Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.4, Aug. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.4.785

실행 파일 형태로 복원하기 위한 Themida 자동 역난독화 도구 구현*

강 유 진,† 박 문 찬, 이 동 훈‡
고려대학교 정보보호대학원

Implementation of the Automated De-Obfuscation Tool to Restore Working Executable*

You-jin Kang,† Moon Chan Park, Dong Hoon Lee‡
Graduated School of Information Security, Korea University

요 약

악성코드를 이용한 사이버 위협이 꾸준히 증가함에 따라 많은 보안 및 백신 관련 업체들이 악성코드 분석 및 탐지에 많은 노력을 기울이고 있다. 그러나 소프트웨어의 분석이 어렵도록 하는 난독화 기법이 악성코드에 적용되어 악성코드에 대한 빠른 대응이 어려운 실정이다. 특히 상용 난독화 도구는 빠르고 간편하게 변종 악성코드를 생성해 낼 수 있기 때문에 악성코드 분석가가 새로운 변종 악성코드의 출현 속도에 대응할 수 없도록 한다. 분석가가 빠르게 악성코드의 실제 악성행위를 분석하도록 하기 위해서는 난독화를 해제하는 역난독화 기술이 필요하다.

본 논문에서는 상용 난독화 도구인 Themida가 적용된 소프트웨어를 역난독화하는 일반적인 분석방법론을 제안한다. 먼저 Themida를 이용하여 난독화가 적용된 실행파일을 분석하여 알아낸 Themida의 동작 원리를 서술한다. 다음으로 DBI(Dynamic Binary Instrumentation) 프레임워크인 Pintool을 이용하여 난독화된 실행파일에서 원본 코드 및 데이터 정보를 추출하고, 이 원본 정보들을 활용하여 원본 실행파일에 가까운 형태로 역난독화할 수 있는 자동화 분석 도구 구현 결과에 대해 서술한다. 마지막으로 원본 실행파일과 역난독화한 실행파일의 비교를 통해 본 논문의 자동화 분석 도구의 성능을 평가한다.

ABSTRACT

As cyber threats using malicious code continue to increase, many security and vaccine companies are putting a lot of effort into analysis and detection of malicious codes. However, obfuscation techniques that make software analysis more difficult are applied to malicious codes, making it difficult to respond quickly to malicious codes. In particular, commercial obfuscation tools can quickly and easily generate new variants of malicious codes so that malicious code analysts can not respond to them. In order for analysts to quickly analyze the actual malicious behavior of the new variants, reverse obfuscation(=de-obfuscation) is needed to disable obfuscation.

In this paper, general analysis methodology is proposed to de-obfuscate the software used by a commercial obfuscation tool, Themida. First, We describe operation principle of Themida by analyzing obfuscated executable file using Themida. Next, We extract original code and data information of executable from obfuscated executable using Pintool, DBI(Dynamic Binary

Received(04. 14. 2017), Modified(1st: 06. 30. 2017, 2nd:08. 09. 2017), Accepted(08. 10. 2017)
* 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임(No. NRF-2014M3C4A7030649)
† 주저자, lovelucky33@naver.com
‡ 교신저자, donghlee@korea.ac.kr(Corresponding author)

Instrumentation) framework, and explain the implementation results of automated analysis tool which can deobfuscate to original executable using the extracted original code and data information. Finally, We evaluate the performance of our automated analysis tool by comparing the original executable with the de-obfuscated executable.

Keywords: De-obfuscation, Program Analysis, Software Protection, Automatic Analysis Tool

I. 서 론

악의적인 소프트웨어 사용자들은 소프트웨어를 무단으로 사용하기 위해 크랙 버전을 만들어 사용하거나 금전적 이익을 취하기 위해 불법적으로 이를 배포하여 소프트웨어 지적재산권을 침해한다. 이러한 악의적인 역공학 및 소프트웨어 변조를 방지하기 위한 대응 방안으로써 소프트웨어 난독화가 활용되고 있다. 난독화란 프로그램의 기능성은 그대로 유지하면서 분석을 어렵게 하기 위해 코드를 변형하는 기술로써[1], 난독화가 적용되면 분석을 방해하여 악의적인 역공학과 소프트웨어 변조를 지연시킬 수 있으므로 소프트웨어 지적재산권을 보호할 수 있다.

그러나 소프트웨어 지적재산권 보호를 위해 쓰이는 난독화 기술이 악성코드의 탐지 회피 및 보호 수단으로 악용되면서 문제가 되고 있다. Fig.1.의 최근 악성코드 동향 보고서[2]에 따르면 백신의 탐지와 분석가의 분석으로부터 악성코드를 은닉하기 위하여 난독화 기법이 적용된 악성코드가 꾸준히 활용되고 있으며, 전체 검증 대상 컴퓨터 중에서 난독화가 적용된 악성코드가 2014년 3분기부터 분기별로 1,1,4,2의 순위를 차지하고 있음을 볼 수 있다. 또한 우리나라의 '6.25 사이버 테러' 사건에서도 악성코드에 안티-디버깅 기능이 포함된 Themida 난독화가 적용되어져 있어 탐지 및 분석의 어려움을 겪었다는 것은 이미 발표된 바 있으며[3,4], 최근 우크라이나에서 일어난 대규모 정전사건에서는 블랙에너지 악성코드를 이용한 공격이 이루어졌었으며 이 또한 악성코드에 난독화가 적용되어져있었다[5]. 이와 같이 사회기반 시설을 겨냥한 사이버 공격이 발발하고 있으며, 악성코드에 난독화가 적용되어져 있어 분석 및 대응에 어려움을 겪고 있음을 알 수 있다.

악성코드에 난독화 기법이 적용될 경우 악성코드의 시그니처가 변경되기 때문에 백신의 탐지를 회피할 수 있고, 분석가 입장에서도 난독화 기술이 해제되기 전까진 실제 악성행위를 수행하는 원본 악성코드 영역을 분석하기 어렵기 때문에 난독화 기법 이 적용된 악성코드에 대해서는 역난독화가 선적용 되어져야 분석이 가능해진다.

난독화의 적용은 주로 상용 소프트웨어 보호 도구를 통해 이루어진다. 상용 소프트웨어 보호 도구 중 Themida가 가장 강력하다고 알려져 있으며 실제 많은 기업에서 자사의 소프트웨어 보호를 위해 Themida를 사용하고 있다.

Fig.2.은 2016년 매출 순위 상위 5개의 게임 회사를 나타낸 것으로 이 회사들은 자사에서 서비스하고 있는 게임 클라이언트 소프트웨어의 보호를 위해 Themida를 활용하고 있다. 넥슨 개발자 컨퍼런스(NDC) 2016에서는 넥슨에서 개발하는 게임에 Themida가 사용되어 배포되는 바이너리의 덤프 분석이 어려워지며, Themida로 인해 파일 정보가 변경되어 분석이 어렵다는 것이 발표되어졌다[6]. 그 밖에도 카카오톡, 금융 프로그램과 같은 주요 소프트웨어도 Themida로 보호되고 있다. Themida로 보

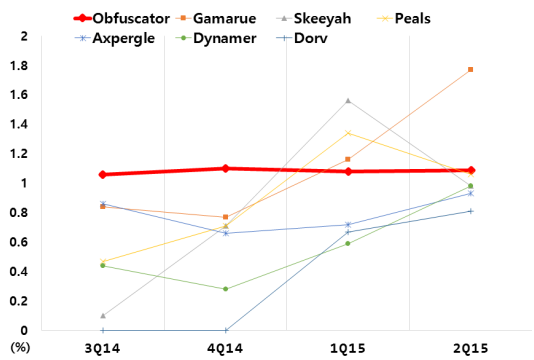


Fig. 1. Encouter rate(percent of all reporting computers)

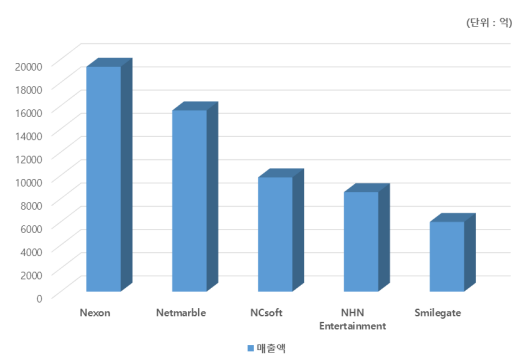


Fig. 2. Top 5 game companies in 2016

호된 프로그램을 실행시키고자 할 때 백그라운드에서 모니터링 프로그램이 돌아가고 있을 경우, 프로그램이 종료되며 Fig.3.과 같은 Themida 오류를 볼 수 있다.

Themida는 실행파일만 입력하면 다양한 난독화 기법이 적용된 실행파일을 자동으로 생성해주기 때문에 사용자가 별다른 노력 없이도 소프트웨어를 쉽게 보호할 수 있다는 장점이 있지만 악성코드 보호에도 악용되어 문제가 되고 있다. Themida는 원본 악성코드에 난독화 기법 적용 시마다 형태가 달라진 폴리모픽 악성코드를 생성하기 때문에 시그니처 탐지를 우회할 수 있게 된다. 또한 빠르게 난독화 적용이 가능하기 때문에 분석가는 현실적으로 변종 악성코드의 출현 속도를 따라갈 수 없다. 그러므로 이러한 문제들을 해결하기 위해 Themida에 대한 역난독화 기술이 필요하다.

기존 Themida 언패킹 도구에서는 Themida의 특정 코드 패턴에 기반한 역난독화를 수행한다[9],[10]. 이는 특정 버전이나 옵션에 종속적이기 때문에 일반적으로 적용하기가 어렵다는 한계점이 있다. 기존의 연구들은 우선 분석해야할 코드를 추출하거나 양을 줄이는 것에 초점이 맞춰져있을 뿐[11],[12],[13] 실행파일 형태로 역난독화하지 못한다는 한계점이 있다. 단순히 코드 추출의 형태로 역난독화할 경우, 이후 원본코드에 대한 동적 분석을 수행할 수 없는 등 분석 상의 제한이 있다.

본 논문에서는 Themida로 난독화된 실행 파일을 역난독화하여 실행파일 형태로 복원하는 일반적 분석방법론을 제안하고 자동 분석 도구를 구현 및 평가한 결과를 제시한다. 여기서 일반적 분석방법론이란 Themida의 특정 옵션이나 버전에 종속되지 않음을 의미한다. 자동 분석 도구는 동적으로 실행파일에 분석용 코드를 삽입할 수 있는 DBI(Dynamic Binary Instrumentation) 프레임워크인 Pintool을 기반으로 원본 코드 및 데이터를 추출하고, 이 원본 데이터를 기반으로 원본 실행파일을 복원한다. 본 논문의 기여도는 다음과 같다.

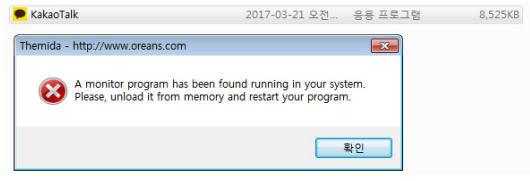


Fig. 3. Obfuscated KakaoTalk Monitoring error

- Themida의 동작 방식 분석을 통해 Themida의 버전과 적용된 보호 옵션과 상관없이 역난독화할 수 있는 일반적 분석방법론 제안함
- DBI 프레임워크인 Pintool의 활용을 통해 제안한 분석방법론을 구현하고 실제 난독화된 파일을 역난독화한 결과를 제시함
- 실행 가능한 형태로 역난독화를 수행하였으므로 역난독화된 실행파일의 동적 분석이 가능함

본 논문의 구성은 다음과 같다. 제 2장에서는 상용 소프트웨어 보호 도구인 Themida와 분석 및 역난독화에서 사용한 Pintool에 대해 알아보고, 제 3장에서는 관련 연구를 알아본다. 제 4장에서는 Themida 동작 방식을 알아보고, 제 5장에서는 역난독화 방안과 동작방식을 알아본다. 제 6장에서는 검증 결과를 서술하고, 제 7장에서 결론으로 끝을 맺는다.

II. 배경 지식

본 논문에서 분석하고자 하는 대상인 Themida와 중점적으로 활용한 분석 프레임워크인 Pintool에 대한 배경지식을 간단히 설명하고자 한다.

2.1 Themida

Themida는 소프트웨어 크랙과 역공학으로부터 소프트웨어를 보호하고자 만든 소프트웨어 보호 도구로써, 다양한 난독화 및 역공학 방지 기법의 옵션을 선택 적용 할 수 있으며 Fig.4.와 같다[7]. 또한, 옵션마다 접근 방식이 다르기 때문에 분석 시에 어려움이 따라 가장 강력한 상용 소프트웨어 보호 도구 중 하나로 알려져 있어 게임 및 상업적인 소프트웨어 보호에 주로 사용된다. Themida는 원본 코드의 제어 흐름은 유지하며 분석을 방해하기 위한 코드를 함께 실행하여 원본 코드를 보호하는 것을 목적이다. 그러므로 Themida 적용 시 파일의 크기가 늘어나고 실행 속도가 다소 저하된다는 단점을 가지지만 역공학 방해 등의 기술이 포함되어 있어 분석이 어려워진다.

Themida에는 다양한 난독화 옵션이 존재한다. IAT 제거와 패킹과 같은 기본적으로 항상 적용되는 보호 옵션이 있고, 추가적으로 선택 적용 가능한 보호 옵션들이 있다. 선택 옵션에는 디버깅이나 모니터

Fig. 4. The overview of the Themida

링 도구와 같은 분석 도구가 탐지되면 바로 프로그램 종료 및 알람 메시지를 띄워주는 anti-analysis 옵션, entry point를 찾기 어렵게 하여 역공학을 막기 위한 entrypoint obfuscation 옵션, 코드 및 리소스를암호화/압축시키는 encryption/compression 옵션, 가상 머신 환경을 탐지하는 vmware/virtualPC 옵션, 실행 파일에서 호출하는 API를 숨기는 API-wrapping 옵션 등이 존재한다. 이런 선택 옵션은 각각 독립적으로 적용 가능하며 다중 옵션 적용으로 보호도 가능하다.각 보호 옵션에 대한 역난독화를 수행할 경우 옵션 선택에 다양한 경우의 수가 있어 모든 경우에 대한 일반적 역난독화 방안을 찾는 것이 어렵다. 또한, 버전 변화로 인해 보호 옵션이 추가 또는 변경된 경우에는 이전 버전에서의 역난독화 방안이 정상적으로 동작하지 않을 가능성이 높다. 그러므로 Themida 버전과 옵션에 무관하게 역난독화할 수 있는 일반적인 분석 방법론이 필요하다.

본 논문에서는 Themida 도구를 이용하여 난독화된 실행 파일에 대한 역난독화를 수행하고자 한다. 특정 Themida 버전이나 난독화 옵션에 대한 기존 역난독화 연구 및 도구는 존재하지만 위에서 언급한 한계가 있어 일반적 분석 방법론이라 하기는 어렵다. 본 논문에서는 Themida의 분석을 통해 동작 방식을 파악하고, 버전과 선택된 보호 옵션에 무관하게 역난독화할 수 있는 방안을 제시한다.

2.2 Pintool

Pintool은 intel에서 제공하는 instrumentation 프레임워크로 JIT(Just In Time) 컴파일러를 이용하여 소스 코드의 재컴파일 없이 바이너리 레벨에서 분석용 코드를 삽입하여 실행할 수 있는 환경을 제공한다. 대상 실행파일에서 하나의 명령어가 실행될 때마다 Pintool 내부적으로 instrumentation 코드와 함께 JIT 컴파일한 코드의 실행을 통해 정보를 추출하여 분석을 도와주는 프레임워크이다[8].

Pintool의 전체적인 동작 방식은 Fig.5.와 같다. Pintool API를 이용하여 정보를 추출하고자 하는 instrumentation 코드(instrumentation.dll)를 제작한다. 제작한 코드는 Pintool과 함께 실행 시 JIT 컴파일러 역할을 하는 Pintool의 PINVM.DLL 내부에서 난독화된 실행파일과 제작된 분석용 코드와 함께 새로 컴파일 되어 코드 캐쉬에서 실행하며 정보를 추출한다.

Pintool의 가장 큰 장점은 사용자가 원하는 형태로 분석용 코드를 작성(programmable instrumentation)할 수 있다는 점이다. 즉, Pintool을 이용하면 사용자가 원하는 분석 도구를 자유롭게 제작할 수 있다. 본 논문에서는 난독화된 실행파일에 대한 분석 및 로그 추출이나 역난독화를 원본 실행파일 정보 추출 등의 다양한 목적을 위해 Pintool을 이용하였다.

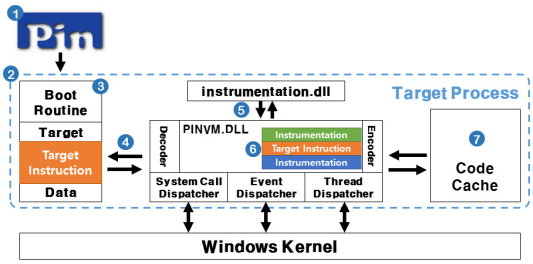


Fig. 5. The overview of the Pintool

III. 관련 연구

백신의 탐지 및 분석을 회피하기 위해 악성코드에 만연하게 난독화가 적용되면서 이에 대응하기 위한 역난독화 연구가 꾸준히 진행되어져왔다.

기존의 언패킹 도구인 Themida +

WinLicense 2.x[9], WinLicnese Ultra Unpacker 1.4[10]는 특정 어셈블리 명령어 패턴을 찾아 언패킹을 진행한다. 그러나 이는 Themida 버전과 옵션에 종속적이기 때문에 Themida의 버전 업데이트 혹은 적용된 옵션이 변경되면 언패킹이 적용되지 않을 가능성이 높아지므로 그 때마다 새로운 언패킹 로직을 구성해야 한다는 한계점이 있다. 또한 언패킹 과정을 거친 후에 해제 코드 등이 추가 존재하여 역난독화 이후에 오히려 실행 파일의 크기가 더 증가하게 된다는 제한 사항이 있다.

목성균 등[11] 연구에서는 동적 슬라이싱 기법을 사용하여 난독화된 실행파일의 전체 트레이스(실행 로그) 중에서 시스템콜 호출과 관련된 명령어만을 추출한 결과를 제시하였다. 제안 기법의 목적은 난독화된 실행파일에서 우선적으로 분석이 되어야 하는 명령어를 추출하는 것으로써 실행파일에 적용된 난독화 및 보호 옵션을 해제하는 역난독화를 수행하지 못하였고 원본 실행파일의 복원 또한 달성하지 못하였다.

Yadegari, B. 등[12] 연구에서는 난독화된 실행파일의 역난독화를 위한 일반적인 접근 방법을 제안하고 있다. 난독화된 실행파일의 입력과 출력을 정의하고 오염 흐름 분석(taint analysis)을 통해 입력에서 출력으로의 데이터 흐름과 관련된 명령어만을 추출하여 제어 흐름 그래프(CFG, control-flow graph)를 구성하는 연구를 수행하였다. 그러나 [12] 연구 역시 [11] 연구와 유사하게 분석해야 할 코드 추출에 초점이 맞춰져있으며 역난독화 및 실행파일의 복원에 대한 연구는 수행하지 않았다.

Kang, M. G. 등[13] 연구에서는 언패킹된 코드를 추출한다는 점에서 본 논문과 유사한 연구 흐름을 가진다. Kang, M. G. 등[13] 연구는 Renovo라는 자동화된 언패킹 시스템을 구현한 것으로 BitBlaze사의 동적 분석 도구인 TEMU를 기반으로 동작한다. Instrumented 코드는 메모리 쓰기 명령어에서 분기 명령어를 수행했을 때 목적지를 “dirty 영역”으로 표시한다. 그리고 명령어가 “dirty 영역”으로 정해진 메모리 영역에서 실행되면 해당 영역이 숨겨진 코드 영역이라 여기고 덤프를 떠서 저장한다. 즉, 같은 영역에서 메모리 쓰기 및 실행이 발생한다면 이를 언패킹된 코드라 정의하는 것이다. 이와 같은 방식으로 반복 실행하여 언패킹된 코드를 복원한다. 그러나 해당 연구는 언패킹된 코드를 복원한 것에 의의가 있지만 실행 가능한 형태까지 복원하지 못했다는 것이 한계점이다.

앞의 3개의 연구는 우선 분석되어야 할 코드를 추출하거나 분석해야할 코드의 양을 줄이는 것에 초점이 맞춰져 있다. 즉, 코드를 추출한 이후에는 정적 분석을 통해서만 분석이 가능하다는 한계가 있다. 그러나 본 논문에서는 실행 가능한 실행파일 형태로 역난독화하는 것이 목적이기 때문에 역난독화 이후에 동적 분석까지 적용 가능하다는 이점이 있다.

이재휘 등[14] 연구에서는 난독화된 실행파일의 API 숨김(wrapping) 옵션에 대해서 역난독화 분석 연구를 진행하였다. 해당 연구에서는 앞의 연구와 달리 존재하는 방해 기법을 제거하고 실행 가능한 형태까지 복원하였으나, API 숨김 옵션에 대한 분석만 수행한 것이므로 다중 보호 매커니즘이 적용되어 있을 경우 원본 프로그램 복원의 목적은 달성할 수 없다는 즉, 적용된 옵션에 종속적이라는 한계가 있다.

IV. Themida 동작방식

본 장에서는 먼저 Themida로 난독화된 실행파일에 대한 정적 및 동적 분석을 통해 알아낸 Themida의 동작 방식에 대해 서술한다. 다음으로는 Themida를 역난독화하기 위한 기본적인 접근 방식(naive approach)과 그 한계점에 대해 서술하고 관련 역난독화 도구를 살펴본다. 마지막으로 Themida 동작 방식에 대한 분석 결과를 기반으로 앞선 기본적 접근 방식의 한계점을 극복하기 위해 Themida를 역난독화할 수 있는 일반적 분석방법론을 제안하고 이의 장점을 제시한다.

4.1 Themida로 난독화된 실행파일 분석

난독화된 실행파일의 동작 방식을 분석하기 위해 Themida로 난독화된 실행파일과 원본 실행파일에 대하여 비교 분석을 수행하였다[15]. 먼저 두 실행파일의 PE 구조적 차이를 알아보기 위해 PE 뷰어 프로그램(PEiD)[16]를 이용하여 정적 분석을 수행하였다. 그 결과는 Fig.6.로 (a)는 원본 계산기 실행파일이며, (b)는 난독화된 계산기 실행파일이다. Fig.6.를 보면 난독화된 실행파일의 entrypoint가 변했으며, 섹션 정보를 확인한 결과, 원본 실행파일에는 없는 새로운 섹션 2개가 생성되었으며 이로 인해 실행 파일의 크기가 증가함을 확인하였다. 난독화된 계산기 실행파일을 보면 myjyhsjk, jadlotuu라는 이름을 가진 2개의 섹션이 생성되었다. 이를

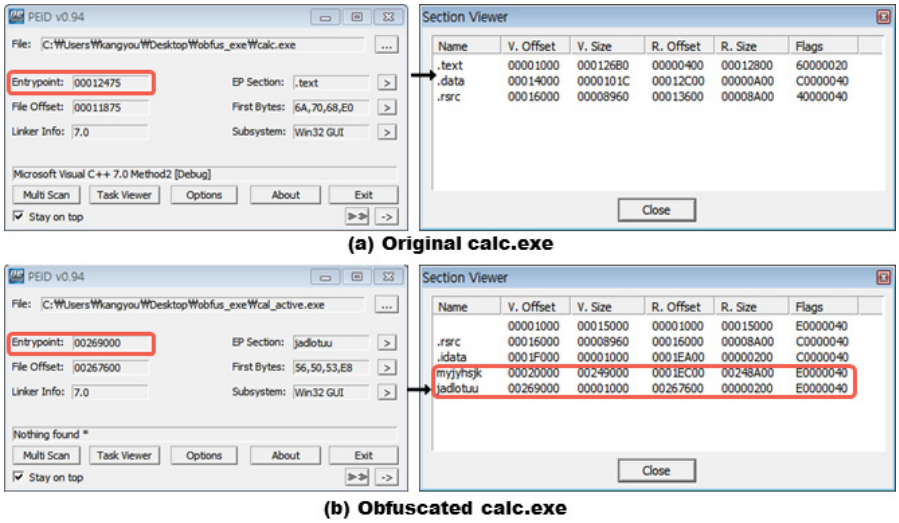


Fig. 6. Static analysis results of original calc.exe and obfuscated calc.exe

본 논문에서는 Themida1 섹션(myjyhsjk), Themida2 섹션(jadlotuu)이라 칭하도록 한다. 또한, 난독화된 실행파일은 난독화 과정을 거치면서 사용되는 API 목록을 가지는 IAT(Import Address Table)가 제거됨을 알 수 있는데, IAT에는 프로그램이 실행되면서 어떠한 dll을 필요로 하는지, 해당 dll에서 어떠한 함수를 사용하는지가 명시되어져있기 때문에 필요하다. 그러므로 역난독화를 하여 정상적인 실행 흐름을 가지려면 IAT를 복원할 수 있어야 한다.

정적 분석 결과를 통해 난독화 적용시, 새로 생긴 Themida 섹션 내에서 분석 방해 기법이 발생한다는 것과 언패킹 과정을 거쳐 원본 실행파일에 대한 코드가 복원될 것이며 IAT 복원도 실행될 것이라는 것을 추론할 수 있다.

다음으로 동적 분석으로 난독화된 실행파일의 동작 방식에 대해서 분F석하였다. 언패킹 로직, IAT 복원, 난독화 로직 분석을 위해서 Pintool 기반 자체 분석 도구를 제작하여 난독화된 프로그램을 다각도로 분석하였다. Table 1.은 자체 분석 도구에 포함된 기능과 그 기능의 구현 목적을 정리한 것이다. 또한 이를 위해 사용한 Pintool API와 그 기능도 함께 정리하였다.

Fig.7.은 동적 분석을 통해 알아낸 난독화된 실행파일의 전체적인 동작방식을 나타낸 것이다. 기본적으로 난독화된 실행파일은 Themida2 섹션을 시작으로 프로그램이 실행되며, Themida 섹션의 경우 분석을 어렵게 하기 위한 예외 처리 루틴과 분석 방

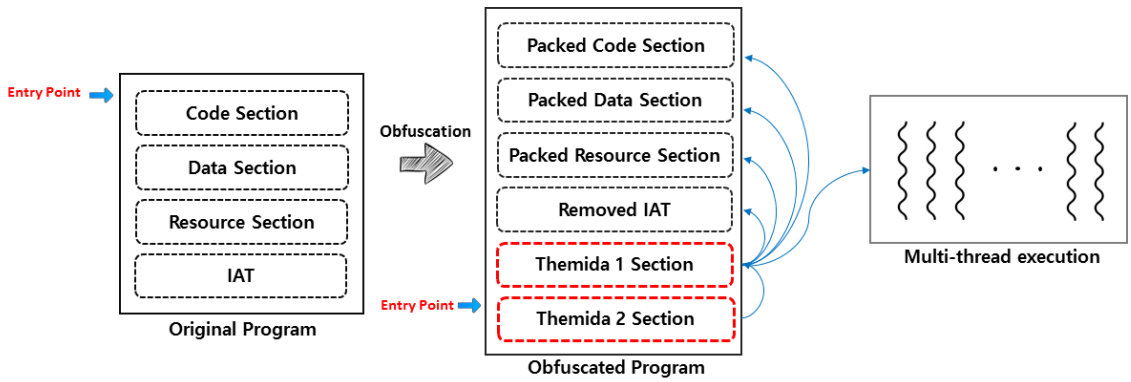


Fig. 7. Dynamic analysis results of obfuscated calc.exe

Table 1. Implementation of the functions included in the our analysis tool

Functionality	Obtained value	Objective	Pintool API used for analysis	API description
Memory write instruction monitoring	Instruction address, destination address, value to write	To analyze the logic to unpack original code and data	INS_IsMemoryWrite(inst.)	Returns true if the instruction writes memory
			INS_Address(inst.)	Returns address of the instruction
Dll loading monitoring	Full path, image base, size, export table of dll	To analyze the logic to restore IAT	IMG_AddInstrumentFunction (call back)	Register the call back to catch the loading of an dll
			IMG_StartAddress(dll)	Returns the pointer to the dll
			IMG_SizeMapped(dll)	Returns the size of the dll
			IMG_Name(dll)	Returns name of the dll
			IMG_RegsymHead(dll)	Returns regular symbols in the dll
Thread execution monitoring	Thread ID, initial context for new thread	To analyze the protection logic corresponding to the options applied through Themida	PIN_AddThreadStartFunction (call back)	Register the call back that is called when a thread starts executing in the application
			PIN_ThreadId()	Get identifier of the current thread ID in Pin

해 루틴이 많이 포함되어 있어 난독화된 실행파일에 많은 분석 시간이 소요된다. 각 동작 순서에 대한 설명은 아래와 같다.

① Themida1 섹션 언패킹: Themida2 섹션에서는 Themida1 섹션을 언패킹하기 위해 Themida1 섹션에 메모리 쓰기 수행을 하고 언패킹이 완료된 이후에는 Themida1 섹션으로 제어권이 넘어가고 Themida2 섹션은 실행되지 않는다.

② 원본 코드 및 데이터 언패킹: Themida1 섹션에서는 원본 실행파일의 코드, 데이터, 리소스와 같은 원본 프로그램 영역 전체를 언패킹하기 위해 원본 영역에 메모리 쓰기를 수행한다. 이 단계가 끝난 이후에는 원본 영역이 언패킹 상태이며, 거의 모든 원본 영역이 복원된다.

③ 동적 dll 로딩: 원본 실행파일에서 사용되는 dll 정보를 포함한 IAT는 난독화된 실행파일에서 사라지므로 Themida1 섹션에서는 동적으로 dll을 로딩하여 사용한다.

④ 멀티쓰레드 실행: Themida1 섹션에서 ②,③의 작업을 수행한 이후에는 메인 쓰레드의 제어권이 원본 코드 영역으로 넘어가게 되며 이후에는 다시 Themida 섹션으로 제어권이 넘어가지 않는다. 즉, 메인 쓰레드는 마치 원본 프로그램을 직접 실행시킨 것과 같이 동작하며 적용된 보호 옵션과는 독립적으로 실행된다. 대신 Themida 섹션에서는 제어권이 원본 코드 영역으로 넘어가기 이전에 다수의 쓰레드를 생성하여 각 쓰레드 별로 적용된 보호 옵션의 기능을 할당하고 주기적으로 실행시킨다. 예를 들어, 안티-디버깅 옵션을 적용할 경우 하나의 쓰레드에서 OllyDbg같은 프로세스가 현재 실행 중인지를 주기적으로 모니터링한다.

4.2 Thmida 분석에 대한 기본적 접근 방식(naive approach)와 그 한계점

본 논문에서는 Themida로 난독화된 실행파일에 대한 역난독화를 목적으로 한다. 역난독화를 위해서는 기본적으로 순차적 분석을 수행하며 분석 방해 루틴이나 난독화 및 언패킹 로직을 마주칠 때마다 이를 해제하여 원본 프로그램을 복원한다. 실제 이런 기본적 접근 방식(naive approach)으로 분석을 수행하기 위해 Themida2 섹션의 언패킹 루틴부터 시작해서 Themida1 섹션의 언패킹 루틴, 동적 dll 로딩 루틴, 각 쓰레드 실행 루틴 등의 실행 로그를 기록하여 순차적으로 분석하였다. 그러나 이런 접근 방식은 다음과 같은 제약사항에 부딪히게 된다.

먼저 난독화된 프로그램의 실행 로그를 남기는 것 자체에서 문제가 발생한다. OllyDbg 상에서는 계속 오류가 발생하여 중간에 중단이 되며, OllyDbg 상의 로그 기능은 레지스터의 값, 간접 점프문의 주소와 같은 메모리 값의 정보는 제공하지 않기 때문에 단순한 실행 명령어 정도 밖에 알 수가 없다. 또한 로그 생성에 너무 많은 시간이 소요된다는 문제점이 있었다. 그래서 실행 명령어 뿐만 아니라 레지스터 값, 메모리값 등을 모두 추출할 수 있는 Pintool 기반자체 로그 생성 도구를 제작하였다. 이는 앞서 말한 로그 생성 과정의 한계점을 극복할 수는 있었으나 하나의 프로그램 실행 로그가 무려 10기가에 달하여, 산술적으로 하나의 보호 옵션당 10기가씩 20개 이상의 보호 옵션을 분석해야 하므로 실질적으로 이를 모두 분석하고 역난독화 로직을 설계하기에는 현실적인 어려움이 따른다. 또한 위에서 언급한 방법을 통해 역난독화 로직을 설계하더라도, Themida 버전의 업데이트나 새 option이 추가 혹은 변경이 생길 때마다 재분석하여 역난독화 모듈을 재설계해야 한다는 한계가 있다.

4.3 역난독화 방안 제안

4.2절에서 언급한 순차적 분석 방식 혹은 언패커와 같은 접근 방식을 통한 분석은 일반적 분석 방법론이라고 보기에는 많은 제약사항과 한계점이 있다. 따라서 4.1절에서 서술한 Themida의 동작 방식을 통해 Themida가 패커의 특징을 가진다는 점에 착안하여 패커에 대한 분석 방식으로 접근하였다. 즉, 분석 방해 로직이나 언패킹 로직에 상관없이 원본 코드의 데이터가 복원되고 실행되는 시점을 탐색하는 방식으로 분석 방법을 전환하였다. 이는 Themida 섹션에서 어떠한 행위를 하든 원본 코드 섹션으로 넘어간 이후부터는 다시 Themida 섹션으로 제어권이 넘어가는 일이 없으므로 원본 코드 섹션이 실행된 시점에서는 코드 및 데이터가 모두 실행 가능한 형태일 것이라는 가정을 따른다.

해당 접근 방식은 역난독화를 위해 Themida 섹션의 기능을 따로 분석할 필요가 없으며, 쓰레드로 실행되는 각각의 보호 옵션 또한 분석할 필요가 없다는 장점이 따른다. 언패킹이 완료되고 원본 코드 영역으로 제어권이 넘어가는 시점만 정확하게 찾아내고

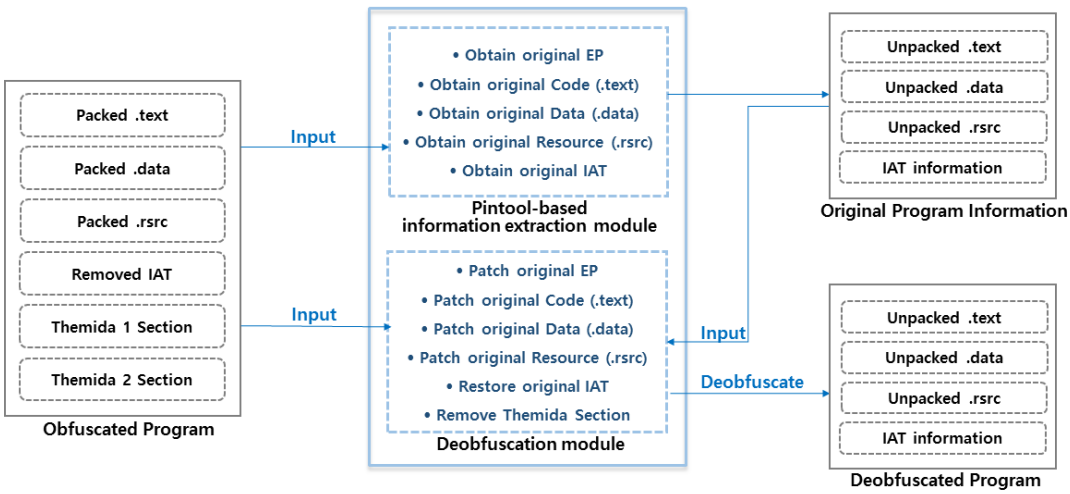


Fig. 8. The overview of the Automated de-obfuscated tool

원본 코드 및 데이터만 획득할 수 있다면 이를 토대로 바이너리 패치를 하면 되기 때문에 Themida 버전이나 적용된 보호 옵션에 관계없이 역난독화가 가능하다.

V. Themida 자동 역난독화 도구 동작 방식

본 장에서는 4.3절에서 제안한 역난독화 방안에 근거하여 Themida로 난독화된 실행파일을 자동으로 역난독화하는 분석 도구의 구현 결과를 제시한다. Themida는 기본적으로 packer의 방식으로 동작하기 때문에 Themida 섹션에서 원본 코드 및 데이터를 언패킹한 이후에는 원본 코드 영역으로 메인 쓰레드의 제어권이 완전히 넘어간다. 즉, 원본 코드 영역으로 제어권이 넘어가는 시점에서는 원본 코드 및 데이터가 완전한 실행파일 형태로 복원된다는 것을 알 수 있다. 그러므로 제어권이 넘어가는 정확한 시점에 원본 코드 및 데이터를 추출하고, 추출한 원본 데이터를 난독화된 실행파일에 패치하여 실행 가능한 형태로 출력하는 자동화된 역난독화 도구를 제작하였다. 본 도구의 자동화된 역난독화 프로세스는 Fig. 8.과 같은 동작 방식을 따르며, 도구의 주요 구성 요소와 그 기능은 다음과 같다.

- Pintool 기반 정보 추출 모듈: 정보 추출 모듈에서는 난독화된 실행파일의 현재 제어권을 모니d터링하여 제어권이 원본 코드 영역으로 넘어가는 시점에 원본 Entrypoint, 코드 및 데이터 섹션의 정보, IAT를 획득하고 이를 파일로 저장한다.
- 실행파일 역난독화 모듈: 역난독화 모듈에서는 정보 추출 모듈을 통해 획득한 원본 실행파일의 정보를 난독화된 실행파일에 바이너리 패치하는 역할을 수행한다. ① PE 헤더에서 Entrypoint 수정, ② 원본 코드 및 데이터 섹션 패치, ③ 원본 IAT 정보 복원, ④ 마지막으로 필요 없는 Themida 섹션을 완전히 제거하여 역난독화된 실행파일을 출력한다.

5.1 Pintool 기반 정보 추출 모듈

Pintool 기반 정보 추출 모듈에서는 난독화된 실행 파일을 입력으로 하여 역난독화 하기 위한 원본 실행파일 정보를 추출한다. 이를 위해서는 난독화된 실행파일의 제어권이 원본 코드 영역으로 넘어가는 시점을 정확히 예측해야 하므로 정보 추출 모듈에서는 현재 제어권이 어느 섹션에 있는지를 모니터링한다. 정보 추출 모듈의 동작 방식은 Fig.9.과 같으며 전체적인 과정은 다음과 같다.

① 실행파일의 섹션 정보 추출: 대상 실행파일을 실행하기 전 해당 실행파일의 모든 섹션 정보(시작 주소 및 크기)를 추출한다. 이 정보는 이 후 제어권을 모니터링할 때 제어권의 소유 섹션을 매칭할 때 사용된다. 섹션 정보 추출을 위해 실행파일의 PE 구조를 직접 파싱하여 정보를 추출하는 것도 가능하지만, Pintool에서는 섹션 정보 추출을 위한 SEC_Address, SEC_Name,

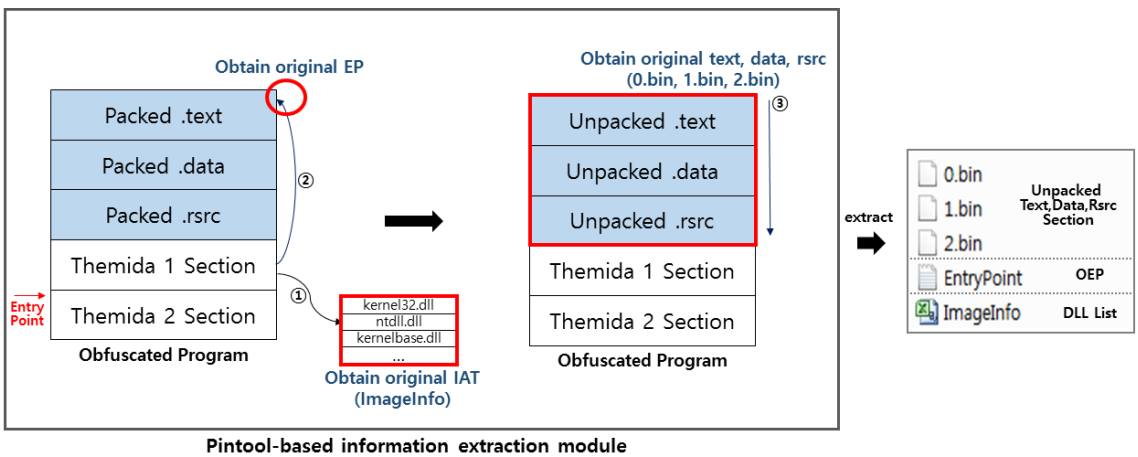


Fig. 9. Pintool-based information extraction module

SEC_Size, SEC_Type 등의 유용한 API를 제공한다.

② DLL 로딩 모니터링: Pintool에서는 DLL을 동적 로딩할 때마다 분석용 instrumentation 코드를 삽입할 수 있는 API를 제공한다(IMG_AddInstrumentFunction). 이를 이용하면 IAT를 복원하는 시점에 원본 실행파일에서 사용하는 DLL 정보를 획득할 수 있다.

③ 제어권 모니터링: 필요한 섹션 정보 추출 후에는 대상 실행파일을 실행하여 현재 제어권이 어느 섹션에 있는지를 모니터링한다. 원본 코드 영역으로 제어권이 넘어가는 시점을 정확히 특정하기 위해 현재 명령어 실행 시마다 해당 명령어의 주소(instruction pointer)를 추출하고 앞서 추출한 섹션 정보와 매칭하여 현재 제어권이 어느 섹션에 있는지를 결정한다. 제어권이 처음으로 원본 코드 영역으로 넘어가는 순간 실행파일의 실행을 정지하고 ④로 넘어간다. 이 때 명령어의 주소가 원본 실행파일의 OEP가 된다.

④ 원본 코드 및 데이터 덤프: 제어권이 원본 코드 영역으로 넘어오는 시점에는 원본 코드 및 데이터가 모두 실행 가능한 형태로 복원이 되었다는 것을 의미하므로 원본 코드와 데이터 영역은 메모리 덤프를 통해 파일 형태로 저장한다. OEP, 원본 코드 및 데이터 정보, DLL 정보 등 역난독화에 필요한 모든 정보를 획득하였으므로 대상 실행파일을 종료한다.

해당 정보 추출 모듈은 Themida의 언패킹 로직이나 보호 옵션에 대한 로직을 전혀 분석하지 않고 영향을 받지도 않는다. 즉, Themida의 버전업을 통해 언패킹 및 보호 옵션에 대한 로직이 변경 및 향상되더라도 그리고 난독화된 실행파일에 적용된 보호 옵션이 어떤 조합인지에 상관없이 원본 실행파일 정보 추출의 일반화가 가능하다는 것을 의미한다.

5.2 역난독화 모듈

역난독화 모듈에서는 정보 추출 모듈을 통해 얻은 원본 실행파일 정보를 기반으로 난독화된 실팽파일을 실행 가능한 형태(working executable)로 바이너리 패치한다. Themida를 통해 실행파일에 난독화를 적용하면 원본 IAT를 제거하고 다른 영역에 IAT를 생성, 섹션 병합 및 추가 등 원본 실행파일과 구조적으로 달라진 형태의 실행파일을 생성하기 때문에 완전히 원본 실행파일과 동일한 형태로 실행파일을 복원하는 것은 불가능하다. 그러나 원본 실행파일과 완전히 동일하진 않더라도 실행 가능한 형태로 복원하면 이후 원본 실행파일에 대한 분석을 수행할 때 동적 분석을 적용할 수 있기 때문에 이전 연구

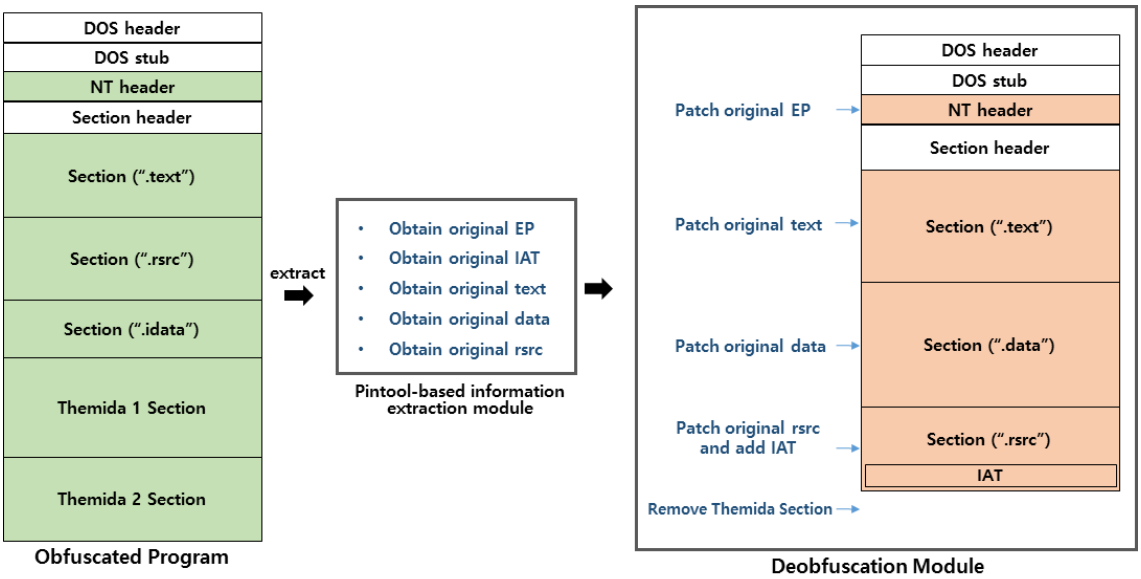


Fig. 10. De-obfuscation module

[11],[12],[13]에서와 같이 단순히 정적 분석이 가능한 형태(ex: 로그 추출)로 원본 코드 및 데이터 등을 추출해내는 것보다 분석 상의 많은 이점을 가질 수 있다.

다음 Fig.10.은 역난독화 모듈의 동작 절차를 나타낸 것이다.

① PE 헤더 수정: PE 헤더에는 EP, 섹션 크기 및 개수 등 실행파일 구조에 대한 전반적인 정보를 담고 있다. 역난독화 모듈에서는 정보 추출 모듈에서 획득한 OEP로 난독화된 실행파일의 EP를 변경한다. 또한 역난독화 이후에는 아무런 기능을 하지 않는 Themida 섹션을 제거하기 위해 Themida 섹션 헤더 정보를 삭제한다.

② 원본 코드 및 데이터 영역 패치: 정보 추출 모듈에서 획득한 원본 코드 및 데이터를 난독화된 실행파일의 코드 및 데이터 영역에 패치한다. 섹션 압축 옵션을 제외한 일반적인 보호 옵션을 적용한 경우에는 그대로 패치를 수행하면 되지만, 압축 옵션을 적용한 경우에는 난독화된 실행파일의 섹션 크기가 획득한 원본 섹션 정보보다 작아져 있는 상태이기 때문에 이를 보정해주기 위한 추가 작업이 필요하다.

③ IAT 복원: Themida로 난독화 시 원본 실행파일의 IAT와는 다른 위치에 새로운 IAT가 생성되고 원본 실행파일에서 사용하는 DLL 정보는 지워진 상태이기 때문에 정확한 원본 IAT 위치에 복원시키는 것은 불가능하다. 그러나 실행 관점에서 IAT의 위치는 영향을 미치지 않기 때문에 난독화된 실행 파일의 IAT 영역에 정보 추출 모듈에서 획득한 DLL 정보를 삽입하더라도 실행은 문제가 없다.

④ Themida 섹션 제거: 앞선 ①~③ 과정만 수행하더라도 역난독화된 실행파일은 정상적으로 실행이 된다. 원본 코드 섹션에서 Themida 섹션으로 제어권이 넘어가는 상황이 발생하지 않기 때문에 Themida 섹션을 제거하지 않더라도 무방하다. 그러나 Themida 섹션은 2 MB 정도의 상당히 큰 크기를 가지고 있기 때문에 제거하는 것이 파일 크기 복원 측면에서 이점이 있다. Themida 섹션을 제거할 경우 원본 실행파일의 크기에 가깝게 복원이 가능하다.

VI. 검증 결과

본 논문에서는 역난독화 구현 및 적용 후 역난독화된 실행 파일에 대하여 원본 실행 파일과의 유사도를 명령어 시퀀스와 제어 흐름 그래프 유사도의 두 가지 측면에서 검증을 수행하였다. 검증한 Themida의 버전은 v2.3.2.0(2014.9.16), v2.4.5.0(2016.09.19)이다.

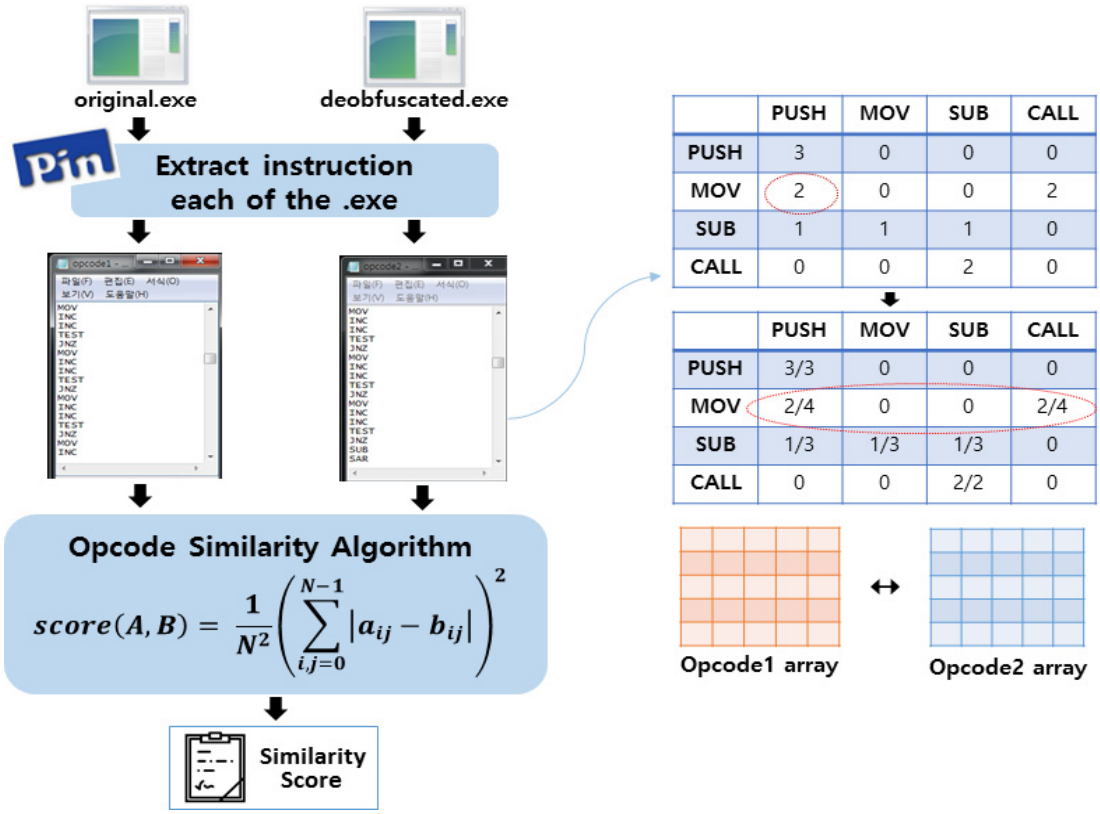
먼저 명령어 시퀀스 유사도는 Pintool 기반 자체 명령어 추출 도구를 구현하여 원본 실행파일과 역난독화된 실행파일의 명령어 시퀀스를 추출한 뒤 비교하였다. 유사도 알고리즘은 Runwal, N. 등[17] 연구에 기반하여 구현하였으며 이의 대략적은 흐름은 Fig.11.으로 다음과 같다. Runwal, N. 등[17]의 연구에서는 실행파일을 실제 실행하여 얻은 명령어 트레이스 정보에서 연달아 실행되는 명령어 빈도가 얼마나 유사한 경향성을 가지는지는 비교한다. 그러므로 실제 실행파일 형태로 역난독화한 결과가 원본 프로그램과 얼마나 유사한 명령어 시퀀스로 실행되는지를 비교하기에 적합하다고 할 수 있다. Runwal, N. 등[18]의 연구는 메타모픽 악성코드 탐지를 위해 처음 제안되었으나, 이후 소프트웨어 무단 복제(software piracy) 탐지에도 좋은 결과를 낸 연구 결과가 있다[18].

① 원본 프로그램(Opcode1.exe)과 역난독화한 프로그램(Opcode2.exe)에 대하여 Pintool 기반 명령어 추출 도구를 이용하여 실행 순서에 따라 명령어 시퀀스를 추출한 후 각각 텍스트 파일 형태로 출력한다.

② 추출된 명령어 시퀀스 텍스트 파일을 통해 각각의 실행파일에 대한 연속된 명령어 쌍의 빈도수를 행렬로 작성하고, 행렬의 각 원소 값을 해당 행의 합으로 나눠 원본 실행파일과 역난독화된 실행파일에 대한 각각의 행렬을 생성한다. 최종 생성된 행렬은 연속된 명령어의 발생 비율을 나타낸다.

③ 완성된 두 행렬을 비교하여 원본 및 역난독화 실행파일의 명령어 시퀀스가 얼마나 유사한지 유사도를 측정한다. 유사도 점수는 모든 행렬 성분의 차를 더한 값으로 결정되며 두 명령어 시퀀스 행렬이 비슷할수록 유사도 점수가 0에 가까워진다.

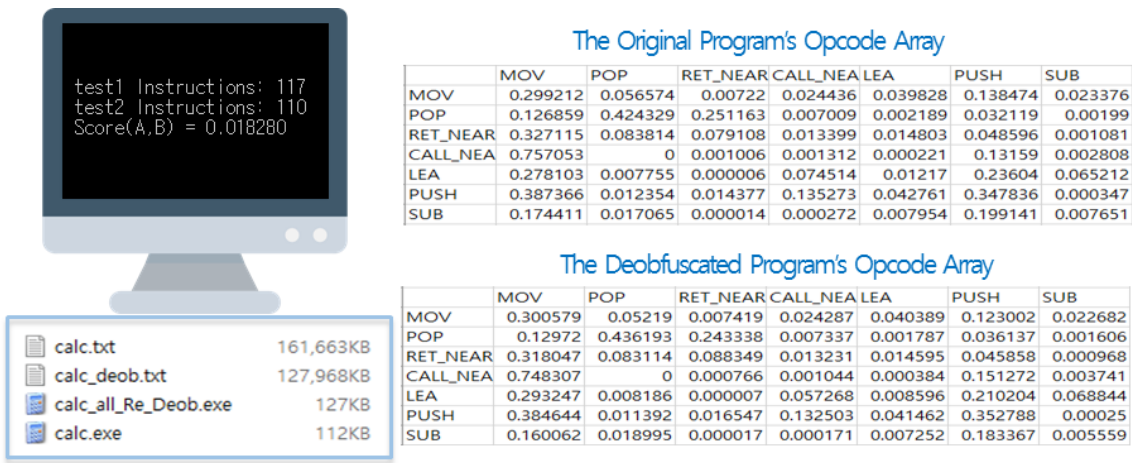
본 논문에서는 Fig.12.과 같이 계산기 실행 파일에 대하여 원본 실행파일과 Themida를 통해 난독



	PUSH	MOV	SUB	CALL
PUSH	3	0	0	0
MOV	2	0	0	2
SUB	1	1	1	0
CALL	0	0	2	0

	PUSH	MOV	SUB	CALL
PUSH	3/3	0	0	0
MOV	2/4	0	0	2/4
SUB	1/3	1/3	1/3	0
CALL	0	0	2/2	0

Fig. 11. Process of opcode similarity algorithm



The Original Program's Opcode Array

	MOV	POP	RET_NEAR	CALL_NEA	LEA	PUSH	SUB
MOV	0.299212	0.056574	0.00722	0.024436	0.039828	0.138474	0.023376
POP	0.126859	0.424329	0.251163	0.007009	0.002189	0.032119	0.00199
RET_NEAR	0.327115	0.083814	0.079108	0.013399	0.014803	0.048596	0.001081
CALL_NEA	0.757053	0	0.001006	0.001312	0.000221	0.13159	0.002808
LEA	0.278103	0.007755	0.000006	0.074514	0.01217	0.23604	0.065212
PUSH	0.387366	0.012354	0.014377	0.135273	0.042761	0.347836	0.000347
SUB	0.174411	0.017065	0.000014	0.000272	0.007954	0.199141	0.007651

The Deobfuscated Program's Opcode Array

	MOV	POP	RET_NEAR	CALL_NEA	LEA	PUSH	SUB
MOV	0.300579	0.05219	0.007419	0.024287	0.040389	0.123002	0.022682
POP	0.12972	0.436193	0.243338	0.007337	0.001787	0.036137	0.001606
RET_NEAR	0.318047	0.083114	0.088349	0.013231	0.014595	0.045858	0.000968
CALL_NEA	0.748307	0	0.000766	0.001044	0.000384	0.151272	0.003741
LEA	0.293247	0.008186	0.000007	0.057268	0.008596	0.210204	0.068844
PUSH	0.384644	0.011392	0.016547	0.132503	0.041462	0.352788	0.00025
SUB	0.160062	0.018995	0.000017	0.000171	0.007252	0.183367	0.005559

Fig. 12. Verification result of opcode similarity

화 기법이 적용된 프로그램에 대해 역난독화한 실행 파일의 유사도 결과를 제시한다. 그 결과로 Table 2.는 원본 실행파일과 역난독화된 실행파일의 유사도 점수로, calculator(계산기), cmd(명령 프롬프트), verifier(드라이버 확인 프로그램), winmine(지뢰찾기) 프로그램에 대한 역난독화 수행 후 유사도 점수 계산 결과를 보인다.

두 번째 방식으로는 실행파일의 코드와 제어 흐름

Table 2. Opcode similarity score of files

file name	opcode similarity score
calculator	0.018280
cmd	0.010029
winmine	0.001255
verifier	0.000298

그래프를 확인해 보았다. IDA Pro 6.6 버전으로 확인해 본 IDA View 결과 Fig.13.과 같이 실행파일의 유사한 제어 흐름 그래프를 확인할 수 있었다. (a)는 원본 계산기 실행파일에 대한 제어흐름 그래프이며, (b)는 난독화된 실행파일, (c)는 역난독화된 실행파일의 제어흐름 그래프이다. (b)는 난독화된 실행파일이므로 packing 되어있어, 프로그램의 흐름이 보이지 않았으며, 이 실행파일에 대해 역난독화를 수행한 후에 실행파일은 다시 원본 실행파일과 유사하게 복원되었음을 확인할 수 있었다.

다음으로, 원본 파일과 역난독화 파일의 코드를 확인해본 결과 Fig.14.와 같이 실행파일의 코드 복

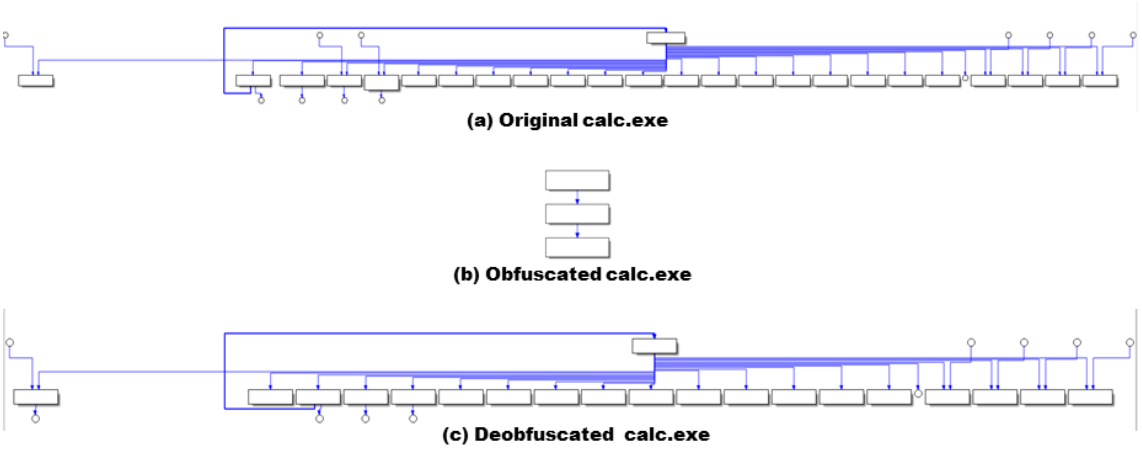


Fig. 13. Control flow graph of original calc.exe and de-obfuscated calc.exe

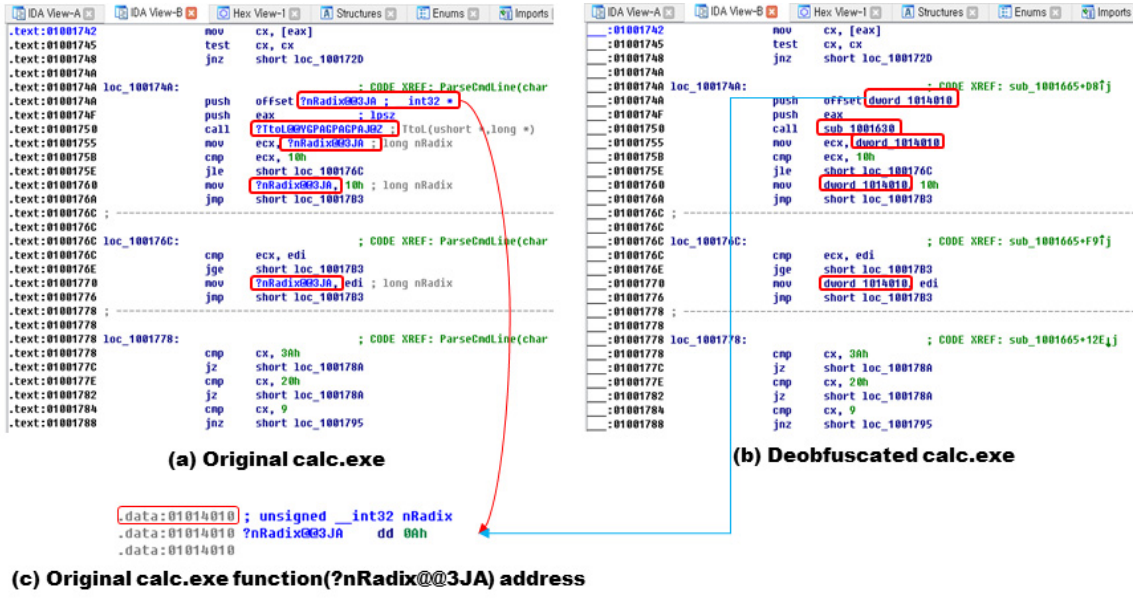


Fig. 14. Control flow graph of original calc.exe and de-obfuscated calc.exe

원이 되었음을 볼 수 있다. 그러나 난독화를 적용하면서 함수명과 같은 심볼(Symbol) 정보가 제거되어 코드에서 표기 상의 차이만 있음을 확인하였다. 실제 Fig.14.으로 확인해보면, 원본 실행파일인 (a)에서는 호출 주소가 함수명으로 파싱되어 나타나지만, 역난독화된 실행파일인 (b)에서는 호출 주소가 함수명인지를 IDA가 식별하지 못하기 때문에 단순히 주소로 나타나있다. 이는 Fig.14.의 (c)에서 처럼 원본 실행파일의 심볼테이블에서 해당 함수의 위치를 확인해 보면 역난독화된 실행파일과 동일한 주소를 가리키고 있음을 확인할 수 있다.

본 논문에서는 이와 같이 구현한 도구를 통해 역난독화를 수행해보았고, 수행 결과 및 검증을 보였다. 계산기를 기반으로 수행한 결과를 보였으며, 다른 실행 파일에 대해서도 역난독화 가능함을 보이기 위해 계산기 실행파일 이외에 cmd, verifier, winmine에 대하여 Fig.15., Fig.16., Fig.17.의 결과를 제시한다.

또한 역난독화 전 후 파일의 사이즈를 Table 3.과 같이 확인해본 결과, 원본 실행파일과 역난독화된 파일의 크기가 유사한 크기로 복구되었음을 확인할 수 있었다. 대부분의 기존 연구[9-12]

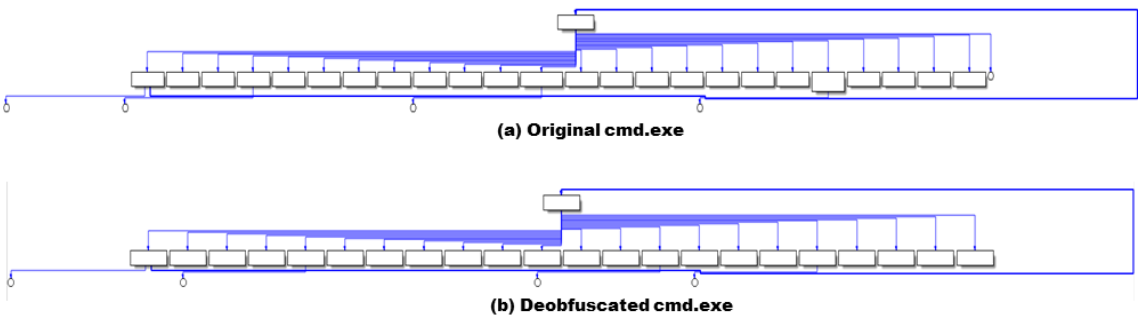


Fig. 15. Control flow graph of original cmd.exe and de-obfuscated cmd.exe

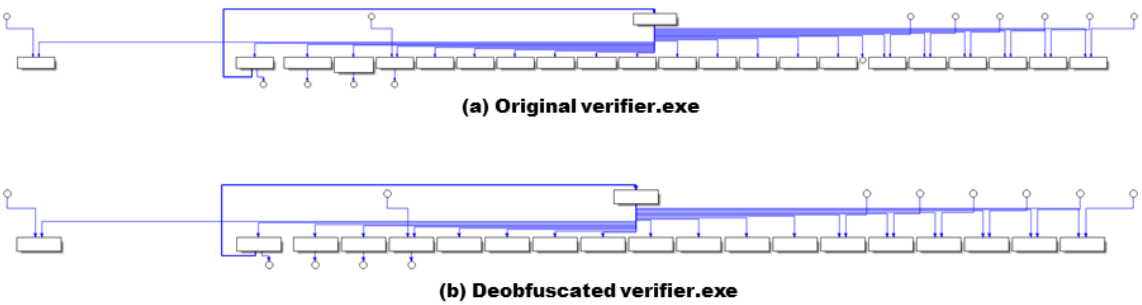


Fig. 16. Control flow graph of original verifier.exe and de-obfuscated verifier.exe

Fig. 17. Control flow graph of original winmine.exe and de-obfuscated winmine.exe

Table 3. Comparison of file's size

File name	File type	File size
calculator	Original	112 KB
	Obfuscated	2,451 KB
	Deobfuscated	127 KB
cmd	Original	474 KB
	Obfuscated	2,452 KB
	Deobfuscated	485 KB
verifier	Original	96 KB
	Obfuscated	2,436KB
	Deobfuscated	110 KB
winmine	Original	117 KB
	Obfuscated	1,995KB
	Deobfuscated	129 KB

에서는 실행 파일 형태로 복원하지 못하였을 뿐만 아니라, 추출한 코드의 사이즈도 증가하였다. [13] 연구에서는 68KB 사이즈의 난독화된 실행파일이 역난독화 되었을 때 1MB 이상으로 파일 사이즈가 증가하였음을 보였다. [14] 연구에서는 본 논문과 비슷한 결과의 역난독화 파일 사이즈를 출력하였으나 API 숨김 옵션에 대해서만 연구를 진행하여 기존 연구가 가지는 한계점을 그대로 가지고 있다.

VII. 기존 연구와의 비교

본 논문에서는 Themida로 난독화된 실행파일에 대해서 분석을 통해 Themida의 동작 방식을 이해하고, 이를 동적 분석 도구 pintool을 통해 필요한 정보를 추출하여 실행 파일 형태로 복원하였다. 기존의 역난독화 도구와 연구의 역난독화 복구 수준 및 연구들의 차이점을 간단히 Table 4.를 통해 알아볼 수 있다.

Table 4. The comparison of related work and restoration levels

	Restoration levels				Description	Restore levels and limitation
	Extract trace	Extract original code	Extract original data	Restore to working executable		
Themida + WinLicense 2.x[8]	O	O	O	O	• Find specific instruction patterns and unpack them. • Themida sections for protection remain after unpacking	• Deobfuscate target as an working executable • Depend on Themida version and protect option applied • Increase output file size due to Themida sections remained
WinLicnese Ultra Unpacker 1.4[9]	O	O	O	O		
[11]	O				• Extracts only the instructions related to system call among the entire trace using dynamic slicing method	• The goal of these researches is to extract or simplify the specific instructions to be analyzed • Failed to extract data and restore to working executable forms • There approaches are not suitable for extracting only original code section in Themida
[12]	O				• Propose semantic-preserving trace simplification techniques and apply them into whole trace to extract simplified code	
[13]		O			• Define unpacked code as the region at which the target performs memory write and then is executed • Find unpacked code regions using taint analysis and dump them	• Extract only original code section • Failed to extract data and restore to working executable forms

	Restoration levels				Description	Restore levels and limitation
	Extract trace	Extract original code	Extract original data	Restore to working executable		
[14]	O	O	O	O	• Perform their research based on existing researches and unpacking tools • Perform deobfuscation as an working executable for the API wrapping option only	• Inherits limitations of existing researches and unpacking tools • Only deobfuscate certain option
Ours	O	O	O	O	• Propose a general deobfuscation method based on Themida's operation that it is similar with other packers to protect original code and data regions	• Deobfuscate target as an working executable • Our method is independent of Themida's version and protection options applied

VIII. 결 론

본 논문에서는 Themida로 난독화된 실행파일에 대한 동작 방식을 분석하고, 난독화된 실행파일을 역난독화할 수 있는 일반적 분석방법론을 제안하고 이를 자동 분석 도구로 구현하였다. 자동 분석 도구는 동적 분석 도구인 Pintool을 기반으로 원본 코드 및 데이터를 추출하고, 추출한 원본 데이터를 기반으로 원본 실행파일을 복원하여 검증 결과를 제시하였다.

본 논문에서는 Themida의 동작 방식(패킹)에에 기반하여 원본 프로그램 정보가 전부 복원되고 원본 코드 영역으로 제어권이 넘어가는 정확한 시점에 이를 저장하였다. 그러므로 제안된 방안은 Themida가 아닌 다른 난독화 도구 등에서 사용하는 난독화 방식에는 동작하지 않을 수 있다는 한계점이 있다. 예를 들어, 난독화된 코드가 한꺼번에 역난독화되지 않고 코드가 실행되면서 일부 필요한 코드만 역난독화하는 경우에는 본 논문에서 제안한 역난독화 방안이 적용되지 않을 가능성이 높다.

다양한 형태의 악성코드의 급격한 증가로 인해 이를 분석 및 대응하기 위해서는 많은 시간과 인력이 필요하게 되었다. 본 논문에서 구현한 역난독화 자동 분석 도구는 악성코드 분석가의 분석 진입 장벽을 낮춰주며, 도구를 통해 코드의 내부 로직을 파악 가능하게 함으로써 분석가의 효율을 높여준다. 또한 Themida로 난독화된 실행파일에 대한 일반적 분석 방법론을 제안한 첫 번째 연구 결과라는 점에서 의의가 있다.

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] Microsoft, Microsoft Security Intelligence Report Volume 20, (also see https://www.microsoft.com/security/sir/default.aspx)

[3] NSHC, 6.25 Cyber terror Analysis Report (also see training.nshc.net/KOR/Document/isac/)

[4] FireEye, FireEye Analysis Report:6.25 Cyber Attack (also see http://www.concert.or.kr/issue/qna_view.php?wr_id=18542&page=2)

[5] AhnLab, AhnLab Analysis Report: Trend of Inftastructure Attacks (also see http://www.ahnlab.com/kr/site/securityinfo/secunews/secuNewsView.do?seq=25074)

[6] Sang-Gi Kim, "Game-specific dump analysis system," Nexon Developers Conference(also see http://ndcreplay.nexon.com/NDC2016/sessions/NDC2016_0049.html#c=NDC2016&t%5B%5D=%ED%94%8

4%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D6)

〔7〕 Oreans Technologies, Themida, Advanced Windows Software Protection System, Revision 2.4, May 2016(also see http://www.oreans.com/themida.php)

〔8〕 C.K.Luk, R.Cohn, R.Muth, H.Patil, A.Klauser, G.Lowney, S.Wallace, V.J.Reddi, and K.Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," Acm sigplan notices, Vol. 40 , No. 6, pp. 190-200, 2005

〔9〕 LCF-AT, "Themida+WinLicense 2.x (Unpacking)," Jul. 2013 (also see https://tuts4you.com/download.php?view.3495)

〔10〕 LCF-AT, "Themida+WinLicense 2.x (Ultra Unpacker v1.4)", Jan. 2014 (also see http://tuts4you.com/download.php?view.3526)

〔11〕 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

〔12〕 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

〔13〕 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

〔14〕 Jae-hwi Lee, Jae-hyeok Han, Min-wook 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

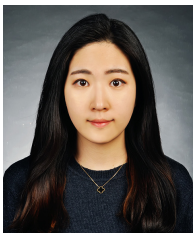
〔15〕 Jae-Hyuk Suk, Sung-hoon Kim and Dong-Hoon Lee, "Analysis of Virtualization Obfuscated Executable Files and Implementation of Automatic Analysis Tool," Journal of the Korea Institute of Information Security & Cryptology, 23(4), pp. 709-720, August, 2013

〔16〕 PEiD, 2009 (also see http://www.peid.info)

〔17〕 N.Runwal, R.M.Low and M.Stamp, "Opcode graph similarity and metamorphic detection," Journal in Computer Virology, No. 8, pp. 37-52, 2012

〔18〕 A.Khalilian, H.Golbaghi, A.Nourazar, H.Haghighi and M.V.Asl,"MetaSPD: Metamorphic Analysis for Automatic Software Piracy Detection," Computer and Knowledge Engineering (ICCKE), 2016 6th International Conference on IEEE, pp. 123-128, 2016

〈저자소개〉



강 유 진 (You-jin Kang) 학생회원
2016년 2월: 서경대학교 컴퓨터공학과 졸업
2016년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 소프트웨어 분석, 소프트웨어 난독화, 소프트웨어 보안

박 문 찬 (Moon-chan Park) 학생회원
2013년 2월: 서울시립대학교 수학과 학사
2013년 3월~2015년 2월: 고려대학교 정보보호대학원 석사졸업
2015년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 소프트웨어 분석, 소프트웨어 난독화, 소프트웨어 보안

이 동 훈 (Dong-hoon Lee) 종신회원
1983년 8월: 고려대학교 경제학사 졸업
1987년 12월: Oklahoma University 전산학과 석사 졸업
1992년 5월: Oklahoma University 전산학과 박사 졸업
1993년 3월~1997년 2월: 고려대학교 전산학과 조교수
1997년 3월~2001년 2월: 고려대학교 전산학과 부교수
2001년 3월~현재: 교려대학교 정보보호대학원 교수
〈관심분야〉 암호 프로토콜, 암호이론, 함수 암호, 소프트웨어 보안, 모바일 보안, 자동차 보안, USN이론