

실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구

A Study on incapacitation algorithm for automatic analysis
of packed malware

수탁기관 : 고려대학교 산학협력단

2010. 12. 30



제 출 문

한국정보보호진흥원 원장 귀하

본 보고서를 “실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구”의 최종연구보고서로 제출합니다.

2010 년 12 월 30 일

수탁 기관: 고려대학교 산학협력단

연구책임자: 교 수 문 종 섭 (고려대학교 전자 및 정보공학과)

참여연구원: 연 구 원 정 만 현 (고려대학교 정보보호대학원)

연 구 원 이 영 훈 (고려대학교 정보보호대학원)

연 구 원 여 성 철 (고려대학교 정보보호대학원)

연 구 원 양 대 엽 (고려대학교 정보보호대학원)

연 구 원 신 승 훈 (고려대학교 정보보호대학원)

연 구 원 김 이 룩 (고려대학교 정보보호대학원)

요 약 문

1. 제목

실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구

2. 연구 개발의 목적 및 중요성

1) 연구개발의 목적

최근 봇이나 웜 등의 악성코드로 인한 사이버 공격이 증가하고 있다. 봇이나 웜 등을 이용하여 좀비 PC를 생성하고 이를 통하여 공격자는 일반 사용자의 개인정보를 수집하거나 금전적인 피해를 입히고 있다. 매일 새롭게 발견되는 악성코드의 수와 종류는 지속적으로 증가하고 있기 때문에 그에 대한 신속한 분석 및 대응이 필요하게 되었다.

최근 공격자들은 악성코드를 제작할 때 신속한 분석 및 대응이 어렵게 하도록 다양한 탐지 및 분석 회피 기술들을 사용하고 있어 악성코드로 인하여 사건이 생겼을 때 신속한 대응이 어려워지고 있다.

다양한 탐지 및 분석 회피 기술 중 실행 압축 기술은 악성코드의 용량을 줄이고 분석가가 코드를 분석할 때 혼란을 주도록 코드가 변형되기 때문에 악성코드의 확산이 용이해지고 분석하는데 시간이 오래 걸려 신속한 대응이 어렵게 만들고 있다.

본 연구 과제에서는 이러한 실행 압축된 악성코드의 분석이 용이하도록 실행 압축된 악성코드 자동 분석을 위하여 실행 압축 기술의 현황 및 탐지 방법 그리고 실행 압축 기술의 무력화 방법을 조사하고 이것을 바탕으로 실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구를 목표로 한다.

2) 연구개발의 중요성

최근 악성코드들은 자신을 안티 바이러스 프로그램들로부터 보호하기 위해서 실행 압축되어 있는 경우가 많다. 안티 바이러스 및 정보보호 관련 연구 기관인 AV-Test사에 따르면 악성코드의 92% 이상이 이러한 실행 압축 기술을 사용하고 있다고 한다.

실행 압축된 악성코드들과 실행 압축되지 않은 악성코드들의 차이는 첫 번째로 실행 압축된 악성코드들은 일반적인 악성코드에 비해 그 크기가 작아 확산이 빠르다. 두 번째로 분석가가 악성코드를 분석할 때 실행 압축된 악성코드는 실행 압축을 해제하기 전까지는 분석이 불가능하다는 장점이 있다. 따라서 실행 압축된 악성코드들이 일반적인 악성코드에 비해 생존 시간이 길어 오랜 시간 유포시킬 수 있게 된다. 세 번째로 같은 악성코드라도 실행 압축 방법에 따라 전혀 다른 모습의 악성코드로 변하기 때문에 다양한 형태의 변형이 나올 수 있다. 따라서 현재 가장 많이 사용되는 시그니처 기반의 안티 바이러스 프로그램들은 변형된 악성코드들을 탐지하지 못하는 문제가 생긴다. 또한 실행 압축 해제 기술에 대한 많은 연구가 진행 중이지만 여전히 성능의 한계가 존재한다.

따라서 효율적인 악성코드의 분석과 대응을 위해서도 이러한 실행 압축 기술을 무력화하는 자동화 알고리즘이 필요하다. 본 연구를 통해서 개발되는 실행 압축된 악성코드 자동분석을 위한 무력화 알고리즘은 알려진 실행 압축 기법과 알려지지 않은 새로운 실행 압축 기법을 통해 만들어진 악성코드를 효율적으로 압축 해제하여 보다 신속한 분석과 대응이 가능하도록 한다. 이를 통해 악성코드에 대한 피해 확산을 최소화시킬 수 있을 것으로 판단된다.

3. 연구개발의 내용 및 범위

본 연구과제에서는 실행 압축된 악성코드 자동분석을 위한 무력화 알고리즘 연구를 위해 다음과 같은 연구 내용을 수행한다.

- 실행 압축 도구의 최신 동향
- 최신 실행 압축 도구 현황 조사

- 실행 압축 도구 탐지 방법 조사 및 분석

- 실행 압축 해제 기법 조사

- 실행 압축 해제 연구 동향 조사
- 매뉴얼 실행 압축 해제 기법 분석
- 실행 압축 해제 S/W 또는 Module 조사 및 분석

- 실행 압축 무력화 기법 제시

- 기존의 실행 압축 해제 연구의 한계점 조사
- 새로운 실행 압축 무력화 기법 제시
- 제시하는 기법에 관한 실험 및 증명

4. 연구 결과

현재까지의 연구 결과 내용은 다음과 같다.

- 중간 연구 보고서

중간 연구 보고서에서는 앞으로의 실행 압축된 악성코드 무력화 알고리즘 연구를 위한 최신 실행 압축 도구 현황 및 탐지 방법 그리고 실행 압축 해제 기법 조사를 하였다. 최신 실행 압축 도구 현황을 파악함으로 앞으로 나타날 실행 압축 도구에 대비하고 그에 따른 실행 압축 도구 탐지 방법을 연구하였다. 또한 현재 나와 있는 실행 압축 해제 연구 동향을 조사하고 정리하였으며 매뉴얼 실행 압축 해제 기법과 실행 압축 해제 S/W 또는 Module을 조사하고 분석하여 정리하였다.

- 최종 연구 보고서

최종 연구 보고서에서는 중간 연구 보고서 이후의 새로 나온 실행 압축 도구 현황 및 탐지 방법 그리고 실행 압축 해제 기법에 관한 조사가 추가로 되었다. 또한 중간 연구 보고서에서 조사한 실행 압축 해제 연구 동향 이외의 최신 연구 동향을 조사하여 정리 하였다. 그리고 기존의 연구에서

제시한 방법의 단점을 보완하는 새로운 실행 압축 무력화 방법을 제시하였다.

5. 활용에 대한 건의

본 최종 보고서의 결과물은 실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구에 배경 연구 자료로 사용되며 이로 인하여 효율적인 실행 압축된 악성코드 자동분석을 위한 무력화 알고리즘 도출

6. 기대효과

본 연구과제에서 도출되는 실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘은 탐지 및 분석을 회피하는 실행 압축 기술이 적용된 악성코드를 분석할 때 보다 효율적으로 분석할 수 있는 환경을 제공할 것이다. 분석을 회피하는 실행 압축 기술을 무력화 시켜 본래의 악성코드를 복원하여 분석을 용이하게 하고 신속한 대응이 가능할 것으로 기대된다.

SUMMARY

1. Title

A Study on incapacitation algorithm for automatic analysis of packed malware

2. Purpose and importance of the study

1) Purpose

Recently, Cyber-attack is increasing by Bot, Worm and Malicious code. The attacker has collected user's personal information or inflicted monetary damages through Zombi-PC which make by Bot, Worm, etc. Every day the number and type of newly discovered Malicious is continuously increasing, so, It's necessary to analysis and response quickly. When the incident happened due to malware, it's hard to respond quickly because when the attackers make malicious code, they use detection and analysis of a variety of avoidance techniques. Packed technique makes difficult to respond quickly because the malicious-code is reduced size that easy to diffusion and changed code that make spend longer time for analysis. This thesis a base research about a neutralization algorithm for automatic analysis. And research Construction and detection methods of the packed technique for easy to analysis of the packed malicious code.

2) Importance

Recently There are many cases that Malicious cods are packed

because of protecting from the Anti-virus program. According to AV-Test (Anti-virus and information security-related research institution) more than 92 percent of the malwares are being used a packing technology. The difference between Packed malware and none packed malware, firstly, packed malware is smaller and the spread speed is faster than normal malware. Secondly, when A analyst want to analyze a malicious code, there is a advantage that can not analyze the malware code until unpack the malware. Therefore, packed malware's survival-time are longer than normal malware and it can be more circulated. Thirdly, the malware are changed difference appearance depend on a method of packing. It can be appeared various transformations. Thus, there is a problem that currently using of the signature-based anti-virus programs can not detect variations of malware. Also There are limits of performance even running a lot of researches on unpacking technology. Therefore, we need a automatic neutralization algorithm for a effective malware analysis and response. This research is being develop neutralization algorithm for a automated analysis of packed malware can rapid analysis and respond to the known packed malware and unknown new packed malware by a effective unpacking. Through this, we can expect to minimize the damage spread of malware.

3. Contents and scope of the study

In this research, we will perform following acts for incapacitation algorithm for automatic analysis of packed malware.

- Trend of Packing tool
 - Survey Packing tool trend
 - Survey and Analyze detecting packing tool
- Survey malware unpack method

- Survey unpack method trend.
 - Analyze manual unpack method.
 - Survey and Analyze unpack software or module
- Proposition of packing incapacitation technique
- Survey limitation of previous studies about unpacking
 - Proposition of novel packing incapacitation technique
 - Experiment and verification of the proposed technique.

4. Results of the study

The current result of the study is as follows.

○ Midterm Report

In the midterm report, we carried out a pre-stage for research of packed malware incapacitation algorithm. The stage is consisted of survey packing tool trend, detecting method and unpacking method. Also we have surveyed the existing automatic unpacking method research, manual unpacking method and unpacking software or module.

○ Final Report

In the final report, we have added the new packing tool trends and detection techniques after the midterm report and surveys of unpacking technique. We also have added the surveys of recent unpacking study trends in addition to the midterm report. Moreover, we have proposed novel packing incapacitation technique complementary to the methods proposed by previous studies.

5. Suggestion about practical use of the project

The result of the research will be applicable to used for background research data of study on incapacitation algorithm for automatic analysis of packed malware. It will draw efficient incapacitation algorithm for automatic analysis of packed malware.

6. Expected Effects

This research is drawn from automated analysis algorithm of packed malware to make efficiently analysing environment for evading detection and analysis of packed malware. It is expected to take advantage of rapid response to malware, which was packed for evading analysis, restoring natural malware and analysis malware possible.

목 차

제 1 장 서론	1
제 1 절 과제의 목적	1
제 2 절 과제의 필요성	1
제 2 장 실행 압축 도구의 최신 동향	5
제 1 절 최신 실행 압축 도구 현황	5
제 2 절 실행 압축 도구 탐지 방법	7
제 3 장 실행 압축 해제 방법 조사	21
제 1 절 실행 압축 해제 연구 동향 조사	21
제 2 절 매뉴얼 실행 압축 해제 기법	45
제 3 절 실행 압축 해제 S/W 또는 Module	66
제 4 장 제안하는 실행 압축 무력화 방법	101
제 1 절 엔트로피 계산	102
제 2 절 엔트로피 그래프 역 분석	104
제 3 절 오리지널 엔트리 포인트 도출	118
제 5 장 결 론	125

그림 목차

(그림 1) 커스터마이즈 또는 해커 개인들이 만들어낸 실행 압축 도구 량	2
(그림 2) 하루(2010.12.26) 동안 분석된 실행 압축 도구 현황	5
(그림 3) PEID 실행 화면	8
(그림 4) Extra Information 창	9
(그림 5) EP 정보 기반 시그니처 생성도	9
(그림 6) Section 정보 기반 시그니처 생성도	10
(그림 7) MRC를 통한 실행 압축 파일 탐지(Scan a Folder)	11
(그림 8) Console mode로 MRCAgent를 실행한 화면	12
(그림 9) Console mode에서 결과를 .xml 파일로 출력한 화면	12
(그림 10) MRC의 엔트로피 계산법	13
(그림 11) Exeinfo PE의 기본 화면	14
(그림 12) Lamer Info에서 실행 압축 해제 정보를 제공	14
(그림 13) ver.0.0.2.7의 detection list	15
(그림 14) Zero-byte Test - Sections	15
(그림 15) ASPack에 대한 PEiD의 탐지	16
(그림 16) ASPack에 대한 Exeinfo PE의 탐지	16
(그림 17) FastScanner 3 기본 화면	17
(그림 18) UPX로 실행 압축된 프로그램을 확인한 결과	18
(그림 19) FastScanner 3의 플러그인 목록	18
(그림 20) GenOEP 플러그인(2004년 2월 15일)으로 UPX의 OEP를 찾은 화면	19
(그림 21) 논문에서 제안하는 실행 압축 여부 분류기 (파란색 부분)	23
(그림 22) 엔트로피를 이용한 실행 압축 해제 기법의 흐름도	26
(그림 23) 실행 압축 기법 별 실행 압축 파일의 엔트로피 변화	27
(그림 24) Renovo의 구조	28
(그림 25) 악성코드 분석 흐름도	30
(그림 26) 논문에서 제안하는 DBI 엔진	32
(그림 27) Rotalume의 구조	33
(그림 28) ReconBin의 구조	35
(그림 29) OEP탐지 흐름도	37

(그림 30) 논문에서 제안하는 프로세스 진행 과정	45
(그림 31) PEID로 실행압축 기법 확인	47
(그림 32) UPX0 섹션 확인	48
(그림 33) 디버깅 도구를 이용하여 본 실행 압축 PE 파일의 시작 부분	49
(그림 34) ESP 위치 보기 화면	50
(그림 35) 브레이크 포인트 설정 화면	50
(그림 36) 오리지널 엔트리 포인트 접근	50
(그림 37) JMP 실행 후 화면	51
(그림 38) 메모리 맵과 비교 화면	51
(그림 39) IAT 수동 복구 과정 (IAT 주소값 확인)	52
(그림 40) IAT 수동 복구 과정 (API 함수 확인)	52
(그림 41) 도구를 이용하여 IAT 복원하는 과정	53
(그림 42) IAT 복구 전 실행 파일 오류 화면	54
(그림 43) IAT 복구 후 정상 실행 화면	54
(그림 44) ASpack 2.12 version 실행 화면	55
(그림 45) PEiD로 ASpack 2.12를 탐지한 화면	56
(그림 46) ollydbg로 ASpack으로 실행 압축된 PE 파일의 시작 부분	56
(그림 47) ESP에 브레이크 포인트 지정	57
(그림 48) OEP에 근접한 화면	58
(그림 49) OEP 위치에 진입	58
(그림 50) Ollydump를 이용한 이미지 덤프	59
(그림 51) ImportREC를 이용한 IAT 복구	60
(그림 52) PEID로 실행 압축 기법 확인	61
(그림 53) Section Viewer 화면	61
(그림 54) PE 파일 시작 부분	62
(그림 55) FSG의 특징 점	62
(그림 56) OEP 로 이동한 화면	63
(그림 57) 도구를 이용하여 IAT 복원하는 과정	64
(그림 58) Import table 정보 확인 화면	65
(그림 59) 실행 압축 해제 후 실행 화면	66
(그림 60) PEID 로 실행 압축 해제 확인 화면	66
(그림 61) FUU 실행 화면	67

(그림 62) 추가적으로 제공되는 도구	68
(그림 63) 플러그인 설정 화면	69
(그림 64) FUU 실험 결과 화면	70
(그림 65) AbstersiverA의 기본 화면	71
(그림 66) AbstersiverA의 Option	71
(그림 67) ASPack 1.08 버전으로 압축된 파일	72
(그림 68) ASPack을 실행 압축 해제 하는 과정	73
(그림 69) Unpack 프로그램 실행 후 PEiD로 확인한 결과	74
(그림 70) Quick Unpack의 기본 화면	75
(그림 71) UPX로 실행 압축된 파일을 실행 압축 해제 하는 과정	78
(그림 72) RL!dePacker 기본 화면	79
(그림 73) RL!dePacker 실행 압축 해제 실행 화면	83
(그림 74) Themida Packer와 Protection Option	85
(그림 75) THEMIDA/WINLICENSE UNPACKER V2.0 기본 화면	86
(그림 76) THEMIDA/WINLICENSE UNPACKER V2.0 실행 화면	87
(그림 77) UPX-iT 기본 화면	88
(그림 78) UPX로 실행 압축된 파일을 등록	88
(그림 79) 실행 압축 해제 실행화면	89
(그림 80) 실행 압축 해제 후 PEiD로 확인한 결과	89
(그림 81) UPX-iT History 확인 결과	90
(그림 82) UPX가 아닌 파일에 대한 결과	90
(그림 83) 최신 버전 UPX(3.07)로 실행 압축된 파일을 실행 압축 해제 한 결과	91
(그림 84) VMUnpacker 기본 화면	92
(그림 85) Mew 11로 실행 압축된 파일을 등록한 화면	95
(그림 86) 실행 압축 해제를 실행한 결과	95
(그림 87) linxerUnpacker 실행 화면	97
(그림 88) MEW 1.1을 실행 압축 해제한 화면	98
(그림 89) 실행 압축 파일 실행 과정	101
(그림 90) 정상 파일에서의 엔트로피 값의 범위	104
(그림 91) UPX로 실행 압축된 구구단 프로그램의 엔트로피 그래프	105
(그림 92) ASpack로 실행 압축된 구구단 프로그램의 엔트로피 그래프	106
(그림 93) FSG로 실행 압축된 구구단 프로그램의 엔트로피 그래프	106

(그림 94) UPX로 실행 압축된 사다리 게임 파일의 엔트로피 그래프	107
(그림 95) ASpack로 실행 압축된 사다리 게임 파일의 엔트로피 그래프	108
(그림 96) FSG로 실행 압축된 사다리 게임 파일의 엔트로피 그래프	108
(그림 97) ASpack로 실행 압축된 악성코드 파일의 엔트로피 그래프	109
(그림 98) FSG로 실행 압축된 사다리 게임 파일의 엔트로피 그래프	110
(그림 99) UPX로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프	111
(그림 100) FSG로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프	111
(그림 101) ASpack으로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프	112
(그림 102) UPX로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프	113
(그림 103) FSG로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프	114
(그림 104) ASpack로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프	114
(그림 105) ASpack로 실행 압축된 악성코드의 후반부 엔트로피 그래프	115
(그림 106) FSG로 실행 압축된 악성코드의 후반부 엔트로피 그래프	116
(그림 107) 쓰레기 코드 삽입한 파일의 엔트로피 그래프 개형 비교	117
(그림 108) 제안하는 기법의 전체 구성도	118
(그림 109) 종료시점부터의 엔트로피 변화 값 평균을 구하는 화면	119
(그림 110) 엔트로피 값의 변화량이 특정범위를 넘어가는 구간 확인	120
(그림 111) OEP 추정 시작점 확인 화면	121
(그림 112) Jump 명령어 지점 확인 화면	122
(그림 113) 실행 압축 해제 완료 확인 화면	122

표 차례

[표 1] PHAD에서 사용한 PE Header 의 특징 값	21
[표 2] 특징 값들의 평균	22
[표 3] 분류를 위한 속성 값	24
[표 4] 실험 결과	25
[표 5] 정상 파일의 엔트로피 측정 결과	102
[표 6] 실행 압축 파일의 엔트로피 측정 결과	103
[표 7] 정상적인 실행 파일의 엔트로피 범위 값	104

제 1 장 서론

제 1 절 과제의 목적

최근 봇이나 웜 등의 악성코드로 인한 사이버 공격이 증가하고 있다. 봇이나 웜 등을 이용하여 좀비 PC를 생성하고 이를 통하여 공격자는 일반 사용자의 개인정보를 수집하거나 금전적인 피해를 입히고 있다. 매일 새롭게 발견되는 악성코드의 수와 종류는 지속적으로 증가하고 있기 때문에 그에 대한 신속한 분석 및 대응이 필요하게 되었다.

최근 공격자들은 악성코드를 제작할 때 신속한 분석 및 대응이 어렵게 하도록 다양한 탐지 및 분석 회피 기술들을 사용하고 있어 악성코드로 인하여 사건이 생겼을 때 신속한 대응이 어려워지고 있다.

다양한 탐지 및 분석 회피 기술 중 실행 압축 기술은 악성코드의 용량을 줄이고 분석가가 코드를 분석할 때 혼란을 주도록 코드가 변형되기 때문에 악성코드의 확산이 용이해지고 분석하는데 시간이 오래 걸려 신속한 대응이 어렵게 만들고 있다.

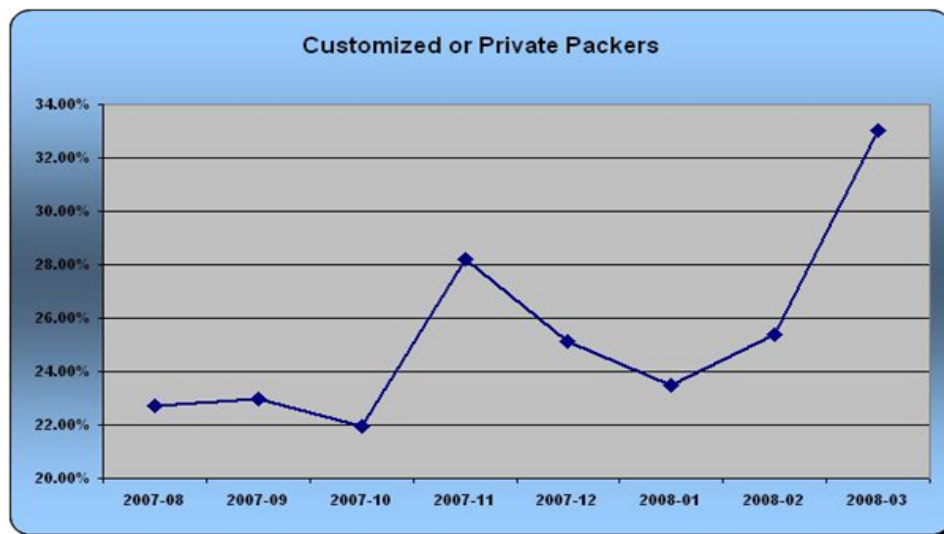
본 연구 과제에서는 이러한 실행 압축된 악성코드의 분석이 용이하도록 실행 압축된 악성코드 자동 분석을 위하여 실행 압축 기술의 현황 및 탐지 방법 그리고 실행 압축 해제 기법을 조사하고 이것을 바탕으로 실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구를 목표로 한다.

제 2 절 과제의 필요성

최근 악성코드들은 자신을 안티 바이러스 프로그램들로부터 보호하기 위해서 실행 압축되어 있는 경우가 많다. 안티 바이러스 및 정보보호 관련 연구 기관인 AV-Test사[1]에 따르면 악성코드의 92% 이상이 이러한 실행 압축 기술을 사용하고 있다고 한다.

실행 압축된 악성코드들과 실행 압축되지 않은 악성코드들의 차이는 첫 번째로 실행 압축된 악성코드들은 일반적인 악성코드에 비해 그 크기가 작아 확산이 빠르고 용이하다. 두 번째로 분석가가 악성코드를 분석할 때

실행 압축된 악성코드는 실행 압축을 해제하기 전까지는 분석이 불가능하다는 장점이 있다. 따라서 실행 압축된 악성코드들이 일반적인 악성코드에 비해 생존 시간이 길어 오랜 시간 유포시킬 수 있게 된다. 세 번째로 같은 악성코드라도 실행 압축 방법에 따라 전혀 다른 모습의 악성코드로 변하기 때문에 다양한 형태의 변형이 나올 수 있다. 따라서 현재 가장 많이 사용되는 시그니처 기반의 안티 바이러스 프로그램들은 변형된 악성코드들을 탐지하지 못하는 문제가 생긴다. 다음 그림은 Panda Research[2]에서 조사한 변형된 실행 압축 기술의 분포를 나타내는 그림이다. 다음과 같이 공격자에 의하여 변형된 실행 압축 기술이 증가하고 있고 그에 따라 변형된 악성코드들이 증가하여 탐지가 어려워지는 문제가 발생한다.



(그림 1) 커스터마이즈 또는 해커 개인들이 만들어낸 실행 압축 도구
량

또한 이러한 실행 압축 해제 기법에 대한 많은 연구가 진행 중이지만 여전히 성능의 한계가 존재한다.

따라서 효율적인 악성코드의 분석과 대응을 위해서도 이러한 실행 압축을 무력화하는 자동화 알고리즘이 필요하다. 본 연구를 통해서 개발되는 실행 압축된 악성코드 자동분석을 위한 무력화 알고리즘은 알려진 실행 압축 기법과 알려지지 않은 새로운 실행 압축 기법을 통해 만들어진 악성

코드를 효율적으로 압축 해제하여 보다 신속한 분석과 대응이 가능하도록 한다. 이를 통해 악성코드에 대한 피해 확산을 최소화시킬 수 있을 것으로 판단된다.

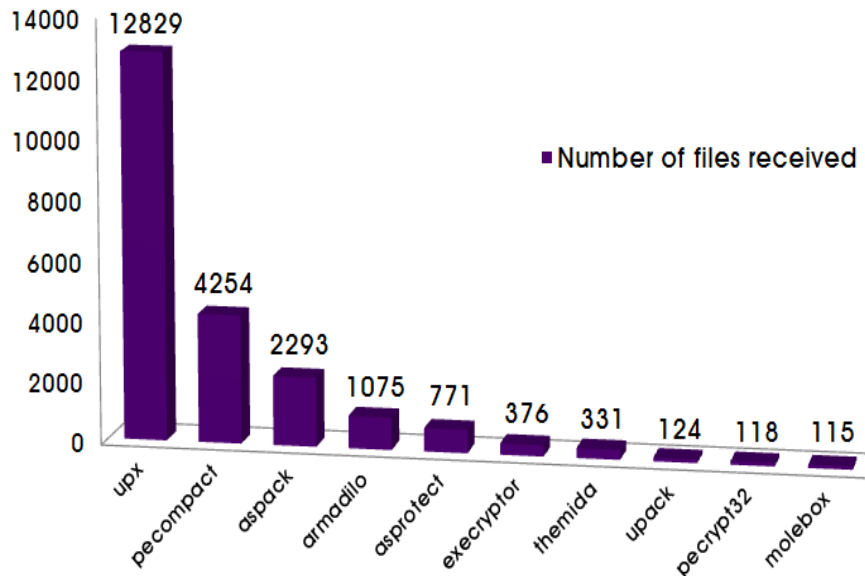
참고문헌

- [1] AV-Test. <http://www.av-test.org>
- [2] Panda Research. <http://research.pandasecurity.com/packer-revolution>

제 2 장 실행 압축 도구의 최신 동향

제 1 절 최신 실행 압축 도구 현황

1. 실행 압축 도구 통계



(그림 2) 하루(2010.12.26) 동안 분석된 실행 압축 도구 현황

악성코드 검사 사이트인 바이러스 토탈(Virus Total)[1]에서는 분석됐던 악성코드를 통해 실행 압축 도구에 대한 통계를 제공하고 있다. 통계 자료에서는 분석된 실행 압축 도구 중 가장 많이 사용된 10개의 종류와 수에 대한 통계치를 제공하고 있다. 통계 자료를 살펴보면 12월 26일 하루 동안 UPX가 12,829개로 가장 많은 비중을 차지하고 있었고, PEcompact 4254개, ASPack 2293개로 가장 많은 비중을 차지하는 것을 알 수 있다.

이중 많이 사용되는 UPX와 ASPack의 경우 실행파일을 압축하는 Compressor역할을 한다. Compressor역할을 하는 실행 압축 도구들의 경우 별다른 안티 리버싱의 기법이 적용되지 않고, 실행 압축 해제도 비교적 쉽다. UPX와 ASPack는 순수한 의도로 사용되는 경우가 많으나 이와는 다

르게 악의적인 의도로 사용되는 실행 압축 도구에는 UPack, PESpin, NSAnti등이 있다.

UPack은 중국의 Dwing이 만든 실행 압축 도구로 원본 파일을 크게 변형시키고 PE헤더를 심하게 훼손한다. 실제로 UPack이 처음 등장했을 때 여러 PE분석 프로그램들이 정상적으로 작동하지 않았고, 이 특징 때문에 악성코드 제작자들이 UPack을 주로 사용하게 되었다. 이로 인해 현재 대부분의 안티바이러스 프로그램은 UPack으로 실행 압축된 파일은 무조건 삭제한다.

Protector역할을 하는 실행 압축 도구는 안티 리버싱기법이 적용되어 실행 압축 해제가 매우 까다롭다. 상용 Protector에는 ASProtect, Themida, SVKP등이 있고, 공개용 Protector에는 UltraProtect, Morphine등이 있다.

이 중 Themida는 옵션마다 풀기위한 접근 방식이 다르기 때문에 강력한 protector중 하나이고 Themida로 실행 압축된 실행파일을 실행 압축 해제하기가 매우 어렵다. 그렇기 때문에 몇몇 악성코드는 강력한 실행 압축을 위해 Themida를 종종 사용한다.

2. 악성코드에 이용되는 실행 압축 도구의 특징

악성코드 제작자들이 실행 압축 도구를 사용하는 이유로는 크게 세 가지를 들 수 있다.

첫째, 악성코드의 크기가 줄어든다. 악성코드는 자기복제, 전송 등의 시간을 줄이기 위해 악성코드의 길이가 짧아야 바이러스를 빨리 유포시킬 수 있다. 그러나 현재는 컴퓨터와 네트워크의 성능으로 인해 악성코드 크기는 그리 민감한 문제가 되지 않는다.

둘째, 백신의 진단을 피할 수 있다. 실행 압축 도구를 통해 실행파일을 압축하게 되면 안티바이러스 프로그램이 압축을 풀기 전까지 악성코드를 진단할 수 없게 된다.

셋째, 분석을 어렵게 한다. 실행 압축을 한 악성코드는 리버스 엔지니어링을 하기까지 시간이 더 오래 걸리므로, 악성코드에 대한 백신이 나오기까지 더 오랜 시간 유포시킬 수 있다. 즉 실행 압축을 통해 악성코드의 생존시간을 늘린다.

3. 악성코드에 이용되는 실행 압축 도구 현황

2004년 악성 IRC봇 소스가 공개되면서 같은 소스에서 접속 IRC를 수정하여 변형된 악성코드가 대량으로 등장했다. 이 당시 악성코드 제작자들은 안티바이러스 프로그램의 진단을 우회하기 위해 실행 압축 도구를 사용함으로써, 안티바이러스 프로그램은 실행 압축 탐지 기능을 강화했다.

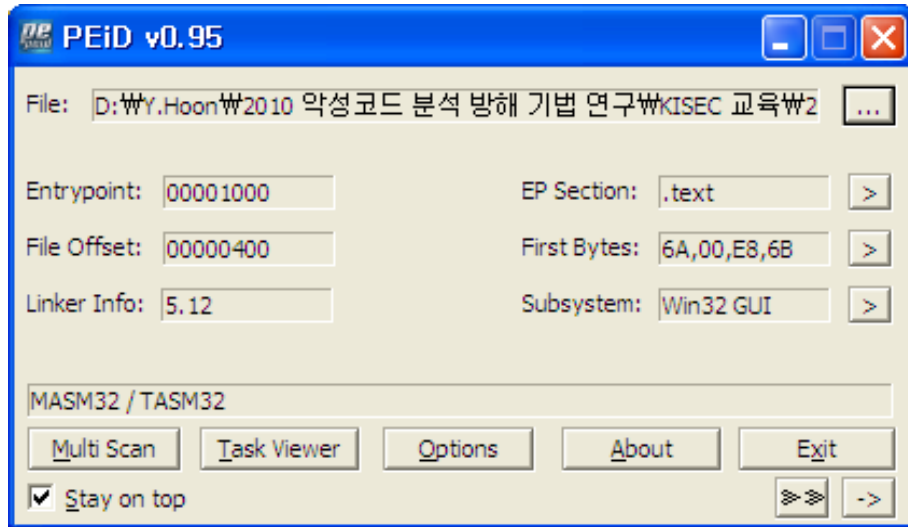
2005년 이후로 악성코드 제작자들은 잘 알려져 있지 않은 실행 압축 도구를 이용하거나 자신만의 실행 압축 도구를 만들어 사용하기 시작했다. 또한 PE 패치로 불리는 실행 코드를 일부 수정해 안티바이러스 프로그램을 우회하는 방법도 증가했다.

포티넷의 브란 루(Bryan Lu)가 바이러스 블루틴 2007 컨퍼런스에서 발표한 자료에 따르면 자사에서 집계한 샘플을 분석한 결과 윈도우 악성코드의 40%가 실행 압축되어 있었다. 바이러스 블루틴(Virus Bulletin) 2007년 10월호에 헝가리 바이러스 버스터(VirusBuster)의 가보르 스자파노스(Gabor Szappanos) 연구원에 따르면 WildList[2]에 오른 739개 샘플 중 54개만 실행 압축되어 있지 않고 나머지 92%가 실행 압축되어 있으며 사용된 실행 압축 도구는 30 가지 이상이라고 한다.

제 2 절 실행 압축 도구 탐지 방법

1. PEID Signature

PEID[3]는 PE 파일의 실행 압축 도구 및 컴파일러를 탐지하는 가장 일반적인 공개 도구이다. 현재 PEID는 600개 이상의 시그니처를 기반으로 PE 파일의 실행 압축 도구 및 컴파일러를 탐지할 수 있다. 실행 화면은 다음 (그림 6)과 같이 Entry Point, File offset, EP section과 컴파일러 정보도 표현한다. 파일분석, 어셈블링, 디어셈블링을 할 때 도움이 되는 소프트웨어이다.

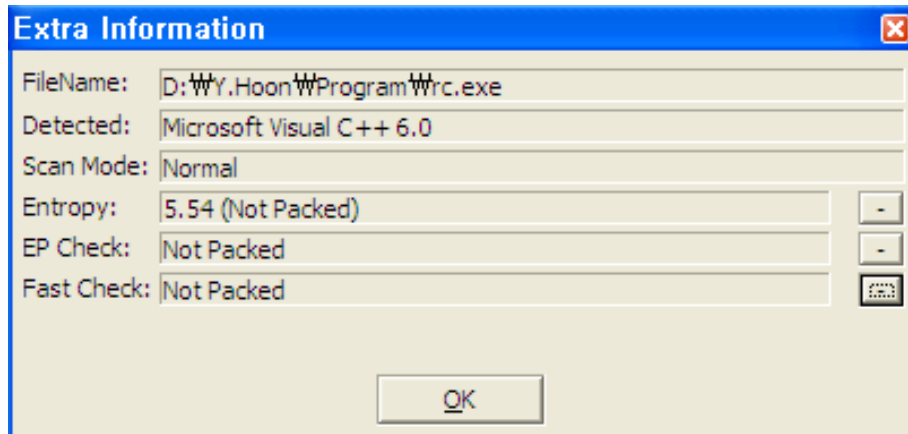


(그림 3) PEID 실행 화면

PEID의 주요한 특징은 다음과 같다.

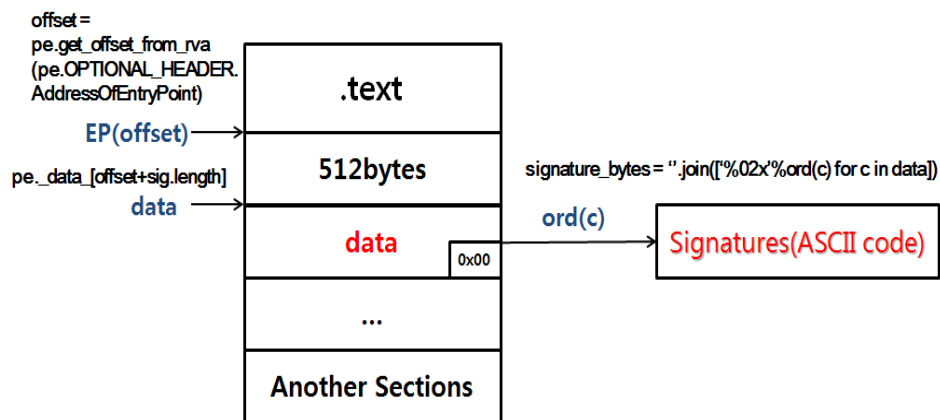
- 알려지지 않은 파일이나 변경된 파일을 감지하기 위한 “ Advanced ” 감지 모드
- 셸 확장, 커맨드 라인 지원, 항상 위에 옵션, 드래그 & 드롭 등의 인터페이스 제공
- 동시에 여러 파일 또는 여러 디렉토리를 스캔 가능
- Task Viewer 나 컨트롤러 제공
- Generic OEP Finder 나 Krypto ANAyzer 와 같은 다양한 플러그인을 사용할 수 있는 인터페이스
- Heuristic Scanning 옵션
- PE details, Imports, Exports, TLS Viewer 제공
- 빠른 디셈블러 내장
- hex스 뷰어 내장

또한 추가적인 정보를 더 제공해준다. Entropy 값을 체크하여 실행 압축이 되어 있는지 아닌지 체크해주고 EP (Entry Point)를 체크하여 실행 압축 여부를 판단해준다. 하지만 오탐율이 높아 신뢰 할 수는 없다.

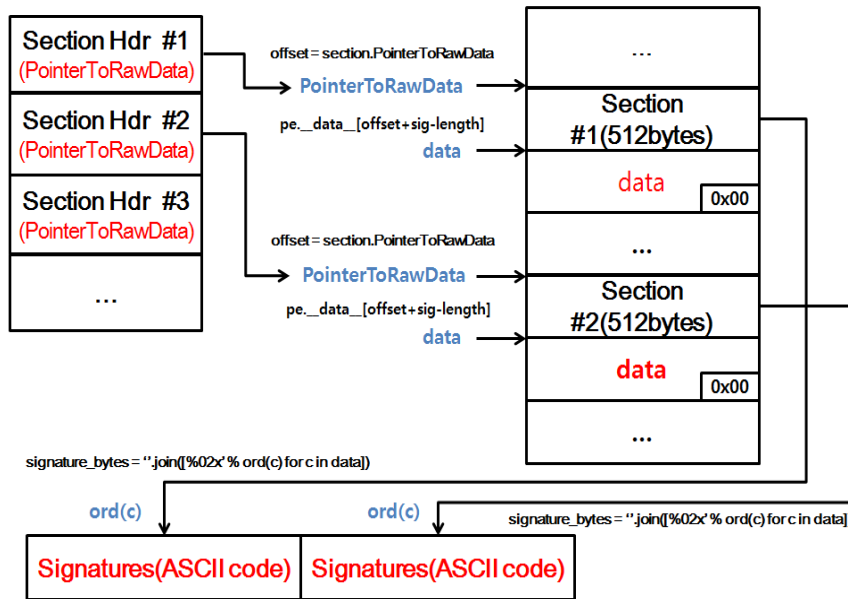


(그림 4) Extra Information 창

PEID의 실행 압축 파일 탐지 방법은 시그니처를 통한 탐지 방법으로 각 실행 압축 도구에 대한 시그니처 추출 방법은 크게 두 가지로 나뉘어진다. EP(Entry point)와 Section 정보를 기준으로 Hex Sequence를 시그니처로 사용하는 방법이다. 다음 그림들은 이러한 방법을 그림으로 표현한 것이다.



(그림 5) EP 정보 기반 시그니처 생성도



(그림 6) Section 정보 기반 시그니처 생성도

2. MRC (Mandiant Red Curtain)

MRC는 시스템 침해사고 대응을 위한 공개 소프트웨어이다. 이 도구는 시스템의 침해 여부를 빠르고 효과적으로 조사하기 위한 목적으로 설계되었다. PE 파일의 구조 분석을 통해 악성 파일인지 정상 파일인지 여부를 판단한다.

암호화 또는 압축 등의 회피 기법이 적용된 데이터의 경우 PE 파일의 엔트로피가 구조화된 데이터와 비교해 상대적으로 높다는 점을 이용해 가중치를 주는 방식이다. 침해 여부 결과는 위협 스코어를 계산하여 조사할 필요가 있는 파일을 Red로 표현하고 의심스러운 것을 Yellow, 정상적인 PE 파일은 Green으로 나타낸다. 그리고 PE 파일의 시그니처 분석을 통해서 실행 압축 여부를 판단한다.

Score	File	Size	Entry Point Signature	Entropy	Code Entropy	Anomaly Count	Signed	Details
0.035	C:\West\ffcalc.default.exe	114688	Microsoft Visual C++ v7.0	0.700	0.700	0	<input checked="" type="checkbox"/>	Details
0.764	C:\West\ffview.infect.exe	75328		0.725	0.699	1	<input type="checkbox"/>	Details
0.887	C:\West\ffWinKeyNT.exe	72824	PECompact v1.86	1.064	1.064	0	<input type="checkbox"/>	Details
0.956	C:\West\ffASpack.exe	30720	ASpack v1.03.03	0.954	0.000	1	<input type="checkbox"/>	Details
1.000	C:\West\ffProject1.exe	10240	ASpack v2.12	0.759	0.000	0	<input type="checkbox"/>	Details
1.025	C:\West\ffUPX.exe	40448	UPX v1.89.6 - v1.02 / v1.05 - v1.22	0.963	0.000	1	<input type="checkbox"/>	Details
1.131	C:\West\ffcalc.exe	70176	PE Diminisher v0.1	0.960	0.960	0	<input type="checkbox"/>	Details
1.688	C:\West\ffASProtect1.2.exe	247296	ASProtect v1.2	1.131	0.000	2	<input type="checkbox"/>	Details
1.912	C:\West\ffDodge.exe	33488	Antirack Software Protector v1.0...	1.073	1.073	2	<input type="checkbox"/>	Details
3.900	C:\West\ffsome.application.1.x-patch.exe	635444		0.992	0.992	4	<input type="checkbox"/>	Details
4.000	C:\West\ffVBOWatch v2.0.exe	112640		0.983	0.983	0	<input type="checkbox"/>	Details
4.400	C:\West\ffCalc17_AcidCrypt.exe	103088		0.943	0.943	3	<input type="checkbox"/>	Details
4.600	C:\West\ffserver.infect.exe	49624		0.907	0.907	5	<input type="checkbox"/>	Details

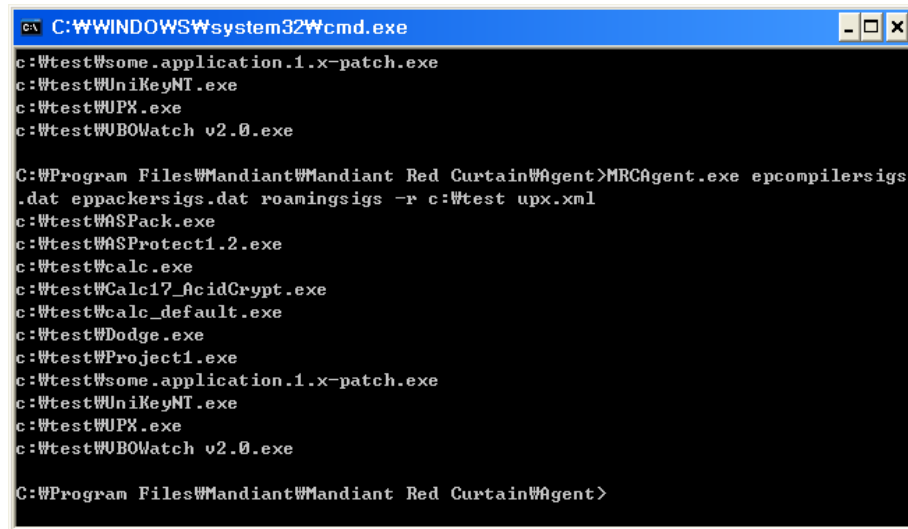
13 records loaded.

(그림 7) MRC를 통한 실행 압축 파일 탐지(Scan a Folder)

MRC는 위의 그림과 같이 Entry Pointer의 Signature를 검사하여 알려진 실행 압축 기술을 탐지한다. 그러나 Armadillo, AcidCrypt, Yoda, MEW와 같은 실행 압축 도구의 경우는 시그니처를 확인하지 못했다. 그리고 File Virus는 0.764의 점수로 정상파일로 판단하였고, Bot 전파 바이러스의 경우 4.800으로 위협 프로그램으로 판단하는 것을 확인하였다. UPX와 ASpack, ASProtect의 경우 코드 엔트로피를 0으로 계산했는데 이것은 Entry point 섹션을 잘못된 섹션으로 계산하여 이러한 결과가 나온 것으로 추정된다.

MRC는 파일의 엔트로피 이외에도 디지털 시그니처와 PE 구조의 이상, 실행 파일의 Imports, Section Permissions(실행 code, read 가능) 등으로 점수를 측정한다. 점수는 0.0 - 0.7은 의심하지 않아도 되고, 0.7 - 0.9는 약간의 관심이 필요하고, 0.9-1.0은 주의하여 봐야하고, 1.0 이상은 매우 주의하여 살펴봐야 한다.

MRC는 세부정보로 Section_starts_unaligned, Checksum_is_zero, Contains_eof_data, Incorrect_imageSize, Corrupted_imports, Empty_section_name, non_ascii_section_name, Overlapping_headers, Oversized_optional_header, Oversized_section, Invalid_entry_point로 이상에 대해서 알려준다.



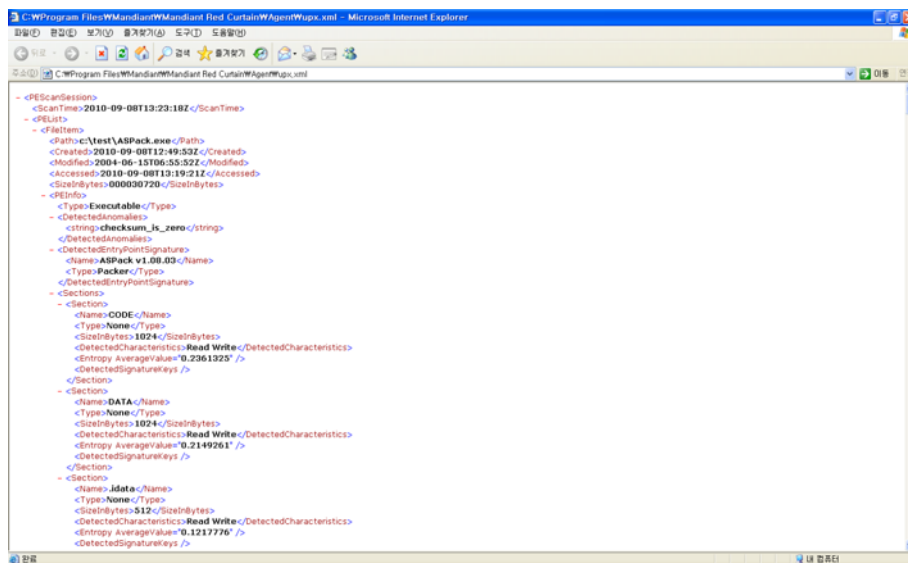
```
C:\WINDOWS\system32\cmd.exe
c:\wtest\some.application.1.x-patch.exe
c:\wtest\UnKeyNT.exe
c:\wtest\UPX.exe
c:\wtest\UBOWatch v2.0.exe

C:\Program Files\Mandiant\Mandiant Red Curtain\Agent>MRCAgent.exe epcompilersigs
.dat eppackersigs.dat roamingsigs -r c:\wtest\upx.xml
c:\wtest\ASPack.exe
c:\wtest\ASProtect1.2.exe
c:\wtest\calc.exe
c:\wtest\Calc17_AcidCrypt.exe
c:\wtest\calc_default.exe
c:\wtest\Dodge.exe
c:\wtest\Project1.exe
c:\wtest\some.application.1.x-patch.exe
c:\wtest\UnKeyNT.exe
c:\wtest\UPX.exe
c:\wtest\UBOWatch v2.0.exe

C:\Program Files\Mandiant\Mandiant Red Curtain\Agent>
```

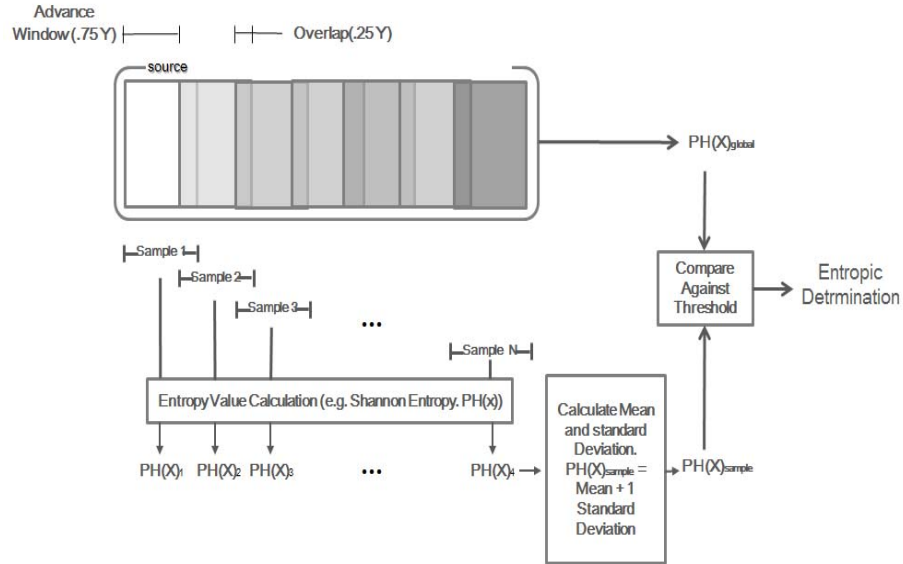
(그림 8) Console mode로 MRCAgent를 실행한 화면

위의 그림과 같이 MRC는 Console 모드에서 MRCAgent로 실행 할 수 있게 되었다. 결과는 txt파일이나 xml파일로 저장하여 볼 수 있고 GUI환경에서 나타난 결과와의 차이는 없었다.



(그림 9) Console mode에서 결과를 .xml 파일로 출력한 화면

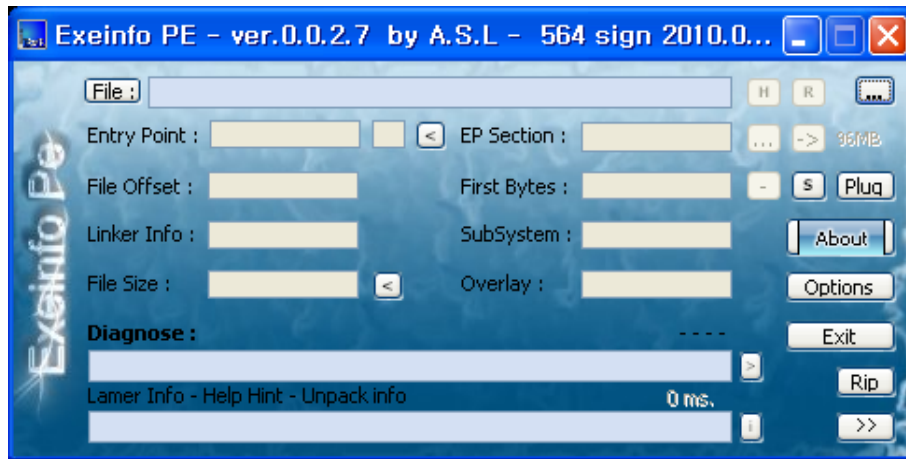
MRC는 파일의 Entropy를 결정하는데 슬라이딩 윈도우를 적용하였다. 이 방법은 작은 섹션을 가진 큰 블록의 데이터가 랜덤한 데이터를 가지고 있을 때 유용하다. 아래는 MRC가 슬라이딩 윈도우를 적용하여 Entropy를 계산하는 방법이다.



(그림 10) MRC의 엔트로피 계산법

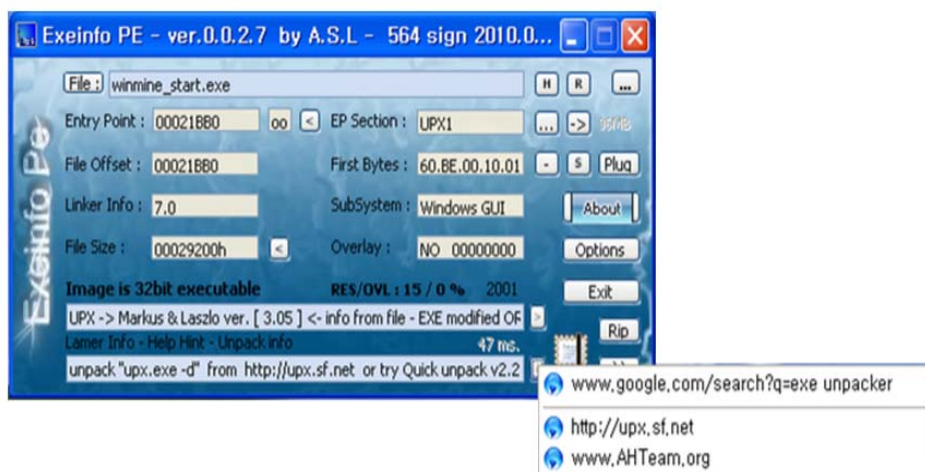
3. Exeinfo PE

PE 스캔툴로는 PEiD가 가장 널리 알려져 있고 많이 쓰이지만 마지막 업데이트인 2008년 10월 21일 이후 오랫동안 업데이트가 없이 지속적으로 플러그인과 시그니처만 업데이트 되었다. Exeinfo PE는 PEiD와 유사한 인터페이스를 제공하지만 프로그램에서 제공하는 다양한 옵션과 실행 압축 도구의 정보뿐만이 아닌 실행 압축 해제 프로그램 정보와 URL 제공, 다른 프로그램과의 연동 등을 제공한다. 또한 지속적인 업데이트를 통해 최신의 시그니처를 업데이트하기 때문에 많은 사용자들이 이용하고 있다.

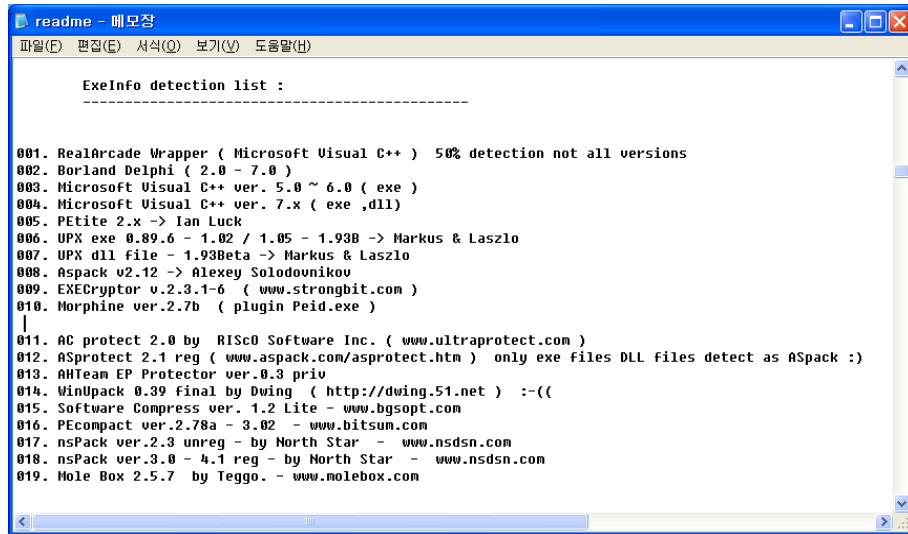


(그림 11) Exeinfo PE의 기본 화면

Exeinfo PE 프로그램은 기존의 PEiD의 인터페이스와 추가적으로 Lamer Info에서 실행 압축 해제 할 수 있는 실행 압축 해제 도구나 실행 압축 해제 프로그램을 받을 수 있는 웹사이트를 알려주는 기능을 제공하고 있다.

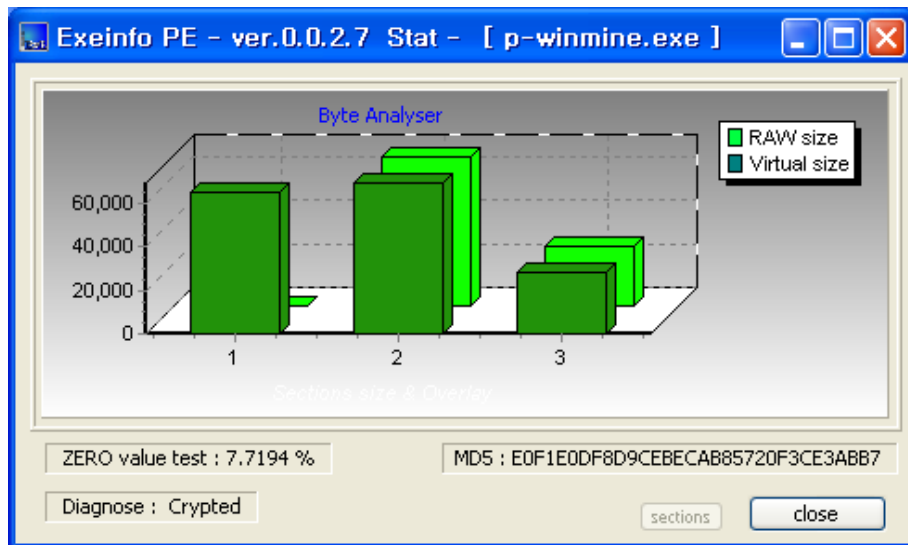


(그림 12) Lamer Info에서 실행 압축 해제 정보를 제공



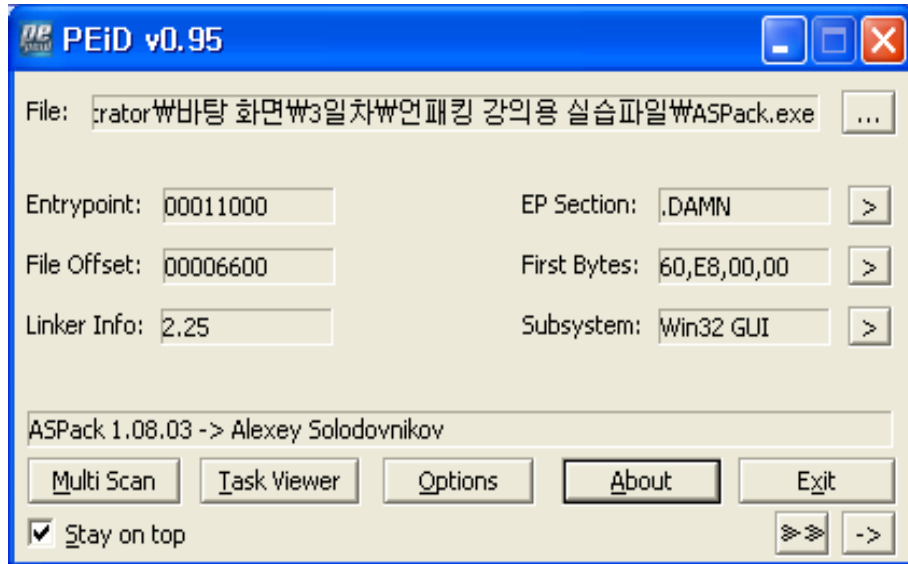
(그림 13) ver.0.0.2.7의 detection list

ver.0.0.2.7(updated 2010. 3. 26)에서는 564개의 시그니처를 탐지할 수 있으며, Exeinfo PE의 기본 폴더의 readme 파일에서 업데이트된 시그니처 정보를 확인할 수 있다.

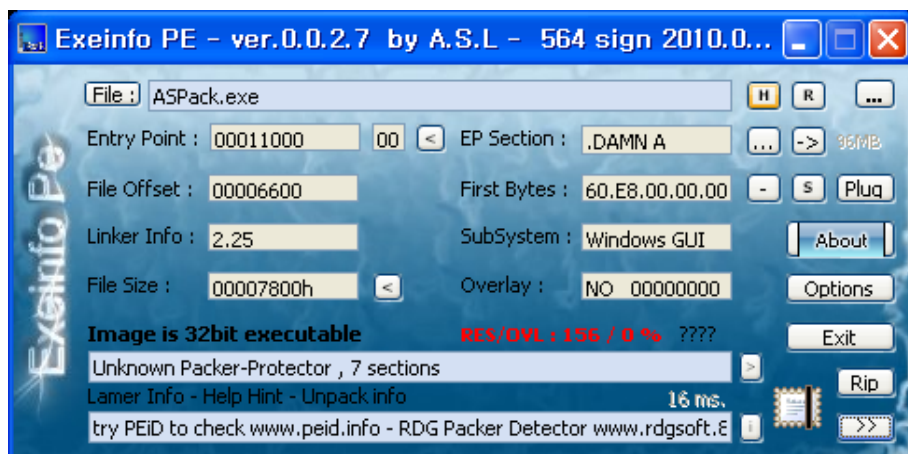


(그림 14) Zero-byte Test - Sections

위의 그림과 같이 Zero-byte 테스트를 통해 UPX와 같은 실행 압축 도구의 경우 섹션의 특징을 확인할 수 있다. 섹션이 Zero-byte로 채워져 있고 복원하는 과정에서 쓰여지는 실행 압축 도구의 경우 이러한 테스트를 통해 실행 압축 도구의 특징을 예측해 볼 수도 있다.



(그림 15) ASPack에 대한 PEiD의 탐지



(그림 16) ASPack에 대한 Exeinfo PE의 탐지

PEiD는 사용자가 시그니처를 생성하여 사용할 수 있지만 Exeinfo PE는 제작자에 의해 업데이트 되는 시그니처 외에 사용자가 시그니처를 업데이트 할 수 없었다. 동일한 파일에 대해 PEiD와 Exeinfo PE로 탐지를 하였을 때 PEiD는 실행 압축 도구를 찾아낸 반면, Exeinfo PE는 실행 압축 도구를 탐지하지 못하는 경우가 발생 하였다.

4. FastScanner

FastScanner 3(2010년 1월 7일 Updated)는 AT4RE(Arab Team 4 Reverse Engineering)[4]에서 만든 프로그램이다. PEiD를 기반으로 제작되었기 때문에, PE(Portable Executable) 시그니처를 분석하여 결과를 제공한다. AT4RE에서 제작한 플러그인과 추가적인 기능들을 제공하여 PEiD에 비해서 사용자가 보기 쉽고 이용하기 편리하게 제작되었다. 또한, 프로그램이 꾸준히 업데이트 되고 있고, 웹을 통하여 AT4RE에서 제공하는 플러그인을 업데이트 할 수 있어서 최신의 플러그인을 사용할 수 있는 장점이 있다. 시그니처 데이터베이스는 PEiD의 시그니처를 쓰고 있다.



(그림 17) FastScanner 3 기본 화면



(그림 18) UPX로 실행 압축된 프로그램을 확인한 결과



(그림 19) FastScanner 3의 플러그인 목록

FastScanner 3는 Options 탭에서 인터넷을 통해 쉽게 플러그인과 시그니처 데이터베이스를 업데이트 할 수 있고, 플러그인을 업데이트 하면 AT4RE에서 제작한 플러그인과 PEiD에서 제공하는 플러그인들을 제공받

을 수 있다.



(그림 20) GenOEP 플러그인(2004년 2월 15일)으로 UPX의 OEP를
찾은 화면

플러그인 목록 중 GenOEP와 PEiD Generic Unpacker를 통해 UPX로 실행 압축된 프로그램을 테스트한 결과 정확한 OEP 지점을 찾아내는 것을 확인하였다. 하지만 다른 실행 압축 도구로 실행 압축된 프로그램으로 테스트한 결과 OEP를 못 찾아내는 경우가 발생하였다.

실행 압축된 파일에 대해서는 PEiD나 Exeinfo PE와 유사한 결과를 보여줬다. 하지만 실행 압축 되지 않은 파일에 대해서는 PEiD나 Exeinfo PE는 실행 압축 되지 않았다고 탐지하였지만, Fastscanner는 오탐을 하는 경우가 발생하였다.

참고문헌

- [1] Virus Total. <http://www.virustotal.com/>
- [2] WildList. <http://www.wildlist.org>
- [3] PEID. <http://www.peid.info>
- [4] AT4RE(Arab Team 4 Reverse Engineering). <http://www.at4re.com>

제 3 장 실행 압축 해제 방법 조사

제 1 절 실행 압축 해제 연구 동향 조사

1. 실행 압축 도구 탐지 동향

가. PE File Header Analysis-based Packed PE File Detection Technique(PHAD)[1]

본 논문에서는 악성코드가 실행 압축된 경우에 압축된 실행코드를 실행 압축 해제를 하여야 악성코드를 분석을 할 수 있기 때문에 실행 압축 해제 전에 실행파일의 실행 압축 여부를 탐지하는 것이 중요하다고 설명하고 있다. 그리고 실행 압축을 탐지하기 위해 Windows PE(portable executable) 파일의 헤더 정보를 이용하여 실행 압축된 파일을 탐지하는 PHAD 방법을 제안한다.

PHAD 는 실행 압축된 파일의 경우 압축된 실행파일을 위해 code 섹션에 실행과 쓰기 속성을 가지게 된다. 하지만 일반 파일의 경우 위의 두 가지 속성을 가지는 경우가 없기 때문에 이와 같은 특징을 이용하여 실행 압축된 파일과 일반 파일의 PE Header를 비교 분석하여 실행 압축 탐지에 필요한 특징 8개를 선택하였다.

[표 1] PHAD에서 사용한 PE Header 의 특징 값

특징 벡터 값	설명
$V_{F,1}$	Executable & Writable 섹션의 수
$V_{F,2}$	Executable 이지만 code가 아니거나 Executable이 아니지만 code인 섹션의 수
$V_{F,3}$	이름이 없는 섹션의 수
$V_{F,4}$	Executable 섹션이 없다면 1
$V_{F,5}$	모든 섹션의 합이 파일 크기보다 크다면 1
$V_{F,6}$	PE Signature의 위치가 IMAGE_DOS_HEADER의 사

	이즈 보다 작을 경우 1
$V_{F,7}$	entrypoint의 섹션이 executable 하지 않으면 1
$V_{F,8}$	entrypoint의 섹션이 코드가 아니면 1

특징들의 값을 (V)로 정의 하고, 파일 F에서 V의 값을 CV(Characteristic Vector)로 하여 다음과 같이 정의 하였다.

$$CV_F = \{V_{F,1}, V_{F,2}, \dots, V_{F,8}\}$$

위에서 선택한 특징을 이용하여 100개의 실행 압축된 파일과 100개의 일반 파일의 특징 값을 계산하여 다음과 같은 결과를 산출하였다.

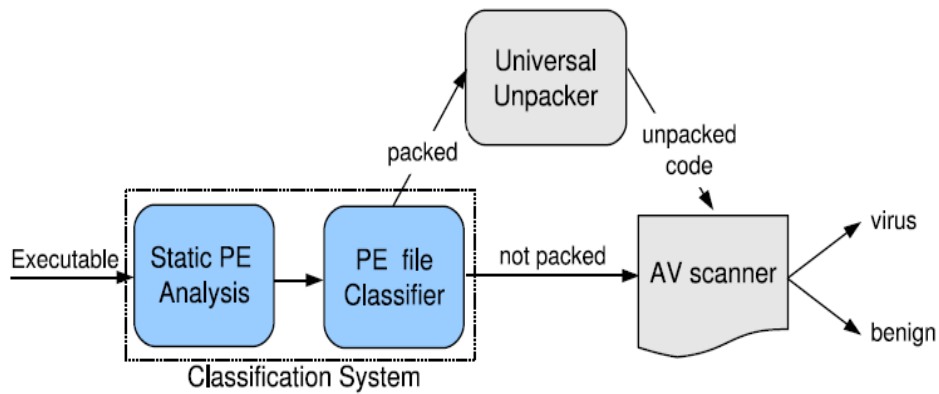
[표 2] 특징 값들의 평균

	$V_{F,1}$	$V_{F,2}$	$V_{F,3}$	$V_{F,4}$	$V_{F,5}$	$V_{F,6}$	$V_{F,7}$	$V_{F,8}$
실행 압축된 파일	2.56	1.36	1.41	0.17	0.08	0.15	0.27	0.51
일반파일	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

위 결과와 같이 일반파일의 경우 특징 값이 없는 점을 이용하여 실행 압축된 100개의 파일의 특징 값을 각각 구하여 Euclidean Distance로 특징 값들을 계산하고 계산된 Euclidean Distance 값들 중에서 가장 작은 값을 임계 값으로 하여 실행 압축 파일과 일반 파일을 탐지하는 방법을 제안 하였다.

나. Classification of Packed Executables for accurate computer virus detection[2]

본 논문은 실행 압축된 실행파일을 실행 압축된 것과 실행 압축되지 않은 것, 2-클래스로 분류할 때의 계산 비용과 정확도에 초점을 맞추었다. 즉, 실행 압축된 실행파일의 빠른 탐지와 효과적이고 정확한 분류에 초점을 맞추었다.



(그림 21) 논문에서 제안하는 실행 압축 여부 분류기 (파란색 부분)

(그림 24)은 본 논문에서 제안한 실행 압축 여부 분류 시스템을 보여준다.

실행파일을 분류하기 위해서, 코드의 이름, 데이터 섹션, 쓰기 가능한 실행 섹션의 개수, 코드와 데이터의 엔트로피 등을 추출하기 위해서 바이너리 정적 분석 방법을 사용하였다. 이 정보들은 패턴인식 기술을 위한 패턴 벡터로 사용되었다. 파일이 실행 압축된 경우 Universal Unpacker로 실행 파일을 실행 압축 해제하여, 안티바이러스 스캐너로 보내 바이러스 여부를 검사할 수 있다.

본 논문에서는 실행 압축된 파일의 판단을 위한 속성 값을 아래와 같이 정리 하였다.

[표 3] 분류를 위한 속성 값

번호	속성	값의 범위
1	Number of standard sections	integer, 0 이상
2	Number of non-standard sections	integer, 0 이상
3	Number of Executable sections	integer, 0 이상
4	Number of Readable/Writable/Executable Section	integer, 0 이상
5	Number of entries in the IAT	integer, 0 이상. IAT가 없을 경우는 -1
6	Entropy of the PE header	[0, 8]
7	Entropy of the code sections	[0, 8] 혹은 코드섹션이 없는 경우 -1
8	Entropy of the data sections	[0, 8] 혹은 데이터 섹션이 없는 경우 -1
9	Entropy of the entire PE file	[0, 8]

1.2. 일반적으로 컴파일 한 프로그램의 경우 standard section 이름 규칙을 따르게 되고, 실행 압축된 파일과 같은 경우에는 해당 실행 압축 프로그램에서 지정하는 섹션명을 사용하는 경우가 많다.

- Standard sections : .text, .data, .rsrc 등의 이름으로 정의된 섹션
- non-standard section : UPX의 경우, .UPX0, .UPX1 등

3. 몇몇 실행 압축된 파일들은 executable section을 가지지 않는다.

4. 실행 압축된 파일은 실행 시 원래의 실행파일로 실행 압축 해제되고, 이를 실행하기 위한 readable/writable/executable section이 적어도 하나는 필요하다. 반면, 일반적으로 executable section(.text)는 쓰기 가능한 기능이 필요 없으며, 해당 플래그역시 설정되지 않는다.

5. 대부분의 실행 압축되지 않은 정상 파일들은 네이티브 윈도우 API 등을 호출하기 위해 IAT가 존재한다. 반면 실행 압축된 파일들은 매우 적은 수의 외부 함수만을 가져오는 경우가 있다. 이는 실행 압축 해제

알고리즘 상에서 외부 함수들을 많이 요구하지 않기 때문이다.

6,7,8,9. 암호화된 코드/데이터의 경우 이를 랜덤하다고 볼 수 있으며, 암호화되지 않은 코드의 경우에는 구조화된 정보라고 말할 수 있다. 따라서 본 연구에서는 PE 파일내의 데이터/코드 섹션의 바이트단위 엔트로피를 구하였다.

몇몇 실행 압축 툴은 암호화된 코드를 PE 헤더의 사용하지 않는 부분에 숨긴다. 또한 PE 파일은 매우 복잡하고 사용하지 않는 다른 여러 공간들을 포함한다. 따라서 PE 헤더와 파일 전체에 대한 엔트로피 역시 구하였다.

위의 속성 값을 데이터 마이닝 도구 Weka를 사용하여 6가지 서로 다른 알고리즘이 적용하였다.

[표 4] 실험 결과

번호	분류기	CV AUC	CV 정확도 (%)	테스트 정확도 (%)
1	Naive Bayes	0.9917(0.0038)	98.42(0.4913)	97.11
2	J48 의사결정 트리 (C4.8)	0.9958(0.0034)	99.57(0.3032)	97.01
3	Bagged-J48	0.9994(0.0010)	99.59(0.3086)	96.82
4	kNN	0.9994(0.0008)	99.43(0.3486)	95.62
5	Multi Layer Perceptron (MLP)	0.9995(0.0008)	99.42(0.3966)	98.91
6	Entropy threshold	0.9766	96.57	91.74

※ Cross Validation (CV)

※ AUC : score로써의 파일 엔트로피 값과

Wilcoxon-Mann-Whitney statistic

※ 괄호 안은 표준 편차를 의미함

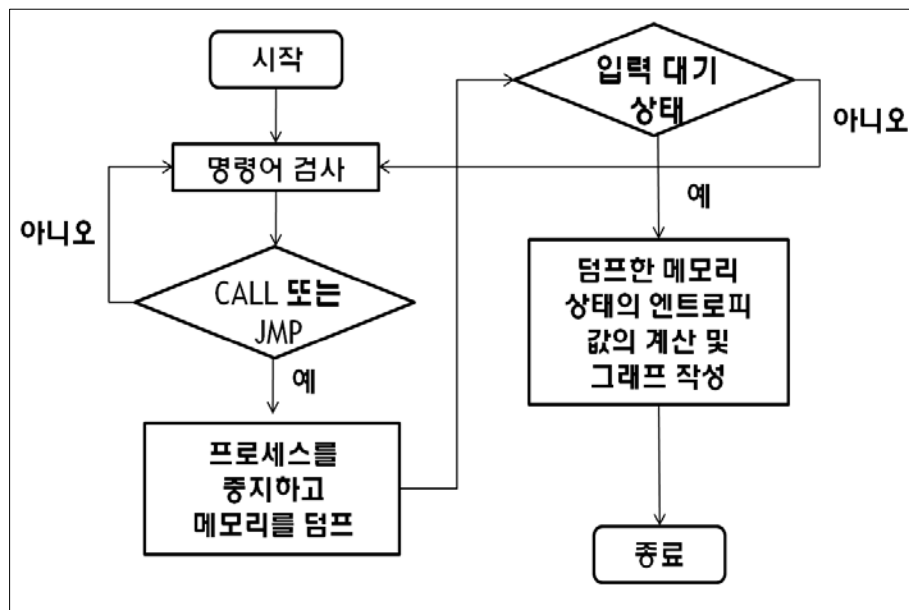
2. 실행 압축 해제 동향

가. 엔트로피를 이용한 실행 압축 해제 기법 연구[3]

본 논문에서는 통계적인 무질서도를 나타내는 엔트로피의 개념을 차용하여 정보 엔트로피의 개념으로 정보의 양을 수치화하고 그 수치를 통하여 실행 압축 기법의 특징점을 찾아 실행 압축이 되었는지 아닌지 구별하고 그 결과를 바탕으로 실행 압축 해제를 수행하게 된다.

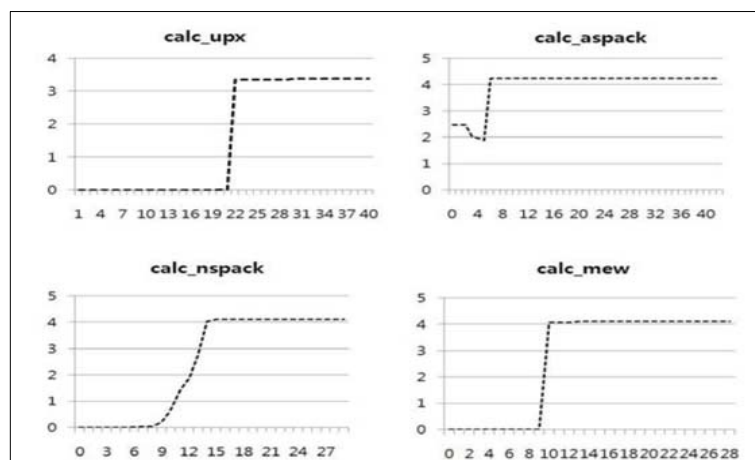
여기서 제안되는 기법은 실행 압축 파일을 실행하면 메모리에 값을 읽고 쓰면서 실행 압축 해제 과정을 거치게 되는데 이 과정에서 메모리 상태의 엔트로피 값 변화를 관찰하여 오리지널 엔트리 포인트를 찾아내는 기법이다. 실행 압축된 파일이 일반 실행 파일에 비해 높은 엔트로피 값을 가지므로 압축 해제가 진행되는 초기 단계에는 실행 압축된 코드로 인해서 해당 프로세스 메모리의 엔트로피 값이 크고 해제하는 과정을 거치면서 점차 메모리의 엔트로피 값이 작아진다는 특징을 이용하여 프로세스를 실행하면서 그 상태를 분석하여 오리지널 엔트로피 값을 찾게 된다.

다음 그림은 해당 기법에서 제안하는 압축 해제 기법의 흐름도이다.



(그림 22) 엔트로피를 이용한 실행 압축 해제 기법의 흐름도

전체적인 엔트로피 변화의 추이를 살피기 위해 타겟 프로세스에 대하여 명령어 단위의 실행을 해야 한다. 하지만 엔트로피의 측정을 명령어가 실행되는 때 순간 수행하는 것은 비효율적이다. 오리지널 엔트리 포인트는 분기문 이후에 나오기 때문에 해당 기법에서는 명령어 JMP 또는 CALL 계열의 명령어가 실행되었을 경우에만 지정된 메모리의 엔트로피 값을 계산한다. 실행 압축 해제 과정을 통해 원래 파일의 데이터와 코드를 다 쓰고 나면 더 이상 관찰하고 있는 영역의 엔트로피는 변하지 않고 고정된다. 엔트로피 값이 고정된 이후 관찰 영역으로 실행 흐름이 옮겨지면 실행 흐름이 옮겨진 주소가 오리지널 엔트리 포인트일 확률이 높기 때문에 오리지널 엔트리 포인트를 찾을 수 있게 된다.



(그림 23) 실행 압축 기법 별 실행 압축 파일의 엔트로피 변화

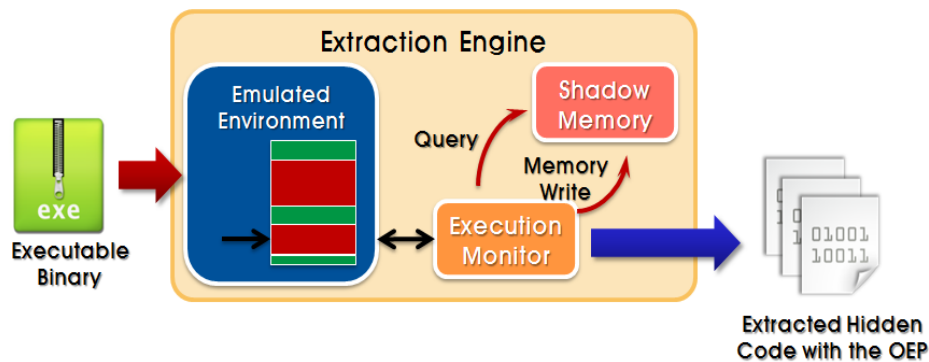
위의 그림은 실행 압축 기법 별 실행 압축 파일의 엔트로피 변화 그래프 그림으로 다음과 같이 엔트로피의 값이 변하지 않는 구간이 나타날 때 그 시점에서 실행 흐름이 변경되는 주소를 찾게 되면 오리지널 엔트리 포인트를 찾을 수 있게 된다.

이 기법의 장점은 실행 압축 알고리즘에 의존하지 않고 엔트로피 분석을 이용하여 압축 해제를 함으로써 알려지지 않은 기법으로 압축된 실행 파일에 대해서도 효율적으로 실행 압축 해제를 할 수 있다는 장점이 있다.

나. Renovo[4]

이 논문은 Carnegie Mellon 대학에서 2007년도에 발표된 논문이다. 논문은 실행 압축된 실행파일의 오리지널 엔트리 포인트를 탐지하고 Original Hidden code와 데이터를 추출하기 위한 매커니즘을 제안하였다. 일반적으로 실행 압축된 실행파일이 실행되면 프로그램 내의 실행 압축 해제 과정이 실행되면서 실행 압축된 데이터를 오리지널 코드로 복원하여 저장한다. 오리지널 코드의 복원이 끝났을 때 오리지널 코드가 실행될 수 있도록 복원된 프로그램의 시작 지점으로 오리지널 엔트리 포인트를 설정해준다. 약 80% 이상의 악성코드 샘플들이 분석을 어렵게 하기 위하여 이러한 과정을 통해 이루지고 있다.

이 논문에서는 안티 리버싱 기술에 관한 부분은 제외하고 실행 압축된 프로그램의 오리지널 코드의 추출에 대해서만 다루고 있다.



(그림 24) Renovo의 구조

위의 그림은 Renovo의 전체적인 구조를 보여주고 있다. Renovo는 아래와 같은 일련의 과정을 통하여 오리지널 코드와 데이터를 추출해 낸다.

1. 프로그램이 메모리에 로드 되면 메모리 맵을 만들고 아무 데이터도 없도록 설정한다.
2. 프로그램이 메모리에 쓰는 명령 예를 들어 mov나 push를 했을 때 목

적지 메모리를 dirty로 표시한다.

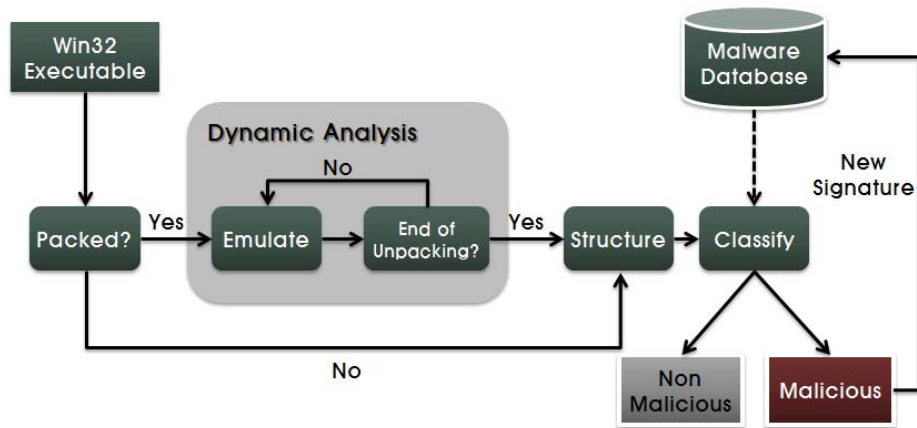
3. Instruction pointer가 새롭게 생성된 지역으로 점프하면 그곳을 오리 지널 프로그램 코드를 숨기고 있는 hidden layer인지 확인하고 Instruction pointer가 가리키는 곳이 OEP인지 결정한다.
4. 나중에 나타날 수 있는 hidden layer를 다루기 위해 메모리 맵을 다시 깨끗하게 만들고 현재 hidden layer에서 추출되는 모든 데이터는 저장 한다.
5. 프로그램이 실행이 끝날 때까지 이 작업을 반복한다.

Renovo는 동적 분석 컴포넌트인 TEMU[5]를 기반으로 동작한다. 가상 머신에서 작동하기 때문에 분석 도구가 원래의 시스템으로부터 격리되어 감염의 위험성이 낮아진다. 논문에서 Universal PE Unpacker(UUnP)와 PolyUnpack[6]이랑 비교한 결과 시간이나 분석 결과에서 우수한 성능을 보였다.

다. Classification of Malware Using Structured control Flow[15]

이 논문에서는 역변환(Decompliation) 기술을 사용하여 제어 흐름 그래프(Control Flow Graph) 시그니처를 구성하는 알고리즘을 제안한다. 실행 압축 된 악성코드를 분석하기 위해서 어플리케이션 레벨의 빠른 예물레이 션을 사용하였다.

논문에서 악성코드를 분류하기 위해서 동적분석과 정적분석의 방법을 모두 사용하였다. 실행파일이 압축되어 있는지를 판단하기 위해 먼저 엔트 로피 분석방법을 사용하였고, 만약에 압축되어 있다면 동적분석을 통해 압 축해제가 끝난 지점을 탐지하여 숨겨진 코드를 파악한다. 그 다음으로 정 적분석을 사용하여 특징을 찾아내고, 구조화 어플리케이션을 이용해 제어 흐름 그래프를 위한 시그니처를 만든다. 이러한 시그니처는 알려진 악성코 드의 데이터베이스를 찾기 위한 정확한 사전기반 검색에 사용된다. 이러한 악성코드 분석의 흐름도는 다음의 그림과 같다.



(그림 25) 악성코드 분석 흐름도

논문에서는 실행압축 된 악성코드의 코드를 분석하기 위해서 OEP를 탐지하여 압축이 풀린 코드를 찾아내는 방식을 사용하였다. 이를 위해서 Renovo에서 제안한 OEP탐지 방법을 향상시킨 알고리즘을 사용하였다. Renovo에서 제안한 OEP탐지 방법은 프로그램 실행 동안 메모리 접근을 기록할 수 있는 Shadow 메모리를 사용하여 재접근이 이루어지면 이를 OEP로 파악하는 방식이다. 하지만 만약 여러 번의 실행압축이 된 악성코드일 경우 한번의 OEP탐지 과정이 끝난 후 다시 Shadow 메모리를 초기화한 후, 에뮬레이션하고 모니터링 하는 작업을 일정 Timeout이 될 때까지 계속 반복해야만 한다. 하지만 논문에서는 엔트로피 분석 방법을 추가로 도입해서, 한번 메모리에 재접근이 이루어 졌을 때 Timeout까지 기다리지 않고, 새롭게 써지거나 사용된 적이 없는 메모리의 엔트로피 값 분석을 통하여 실행압축 해제가 완전히 끝났는지 여부를 확인한다.

논문의 시스템을 평가를 위해서 14개의 알려진 실행압축 도구를 이용하였다. 이 도구들은 압축, 암호화, 코드 불명료화, 디버거 탐지, VM탐지 기능을 제공하고 있다. 샘플은 윈도우 XP 시스템의 hostname.exe 와 calc.exe를 이용하였다.

실험결과 대부분의 실행압축 된 샘플들의 숨겨진 코드를 파악하는데 성공하였다. 실행압축 종류의 하나인 'Pepsin'은 OEP를 탐지하는데 실패하였는데, 이는 실행압축 해제 과정에서 사용하지 않는 암호화된 데이터가

엔트로피 수치를 높이기 때문에 잘못된 OEP를 탐지한 것으로 판단된다.

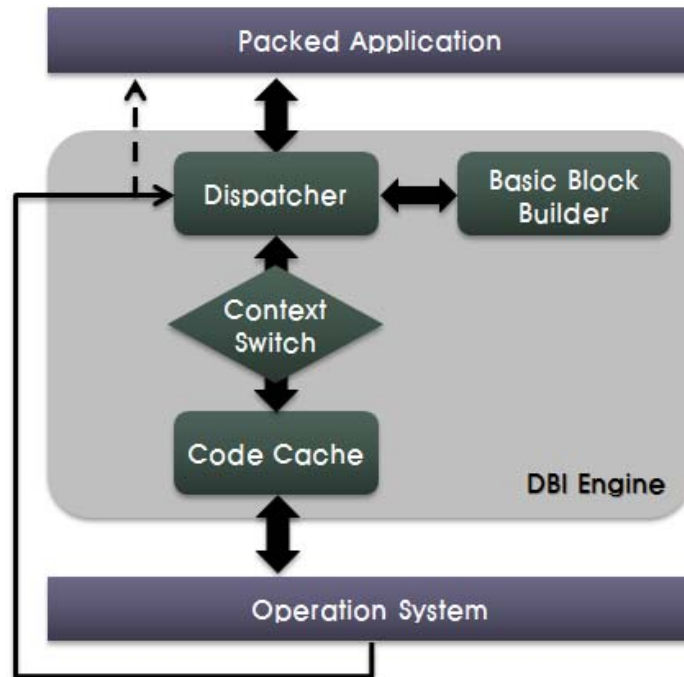
라. Generic Unpacking of Self-modifying, Aggressive, Packed Binary Programs[16]

이 논문에서는 자기수정(self-modifying) 및 강력한 압축기술을 통한 악성 코드 탐지를 우회하는 바이너리 코드를 분석하기 위한 Dynamic Binary Instrumentation 기술인 MmmBop를 제안하고 있다. MmmBop는 대부분의 실행압축 알고리즘을 처리할 수 있고, 최근에 사용되는 안티리버싱 트릭을 성공적으로 우회할 수 있다.

실행압축된 어플리케이션은 먼저 인젝션 모듈을 통해서 DBI 엔진에 삽입되게 된다. 인젝션 모듈의 주요 작업은 다음과 같다.

1. 목표 어플리케이션의 정지된 프로세스 생성
2. DBI엔진을 프로세스 영역으로 로딩
3. 현재 프로그램의 엔트리 포인트를 DBI 엔진에게 전달
4. 실행파일을 DBI가 생성한 Basic Block에게 전달

전체 인젝션 과정은 실제 물리적인 파일을 수정하지 않고, 가상으로 이루어진다. 인젝션 과정이 모두 끝나면 자기 자신을 종료하고, 목표 프로세스의 실행을 시작한다.



(그림 26) 논문에서 제안하는 DBI 엔진

위의 그림은 DBI 엔진의 전체적인 구조를 나타내고 있다. 논문에서 개발한 DBI엔진은 Code Cache, Basic Block Builder, Context Switch, Dispatcher로 구성된다.

Code Cache는 에뮬레이션을 하지 않고 네이티브 코드를 실행할 수 있도록 한다. 이 방법은 에뮬레이션을 사용할 때 생기는 속도저하를 막을 수 있게 해준다. 캐시된 코드는 원래 어플리케이션 코드와 같은 로직을 포함하고, 유일한 차이점은 DBI엔진이 새로운 Basic Block이 실행되기 전에 계속 제어권을 유지할 수 있도록 해주는 수정된 흐름제어 명령어 부분이다.

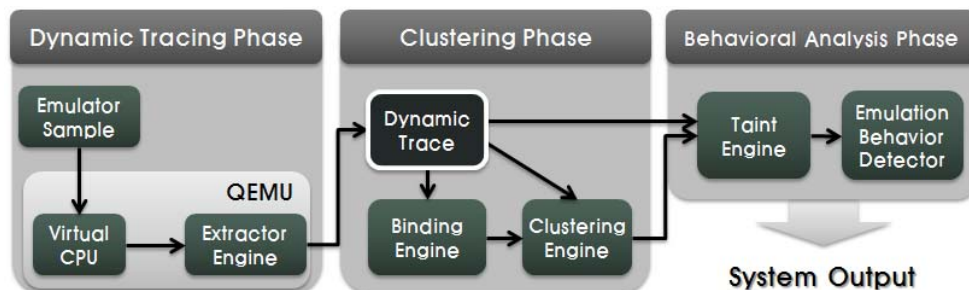
Basic Block Builder는 Basic Block 생성을 담당한다. Basic Block은 프로그램 흐름 제어를 위한 Single point of Entry와 Single Point of Exit를 포함하는 명령의 집합이다. Context Switch는 MmmBop의 내부 메커니즘에서 원래 프로그램의 CPU상태를 분리하기 위해 사용된다.

Dispatcher는 Code Cache에서 이야기 한 것처럼 MmmBop가 계속 제어권을 가질 수 있도록 수정된 흐름제어 명령어를 사용할 수 있도록 담당한다.

다. 또한 안티리버싱 기술을 사용하여 고의적인 예외를 발생시키거나, 실행흐름을 임의로 다른 곳으로 옮기는 식의 방법을 막기 위해서 Exception Dispatcher와 Continue Dispatcher가 사용된다. Exception Dispatcher에서는 예외가 발생하는 상황을 미리 KiUserExceptionDispatcher 함수를 이용한 후킹으로 차단하고 이를 수정한다. 또한 Continue Dispatcher는 실행흐름을 임의로 다른 곳으로 옮기기 위해 쓰레드의 컨텍스트를 간접적으로 수정하는 것을 NtContinue를 후킹하여 처리한다.

마. Rotalume : A Tool for Automatic Reverse Engineering of Malware Emulators[17]

이 논문에서는 에뮬레이션 기술을 사용하는 악성코드를 분석하기 위한 방법을 제시한다. 동적 분석 방법을 이용하여 보호된 환경에서 에뮬레이트된 악성코드를 실행하고 에뮬레이터에 의해 생성된 전체 명령어를 기록한다. 그리고 바이트코드 명령어 집합에 관한 Syntactic & Semantic 정보 추출과 바이트코드 프로그램을 포함하고 있는 데이터 버퍼를 식별하기 위한 동적 데이터흐름(Dynamic Data-Flow)과 감염 분석(Taint Analysis)을 사용한다. 이러한 분석결과와 함께, 제어 흐름 그래프와 같은 이후의 악성코드 분석을 위한 기반을 제공하는 데이터 구조를 생성해 낸다.



(그림 27) Rotalume의 구조

Rotalume의 전체적인 구조는 위의 그림과 같다. 악성코드 분석은 총 세 가지 단계로 나뉘지는데 이는 동적추적 단계, 클러스터링 단계 그리고 행

동 분석 단계이다.

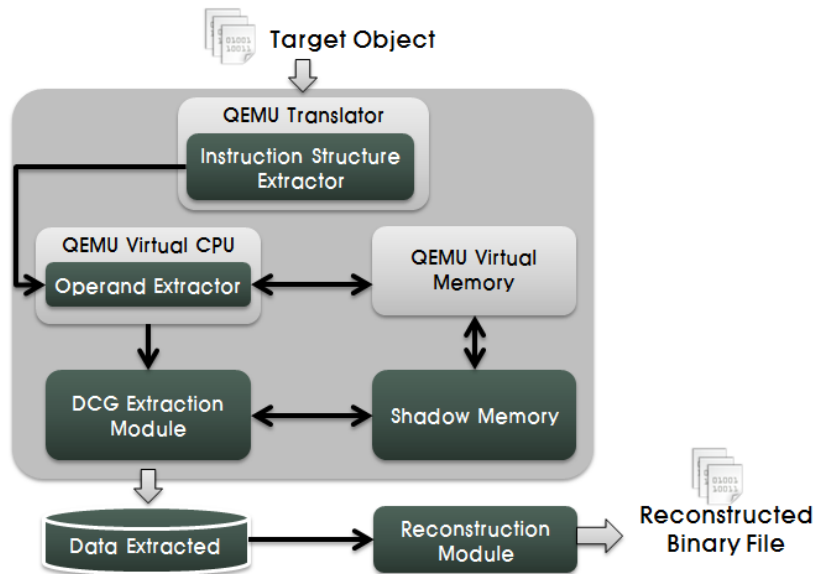
동적 추적 단계에서는 먼저 QEMU Guest OS위에서 실행되고 있는 프로그램의 동적 명령어 추적 정보를 모은다. 이를 가능케 하기 위해, QEMU를 수정하여 실행 시 모든 명령어를 추적할 수 있는 콜백 함수 'Rotalume's Trace Extractor Engine(EE)'을 추가하였다. 이렇게 모인 명령어들을 이용하여 클러스터링 단계에서는 BE(Binding Engine)와 CE(Clustering Engine)를 통해서 그룹화 시킨다. 마지막으로 행동 분석 단계에서는 클러스터링 단계에서 그룹화 된 명령어의 정보를 이용하여, 악성코드가 실제 어떠한 행동을 하는지 분석한다. Taint Engine을 통해서 시스템에 어떠한 변화가 있는지 파악하고, 에뮬레이션 위에서의 악성코드 행동을 파악한 후 제어 흐름 그래프와 같은 최종 결과물을 낸다.

논문의 실험에는 모의프로그램, 압축되지 않은 실제 악성코드, 가상화 기술을 사용한 악성코드와 같이 총 3가지 종류의 프로그램에 대해서 분석하였다. 모든 프로그램은 VMProtect, Themida, Code Virtualizer를 사용하여 임의의 코드 및 함수를 압축하였다. 실험결과 위의 세 가지 종류의 프로그램에서 모두 성공적으로 악성코드를 추출해 내고, 행동을 분석할 수 있었다.

바. ReconBin: Reconstructing Binary File from Execution for Software Analysis[18]

논문에서는 정적 분석을 위해서 암호화되고 압축된 프로그램을 바이너리 파일로 재구성하는 기술을 제안한다. 논문에서 제시하는 방법은 실행 시간 동안 동적으로 명령어 식별 및 압축 해제를 할 수 있고, 동시에 제어 전환 정보를 기록할 수 있으며, 원래의 실행파일에 기반을 둔 바이너리 파일을 재구성할 수 있음을 보여준다. 프로토타입인 ReconBin을 통한 실험은 이 접근방법이 SMC(Self-Modifying Code)와 실행압축으로 보호된 실행파일을 정확히 재구성할 수 있고, 재구성된 바이너리 파일은 IDA Pro와 같은 정적 분석 도구에 의해 성공적으로 분석됨을 보여준다. 또한 가상 머신, 에뮬레이터, 그리고 스택이나 직접적인 실행 흐름에 삽입된 공격 코드로

인한 버퍼 오버플로우 공격에 의해서 동적으로 생성된 코드 분석에 사용될 수 있음을 보여준다.



(그림 28) ReconBin의 구조

위 그림은 ReconBin의 전체적인 구조를 나타낸다. 논문에서 구현한 ReconBin은 QEMU를 기반으로 구현하였다. 또한 동적분석을 통해 명령어를 추출해 내고, 어떠한 메모리 연산이 일어나는지 기록하기 위해서 다음 네 가지의 모듈을 추가하였다.

- QEMU Translator에 명령어 구조 추출(Instruction Structure Extractor) 모듈 추가
- QEMU 가상 CPU에 연산자 추출(Operand Extractor) 모듈 추가
- 명령어 구조의 정보를 저장하는 Translation Block(TB) 확장
- 메모리 연산을 기록하는 모듈인 Shadow memory 추가

동적으로 생성되는 코드를 파악하기 위해서, 논문에서는 동적 생성 코드의 특징을 다음 두 가지로 정의하였다.

- a) 코드는 실행시간 동안 생성되며, 메모리에 쓰여진다.
- b) 메모리에 쓰여진 내용의 전체 혹은 일부분은 실행흐름의 일정 지점에서 실행되게 된다.

이렇게 정의된 특성에 따라, 동적 분석단계에서는 실제 명령어를 순차적으로 추적한 후 다음의 두 가지 타입의 명령어가 나올 경우 이를 분석한다.

- Memory Modification (MOV, MOVS, STOSB, etc.)
- Control Transfer (direct/indirect jump/call, return)

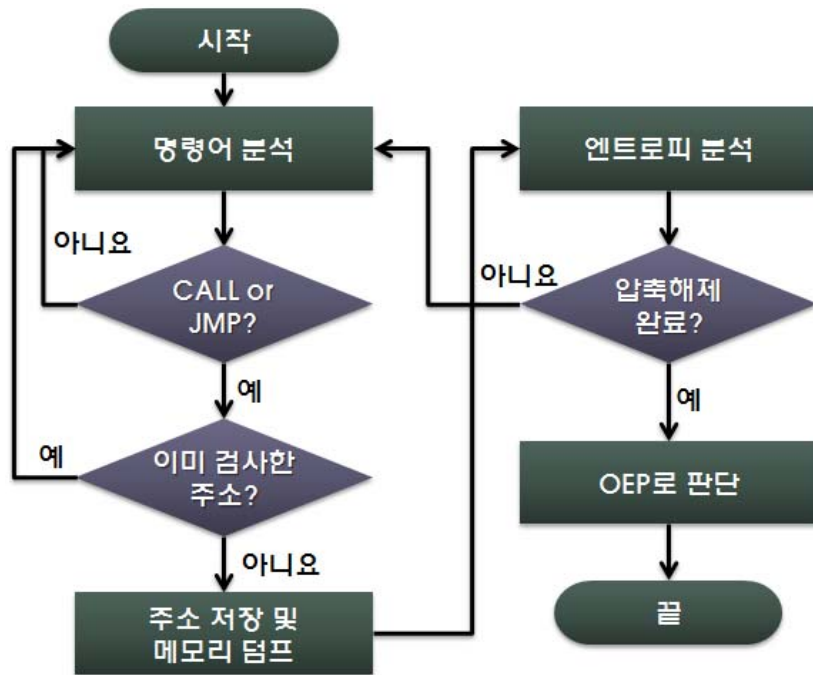
명령어 분석을 위해서 논문에서는 실제 메모리 변화를 기록할 수 있는 Shadow Memory를 만들었다. 이러한 명령어 추적과 Shadow Memory를 이용하여 동적으로 생성된 코드를 파악할 수 있다.

바이너리 파일의 재구성 단계에서는 추출된 코드를 보호된 실행파일로 변환하고, 제어흐름을 복구하여 바이너리 파일을 생성한다. 또한 생성된 실행파일이 다시 자기 자신을 수정하는 것을 방지하기 위해 수정코드를 모두 무효화 시킨다.

사. Generic Unpacking using Entropy Analysis [19]

이 논문에서는 엔트로피 분석을 통해 OEP를 찾아내어 실행압축을 해제할 수 있는 방법을 제시하고, 더 나아가 실행압축이 풀리는 과정에서의 엔트로피 수치 변화량을 이용하여 실행압축이 풀리는 알고리즘을 몇 개의 카테고리로 분류 할 수 있음을 보여준다.

OEP 탐지는 실행압축이 풀리기 전후의 엔트로피 수치에 변화가 있는 점을 이용하여, 실행압축이 풀리는 과정에서 엔트로피 수치를 관찰하여 더 이상 변화가 없는 시점을 실행압축이 완전히 풀린 시점으로 간주한다. 구체적인 OEP탐지 과정은 다음과 같다.



(그림 29) OEP탐지 흐름도

실행압축 된 프로그램을 실행시키고, 특정 명령어(JMP, JCC, CALL, RET)가 나오는 위치에서 프로그램을 중단 후 메모리의 엔트로피 분석을 수행한다. 특정 명령어가 나올 때만 수행하는 이유는 실행압축이 모두 풀린 시점에서 OEP로 실행흐름이 옮겨 갈 때 주로 특정 명령어들이 나오기 때문이다.

엔트로피 분석은 프로그램 각 섹션의 엔트로피 값 분석을 통해 실행압축 해제가 모두 끝났는지를 검사하고, 압축이 풀린 코드가 쓰여진 섹션의 주소로 명령이 옮겨졌는지 검사한다. 실행압축 해제가 모두 끝났다면 이 위치를 OEP로 판단하고, 아직 풀리는 중이면 중단된 프로세스를 다시 진행시킨다.

실행압축이 모두 풀린 시점의 엔트로피 값을 정하기 위해서, 논문에서는 실험을 통해 엔트로피 값의 최대치(E_{max})와 최소치(E_{min})를 구했다. 엔트로피 분석을 할 때, 이렇게 구한 영역에 엔트로피 값이 포함된다면 이 때를 압축이 모두 풀린 시점으로 간주한다. 또한, 엔트로피 값 측정은 최대한 정

확도를 높이기 위해서 code section의 엔트로피 수치만 측정하였다.

실행압축을 푸는 모듈은 보통 다수의 반복문으로 구성되어 있다. 이러한 반복문은 OEP를 탐지하기 위한 분석 시간을 증가시킨다. 반복문을 통한 분석시간 증가의 문제를 해결하기 위해서 분석 중에 나왔던 JMP나 JCC명령어의 주소를 기록해두고, 만약 다시 이 주소로 재접근이 일어날 경우 엔트로피 분석을 건너뛰는 방법인 Cache를 사용하였다.

OEP탐지 실험은 윈도우 폴더 안에 있는 임의의 11개 실행파일을 10종류의 실행압축 프로그램을 이용하여 총 110개의 파일에 대해서 실험을 수행하였다. 실험 결과 72%의 경우에 완벽하게 OEP를 탐지해 냈으며, 6%의 경우는 잘못 탐지했으며, 나머지 22%는 탐지에 실패하였다.

잘못 탐지한 경우는 실행압축 프로그램 중 'mpress'를 사용한 경우에 주로 발생했는데, 이는 mpress의 실행압축 해제 알고리즘이 실행압축을 거의 해제하기 직전에 OEP가 아닌 실행압축이 해제된 코드 중 임의의 지점으로 실행흐름이 옮겨지기 때문에 발생한다. 22%의 실패는 OEP에 도달한 시점의 엔트로피 수치가 E_{max} 와 E_{min} 사이에 있지 않을 경우 발생한다.

추가적인 부분으로 실험을 통해서 실행압축이 해제되는 과정의 엔트로피 값 변화의 패턴을 분석한 결과, 실행압축이 해제되는 방식에 따라 분류할 수 있음을 알아냈다. 대분류로 TYPE I, II로 나누고, 이를 다시 세 가지(i, ii, iii)로 나누었다.

- TYPE I : 실행압축이 풀리는 동안 엔트로피 값이 증가
- TYPE II : 실행압축이 풀리는 동안 엔트로피 값이 감소

- i : 일정하게 증가 (TYPE II의 경우 감소)
- ii : 감소하다가 증가 (TYPE II의 경우 증가하다가 감소)
- iii : 급격하게 증가 (TYPE II의 경우 급격하게 감소)

아. Reverse Engineering Self-Modifying Code : Unpacker Extraction

[20]

현재 대부분의 실행압축 된 악성코드 분석에 관한 연구는 실행압축 여부를 탐지하고, 실행압축이 해제된 코드를 추출하는데 집중하고 있다. 하지만 이는 실행압축 해제 코드가 실제 악성코드와 분리되어 있지 않는 상황에서는 적용하기 힘들다는 단점이 있다. 논문에서는 이러한 문제를 해결하기 위해 실행압축을 해제하는 코드 추출에 초점을 맞추었다.

실행압축 해제에 관한 기존의 연구는 실행압축 해제 과정은 프로그램이 시작된 후 바로 수행되고, 모두 완료된 후에 실제 압축이 풀린 코드가 실행된다고 가정하고 있다. 하지만 실행압축 해제 과정을 조금만 분산시켜 놓으면 기존의 연구는 모두 적용할 수 없게 된다. 예를 들면, 실행압축 해제 코드가 실제 악성코드와 합쳐져 있어서 일정 부분까지만 압축을 풀고, 압축이 풀린 악성코드를 실행한 후 다시 압축을 푸는 과정을 반복할 수 있다. 이러한 기술을 사용한 악성코드는 기존의 접근방법으로는 정확하게 압축이 풀린 원래의 악성코드만 추출해 낼 수 없게 된다.

기존의 접근방식에 따른 문제점을 해결하기 위해 논문에서는 직접 실행압축 해제 코드를 추출해 내는 방법을 제안하였다. 실행압축 해제 코드를 추출함에 따른 장점은 다음과 같다.

- 압축이 풀린 원래의 코드에 초점을 맞출 수 있도록 리버스 엔지니어링 시간을 단축
- 실행압축 해제 기술에 상관없이 정확한 악성코드 분류 가능
- 악성코드의 행동 분석에 용이
- 실행압축 해제 코드 자체 분석 가능

실행압축 해제 코드를 추출하는 과정은 3단계로 나뉜다. 첫 번째 단계는 압축해제 및 실행과정을 논문에서 정한 임의의 단계(Phase)로 분류하고, 각 단계에서 압축이 풀린 명령어를 식별해 낸다. 두 번째로 압축이 풀린 명령어 중 다른 코드를 수정하는 명령어(Modifier)를 찾아낸다. 마지막으로 Slicing 기술을 이용하여 실행압축 해제 코드를 추출해 낸다.

실행압축 해제 단계(Phase) 및 압축이 풀린 코드를 식별하기 위해서 한번의 실행을 통해서 일단 명령들을 수집하고, 수집된 명령을 분석하여 각각의 단계(Phase)로 나눈다. 식별과정에서 다음의 두 메모리위치 세트를 저장한다.

- GlobalWriteSet : 프로그램의 시작부터 수정된 메모리의 위치를 기록
- CurrWriteSet : 현재 단계(Phase)에서 지금까지 수정된 메모리 위치를 기록

각각의 단계를 차례로 추적하면서 만약에 각 단계가 GlobalWriteSet와 일치한다면 그 단계의 코드 실행압축 해제된 것이고, CurrWriteSet과 일치한다면 새로운 단계의 시작을 가리키는 것이다. 이러한 방법을 통해서 단계와 압축이 풀린 코드를 식별한다.

이렇게 식별된 압축이 풀린 코드를 보고 다음 단계에서는 수정 명령어(Modifier)를 식별한다. 다른 위치의 코드를 수정하는 명령어의 식별은 수집된 명령어를 거꾸로 추적하면서 행해진다. 메모리에 쓰기 연산을 하는 명령이 다음에 나올 위치에 기록을 하게 되면 이를 수정하는 명령어로 판단하게 된다.

위의 두 단계를 통해서 파악한 실행압축이 풀린 코드와 수정명령어를 이용하여 실제로 실행압축을 해제하는 코드가 무엇인지 정확하게 파악한다. 기존의 동적 슬라이싱 기술은 자기수정코드에 적용할 수 없기 때문에, 논문에서는 자기수정코드에 적용할 수 있는 동적 슬라이싱 기술을 개발하여 각 단계의 코드 의존도를 파악하였다. 이러한 관계를 통해 실제 실행압축 해제에 직접적인 영향을 미치지 않는 명령어는 제거함으로써 정확한 실행압축 해제 코드 추출을 할 수 있다.

실험에는 다음 4개의 악성코드를 이용하였다. 악성코드의 종류와 사용된 실행압축 기술은 다음과 같다.

- Breatle.J : custom packer
- Hybris.C : custom packer
- Mydoom.Q : UPX
- Netsky.AA : Aspack and UPX

실험의 안전을 위해 VMWare 가상머신 위에서 실험을 하였고, OllyDbg를 이용하여 명령어를 수집하였다. 실험결과 수정 명령어와 명령어 간의 의존성을 성공적으로 파악할 수 있었고, 정확한 동적 슬라이싱 계산을 통하여 실행압축 해제 코드를 추출하는데 성공하였다.

자. Detection of Polymorphic Viruses in Windows Executables

다형성을 가진 바이러스는 실행시간 동안 자기 자신의 압축을 풀고, 다른 파일을 감염시킬 때 새로운 형태로 변이시킨다. 이 논문에서는 이러한 바이러스를 탐지하기 위해서 에뮬레이션 기술을 이용한 탐지 기술을 소개한다. 주 목적은 탐지율을 향상시키고 오탐율을 낮추는 것이다. 논문에서는 이러한 탐지를 위해 Bochs라는 에뮬레이터를 사용하였다.

압축되거나 암호화된 코드의 복호화 과정은 모두 에뮬레이터 위에서 이루어진다. 실행파일은 먼저 에뮬레이터를 통해 한 번에 한 명령어씩 실행되고, 모든 명령은 메모리 쓰기를 수행하는지 검사된다. 만약에 메모리에 쓰여지는 명령이 수행된다면 주소를 표시해 둔다. 한번 이상 수정된 메모리로 분기되는 구문이 발생하면, 복호화가 모두 끝났다고 가정한다. 복호화를 모두 마치면 메모리의 이미지를 가져와서 정적분석을 위해 새로운 실행파일로 구성한다.

또한 이렇게 동적으로 생성된 코드를 파악하기 위해서는 수정된 주소의 리스트를 저장해야 할 필요가 있다. 이를 위해 수정된 주소를 저장할 수 있는 동적 데이터 테이블을 사용하여, 물리적인 쓰기 연산이 일어날 때 마다 테이블을 수정한다.

실험은 실제 6종류의 바이러스 (Antares, cjdisease, Egypt, omokaine,

seraph, rit, spein)를 이용하여 압축이 풀린 코드를 찾아내고, 이렇게 찾아낸 코드를 통해서 몇 개의 안티바이러스 엔진(총 16개)에서 악성코드임을 검출해 낼 수 있는지 실험하였다.

실험 결과 모든 종류의 바이러스에서 압축이 풀린 코드를 찾아 낼 수 있었고, 안티바이러스 엔진에서도 바이러스를 탐지해 낼 수 있었다.

차. Hybrid anlysis and control of malware

대부분의 악성코드 바이너리들은 분석과 제어를 어렵게 하기 위해 코드 실행 압축, 코드 오버라이팅, 프로그램 흐름 교란과 같은 분석 방해 기법을 쓴다. 코드 실행 압축 기술은 악성 코드의 전부나 부분을 압축하고 압축 해제하는 부분을 코드의 패키지로 하여 실행 시에 프로그램의 주소로 압축 해제한다. 이러한 기법은 악성코드의 75%에서 보여지고 있다. 최근의 트렌드는 잘 알려진 악성코드의 사용을 피하고 있고, 그래서 많은 새로운 패킹된 바이너리 코드는 커스텀된 패킹 기술을 쓴다.

더욱 복잡한 이유는 많은 패킹 툴과 악의적인 프로그램들이 실행 중에 오버라이팅을 하기 때문이다. 코드 실행 압축 해제는 정적분석에 영향을 미쳐 완벽하지 못하게 한다. 코드 오버라이팅은 분석을 완전하지 못하게 한다. 정적 분석은 self modifying program에서 적은 정보만을 산출해낸다.

악성코드는 정적 분석 알고리즘에 의한 긴 바이너리 코드의 분석을 부정확하게 하기 위해 control-transfer obfuscation을 이용한다.

이 논문에서 제안하는 방법은 아래와 같다.

Binary code를 찾아내고, 분석하고, 측정하는 알고리즘

- (1) 프로그램을 메모리로 로드하고, Entry Point에서 멈춘다.
- (2) 디버깅 방해 요소를 제거한다.
- (3) Entry Points부터 정적으로 parsing한다.
- (4) 새롭게 생성되는 코드들을 찾아낸다.
- (5) 프로그램을 다시 실행한다.

- (6) 코드 생성 이벤트를 처리하여 새로운 Entry Point를 추가한다.
- (7) (3)과정으로 돌아간다.

여기서 사용된 구체적인 기술들에 대한 내용은 아래와 같다.

○ Parsing.

Parsing 알고리즘의 목적은 정확하게 바이너리 코드를 알아보고 프로그램의 구조를 분석하고, 프로그램의 프로시저 사이의 CFG를 생성하기 위한 것이다.

○ Dynamic Capture

Parsing을 통해 생성된 CFG의 정적 분석을 통해 많은 code가 분석되고, 찾아졌으면 Dynamic Capture 기술을 이용해 정적분석에서 발견되지 않은 코드를 찾는다.

○ Response to Overwritten code

Code Overwrite는 바이너리 코드 분석에 중요한 문제이다. 대부분의 분석 툴은 정적 CFG 표현을 이용하기 때문에 overwritten code를 분석하지 못한다. 이 논문에서는 명령어 실행되기 전에 수정된 명령어들을 찾아내기 위해, 실행된 명령어 바이트가 수정되는 순간 체크하거나, write code가 발생할 때마다 체크한다. 이러한 방법으로 overwritten code가 실행되기 전에 프로그램의 CFG를 업데이트 한다.

○ Signal- and Exception-Handler Analysis

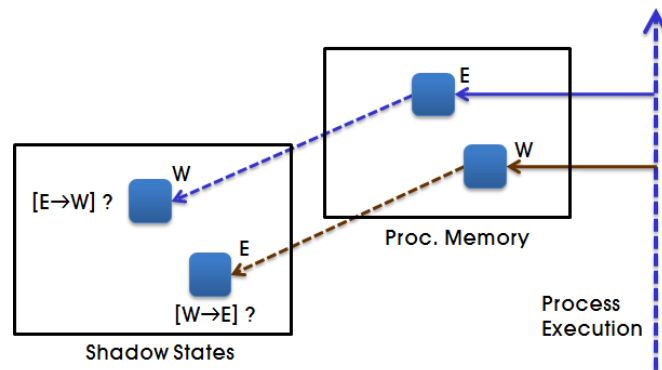
1. 접근 방해(access violation)에 의해 address 0에 저장되는 것과 OS는 Call stack의 fault PC를 저장
 - 1a: OS는 SD-Dyninst에 exception이 일어난 프로세스를 알려준다.
 - 1b: SD-Dyninst는 저장된 exception handler를 분석하고, 명령어의 exit point를 OS로 돌려준다.
- 2: OS는 프로그램의 Exception handler를 call한다.
- 3: handler는 프로그램의 Original Entry point 주소를 저장된 PC에 덮어쓴다.

- 3a: handler의 exit point의 명령어는 SD-Dyninst를 실행한다.
- 3b: SD-dyninst는 수정된 PC값을 찾아내고, 주소의 코드를 분석하고, Handler의 실행을 다시 시작한다.
- 4: Handler는 OS로 리턴한다.
- 5: OS는 수정된 PC값(프로그램의 original Entry Point)으로부터 프로그램의 실행을 다시 시작한다.

카. Toward Generic Unpacking Techniques for Malware Analysis with Quantification of Code Revelation

이 논문에서는 기존의 많은 악성코드들은 실행 압축되어 있고, 이 논문의 연구에서 수집된 643,409개의 악성코드 중 50%가 PEID에서 Packing 됐다고 탐지가 되었다고 하였다. 또한 기존에 많은 논문에서 발표된 Entropy 분석결과 threshold(파일에서 6.7, Section Entropy가 7.2)보다 90%의 샘플이 많게 측정되었다. 그 이유는 Source code를 포함하거나 악성코드 제작자가 악성코드 분석을 피하기 위해 제작했기 때문이다.

이 논문에서 제안하는 방법은 실행 압축/해제를 담당하는 코드를 stub code라고 하고, stub code의 동작을 수량화 하는 방법을 이용하여 실행 압축 해제 여부를 판단한다. stub code가 실행 압축 해제 과정에서 실행되면, 실행 압축된 code가 발견되고 이 발견된 코드의 양을 셀 수 있다고 생각하였다. 이 양을 알아내기 위하여 이 논문은 프로세스 메모리 영역이 실행되어 지는 곳을 알아내기 위한 shadow state를 생성한다. shadow state는 1) 쓰여진 byte가 실행되었느냐?(unpacking에 의한 것) 2) 실행된 byte가 쓰여진 적이 있느냐?(overwriting에 의해 사라진 것)로 나타내진다.



(그림 30) 논문에서 제안하는 프로세스 진행 과정

이 두 가지로 Unpacking에 의해 쓰여진 코드인지를 탐지한다. Shadow memory를 이용해서 stub code의 행동을 탐지하면 OEP 부근에서 정적분석을 위해 dump를 뜯 수 있게 된다. 이 논문에서는 실험을 위해 Intel의 PIN 동적 바이너리 장치를 이용하였다. 일부 data가 쓰여지고 그 byte가 실행이 되면 새로운 byte가 생성된 것이라고 보고 이렇게 쓰여진 byte를 r이라고 한다. 프로그램 실행 과정에서 r이 급격히 증가하는 지점의 instruction pointer를 추적하여 OEP를 찾아낸다.

제 2 절 매뉴얼 실행 압축 해제 기법

실행 압축 해제 도구이나 앞의 실행 압축 해제 기법 및 방법으로 해결이 되지 않는 실행 압축 파일이 있다면 매뉴얼 실행 압축 해제 기법을 통하여 실행 압축 해제하는 수밖에 없다. 매뉴얼 실행 압축 해제 방법은 사람이 직접 실행 압축 분석 도구와 디버거 등을 이용하여 실행 압축 해제 하는 방법으로 매뉴얼 실행 압축 해제 과정을 통해 나오는 특징점이나 특성을 이용하여 해당 실행 압축 기법에 맞는 자동화 실행 압축 해제 도구를 만들거나 플러그인을 제작 할 수 있게 된다. 이 방법은 실행 프로그램의 모든 코드를 사람이 직접 따라가면 분석하는 방식이므로 많은 시간이 소요되는 단점이 존재한다. 하지만 타겟 실행 압축 파일을 실행 압축 해제 하는 자동화 도구나 기법이 존재하지 않는 경우 매뉴얼 실행 압축 해제가 최선의

선택이다.

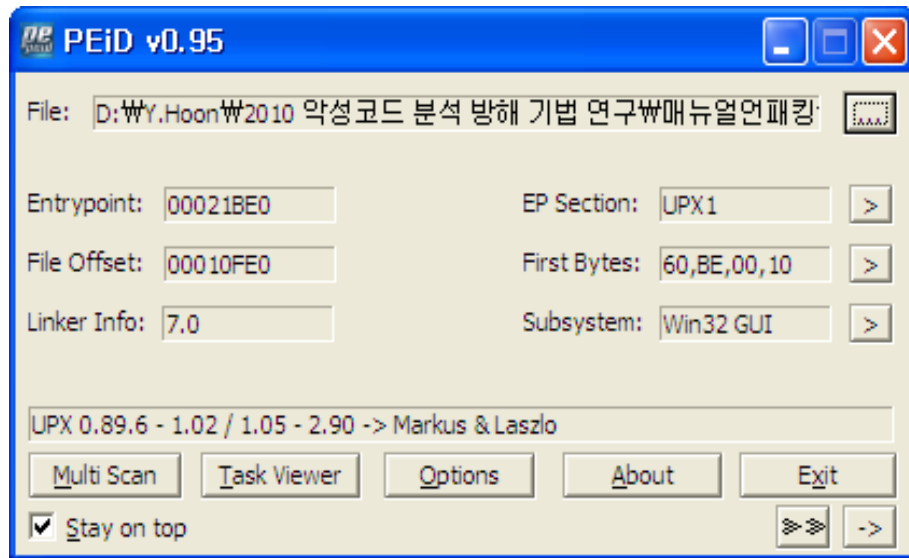
본 절에서는 몇 가지 실행 압축된 파일을 선정하여 매뉴얼 실행 압축 해제 기법에 대하여 알아본다. 매뉴얼 실행 압축 해제 기법은 크게 3가지로 나눌 수 있다.

- 오리지널 엔트리 포인트 위치 확인
- 덤프 파일 생성
- IAT 복구

다음과 같은 매뉴얼 실행 압축 해제 과정을 지나고 나면 모든 실행 압축 파일들은 실행 압축 해제 된다.

1. UPX (Ultimate Packer for eXecutable)

UPX[7]는 가장 많이 쓰이는 실행 압축 프로그램이다. 앞의 통계 자료에서도 나오듯이 전체 실행 압축 기법 중에 약 70%를 차지하는 실행 압축 프로그램이다. UPX는 여러 운영체제에서 수많은 파일 포맷을 지원하는 오픈 소스 실행 압축 프로그램이므로 많은 악성코드 제작자들이 이 UPX를 변형하여 악성코드를 실행 압축 시켜 다양한 탐지 방법을 우회하거나 분석이 어렵게 만들고 있다. UPX에 대한 매뉴얼 실행 압축 해제는 UPX로 직접 실행 압축한 지워찾기 프로그램을 사용한다. 2장의 실행 압축 도구 탐지 방법 중 PEiD 도구를 사용하면 다음 그림과 같이 실행 압축되었음을 알 수 있다.



(그림 31) PEiD로 실행압축 기법 확인

다음과 같이 탐지 도구를 이용하여 탐지 할 수 있지만 탐지 도구가 없다면 실행 압축 도구 탐지 동향에서 나온 방법을 사용하여 탐지 할 수 있다. 특히 UPX는 디버깅 도구를 이용하여 UPX를 분석해보면 메모리 맵 상에서 나오는 UPX0 섹션에는 다음 그림과 같이 데이터가 모두 0으로 채워져 있는 것을 확인 할 수 있다.

Address	Size	Owner	Section	Contains	Type	Access
002F0000	00006000				Map	R E
003B0000	00002000				Map	R E
003C0000	00103000				Map	R
004D0000	00008000				Priv	RW
004E0000	00002000				Map	R E
007E0000	00001000				Priv	RW
007F0000	00001000				Priv	RW
00800000	00002000				Map	R
00810000	00002000				Map	R
00820000	00001000				Priv	RW
01000000	00001000	P305_win		PE header	Imag	R
01001000	00010000	P305_win	UPX0		Imag	R
01011000	00011000	P305_win	UPX1	code	Imag	R
01022000	00007000	P305_win	.rsrc	data,imports	Imag	R
62340000	00001000	LPK		PE header	Imag	R
62341000	00005000	LPK	.text	code,imports	Imag	R
62346000	00001000	LPK	.data	data	Imag	R
62347000	00001000	LPK	.rsrc	resources	Imag	R
62348000	00001000	LPK	.reloc	relocations	Imag	R
73F80000	00001000	USP10		PE header	Imag	R
73F81000	00004000	USP10	.text	code,imports	Imag	R
73FC5000	00010000	USP10	.data	data	Imag	R
73FD5000	00002000	USP10	Shared		Imag	R
73FD7000	00012000	USP10	.rsrc	resources	Imag	R
73FE9000	00002000	USP10	.reloc	relocations	Imag	R
762E0000	00001000	IMM32		PE header	Imag	R
762E1000	00015000	IMM32	.text	code,imports	Imag	R
762FE000	00001000	IMM32	.data	data	Imag	R

(그림 32) UPX0 섹션 확인

일단 UPX로 실행 압축 되었다고 확인이 되었다면 매뉴얼 실행 압축 해제를 시작한다.

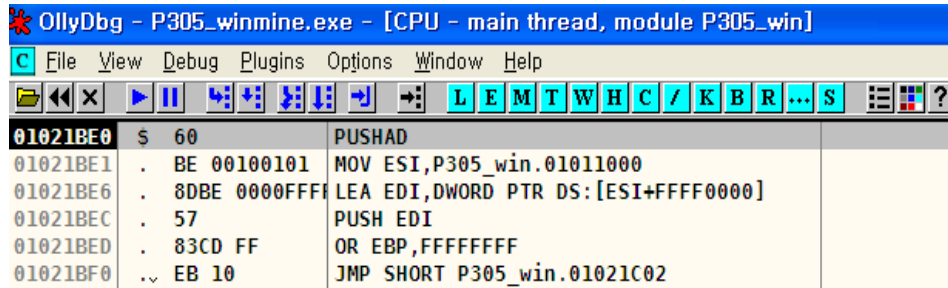
가. 오리지널 엔트리 포인트 찾기

매뉴얼 실행 압축 해제를 하기 위해서는 가장 먼저 해야 할 일이 오리지널 엔트리 포인트를 찾는 것이다. 정확한 오리지널 엔트리 포인트를 찾기 위해서는 프로그램 코드를 한 줄 한 줄 실행하며 오리지널 엔트리 포인트를 찾는 것이 가장 정확한 방법이지만 시간이 오래 걸리고 비효율적이므로 다음과 같은 두 가지 방법을 주로 사용한다.

첫 번째는 자동화 도구를 이용하는 방법이고 두 번째는 실행 압축 기법의 특징과 스택의 원리를 이용하는 것이다. 자동화 도구는 오리지널 엔트리 포인트를 못 찾거나 오탐이 일어날 가능성이 있다 따라서 이번 절에서는 두 번째 방법을 이용하여 매뉴얼 실행 압축 해제를 진행한다.

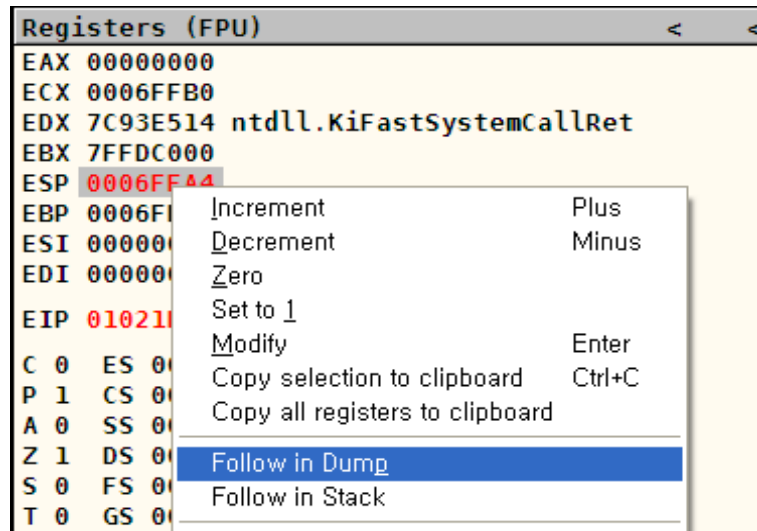
일반적으로 실행 압축 기법들은 실행 압축 해제 과정이 끝나면 오리지널 엔트리 포인트로 진입하기 전에 프로그램 시작 시 저장한 본래의 레지스터 값들을 복구하게 된다. 따라서 이러한 특징을 이용하여 스택에서 레지

스터들이 POP 되는 위치를 찾게 되면 오리지널 엔트리 포인트로 진입하는 부분을 찾을 수 있다. UPX의 경우 프로그램 초기에 PUSHAD 라는 어셈블리 명령어로 모든 레지스터 값을 스택에 저장하는 특성을 가진다.

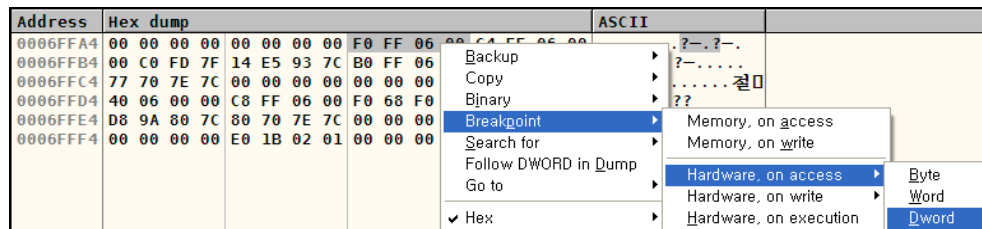


(그림 33) 디버깅 도구를 이용하여 본 실행 압축 PE 파일의 시작 부분

PUSHAD는 DWORD 형태로 레지스터 값을 저장한다. PUSHAD 명령어를 실행한 후 레지스터 값들이 스택에 저장되는데 ESP 스택 포인터는 그 스택의 TOP을 가리키게 된다. 따라서 이 스택 포인터의 값들로 다시 접근이 된다면 그 때가 원래 프로그램의 시작부분인 오리지널 엔트리 포인트가 되는 것이다. 따라서 다음 그림과 같이 해당 위치에 브레이크 포인트를 설정하고 해당 위치에 접근 할 때까지 프로그램을 실행 한다.



(그림 34) ESP 위치 보기 화면



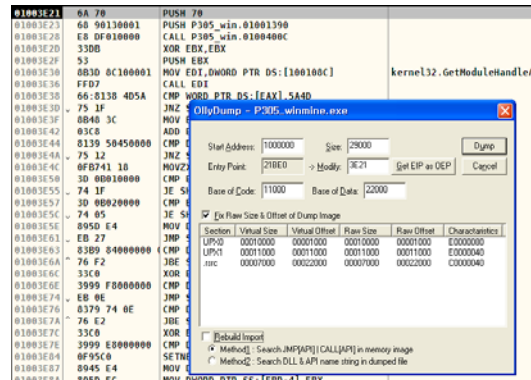
(그림 35) 브레이크 포인트 설정 화면

브레이크 포인트를 설정하고 프로그램을 실행하면 다음 그림과 같이 PUSHAD를 하는 부분과 하단에 나오는 JMP 문을 통하여 오리지널 엔트리 포인트로 접근하는 명령어를 찾을 수 있다.

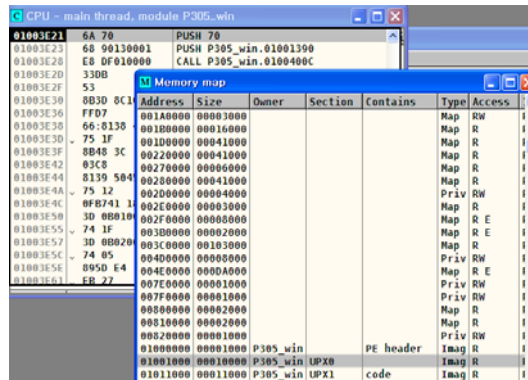
01021D5D	. 61	POPAD
01021D5E	. 8D4424 80	LEA EAX,DWORD PTR SS:[ESP-80]
01021D62	> 6A 00	PUSH 0
01021D64	. 39C4	CMP ESP,EAX
01021D66	. ^ 75 FA	JNZ SHORT P305_win.01021D62
01021D68	. 83EC 80	SUB ESP,-80
01021D6B	.- E9 B120FEFF	JMP P305_win.01003E21

(그림 36) 오리지널 엔트리 포인트 접근

JMP 문을 실행하면 오리지널 엔트리 포인트로 이동하는 것을 확인 할 수 있다.



(그림 37) JMP 실행 후 화면



(그림 38) 메모리 맵과 비교 화면

메모리 맵을 통하여 비교 해보면 초기에 UPX0 섹션은 0값으로 이루어져 있었는데 실행 압축 해제 후에 해당 섹션에 실행 압축 해제된 데이터가 쓰였음을 확인하고 또한 해당 섹션으로 실행 포인터가 옮겨졌으므로 오리지널 엔트리 포인트임을 확신 할 수 있다.

나. 덤프 파일 생성

오리지널 엔트리 포인트를 찾았으니 해당 시점에서 덤프 파일을 생성 한

다. 메모리에 매핑된 이미지를 디스크로 덤프하면 실행 압축된 파일로부터 실행 압축 해제된 본래의 파일을 찾을 수 있다. 이미지 덤프 작업을 해주는 도구는 여러 가지 있지만 여기서는 현재 분석 도구인 OllyDbg[8]에서 제공하는 Ollydump를 사용하여 덤프 파일을 생성한다.

다. IAT 복구

IAT를 복구하는 방법은 수동으로 IAT 찾아서 해당 값을 생성한 덤프 파일에 다시 복원시켜주는 방법과 IAT 복구 도구를 이용하여 복구하는 방법이 존재한다. 먼저 수동으로 복구하는 방법으로는 디버깅 툴의 Disassembler Window에서 Ctrl + B 키를 이용하여 FF25 라는 HEX 값을 Search 하여 찾을 수도 있고, 오리지널 엔트리 포인트를 찾았으니 해당 오리지널 엔트리 포인트 주변을 수동으로 찾아보는 방법이 존재한다.

01001923	-	FF25 0C110001	JMP DWORD PTR DS:[100110C]	USER32.GetSystemMetrics
01001929		8B35 0C110001	MOV ESI,DWORD PTR DS:[100110C]	USER32.GetSystemMetrics
0100192F		6A 4F	PUSH 4F	
01001931		FFD6	CALL ESI	
01001933		85C0	TEST EAX,EAX	
01001935	✓	75 15	JNZ SHORT P305_win.0100194C	
01001937		6A 01	PUSH 1	

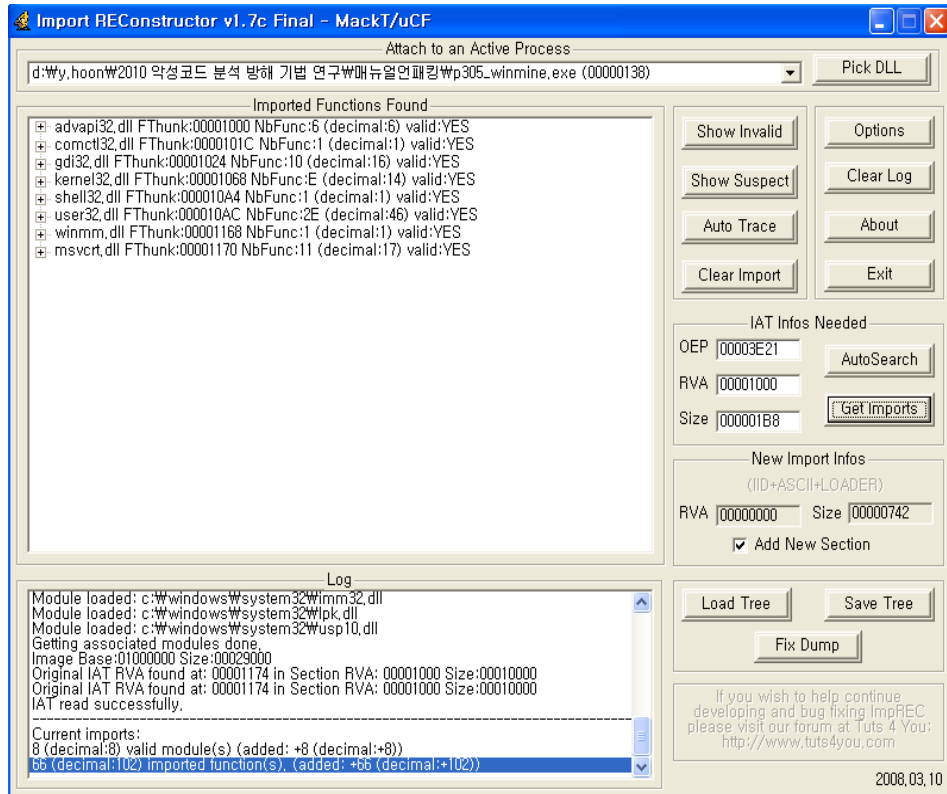
(그림 39) IAT 수동 복구 과정 (IAT 주소값 확인)

Address	Value	Comment
0100110C	7FCFBF9C	USER32.GetSystemMetrics
01001116	7F00BF95	USER32.InvalidDataRecovery
0100111A	7F00BF A6	USER32.SendMessage
0100111B	7F00BF 9E	USER32.ReceiveMessage
0100111C	7F04CC6E	USER32.GetMenuItems
01001120	7F047A80	USER32.GetDlgItemText
01001124	7F00BD20	USER32.DefWindowProcW
01001126	7FCFBF90	USER32.DefWindowProc
0100112C	7FCFBF9C	USER32.DefWindowProc
01001130	7F04AF56	USER32.DefWindowProc
01001134	7F04AF56	USER32.DefWindowProc
01001138	7F04AF5A	USER32.DefWindowProc
0100113C	7FCFBF9C	USER32.DefWindowProc
01001140	7F00BF95	USER32.DefWindowProc
01001144	7F00BF95	USER32.DefWindowProc
01001148	7F00BF95	USER32.DefWindowProc
0100114C	7F00BF95	USER32.DefWindowProc

(그림 40) IAT 수동 복구 과정 (API 함수 확인)

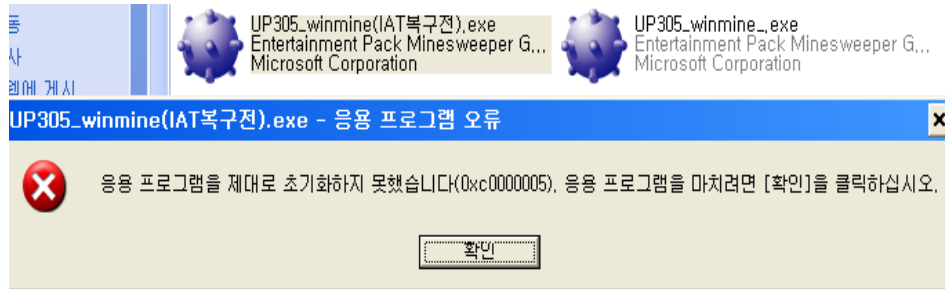
위의 그림에서 보이는 화면 같이 IAT를 찾고 해당 IAT를 수동으로 복원하는 방법은 복구하는 개인의 역량에 따라 찾는 속도가 차이가 나고 경험 이 많을수록 시간이 적게 걸린다.

다음 방법은 IAT Rebuild 툴을 사용하는 방법이다. 대표적인 툴로는 Import REConstructor 라는 툴이 있고 도구를 사용하여 복원하는 방법에서는 Import REConstructor 툴을 이용하여 IAT를 복원하도록 하겠다.



(그림 41) 도구를 이용하여 IAT 복원하는 과정

ImportREC 툴을 이용하여 위의 그림과 같이 디버깅 중인 파일을 Attach 시켜주고 찾은 오리지널 엔트리 포인트 값을 입력한 후에 IAT AutoSearch 버튼을 눌러주고 Get Imports 버튼을 누르면 위와 같이 RVA와 Size를 자동으로 계산하여 주고 해당 위치에서 찾은 API 함수와 dll 파일을 결과물로 나타내어 준다. 나온 결과물을 Fix Dump 버튼을 이용하여 덤프 파일에 변경하고 프로그램을 실행하게 되면 아래의 그림과 같이 IAT 복구하기 전에 오류가 발생하던 실행 파일이 온전히 실행됨을 확인할 수 있다.



(그림 42) IAT 복구 전 실행 파일 오류 화면



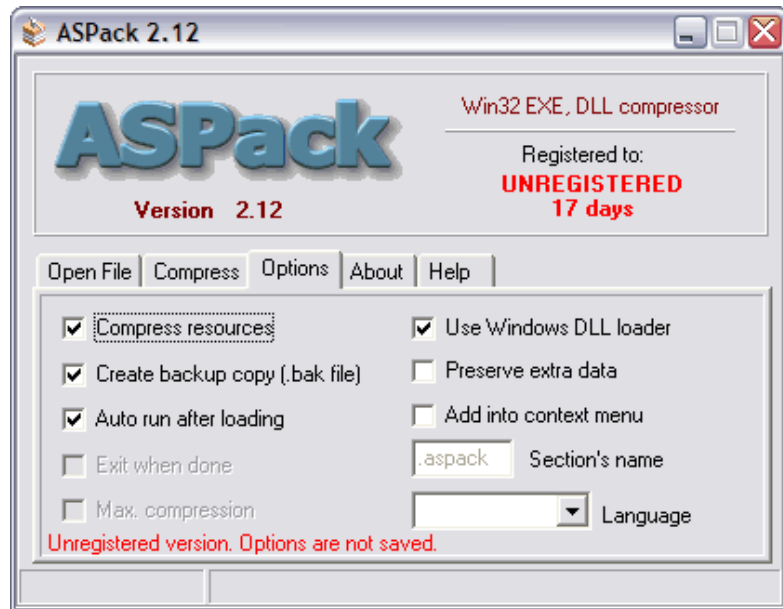
(그림 43) IAT 복구 후 정상 실행 화면

2. ASpack

ASpack은 앞의 통계 자료에서 볼 수 있듯이, 가장 많이 쓰이는 실행 압축 도구 중의 하나이다. ASpack은 상용 프로그램이고 32비트 윈도우 프로그램의 크기를 약 70%이상 줄일 수 있는 성능을 가진 실행압축 도구이다. ASpack은 프로그램과 라이브러리의 크기를 작게 해주고, 네트워크 간 로드 시간과 다운로드 시간을 줄여준다. 또한 프로그램을 리버스 엔지니어링으로부터 지켜준다. ASpack으로 압축된 프로그램은 실행 직전에 압축이

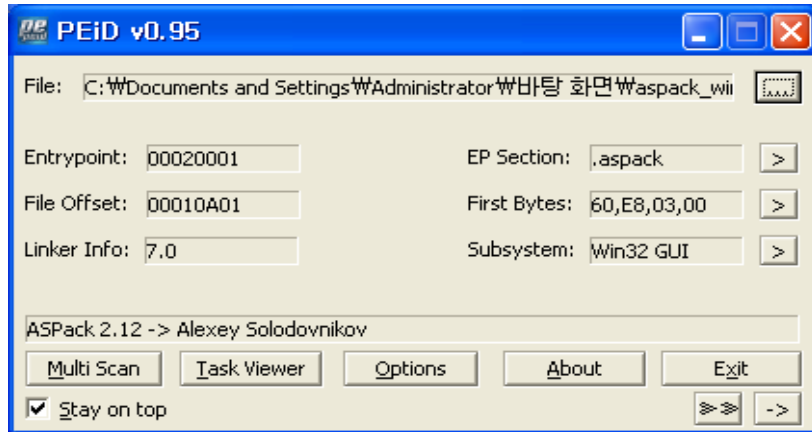
풀리며, 실행성능 패널티가 없다.

ASpack의 특징으로는 프로그램의 코드, 데이터, 리소스를 암호화와 압축을 하고, 다른 실행 압축 도구보다 빠른 복호화 루틴을 통해 높은 성능을 제공한다. 또한 상용 프로그램으로 꾸준한 개발이 이뤄지고 있기 때문에 호환성이 높다.



(그림 44) ASpack 2.12 version 실행 화면

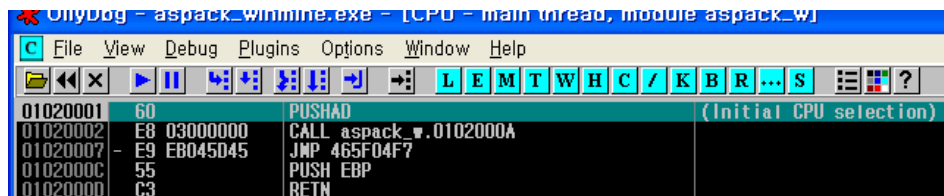
ASpack에 대한 매뉴얼 실행 압축 해제는 ASpack 2.12 버전으로 실행 압축된 지뢰찾기 프로그램을 사용하였다. 2장의 실행 압축 도구 탐지 방법 중 PEiD 도구를 사용하면 다음 그림과 같이 실행 압축되었음을 알 수 있다.



(그림 45) PEiD로 ASpack 2.12를 탐지한 화면

가. 오리지널 엔트리 포인트 찾기

ASpack으로 압축한 실행 압축 파일도 UPX와 같이 PUSHAD로 DWORD 형태로 레지스터 값을 저장한다. PUSHAD 명령어를 실행한 후 레지스터 값들이 스택에 저장되는데 ESP 스택 포인터는 그 스택의 TOP을 가리키게 된다.



(그림 46) ollydbg로 ASpack으로 실행 압축된 PE 파일의 시작 부분

일반적으로 실행 압축 프로그램은 특정 위치의 레지스터 값들을 백업해 놓기 위해 처음에 PUSH 명령을 사용한다. 스택에 레지스터 백업들을 저장하고 실행 압축 해제 작업을 위한 루틴이 시작된 후, 실행 압축 해제 루틴이 끝나고 본래 프로그램의 진입점이 시작되기 전에 POP 명령으로 처음의 스택에 저장되어 있던 레지스터들을 POP한다.

실행 압축 해제 과정을 마치고 본래의 프로그램이 실행되기 전에 POP을

하기 때문에 ESP의 위치에 접근할 때 브레이크 포인트를 걸어서 원래의 프로그램 시작부분(OEP)를 찾을 수 있다. ESP의 위치에서 Follow in Dump를 실행하여 Hex dump 창에 나타난 값에 4byte를 지정해 준 후 메뉴에서 Break Point Hardware, on access - DWord를 선택한 후 실행하면 지정된 메모리에 접근할 때 Break가 되고 RETN과 같은 레지스터를 복원하는 명령어를 볼 수 있다.



(그림 47) ESP에 브레이크 포인트 지정

TOP 부분에 브레이크 포인트를 지정했기 때문에 실행을 나머지 코드는 분석할 필요가 없다. 실행(F9:Run)을 하면 OEP 지점에 접근 할 수 있다.

브레이크 포인트를 지정하고 실행하면 POPAD 후에는 브레이크가 걸린다. 그리고 아래 부분에 PUSH 01003E21 / RETN이 보이는데 PUSH/RETN은 Call과 같은 명령이다. 따라서 Call 01003E21과 같은 의미이다. 이것은 언패킹이 끝나고 OEP로 점프하는 것이라고 예상할 수 있다. 한 단계 진행(F7:Step into)를 하면 OEP의 위치를 찾아낼 수 있다.

01020416	75 08	JNZ SHORT aspack_▯.01020420
01020418	B8 01000000	MOV EAX,1
0102041D	C2 0C00	RETN 0C
01020420	68 213E0001	PUSH aspack_▯.01003E21
01020425	C3	RETN
01020426	8B85 8C040000	MOV EAX,DWORD PTR SS:[EBP+48C]
0102042C	8D8D A1040000	LEA ECX,DWORD PTR SS:[EBP+4A1]
01020432	51	PUSH ECX
01020433	50	PUSH EAX
01020434	FF95 A50F0000	CALL DWORD PTR SS:[EBP+FA5]
0102043A	8985 B1050000	MOV DWORD PTR SS:[EBP+5B1],EAX
01020440	8D85 AD040000	LEA EAX,DWORD PTR SS:[EBP+4AD]
01020446	50	PUSH EAX
01020447	FF95 AD0F0000	CALL DWORD PTR SS:[EBP+FAD]
0102044D	8985 90040000	MOV DWORD PTR SS:[EBP+490],EAX
01020453	8D8D B8040000	LEA ECX,DWORD PTR SS:[EBP+4B8]
01020459	51	PUSH ECX

(그림 48) OEP에 근접한 화면

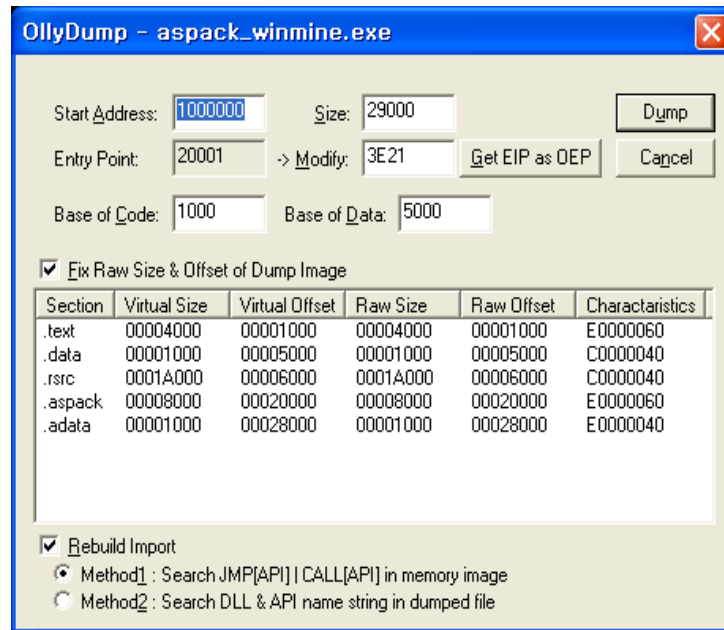
그림과 같이 OEP는 01003E21인 것을 알 수 있다.

01003E21	6A 70	PUSH 70	
01003E23	68 90130001	PUSH aspack_▯.01001390	
01003E28	E8 DF010000	CALL aspack_▯.0100400C	
01003E2D	330B	XOR EBX,EBX	
01003E2F	53	PUSH EBX	
01003E30	8B3D 8C100000	MOV EDI,DWORD PTR DS:[100108C]	
01003E36	FFD7	CALL EDI	
01003E38	66:8138 4D5A	CMP WORD PTR DS:[EAX],5A4D	
01003E3D	75 1F	JNZ SHORT aspack_▯.01003E5E	
01003E3F	8B48 3C	MOV ECX,DWORD PTR DS:[EAX+3C]	
01003E42	03C8	ADD ECX,EAX	
01003E44	8139 50450000	CMP DWORD PTR DS:[ECX],4550	
01003E4A	75 12	JNZ SHORT aspack_▯.01003E5E	
01003E4C	0FB741 18	MOVZX EAX,WORD PTR DS:[ECX+18]	
01003E50	3D 0B010000	CMP EAX,10B	
01003E55	74 1F	JE SHORT aspack_▯.01003E76	
01003E57	3D 0B020000	CMP EAX,20B	
01003E5C	74 05	JE SHORT aspack_▯.01003E63	

(그림 49) OEP 위치에 진입

나. 덤프 파일 생성

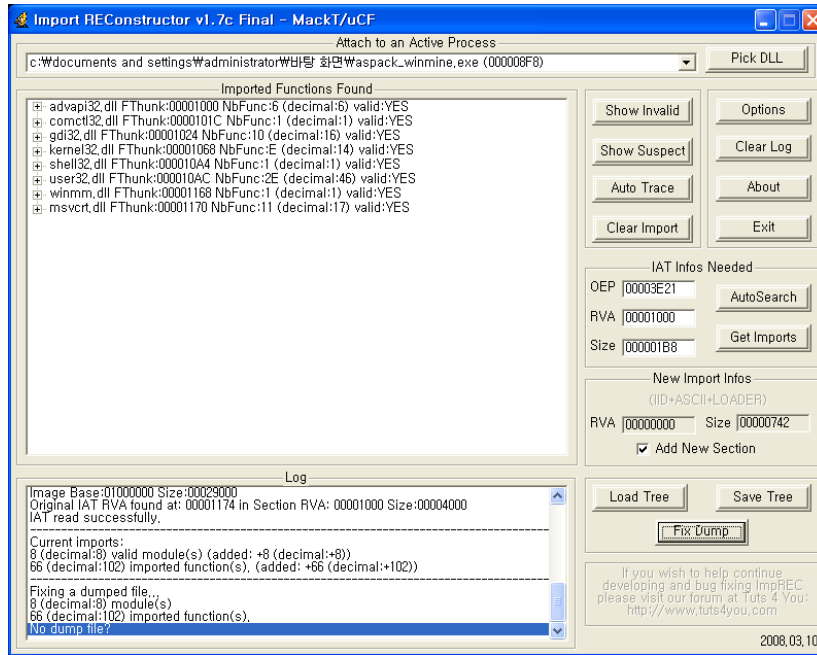
OEP를 찾았으면 다음으로 해야 할 일은 메모리에 매핑된 이미지를 디스크로 덤프하는 것이다. 이미지를 덤프하여 실행압축 된 파일로부터 본래의 파일을 얻을 수 있다. Ollydbg 디버거에는 메모리를 덤프하는 플러그인으로 Ollydump를 제공한다. OEP 부분에서 오른쪽 버튼을 클릭하여 Dump debugged process를 선택하면 Ollydump를 실행할 수 있다. Ollydump는 다음 그림에서 보는 것과 같이 Entry Point를 OEP로 변경한다. Rebuild Import는 자동으로 IAT를 복구해주는 기능이지만 대부분의 경우 정확한 복구를 수행하지 못하기 때문에 체크를 해제 한다.



(그림 50) Ollydump를 이용한 이미지 덤프

다. IAT 복구

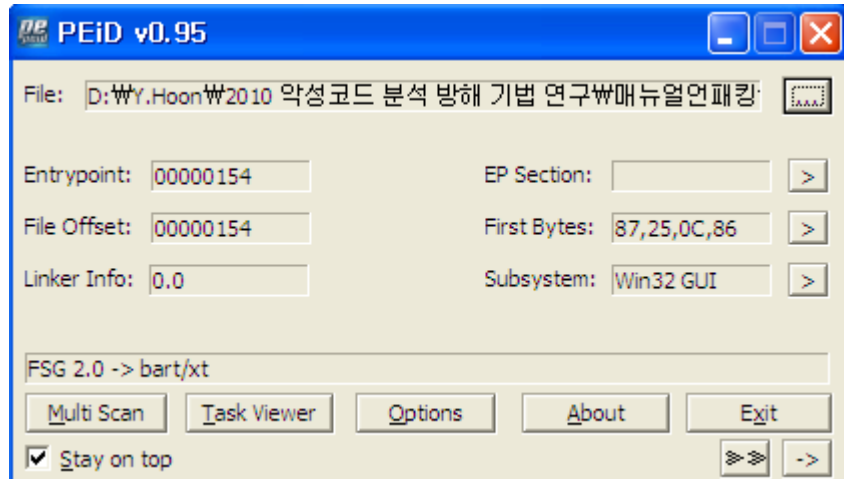
ImportREC를 실행하고 디버깅을 수행중인 실행압축 된 프로그램을 Attach한 후 찾아낸 OEP 값을 입력해주면 자동으로 RVA와 Import Function을 찾을 수 있다. 최종으로 IAT까지 복구하면 실행압축 된 파일을 확인할 수 있다.



(그림 51) ImportREC를 이용한 IAT 복구

3. FSG

FSG (Fast Small Good)를 사용하여 지뢰찾기 프로그램을 실행 압축 하였다. FSG로 실행 압축한 지뢰 찾기 프로그램에 대한 실행 압축 해제 방법을 알아본다.



(그림 52) PEID로 실행 압축 기법 확인

PEID 로 FSG 2.0 으로 실행 압축되었음을 확인 하였다. FSG의 특징은 다음 (그림 53) 과 같이 각 섹션 테이블에 이름이 보이지 않는 것이 특징이다.

Section Viewer					
Name	V. Offset	V. Size	R. Offset	R. Size	Flags
	00001000	00020000	00000000	00000000	C00000E0
	00021000	00018000	00000200	00017649	C00000E0

(그림 53) Section Viewer 화면

가. 오리지널 엔트리 포인트 찾기

01000154	8725 0C860301	xchg dword ptr ds:[103860C], esp
0100015A	61	popad
0100015E	94	xchg eax, esp
0100015C	55	push ebp
0100015C	A4	movs byte ptr es:[edi], byte ptr ds:[es
0100015E	B6 80	mov dh, 80
01000160	FF13	call near dword ptr ds:[ebx]

(그림 54) PE 파일 시작 부분

다음 그림은 PE 파일의 시작 부분이다. FSG에는 섹션 테이블에 이름이 없는 특징 이외에 실행 압축 해제를 할 때 큰 특징이 존재한다. 그 특징은 다음 시작 부분에서 Step over (F8)를 통해 코드를 진행하다보면 다음 그림과 같은 3개의 Jump 문 이전에 계속 루프를 도는 것을 확인 할 수 있다.

010001C4	8B07	mov eax, dword ptr ds:[edi]
010001C0	40	inc eax
010001C0	78 F3	js short 사본_-.w,010001C2
010001C9	75 03	jnz short 사본_-.w,010001D4
010001D1	FF63 0C	jmp near dword ptr ds:[ebx+C]
010001D4	50	push eax
010001D9	55	push ebp
010001D8	FF53 14	call near dword ptr ds:[ebx+14]

(그림 55) FSG의 특징 점

루프를 돌아 보면 해당 루프에서 실행 압축이 해제되고 있음을 알 수 있습니다. 따라서 3개의 Jump 문 이후의 이동 주소가 오리지널 엔트리 포인트임을 알 수 있습니다. 3개의 Jump 문 중 마지막 jmp 문에 브레이크 포인트를 걸고 실행 한 후 해당 Jump문 이후의 주소로 이동해보면 다음 그림과 같이 OEP 로 이동했음을 알 수 있다.

01003E21	8A 70	push 70	
01003E23	68 90130001	push 사본_-.w.01001390	
01003E26	E8 0F010000	call 사본_-.w.0100400C	
01003E2D	33DB	xor ebx, ebx	
01003E2F	53	push ebx	
01003E30	8B3D 8C10000	mov edi, dword ptr ds:[100108C]	
01003E36	FFD7	call near edi	
01003E38	66 8138 405A	cmp word ptr ds:[eax], 5A40	
01003E3D	75 1F	jnz short 사본_-.w.01003E5E	
01003E3F	8B48 3C	mov ecx, dword ptr ds:[eax+3C]	
01003E42	03C8	add ecx, eax	
01003E44	8139 5045000	cmp dword ptr ds:[ecx], 4550	
01003E4A	75 12	jnz short 사본_-.w.01003E5E	
01003E4D	0FB741 18	movzx eax, word ptr ds:[ecx+18]	
01003E50	3D 0B010000	cmp eax, 10B	
01003E55	74 1F	jb short 사본_-.w.01003E76	
01003E57	3D 0B020000	cmp eax, 20B	
01003E5D	74 05	jb short 사본_-.w.01003E63	
01003E5E	895D E4	mov dword ptr ss:[ebp-1C], ebx	
01003E61	EB 27	jmp short 사본_-.w.01003E8A	
01003E63	83B9 8400000	cmp dword ptr ds:[ecx+84], 0E	
01003E6A	76 F2	jbe short 사본_-.w.01003E5E	
01003E6D	33C0	xor eax, eax	
01003E6E	3999 F800000	cmp dword ptr ds:[ecx+F8], ebx	

pModule => NULL
kernel32.GetModuleHandleA
GetModuleHandleA

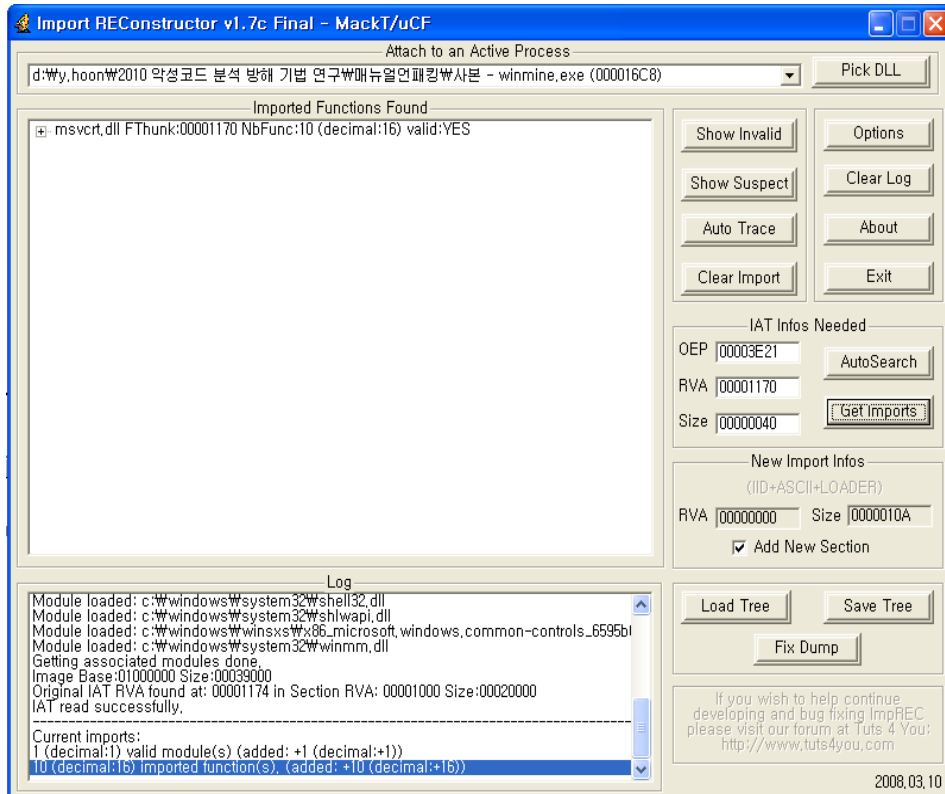
(그림 56) OEP 로 이동한 화면

나. 덤프 파일 생성

오리지널 엔트리 포인트를 찾았으니 해당 시점에서 덤프 파일을 생성 한다. 메모리에 매핑된 이미지를 디스크로 덤프하면 실행 압축된 파일로부터 실행 압축 해제된 본래의 파일을 찾을 수 있다. 이미지 덤프 작업을 해주는 도구는 여러 가지 있지만 여기서는 현재 분석 도구인 OllyDbg[8]에서 제공하는 Ollydump를 사용하여 덤프 파일을 생성한다.

다. IAT 복구

IAT 복구를 위하여 Import REconstructor 라는 IAT 복구 프로그램을 이용하여 복구한다.



(그림 57) 도구를 이용하여 IAT 복원하는 과정

이제 IAT를 복구하기 위해서 IAT infos Needed 에 필요한 정보를 입력해야 한다. 입력해야 할 정보는 다음과 같다.

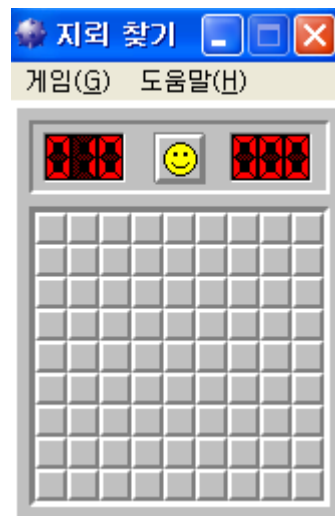
- OEP (RVA)
- Import table RVA
- Import table Size

여기서 오리지널 엔트리 포인트는 알았으니 Import table에 관한 정보를 알아내야 한다. 해당 정보를 알아내기 위해 Ollydbg에서 다음 그림과 같이 Import table에 관한 정보를 확인한다.

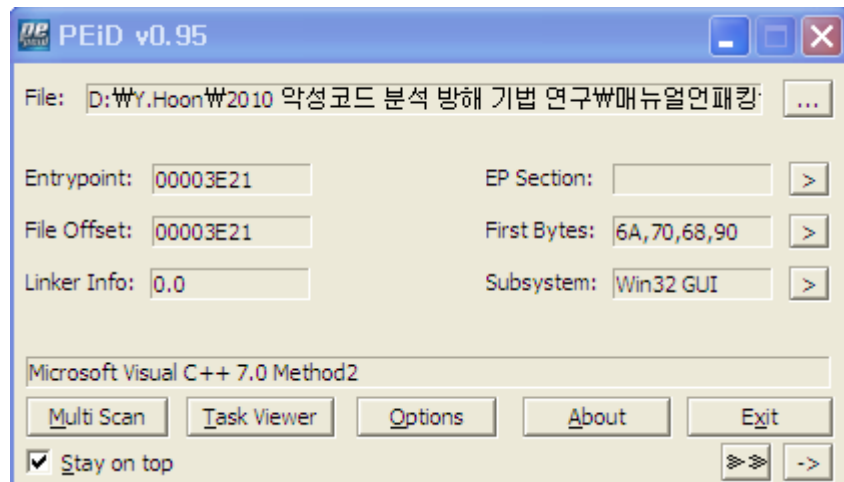
Address	Hex dump	Data	Comment
01000048	00020000	DD 00000200	FileAlignment = 200
0100004C	0500	DW 0005	MajorOSVersion = 5
0100004E	0100	DW 0001	MinorOSVersion = 1
01000050	0500	DW 0005	MajorImageVersion = 5
01000052	0100	DW 0001	MinorImageVersion = 1
01000054	0400	DW 0004	MajorSubsystemVersion = 4
01000056	0000	DW 0000	MinorSubsystemVersion = 0
01000058	00000000	DD 00000000	Reserved
0100005C	00900300	DD 00039000	SizeOfImage = 39000 (233472,)
01000060	00020000	DD 00000200	SizeOfHeaders = 200 (512,)
01000064	00000000	DD 00000000	Checksum = 0
01000068	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
0100006A	0000	DW 0000	DLLCharacteristics = 0
0100006C	00000400	DD 00040000	SizeOfStackReserve = 40000 (262144,)
01000070	00400000	DD 00004000	SizeOfStackCommit = 4000 (16384,)
01000074	00001000	DD 00100000	SizeOfHeapReserve = 100000 (1048576,)
01000078	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096,)
0100007C	00000000	DD 00000000	LoaderFlags = 0
01000080	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16,)
01000084	00000000	DD 00000000	Export Table address = 0
01000088	00000000	DD 00000000	Export Table size = 0
0100008C	C8850300	DD 000385C8	Import Table address = 385C8
01000090	84000000	DD 00000084	Import Table size = 84 (132,)
01000094	00100200	DD 00021000	Resource Table address = 21000
01000098	24680000	DD 00006824	Resource Table size = 6824 (26660,)
0100009C	00000000	DD 00000000	Exception Table address = 0
010000A0	00000000	DD 00000000	Exception Table size = 0
010000A4	00000000	DD 00000000	Certificate File pointer = 0
010000A8	00000000	DD 00000000	Certificate Table size = 0
010000AC	00000000	DD 00000000	Relocation Table address = 0
010000B0	00000000	DD 00000000	Relocation Table size = 0
010000B4	00000000	DD 00000000	Debug Data address = 0
010000B8	00000000	DD 00000000	Debug Data size = 0

(그림 58) Import table 정보 확인 화면

얻은 정보를 바탕으로 Import REconstructor 도구를 이용하여 IAT를 복구한다. 복구가 완료 되었으면 확인한다.



(그림 59) 실행 압축
해제 후 실행 화면



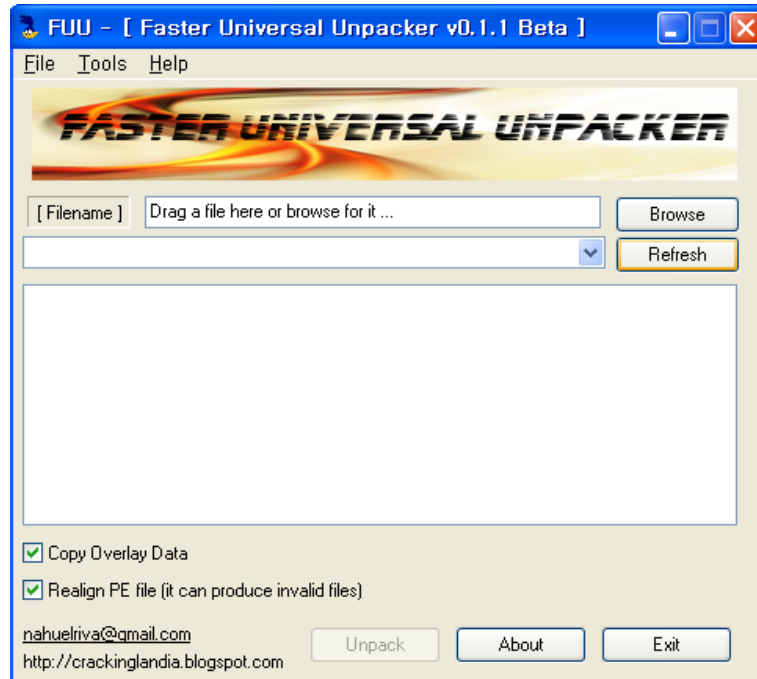
(그림 60) PEID 로 실행 압축 해제 확인 화면

위의 그림과 같이 실행 압축 해제가 잘 되었음을 확인한다.

제 3 절 실행 압축 해제 S/W 또는 Module

1. FUU (Faster Universal Unpacker)

FUU[9]는 잘 알려진 UPX, ASPack, FSG, ACProject 등의 실행 압축 도구로 실행 압축된 파일을 플러그인을 사용하여 자동적으로 실행 압축 해제 해주는 도구이다.

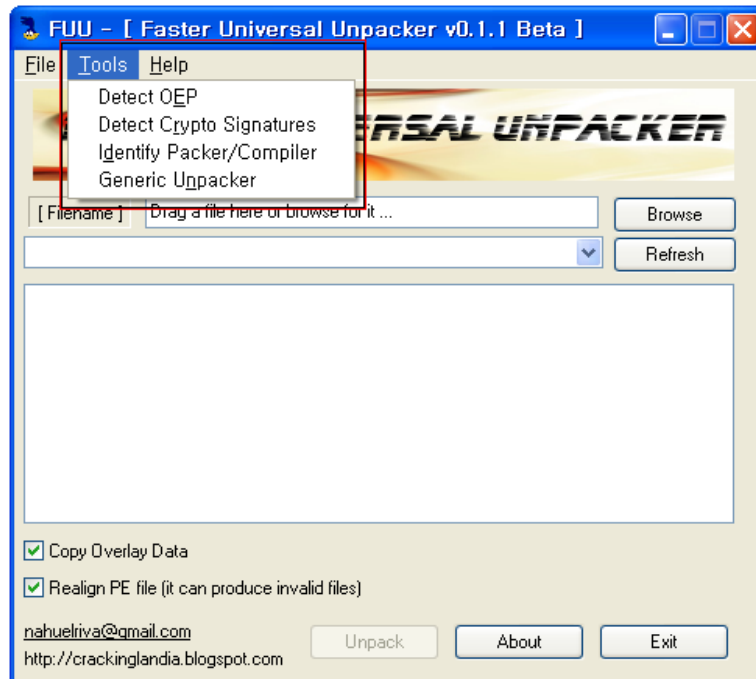


(그림 61) FUU 실행 화면

FUU 플러그인들의 핵심은 ReversingLabs 의 TitanEngine SDK를 이용하고 있다. 이것은 플러그인 개발자들에게 반복적인 작업과 덤프와 같은 지루한 기능을 걱정하지 않고도 쉽고 빠르게 IAT를 고치고 섹션을 추가하는 플러그인을 만들 수 있게 한다. 사용자는 FUU의 플러그인을 TitanEngine[10]을 이용하여 매우 쉬운 방법으로 제작 할 수 있게 된다.

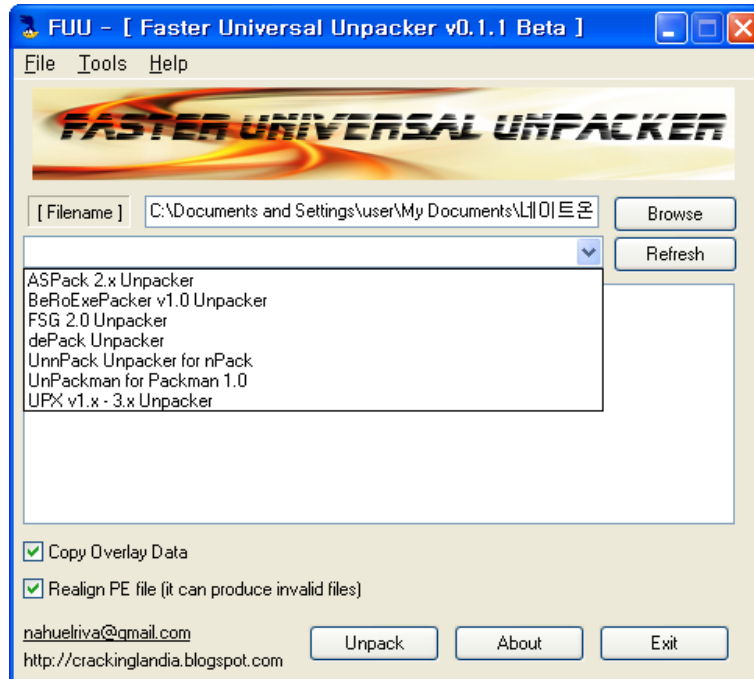
또한 FUU 는 다른 추가적인 툴들을 프로그램 내에서 제공한다.

- Generic OEP Finder
- Crypto Signature Detector
- Generic Unpacker
- Signatures Detector (by marcianito at gmail dot com)



(그림 62) 추가적으로 제공되는 도구

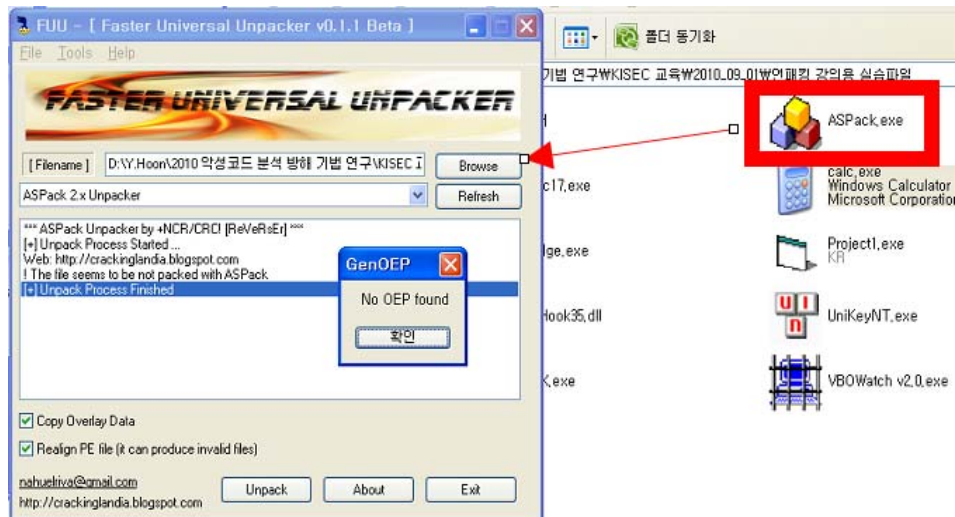
Generic OEP Finder, Cryto Signature Detector 와 Generic Unpacker 는 PEiD 팀으로부터 제공되는 것이다.



(그림 63) 플러그인 설정 화면

FUU 에서는 현재 7가지의 플러그인을 통하여 실행 압축 해제를 하고 있다. 앞에서 언급한 내용처럼 FUU는 각 실행 압축 도구에 맞게 TitanEngine을 이용하여 플러그인을 제작하고 제작된 플러그인을 통하여 쉽게 실행 압축 해제 할 수 있다는 장점이 있다.

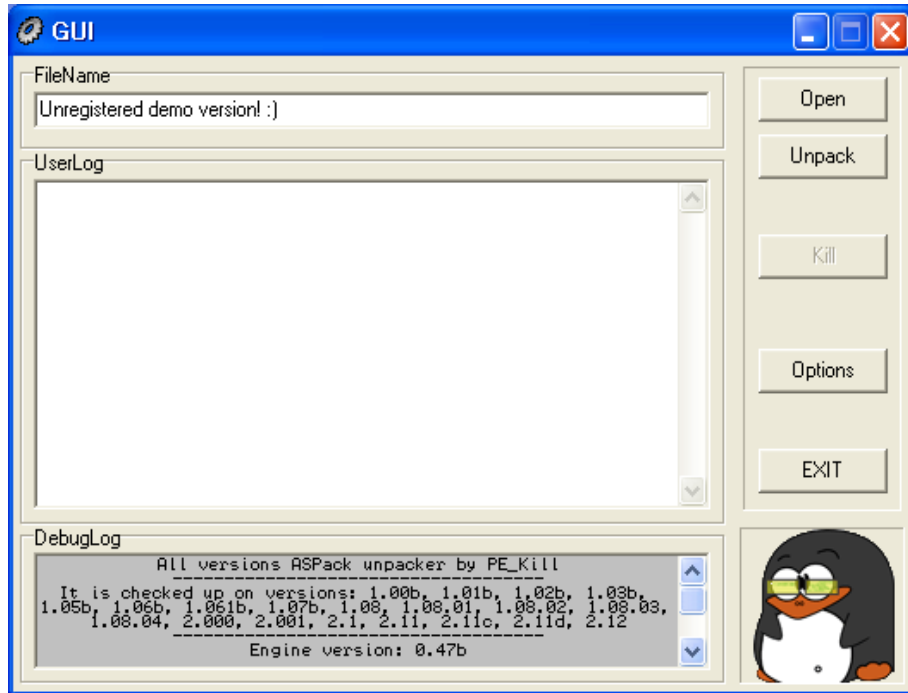
하지만 실제 FUU를 가지고 실험을 해 본 결과 UPX는 비교적 실행 압축 해제가 잘 되나 ASPack등 다른 실행 압축 도구들로 실행 압축한 파일에 대해서는 실행 압축이 안되었다고 나오거나 오리지널 엔트리 포인트를 찾지 못하여 실행 압축 해제에 실패하는 경우가 많이 있다.



(그림 64) FUU 실험 결과 화면

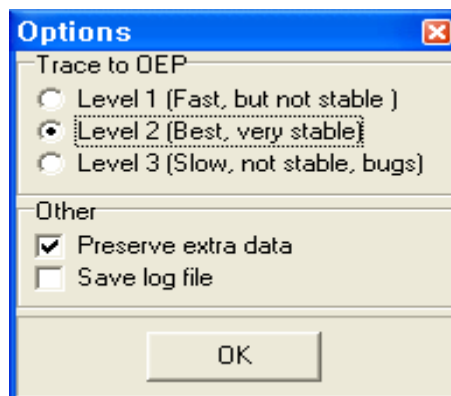
2. AbstersiverA

AbstersiverA는 ASPack에 대한 실행 압축 해제 프로그램으로 현재 ASPack 최신 버전인 2.24에 대한 실행 압축 해제는 하지 못하지만, 1.00~2.12 버전까지는 자동 실행 압축 해제가 가능한 프로그램이다.



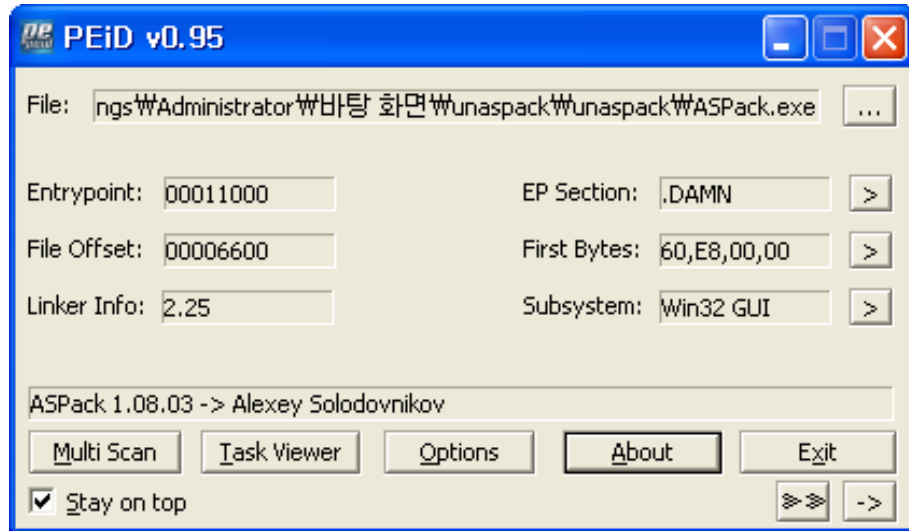
(그림 65) AbstersiverA의 기본 화면

AbstersiverA는 단순히 ASPack에 대한 실행 압축 해제 기능만 제공한다. Open으로 ASPack으로 실행 압축된 프로그램을 선택하고, Unpack 버튼을 클릭하면 실행 압축 해제 과정에 대한 로그가 출력이 되고, 실행 압축 해제 과정이 완료된다.



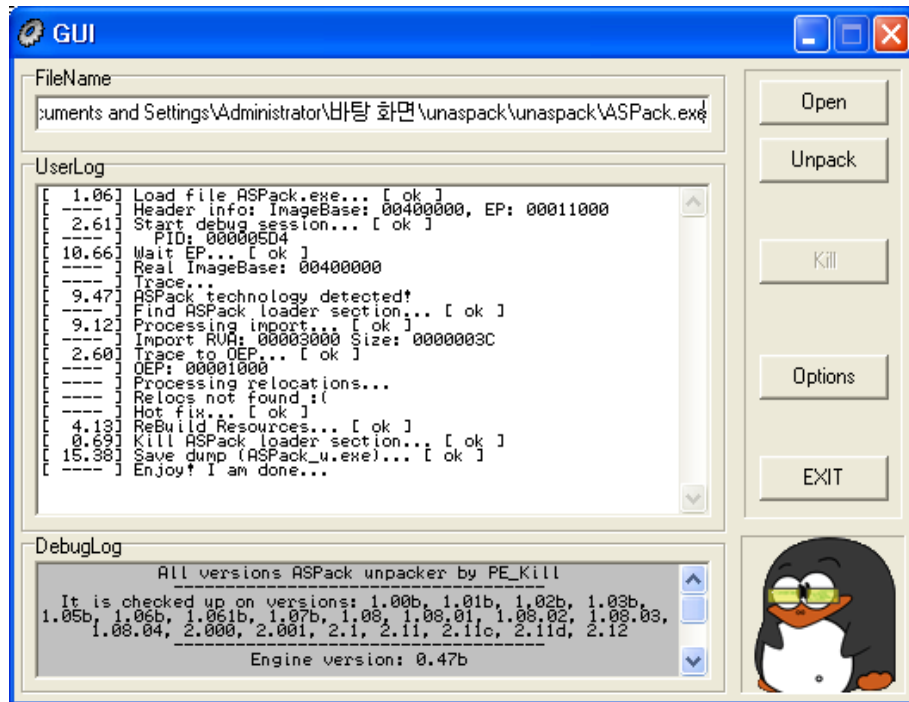
(그림 66) AbstersiverA의 Option

기본적으로 Level 2의 옵션으로 선택되어 있지만, 실행 압축 해제가 잘 수행되지 않는 경우 Option에서 설정을 바꿔가면서 적용하면 된다.



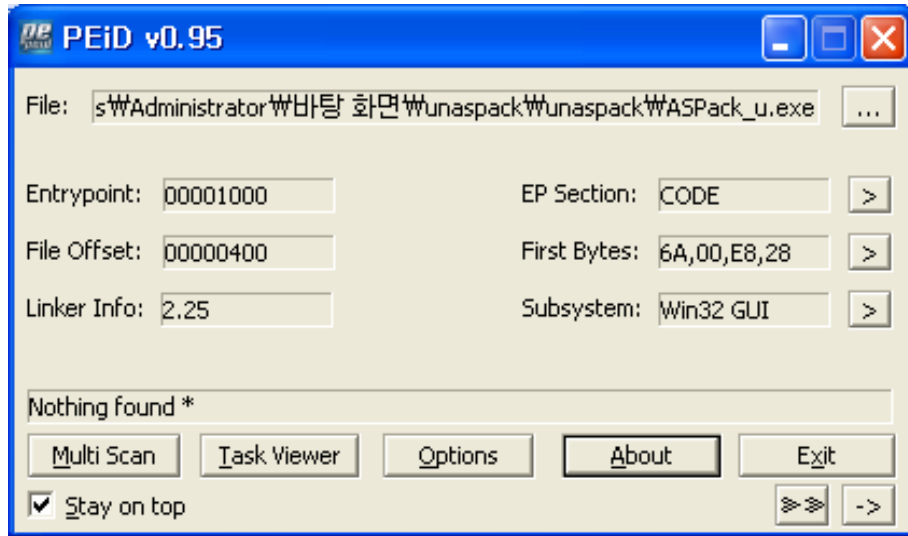
(그림 67) ASPack 1.08 버전으로 압축된 파일

테스트를 위해 PEiD로 ASPack 1.08.03 버전으로 압축된 파일을 확인하였다.



(그림 68) ASPack을 실행 압축 해제 하는 과정

AbstersiverA로 실행 압축 해제 하는 Log를 통해 실행 압축 해제 과정을 살펴보면, Header의 정보와 ImageBase를 확인하고, ASPack의 정보를 확인 후 그 버전에 맞는 실행 압축 해제 프로세스를 적용하여 오리지널 엔트리 포인트를 찾고 재정의 하는 과정으로 이뤄지는 것을 알 수 있다. 이 과정은 사람이 수동으로 실행 압축 해제 하는 과정과 유사하다.



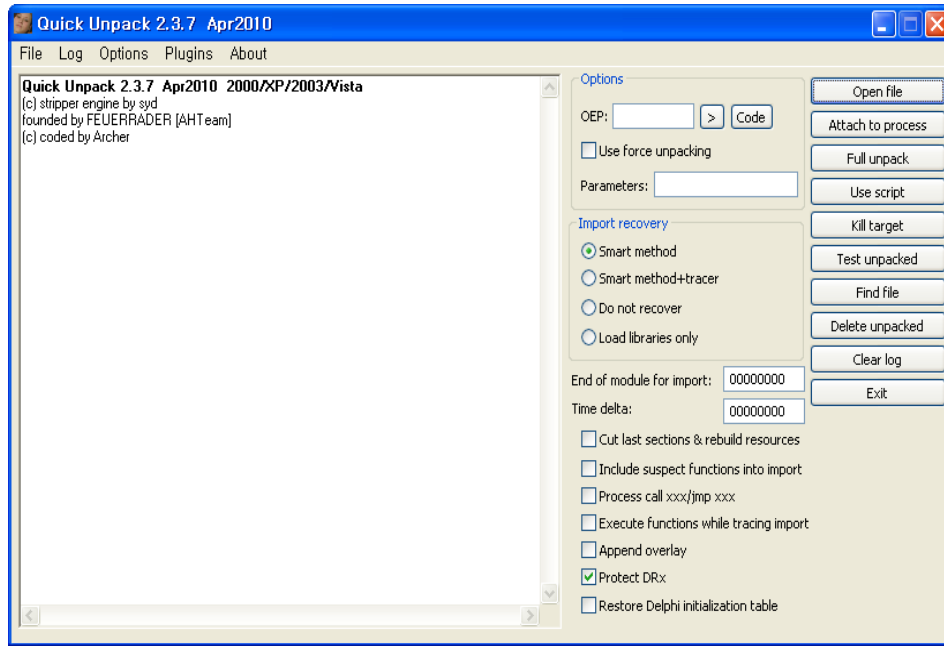
(그림 69) Unpack 프로그램 실행 후 PEiD로 확인한 결과

실행 압축 해제 프로그램 실행 후 PEiD로 확인한 결과 실행 압축 해제된 것을 확인 할 수 있었다.

ASPack이 아닌 다른 프로그램으로 실행해본 결과 입력된 ASPack이 아닌 프로그램에 대해서는 추적을 하면서 프로그램이 실행이 되는 것을 확인했다. 이 결과로 프로그램의 코드를 하나씩 실행하면서 입력된 코드에 대해서만 탐색이 가능한 것을 알 수 있었다.

3. Quick Unpack 2.3.7

Quick Unpack은 ALIEN Hack Team(ahteam)에서 만들어서 제공하는 툴로 현재까지 꾸준히 업데이트 되고 있는 툴 중의 하나이다. 아래 그림에서 보이는 것과 같이 2010년 4월에도 업데이트가 이루어 졌으며, 현재 최신 버전은 2.3.7이다. ALIEN Hack Team 홈페이지[11]에서 툴을 제공하고 있으며, 국내 유저들에 의해 한글화된 버전도 배포되고 있다.



(그림 70) Quick Unpack의 기본 화면

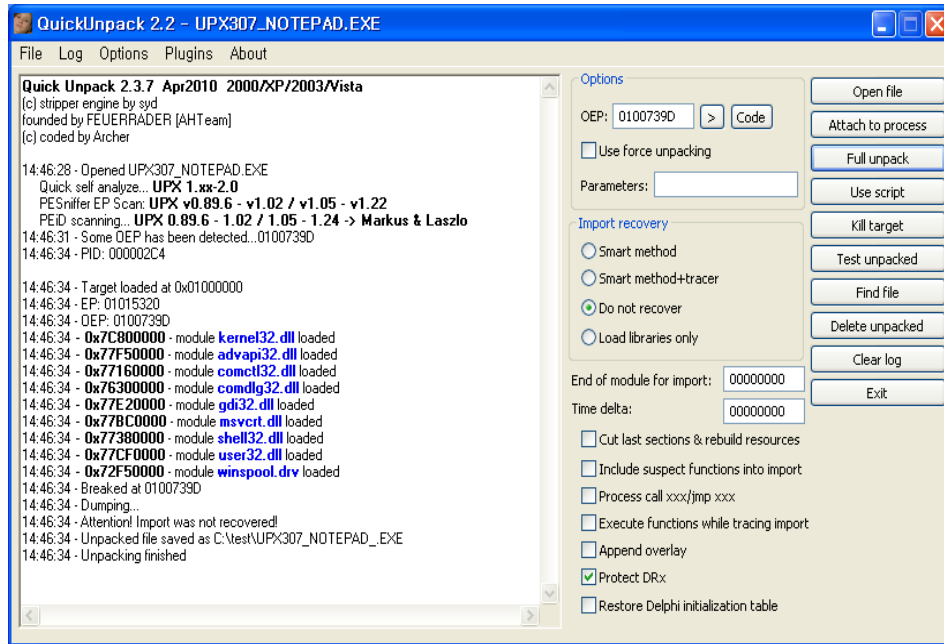
다음은 Quick Unpacker 2.3.7에서 실행 압축 해제를 제공하는 실행 압축 도구 목록이다.

- 32Lite
- AnslymPacker
- AREA51 Cryptor
- Armadillo (minimal protection)
- AsdPack
- ASPack
- ASProtect (old versions)
- BeroEXEPacker
- CD-Cops
- DDeM
- depack
- DragonArmor
- Exe32Pack

- ExeCryptor (old versions)
- ExeFog
- ExeSax
- ExeShield
- ExeStealth
- fEaRz Crypter
- FreeCryptor
- FriCryptor
- FSG
- HidePE
- HidePX
- hmimys-Packer
- JDPack
- KByS
- kkrunchy
- LameCrypt
- Manolo
- MEW
- Minke
- NeoLite
- NME
- NsPack
- Orien
- PackMan
- PECompact
- PEDiminisher
- PE-PACK
- PEncrypt
- Perplex PE-Protector
- PeTite
- PEX
- PI Cryptor

- PKLite32
- PollyBox
- PolyEnE
- Protection Plus
- QrYPt0r NuTraL Poly
- QuickPack
- RLPack
- Sopelka
- StealthPE
- TeLock (not all versions)
- TheMida (minimal protection)
- unnamed Scrambler
- UPack
- UPolyX
- UProtector
- UPX
- WindOfCrypt
- WinUPack
- WWPack32
- Yoda Crypter
- Yoda Protector
- YZPacker
- ...many others...

Open file을 눌러서 실행 압축 해제 할 프로그램을 선택하면 3가지의 PE Scan으로 파일의 정보를 알아낸다. Option에서 OEP Finder를 선택하면 오리지널 엔트리 포인트가 찾아진다. Import Recovery에서 Smart mode, Smart method+tracer, Do not recover, Load libraries only 옵션을 지정하면 실행 압축 해제 수행 후 Import Table 복원까지 설정이 된다. 또한 플러그인을 추가해서 플러그인의 추가적인 기능들을 이용할 수 있고, 그 이외에 프로그램에서 제공하는 추가적인 옵션들을 지정할 수 있다.

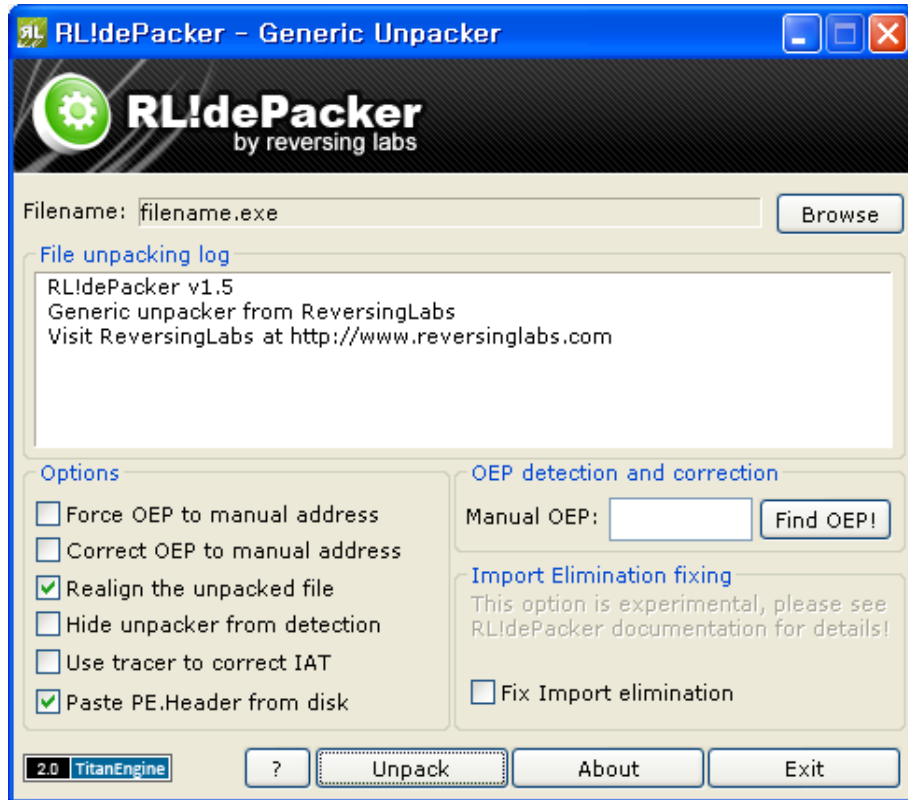


(그림 71) UPX로 실행 압축된 파일을 실행 압축 해제 하는 과정

프로그램 실행 결과 프로그램에서 제공하는 OEP Finders들이 제대로 된 오리지널 엔트리 포인트를 찾지 못하는 경우가 발생하였고, Import 복원을 Smart method나 Smart method+tracer로 설정하면 대부분 오류가 발생하였다. 또한 실행 압축 해제의 실행이 완료된 후 실행 압축 해제 된 파일이 실행되지 않았다.

4. RL!dePacker

RL!dePacker는 Reversing labs[12]에서 개발하여 공개한 소프트웨어이다. 최신 버전은 1.5이고 2009년에 발표되었다. 프로그램은 Reversing labs 홈페이지에서 공개 되어있다. 이 툴은 Reversing labs에서 제작한 Titan Engine 2.0을 기반으로 작동한다. Titan Engine은 매뉴얼 실행 압축 해제하는 방식을 모방하여 만든 프로그램이다. Titan Engine 또한 Reversing labs 홈페이지에서 제공하고 있다.



(그림 72) RL!dePacker 기본 화면

RL!dePacker를 실행하면 그림과 같은 화면이 나타난다. 실행 압축 해제 하려는 파일을 Browse 버튼을 눌러 추가하거나 드래그해서 추가하고 Options에서 원하는 옵션을 선택한 후 Unpack 버튼을 누르면 실행 압축 해제가 진행된다.

RL!dePacker에서 실행 압축 해제가 가능한 것으로 테스트된 실행 압축 도구의 종류는 101개 이상이고 그 목록은 아래와 같다. 각 실행 압축 도구로 실행 압축 할 때마다 주어지는 옵션에 따라 실행 압축 해제 가능 여부가 조금씩 다르며, 상세한 내용은 RL!dePacker에서 제공하는 사용 설명서에 실행 압축 도구의 옵션에 따른 실행 압축 해제 가능 여부가 설명되어 있다.

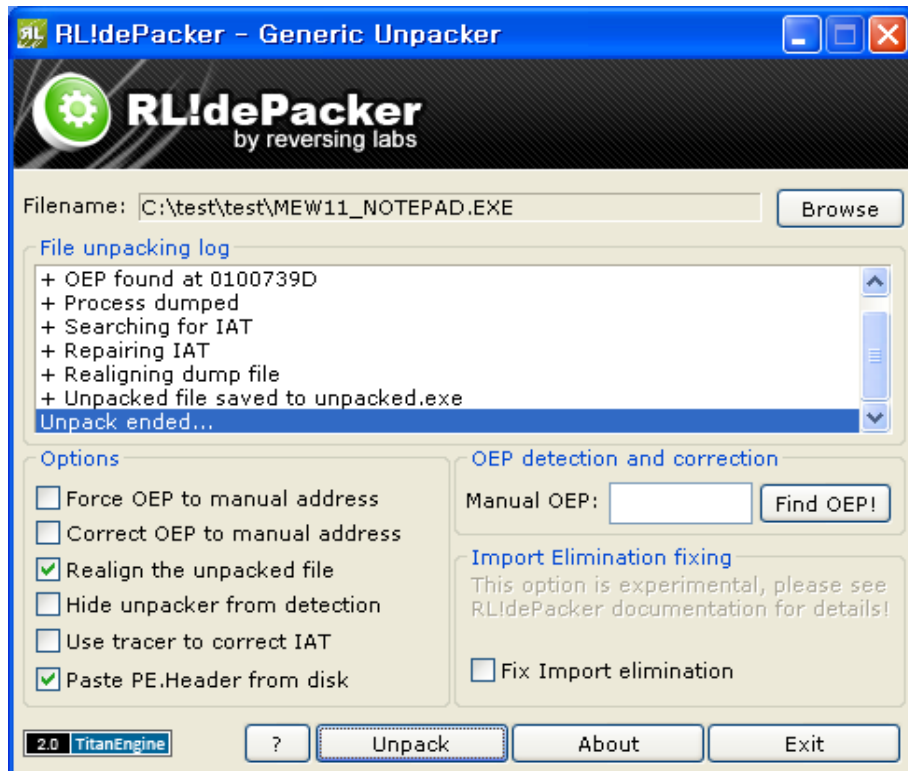
RL!dePacker 실행 압축 해제 테스트된 실행 압축 도구 목록 101개:

- aUS [Advanced UPX Scrambler] 0.4 - 0.5
- ASPack 1.x - 2.x
- ASPack Scrambler 0.1
- ASDPack 2.0
- AHPack 1.x
- AlexProtector 1.x
- ARMProtector 0.x
- BJFNT 1.3
- BeRoEXEPacker 1.x
- BamBam 0.x
- CryptoPeProtector 0.9x
- Crunch x.x
- CodeCrypt 0.16x
- dot Fake Signer 3.x
- dePack
- eXPressor 1.2.x - 1.5.x
- EZip 1.0
- EP Protector 0.3
- Escargot 0.x
- EXEStealth 2.x
- ExeSax 0.9x
- FSG 1.xx & 2.0
- Fusion x.x
- Goat's PE Mutilator 1.6
- hmimys-Packer 1.x
- Hmimys PePack 1.0
- HidePX 1.4
- HidePE 2.1
- ID Application Protector 1.2
- JDPack 1.x

- JDProtect 0.9
- Just Another Pe Packer 0.5
- KByS Packer 0.2x
- Krypton 0.x
- LameCrypt 1.0
- MEW 1.x
- mkfpack
- NakedPack 1.0
- nSPack 2.x - 3.x
- nPack 1.x
- NeoLite 1.x - 2.0
- NWCC
- OrIEN 2.1x
- PECompact 1.x - 2.x
- PeX 0.99
- PC Shrink 0.71
- Polyene 0.01
- PackMan 0.0.0.1 & 1.0
- PE Diminisher 0.1
- PolyCrypt PE 2.1.5
- PeTite 1.x
- PESTubOEP 1.6
- PELockNT 2.x
- PePack 1.0
- PC PE Encryptor alpha
- PackItBitch
- PEncrypt 4.0
- PEnguinCrypt 1.0
- PeLockNt 2.x
- PeLock 1.0x
- PESHIELD 0.25
- Perplex PE-Protector 1.x

- PeTite 1.0 - 1.3
- PKLITE32 1.x
- RLP 0.6.9 - 0.7.x
- RLPack Basic Edition 1.x
- RLPack Modifier Edition 1.x
- ReCrypt 0.15 - 0.80
- Stone's PE Encryptor 2.0
- StealthPE 2.1
- Software Compress 1.x
- SPLayer 0.08
- ShrinkWarp 1.4
- SPEC b3
- SmokesCrypt 1.2
- Simple UPX-Scrambler
- SimplePack 1.x
- SLVc0deProtector 1.x
- tELock 0.x
- UPX 0.8x - 2.x
- UPXRedir
- UPXCrypt
- UPX Inkvizitor
- UPXFreak 0.1
- UPolyX 0.x
- UPXLock 1.x
- UG Chruncher 0.x
- UPX-Scrambler RC 1.x
- UPX Protector 1.0x
- UPXShit 0.06 & 0.0.1
- UPXScramb 2.x
- VirogenCrypt 0.75
- VPacker 0.02.10
- WWPack32 1.x

- WinUPack 0.2x - 0.3x
- Winkript 1.0
- yC 1.x
- yZPack 1.x - 2.x
- 32Lite 0.3a
- !EP (ExE Pack) 1.x
- [G!X]'s Protector 1.2



(그림 73) RL!dePacker 실행 압축 해제 실행 화면

RL!dePacker에서 실행 압축 해제를 실행하면 위 그림과 같이 나타난다. 위의 파일은 MEW 1.1 버전의 실행 압축 도구로 메모장을 실행 압축한 후 RL!dePacker로 실행 압축 해제한 과정이다. 프로그램에서 나타나는 로그 목록을 보면, 오리지널 엔트리 포인트를 찾고 프로세스를 덤프 뜯 후, IAT 검색, IAT 복구, 덤프 파일을 재조정하고 unpacked.exe로 저장하는 과정

을 볼 수 있다. 이러한 과정은 사람이 매뉴얼 실행 압축 해제 하는 과정과 유사하며 RL!dePacker는 사람이 실행 압축 해제 하는 과정을 자동화 시켜 놓은 것으로 생각할 수 있다.

목록에 나와 있는 실행 압축 도구 중 10여개의 실행 압축 도구로 실행 압축한 후 테스트 해본 결과, 대부분의 실행 압축 도구에 대해서 실행 압축 해제가 제대로 실행되었고, 실행 압축 해제 후 프로그램의 실행이 가능하였다. 하지만 PEDiminisher 0.1 버전으로 실행 압축 한 파일은 실행 압축 해제가 성공되었다는 메시지가 출력되었지만 프로그램이 실행이 되지 않았다. 오리지널 엔트리 포인트를 찾는 과정에서 문제가 있었던 것으로 보인다.

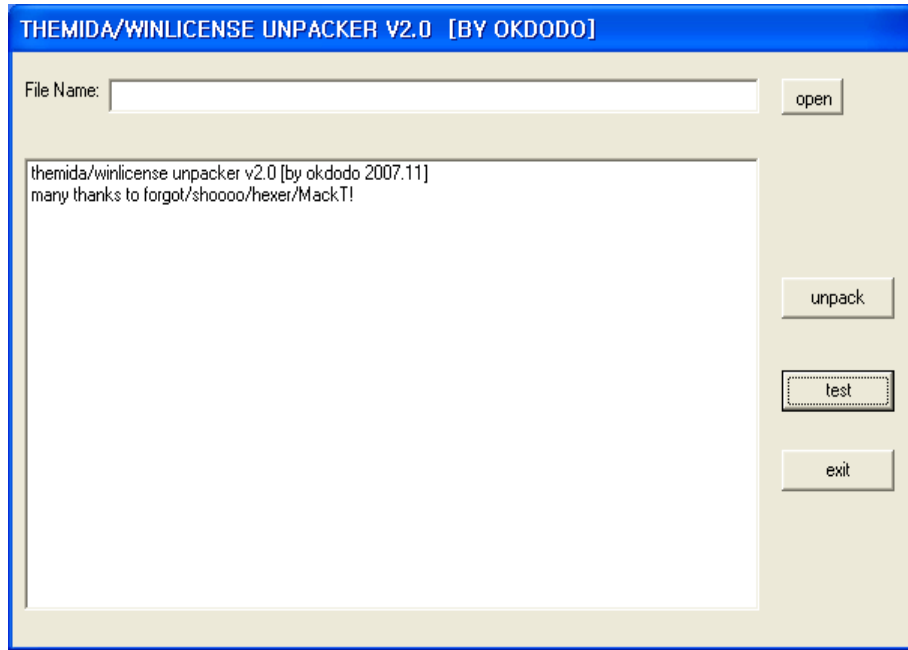
5. THEMIDA/WINLICENSE UNPACKER V2.0

이 툴은 이름에서 알 수 있듯이 Themida로 실행 압축된 파일을 실행 압축 해제하기 위해 만들어진 툴이다. Themida는 Oreans Technology[13]에서 제작한 실행 압축 도구로 현재 상용화된 실행 압축 도구 중에 가장 실행 압축 해제하기 어렵다고 알려져 있다. Themida로 실행 압축 할 때 아래 그림과 같이 분석을 힘들게 하기 위해 선택하는 옵션의 종류도 많으며, 안티 디버깅 옵션들도 많기 때문에 분석하는데 많은 시간을 투자하게 된다. THEMIDA/WINLICENSE UNPACKER V2.0는 Themida를 자동적으로 실행 압축 해제 할 수 있게 하여 분석가를 돕는 툴이다.



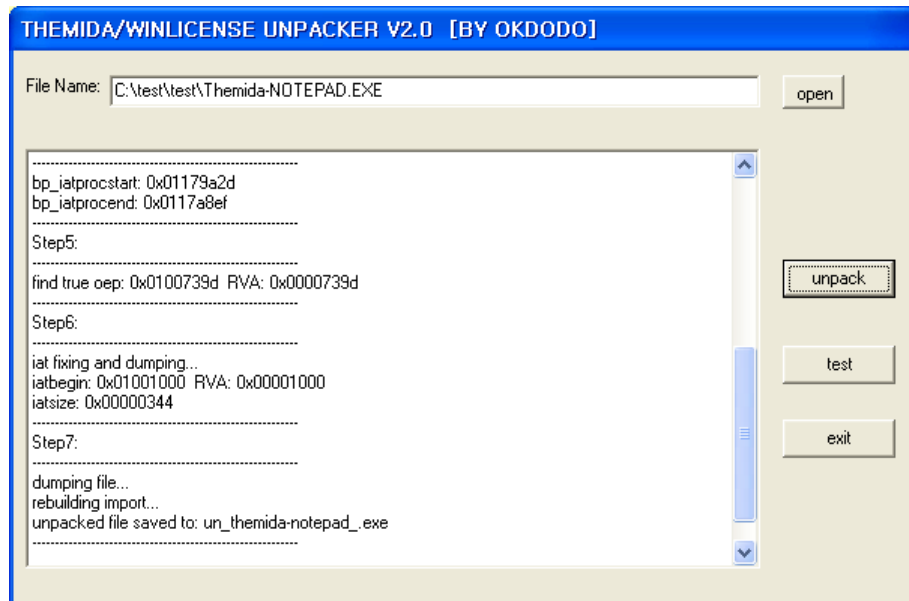
(그림 74) Themida Packer와 Protection Option

Themida는 상용화된 실행 압축 도구이기 때문에 Oreans Technology에서 제공하는 2.0.4.0 Demo 버전으로 파일을 실행 압축 하고 테스트를 하였다.



(그림 75) THEMIDA/WINLICENSE UNPACKER V2.0 기본 화면

THEMIDA/WINLICENSE UNPACKER V2.0은 단순한 구조로 되어 있다. 사용자가 선택할 수 있는 옵션은 아무 것도 없으며, 파일을 Open하고 Unpack, Test 만 할 수 있는 구조이다.



(그림 76) THEMIDA/WINLICENSE UNPACKER V2.0 실행 화면

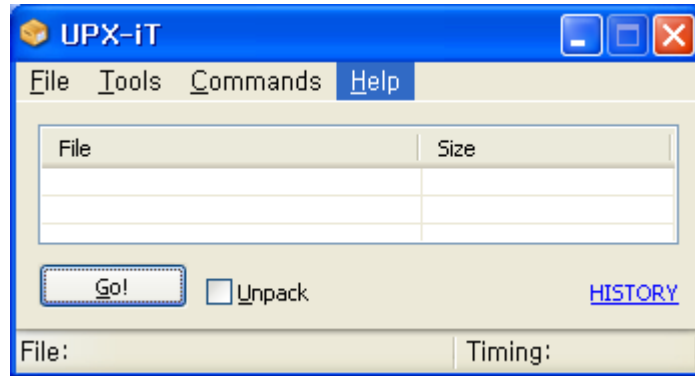
테스트 파일은 Themida의 모든 보호 옵션을 최고로 설정한 것과, 설정하지 않은 파일 2개를 가지고 하였다. 위 그림은 Themida의 보호 옵션을 설정하지 않은 파일을 실행 압축 해제 한 결과화면으로, 보호 옵션을 설정하지 않은 파일에 대해서는 프로그램으로 실행 압축 해제가 성공하였다. 하지만 모든 보호 옵션을 최고로 설정한 파일은 실행 압축 해제 과정 중 오류가 발생하였고, 프로그램이 종료되었다. 이러한 오류는 Themida의 안티 디버깅 옵션 때문에 발생하는 것으로 보인다.

THEMIDA/WINLICENSE UNPACKER V2.0에서 나타나는 로그를 보면 다른 실행 압축 해제 프로그램에서 나타나는 로그와 마찬가지로 오리지널 엔트리 포인트를 찾고 덤프를 수행하고 IAT를 재정의 하는 순서로 나타나는 것을 확인 할 수 있다.

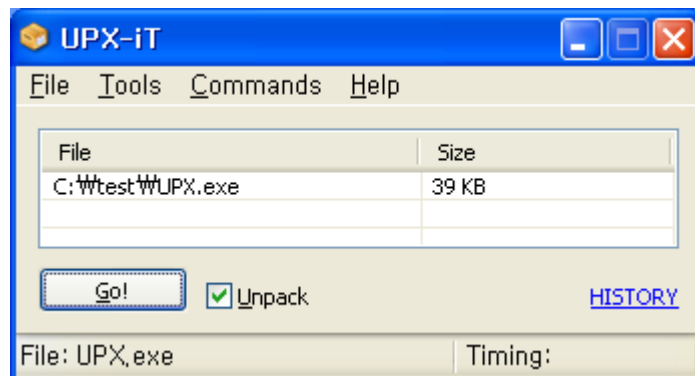
6. UPX-iT

UPX로 실행 압축된 파일을 실행 압축 해제 할 수 있는 프로그램이다. 2005년도에 1.6.2버전이 나온 후 업데이트가 이뤄지지 않고 있다. UPX 프

로그래밍이 커맨드 상에서 실행 압축, 실행 압축 해제 하는 것을 GUI 환경에서 좀 더 편리하게 이용할 수 있도록 나온 프로그램이다. UPX의 경우 프로그램에서 제공하는 -d 옵션이 실행 압축 해제를 할 수 있도록 제공한다.

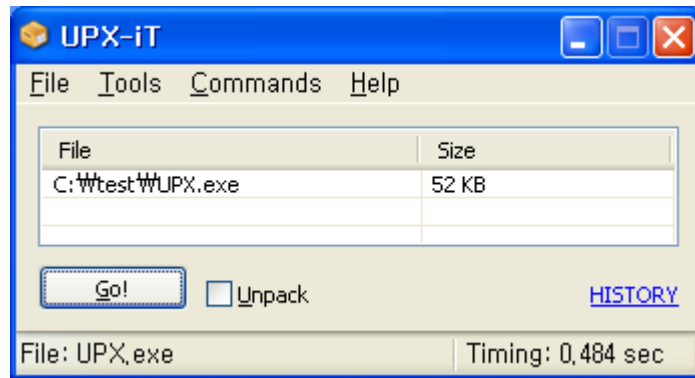


(그림 77) UPX-iT 기본 화면



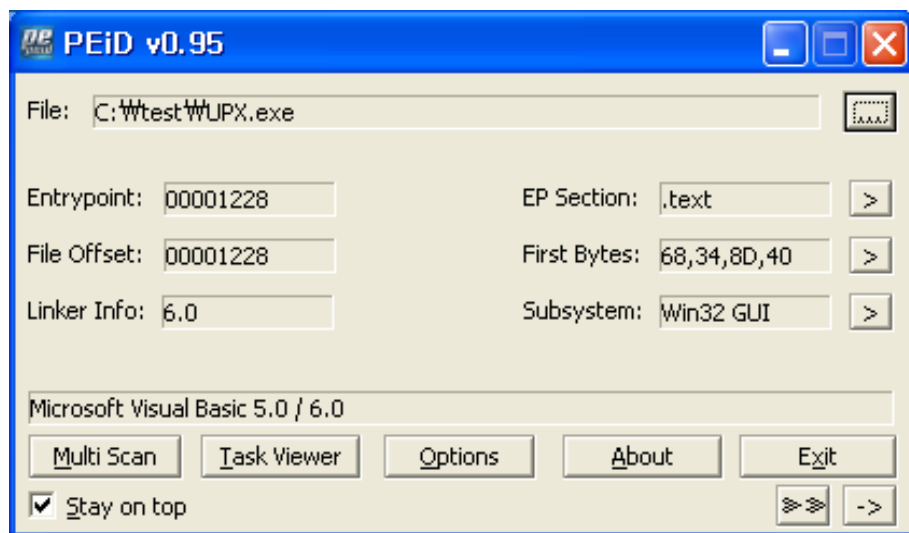
(그림 78) UPX로 실행 압축된 파일을 등록

UPX 2.9 버전으로 압축된 파일을 등록한 결과 파일의 경로와 크기에 대한 정보가 나왔다.



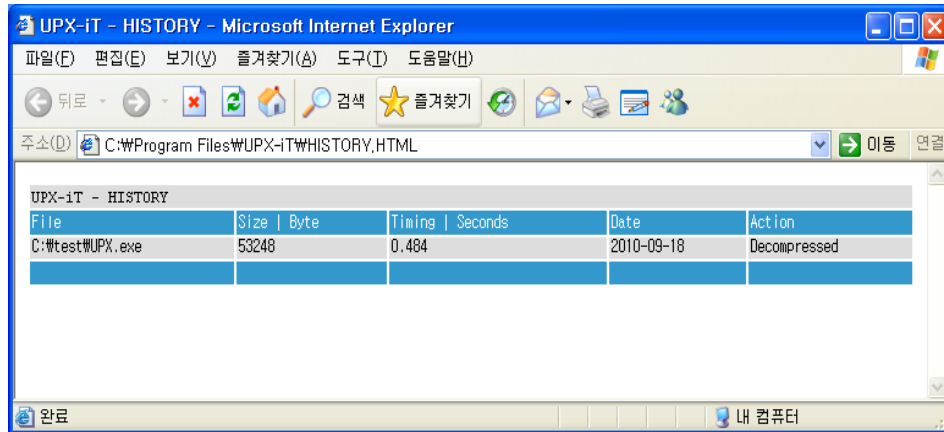
(그림 79) 실행 압축 해제 실행화면

위의 그림과 같이 약 0.484초 만에 실행 압축 해제가 실행된 것을 확인할 수 있다. 또한 파일의 용량이 39KB이었던 파일이 실행 압축 해제 되면서 52KB로 늘어나는 것을 확인할 수 있었다.



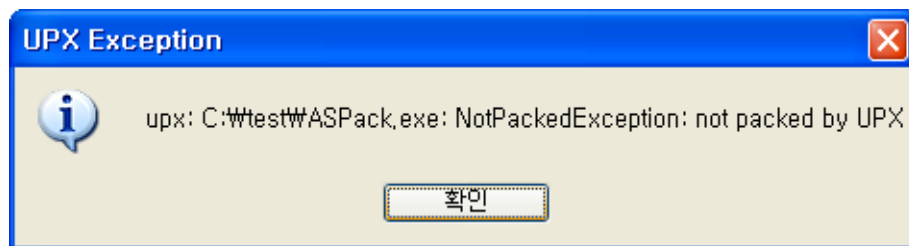
(그림 80) 실행 압축 해제 후 PEiD로 확인한 결과

실행 압축 해제 작업 완료 후 PEiD로 확인한 결과 Visual Basic 프로그램으로 작성된 것을 확인할 수 있었다.



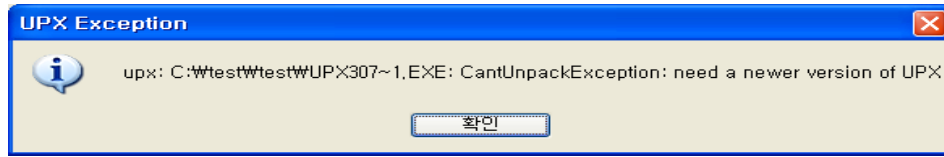
(그림 81) UPX-iT History 확인 결과

UPX-iT은 Option-History에서 UPX-iT으로 실행 압축 해제했던 History 들을 확인 할 수 있다. History에서는 실행 압축 해제 된 후의 파일 크기와 실행 압축 해제에 걸린 시간, 날짜 등을 확인 할 수 있다.



(그림 82) UPX가 아닌 파일에 대한 결과

UPX가 아닌 다른 실행 압축된 파일로 실행한 결과 Exception 창이 실행 되는 것을 볼 수 있다. UPX가 아닌 다른 실행 압축 도구로 실행 압축된 파일에 대해서는 실행 압축 해제가 실행되지 않았다. 이것을 보고 다른 실행 압축 도구의 전용 실행 압축 해제 도구와 같이 정해진 버전이나 실행 압축 도구에 대해서만 실행되는 것을 볼 수 있다.



(그림 83) 최신 버전 UPX(3.07)로 실행 압축된 파일을 실행 압축 해제한 결과

또한 UPX 최신 버전인 3.07로 실행 압축한 파일에 대해서는 새로운 버전의 UPX가 필요하다는 오류 메시지와 함께 실행 압축 해제가 수행되지 않았다. 이 프로그램은 UPX와의 연동을 통하여 UPX에서 제공하는 -d 옵션(실행 압축 해제)을 이용한 실행인 것으로 추측된다.

변형되지 않은 UPX와 설정된 UPX 보다 낮은 버전에 대해서는 실행 압축 해제가 성공적으로 수행된다. 하지만 변형된 UPX에 대해서는 실행 압축 해제가 수행되지 않는 한계가 존재한다.

7. VMUnpacker V1.3

VMUnpacker는 중국 Sucop[14]에서 개발하여 공개한 툴이다. 이 툴은 가상 머신 기술을 기반으로 동작하고, 알려지거나 알려지지 않은 실행 압축 도구를 실행 압축 해제 할 수 있다. 모든 코드는 가상 머신 기반으로 작동하기 때문에 악성코드 분석에서 시스템의 감염이나 변형에서 안전하다.



(그림 84) VMUnpacker 기본 화면

이 프로그램에서 제공하는 테스트 결과에 의하면, 61개의 실행 압축 도구에 300여개 이상 버전에 대해 지원을 한다.

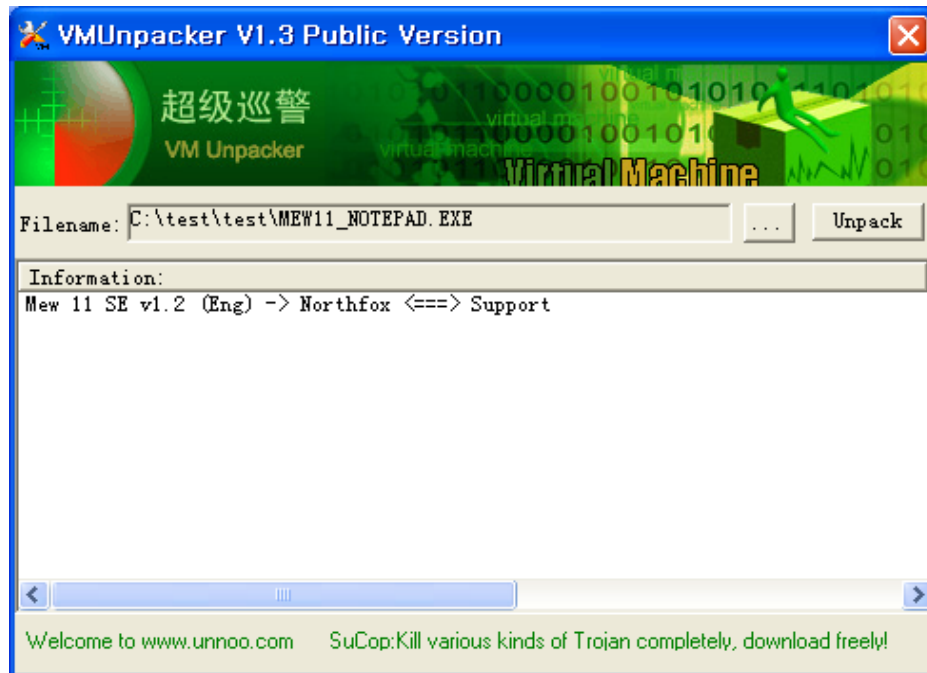
아래는 VMUnpacker v1.3에서 지원하는 실행 압축 도구의 목록이다.

- upx 0.5x-3.01 All Version
- BeRoEXEPacker
- aspack 1.x--2.x All Version
- PEcompact 0.90--1.76 2.06--2.79 All Version
- fsg v1.0 v1.1 v1.2 v1.3 v1.31 v1.33 v2.0 All Version
- vgcrypt v0.75
- nspack 1.4--4.1 All Version
- expressor v1.0 v1.1 v1.2 v1.3 v1.4 v1.501
- npack v1.5 v2.5 v3.0
- dxdpack v0.86 v1.0

- !epack v1.0 !epack v1.4
- bjfmt v1.2 v1.3
- mew5 mew v1.0 v1.1
- packman v1.0
- PEDiminisher v0.1
- pex v0.99
- petite v1.2 v1.3 v1.4 v2.2 v2.3 All Version
- winkript v1.0
- pklite32
- pepack v0.99 v1.0
- pcshrinker v0.71
- wwpack32 1.0--1.2
- upack 0.1--0.399
- rlpack 1.11--1.19
- exe32pack v1.42
- kbys v0.22 v0.28
- yoda's protector v1.02 v1.025 v1.03.2 v1.03.3
- yoda's crypt v1.1
- yoda's crypt v1.2 v1.3 v1.xModify
- XJ
- exestealth 2.72--2.76
- hidepe v1.0 v1.1
- jdpack v1.01 v2.1 v2.13
- jdprotect 0.9b
- PEncrypt v3.0 v3.1 v4.0
- Stone's PE Crypt v1.13
- telock v0.42 v0.51 v0.60 v0.70 v0.71 v0.80 v0.85 v0.90 v0.92 v0.95 v0.96
v0.98 v0.99
- ezip
- hmimys_pack v1.0
- lamecrypt v1.0
- depack

- polyene v0.01
- dragonArmour
- EP Protector v0.3
- PackItBitch
- trojan_protect
- anti007 v2.5 v2.6
- mkfpack
- yzpack v1.1 v2.0
- spack method1 spack method2
- naked packer v1.0
- upolyx v0.51
- stealthPE v1.01 stealthPE v2.2
- mslrh v0.31 v0.32
- mslrh v0.2 == [G!X]'s Protect
- morphine v1.3 morphine v1.6 morphine v2.7
- rlpack full edition
- EXEFog v1.1
- ASDPack
- PEBundle
- Neolite

VMUnpacker에 파일을 등록하면 프로그램에서 실행 압축 도구의 정보와 버전을 확인하고, 실행 압축 해제를 지원하는 버전인지 확인하여 지원하는 실행 압축 도구에 대해서는 Support라고 표시를 해주고 지원하지 않는 실행 압축 도구에 대해서는 Can't Unpack 이라고 표시를 해준다. 또한, 실행 압축 도구의 정보를 확인할 수 없는 경우에는 Unidentified packer or not a PE file 이라고 표시를 해준다.



(그림 85) Mew 11로 실행 압축된 파일을 등록한 화면



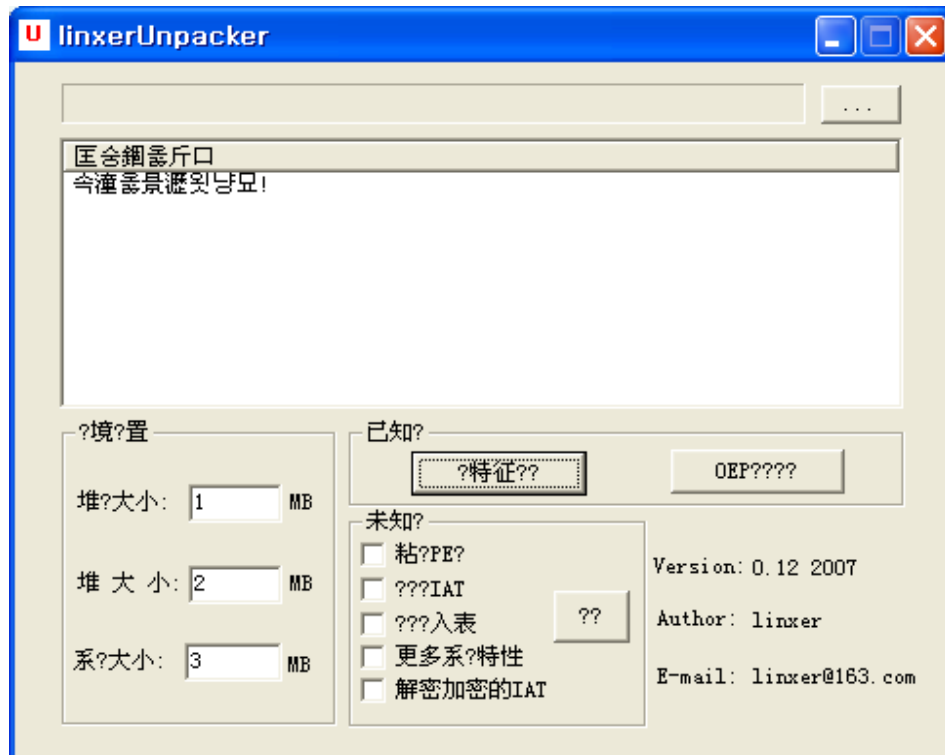
(그림 86) 실행 압축 해제를 실행한 결과

테스트 결과 Support라고 표시된 실행 압축 프로그램 대부분이 실행 압축 해제가 진행되었다. 하지만 실행 압축 해제가 제대로 실행되었으나 프로그램 실행 결과 실행이 되지 않는 경우가 발생하였다. 예를 들어 PEcompact의 모든 버전에 대해서 실행 압축 해제를 지원한다고 되어 있었으나 PECompact 1.68 버전으로 실행 압축된 파일을 실행 압축 해제 했을 때 성공이라고 출력되었으나 프로그램 실행 시 프로시저를 시작점을 DLL에서 찾지 못 했다는 메시지 창이 실행되면서 프로그램이 종료되었다.

8. LinxerUnpacker

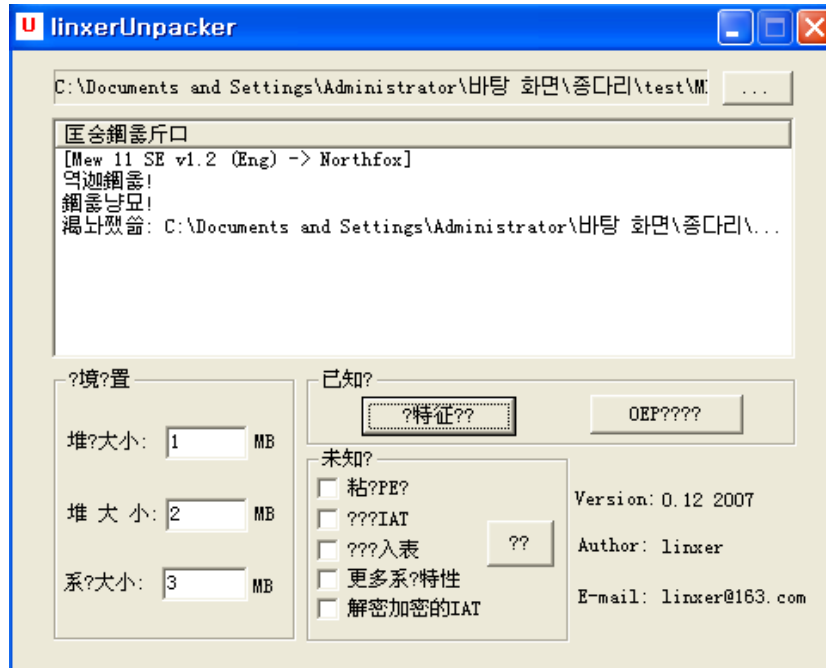
LinxerUnpacker는 2007년 중국의 Unnoo-Data Security Workbase Lab의 Linxer가 만들었다. Linxer은 2007년에 열린 xCon2007 (XFocus Information Security Conference)에서 “Unpacking Technology for AV Engine” 이라는 제목으로 발표를 했었다. xCON은 중국의 언더그라운드 해커인 xfocus의 멤버들이 진행하고 있으며, 미국국방부, 미국 제1정보작전 통제본부, 일본 blackhat의 organiser, 대만 DrayTek, 홍콩, 브라질, 싱가포르, 러시아등 여러 국가의 해커들이 참여하는 국제적인 컨퍼런스이다.

linxerUnpacker의 시작화면은 아래와 같다. 중국어로 된 프로그램이기 때문에 각 옵션에 대한 기능들에 대해서는 파악하지 못했고, 실행 압축 해제의 성공률에 대해서만 초점을 맞추고 조사를 하였다.



(그림 87) linxerUnpacker 실행 화면

linxerUnpacker는 약 80여개의 실행 압축 도구에 대한 실행 압축 해제를 제공하고 있다. linxer에서 실행 압축된 파일을 open하면 실행 압축 도구에 대한 정보를 제공한다. linxerUnpacker에 제공되는 파일 중 PEiD의 시그니처 파일이 함께 제공됐는데, 이 파일에서 제공되는 정보와 같은 데이터가 출력이 된다. 따라서 PEiD와 같이 PE의 정보를 통해 실행 압축 도구의 종류를 파악하는 것이라고 생각된다.



(그림 88) MEW 1.1을 실행 압축 해제한 화면

linxerUnpacker를 통해 실행 압축 해제를 실행하면, 실행 압축 파일이 있는 폴더에 Unpacked라는 파일 이름으로 실행 압축 해제된 파일이 생성이 된다. UPX, MEW, PEcompact, ASpack 등 많이 쓰이는 실행 압축 도구들로 실행 압축된 파일로 테스트해봤을 때, 대부분의 경우 실행 압축 해제가 잘 이뤄졌으며 파일의 실행까지 원활하게 이루어졌다. 하지만 Themida에 대한 실행 압축 해제를 실행해 봤을때는 실행 압축해제가 이뤄지지 않았다.

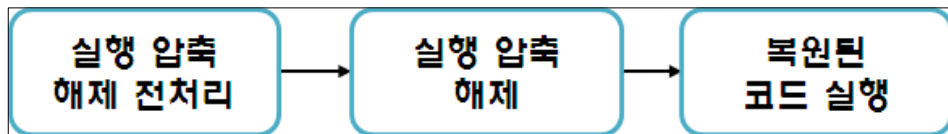
참고문헌

- [1] Yang-seo Choi, Ik-kyun Kim, Jin-tae Oh, Jae-cheol Ryou, "PE File Header Analysis-Based Packed PE File Detection Technique (PHAD)," International Symposium on Computer Science and its Applications, pp. 28~31, 2008.
- [2] Roberto Perdisci, Andrea Lanzi, Wenke Lee, "Classification of packed executables for accurate computer virus detection, Pattern Recognition Letters" Volume 29, Issue 14, Pages 1941-1946, 15 October 2008.
- [3] 정구현, 추의진, 이주석, 이희조, "엔트로피를 이용한 실행 압축 해제 기법 연구", 한국정보기술학회논문지, 한국정보기술학회, 제7권 제1호, 232~238, 2009년
- [4] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. "Renovo: A Hidden Code Extractor for Packed Executables," In Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM'07), 2007.
- [5] TEMU: The BitBlaze Dynamic Analysis Component.
<http://bitblaze.cs.berkeley.edu/temu.html>
- [6] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, Wenke Lee, "PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware", Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual, 289~300, 2006.
- [7] UPX. <http://upx.sourceforge.net>.
- [8] OllyDbg. <http://www.ollydbg.de>.
- [9] Faster Universal Unpacker. <http://code.google.com/p/fuu>.
- [10] TitanEngine.
<http://www.reversinglabs.com/products/TitanEngine.php>.
- [11] ALIEN Hack Team. <http://qunpack.ahteam.org>.
- [12] Reversing labs. www.reversinglabs.com.
- [13] Oreans Technology. <http://www.oreans.com>.
- [14] Sucop. <http://sucop.com>.

- [15] Silvio Cesare, Yang Xiang, "Classification of Malware Using Structured control Flow", Proceeding AusPDC '10 Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107, 2010.
- [16] Piotr Bania, "Generic Unpacking of Selfmodifying, Aggressive, Packed Binary Programs", <http://piotrbania.com/all/articles/pbania-dbi-unpacking2009.pdf>, 2009.
- [17] Monirul Sharif, Andrea Lanzi, Jonathon Giffin, Wenke Lee, "Rotalume : A Tool for Automatic Reverse Engineering of Malware Emulators", 2009.
- [18] Lingyun YING, Purui SU, Dengguo FENG, Xianggen WANG, Yi YANG, Yu LIU, "ReconBin: Reconstructing Binary File from Execution for Software Analysis", Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on, 2009.
- [19] Guhyeon Jeong, Euijin Choo, Joosuk Lee, Munkhbayar Bat-Erdene, Heejo Lee, "Generic Unpacking using Entropy Analysis", 2010 5th International Conference on Malicious and Unwanted Software, 2010.
- [20] Saumya Debray, Jay Patel, "Reverse Engineering Self-Modifying Code : Unpacker Extraction", WCRE 2010 : 17th Working Conference on Reverse Engineering, 2010.
- [21] Abhiram Kasina, Amit Suthar, Rajeev Kumar, "Detection of Polymorphic Viruses in Windows Executables", Communications in Computer and Information Science, 2010.
- [22] Robert Lyda, James Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware,"IEEE Security and Privacy, Vol. 5, no. 2, pp. 40-45, Mar/Apr, 2007.

제 4 장 제안하는 실행 압축 무력화 방법

이 장에서는 제안하는 실행 압축 무력화 방법에 대하여 설명한다. 실행 압축 파일을 실행하면 메모리에 값을 읽고 쓰면서 실행 압축 해제 과정을 거치게 된다. 실행 압축 해제가 완료되면 그 다음부터 복원된 실제의 프로그램이 실행되게 된다. 다음 (그림 89)는 실행 압축 파일 실행 과정을 나타내는 그림이다.



(그림 89) 실행 압축 파일 실행 과정

실행 압축 파일의 실행 과정에서 엔트로피 값은 초기 실행 부분에서는 아직 실행 압축 해제가 되지 않았기 때문에 랜덤성이 높아 엔트로피의 값이 높게 나온다. 실행 압축 파일이 실행되면서 이후에는 압축 해제 과정이 진행되므로 랜덤성이 떨어져 엔트로피 값이 작아진다. 또한 압축 해제 완료된 시점에서부터는 엔트로피의 값이 크게 변하지 않게 되므로 엔트로피 그래프는 엔트로피의 변화가 일정 범위에서 조금씩 변화하는 모습을 나타내게 된다. 기존의 연구에서의 문제점은 엔트로피의 값으로 실행 압축되어 있는지 압축이 풀렸는지 판단한다는 것이다. 일반적으로 실행 압축된 파일의 엔트로피 값은 실행 압축 안 된 파일에 비해 크지만 특정 파일의 엔트로피 값은 실행 압축이 되어 있지 않아도 기존 연구에서 주장하는 실행 압축되었다고 판단되는 엔트로피 값의 범위에 들어가는 경우도 있다. 그리고 실행 압축 과정에서 코드 섹션에 쓰레기 값을 넣어 인위적으로 엔트로피 값을 변화 시킬 수도 있기 때문에 문제가 존재한다.

이번에 제안하는 방법은 랜덤성을 측정하기 위하여 엔트로피를 사용하지만 엔트로피의 값만으로 실행 압축 여부 및 오리지널 엔트리 포인트를 찾지 않고 엔트로피의 값의 변화량을 가지고 실행 압축 무력화를 수행한

다. 제안하는 방법의 가장 중요한 포인트는 실행 압축 파일 실행 과정 중 실행 압축이 해제 되고 복원된 코드가 실행 하게 되면 랜덤성이 크게 변하지 않는 엔트로피 그래프 구간이 생긴다는 것이다.

가상환경에서 실행 압축 프로그램을 실행시켜 종료시점까지의 각 명령어마다의 엔트로피 변화를 그래프로 표현하고 보면 종료 시점 부분에는 복원된 코드가 실행되어 엔트로피 값이 특정 범위 내의 값을 나타낸다. 종료 시점부터 엔트로피 변화량을 역분석하여 특정 범위내의 변화를 넘어서는 부분이 나오면 그 구간은 오리지날 엔트리 포인트가 있는 구간이 된다. 따라서 이 구간에서의 Jump나 call 명령어를 찾고 Jump나 call 명령어 이후의 도착지점이 오리지날 엔트리 포인트가 되게 된다.

제 1 절 엔트로피 계산

실험 데이터로 구구단을 1단부터 9단까지 출력하게 하는 PE 파일(7KB)과 사다리 게임 파일(16.5KB) 그리고 악성코드 샘플 하나(12KB), 총 3가지의 PE 파일에 대한 일반적인 엔트로피 측정과 UPX, ASpack, FSG 총 3가지의 실행 압축 도구를 사용하여 실행 압축하고 해당 PE 파일에 대한 엔트로피 측정을 하였다. 실험에 사용된 악성코드는 2007년에 발견된 것으로 악성코드 제거 프로그램을 강제로 설치하고 계속되는 경고 메시지와 팝업, 종료 불가, 강제 검사와 업데이트 등의 현상을 보이는 악성코드이다.

실행 압축 전의 각 파일의 엔트로피 값을 측정해보았다.

[표 5] 정상 파일의 엔트로피 측정 결과

파일명	엔트로피 값
구구단 프로그램	6.9
사다리 게임 프로그램	6.8
악성코드	7.7

이전의 엔트로피를 이용한 무력화 연구에서는 실행 압축된 파일의 엔트로피 값이 7.1 ~ 7.2 정도 된다면 실행 압축되었다고 판단하였다. 하지만 일반적인 악성코드 안에서도 7.7 의 엔트로피 값이 측정되는 악성코드가

발견되었다. 따라서 엔트로피 값으로 실행 압축 여부를 판단하는 것은 오탐을 일으킬 가능성이 존재한다는 것이다. 또한 특정 실행 압축 기법이 이러한 엔트로피 값의 탐지를 피하고자 쓰레기 값을 코드에 넣어 엔트로피 값을 변화시킨다면 해당 방법은 많은 오탐을 일으키게 될 것이다.

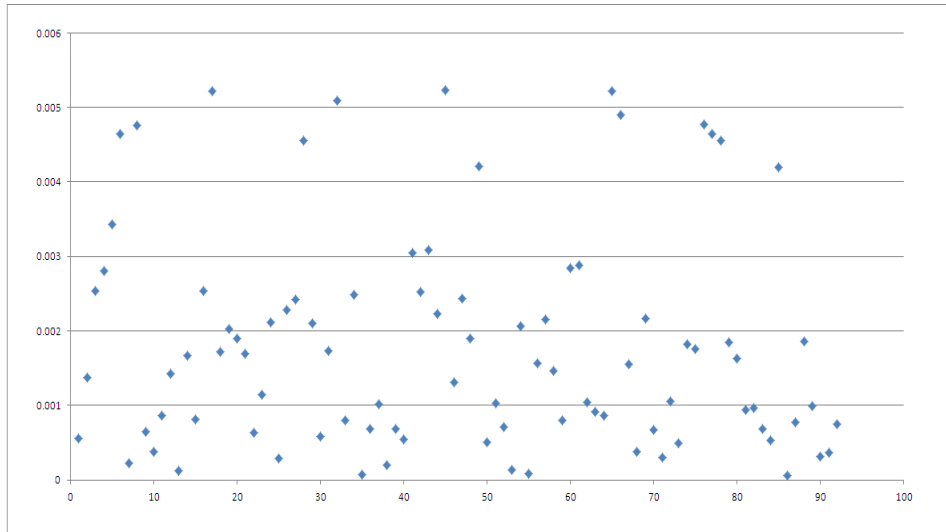
다음은 실행 압축 후의 엔트로피 값을 측정한 것이다.

[표 6] 실행 압축 파일의 엔트로피 측정 결과

파일명	엔트로피 값
구구단 프로그램 (UPX)	7.5
구구단 프로그램 (ASpack)	7.6
구구단 프로그램 (FSG)	7.5
사다리 게임 프로그램 (UPX)	7.7
사다리 게임 프로그램 (ASpack)	7.7
사다리 게임 프로그램 (FSG)	7.6
악성코드 (ASpack)	7.8
악성코드 (FSG)	7.9

악성코드의 크기가 작은 관계로 UPX를 이용한 실행 압축은 불가능하였다. 또한 제안하는 방법의 핵심인 정상적인 프로그램이 실행되었을 때의 엔트로피 값이 얼마나 변화하는지 실험 하였다. 실험에 쓰인 정상적인 프로그램은 Window XP의 System32 내에서 존재하는 여러 실행 파일 중 랜덤하게 100개를 뽑아 측정하였다. 프로그램이 종료하는 시점까지의 매 명령어마다 메모리 덤프를 수행하고 덤프 파일에서 엔트로피 값을 구한다.

다음은 실험 결과로 나온 정상적인 프로그램의 엔트로피 값 범위를 나타내는 그림과 표이다. 범위 값의 측정은 정상파일의 엔트로피 값의 최대값에서 최소값을 차이를 이용하여 측정하였다.



(그림 90) 정상 파일에서의 엔트로피 값의 범위

정상적인 실행 파일의 엔트로피 범위의 최대값은 0.005231로 측정되었고, 최소값은 0.000064로 측정되었고, 엔트로피 범위의 평균은 0.001805696로 측정되었다. 실험에 쓰였던 실행 압축된 사다리 프로그램의 엔트로피 값이 UPX는 0.3, ASpack은 0.8, FSG는 0.4 정도의 범위에서 변화하는 것을 봤을 때 실험 결과에서 보이는 엔트로피 값은 아주 작은 범위 안에서 변화하는 것을 알 수 있었다.

[표 7] 정상적인 실행 파일의 엔트로피 범위 값

엔트로피 범위의 최대값	엔트로피 범위의 최소값	엔트로피 범위의 평균값
0.005231	0.000064	0.001805696

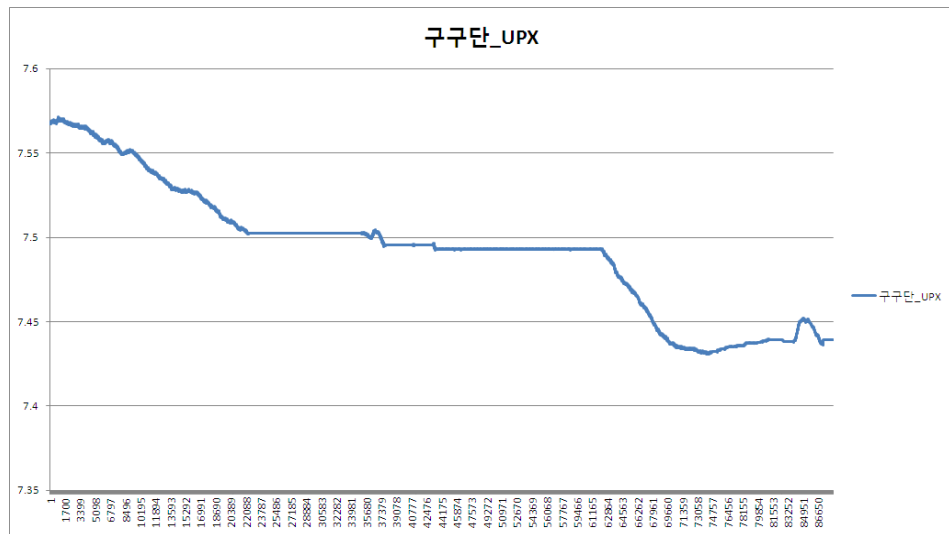
제 2 절 엔트로피 그래프 역 분석

실행 압축 파일은 실행되면 실행 압축을 해제하고 해제가 완료되는 시점

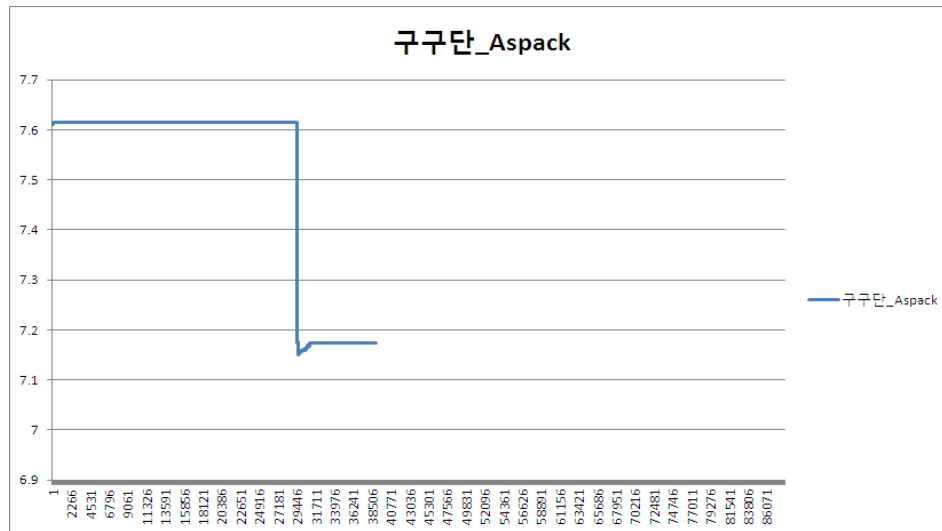
에부터 다시 원래의 복원된 코드를 실행하는 구조이므로 실행 압축 파일의 실행 시 나오는 압축 해제과정과 복원 코드를 실행하는 부분의 전체적인 엔트로피 그래프 개형을 분석하기 위해 1절에서는 OllyDbg에서 제공하는 OllyScript를 이용하여 매 명령어마다 메모리 덤프를 수행하고 해당 덤프 파일의 엔트로피 값을 측정하여 그래프로 그렸다.

제안하는 방법에서는 종료시점부터 역으로 엔트로피 값의 변화량을 분석하므로 초기의 엔트로피 그래프의 개형이 어떠한 변화를 가지고 오던지 상관이 없다. 따라서 이번 절에서는 결과 값으로 나온 엔트로피 그래프 종료시점부터 앞에서 나온 정상적인 코드가 실행 될 때의 엔트로피 값 변화 임계치를 넘는 시점이 나오는 구간까지의 그래프를 그려보았다.

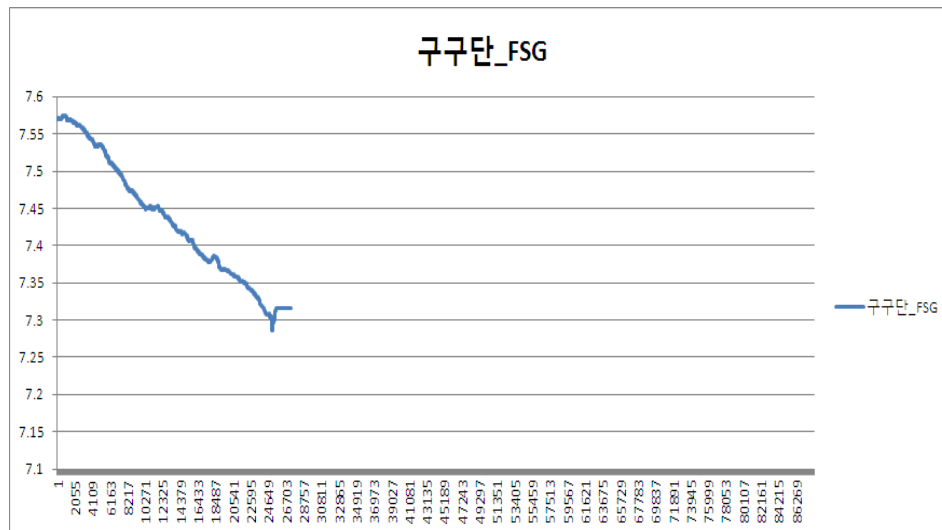
그래프의 X축은 명령어의 실행 순서를 나타내고 Y축은 엔트로피 값을 나타낸다. 다음 그림은 구구단 프로그램부터 사다리 게임 프로그램, 악성 코드 순으로 UPX, ASpack, FSG 순으로 실행 압축된 파일의 전체 엔트로피 그래프를 나타낸 그림이다.



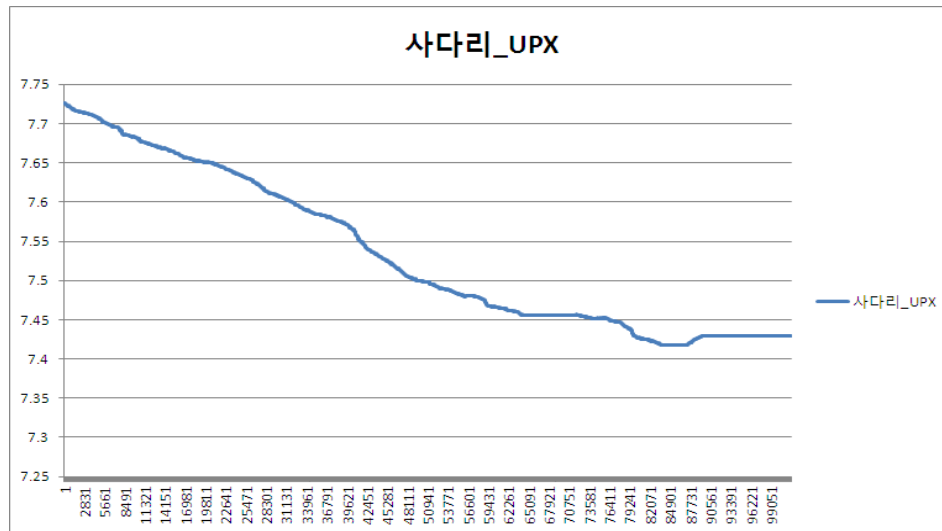
(그림 91) UPX로 실행 압축된 구구단 프로그램의 엔트로피 그래프



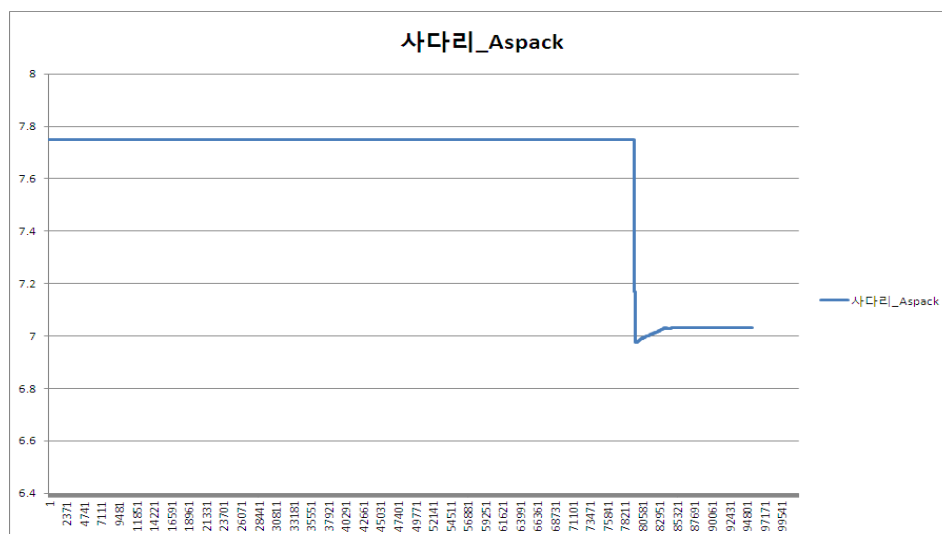
(그림 92) Aspack로 실행 압축된 구구단 프로그램의 엔트로피 그래프



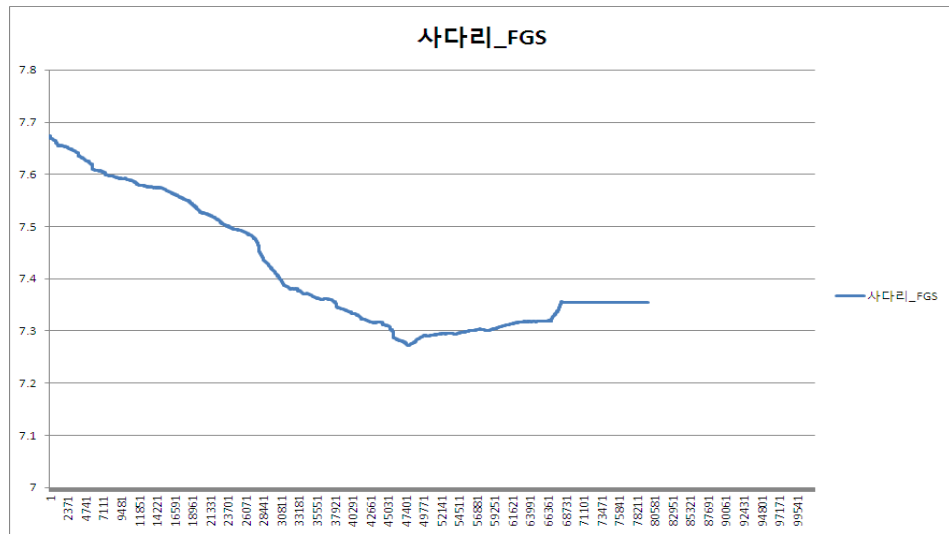
(그림 93) FSG로 실행 압축된 구구단 프로그램의 엔트로피 그래프



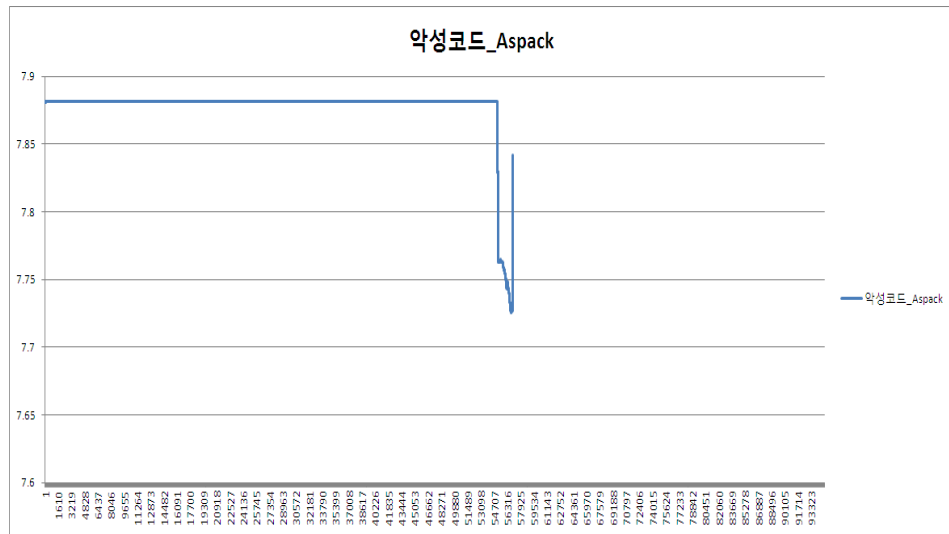
(그림 94) UPX로 실행 압축된 사다리 게임 파일의 엔트로피 그래프



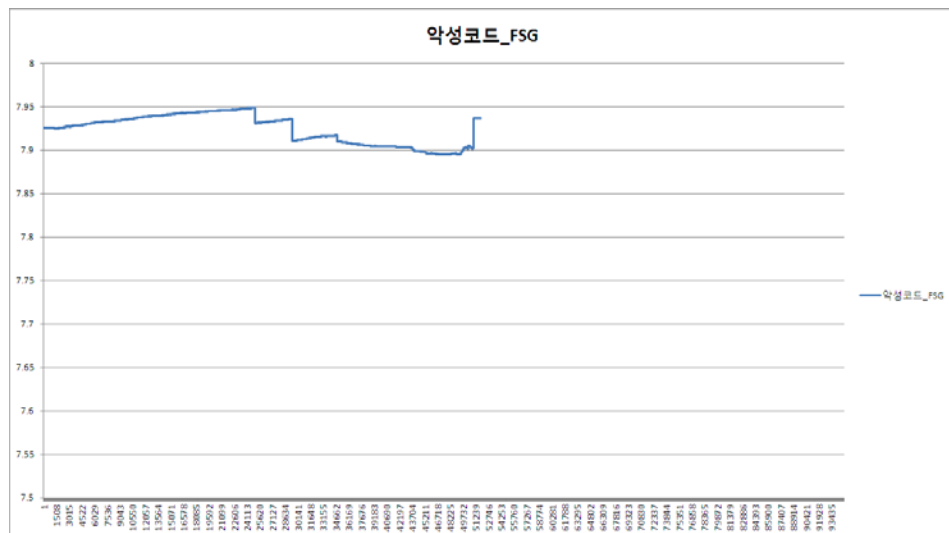
(그림 95) ASpack로 실행 압축된 사다리 게임 파일의 엔트로피 그래프



(그림 96) FSG로 실행 압축된 사다리 게임 파일의 엔트로피 그래프



(그림 97) ASpack로 실행 압축된 악성코드 파일의 엔트로피 그래프

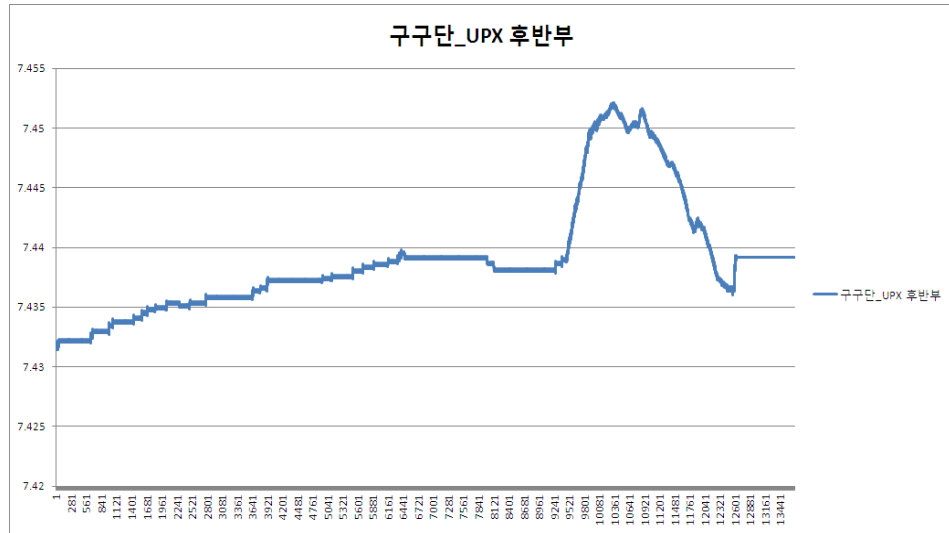


(그림 98) FSG로 실행 압축된 사다리 게임 파일의 엔트로피 그래프

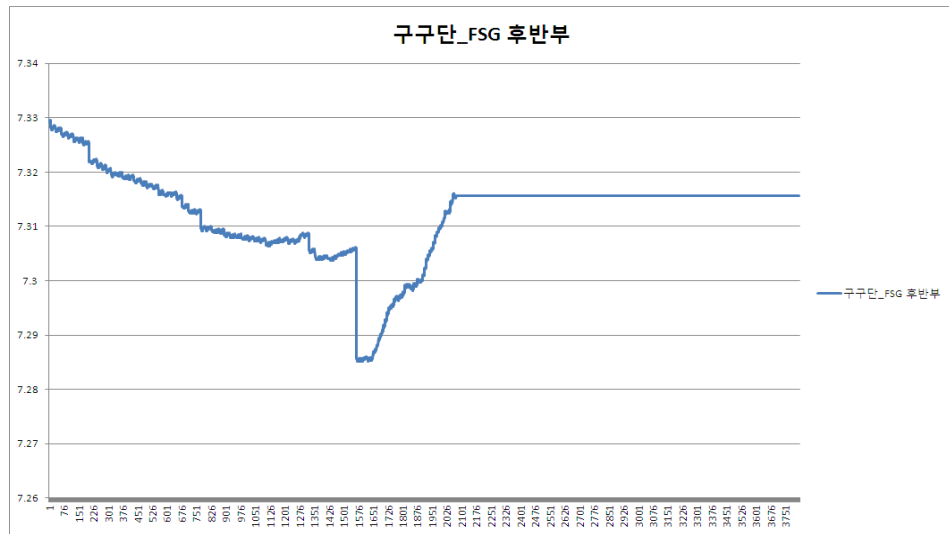
위의 그림들은 앞에서 설명한 방법으로 엔트로피 값을 측정하여 그래프

를 그려본 것이다. 오리지날 엔트리 포인트를 찾기 위해 종료시점부터의 엔트로피 값의 변화를 보며 정상적인 파일의 엔트로피 값 변화량 이상으로 변화되는 시점을 찾게 된다.

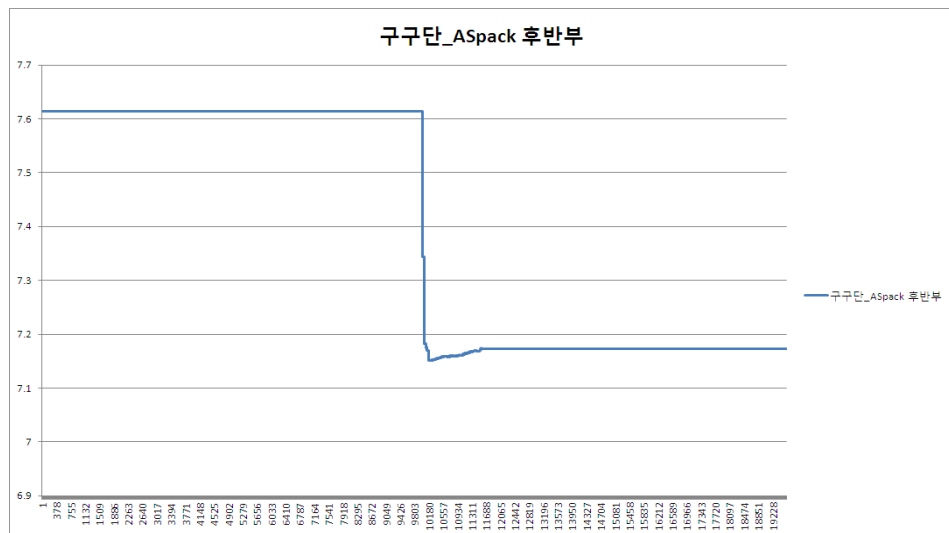
다음 그림은 종료 시점부터 엔트로피 값의 변화를 보기 위해 전체의 그래프가 아닌 후반부의 그래프 개형만을 나타내는 그림이다.



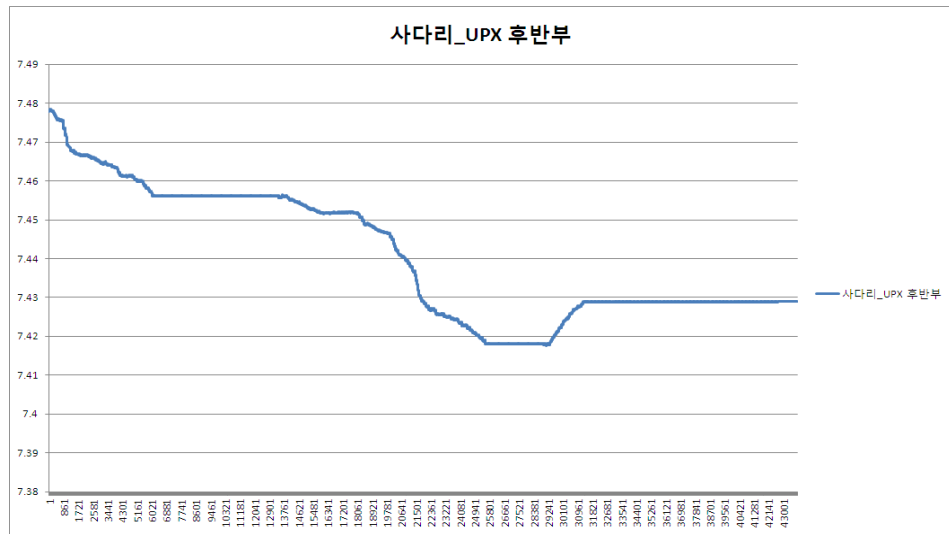
(그림 99) UPX로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프



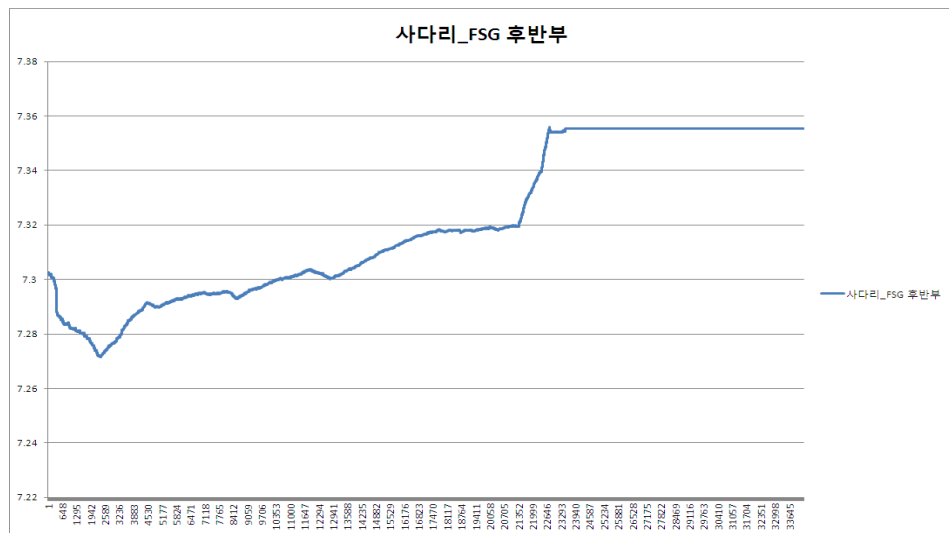
(그림 100) FSG로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프



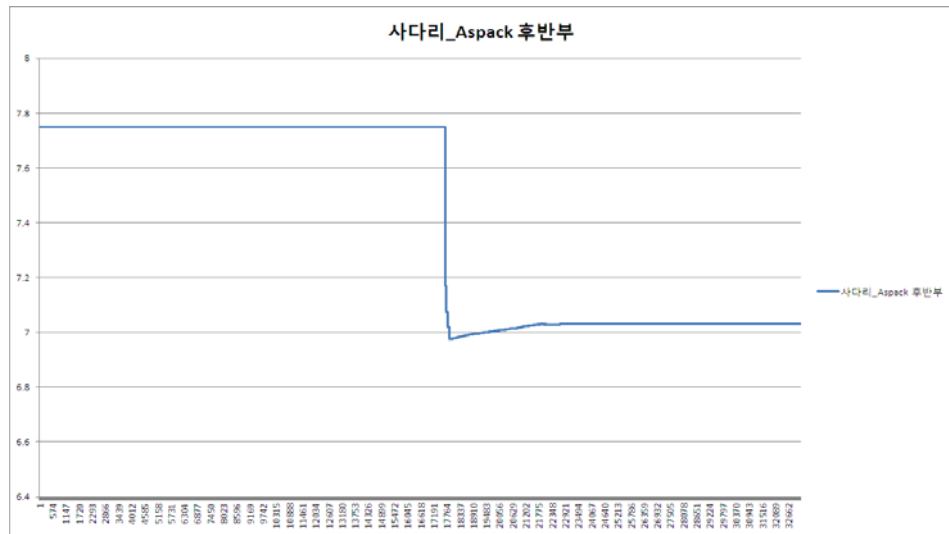
(그림 101) ASpack으로 실행 압축된 구구단 프로그램의 후반부 엔트로피 그래프



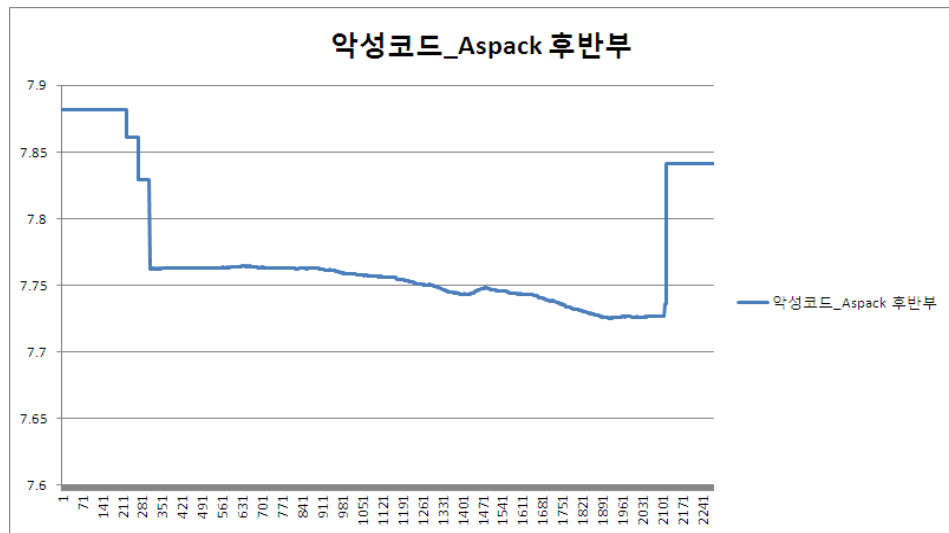
(그림 102) UPX로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프



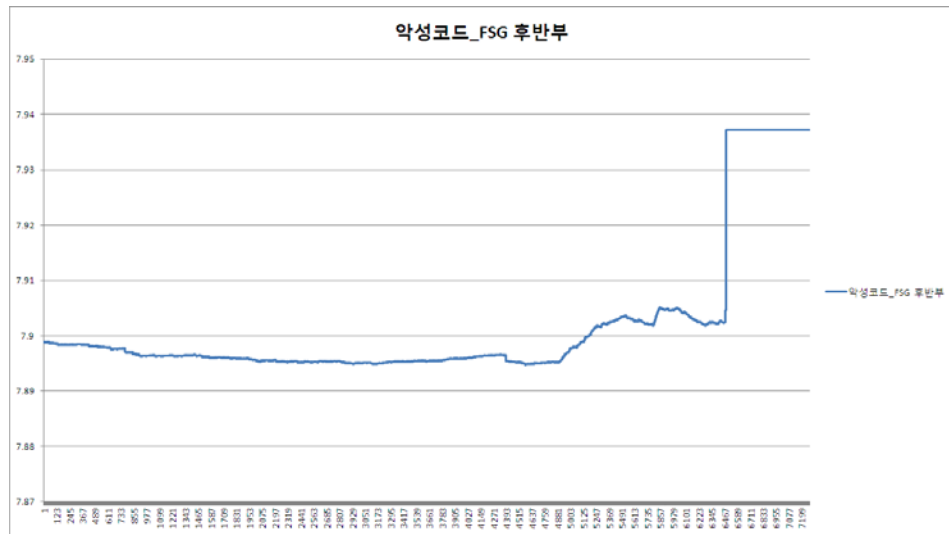
(그림 103) FSG로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프



(그림 104) ASpack로 실행 압축된 사다리 게임 프로그램의 후반부 엔트로피 그래프



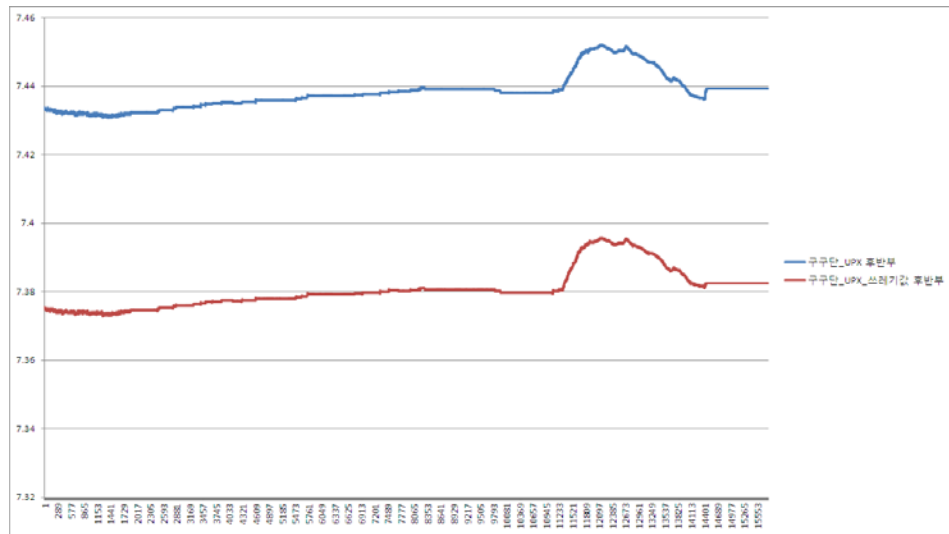
(그림 105) ASpack로 실행 압축된 악성코드의 후반부 엔트로피 그래프



(그림 106) FSG로 실행 압축된 악성코드의 후반부 엔트로피 그래프

이전 연구에서의 단점인 실행 압축 파일에 쓰레기 값이 들어가서 엔트로피의 값을 변화시킬 경우 실행 압축 탐지와 오리지널 엔트리 포인트 시점을 찾기 어려웠던 단점은 본 보고서에서 제안하는 엔트로피 값의 변화량을 통한 분석으로 하면 해결된다. 쓰레기 값이 들어가서 엔트로피 값이 변화하여도 그래프의 개형의 변화는 없기 때문에 충분히 오리지널 엔트리 포인트를 찾을 수 있게 된다.

실험에 사용된 UPX로 실행 압축한 구구단 프로그램은 코드섹션에 이용하지 않는 부분에 쓰레기 값을 추가하여 프로그램의 실행에는 영향을 주지 않고 엔트로피 값만 변화 할 수 있게 설정 하였다. 다음 그림은 이와 같은 방법으로 실험하였을 때 나오는 결과 값이다.



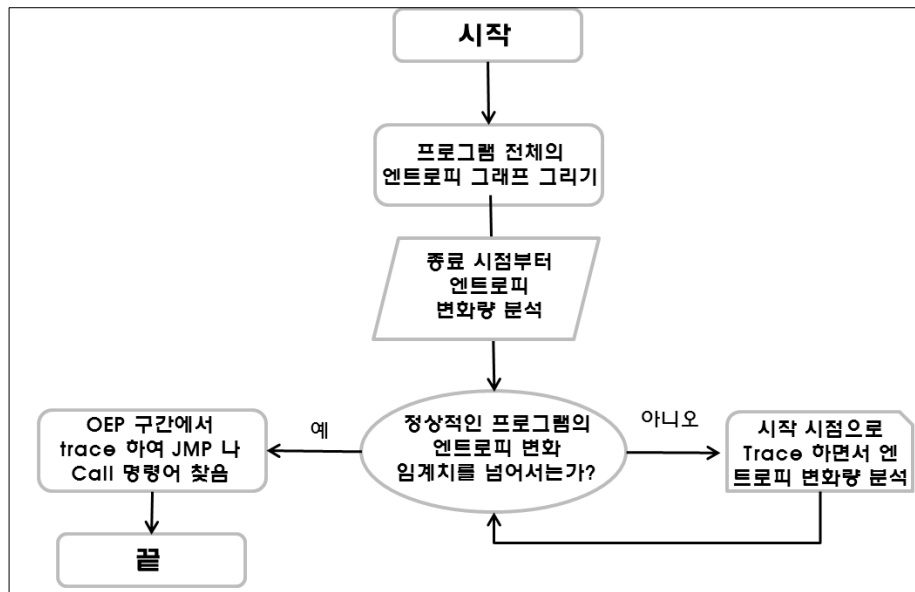
(그림 107) 쓰레기 코드 삽입한 파일의 엔트로피 그래프 개형 비교

제 3 절 오리지널 엔트리 포인트 도출

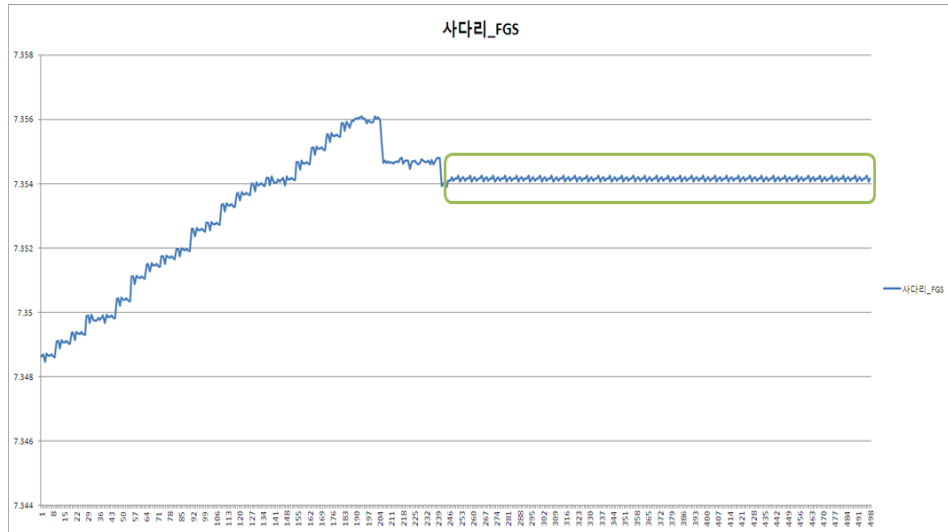
종료 시점부터 엔트로피 변화량을 역분석하여 엔트로피 변화 값이 정상적인 파일의 엔트로피 변화 범위(± 0.0005)를 넘어서는 부분이 나오면 그 구간을 오리지널 엔트리 포인트가 있는 구간으로 추정한다. 따라서 이 구간에서의 Jump나 call 명령어를 찾고 Jump나 call 명령어 실행 후 주소가 오리지널 엔트리 포인트가 되게 된다.

본 보고서에서의 제안 방법으로 FSG로 실행 압축된 사다리 게임의 오리지널 엔트리 포인트를 도출하고 이를 검증한다.

제안하는 실험 방법은 다음과 같다.

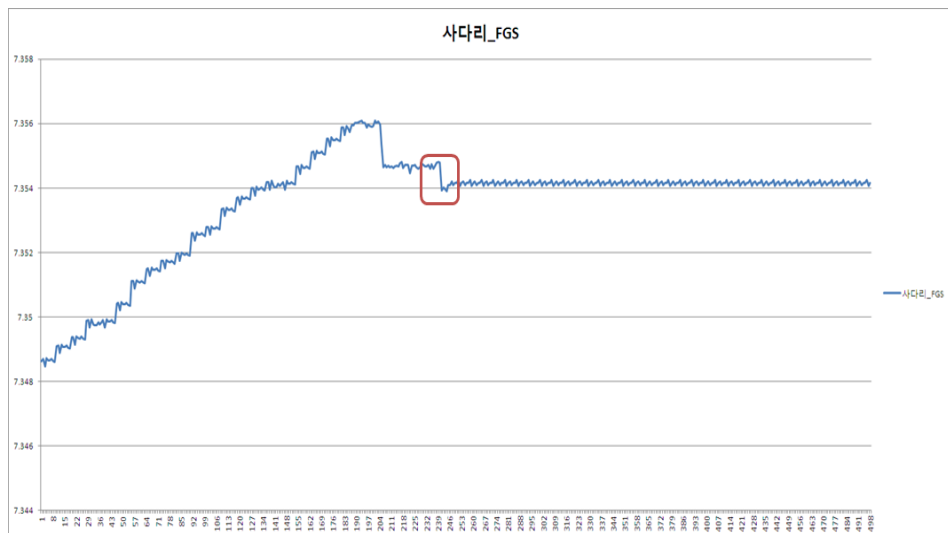


(그림 108) 제안하는 기법의 전체 구성도



(그림 109) 종료시점부터의 엔트로피 변화 값 평균을 구하는 화면

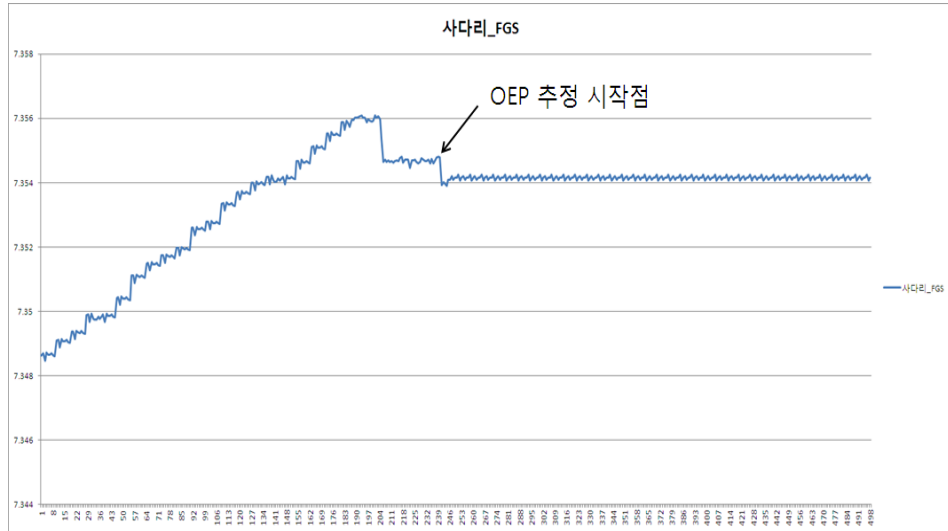
다음 (그림 109)에서 박스안의 실험 결과 변화값의 평균은 0.0001 이 나온다.



(그림 110) 엔트로피 값의 변화량이 특정범위를 넘어가는 구간 확인

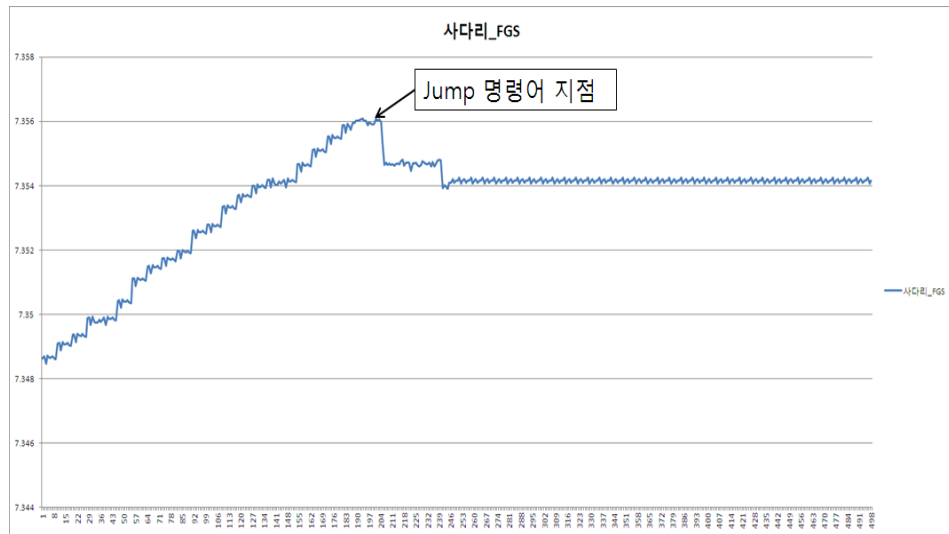
(그림 110)에서 보듯이 해당 구간에서는 엔트로피의 변화량이 0.005 로

기존의 변화량 보다 5배의 차이를 보이며 정상적인 파일의 변화 범위인 ± 0.0005 을 넘어서 변화함을 확인 할 수 있다.



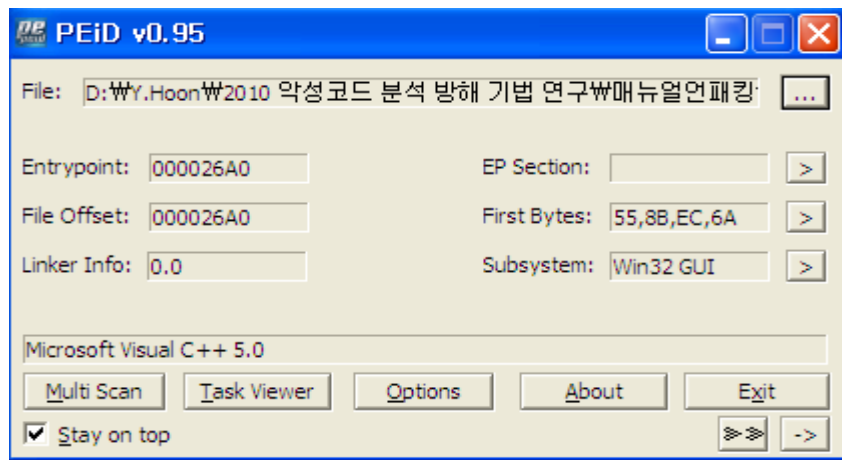
(그림 111) OEP 추정 시작점 확인 화면

다음 오리지날 엔트리 포인트 추정 시작점부터 Jump 명령어를 검색한다. 다음 그림은 추정 시작점부터 Jump 명령어를 찾아 Jump 시점을 찾은 화면이다.



(그림 112) Jump 명령어 지점 확인 화면

위의 그림에서의 Jump 명령어 지점에서 이동하는 주소는 004026A0 이다. 오리지널 엔트리 포인트라고 확인된 해당 주소에서 덤프 파일을 IAT 테이블을 복원해주는 프로그램을 통하여 IAT 테이블을 복원하였다.



(그림 113) 실행 압축 해제 완료 확인 화면

PEID에서 확인 한 결과 실행 압축이 해제되었음을 확인 할 수 있다.

참고문헌

- [1] Robert Lyda, James Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," IEEE Security and Privacy, Vol. 5, no. 2, pp. 40-45, Mar/Apr, 2007.
- [2] Thomas M. Cover and Joy A. Thomas, "Elements of Information Theory," Second Edition. Wiley Interscience, New York, NY, 2006.

제 5 장 결 론

지금까지 실행 압축 기술의 동향 및 통계를 조사하고 실행 압축 기술을 무력화 시키는 실행 압축 해제 방법을 조사하였다. 그리고 엔트로피 그래프 변화량을 이용한 실행 압축 무력화 방법을 제안하였다.

제안하는 방법을 검증하기 위해 3가지의 실행 파일과 3가지의 실행 압축 기법을 사용하였다. 그리고 정상적인 프로그램의 엔트로피 변화량을 확인하기 위하여 100여개의 정상적인 프로그램의 엔트로피 변화량을 측정하였다. 하지만 제안하는 방법의 정확한 검증을 위해서는 보고서에서 사용한 실험 데이터 이외의 더 많은 실행 파일과 실행 압축 기법을 적용하여야 할 것이다.

제안된 방법은 앞으로 효율성을 고려하여 개선되어야 한다. 즉 엔트로피 변화값을 측정하는 과정에서의 효율적인 알고리즘 및 방법 개선이 필요하다.

기존의 엔트로피를 이용한 무력화 방법은 엔트로피 값만으로 실행 압축을 무력화하므로 쓰레기 값 등 엔트로피 값에 변화를 주면 오탐이 발생하는 단점이 있다. 본 보고서의 제안 방법은 엔트로피 변화량으로 실행 압축을 무력화하므로 기존의 방법의 단점을 보완하고 보다 정확하고 효율적인 분석이 가능할 것으로 기대된다.

실행 압축된 악성코드 자동 분석을 위한 무력화 알고리즘 연구

인 쇄 : 2010년 12월

발 행 : 2010년 12월

발행인 : 서 중 렬

발행처 : 한국인터넷진흥원(KISA, Korea Internet & Security Agency)

서울시 송파구 가락동 79-3 대동빌딩

Tel: (02) 4055-114

인쇄처 : 다정문화사

Tel: (02) 929-7003

〈〈 비 매 품 〉〉

1. 본 연구보고서는 정보통신진흥기금으로 수행한 정보통신연구개발사업의 연구결과입니다.
2. 본 연구보고서의 내용을 발표할 때에는 반드시 한국인터넷진흥원 정보통신연구개발사업의 연구결과임을 밝혀야 합니다.
3. 본 연구보고서의 판권은 한국인터넷진흥원이 소유하고 있으며, 당 진흥원의 허가 없이 무단 전재 및 복사를 금합니다.