情報保護學會論文誌
第 22 卷 第 2 號, 2012. 4

엔트로피 값 변화 분석을 이용한 실행 압축 해제 방법 연구*

이 영 훈,[1†] 정 만 현, 정 현 철[2‡], 손 태 식[3], 문 종 섭[1]
[1]고려대학교 정보경영공학전문대학원, [2]한국인터넷진흥원, [3]아주대학교 정보컴퓨터공학부

A Study on Generic Unpacking using Entropy Variation Analysis*

Young-hoon Lee,[1†] Man-Hyun Chung, Hyun-Cheol Jeong[2‡],
Tae-shik Shon[3], Jong-sub Moon[1]
[1]Graduate School of Information Management and Security, Korea University
[2]Korea Internet & Security Agency
[3]Division of Information and Computer Engineering, Ajou University

요 약

악성코드의 탐지 및 분석 회피 기술 중 실행 압축 기술은 악성코드의 용량을 줄이고 분석가가 코드를 분석할 때 혼란을 주도록 코드를 변형하고 있다. 따라서 악성코드의 확산이 용이해지고 분석하는데 시간이 오래 걸려 신속한 대응이 어렵게 만들고 있다. 최근에는 이러한 실행 압축된 악성코드에 대응하기 위하여 실행 압축 해제 관련 연구가 진행되고 있다. 실행 압축 프로그램은 실행되면 실행 압축을 해제하게 된다. 실행 압축 해제 때 압축되어 있던 데이터가 해제 되면서 실행 압축 파일의 데이터가 변경되거나 추가되어 데이터의 변화가 생기게 된다. 이때 이러한 변화 때문에 실행 압축 파일의 엔트로피 값이 변화하게 된다. 실행 압축 해제가 끝나게 되면 이러한 데이터 변화가 끝나고 실제적인 프로그램이 수행되므로 엔트로피 값이 변화하지 않게 된다. 그러므로 이러한 성질을 이용하여 실행 압축 해제되는 시점을 찾게 되면 실행 압축 알고리즘에 상관없이 실행 압축을 해제 할 수 있게 된다. 본 논문에서는 실행 압축 파일의 압축 해제 때의 엔트로피 값 변화량을 보고 실행 압축 해제가 끝나는 시점을 판단하여 실행 압축을 해제하는 방법을 제안한다.

ABSTRACT

Packing techniques, one of malicious code detection and analysis avoidance techniques, change code to reduce size and make analysts confused. Therefore, malwares have more time to spread out and it takes longer time to analyze them. Thus, these kind of unpacking techniques have been studied to deal with packed malicious code lately. Packed programs are unpacked during execution. When it is unpacked, the data inside of the packed program are changed. Because of these changes, the entropy value of packed program is changed. After unpacking, there will be no data changes; thus, the entropy value is not changed anymore. Therefore, packed programs could be unpacked finding the unpacking point using this characteristic regardless of packing algorithms. This paper suggests the generic unpacking mechanism using the method estimating the unpacking point through the variation of entropy values.

Keywords: Unpacking, Entropy, malware

접수일(2011년 5월 30일) 수정일 (2011년 8월 1일)
채택일(2011년 8월 23일)
* 본 연구는 방송통신위원회의 정보보호원천기술개발사업의 연구결과로 수행되었음(KCA-2011-(10914-06001))
† 주저자, wizard998@korea.ac.kr
‡ 교신저자, jsmoon@korea.ac.kr

I. 서 론

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. 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 있다. 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. 그러므로 실행 압축 기술에 대한 대응방안이 필요하게 되었고 그에 따른 연구가 진행되고 있다. 실행 압축 기술 관련 연구는 크게 두 가지로 실행 압축된 악성코드를 탐지하는 방법과 실행 압축된 악성코드를 해제하는 방법에 초점을 맞춘 연구로 나누어진다. 실행 압축 기술의 탐지에 관한 연구로는 PHAD[2], Roberto[3]의 바이너리 정적 분석 방법을 이용한 실행 압축 파일 분류, Lyda[4]의 엔트로피 분석을 기반으로 하는 실행 압축 탐지 기술이 있다. 실행 압축 파일은 실행 압축이 해제되는 부분과 압축이 해제된 실제 프로그램이 실행되는 부분으로 나눌 수 있다. OEP(Original Entry Point)란 프로그램의 실행 압축이 끝나고 실제 프로그램이 시작되는 첫 번째로 실행되는 명령어의 주소로 실행 압축 해제 기술에서는 실행 압축 파일의 OEP를 찾는 것이 가장 중요하다. 실행 압축 파일을 실행하면 메모리에 값을 읽고 쓰면서 실행 압축 해제 과정을 거치게 되고 실행 압축 해제가 완료되면 OEP 지점부터 복원된 실제의 프로그램이 실행되게 되는데 이러한 특성을 이용하여 최근에는 실행 압축 해제 방법에 관한 연구가 이루어지고 있다. 이에 본 논문에서는 엔트로피 변화를 이용한 실행 압축 해제 기법을 제안하고자 한다. 본 논문의 2장에서는 실행 압축 기법 해제와 관련된 연구에 대해 조사하였고, 3장에서는 제안하는 기법에 관한 내용을 설명하고 4장에서는 실험을 통하여 3장에서 제안한 방법을 검증한다. 마지막으로 5장에서는 본 연구의 결론 및 향후 연구에 대해 기술 하였다.

II. 실행 압축 해제 방법

실행 압축 해제 방법은 크게 세 가지로 분류 할 수 있다. 첫 번째는 디버거 및 분석도구를 이용한 직접 분석 방법을 이용한 실행 압축 해제 방법이다. 이 방법은 초창기에 많이 쓰였던 방법이고 현재도 쓰이고 있는 방법이다. 사람이 직접 분석 도구를 이용하여 실행 압축 해제하기 때문에 가장 정확한 압축 해제가 가능하다. 하지만 실행 압축 프로그램의 수많은 명령어를 모두 분석해야 하므로 많은 시간이 걸린다는 단점이 있다. 두 번째 방법은 실행 압축 알고리즘의 특징에 기반을 둔 방법이다. 이 방법은 특정의 실행 압축 프로그램에 쓰인 실행 압축 알고리즘을 알게 되면 해당 알고리즘의 특징을 이용하여 실행 압축을 해제하게 된다. 하지만 새로운 알고리즘이나 어떤 압축 알고리즘이 쓰였는지 모르는 경우에는 해제가 불가능하다는 단점이 있다. 세 번째는 실행 압축 기법에 의존하지 않는 실행 압축 해제 방법이다. 이 방법은 앞에서 말한 두 가지 방법의 단점을 보완하는 방법으로 많은 연구가 이루어져 왔다.

PolyUnpack[5]은 실행 압축 파일이 실행될 때 결국 원래 실행 파일의 숨겨진 코드가 실행되기 위해 메모리에 로드되고 실행된다는 실행 압축 파일의 본질적인 성질을 이용하여 먼저 실행 압축된 실행 파일의 코드와 데이터 섹션을 역어셈블하여 기록하고 실행 압축된 코드를 연속적으로 일부분씩 실행한다. 그리고 그 부분에 대한 분석을 통하여 숨겨진 코드 섹션을 실행하는 위치를 확인한다. 하지만 이러한 방법은 모든 코드를 다 실행하고 역어셈블하므로 비효율적이다. OmniUnpack[6]은 PolyUnpack과 달리 모든 실행 코드를 분석하지 않고 프로그램을 실행하면서 메모리를 관찰하여 프로세스가 데이터를 쓴 메모리 영역으로 실행하는 순간의 영역을 분석한다. 따라서 PolyUnpack 에 비하여 분석시간을 단축시키는 장점이 있다. OmniUnpack은 'write-xor-execute' 정책으로 메모리 접근을 관찰한다. 하지만 이것은 Dual mapping[7]을 이용하여 쉽게 회피가 가능하다는 단점이 존재한다.

Renovo[15]는 가상머신을 이용하여 실행압축을 해제하는 기법으로 가상머신에 가상환경을 만들고 프로그램을 실행 시킨다. 프로그램이 메모리에 로드되면 메모리 맵을 만들고 메모리에 mov나 push등의 명령어 수행을 관찰하고 메모리에 쓰인 곳으로 Instruction 포인터가 점프하면 그곳이 오리지널 엔트리 포인트라고 결정한다. 가상머신을 이용하여 악성코드에 의한 감염 위험성이 없다는 장점이 있지만 가상머신을 이용하여 속도가 느리다는 단점과 정확한 오리지널 엔트리 포인트를 찾을 수 없다는 단점이 있다.[12]

Gunhyeon Jeong[8]은 Generic Unpacking using Entopy Analysis를 제안하였다. 이 방법은 엔트로피분석을 통하여 오리지널 엔트리 포인트를 찾아내어 실행 압축을 해제하고 더 나아가 실행 압축이 풀리는 과정에서의 엔트로피 수치 변화량을 이용하여 실행압축이 풀리는 알고리즘을 몇 개의 카테고리로 분류하였다. 이 방법의 단점은 엔트로피의 값에 의존하여 실행 압축을 해제하므로 높은 엔트로피 수치를 가지는 일반적인 파일을 실행 압축 파일로 잘 못 판단할 수 있다는 점이다. 또한 쓰레기 값 등을 넣어 엔트로피 값을 변화 시켰을 때에 오탐이 있다는 단점이 있다.

Cesare[9]는 Classification of Malware Using Structured control Flow 라는 논문에서 역변환 기술을 사용하여 제어 흐름 그래프(Control Flow Graph) 시그니처를 구성하는 알고리즘을 제안하였다. 실행압축 된 악성코드를 분석하기 위해서 어플리케이션 레벨의 빠른 에뮬레이션을 사용하였다. 제안하는 방법에서는 악성코드 분류를 위해서 동적 분석과 정적분석의 방법을 모두 사용하였다. 실행파일이 압축되어 있는지 판단하기 위해 먼저 엔트로피 분석방법을 사용하였고, 만약에 압축되어 있다면 동적 분석을 통해 압축해제가 끝난 시점을 탐지하여 숨겨진 코드를 파악한다. 그 다음으로 정적분석을 사용하여 특징을 찾아내고, 구조화 어플리케이션을 이용해 제어흐름 그래프를 위한 시그니처를 만든다. 이러한 시그니처는 알려진 악성코드의 데이터베이스를 찾기 위한 정확한 사전기반 검색에 사용된다. 하지만 이러한 방법의 문제점은 앞서 나온 엔트로피의 문제점과 시그니처 방식의 문제점을 동시에 가지고 있다는 단점을 가지고 있다.

III. 제안 기법

엔트로피 값 변화를 이용한 실행 압축 해제 방법

일반적으로 엔트로피는 통계적인 무질서도를 나타낸다. 일반 파일이 실행 압축되면 파일 내의 무질서도가 증가하여 엔트로피 값이 증가하게 된다.[4] 실행 압축 파일이 실행되면 실행 압축이 해제 되어야 하고 무질서도는 떨어지게 된다. 그리고 실행 압축 해제 후 실제 프로그램이 실행되면 엔트로피 값의 변화는 거의 없게 된다.

Lyda[4]는 일반 파일과 실행 압축 파일의 엔트로피 값을 분석하면 특정의 엔트로피 범위 값이 존재함을 증명하고 실행 압축 파일을 탐지하는 방법을 제시하였다. 기존의 엔트로피를 이용한 실행 압축 해제 기법[8][12]은 실행 압축 파일 실행 과정의 Jump나 Call 계열의 명령어가 수행될 때 엔트로피 값 수치를 측정한다. 이때 측정한 엔트로피 수치가 Lyda[4]가 주장한 일반 파일의 엔트로피 범위 값에 들어오면 실행 압축이 풀렸다고 판단하는 방법으로 해당 Jump, Call 명령어를 OEP 지점으로 찾고 실행 압축을 해제하였다. 하지만 이러한 방법은 엔트로피 값이 높은 일반 파일과 쓰레기 값 등을 실행 압축 파일에 넣어 엔트로피 값을 조작한 실행 압축 파일에는 OEP를 찾을 수 없다는 단점이 존재한다. 따라서 엔트로피 값에 초점을 맞추지 않고 실행 압축 해제 후 실제 프로그램이 실행되면 엔트로피 값의 변화는 거의 없게 된다는 특성을 이용하여 엔트로피 값의 변화량에 초점을 맞춘 실행 압축 해제 방법을 제안한다. 실행 압축 파일의 엔트로피 값에 변화량으로 OEP 지점을 찾게 되므로 기존의 엔트로피를 이용한 실행 압축 해제 기법[8][12]에서의 단점을 보완할 수 있다.

다음은 제안하는 방법에서 쓰이는 엔트로피 공식이다. 는 가 발생할 확률이고 는 이산 확률 변수 X의 자기 정보량(Self-information)을 의미한다. log 밑인 b의 값으로는 일반적으로 2, 오일러 수 e, 10을 자주 사용한다.

$$H(X) = \sum_{i=1}^{n} p(x_i)\, I(x_i) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \quad (1)$$

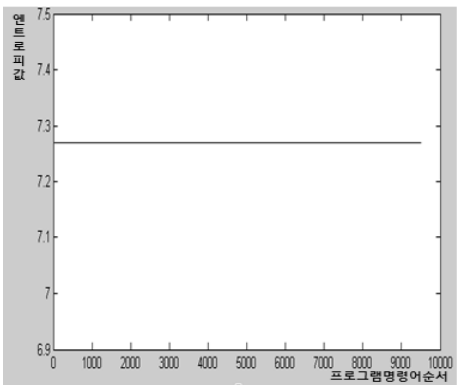
실행 압축 프로그램의 엔트로피 값의 변화를 보기 위해서는 프로그램 명령어 단위의 엔트로피 계산이 필요하다. 실행 압축 프로그램의 명령어 단위의 엔트로피 값을 x 로 표현하고 최종 종료 시점의 엔트로피 값을 x_0, n 개의 명령어 개수를 가진 프로그램의 시작 시점의 엔트로피 값을 x_n 이라 하면 실행 압축 프로그램의 명령어 단위 엔트로피 값은 $x_0 \sim x_n$ 으로 표현한

다. 다음과 같이 시작 시점을 으로 설정하는 이유는 프로그램 종료시점부터 엔트로피 변화량을 분석하는 것이 프로그램 시작 시점부터 변화량을 분석하는 것 보다 명확하게 OEP 지점을 찾을 수 있기 때문이다. 악성코드나 일반 프로그램의 실행 과정에서 엔트로피 값 변화를 살펴보면 〔그림 2〕에서 보는 것 같이 초기에는 변화가 심하다가 실제 프로그램이 구간에서는 〔그림 1〕의 박스부분과 같이 엔트로피 값의 변화가 거의 없는 것을 확인할 수 있다. 따라서 본 논문에서는 변화가 심한 시작 시점부터 분석을 시작하지 않고 변화가 없는 종료 시점부터 분석을 하여 변화가 없다가 처음으로 큰 변화가 일어나는 부분이 OEP 지점임을 알 수 있게 된다.

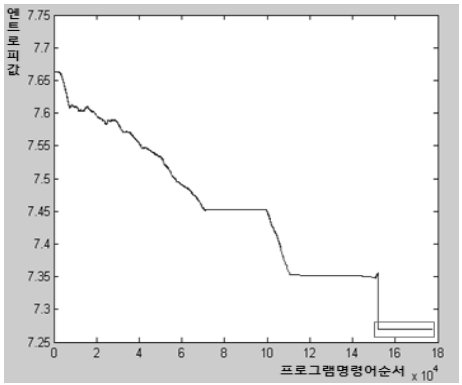
여기서 x축은 프로그램 명령어 순서를 나타내고 y축은 해당 명령어에서 프로그램의 엔트로피 값을 나타낸다.

각 명령어 구간의 엔트로피 변화를 보기 위하여 해당 구간 이전의 엔트로피 값 평균을 구한다. 각 명령어 구간 이전의 엔트로피 값을 모두 평균 내는 것은 비효율적이므로 정해진 크기의 슬라이드 윈도우에 i개의 엔트로피 값을 넣고 그 평균 값 y_n 을 구한다. 아래의 〔그림 3〕은 엔트로피 평균값을 구하기 위한 슬라이드 윈도우 개념도 이다. 즉 y_n 은 x_{n-i} 과 x_n 사이의 평균 엔트로피 값을 나타낸다. 그리고 엔트로피 변화량 Δ_n는 $x_n - y_{n-1}$ 의 절대 값이 되고 각 엔트로피 변화량 Δ_n 은 이전의 엔트로피 변화량의 평균 z_{n-1}과 비교하여 변화량이 차이를 계산한다. 엔트로피 변화량 평균은 아래 수식 2 와 같이 계산한다.

$$z_n = \sum_{j=1}^{n} \frac{\Delta_j}{n} \qquad (2)$$



〔그림 1〕 실행 압축 해제 이후의 악성코드 엔트로피 변화 그래프

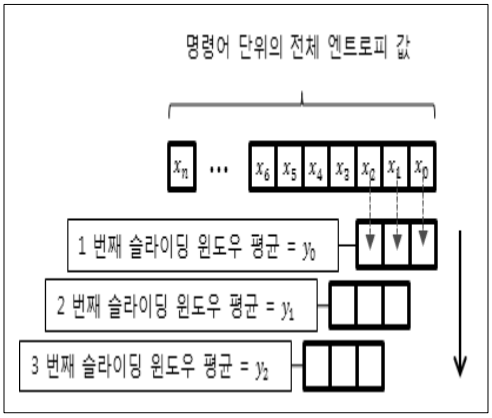


〔그림 2〕 악성코드의 전체 엔트로피 변화 그래프

엔트로피 값의 변화량을 분석하다보면 실행 압축이 해제된 이후 구간에서 한 순간 변화량이 커졌다가 다시 원래대로 돌아오는 경우를 확인할 수 있다. 이런 경우 엔트로피 값이 확실히 변하는 구간이 아니기 때문에 이러한 구간을 OEP 예상 구간이라고 판정하면 오탐이 일어나게 된다. 따라서 이러한 경우에는 해당 지점 x_i을 기준으로 앞의 엔트로피 값 10개의 평균과 기존의 엔트로피 평균값을 비교하였을 때 변화가 있는가를 확인하면 오탐을 피할 수 있다. 즉 엔트로피 값의 변화가 기존의 변화보다 큰 변화를 보일 때, 아래의 수식 3,4 의 계산을 통하여 해당 변화 구간이 OEP 예상 구간이 아닌지 판별하게 된다.

$$\sum_{j=0}^{10} \frac{x_{i+j}}{10} \rangle \frac{x_i + y_{i-1}}{2} \rangle y_{i-1}, \ (i \le n-10) \qquad (3)$$

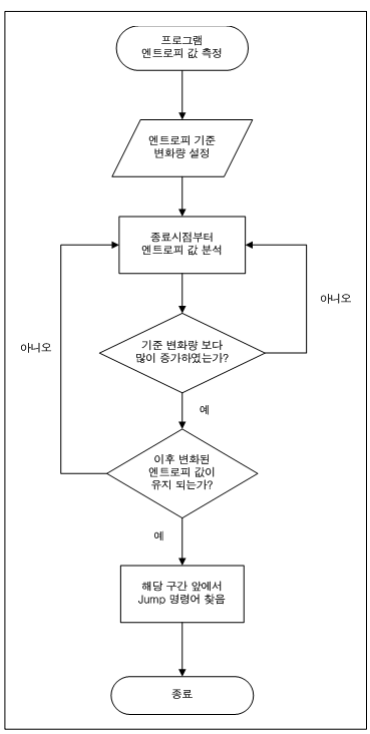
$$\sum_{j=0}^{10} \frac{x_{i+j}}{10} \langle \frac{x_i + y_{i-1}}{2} \langle y_{i-1}, \ (i \le n-10) \qquad (4)$$



〔그림 3〕 엔트로피 평균을 구하기 위한 슬라이드 윈도우

OEP 예상 구간이 나오면 해당 시점에서 앞으로 Jump 명령어를 찾게 되고 Jump 명령어가 있는 지점이 OEP 지점이 되는 것이다. OEP 지점인 Jump 명령어를 찾게 되는 것은 기존의 엔트로피를 이용한 실행 압축 해제 방법[8][12]과 같다 하지만 기존의 방법은 엔트로피 값의 변화가 심한 프로그램 시작 시점부터 엔트로피 값을 분석하여 오탐의 가능성이 있으나 본 논문에서는 실행 압축이 끝나 엔트로피 값이 안정된 시점인 프로그램의 종료 시점부터 엔트로피 값을 분석하여 이러한 오탐의 가능성을 줄였다. 또한 기존의 방법은 Jump 명령어 수행 시 엔트로피 수치 값에 의존하여 실행 압축 해제 여부를 판단하기 때문에 오탐 및 OEP를 찾을 수 없는 문제가 있었다. 하지만 본 논문에서는 실제 프로그램이 수행되면 엔트로피 값의 변화가 없다는 성질을 이용하여 프로그램 종료시점부터 엔트로피 값 변화량을 분석하게 되고 급작스러운 변화가 생기는 시점에서 OEP 지점을 찾아내어 실행 압축을 해제하여 이전의 문제점을 보완하는 방법을 제안한다.

다음 [그림 4]는 지금까지 설명한 제안하는 기법의 흐름도이다.



[그림 4] 엔트로피 변화 분석을 이용한 실행 압축 해제 기법의 흐름도

3.2 로우 패스 필터 (Low Pass Filter)를 통한 제안 기법의 성능 향상

본 논문에서는 엔트로피 값의 변화 분석을 보다 용이하게 보기 위하여 로우 패스 필터를 이용한 필터링을 하였다. 로우 패스 필터는 낮은 주파수만을 통과시키고 높은 주파수의 성분을 버리는 것으로 이러한 로우 패스 필터를 사용하면 엔트로피 값 변화 그래프에서의 노이즈 값을 없애고 변화되는 시점을 보다 효율적 확인 할 수 있게 된다. 일부 프로그램은 실행되는 과정 중에 데이터의 변화가 많은 경우가 존재한다. 이 경우에는 실제 프로그램이 실행되는 구간에서 엔트로피 값의 변화가 일반적인 실행 압축 프로그램보다 많이 일어나는 현상이 발생한다. 따라서 로우 패스 필터를 이용하여 노이즈 값을 없애고 엔트로피 값의 변화를 분석하면 로우패스 필터를 사용하지 않았을 때 보다 더 좋은 결과를 낼 수가 있다. 다음은 본 논문에서 사용된 로우 패스 필터의 공식이다.

$$L(n) = L(n-1) + 2\pi f_L(x(n) - L(n-1)) \quad (5)$$

각 인자의 의미는 다음과 같다. L(n)는 필터링된 결과 값이며 x는 원본 데이터의 값, n은 데이터의 번호, f_L 은 필터 주파수 이다. 여기서 말하는 필터 주파수란 로우 패스 필터를 통과 시키는 제한 주파수, 즉 필터의 차단 주파수를 의미하며 단위는 Hz 이다. 필터 주파수를 낮게 설정할수록 진폭의 진동 성분이 크게 감소한다. 따라서 필터 주파수는 낮을수록 많은 노이즈를 제거 할 수 있으나 그에 따라 동시에 감응도도 낮아지므로 적절한 주파수를 조정해야 한다. 따라서 이번 실험에서 사용되는 실행 압축 파일들을 대상으로 주파수를 변화시켜가면서 로우패스필터를 적용하여 보았다. [표 1]은 alg.exe 파일을 UPX로 실행 압축한 파일을 대상으로 실험한 결과이다. 필터를 적용하지 않았을때는 OEP 구간에서 12.9배의 변화를 보였다. 0.5Hz의 로우패스필터를 적용하게 되면 값이 OEP 구간을 못찾는 오류가 발생한다. 그리고 점점 낮은 주파수를 사용 할수록 필터를 적용하지 않았을 때 보다 노이즈가 제거되어 변화량을 확실히 볼 수 있는 것을 볼 수 있다. 하지만 0.05Hz 보다 낮은 주파수를 사용하게 되면 너무 많은 노이즈를 제거하게 되어 OEP 지점을 찾지 못하는 오류가 발생하는 것을 확인할 수 있다. 따라서 실험 결과 0.05Hz 의 주파수가 가장 적절한 것으로 나왔다. 그러므로 이번 실험에

〔표 1〕 로우패스필터 적정 주파수 실험 결과 (alg_upx)

주파수 / 명령어 단위	필터 적용 x	0.5	0.2	0.1	0.05	0.04
OEP에서의 엔트로피 변화량	12.9배	오류 발생	13.9배	12.8배	14.8배	오류 발생

서는 해당 주파수를 통해 실험을 진행하였다.

IV. 실험 내용

이 장에서는 제안하는 실행 압축 해제 기법에 대한 실험 내용을 설명한다. 그리고 실험 내용을 바탕으로 결과를 분석하게 된다.

4.1 엔트로피 변화 분석 전처리

본 논문에서 제안하는 엔트로피 값의 변화를 보기 위해서 실험 환경은 Window XP SP3 에서 수행하였다. 실험 데이터로 일반 PE 파일을 Windows의 'System32' 디렉토리 안에서 랜덤으로 10개를 추출하였고 인터넷〔13〕에서 수집한 악성코드 10개를 사용하였다. 실험 도구로는 OllyDbg〔14〕를 사용하였다. 실험에 사용된 실행 압축 기법으로는 UPX, Aspack, FSG, Mpress, PeCompack, Packman, RLpack, Nspack 으로 총 8가지를 바탕으로 실험을 진행하였다. 20개의 파일에 8가지 실행 압축 기법을 적용하였고 총 160개의 실행 압축 파일에 대한 실험을 진행하였다. 엔트로피 값의 변화를 보기 위해서 프로그램이 실행되는 명령어 단위마다 파일 전체를 OllyDbg〔14〕 의 OllyDump 플러그인을 통하여 덤프를 수행하였다. 덤프 과정은 OllyScript를 이용하고 덤프할 때 call 같은 함수 부분은 하나의 명령어로 간주하고 덤프를 수행하였다. 그 이유는 call 같은 함수 부분에서 그 내부의 명령어까지 모두 보게 되면 명령어의 양이 너무 많아져 비효율적이다.

일반 파일을 대상으로 하나는 함수를 명령어 하나로 간주하여 함수 내부의 명령어는 살펴보지 않고 엔트로피 값의 변화를 보았고 다른 하나는 함수 내부의 명령어까지 하나의 명령어로 간주하여 엔트로피 값의 변화를 보았다. 그 결과 두 경우의 엔트로피 값의 변화는 큰 차이가 없었기 때문에 이번 실험에서는 call 같은 함수 부분은 하나의 명령어로 간주하고 덤프를 수행하였다. 덤프 파일의 엔트로피 값은 앞에서 말한 엔트로피 계산식을 이용하여 엔트로피 값을 계산하였다.

4.2 엔트로피 변화 분석

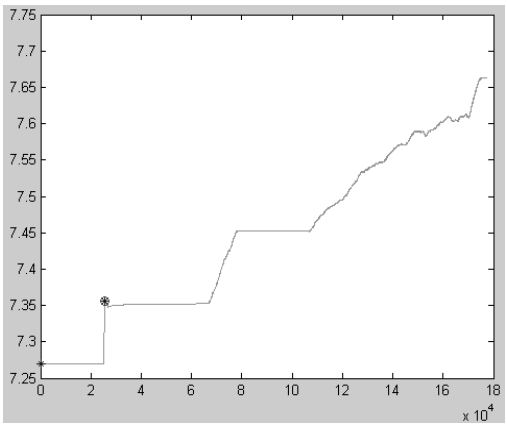
실험은 각 실행 압축 파일에 대하여 슬라이드 개수를 1, 2, 5, 10, 20, 30 으로 설정하고 엔트로피 값의 변화가 기존의 변화량보다 5, 10배 이상 되었을 때 OEP 예상 구간으로 정하고 확인하는 공식을 통하여 OEP 구간을 찾는 실험을 진행하였다. 아래의 〔표 2〕는 제안하는 방법의 실험 결과의 실행 압축 해제 성공률을 정리한 표이다.

실험 결과 슬라이드의 개수는 10개 그리고 변화량은 5배로 설정하였을 때 가장 OEP를 찾을 확률이 높았으며 실험 도중 오류가 발생한 2개를 제외한 총 158개의 실행 압축 프로그램 중 147개를 실행 압축 해제시켜 성공률이 93.04%가 되었다. 실험 결과 본 논문에서 제안하는 방법으로 실행 압축을 해제할 때 기존의 방법〔8〕보다 성공률이 21% 향상되었음을 확인할 수 있다. 실패한 11개의 실행 압축 프로그램은 OllyDbg를 이용한 덤프 과정에서 OEP 이후의 명령어가 주로 함수를 호출하여 실행되는 구조였다. 따라서 함수를 명령어 하나로 간주하다보니 명령어의 개수가 적게 나오게 되어 정확한 OEP 예상 지점을 찾지 못하였다. 아래의 〔그림 5,6,7,8〕은 UPX로 압축된

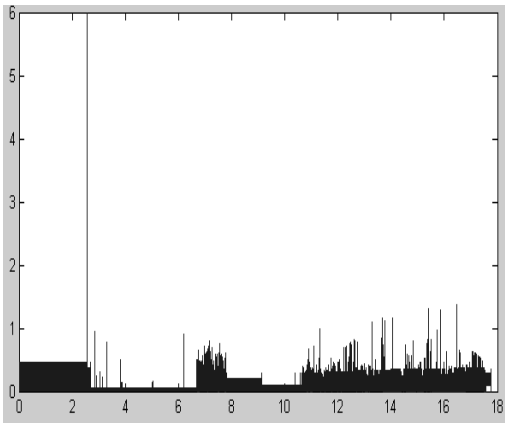
〔표 2〕 제안하는 방법의 실험 결과

슬라이드개수 / 변화량	1	2	5	10	20	30
5 배	73.44%	76.56%	81.25%	93.04%	85.94%	76.56%
10 배	71.88%	75.00%	84.38%	89.06%	85.94%	76.56%

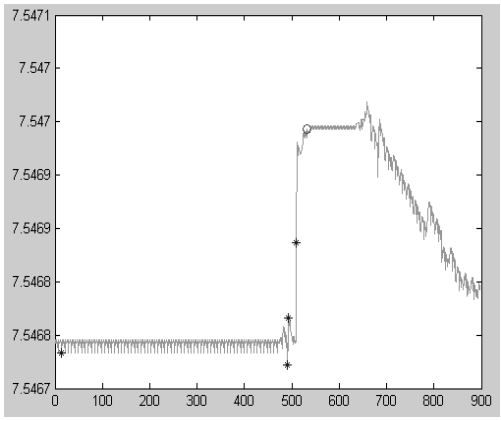
winmine.exe 파일과 악성코드인 erci.exe.의 실험 결과 그래프이다. 〔그림 7〕 같은 경우 OEP 부분의 엔트로피 변화 그래프를 확대하여 표현한 그래프이다. 또한 〔그림 5,6,7,8〕은 제안된 방법과 같이 종료 시점부터 시작 시점으로 변환시킨 그래프이다. 따라서 실제 엔트로피 변화 그래프의 시작 시점은 오른쪽부터가 된다. 그림 5에서 보면 * 표시 있는 부분이 기존의 변화량 보다 5배 이상 증가된 변화량을 보인 부분이고 마지막으로 나오는 표시 지점이 이러한 변화가 유지되는 지점으로 OEP 예상 구간임을 알 수 있다. 해당 지점에서 앞으로 Jump 명령어를 찾아내면 Jump 명령어가 있는 부분이 OEP 가 된다. O 부분이 실제 OEP 이며 해당 부분에서 Jump가 일어남을 알 수 있다.



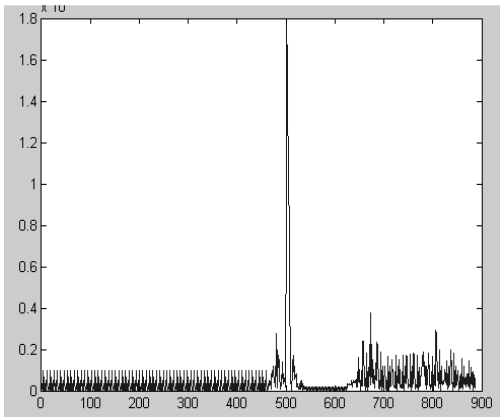
〔그림 5〕 UPX로 압축한 erci.exe 의 전체 엔트로피 값 그래프



〔그림 6〕 UPX로 압축한 erci.exe(악성코드) 의 엔트로피 변화량 그래프



〔그림 7〕 UPX로 압축한 winmine.exe 의 엔트로피 값 그래프

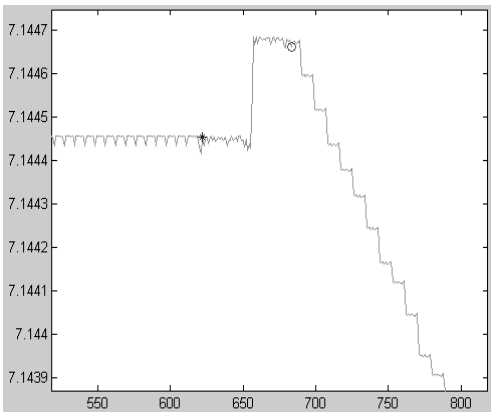


〔그림 8〕 UPX로 압축한 winmine.exe 의 엔트로피 변화량 그래프

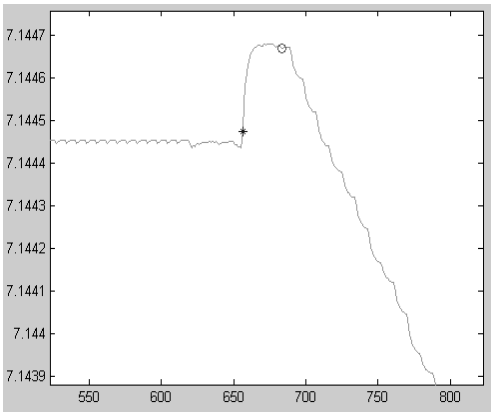
그리고 일부 실행 압축 파일에 대해서 다른 설정 값을 적용하였을 때 더 좋은 결과를 나타내기도 하였다. 그리고 로우 패스 필터를 사용하여 노이즈를 제거하고 실험을 하면 아래의 〔그림 9,10〕의 결과처럼 보다 좋은 결과를 나타내는 것을 확인 할 수 있었다. 〔그림 9, 10〕 도 마찬가지로 종료시점부터 시작시점으로 제안하는 방법과 같이 변환한 그림이다.

V. 결 론

기존의 실행 압축 기법의 특징을 이용한 실행 압축 해제 방법은 변형되거나 새로운 실행 압축 기법에는 무력하다. 또한 기존의 엔트로피를 이용한 실행 압축 해제 기법은 OEP 지점을 찾기 위하여 엔트로피 값의



〔그림 9〕 로우 패스 필터 적용 전의 FSG로 압축된 calc.exe 엔트로피 그래프



〔그림 10〕 로우 패스 필터 적용 후의 FSG로 압축된 calc.exe 엔트로피 그래프

범위에 의존하여 엔트로피 값을 변형하는 방법에는 오탐이 일어나는 문제점이 있었다.

본 논문에서는 기존의 방법의 문제점을 보완하는 엔트로피 값 변화 분석을 통한 실행 압축 해제 기법을 제안하였다. 실행 압축 기법들은 본래의 프로그램을 실행시키기 위하여 반드시 실행 압축을 해제하여야 하는데 실행 압축이 해제된 시점에서의 엔트로피 값의 변화는 크게 일어나지 않음을 실험을 통하여 확인하였다. 따라서 이러한 성질을 이용한 프로그램 종료 시점부터의 엔트로피 값 변화 분석으로 엔트로피 값의 변화가 없다가 크게 생기는 부분을 찾게 되면 정확한 OEP 지점을 찾을 수 있음을 확인하였다. 또한 제안하는 기법이 기존 엔트로피를 이용한 실행 압축 해제 방법[8][12]의 문제점을 보완하고 실행 압축 해제 성공률을 21% 향상 시킨 기법임을 증명하였다.

앞으로는 Anti-Debugging 기법이나 가상화 기법이 적용된 실행 압축 기법들을 대상으로 Anti-Debugging 기법을 우회하면서 실행 압축 해제하는 연구가 진행되어야 할 것이다. 그리고 실행 압축 기법과 압축되는 프로그램의 종류에 따라 엔트로피 값의 변화가 차이가 있었기 때문에 더 많은 실험을 통하여 이것의 상관관계 및 가장 최적의 압축 해제 설정 값을 찾아내는 연구를 향후 진행할 예정이다.

참고문헌

〔1〕 AV-Test. http://www.av-test.org

〔2〕 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, Oct. 2008.

〔3〕 Roberto Perdisci, Andrea Lanzi and Wenke Lee, "Classification of packed executables for accurate computer virus detection", Pattern Recognition Letters" vol. 29, no. 14, pp. 1941-1946, Oct. 2008.

〔4〕 Robert Lyda and James Hamrock, "Using entropy analysis to find encrypted and packed malware", Security & Privacy IEEE, vol. 5, no. 2, pp. 40-45, Mar. 2007

〔5〕 Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds and Wenke Lee, "PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, Computer Security Applications Conference 2006. ACSAC '06. 22nd Annual, pp. 289-300, Dec. 2006

〔6〕 Martignoni, L. Christodorecu. M, and Jha, S, "OmniUnpack: Fast, Generic and Safe Unpacking of Malware," Computer Security Applications Conference 2007. ACSAC 2007. Twenty-Third Annual, pp. 431-441, Dec. 2007.

〔7〕 Skap. Using dual-mapping to evade auto mated unpacked. http://uninformed.org/?v=10&a=1.

〔8〕 Guhyeon Jeong, Euijin Choo, Joosuk Lee, Munkhbayar Bat-Erdene and Heejo Lee, "Generic Unpacking using Entropy Analysis", 2010 5th International Conference on Malicious and Unwanted Software, pp 98-105 . Oct. 2010.

〔9〕 Silvio Cesare and Yang Xiang, "Classification of Malware Using Structured control Flow", Proceeding AusPDC '10 Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing, vol. 107, pp 61-70, Jan 2010.

〔10〕 Thomas M. Cover and Joy A. Thomas, Elements of Information Theory : Second Edition, Wiley Interscience, pp. 1-16, Jul. 2006.

〔11〕 한승원, 이상진, "악성코드 포렌식을 위한 패킹 파일 탐지에 관한 연구", 한국정보처리학회논문지, 16-C(5), pp 555-562, 2009년 10월.

〔12〕 정구현, 추의진, 이주석, 이희조, "엔트로피를 이용한 실행 압축 해제 기법 연구", 한국정보기술학회논문지, 7(1), pp.232-238, 2009년 2월.

〔13〕 malware, http://www.offensivecomputing.net/?q=taxonomy/term/1

〔14〕 OllyDbg, http://www.ollydbg.de/odbg110.zip

〔15〕 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), pp 46-53. Nov. 2007.

〈著者紹介〉

이 영 훈 (Young-hoon Lee) 학생회원
2009년 8월: 고려대학교 컴퓨터정보학과 졸업
2009년 9월~현재: 고려대학교 정보경영공학전문대학원 정보경영공학과 석사과정
〈관심분야〉 네트워크 보안, 시스템 보안, 역공학, 악성코드 분석

정 만 현 (Man-Hyun Chung) 학생회원
2006년 2월: 동국대학교 컴퓨터학과 학사
2009년 2월: 고려대학교 정보경영공학전문대학원 정보경영공학과 석사
2010년 9월~현재: 고려대학교 정보경영공학전문대학원 정보보호학과 박사과정
〈관심분야〉 패턴인식, 시스템 보안, 네트워크 보안

정 현 철 (HyunCheol Jeong) 종신회원
1989년 2월: 서울시립대학교 전산통계학과 졸업
1999년 8월: 광운대학교 전자계산학과 석사
2006년 9월~2008년 8월: 고려대학교 정보보호대학원 박사과정 수료
1996년 7월~현재: 한국인터넷진흥원 인터넷침해대응센터 연구개발팀장
〈관심분야〉 침해사고대응, 융합서비스보안, 네트워크보안, 컴퓨터 포렌식

손 태 식 (Tae-shik Shon) 정회원
2005년 8월: 고려대학교 정보보호학과 박사 졸업
2005년 3월~2007년 2월: 삼성전자 통신 연구소 선임연구원
2007년 3월~2011년 2월: 삼성전자 Digital Media & Communication 연구소 책임 연구원
2011년 3월~현재: 아주대학교 정보컴퓨터공학부 교수
〈관심분야〉 오픈 플랫폼 보안, 스마트그리드, 차량 네트워크 보안, 암호 알고리즘

문 종 섭 (Jong-sub Moon) 정회원
1981년~1985년: 금성 통신 연구소 연구원
1991년: Illinois Institute of technology 전산학 박사
1993년~현재: 고려대학교 전자 및 정보공학부 교수
〈관심분야〉 생체인식, 침입탐지, 네트워크 보안, 운영체제, 시스템 보안