다. 이런 자체 수정 코드는 리버스 엔지니어가 프로그램의 원래 코드를 정확히 파악하기 어렵게 만들며, 분석 도중 예상치 못한 코드 변화로 인해 분석 난이도가 크게 증가합니다.

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

## 법적 측면

## 저작권 보호

## 긍정적인 영향:

## Protector는 소프트웨어의 소스 코드와 구조를 보호하여, 불법 복제나 저작권 침해를 방지하는 데 중요한 역할을 합니다. 이를 통해 소프트웨어 개발자와 회사는 자산을 안전하게 보호하고, 정당한 수익을 보장받을 수 있습니다.

## 부정적인 영향:

## 그러나 Protector가 너무 강력하게 작동할 경우, 사용자가 정당하게 구매한 소프트웨어에 대한 정당한 사용권이 제한될 수 있습니다. 예를 들어, 사용자가 소프트웨어를 백업하거나 분석하는 것이 불가능해질 수 있으며, 이러한 문제로 인해 법적 분쟁이 발생할 수 있습니다.

## 리버스 엔지니어링과 법적 허용성

## 많은 국가에서는 리버스 엔지니어링이 특정 조건에서 합법적으로 인정됩니다. 예를 들어, 소프트웨어 상호 운용성을 확보하거나 보안 결함을 찾기 위한 목적의 리버스 엔지니어링은 허용될 수 있습니다. 그러나 Protector는 이러한 합법적인 목적을 방해할 수 있으며, 사용자가 법적으로 허용된 권리를 행사하는 데 어려움을 줄 수 있습니다.

## 일부 법적 시스템에서는 소비자 권리를 지나치게 침해하는 보호 기술을 제한하는 규제를 마련할 가능성도 있습니다.

## 디지털 밀레니엄 저작권법(DMCA)및 유사 법률

## 미국의 DMCA와 유럽의 유사 법률은 소프트웨어의 무단 복제를 막기 위한 기술적 보호 조치를 우회하는 행위를 불법으로 규정합니다. Protector는 이러한 법률에 따라 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 강화하는 역할을 하지만, 때로는 정당한 사용자 권리를 지나치게 제한할 수 있습니다. 사용자는 자신이 구매한 소프트웨어를 백업하거나 수정할 권리가 있을 수 있지만, Protector는 이러한 행위를 어렵게 만들 수 있습니다. 특히 소프트웨어를 맞춤 설정하거나 문제를 고치려는 합법적인 시도를 차단하면서 윤리적 논란이 발생할 수 있습니다.

## 보안 연구 제한

## Protector는 소프트웨어의 보안 취약점을 분석하려는 보안 연구자들의 활동을 방해할 수 있습니다. 보안 연구자들은 소프트웨어의 결함을 발견하고 이를 해결해주지만, Protector는 이러한 분석을 방해해 보안 문제를 드러내지 못하게 할 수 있습니다. 이는 소프트웨어가 심각한 보안 취약점을 가지고 있을 때 문제가 될 수 있습니다.

## 소프트웨어의 장기 보존

## Protector는 소프트웨어를 장기적으로 보존하기 위한 아카이빙 작업에도 어려움을 줄 수 있습니다. 시간이 지나 소프트웨어가 더 이상 지원되지 않거나 구형이 될 때, Protector가 너무 강력하게 작동하면 소프트웨어를 분석하고 보존하려는 시도가 차단될 수 있습니다. 이는 소프트웨어의 역사적 가치나 기술적 진보를 기록하는 데 걸림돌이 될 수 있습니다.

## 투명성 부족

## Protector는 사용자가 소프트웨어가 실제로 어떤 행동을 하는지 정확히 알기 어렵게 만듭니다. 특히, 악성 소프트웨어나 스파이웨어가 Protector를 사용해 자신을 숨기는 경우, 이는 큰 윤리적으로 문제가 됩니다. 사용자는 자신이 설치한 소프트웨어가 어떻게 동작하는지 알 권리가 있지만, Protector는 이 권리를 침해할 수 있습니다.

## Ⅴ. 결론

Protector는 소프트웨어의 보안과 저작권 보호를 위해 필수적인 도구입니다. 이를 통해 리버스 엔지니어링을 방해하고, 소프트웨어의 무단 복제와 변조를 방지하여 개발자와 기업의 자산을 효과적으로 보호할 수 있습니다. 그러나 이러한 강력한 보호 기능이 사용자의 합법적인 권리와 보안 연구 활동을 제한할 수 있는 문제도 존재합니다.

따라서 Protector 기술을 사용할 때는 **법적**과 **윤리적** 측면에서 균형을 잘 맞춰야 합니다. 소프트웨어의 보안을 강화하면서도 정당한 사용자가 소프트웨어를 사용할 수 있도록 배려하는 방식을 선택하는 것이 중요합니다. 이렇게 할 때, Protector는 보안과 사용자 권리 사이에서 **조화로운** 역할을 할 수 있을 것입니다.

## 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