Journal of Digital Contents Society Vol. 15 No. 3 Jun. 2014(pp. 373-380)
http://dx.doi.org/10.9728/dcs.2014.15.3.373

명령어 주소 엔트로피 값을 이용한 실행 압축 해제 방법 연구

이원래*, 김형중**

요 약

악성코드는 분석가가 탐지 및 분석을 어렵게 하기 위하여 실행 압축 기술을 이용하고 있다. 최근에는 실행 압축 기술이 적용된 코드에 대응하기 위하여 실행 압축 기술에 대한 연구가 진행되고 있다. 실행 압축 기술은 압축된 실행코드를 해제해야 동작이 가능하여 실행 코드를 압축하는 과정에서 반복되는 코드를 이용한다. 따라서 일반 코드와 비교하여 반복되는 코드가 많아서 동일한 코드가 유사한 주소 값을 가지고 수행되는 특성이 있다. 코드영역을 일정한 영역으로 구분하면 실행 압축이 해제되는 코드는 원래의 코드와 비교하여 낮은 엔트로피값을 갖는다. 이를 이용하면 실행 압축 알고리즘을 알지 못한 상태에서 실행 압축 여부를 판단할 수 있으며 실행 압축코드를 해제할 수 있다. 본 논문에서는 압축이 해제되는 코드에서 명령어의 주소 값이 작은 엔트로피값을 갖는다는 것을 이용하여 실행압축을 해제하는 방법을 제안한다.

키워드 : 실행압축, 악성코드, 언패킹, 엔트로피

A Study on Generic Unpacking using Entropy of Opcode Address

Won Lae Lee*, Hyoung Joong Kim**

Abstract

Malicious codes uses generic unpacking technique to make it hard for analyzers to detect their programs. Recently their has been several researches about generic packet to prevent or detect these techniques. And they try to focus on the codes that repeats while generic packing is doing compression because generic packing technique executes after it is decompressed. And they try to focus on the codes that repeats while generic packing is doing compression because generic packing technique executes after it is decompressed. Therefore, this makes a interesting performance which shows a similar address value from the codes which are repeated several times what is different from the normal program codes. By dividing these codes into regularly separated areas we can find that the generic unpacking codes have a small entropy value compared to normal codes. Using this method, it is possible to identify any program if it is a generic unpacking code or not even though we do not know what kind of algorithm it uses. This paper suggests a way of disarming the generic codes by using the low value entropy value which comes out from the Opcode addresses when generic unpacking codes try to decompress.

Keywords : Entropy, Generic Unpacking, Malware, Packing

1. 서론

악성코드는 지속적으로 제작되어 배포되고 있다. 공격자가 자기를 과시하거나 금전적인 이익을 목적으로 좀비PC를 만들기 위하여 악성코드를 제작하여 배포한다. 또는 회사나 국가의 정보 시스템 인프라를 파괴하려는 목적의 악성코드도 만들어 배포하기도 한다.

공격자는 좀비 PC를 생성하고자 하는 목적으로 악성코드가 포함된 메일을 보내어 PC를 장악하거나 업데이트 서버를 장악하여 일시에 많은

※ 교신저자(Corresponding Author): Hyoung Joong Kim
접수일:2014년 04월 29일, 수정일:2014년 06월 18일
완료일:2014년 06월 18일
* 고려대학교 정보보호대학원
Tel: +82-2-2255-1683, Fax: +82-2-2255-1660
email: wallerle@naver.com
** 고려대학교 정보보호대학원
Tel: +82-2-3290-4895, Fax: +82-2-928-9109
email: khj-@korea.ac.kr

PC를 좀비PC로 만들기도 한다.

이러한 공격에 대비하기 위해 악성코드를 탐지하고 분석하기 위한 연구도 계속되었다. 악성코드 감염을 방지하고 감염된 PC에서 악성코드 제거를 목적으로 안티바이러스 프로그램이 배포되고 운영된다. 또한, 공격자도 안티바이러스 프로그램을 우회하기 위하여 보호기술을 적용하고 있다. 공격코드를 은닉하고 공격 코드를 작게 만들기 위하여 많은 악성코드에서 실행압축기술을 사용하고 있다[1].

분석가가 실행압축기술이 적용된 악성코드를 분석하기 위해서는 실행압축 해제가 전제되어야 한다. 동적 분석을 통하여 악성코드의 행위 일부를 분석할 수 있지만 악성코드 전체에 대한 분석이 가능하기 위하여 실행압축에 적용된 기술 해제는 필수요소다.

실행압축기술이 적용된 악성코드의 경우 컴퓨터의 메모리 주소공간에서 압축을 해제하는 코드가 반복적으로 실행된다. 재압축을 적용한 경우도 반복하는 코드가 실행된다[2]. 압축을 해제하는 코드는 동일한 코드가 반복적으로 실행되는 특성이 있으므로 실행압축 해제코드와 원래 프로그램의 코드를 비교하면 통계학적으로 다른 분포가 생긴다. 이러한 원리를 이용하여 악성코드의 실행압축기술 적용여부를 판별하고 해제하는 방법을 제안한다.

이 방법은 명령어 주소(Operation Code Address)를 이용한 엔트로피 분석방법이다. 명령어주소 엔트로피 분석방법은 프로그램이 동작하는 과정에서 기계어 명령어(Operation Code)가 실행되는 주소를 추적하여 엔트로피의 변화를 분석한다.

실행압축코드가 적용된 프로그램은 실행압축을 해제하는 코드와 원래의 악성코드로 분리되어 구성되어 있다. 실행압축을 해제하는 코드는 유사한 코드가 반복되어 수행되므로 일반코드가 수행되는 코드보다 작은 주소공간을 사용한다.

예를 들면 실행압축이 적용된 프로그램코드를 100으로 나누었을 경우 압축을 해제하는 코드는 1~2개이다. 또한 1~2개의 일부부의 코드가 반복적으로 실행되므로 상대적으로 작은 주소 영역을 사용하게 된다. 동일한 구간에서 작은 주소영역을 사용하면 동일한 구간에서 많은 주소영역을 사용하는 코드와 비교하여 통계적으로 차이가 발생한다. 이 방법을 이용하여 실행 압축이 해제된 프로그램의 시작점(Original Entry Point)을 찾는다.

명령어주소 엔트로피 분석방법은 컴퓨터의 운영체계와 상관없이 적용이 가능하다. 그러나 많은 악성코드가 윈도우즈 기반에서 작동하므로 윈도우즈 운영체제를 중심으로 설명한다.

본 논문에서는 윈도우즈 운영체계에서 실행압축된 프로그램이 해제되는 과정에서 사용하는 메모리 주소를 추적하여 엔트로피를 분석한다. 2장에서는 실행압축해제 관련 연구를 설명하였으며 3장에서는 제안한 명령어 주소 엔트로피 분석방법에 대하여 자세히 설명하고 4장에서는 실험을 통하여 이를 설명한다. 5장에서는 본 연구의 성과를 설명한다.

2. 실행압축해제 방법

실행압축을 해제하는 기술에서 가장 일반적인 기술은 디버거나 분석도구를 이용하여 수작업으로 분석하는 것이다. 디버깅의 기술을 가지고 있는 인력이 직접 분석을 실시하므로 정확한 분석이 가능한 장점이 있으나 시간이 많이 걸리는 단점이 있다.

실행압축 알고리즘을 알고 있는 경우 실행압축 알고리즘의 특징을 이용하여 압축해제가 가능하다. 이러한 방법은 압축 알고리즘을 모르는 경우 압축 해제가 불가능하다는 단점이 있다.

수작업에 의한 방법이나 실행압축 알고리즘의 특징을 이용한 압축 해제의 단점을 보완하기 위하여 많은 연구가 진행되었다.

Lyda[3]는 실행파일을 대상으로 실행압축이 적용된 파일이 일반파일과 비교하여 높은 엔트로피값을 가지는 것을 이용하여 실행압축여부를 판단하였다. 엔트로피는 일반 TEXT 파일이 가장 낮고 실행파일, 실행압축 파일, 암호화 파일 순으로 엔트로피가 높다.

Kang[4]은 가상머신을 이용하여 실행압축을 해제하는 방법을 제안한다. 가상머신에 실행되는 환경을 만들고 메모리 영역을 관찰하여 프로세스가 메모리 영역에 쓰고 해당 영역에서 프로세스가 실행되는 경우 원래의 프로그램 시작점이라고 결정한다. Kang이 사용한 Renovo 가상머

신은 악성코드 감염 위험성이 낮아지며, 악성코드가 동작하는 전체를 분석할 수 있다는 장점이 있다. 반면 가상머신을 별도로 구축을 해야 하고, 속도가 느려진다는 단점이 있다. 또한 영역만을 감시하여 압축 해제된 프로그램의 Original Entry Point를 판단하므로 쉽게 우회가 가능하며 탐지오류 가능이 존재한다.

Jeong[5]은 실행압축 해제과정을 진행하고 있을 때, 메모리 상태의 엔트로피 값의 변화를 관찰하여 Original Entry Point를 찾아내었다. 이러한 방법을 사용하기 위하여 전체적인 엔트로피 변화의 추이를 살피는 것이 필요하다. 이를 위하여 대상 프로세스에 대하여 명령어가 수행되는 매 순간의 엔트로피를 측정해야 한다. 그러나 메모리의 상태를 프로세스가 진행되는 명령어를 실행하는 매 순간 측정하는 것은 시스템의 부하를 주어 거의 불가능하다. 이러한 단점을 보완하기 위하여 측정하는 메모리 영역을 일부분으로 한정하고 전체 명령어 수행이 아닌 JMP나 CALL명령어 발생시점을 기준으로 측정하는 방법을 이용하였다. 이러한 방법도 전체 프로세스 분석이 불가하다는 단점은 여전히 존재한다.

Quist[6]은 실행파일에서 명령어 주소를 추출하여 시각화함으로써 악성코드를 분석하는 방법을 제안하였다. 프로그램이 수행되는 전체의 주소공간을 시각화하여 분석이 수월하다는 장점이 존재한다. 그러나 모든 명령어 주소를 이용하여 시각화를 하기 때문에 속도가 느려지며, 대용량의 PC를 필요로 하기 때문에 가상환경을 구축하여 악성코드를 분석이 분석하는 것이 불가능하다는 단점이 있다.

Lee[7]는 Jeong[5]이 엔트로피 값에 의존하여 실행압축을 해제하므로 높은 엔트로피값을 가지는 파일에서 발생하는 탐지의 오류를 줄이기 위하여 실행압축된 파일을 해제하면서 엔트로피 값 변화량을 관찰하여 실행압축 해제가 끝나는 시점을 판단하는 기법을 제안하였다. Lee의 방법도 Jeong의 한계인 가상 메모리 영역을 한정하고 JUMP나 CALL명령어 발생시점을 기준으로 측정하는 방법을 이용하므로 전체 프로세스에 대한 분석이 불가하다는 단점이 여전히 존재한다.

3. 제안기법

본 논문에서 제안하는 기법은 프로그램이 실행될 때 수행되는 각 명령어 주소를 추출하여 엔트로피값을 분석하는 방법이다. 엔트로피는 Shannon[8] 이 제안한 정보 엔트로피 개념을 이용하였다.

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

이 개념은 엔트로피가 높은 데이터일수록 모든 비트들이 고르게 존재함을 의미하며, 반대로 엔트로피가 낮은 데이터일수록 모든 비트가 존재하지 않고 일부의 비트들만이 존재한다는 것을 나타낸다.

파일이 압축되면 파일내의 무질서도가 증가하여 엔트로피가 증가한다. 또한 압축된 코드는 실행되기 위하여 압축을 해제해야 하는데 압축을 해제하게 되면 무질서도가 감소하게 된다.

파일에 실행압축 존재여부 및 패킹이 풀리는 시점은 엔트로피의 변화를 관찰하여 엔트로피의 변화량이 가장 많이 발생하는 지점을 Original Entry Point로 판단한다.

정확한 Address를 판단하기 위하여 Information Gain을 사용한다. Information Gain은 Kullback-Leibler divergence[9]라고도 하며 상대적인 엔트로피의 차이를 나타낸다. 실행압축을 해제하는 코드영역과 실행압축 이전의 코드영역을 분리하면 정확하게 분리되는 주소에서 가장 낮은 Information Gain값을 갖는 원리이다.

일반적인 의미에서 Information Gain은 이전 상태에서 정보의 변화 때문에 변경되는 되는 Information Entropy의 변경 값이다. [10]

$$IG(T,a) = H(T) - H(T|a) \quad (2)$$

T를 테스트 샘플로 나타나면, Information Gain의 공식을 (3)과 (4)로 나타낼 수 있다.

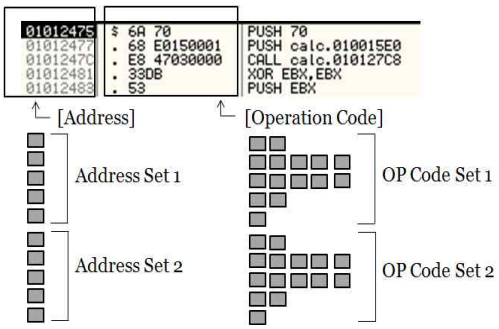
$$Let\ A = \{|x \in T|x_a = v|\} \quad (3)$$

$$IG(T,a) = H(T) - \sum_{v \in vals(a)} \frac{|A|}{|T|} \cdot H(A) \quad (4)$$

Lyda[3]와 같이 실행파일이 가지고 있는 비트의 분포의 엔트로피를 분석하여 실행압축 여부를 판단하거나 Jeong[5], Lee[7]와 같이 프로세스가 실행하고 있는 매 순간의 메모리를 비트의 엔트로피를 분석하여 Original Entry Point를 찾으려는 연구는 있었다. 그러나 본 논문에서와 같이 프로세스의 명령어 주소의 엔트로피를 분석하여 Original Entry Point를 찾으려는 연구는 없었다.

프로그램이 실행되면서 실행 코드는 가상 메모리에 저장된다. 기계 명령어는 주소를 가지고 있고 프로세스가 실행되는 것은 해당 주소에 저장된 기계 명령어가 실행되는 것이다.

(그림 1) 명령어코드 주소와 명령어코드



(Figure 1) Address of the Operation Code and the Operation Code

제안하는 방법은 명령어 코드를 이용하지 않고 명령어 코드가 실행되는 주소 자체를 이용하여 엔트로피를 계산하는 방법이다. (그림 1)에서 Address Set으로 설명된 명령어코드 주소를 이용하는 방법이다. 이 방법은 특정 영역을 지정하여 압축이 풀리는 것을 예측하여 엔트로피를 측정하는 기존의 방법을 개선할 수 있다. 또한, 특정 시점만이 아닌 프로세스가 실행되는 전체 구간에서 엔트로피의 변화를 통한 프로그램의 특정을 파악할 수 있다. 이렇게 되면 프로그램이 실행되는 중간에 패킹 로직이 수행되는지 여부도 판별이 가능하다.

한편, 특정 시점의 메모리 값의 엔트로피가 아닌 전체 프로그램이 수행되는 매 순간의 명령어 주소를 가지고 분석을 실시하므로 임의의 공간을 지정하여 실행 압축을 해제하는 우회방법도 추적이 가능하며, 적은 메모리 공간으로 효율적으로 프로그램 전체를 추적할 수 있다.

4. 실험내용

4.1 실험준비

실험은 Windows XP SP3 환경에서 실시하였다

1차 실험에서는 직접 작성하여 컴파일한 프로그램을 이용하여, 정상 프로그램과 실행 압축이 적용된 프로그램을 대상으로 Opcode의 Address를 이용하여 엔트로피를 시각화하여 패킹여부 판별이 가능하지 확인하였다.

2차 실험에서는 Windows 디렉터리에 포함된 10개의 프로그램을 선택하여 패킹을 실시한 후 OEP를 찾는 것이 가능한지 확인한다. Opcode의 Address를 이용하여 엔트로피 변화량을 비교하여 OEP를 찾을 수 있는 실험을 실시한다.

실험을 위하여 명령어 주소 추출은 Pintool [11]을 이용한다. Pintool은 프로그램 분석 프로그램으로 Android, Linux, OSX 와 Windows 운영체계에서 32bit와 64bit 구조를 위한 실행프로그램을 지원한다. Pintool은 실행파일이 실행하는 동안 C 혹은 C++ 작성된 임의의 코드를 실행하여 명령어가 실행되는 주소, 명령어가 접근하는 메모리 주소, 명령어 실행 시의 레지스터 상태 등을 모니터링 하거나 변경하는 기능을 제공한다.

4.2 시각화를 이용한 패킹여부 인지 실험

첫 번째 실험은 Microsoft의 Visual C++ 10[12]을 이용하여 컴파일한 프로그램을 이용한 실험이다. 특정 문자열을 화면에 출력하는 기능만을 가진 프로그램이다. 프로그램이 가지는 특성을 최소화하였을 경우 실행단계에서 변경되는 주소영역을 관찰한다.

<소스 1> HelloWorld.c 프로그램

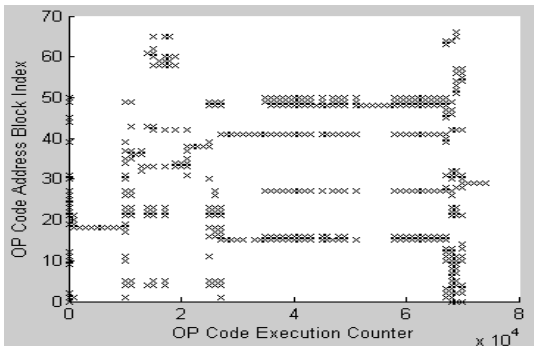
```
int main(){
    printf("Hello, World.\n"); }
```

<source 1> HelloWorld.c program

실험을 위한 시각화에서 사용한 알고리즘은 가로축은 전체 명령어 주소를 일정 블록으로 나누고, 세로축은 명령어를 그룹화하기 위하여 명령어 주소를 0xFFFFFF00와 AND 연산하여 합을 구한다. 사용되지 않는 주소 값을 출력하지 않으며, 시스템에서 사용하는 주소의 영역도 출력하지 않는다. 사용자 영역에서 실행된 주소 값을 추적하여 주소의 크기가 낮은 순서에서 높은 순서로 정렬하여 분포를 확인한다.

HelloWorld.c 프로그램을 컴파일하여 만든 실행프로그램에서 명령어 코드(Operation Code)의 주소를 추출하여 블록을 만든 후 시각화하면 (그림 2)와 같다. 세로축은 75,615 명령어코드 주소와 0xFFFFFF00값을 AND 연산하여 75 블록이 된다. 가로축은 75,615개의 코드를 1,000로 나누어 68개의 블록을 만든다. 명령어 주소가 편중되지 않고 고르게 분포한다.

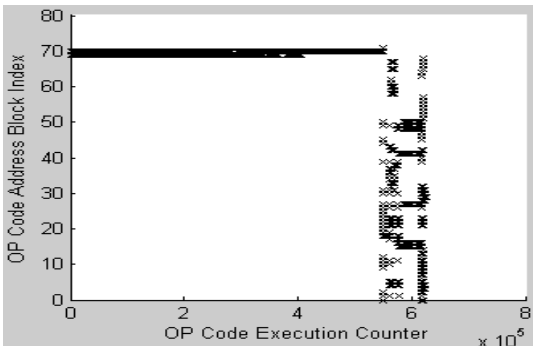
(그림 2) HelloWorld 명령어코드 시각화



(Figure 2) Visualization of the Operation Code Address with HelloWorld Program
Horizontal axis: 75,615/1,000 (75 Blocks),
Vertical axis: AND Operation 0xFFFFFF00
with Addresses of 75,615 Operation Codes (68 Blocks)

(그림 3)의 경우는 (그림 2)에서 사용한 프로그램으로 UPX로 패킹한다. 동일한 방식으로 블록을 만들어서 시각화하면, 일정 시점까지 동일한 주소가 반복되어 실행됨을 알 수 있다. 세로축은 629,937개의 명령어를 0xFFFFFF00와 AND 연산을 하여 72개의 블록이 된다. 가로축은 629,937개의 명령어를 10,000으로 나누어 62개 블록인 된다. (그림 3)과 같이 압축을 해제하는 코드가 특정영역에서 실행된다.

(그림 3) UPX 실행압축 적용된 HelloWorld 명령어코드 시각화



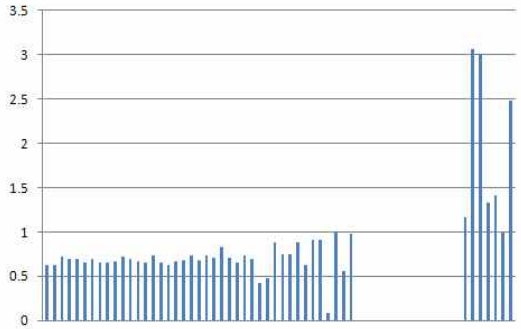
(Figure 3) Visualization of the Operation Code Address with HelloWorld Program After UPX packing,
Horizontal axis: 629,937/10,000 (62 Blocks),
Vertical axis: AND Operation 0xFFFFFF00
with Addresses of 629,937 Operation Codes (72 Blocks)

일정한 크기의 주소가 속한 블록을 만들었을 경우 해당 블록이 반복해서 실행된다. 블록안에는 일정한 개수의 주소 값이 저장되어 있으므로 엔트로피를 계산할 수 있다. (그림 4)의 경우 (그림 3)에서 구한 동일한 데이터를 이용하여 엔트로피를 계산한다. 전체 62개의 블록 중에서 55개 블록은 1이하의 낮은 엔트로피를 가지며 56번째부터 1이상의 엔트로피 값이 계산된다. 1이하의 낮은 엔트로피가 반복된다는 것은 이 블록이 압축을 해제하는 코드가 반복적으로 실행된다는 것을 나타낸다.

사용된 주소영역 전체에 대한 엔트로피를 분석하였을 경우 동일한 주소에서 반복되는 부분은 엔트로피가 낮으며, 원래의 명령어가 실행되는 영역에서 엔트로피가 높아진다. 추출한 엔트로피를 그래프로 나타내면 (그림 4)에서 보는 보아 같이 압축이 해제되는 시점에서 엔트로피가 증가되는 것을 볼 수 있다. 이 시점을 찾아내면 실행 압축 알고리즘에 상관없이 실행 압축이 적용된 프로그램에서 압축 적용 이전 영역을 분리

할 수 있다.

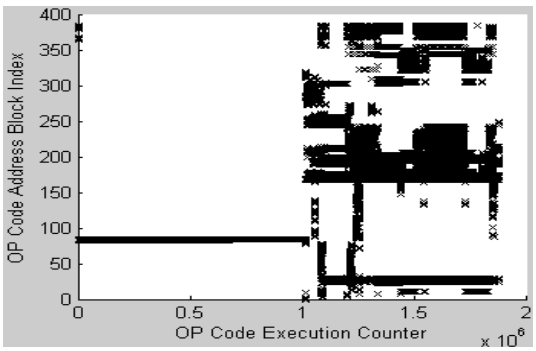
(그림 4) UPX 실행압축 적용된 HellowWorld 명령어코드 엔트로피 그래프



(Figure 4) Entropy Graph of Operation Code Address Block with HelloWorld Program After UPX packing
Horizontal axis: 629,937/10,000 (62 Blocks), Vertical axis: Entropy of AND Operation 0xFFFFFF00 with Addresses of 629,937 Operation Codes (72 Blocks)

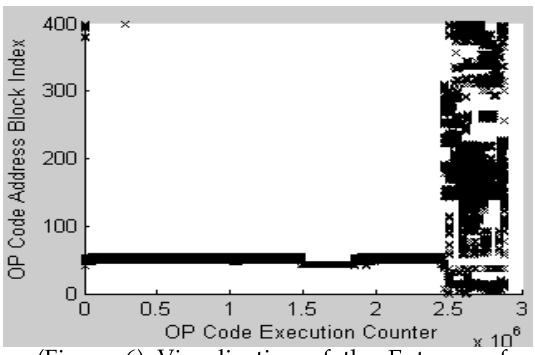
윈도우 프로그램에서 계산기와 메모장을 선택하여 실해 압축 적용 후 명령어 주소의 분포를 확인하였다. (그림 5)와 (그림 6)에서 확인되는 바와 같이 Original Entry Point를 기준으로 주소 값의 분포가 확대되는 것을 알 수 있었다. 일정한 영역에서 선으로 그려지는 부분이 압축을 해제하는 코드가 실행되는 영역이다.

(그림 5) UPX 실행압축 적용된 계산기 프로그램의 명령어코드 시각화



(Figure 5) Visualization of the Operation Code Address Block with calc.exe Program After UPX packing
Horizontal axis: 1,882,844/10,000 (188 Blocks), Vertical axis: AND Operation 0xFFFFFF00 with Addresses of 1,882,844 Operation Code (385 Blocks)

(그림 6) ASPACK 실행압축 적용된 메모장 프로그램의 명령어코드 시각화



(Figure 6) Visualization of the Entropy of Operation Code Address Block with notepad.exe Program After ASPACK packing
Horizontal axis: 2,886,844/10,000(288 Blocks), Vertical axis: AND Operation 0xFFFFFF00 with Addresses of 2,886,844 Operation Code (399 Blocks)

실행압축이 종료되는 지점에는 일정한 갯수(실험에서는 10,000개)의 명령어가 포함되어 있다. 이 중에서 정확한 Address 는 Information Gain의 값을 이용하여 구할 수 있다. 실행압축을 해제하는 코드가 분리되었을 경우 Information Gain의 값이 가장 작으며 해당 주소가 Original Entry Point가 된다.

4.3 엔트로피 변화량을 이용한 실행압축 알고리즘 자동해제 여부 판별 실험

실험에 사용된 압축방법은 Lee[7]이 실험에 사용한 8개 알고리즘중에서 fsg, upx, aspack, mpress 4가지를 선택하였으며 대상 프로그램은 윈도우 디렉터리에 존재하는 실행파일 중에서 10개를 선택하였다.

4개의 압축 알고리즘과 10개의 프로그램을 이용하여 Original Entry Point 추적에 대한 실험을 총 40회 실시하였다. 테스트 결과 40개 중에

서 37개의 실험에서 실행 압축을 해제하는 로직과 원래 실행파일을 정확하게 분류를 했다.

<표 1> 일반 실행압축 해제 실험 결과 (C: 성공, F: 실패)

	fsg	upx	aspack	mpress
calc.exe	C	C	C	C
dxdiag.exe	C	C	C	F
freecell.exe	C	C	C	C
mshearts.exe	C	C	C	C
msiexec.exe	C	C	C	C
mspaint.exe	C	C	C	F
notepad.exe	C	C	C	C
spider.exe	C	C	C	F
telnet.exe	C	C	C	C
winmine.exe	C	C	C	C

<Table 1> Test Result of the Generic Unpaking Result (C: Correct, F: Fail)

fsg, upx, aspack 알고리즘은 10개 파일 모두 실행 압축 로직과 원래의 실행 코드 분류가 가능하였으며 mpress로 실행 압축한 경우는 7개 파일은 압축 해제가 가능하였지만 3개는 실행압축이 적용된 부분에서 원래의 코드를 분류하지 못했다. 분석 결과, 분류가 실패한 3개는 압축 해제를 진행하면서 생성한 로그 파일이 500M 이상인 경우였다. 실행 프로그램이 다른 파일에 비교하여 상대적으로 큰 파일이며, 실행 압축을 해제 시에 실행 명령어 주소의 추출량이 커서 명령어 주소에 대한 분석 로직이 원활하게 실행되지 못했기 때문이다.

5. 결론

기존의 실행 압축 기법의 특징을 이용한 실행 압축 해제 방법은 실행 압축이 풀리는 지점이 일정하지 않는 경우는 탐지가 불가능하다. 이는 기존의 방법이 파일 또는 메모리의 저장되는 명령어 자체에 대한 엔트로피를 분석하기 때문에 프로세스가 실행되는 전체 구간에 대한 분석이 불가능하고 일정 영역을 지정하여 분석하기 때문이다.

본 논문에서는 제안한 방법은 명령어가 실행되는 주소의 엔트로피를 분석하여 실행 압축 및 탐지에 대한 연구를 진행하였다. 실행 압축이 풀리는 로직 영역의 명령어가 일반 코드보다 작은 영역에서 실행되므로 작은 엔트로피를 가지는 원리를 이용하였다. 프로세스가 실행되는 동안 명령어 주소를 추출하여 엔트로피의 변화를 분석하면 실행 압축을 푸는 영역과 풀린 후의 영역을 정확하게 구분할 수 있다는 것을 실험을 통하여 증명하였다.

명령어의 주소를 추출하는 과정에서 추가적인 처리 없이 명령어 주소를 그대로 추출하므로 용량이 큰 실행 파일의 경우 분석이 불가능 하였다. 앞으로는 큰 용량의 파일도 분석이 가능하도록 필요한 명령어 주소만 추출하는 로직을 추가할 필요가 있으며 보다 많은 파일에 대하여 연구를 계속할 필요가 있다.

References

[1] AV-Test. http://www.av-test.org

[2] Ho Young Whang, Hyoung Joong Kim, “Reversible Watermarking for Audio Using Recompression Method”, Journal of Digital Contents Society, vol. 14, no. 2, pp. 199-206, Jun. 2013

[3] 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

[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), pp 46-53. Nov. 2007

[5] GuHyeon Jeong, Euijin Choo, Joosuk Lee and Heejo Lee, “Generic Unpacking Using Entropy Analysis”, JKIIT, Vol.7, No.2, pp.232-238, Feb 2009

[6] Daniel A. Quist and Lorie M. Liebrock. Reversing compiled executables for malware analysis via visualization, Information Visualization. 10(2), April 2011. (doi:10.1057/ivs.2010.11)

[7] YH Lee, MH Jeong, HC Jeong, TS Son, JS Moon "A Study on Generic Unpacking using Entropy Variation Analysis", JKIIT, Vol.22, No.2, pp. 179-188, April 2012

[8] C.E. Shannon and W. Weaver, The Mathematical Theoryof Communication, Univ. of Illinois Press, 1963

[9] Kullback, S. "Letter to the Editor: The Kullback-Leibler distance". The American Statistician 41 (4): 340?341. JSTOR 2684769. 1987

[10] information gain http://en.wikipedia.org/wiki/Information_gain_in_decision_trees

[11] pintool http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[12] Visual C++ 10, http://www.microsoft.com

이 원 래

2014년 : 고려대학교
정보보호대학원 석사과정

1996년~현 재: 삼성SDS
관심분야 : 정보보호, 악성코드 분석 등

김 형 중

1978년:서울대학교 공과대학(B.S)
1986년:서울대학교 공과대학(M.S.)
1989년:서울대학교 공과대학(Ph.D)

1989년~2006년: 강원대학교
1992년~1993년: University of Southern California
2007년~현재: 고려대학교 정보보호대학원 교수
관심분야 : 멀티미디어 컴퓨팅, 멀티미디어 보안, 멀티미디어 포렌직, 머신러닝, 분산컴퓨팅, 소셜네트워킹, 정보보호응용 등