오탐지 정확도를 개선한 실행압축 판단 기법 연구

노한영, 김광조*

* 한국정보통신대학교 암호와 정보보안 연구실

A Research on Packer Classification to Reduce False Negative Rate

Hanyoung Noh, Kwangjo Kim*

* Cryptology & Information Security Lab., Information and Communications University

요 약

최근의 악성코드는 자기 자신을 보호하기 위해 실행압축 기법을 사용하고 있다. 악성코드 탐지 프로그램에서는 실행압축된 파일에 대해 실행압축 여부를 판단하고 실행압축이 되어 있으면 실행압축을 해제한다. Bintropy[6]는 파일의 실행압축 여부를 판단하는 기법 중 하나로 매우 빠르고 정확도가 높은 기법이다. 하지만, Bintropy에서의 실험은 샘플의 숫자가 적기 때문에 그 정확성을 판단하기가 쉽지 않다. 본 논문에서는 Bintropy를 더 많은 샘플에 대해 실험하고 그 한계를 파악하여 개선하였다. 본 논문에서 제안한 기법은 오탐지(False negative)를 5% 정도 낮춘 보다 정확한 실행압축 파일 판단 기법이다.

I. 서론

악성코드 탐지 프로그램은 일반적으로 파일의 특정 바이트를 서명으로 사용하여 악성코드를 탐지한다. 하지만, 최근의 악성코드는 실행압축 기법을 이용하여 자기 자신의 서명을 변형하여 악성코드 탐지 프로그램의 탐지를 피한다. 실행압축 기법은 처음에는 프로그램의 용량을 줄이고, 역공학으로부터 파일을 보호하기 위해 많이 사용이 되었지만, 최근에는 악성코드에 많이 사용이 되고 있다.

그림 1 . 악성 코드 탐지 프로그램 동작

실행압축 기법에 대응하기 위해 악성 코드 탐지 프로그램의 동작 단계가 그림 1과 같이 변화하게 되었다. 먼저 입력된 파일이 실행압축 되어있는지를 판별하는 단계, 그리고 실행압축을 해제 하는 단계이다. 최근의 악성코드의 92%는 실행압축이 되어있으며, 실행압축의 방식은 매 달 10~15개 이상이 새로 만들어지고 있다[1]. 또한 소스가 공개된 실행압축 방식은 악성코드 제작자가 쉽게 수정이 가능하기 때문에, 새로운 실행압축에 대응하기 위한 방법들이 많이 연구가 되고 있다.

실행압축 해제에 관해서는 많은 연구[2, 3, 4]가 이루어졌으나 아직까지는 해제 속도와 정확도 등에서는 개선의 여지가 있다. 하지만, 파일이 실행압축 되어 있는지를 판별하는 연구는 많이 이루어져 있지 않다[5, 6]. 본 논문에서는 파일의 실행압축 여부를 판별하는 기법 중에서 Bintropy를 설명하고 그 한계를 분석한다. 그리고 그 한계를 극복하기 위한 개선한 기법을 제안한다.

본 논문의 아래와 같이 구성이 된다. Ⅱ장에서는 Bintropy에 대한 설명과 관련 연구를 설명

하고, Ⅲ장에서는 Bintropy의 한계를 찾기 위한 실험과 결과를 설명한다. 그리고, Ⅳ장에서는 Bintropy를 개선한 우리의 제안 기법에 대한 설명과 실험 및 분석을 하고. 마지막으로 Ⅴ장에서는 본 논문을 요약하고 마무리한다.

II. 관련 연구

일반적으로 윈도우 실행파일의 크기가 크기 때문에 디스크 공간과 다운로드 효율을 위해 실행압축 기법을 사용하여 파일의 용량을 줄였다. 하지만, 최근에는 프로그램을 보호하기 위해 다양한 역공학방지 및 암호화 기법이 실행압축 기법에 추가가 되었다. 악성코드 제작자는 이 실행압축 기법을 악성코드에 사용하게 되었고, 이렇게 실행압축 기법이 적용된 악성코드는 바이트 구조가 바뀌게 되고 파일의 서명이 바뀌게 된다. 따라서 서명 기반의 악성코드 탐지 프로그램에서 이를 탐지하기가 어렵다.

실행압축이 되어있는지 안되어 있는지를 판단하는 문제는 중요하다. 실행압축이 되어있지 않은 파일을 실행압축이 되어있다고 판단할 경우(과탐지, False positive)에는 실행압축 해제 단계가 필요하게 되는데, 실행압축 해제는 일반적으로 사용자의 메모리, CPU 등의 자원을 많이 사용하고, 많은 시간을 필요로 한다. 하지만, 실행압축이 되어 있는 파일을 실행압축이 되어 있지 않다고 판단할 경우(오탐지, False negative)에는 악성코드 탐지 프로그램이 파일의 서명을 탐지하지 못하기 때문에 악성코드가 컴퓨터에 남아있을 수 있으므로 더 큰 문제가 될 수 있다.

실행압축이 되어있는 여부를 판단하기 위한 기법은 크게 서명기반과 엔트로피 기반의 판단법이 있다. 먼저, 서명기반의 실행압축 판단법은 실행압축 기법의 종류에 따라 시작 바이트 패턴이 같다는 점을 이용하고 있으며, 가장 유명한 서명기반 프로그램은 PEiD[5]가 있는데, 현재 약 750여개의 서명을 가지고 있다. 이 기법은 파일의 일부 바이트만을 검사하여 실행압축 적용 여부를 판단하기 때문에 매우 빠르고 정확한 기법이지만, 새로운 실행압축 기법은 탐지가 불가능하다는 단점이 있다.

반면에, 엔트로피 기반의 판단법은 파일의 정보 특성을 이용하여 실행압축 여부를 판단을 하기 때문에 새로운 실행압축 기법들까지 모두 진단이 가능하다. Bintropy[7]는 엔트로피 판단 기법 중 하나로 엔트로피를 계산하는 식은 다음과 같다.

$$H(x) = -\Sigma_{i=0}^{255} p(i)\log_2 p(i) \quad - (1)$$

식(1)에서 $p(i)$는 각 바이트 값 0x00에서 0xFF 까지의 분포 확률을 말한다. 일반적인 실행압축 기법들이 압축기법과 암호화기법을 사용하는데, 이는 모두 파일의 엔트로피를 높인다. 따라서 식 (1)에서의 결과 값 $H(x)$가 임계값을 넘으면 실행압축 되었다고 판단하고, 임계값을 넘지 않으면 실행압축이 되어있지 않다고 판단한다. 하지만, 이 방식은 오탐지와 과탐지 가능성이 있다는 단점이 있다. 또한, Bintropy의 실험에서는 실행압축 되어있지 않은 테스트 샘플로 윈도우 운영체제의 Windows 폴더와 Windows\system32 폴더의 100개 실행파일만을 대상으로 하고 있기 때문에 실험의 신빙성이 부족하다고 생각된다. 본 논문에서는 Bintropy의 한계를 찾고, 이를 극복한 새로운 분류 방법을 제안한다.

III. Bintropy의 한계

Bintropy에서 오탐지가 일어날 수 있다는 것을 확인하기 위해 Bintropy를 구현하였으며, 실행파일 전체가 아닌 실행 파일의 섹션만을 대상으로 엔트로피를 계산하였다. 또한, 테스트 샘플은 윈도우 운영체제의 기본 파일 뿐만 아니라 다양한 파일에 대해 테스트를 하였다. 우리의 실험 결과 몇 가지 오탐지 사례를 찾았다.

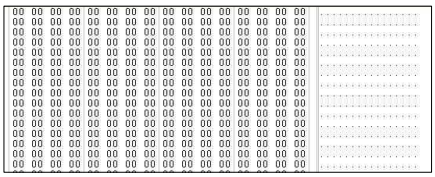

그림 2. Debug 모드 컴파일된 파일의 0x00 바이트

첫 번째로 그림 2와 같이 Debug 모드로 컴파일된 파일은 디버그를 위해 각 섹션을 매우 크게 잡고 있으며 데이터를 쓰고 남는 섹션은 모두 0x00(Null Byte)으로 채워지게 된다. 이는 Release 모드로 컴파일된 파일에 비해 매우 많다. 따라서, Debug 모드로 컴파일된 파일은 0x00 바이트가 매우 많아 엔트로피가 낮다. Debug 모드로 컴파일된 파일을 실행압축할 경우에도 Release 모드로 컴파일된 파일을 실행압축한 파일보다 엔트로피가 더 낮다.

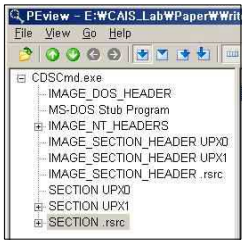

그림 3. 실행압축되지 않은 .rsrc 섹션

두 번째로 그림 3과 같이 실행압축 기법에 따라 리소스 섹션은 실행압축을 하지 않는다. 리소스 섹션의 경우에는 0x00 바이트의 수가 매우 많은 경우가 보통이며, 이 경우에도 엔트로피가 낮게 나타난다.

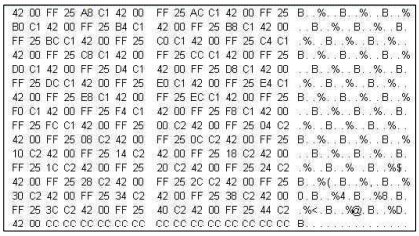

그림 4. 간단한 암호화가 된 파일

마지막으로 그림 4와 같이 XOR 연산과 같은 기초 수준의 암호화가 실행압축 기법에 사용이 되면, 0x00 바이트가 특정한 패턴 바이트로 바뀌게 되고, 이런 경우에는 0x00 바이트가 아닌 다른 특정 바이트의 수가 매우 많아지게 된다. 이 경우에도 엔트로피가 낮아진다. 이처럼 몇 가지 경우에 있어서 엔트로피가 비정상적으로 낮게 나타나는 경우가 있으며, 이러한 파일들은 임계값 이하의 엔트로피가 나타나는 경우가 있고, 이 경우에는 오탐지가 일어날 수 있다. 본 논문에서는 오탐지를 줄일 수 있는 방법을 제안한다.

IV. 개선된 Bintropy

Bintropy는 엔트로피 기반의 탐지를 하고 있으나, 앞에서 살펴본 특정한 경우처럼 엔트로피가 비정상적으로 나오는 경우가 있다. 본 논문에서는 앞의 오탐지를 극복하기 위한 방법으로, 식 (2)와 같이 엔트로피를 계산할 때 파일에서 가장 많이 나타나는 바이트를 제외하고 엔트로피를 계산 한다.

$$H_m(x) = -\Sigma_{i=0}^{255} p(i)\log_2 p(i) \quad except\ Max(p(i)) \quad - (2)$$

실험을 위한 샘플 (1829개)은 윈도우와 윈도우 시스템 폴더, 그리고 그 외 각종 프로그램들의 실행 파일을 수집하였다. 또한, 실행압축된 파일 샘플(510개)는 PEiD로 분류한 실행압축된 파일 외에 UPX, ASPack, ASProtector, 그리고 PE2Compact 4개의 실행압축기법을 이용하여 일반 실행파일을 실행압축하여 사용하였다.

그림 5와 6은 각각 Bintropy와 제안한 기법의 실험 결과를 정규분포에 따라 곡선을 그렸으며, 이를 이용하여 임계값을 정하였다. 제안한 기법의 임계값이 더 높은 이유는 전체 파일의 바이트 총합에서 가장 많이 사용된 바이트가 차지하는 비율이 실행압축된 파일은 평균 8%, 실행압축되지 않은 파일은 평균 25%를 차지하고 있기 때문이다.

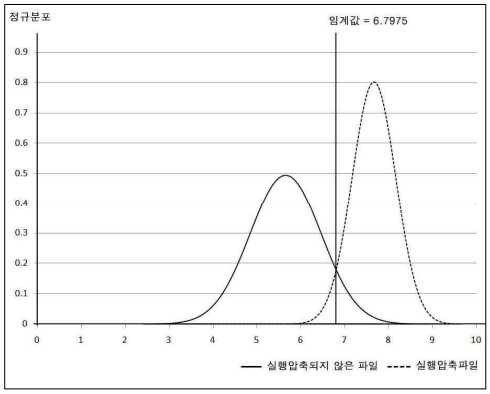

그림 5. Bintropy 실험 결과

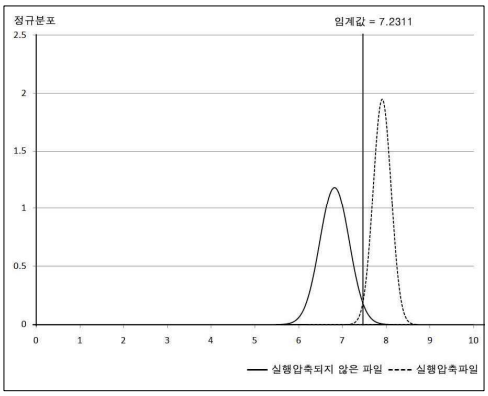

그림 6. 제안한 기법의 실험 결과

그림에서 확인 할 수 있듯이, 제안한 기법은 기존의 기법보다 겹치는 부분이 더 적게 나타나고 있어서, 정확도가 더 높다고할 수 있다. 표 1은 실험결과를 정리하고 있다.

표 1. 실험 결과 분석

	실행압축된 파일의 평균 엔트로피 / 표준편차		실행압축되지 않은 파일의 평균 엔트로피 / 표준편차	
기존의 기법	7.6585	0.4974	5.6478	0.8074
제안한 기법	7.9049	0.2046	6.8165	0.3358

실험 결과 제안한 기법을 사용했을 때 평균 엔트로피의 표준편차가 기존의 기법보다 적게 나타나고 있으며, 오탐지 확률이 더 낮다고 할 수 있다. 또한, 실험으로 얻은 임계값을 사용하여 각 파일을 분류했을 때의 탐지 정확도를 정리하면 표 2 와 같다.

표 2. 탐지 정확도 비교

	과탐지율	오탐지율	합
Bintropy	35/510 = 6.86%	105/1319 = 7.96%	140/1829 = 7.65%
제안기법	34/510 = 6.67%	34/1319 = 2.58%	68/1829 = 3.72%

이처럼 본 논문에서 제안한 기법은 정확도 측면에서 오탐지를 크게 줄일 수 있었다.

V. 결론

본 논문에서는 실행압축 여부를 판별하는 기법 중에서 엔트로피에 기반한 Bintropy의 오탐지 사례를 찾고, 이를 개선한 기법을 제안하였다. 제안한 기법은 실험 결과 기존의 기법에 비해 오탐지 확률이 더 낮았다. 본 논문에서 제안한 기법은 기존의 기법보다 정확한 기법이긴 하지만, 과탐지 측면에서 볼 때 아직도 개선의 여지가 필요할 것으로 보인다. 따라서, 앞으로 과탐지가 일어나는 사례를 찾아 이를 극복할 수 있는 방법에 대해 더 연구할 예정이다.

[참고문헌]

[1] Adrian Stepan, Improving Proactive Detection of Packed Malware, Proceedings of the Virus Bulletin Conference, 2006

[2] 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), Oct, 2007.

[3] Miroslav Vnuk, Pavol Navrat, Decompression of Run-Time Compressed PE-Files. In: Studies in Informatics and Control, Bucharest, National Institute for R+D in Informatics, Vol. 15, No. 2, 169-180, 2006.

[4] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. PolyUnpack: Automating the hidden-code extraction of unpack-executing malware. In ACSAC '06: Proceedings of the 22nd, pages 289-300, Washington, DC, USA, 2006.

[5] PEiD. http://www.secretashell.com/codomain/peid/

[6] Rosert Lyda, James Hamrock, Using Entropy Analysis to Find Encrypted and Packed Malware, IEEE SECURITY AND PRIVACY MAGAZINE, No. 2 pages 40-45, 2007.