*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.30, NO.4, Aug. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.4.605

# VMProtect 동작원리 분석 및 자동 역난독화 구현

방 철 호,[1†] 석 재 혁,[1] 이 상 진[2‡]
[1,2]고려대학교(대학원생, 교수)

# VMProtect Operation Principle Analysis and Automatic Deobfuscation Implementation

Cheol-ho Bang,[1†] Jae Hyuk Suk,[1] Sang-jin Lee[2‡]
[1,2]Korea University(Graduate student, Professor)

요 약

난독화 기술은 프로그램의 기능성은 그대로 유지하면서 자료구조, 제어흐름 등 내부 로직을 변형함으로써 프로그램의 분석을 지연시키는 기술이다. 그러나 이러한 난독화 기술을 악성코드에 적용함으로써 안티바이러스 소프트웨어의 악성코드 탐지율을 저하시키는 사례가 빈번하게 발생하고 있다. 소프트웨어 지적재산권을 보호하기 위하여 적용되는 난독화 기술이 역으로 악성코드에 적용됨으로써 악성코드 탐지율을 저해할 뿐만 아니라 이의 분석을 어렵게 하여 악성코드의 기능성 파악에도 어려움을 주게 되므로 난독화가 적용된 코드를 원본에 가깝게 복원할 수 있는 역난독화 기술의 연구 또한 꾸준히 지속 되어야 한다. 본 논문에서는 상용 난독화 도구 중 대중적으로 널리 알려져 있는 도구인 VMProtect 3.4.0에서 제공하는 세부 난독화 기술 중 Pack the Output File, Import Protection의 옵션이 적용되어 난독화 된 코드의 특징을 분석하고 이의 역난독화 알고리즘을 제시하고자 한다.

ABSTRACT

Obfuscation technology delays the analysis of a program by modifying internal logic such as data structure and control flow while maintaining the program's functionality. However, the application of such obfuscation technology to malicious code frequently occurs to reduce the detection rate of malware in antivirus software. The obfuscation technology applied to protect software intellectual property is applied to the malicious code in reverse, which not only lowers the detection rate of the malicious code but also makes it difficult to analyze and thus makes it difficult to identify the functionality of the malicious code. The study of reverse obfuscation techniques that can be closely restored should also continue. This paper analyzes the characteristics of obfuscated code with the option of Pack the Output File and Import Protection among detailed obfuscation technologies provided by VMProtect 3.4.0, a popular tool among commercial obfuscation tools. We present a de-obfuscation algorithm.

**Keywords:** Digital Forensic, Code De-obfuscation, VMProtect, Packer, Protector

## I. 서 론

상용 소프트웨어의 역공학을 막는 방법 중 하나로 난독화 도구가 사용되고 있다. 소프트웨어 역공학을 어렵게 하기 위한 난독화 기술은 소프트웨어의 원래 기능은 그대로 유지하면서 코드와 데이터 등의 구조를 변형하여 분석을 어렵게 한다.

Colleberg 등[1]은 난독화를 배치 난독화, 자료 난독화, 제어 난독화, 방지 난독화로 분류하였고, 이경률 등[2]은 Colleberg의 분류에 기타 난독화(패

Received(04. 21. 2020), Modified(06. 22. 2020), Accepted(06. 22. 2020)

† 주저자, chbang79@gmail.com
‡ 교신저자, sangjin@korea.ac.kr(Corresponding author)

킹, 가상화)를 추가하여 분류하였다. 우리는 통상 이런 난독화 기술을 조합하여 만들어진 소프트웨어를 난독화 도구라고 부른다.

난독화 도구는 원래 소프트웨어에 대한 저작권 보호 또는 구현 기술을 보호하기 위해 사용되었으나, 악성코드 제작자들이 악성코드에도 난독화 기술을 적용하면서 시그니쳐 기반 탐지하는 안티바이러스 소프트웨어가 무력화되는 역기능들이 생겨나기 시작했다. 따라서 난독화가 적용된 악성코드를 원본에 가깝게 복원할 수 있는 역난독화 기술의 연구 또한 꾸준히 지속 되어야 할 필요가 있다.

대중적으로 널리 알려진 상용 난독화 도구로는 Themida, VMProtect 등이 있다. Themida와 관련한 연구는 Themida+WinLicense 2.x[3], Ultra Unpacker v1.4[4]에서 특정 어셈블리 패턴을 찾아서 언패킹을 진행하는 방식의 연구가 있었고, 목성균 등[5] 연구에서는 동적 슬라이싱 기법을 사용하여 난독화 된 실행일의 전체 트레이스(실행 로그)중 시스템콜 호출과 관련된 명령어만을 추출한 결과를 제시하였으며, Yadegari 등[6]은 난독화 된 실행파일의 역난독화에 대한 일반적인 접근 방법을 제안하였다. 또한 Kang 등[7]은 메모리 쓰기 명령어에서 분기 명령어를 수행했을 때 목적지를 "dirty 영역"으로 표시하고 명령어가 해당 영역에서 실행되었을 때 전체를 덤프하여 분석하는 연구를 하였고, 이재휘 등[8]은 난독화 된 실행 파일의 API 숨김(wrapping) 옵션에 대해서 역난독화 분석을 진행하였으며, 강유진 등[9]은 Themida의 동작 방식(패킹)에 기반하여 원본 프로그램 정보가 전부 복원되고 원본 코드 영역으로 제어권이 넘어가는 정확한 시점에 이를 저장하는 방식을 연구하였는데 본 논문의 연구 방향과 일부 유사한 면이 있다. 마지막으로 이재휘 등[10]은 최신 버전의 Themida에서 가상 메모리를 할당하지 않는 API 난독화를 분석하는 방안을 제시하였다.

본 논문은 악성코드를 난독화 하는데 많이 사용되는 난독화 도구인 VMProtect의 역난독화를 다룬다. 파이어 아이의 보고서[11]에 따르면 중국 정부가 지원하는 해커집단이 헬스케어, 하이테크, 미디어, 게임, 암호화폐 관련 기업을 해킹하는 사이버 공격에 사용된 악성코드가 VMProtect로 난독화 되었다고 한다. 안티 바이러스 프화벽 등, 정보보호 관련 제품을 개발하고 있는 슬로바키아에 위치한 소프트웨어 회사인 ESET에서 발간한 보고서[12]에 따르면 중국에 근거지를 두고 있는 해커 조직인 winnti가 VMProtect를 사용하였다고 한다.

VMProtect의 난독화 옵션에는 메모리에 있는 데이터가 변경되면 에러 메시지를 띄우는 Memory Protection, 임포트 되는 API 리스트를 숨기는 Import Protection, 파일을 실행 압축하는 Pack the Output File, 디버거를 탐지하는 Debugger Detection, 가상 환경을 탐지하는 Virtualization Tools Detection 등이 있다.

그뿐만 아니라 VMProtect가 가진 기능 중에는 보호 대상 코드 영역에 마커(Marker)를 삽입하여 해당 영역을 자체 제작된 가상머신에서 실행되도록 코드자체에 변경을 가하는 가상화 난독화 기술도 존재한다. 가상화가 적용되는 보호 대상 코드영역은 원본의 코드로 복원되는 시점이 존재하지 않기 때문에 일반적인 코드 packing이나 코드 암호화 기술에 비하여 보호 강도가 높은 기술이다. 그러나 가상화 난독화 기술이 실제 악성코드에 적용되면 악성코드 파일 자체의 크기가 극단적으로 증가하기 때문에 악성행위 수행에 소모되는 리소스가 크게 증가하여 실질적으로는 많이 사용되지 않을 것으로 판단되어 본 논문에서는 가상화 난독화 기술에 대해서는 다루지 않는다. 김순곤[13]은 VMProtect의 난독화 옵션 중 가상화 난독화 기술이 적용된 바이너리 파일에 대해 정적 코드 분석, 동적 코드 분석, 최적화를 통해 역난독화하는 연구를 진행하였다.

본 논문에서는 가상화 기반 연구를 제외한 VMProtect의 기본적인 옵션에 대한 연구로 VMproetct 난독화 옵션 중 Pack the Output File, Import Protection에 한정하여 연구를 진행하였으며 Intel에서 제작/배포하는 동적 바이너리 계측 도구인 Pin[14]을 이용한 자동 역난독화 방안을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 배경 지식을 서술하며, 3장에서 VMProtect 난독화 특징 분석, 4장에서 VMProtect 자동 역난독화 방안 설계, 5장에서 역난독화 수행 결과를 서술하며, 6장에서 결론을 제시한다.

## II. 배경지식

이 절에서는 VMProtect의 사용에 따른 실행 파일의 내부 구조 변형을 분석하기 위해 PE(Portable Executable) 구조를 간략히 설명하고, VMProe

tct의 기능에 대해 간략히 살펴본다. 이후 역난독화에 사용하기 위한 도구인 Pin을 간략히 소개한다.

### 2.1 32비트 윈도우 실행파일 PE 구조

32비트 PE 파일의 구조는 Fig. 1.과 같다. IMAGE_DOS_HEADER, IMAGE_FILE_HEADER, IMAGE_OPTIONAL_HEADER, 복수 개의 IMAGE_SECTION_HEADER, 복수 개의 Section Body로 구성된다. PE 파일의 최상단에 IMAGE_DOS_HEADER 구조체(총 64바이트)가 존재하며, IMAGE_DOS_HEADER 내부의 e_lfanew 필드 값에 IMAGE_FILE_HEADER 구조체(총 20바이트)의 시작 위치가 저장되어 있다. IMAGE_FILE_HEADER의 내부 필드 중 NumberofSections에 섹션 헤더와 섹션 바디의 개수가 기록되어 있다. 또한 SizeofOptionalHeader에는 IMAGE_OPTIONAL_HEADER의 크기가 저장되어 있고, IMAGE_FILE_HEADER에 연이어 IMAGE_OPTIONAL_HEADER가 있다. IMAGE_OPTIONAL_HEADER 내부 필드 중 ImageBase 필드 값에 PE 이미지의 프로세스 상 시작주소가 있고, SizeofImage를 통해 PE 이미지의 전체 크기를 알 수 있다. 마지막으로 IMAGE_OPTIONAL_HEADER에 연이어 IMAGE_SECTION_HEADER가 시작되며, IMAGE_SECTION_HEADER 내의 VirtualAddress와 ImageBase 필드 값을 더함으로써 각 섹션의 시작 위치를 알 수 있다.

| IMAGE_DOS_HEADER(64byte) |
|---|
| DOS stub |
| IMAGE_FILE_HEADER(20byte) |
| IMAGE_OPTIONAL_HEADER(224byte) |
| [0] IMAGE_SECTION_HEADER(40byte) |
| [1] IMAGE_SECTION_HEADER(40byte) |
| [2] IMAGE_SECTION_HEADER(40byte) |
| .... |
| .text text section |
| .data, data section |
| .idata, import section |
| ... |

Fig. 1. 32bit PE file format

### 2.2 VMProtect 기능 소개

VMProtect는 Ekaterinburg 사에서 출시한 난독화 도구로 2019. 8. 3.에 버전 3.4가 발표되었다. VMProtect를 구동하면 Fig. 2.와 같은 화면이 뜨면서 난독화 옵션으로 Memory Protection, Import Protection, Resource Protection, Pack the Output File, Debugger Detection, Virtualization Tools Detection 등이 제시된다.

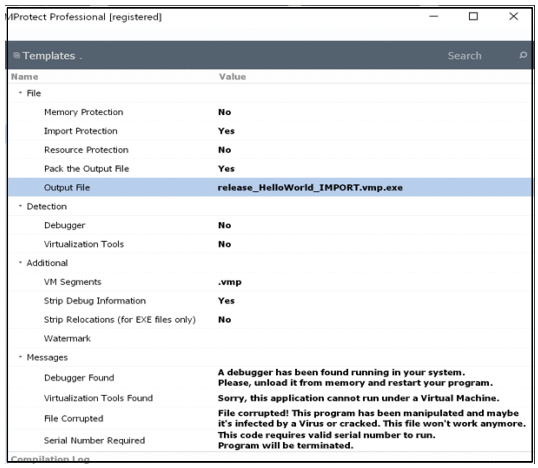


Fig. 2. Obfuscation Options of VMProtect

### 2.3 Pin 소개

Pin은 실행 파일의 특성을 분석하고 계측하는 도구로써, 전체적인 구조는 Fig. 3.과 같다. 실행파일을 자동으로 분석하고 계측하는 라이브러리인 Instrumentation API와 계측하고자 하는 실행파일을

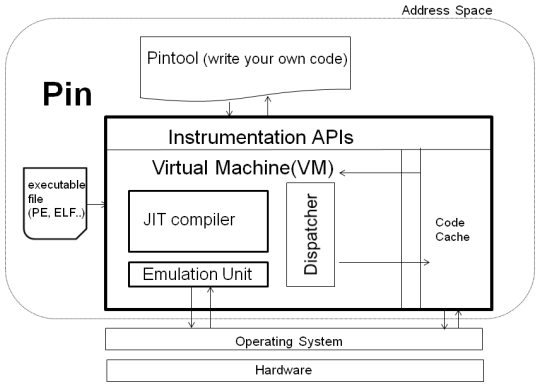


Fig. 3. Pin software Architecture

입력으로 받아 에뮬레이션 하는 JIT (Just-In-Time) 컴파일러 형태로 실행시키는 VM(Virtual Machine)으로 구성되어 있다. 프로그래머가 Instrumentation API를 이용하여 계측하고자 하는 목적에 맞게 코드를 설계한 Pintool로 Pin을 구동하면서 입력으로 받은 실행 파일의 특징을 분석하고 계측한다.

## III. VMProtect 난독화 특징 분석

### 3.1 Pack the Output File0l 적용된 난독화 특징

PE 파일에 VMProtect의 Pack the Output 옵션을 적용하면, Fig. 4.와 같이 PE 구조 자체에 외형적인 변형이 발생한다. PE 파일 상에서 vmp0, vmp1 섹션 헤더가 추가되고, 실제 섹션에는 vmp1과 rsrc만 파악할 수 있다.

vmp1, rsrc을 제외한 나머지 섹션은 Pointer to Raw Data가 0으로 기록되어 있으며 Virtual Size만 기록되어 있기 때문에 정적분석 도구로 분석하면 이러한 섹션은 모두 감춰진다.

Fig. 5.와 같이 Pack the Output File 옵션을 적용하면 DOS stub에도 변경이 일어나지만 PE 파일의 실행에는 영향을 주지 않기 때문에 DOS stub은 조사할 필요가 없다. Pack the Output File 옵션이 적용되었을 때 섹션 내에서 데이터만 변경

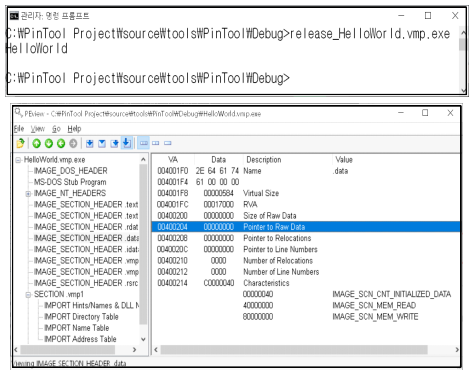


Fig. 4. applying VMProtect to the original program that outputs HelloWord string

되는 부분은 IMAGE_FILE_HEADER의 Number of Section, Time Date Stamp 필드와 IMAGE_OPTIONAL_HEADER의 Address of Entry Point, Size of Image, 그리고 DATA_DIRECTORY_ARRAY 값이다. IMAGE_SECTION_HEADER.text, IMAGE_SECTION_HEADER.data, IMAGE_SECTION_HEADER.idata에서는 공통적으로 Size of Raw Data, Pointer to Raw Data 필드의 값이 변경된다.

이에 따라 text, rdata, data, idata 섹션은 모두 사라지고, vmp1 섹션이 추가된다.

Pack the Output File 옵션이 적용된 PE 파일은 Address of Entry Point에 ImageBase 값

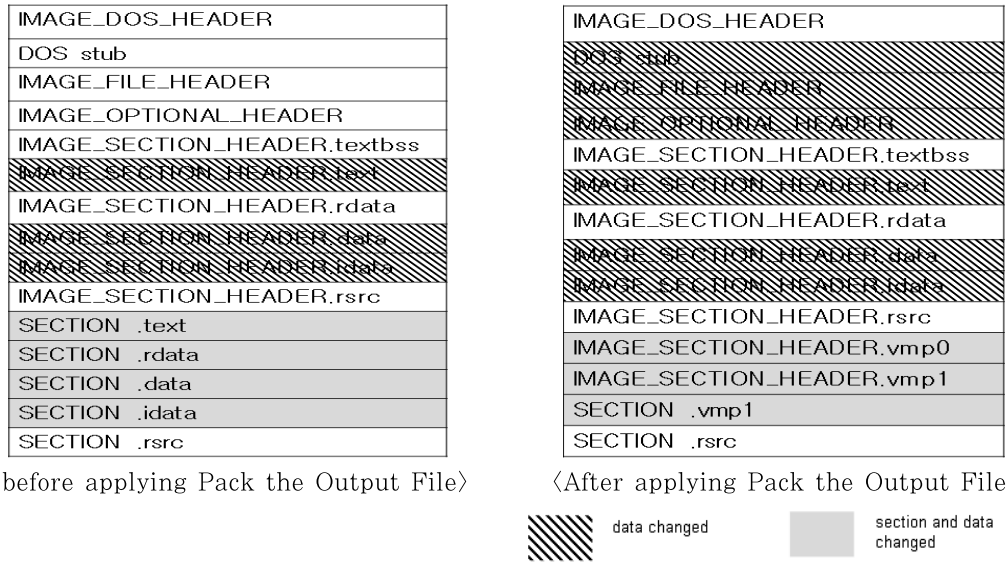


Fig. 5. Section comparison before and after Pack the Output File option

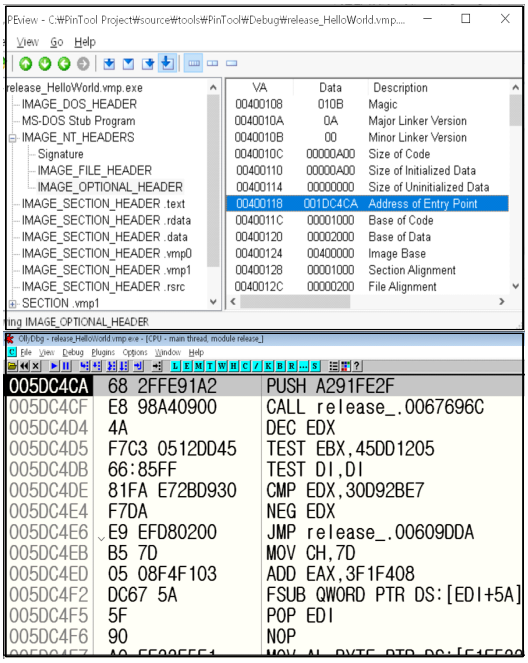


Fig. 6. Checking the Entry Point by applying the Pack the Output option to the original program (Helloworld.exe)

을 더한 주소인 0x00500C4CA에서 실행이 시작되는데, 이 위치는 Fig. 6.과 같이 vmp1 섹션 대역 내부에 해당한다. 즉, Pack the Output File 옵션이 적용된 PE 파일은 vmp1 섹션이 Entry Point Section이 된다.

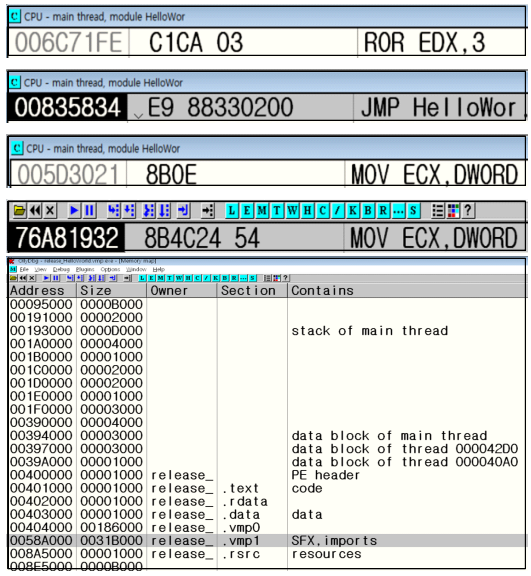


Fig. 7. Address band where the program with the Pack the Output File option (Helloworld.exe) is execute

Fig. 7.과 같이 Pack the Output File 옵션이 적용된 Helloworld.exe가 실행되는 위치의 주소대역은 OriginalEntryPoint로 이동하기 전까지 vmp1 섹션의 주소 대역인 0x0058A000~0x008A000을 벗어나지 않고 vmp1 섹션 내에서 루틴이 지

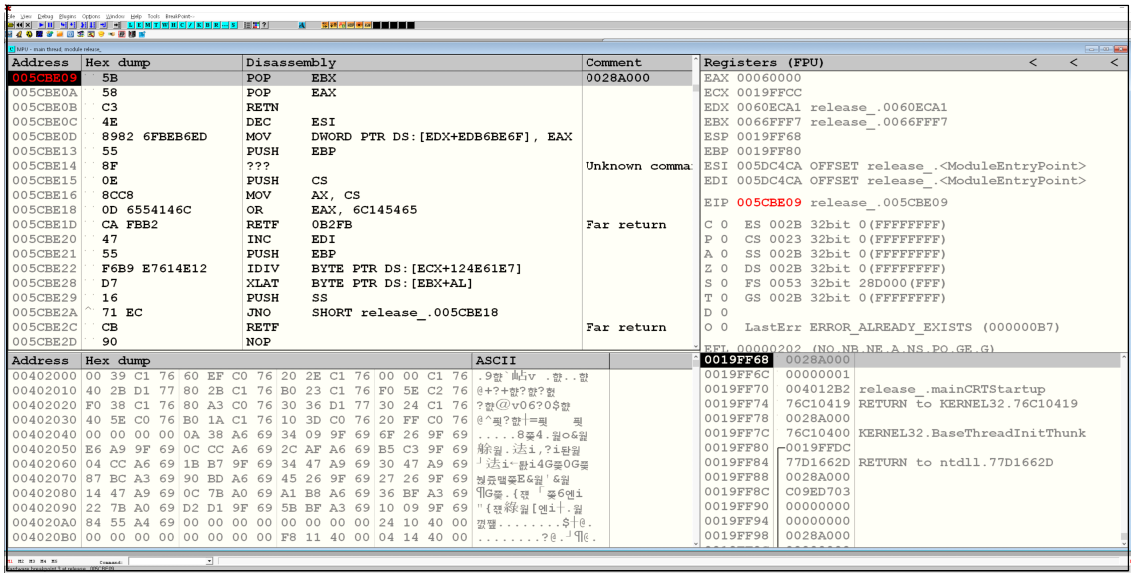


Fig. 8. Immediately before calling the Original Entry Point

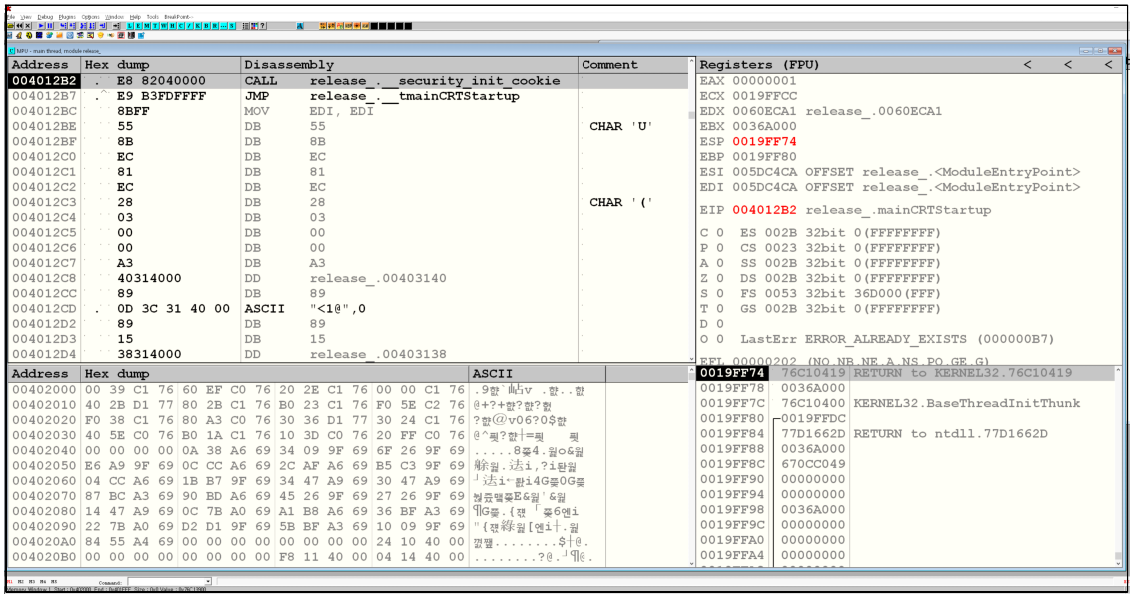


Fig. 9. Result of Control Transfer to Original Entry Point

속적으로 실행된 뒤, Fig. 8.과 같이 실행되는 명령어의 주소가 vmp1 섹션의 주소대역을 벗어나는 지점이 존재한다.(스택의 TOP에서 3번째 위치의 값에 OEP 주소가 위치할 때 RETN 명령에 의해서 OriginalEntryPoint로 이동하게 된다. 해당 예시에서는 0x004012B2가 OEP 주소이다.)

Fig. 9.와 같이 vmp1 주소대역을 벗어난 후 최초로 제어권이 이동되는 대상 주소인 0x4012B2가 원본 코드가 실행되는 OriginalEntryPoint임을 알 수 있다. 이러한 특성을 이용하여 OEP 탐색 알고리즘을 Pin으로 구현한 결과를 4절에서 제시한다.

### 3.2 Import Protection이 적용된 난독화 특징

Import Protection을 적용하면 라이브러리 파일 내의 함수(API)를 호출하는 방식이 변형된다. 원본 프로그램의 경우 IAT(Import Address Table)에 적재된 API 주소를 참조하여 해당 API를 호출하는 간접호출 방식을 사용하고 있으며, 이에 따라 원본 프로그램의 코드영역에서도 간접 호출문(Indirect Call)을 발견할 수 있다. 반면, Import Protection 옵션이 적용되면, API 호출이 간접호출 방식에서 직접호출 방식으로 변경된다.

Fig. 10.과 같이 일반적으로 printf 함수를 호출하는 경우, IAT의 주소인 0x004020A0를 참조하여 실제 printf 함수의 시작주소인 0x64595584를 호출하는 형태로 간접호출이 수행되나, Import Protection을 적용하게 되면 IAT를 참조하여 간접호출하는 방식이 아닌 0x00404000 ~ 0x00586fff 대역의 vmp0 섹션의 주소인 0x0041475A를 직접호출하며 특정한 API Address Resolving 과정을 통하여 0x69275584의 printf 주소를 호출하는 형태로 호출 방식이 변경된다.(Fig. 11. 참고).

이것은 일종의 Import Address Resolving 과정으로 볼 수 있으며 원본 코드가 unpacking 되어 제어권이 원본 코드 영역에서 실행되고 있다가, 다시 vmp0 섹션으로 되돌아가는 경우 이러한 직접 호출이 발생하는 것으로 판단할 수 있다. 또한, vmp0 섹션에서 최초로 벗어난 지점이 printf 함수 시작주소이므로 이를 PinTool에 구현하는 것이 가능하다. 즉, Import Address Resolving 과정을 상세분석하지 않더라도 Resolving 종료 시점을 파악할 수 있기 때문에 실제 피호출 함수의 주소를 PinTool로 기록하는 것이 가능하며, 이는 Import Address Protection 옵션에 대한 역난독화 방안이 된다.

요약하자면, 역난독화를 위해서는 vmp0 섹션에서 최초로 벗어나서 처음 실행된 명령어의 시작 주소가 import된 함수의 시작 주소라는 점을 이용하여 실제 import 함수의 시작 주소를 알 수 있다.

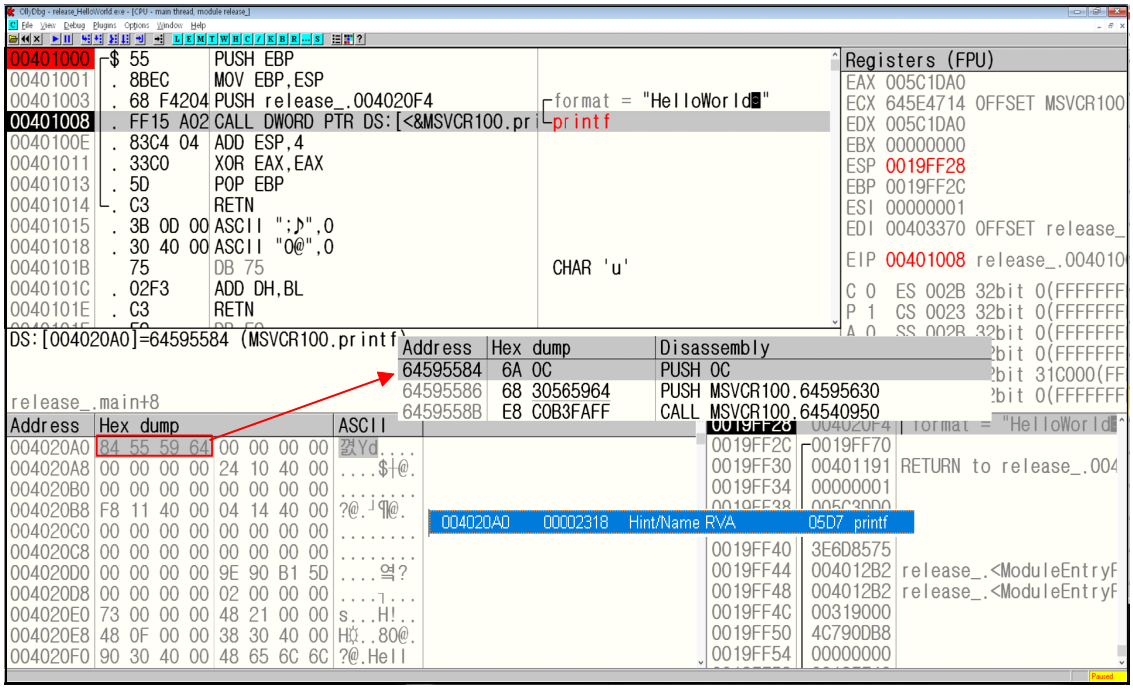


Fig. 10. Indirect Call Method without Import Protection

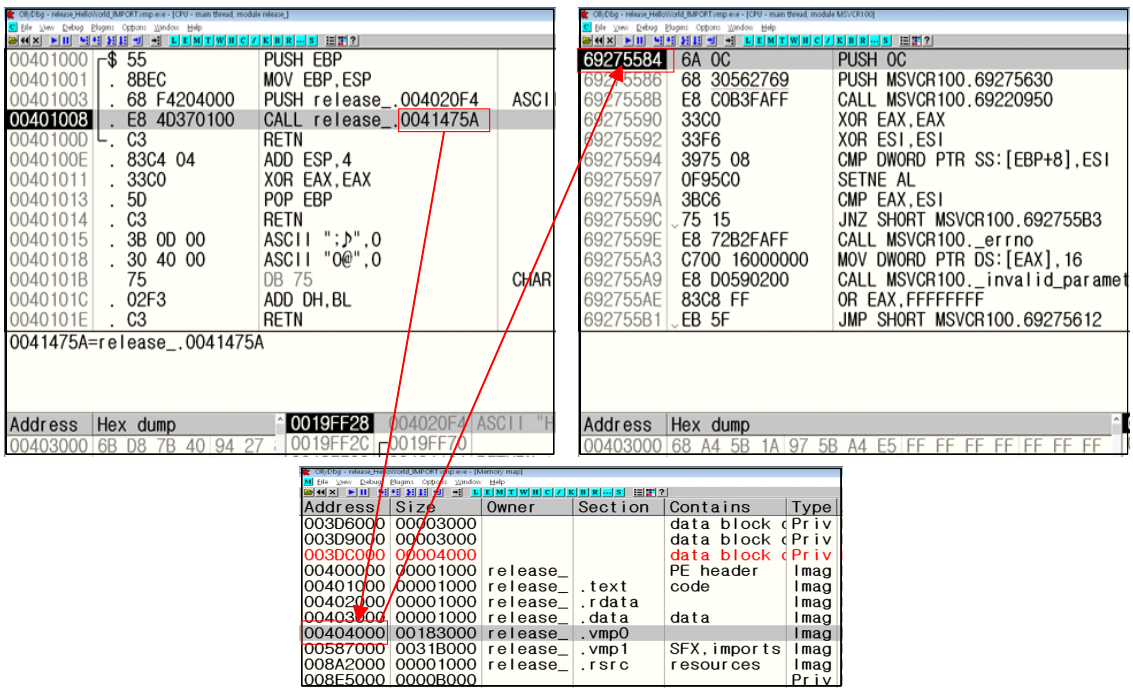


Fig. 11. Direct call method with Import Protection

## IV. VMProtect 자동 역난독화 방안 설계

### 4.1 Pack the Output File이 적용된 경우의 자동 역난독화 방안

Pack the Output File 옵션이 적용된 난독화의 역난독화 방안은 Fig. 12. 알고리즘과 같다. Pack the Output File 옵션을 적용한 난독화된 바이너리 파일을 Pin으로 실행시키면 우선 난독화된 바이너리 파일의 PE 이미지 내에서 섹션 헤더 정보를 이용하여 각 섹션의 시작주소와 섹션의 크기를 모두 특정 배열들에 저장한다. 이후 Pin 상에서 다시 난독화된 바이너리 파일의 명령어(인스트럭션)가 실행된다. 명령어가 실행되면서 현재 실행 중인 명령어의 주소가 원본코드 주소 대역에 들어가게 되는 순간 명령어의 시작주소를 난독화된 바이너리 파일의 OEP로 간주하여 저장한다. 이 순간의 실제 데이터 및 코드가 들어있는 영역도 난독화가 해제된 상태의 데이터이므로 이들을 저장하는 형태의 알고리즘이며, Table 1. 은 해당 알고리즘을 pintool를 이용하여 C++로 구현한 코드 부분이다.

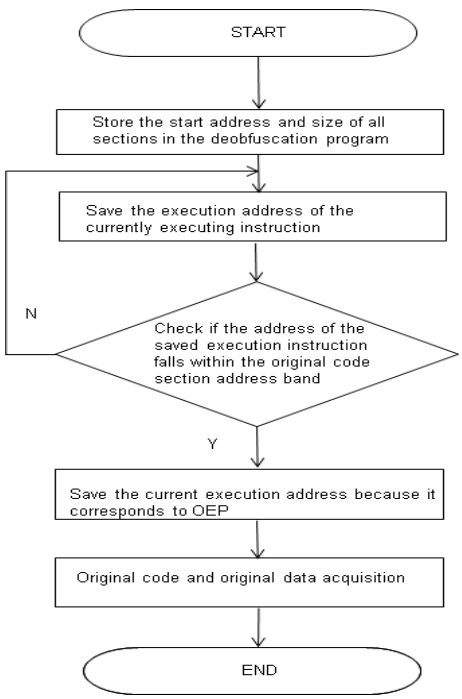


Fig. 12. Algorithm for acquiring original entry point and original data

Table 1. c++ code for acquiring original entry point and original data using pintool

```
Addr = INS_Address(ins);
if(OEPFlag == false)
{
if(SectionAddr[0] <= Addr && Addr < Section
Addr[2]) {
    for (ADDRINT Addr = SectionAddr[Inde
x]; Addr < SectionAddr[Index+1]; Addr +=
4)
    {fwrite((ADDRINT*)Addr, 1, 4, OriginalIn
fo);}
                }
OEP = Addr;
}
```

### 4.2 Import Protection이 적용된 경우 자동 역난독화 방안

3.2. 소절에서 설명한 바와 같이 Import Protection 옵션을 적용하게 되면, 간접호출 방식이 특정한 resolving 과정을 통해서 직접 주소를 호출하는 방식으로 바뀌는 것을 알 수 있다. 명령어 제어의 이동이 원본코드 주소 대역에 들어갔다가 다시 vmp0를 벗어나자마자 실행되는 명령어 주소가 실제 호출된 함수의 시작 주소이다.

따라서 Import Protection 옵션의 역난독화 알고리즘은 Fig. 13. 알고리즘과 같다. 우선 명령어 제어의 이동이 원본코드 주소대역 들어간다는 가정하에 현재 실행 중인 명령어의 주소를 저장하고 이 명령어의 주소가 vmp0 섹션 대역 내의 주소인지를 확인한다. 이 명령어가 vmp0 섹션 대역 내의 주소가 맞다면 플래그를 세팅하고 다시 이 명령어의 주소를 저장한다. 이 명령어 주소의 바로 직전 명령어의 주소가 Import Protection 옵션을 적용하지 않았을 때 Import address table의 주소를 호출하는 함수에 해당하는 주소이다. 플래그가 세팅된 상황에서 계속 현재 실행 중인 명령어의 주소를 저장하고 이 주소가 다시 vmp0 섹션 대역 외의 최초 주소이면 이 주소 자체가 직접 점프하는 함수의 시작 주소가 되는 것이다. 결국 import 함수를 호출한 지점의 주소와 매칭되는 import 함수의 주소를 알게 됨으로써 난독화를 해제할 수 있는 것이다. Table 2는 해당 알고리즘을 pintool을 이용하여 C++로 구현한 코드부분이다.

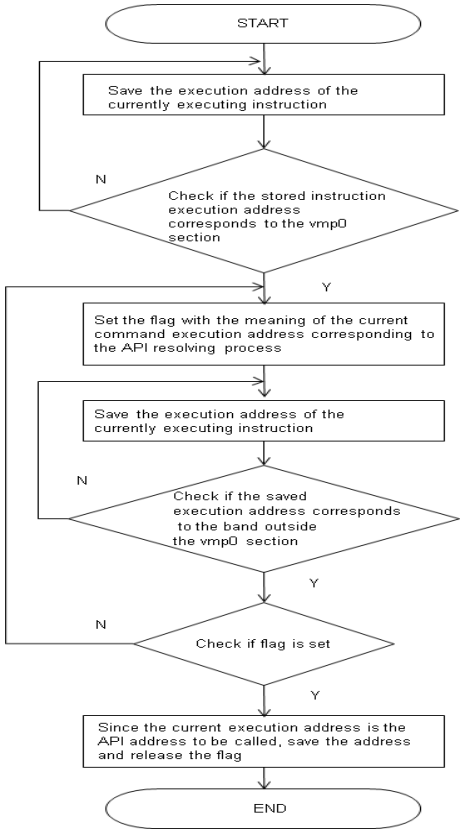


Fig. 13. algorithm for deobfuscating import protection

Table 2. c++ code for deobfuscating import protection using pintool

```
if((OEPFlag == true)&&(vmp0_SectionAddr
<= Addr && Addr < (vmp0_SectionAddr+vmp
0_SectionSize)))
{
      VMP0Flag = true
}
if((VMP0Flag == true)&&(vmp0_SectionAddr
> Addr || Addr >= (vmp0_SectionAddr+vmp0_
SectionSize)))
{
  printf(" Deob addr : 0x%X\n", Addr);
  VMP0Flag = false
}
```

## V. 역난독화 수행 결과

본 논문에서는 Microsoft visual c++ 2010 Express 상에서 Pin Instrumentation API를 이용하여 Original Entry Point 와 원본 데이터를 획득하는 알고리즘과 Import Protection을 역난독화 하는 알고리즘을 적용한 dll(dynamic linkng library) 파일인 Pintool 라이브러리를 설계하여 역난독화를 진행하였다.

Fig. 14.와 같이 cmd.exe 원본 파일에 난독화를 적용하고 다시 난독화를 해제한 뒤 IMAGE_OPTIONAL_HEADER까지 복원하여 난독화를 적용하기 전과 난독화를 적용하고 다시 난독화를 해제한 결과의 ImageBase값과 Address of Entry Point의 값이 각각 둘 다 동일함을 확인할 수 있다.

Fig. 16.와 같이 Import Protection이 적용된 실행 파일에서 import된 함수의 시작위치를 디버깅을 통해 찾아낸 주소 0x66FD5584와 Pin을 이용하여 Import Protection이 적용된 실행파일에 대해서 Import Protection을 역난독화하여 import된 함수의 시작 주소가 동일한 0x66FD5584 임을 알 수 있다.

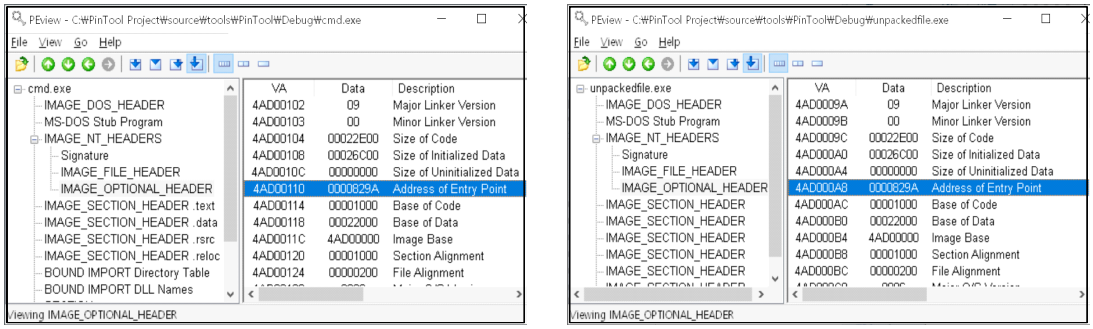


Fig. 14. IMAGE_OPTIONAL HEADER information from the original file (left) and IMAGE_OPTIONAL_HEADER information from the unpacked file after Pack the Output File obfuscation (right)

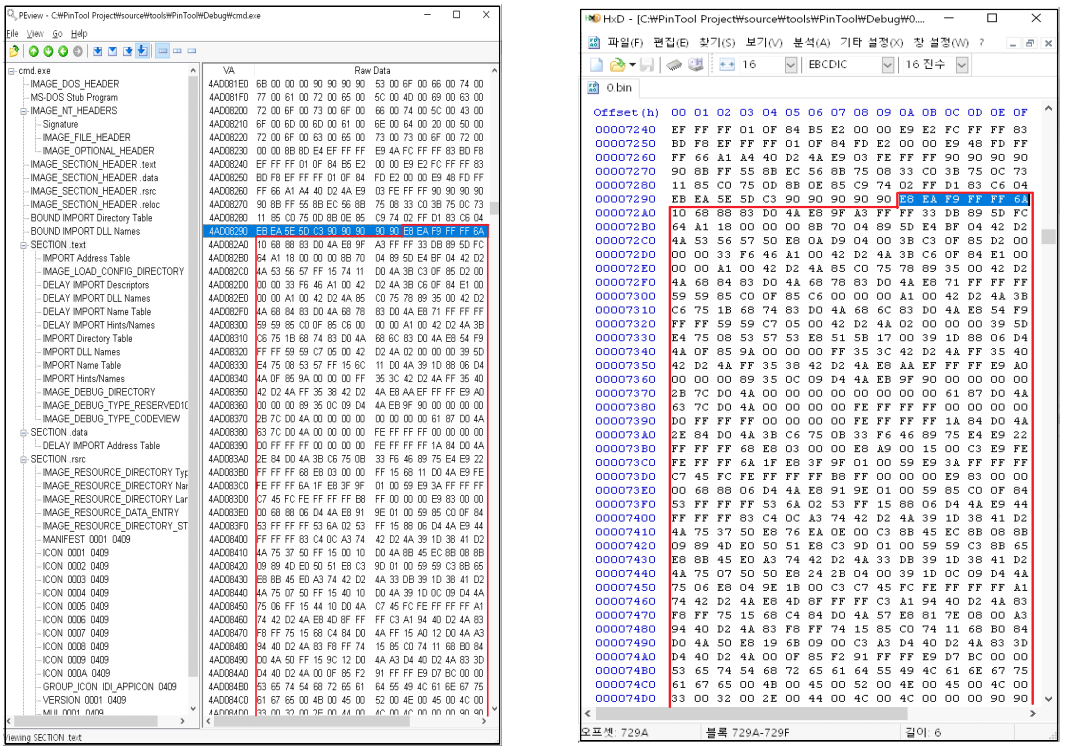

Fig. 15. Information of the ..Text section of the original file (left) and information of the .Text section of the file that was de-obfuscated after applying the Pack the Output File (right).

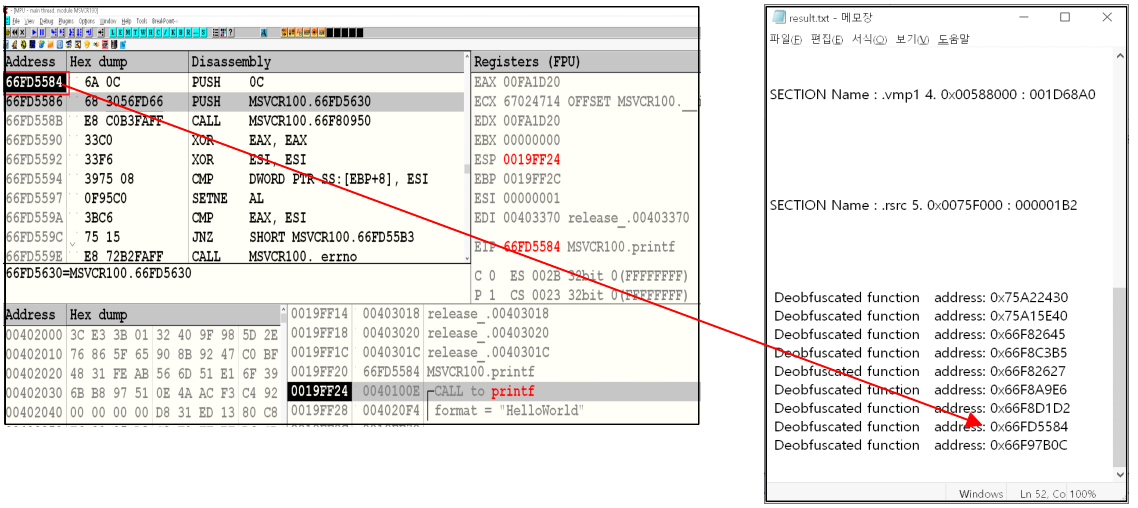

Fig 16. Figure, where olly debugger finds the starting position of an imported function in an executable with Import Protection (left). Figure that finds the starting position of the imported function by de-obfuscated using Pin (right).

## VI. 결 론

악성코드에 적용되는 난독화 프로그램들 중 VMProtect 최신 버전을 대상으로 하여 Pack the Output File 옵션과 Import Protection 옵션이 적용되었을 때 이를 자동으로 역난독화 하는 방법을 제시

하였다. 그 결과로 제어가 .Text 영역으로 이동하는 시점에서 원본과 동일한 데이터가 생성되는 것을 확인하였고 이 시점이 OEP라는 것을 확인할 수 있었다. 또한 Import protection 옵션은 실행 파일 내부에 참조하는 import 함수의 주소들을 난독화하는데, 이를 역난독화 함으로써 import 함수의 실제 위치를 자동으로 찾을 수 있었다.

VMProtect의 모든 난독화 옵션에 대해 분석하지 못하였으나 본 논문에서 제안한 방법을 통해 VMProtect가 적용된 악성코드를 분석하는데 걸리는 시간과 노력을 단축할 수 있을 것이다. 향후 연구로써 IAT를 프로그램으로 재구축하여 완전 자동화하고 나머지 옵션들에 대해서도 연구를 진행할 필요가 있다.

본 연구는 VMProtect의 역난독화에 대한 자동화 가능성을 제시한 것으로 의의가 있으며, 후속 연구에서 좀 더 일반적인 환경에서의 연구를 통해 지속적인 발전이 이루어지길 희망한다.

## References


[1] C.Collberg, C.Thomborson, and D. Low, “A taxonomy of obfuscating transformations,” Department of Computer Science, The University of Auckland, New Zealand, 1997.

[2] Gyeong-Ryul Lee, Heong-Jun Yuk, Gang-Bin Im, Il-Seon Yu, “Trends in obfuscation technology for software security.” Communications of the Korean Institute of Information Scientists and Engineers. 34(1):22-27. 2016.

[3] LCF-AT, “Themida+WinLicense 2.x (Unpacking),” Jul. 2013 (also see https://tuts4you.com/download.php?view.3495)

[4] LCF-AT, “Themida+WinLicense 2.x (Ultra Unpacker v1.4)”, Jan. 2014. (also see http://tuts4you.com/download.php?view.3526)

[5] Seong-Kyun Mok, Hyeon-gu Jeon and Eun-Sun Cho, "Program Slicing for Binary code Deobfuscation," Journal of the Korea Institute of Information Security & Cryptology, 27(1), pp. 59-66, 2017.

[6] B.Yadegari, B.Johannesmeyer, B.Whitely and S.Debray, "A generic approach to automatic deobfuscation of executable code," IEEE Symposium on Security and Privacy, pp. 674-691, 2015.

[7] Min-Gyung Kang, P.Poosankam and H.Yin, "Renovo: A hidden code extractor for packed executables," Proceedings of the 2007 ACM workshop on Recurring malcode. ACM, pp. 46-53, 2007.

[8] Jae-hwi Lee, Jae-hyeok Han, Min-woo k Lee, Jae-mun Choi, Hyun-woo Baek and Sang-jin Lee, "A Study on API Wrapping in Themida and Unpacking Technique," Journal of the Korea Institute of Information Security & Cryptology, 27(1), pp. 67-77, Feb, 2017.

[9] You-jin Kang, Moon Chan Park, Dong Hoon Lee. “Implementation of the Automated De-Obfuscation Tool to Restore Working Executable.” Journals of the Korea Institute Of Information Security And Cryptology, 27(4), 785-802, 2017.

[10] Jae-hwi Lee, Byung-hee Lee, Sang-hyun Cho. “A Study on the Analysis Method to API Wrapping that Difficult to Normalize in the Latest Version of Themida.” Journals of the Korea Institute Of Information Security And Cryptology, 29(6), 1375-1382, 2019.

[11] https://www.fireeye.com/blog/threat-research/2019/10/lowkey-hunting-for-the-missing-volume-serial-id.html

[12] https://www.welivesecurity.com/wp-content/uploads/2019/10/ESET_Winnti.pdf

[13] Soon-Gohn Kim. “Code Automatic Analysis Technique for Virtualization-based Obfuscation and Deobfuscation”, Journal of Korea Institute of Information, Electronics, and Communication Technology, 11(6), 724-731, 2018.

[14] Pin: Chi-Keung Luk Robert Cohn Robert Muth Harish Patil Artur Klauser Geoff Lowney Steven Wallace Vijay Janapa Reddi Kim Hazelwood, Building Customized Program Analysis Tools with Dynamic Instrumentation, page 190-192.

## 〈저자소개〉

방 철 호 (Cheol ho Bang) 정회원
2005년 2월: 연세대학교 의용전자공학과 학사
2018년 9월~현재: 고려대학교 정보보호대학원 석사과정
2019년 3월~현재: 서울지방경찰청 국제범죄수사대 산업기술보호수사팀 근무
〈관심분야〉 디지털포렌식, 정보보호, 코드 난독화, 산업기술보호수사

석 재 혁 (Jae Hyuk Suk) 학생회원
2012년 2월: 서울시립대학교 전자전기컴퓨터공학부 학사
2014년 2월: 고려대학교 정보보호대학원 석사
2014년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 소프트웨어 보안, 소프트웨어 역공학, 코드 난독화

이 상 진(Sang-jin Lee) 종신회원
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
〈관심분야〉 디지털포렌식, 심층암호, 해시암호