논문 18-43-11-22 The Journal of Korean Institute of Communications and Information Sciences '18-11 Vol.43 No.11
https://doi.org/10.7840/kics.2018.43.11.1929

Anti-VM/Debugging 무력화 환경에서 악성코드 자동 언패킹 시스템 설계 및 구현

김 선 균◆, 김 하 진*, 최 미 정°

Design and Implementation of Malware Automatic Unpacking System in Anti-VM/Debugging Environment

Sun-Kyun Kim◆, Hajin Kim*, Mi-Jung Choi°

요 약

최근 인터넷 사용률이 증가함에 따라 악성코드 노출 위험도 증가하고 있다. 악성코드는 탐지되더라도 분석을 어렵게 하기 위해 분석 방해/지연 기술이 탑재되어 있다. 분석 방해/지연 기술 중 가장 대표적인 것은 패킹과 Anti-VM, Anti-Debugging이다. 본 논문에서는 악성코드 분석을 위해 Anti-VM/Debugging 무력화 환경에서 악성코드 자동 언패킹 시스템을 설계 및 구현한다. 제안하는 시스템은 초기 분석과 정적/동적 분석으로 이루어져있다. 초기 분석 단계에서는 PE 정보를 추출하여 Anti-VM/Debugging을 우회하고 패킹(Packing) 여부와 패커(Packer) 종류를 탐지한다. well-known 패커로 패킹된 경우 공개된 알고리즘 기반으로 구현된 언패킹툴을 사용하여 정적 분석을 수행하고, custom 패커로 패킹된 경우 엔트로피 기반 동적 분석을 수행한다. 본 연구는 악성코드 분석 방해 기법인 Anti-VM/Debugging을 무력화시킨 환경에서, 초기 분석에서 탐지한 악성코드의 패커 종류에 따라 악성코드의 언패킹을 자동으로 수행하는 시스템을 설계 및 구현한 시도로 악성코드 연구에 기여할 자료로 사용될 것이다.

Key Words : Packing, Unpacking, Anti-VM/Debugging, Packing Detection, PE analysis, Malware

ABSTRACT

Recently, as the Internet usage rate has increased, the risk of malignant code exposure has increased. Malware is equipped with analysis interruption/delay technology to make analysis difficult even if detected. The most common analysis interrupt/delay techniques are Packing, Anti-VM and Anti-Debugging. In this paper, we design and implement a malware automatic unpacking system in Anti-VM/Debugging disable environment for malware analysis. The proposed system consists of initial analysis and static/dynamic analysis. In the initial analysis phase, PE data is extracted to bypass Anti-VM/Debugging and to detect packing and type of packer. Static analysis is performed using unpacking tool when packed with well-known packer, and entropy-based dynamic analysis when packed with custom packer. Next, based on the system design, we implement the PE analysis, the packing detection, the packer type detection, and the static analysis using the well-known algorithm-based unpacking tool. This study is an attempt to design and implement a system that automatically performs unpacking of malicious

※ 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00158, 국가 차원의 침해사고 대응을 위한 사이버 위협 인텔리전스 분석(CTI) 및 정보 공유 기술 개발)

◆ First Author : Kangwon National University Department of Computer Science, kyun@kangwon.ac.kr, 학생회원
° Corresponding Author : (ORCID:0000-0002-9062-4604)Kangwon National University Department of Computer Science, mjchoi@kangwon.ac.kr, 종신회원
* Kangwon National University Department of Computer Science, hajinkim@kangwon.ac.kr
논문번호 : 201806-D-034-RE, Received May 28, 2018; Revised August 28, 2018; Accepted September 12, 2018

code according to the types of packers of malicious codes detected in the initial analysis, in the environment where Anti-VM / Debugging is disabled. This will be used as a resource to contribute to malware research.

Ⅰ. 서 론

최근 인터넷 사용률이 증가함에 따라 악성코드 노출 위험도 증가하고 있다. 2017년 AV-Test 악성코드 동향 보고서에 따르면 DDos(Distributed Denial of Service), 스팸 발송, APT(Advanced Persistent Threat) 공격 등에 사용된 악성코드는 연간 기준 약 60억 개에 달한다[1,2]. 뿐만 아니라 악성코드는 탐지되더라도 분석을 회피하기 위해 Anti-VM, Anti-Debugging, 패킹 등과 같은 분석 방해/지연 기술이 탑재되어 있다[3]. 악성코드의 등장과 함께 이에 대응하기 위한 연구 또한 활발하게 진행되고 있지만 악성코드의 수와 고도화된 분석 회피 기술이 증가하고 있기 때문에 이에 대한 신속한 대응이 필요하다[4-6].

분석 방해/지연 기술 중 가장 대표적인 것은 패킹과 Anti-VM, Anti-Debugging이다. 패킹은 '실행 압축'이라고도 하는데, 실행 파일을 압축하여 형태를 유지하면서 파일 크기를 줄이는 것이다. 패킹은 파일의 크기를 줄여 저장 공간을 확보하기 위해 개발되었으나 악성코드 제작자들이 악성코드의 은닉을 위해 사용한다. AV-Test사에 따르면 92% 이상의 악성코드에 실행 압축 기술이 적용되어있다고 한다[7]. 따라서 악성코드를 분석하기 위해 PE 파일의 패킹을 해제하는 언패킹을 수행해야 한다.

Anti-VM은 악성코드가 현재 시스템이 가지고 있는 정보들을 이용해 가상 머신임을 식별하여 실행 흐름을 제어하는 기법이다[8]. Anti-VM이 탑재된 악성코드는 프로세스 수행 중 가상 머신이 식별되면 악성 행위를 수행하지 않거나 프로세스를 종료한다. 악성코드는 레지스트리 기반으로 가상머신을 탐지하기 때문에 레지스트리 값을 변경하면 Anti-VM을 무력화가 가능하다. Anti-Debugging은 프로그램을 디버깅 하지 못하게 하는 일련의 작업을 의미한다. Anti-Debugging을 적용한 악성코드가 실행 중 디버깅을 당한다면 디버깅을 하지 못하도록 해당 디버거 프로그램을 종료시키거나 에러를 발생시켜 분석을 방해한다[8]. 따라서 Anti-Debugging도 Anti-VM과 마찬가지로 디버깅에 대한 정보를 변경하여 우회할 수 있다.

본 논문에서는 악성코드 분석을 위해 Anti-VM/Debugging 무력화 환경에서 악성코드 자동 언패킹 시스템을 설계 및 구현한다. 제안하는 시스템은 초기 분석과 정적/동적 분석으로 이루어져있다. 초기 분석 단계에서는 PE 정보를 추출하여 Anti-VM/Debugging을 우회하고 패킹 여부와 패커 종류를 탐지하고, 탐지된 패커 종류에 따라 정적/동적 분석으로 나누어 언패킹을 수행한다. well-known 패커로 패킹된 경우 언패킹툴을 사용하여 정적 분석을, custom 패커로 패킹된 경우 엔트로피 기반 동적 분석을 수행한다. 그리고 시스템 설계를 바탕으로 PE 정보 추출, 패킹 여부 탐지, 패커 종류 탐지, 언패킹툴을 사용한 정적 분석을 구현한다.

본 연구의 공헌은 다음과 같다. 첫째, 악성코드의 Anti-VM/Debugging을 무력화시킨 환경에서 자동 언패킹 시스템을 설계 및 구현한다. 둘째, 진입점 섹션의 엔트로피 값과 속성 값을 사용하여 Custom 패커 또는 시그니처가 아직 밝혀지지 않은 패커의 패킹 여부를 탐지한다. 본 연구는 악성코드 분석 방해 기법인 Anti-VM/Debugging을 무력화시킨 환경에서, 초기 분석 단계에서 탐지한 악성코드의 패커 종류에 따라 악성코드의 언패킹을 자동으로 수행하는 시스템을 설계 및 구현한 시도로 악성코드 연구에 기여할 자료로 사용될 것이다.

본 논문의 구성은 다음과 같다. 제2장에서 관련연구인 악성코드 분석 기법과 PE 포맷 분석, 패킹 탐지 및 언패킹 기법에 대해 설명한다. 제3장에서는 본 논문에서 제안하는 시스템 설계에 대해 설명한다. 제4장에서는 3장에서 설계한 내용을 바탕으로 시스템을 구현한다. 제5장에서 각 기능들이 제대로 수행되는지 검증한다. 마지막으로 제6장에서 결론을 맺는다.

Ⅱ. 관련연구

2.1 PE 포맷 분석

본 절에서는 초기 분석에 필요한 PE 파일을 분석한다. PE 파일 포맷은 Windows 환경에서 사용되는 실행 파일의 모든 포맷으로 윈도우 로더가 실행 가능한 코드를 관리하는데 필요한 정보를 캡슐화하여 저장한 데이터 구조체이다[9, 10]. 그림 1은 PE 파일의 구조를 나타낸다. 그림에서, 왼쪽은 PE 파일의 구조이며 오른쪽은 메모리에 로드 되었을 때의 상태를 나타낸다. PE 파일은 그림 1과 같이 DOS 헤더, DOS Stub, PE 헤더, 각 섹션 헤더, 각 섹션 테이블로 구성되어 있다.

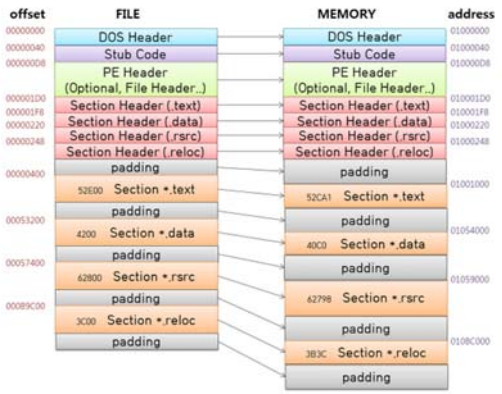


그림 1. PE 파일 구조.
Fig. 1. PE File Structure.

이 중 PE 헤더는 PE 파일의 실행 환경 정보를 가지고 있는 구조체로 DOS 헤더부터 Section 헤더까지를 말한다. PE 헤더는 PE 고유의 식별자로 시작하며, 언패킹에 필요한 정보인 실행될 수 있는 시스템, 섹션 수, 실행 속성, 진입점 섹션의 엔트로피 값, 진입점 섹션의 속성 값 등을 포함하고 있다. 따라서 패킹된 PE 파일의 PE 헤더를 분석하면 패킹 여부 탐지와 패커 종류 탐지가 가능하다.

2.2 악성코드 분석 기법

악성코드 분석 방법은 크게 초기 분석, 정적 분석, 동적 분석으로 나뉜다. 초기 분석이란 악성코드의 PE 파일의 정보를 추출하여 악성 코드 파일의 기본 속성을 분석하는 것으로, PE 파일의 압축 해제를 위한 사전 단계이다. 초기 분석에서 추출되는 기본 정보는 파일 이름, 크기, 오프셋, 엔트리 포인트 주소, 엔트리 포인트 섹션 이름, PE 파일 확장자, 진입점 섹션의 엔트로피 값, 진입점 섹션의 속성 값 등이다. 초기 분석에서 추출되는 정보들을 통해 악성코드의 패킹 탐지가 가능하기 때문에 초기 분석은 중요한 단계이다.

정적 분석이란 악성코드를 실행하지 않은 상태에서 내부 코드와 구조를 확인하거나 PE 헤더 및 문자열 정보를 이용하여 악성코드를 분석하는 것이다[11]. 정적 분석은 동적 분석과 달리 실행 조건에 제한 없이 분석을 진행할 수 있으며, 프로그램의 전체 실행에 대해 분석할 수 있다. 또한, 악성코드를 실행하지 않은 상태에서 분석을 진행하기 때문에 악성코드 감염의 위험과 악성코드 실행에 따른 자원 과부하가 없다는 장점이 있다.

동적 분석이란 가상머신 등의 환경에서 악성코드를 직접 실행시켜 코드의 흐름과 메모리 상태를 직접 모니터링 하는 방법을 말한다[12, 13]. 동적 분석은 주로 악성행위를 감시 및 추적하고 실제 동작방식을 분석하는 데 사용된다. 또한, 동적 분석은 행위 정보 기반으로 분석하기 때문에 신규 악성코드에 대한 탐지 가능성이 높고, 분석 기능의 자동화가 가능해 분석 시간 단축이 가능하나, 가상머신 등과 같이 악성코드를 실행시키기 위한 환경과 조건이 필요하고 악성코드가 분석 환경을 인지하여 회피가 가능하다는 단점이 있다[14,15]. 따라서 각 방법의 특징에 맞게 초기, 정적, 동적 분석을 목적에 따라 적절히 활용해야 한다.

악성코드 분석 방법에 따라 언패킹 기법도 나뉜다. 먼저 사람이 직접 분석도구를 이용하여 언패킹하는 직접 분석 방법이다. 이는 정확한 압축 해제가 가능하지만 시간이 오래 걸린다는 단점이 있다. 다음으로 실행 압축 알고리즘의 특징을 기반으로 언패킹하는 방법이다. 이는 well-known 패커와 같이 어떤 패킹 알고리즘이 쓰였는지 아는 경우에 언패킹이 가능하며 정적분석과 함께 쓰인다. 하지만 어떤 패킹 알고리즘이 쓰였는지 모르는 경우 사용이 불가하다는 단점이 있다. 마지막으로 패킹 알고리즘에 의존하지 않는 언패킹 기법이다. 패킹 알고리즘에 관계없이 언패킹을 진행하는 방법으로, 주로 동적분석과 함께 쓰인다. 어떤 패킹 알고리즘으로 패킹된 파일도 언패킹할 수 있다는 장점이 있지만 시간이 오래 걸린다는 단점이 있다.

언패킹 기법의 기존 연구는 다음과 같다. Jeong은 엔트로피를 분석하여 오리지널 엔트리 포인트를 찾아 언패킹을 수행하고, 이 과정에서 엔트로피 값 변화량을 측정해 언패킹 알고리즘을 몇 개의 클러스터로 분류하는 방법을 제안했다[16]. Cesare는 역변환 기술을 사용하여 제어 흐름 그래프 시그니처를 구성하는 알고리즘을 제안하였다[17]. 이 때 PE 파일이 패킹되어 있는지 판단하기 위해 먼저 엔트로피 분석방법을 사용하고, 패킹되어 있다면 동적 분석을 통해 패킹이 끝난 시점을 파악하여 숨겨진 코드를 파악한다. Lee는 언패킹될 때 엔트로피 값이 변화하는 성질을 이용하여 PE 파일의 언패킹 때의 엔트로피 값 변화량을 보고 언패킹이 끝나는 시점을 판단하여 언패킹하는 방법을 제안하였다[18].

2.3 패킹 탐지 기법

Lyda는 엔트로피 분석을 기반으로 암호화 또는 패킹된 악성코드를 탐지하는 기법을 제안하였다[19]. 즉, 전체 파일에 대한 엔트로피 값으로 패킹된 파일과 일반 파일을 구분한다. 하지만 패킹이 되어 있지 않은 파일의 엔트로피 값이 높게 계산되어 오탐이 발생하

거나 패킹된 파일의 엔트로피 값이 낮게 계산되어 미탐이 발생할 수 있다.

Choi는 PE 파일의 헤더를 분석하여 패킹 파일을 탐지하는 'PHAD'를 제안하였다[20]. 이 방법은패킹된 파일의 PE 헤더에 일반적이지 않은 특징 변수들이 포함된다는 특징을 이용하여 패킹 여부를 탐지한다. 8개의 변수로 구성되는 특징 벡터(CV)의 유클리디언 거리를 계산하여 패킹된 파일과 그렇지 않은 파일을 구분하는 방법을 사용한다.

Han은 진입점 섹션의 엔트로피 값과 패킹의 필수 요소를 이용하여 패킹 여부를 탐지하는 방법을 제안하였다[21]. 이 방법은 패킹된 파일의 진입점 섹션의 엔트로피 값과 섹션의 속성에 패킹 파일의 필수 요소인 'WRITE' 속성 여부에 따라 패킹 파일을 탐지한다.

Ⅲ. 시스템 설계

3.1 시스템 구성도 및 순서도

본 논문에서 제안하는 시스템의 구성도는 그림 2와 같다. 먼저 악성코드(PE 파일)가 시스템에 들어오면 PE 파일 정보를 추출한다. 다음으로 추출된 PE 정보를 기반으로 Anti-VM/Debugging 여부를 탐지한다. 다음으로 패킹 여부와 패커 종류를 탐지한다. 탐지된 패커가 well-known 패커일 경우 기존 언패킹툴을 사용하여 정적 분석으로 언패킹하고, custom 패커일 경우 엔트로피 기반 동적 분석으로 언패킹한다.

그림 3은 제안하는 시스템의 순서도이다. 시스템은 PE 정보 추출, 패킹 여부 탐지, 패커 종류 탐지, well-known/custom 패커의 자동 언패킹 단계로 수행된다. 자세한 설명은 다음과 같다.

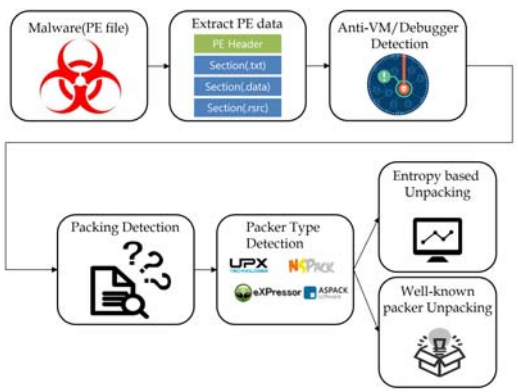


그림 2. 자동 언패킹 시스템 구성도
Fig. 2. Automatic Unpacking System Diagram.

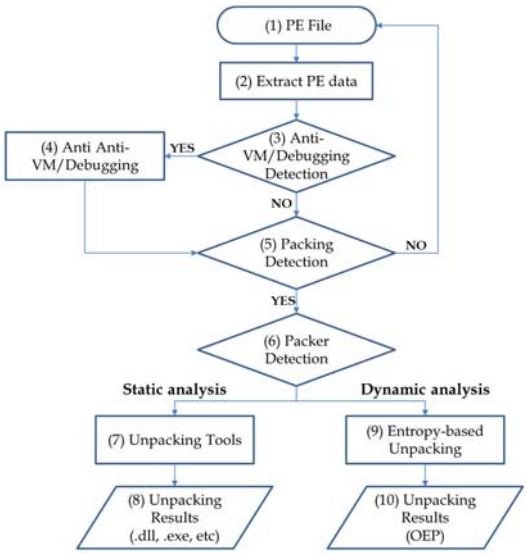


그림 3. PE 파일 자동 언패킹 순서도
Fig. 3. PE File Automatic Unpacking Flowchart.

(1) PE File: 본 시스템에 악성코드 및 PE 파일이 입력으로 주어진다.

(2) Extract PE data: PE 파일의 정보를 추출한다. 이 때 PE 시그니쳐를 검사하여 만약 PE 파일이 아닐 경우 본 시스템은 분석을 더 이상 진행하지 않는다.

(3) Anti-VM/Debugging detection: 추출한 PE 파일 정보의 레지스트리를 기반으로 해당 PE 파일이 Anti-VM/Debugging 속성을 가지고 있는지 확인한다. PE 파일에 Anti-VM 속성이 있다면 (4) Anti Anti-VM/Debugging을 수행하고, 그렇지않다면 (5) Packing detection을 수행한다.

(4) Anti Anti-VM/Debugging: 만약 PE 파일에 Anti-VM/Debugging 속성이 있다면 Anti Anti-VM/Debugging을 수행하여 Anti-VM/Debugging을 무력화한다.

(5) Packing Detection: 추출된 PE 정보를 기반으로 패킹 여부를 탐지한다. 패킹이 되어 있지 않다면 PE 파일 분석을 마치고 (1) 다음 PE 파일을 입력 받고, 패킹된 파일이라면 (6) Packer Detection을 수행한다.

(6) Packer Detection: PE 파일이 패킹되어 있다면 시그니처 기반으로 패커 종류를 탐지한다. 탐지된 패커가 well-known 패커 중 언패킹이 가능한 패커의 경우 정적 분석인 (7) Unpacking Tools을 수행하고, custom 패커인 경우 동적 분

석으로 (9) Entropy-based Unpacking을 수행한다.

(7) Unpacking Tools: 언패킹 툴을 사용하여 언패킹한다.

(8) Unpacking Results: 언패킹된 실행파일을 도출한다.

(9) Entropy-based Unpacking: custom 패커로 패킹된 파일은 동적 분석을 통해 엔트로피 기반으로 언패킹한다.

(10) Unpacking Results: 언패킹 결과(OEP)를 도출한다.

제안하는 시스템에서는 탐지된 패킹 여부와 패커 종류에 따라 정적/동적 분석으로 나누어 수행한다.PE 파일이 well-known 패커로 패킹된 경우 언패킹 툴을 이용한 정적 분석을, custom 패커로 패킹된 경우 엔트로피 기반 동적 분석을 수행한다. well-known 패커와 custom 패커로 구분하여 언패킹을 수행하는 이유는 엔트로피 분석에 따른 오버헤드 때문이다. 따라서 UPX, Aspack, Nspack, Upack 등 well-known 패커로 패킹된 파일은 해당 악성코드의 정보를 이용한 언패킹툴을 사용하는 정적 분석을, custom 패커로 패킹된 파일은 악성코드의 엔트로피를 분석하여 언패킹하는 동적 분석을 수행한다.

정적 분석은 오픈 소스로 제공되는 'PEframe'이라는 정적 분석 툴을 활용한다. 버전 정보 및 메타데이터, 상속된 dll 및 API, 문자열 등이 시스템의 분석 결과로 출력된다. 분석 결과를 기반으로 패커의 종류에 맞는 언패킹 툴을 사용하여 언패킹을 진행한다. 동적 분석은 PE파일을 메모리에 올려 코드의 흐름을 분석한다. 패킹된 파일의 Original Entry Point(OEP)를 찾는 것이 동적 분석에 필수적인 요소이다. OEP는 패킹된 파일이 메모리 상에서 언패킹이 진행 되는 과정에서 원본 파일의 코드가 처음 실행되는 위치를 나타내기 때문이다.

3.2 Anti-VM/Debugging 탐지 및 무력화 설계

본 절에서는 제안하는 시스템에서 사용하는 Anti-VM과 Anti-Debugging 탐지 방법에 대해 설명한다. 먼저 Anti-VM 탐지는 레지스트리 기반으로 동작한다. 레지스트리는 MicroSoft사의 Windows 32/64 Bit 버전의 설정과 선택 항목을 담고 있는 데이터베이스로 모든 하드웨어, 운영체제, 소프트웨어, 비-운영체제 소프트웨어 등에 대한 정보가 저장되어있다[22]. Anti-VM을 위한 윈도우 시스템에 대한 정보는 레지스트리 편집기의 'HKEY_LOCAL_MACHINE\'에서 확인할 수 있다. 해당 항목에는 윈도우에서 사용하는 파일 시스템, 드라이버, 커널(Kernel)이 사용하는 정보와 같이 시스템과 관련된 다양한 정보들이 저장되어있어 악성코드는 이 경로를 통해 시스템 및 하드웨어 정보를 확인하여 가상 머신 여부를 식별한다. 따라서 레지스트리의 데이터 값을 변경하면 Anti-VM 우회가 가능하다[23]. 가상 머신에 올라간 윈도우의 경우 데이터 값에 'VMware', 'Virtual Box', 'VBox'와 같은 가상 머신 프로그램의 문자열이 포함되어 있으므로 가상머신을 나타내는 문자열을 임의의 다른 문자열로 바꾸면 Anti-VM을 우회할 수 있다. 그림 4는 가상 머신을 구동할 수 있는 프로그램인 Virtual Box와 VMware에 설치한 Windows 7의 레지스트리 정보를 일부 추출한 결과이다. 그림에서 'SystemManufac', 'SystemProduct'의 레지스트리 정보에 가상머신을 나타내는 'VMWare' 문자열이 포함돼 있는 것을 알 수 있다.

다음으로 Anti-Debugging 탐지 방법을 설명한다. Anti-Debugging은 Static Anti-Debugging과 Dynamic Anti-Debugging으로 나뉜다. Static Anti-Debugging은 프로그램이 실행될 때 Anti-Debugging이 실행되고 디버기 프로세스에서 자신이 디버깅 당하는지 여부를 파악하는 기법으로, 디버깅 중이라고 판단되면 프로세스를 멈추는 등 일반 실행과 다른 코드를 실행한다. 따라서 Static Anti-Debugging은 디버깅을 시작할 때 한 번만 해제하면 무력화가 가능하다. Dynamic Anti-Debugging은 프로그램의 코드를 트레이싱 하지

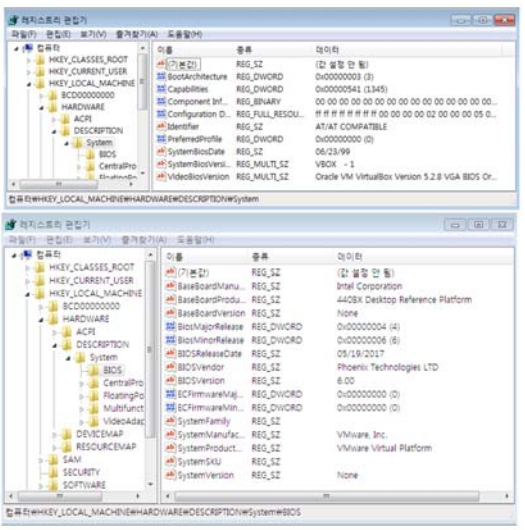


그림 4. Virtual Box, VMware 윈도우의 레지스트리 정보
Fig. 4. Registry data in Virtual Box, VMware.

못하도록 지속적으로 방해하여 원본 프로그램의 코드와 데이터를 확인할 수 없도록 만드는 기법이다. 따라서 Dynamic Anti-Debugging을 무력화시키기 위해서는 디버깅을 진행하면서 Anti-Debugging을 만날 때마다 무력화해야 한다.

제안하는 시스템은 현재 프로세스의 디버깅 여부를 판단하는데 널리 사용되는 PEB(Process Environment Block) 구조체 정보를 이용하여 Static Anti-Debugging을 무력화한다. PEB 구조체의 'BeingDebugged', 'LDR', 'ProcessHeap', 'NtGlobalFlag' 멤버를 'CHECKREMOTEDEBUGGER()', 'IsDebugged()', 'OutputDebugString()' 등의 여러 함수들을 이용하여 분석하면 디버깅 여부를 판단할 수 있다. 이 때 탐지 코드에서 얻어오는 구조체 멤버 정보를 변경하면 Anti-Debugging 우회가 가능하다[8].

또한, Dynamic Anti-Debugging을 우회하기 위해 제안하는 시스템은 예외 처리방법을 이용한다. 정상 프로세스에서 예외가 발생하면 운영체제가 예외를 받아 프로세스에 등록된 SHE(Structured Exception Handling)를 호출하는 반면, 디버깅이 수행되는 프로세스에서 예외가 발생하면 디버거에서 예외를 담당한다. 이러한 정상 프로세스와 디버기의 예외 처리 방법이 다른 특징을 이용해 Anti-Debugging 프로그램이 디버깅 여부를 탐지하기 때문에, 본 시스템에서는 이 중 가장 대표적인 'INT 3' 예외를 무시하여 Anti-Debugging을 우회한다.

3.3 패킹 여부 및 패커 종류 탐지 설계

본 절에서는 본 시스템에서 사용하는 패킹 여부 탐지 방법에 대해 설명한다. 본 시스템에서는 Han[18]이 제안한 진입점 섹션의 엔트로피와 'WRITE'속성을 기반 패킹 파일 탐지를 수행한다. 먼저, 진입점 섹션의 엔트로피 기반 방법은 패킹 여부를 탐지하기 위해 PE 파일의 정보 엔트로피를 사용한다. 전체 PE 파일의 엔트로피 값으로 패킹 여부를 탐지하면 일반 PE 파일의 엔트로피 값과 겹치는 부분이 발생한다. 하지만 진입점 섹션의 엔트로피 값으로 패킹 여부를 탐지하면 패킹된 파일과 일반 파일의 PE 파일의 값이 겹치는 부분이 없어진다[18]. 따라서 제안하는 시스템에서는 Han[18]이 제안한 바와 같이, 패킹 파일과 일반 파일을 구분하기 위해 진입점 섹션의 엔트로피 값을 사용하고 두 파일의 경계의 엔트로피 값 기준을 6.85로 설정한다.

다음으로, 진입점 섹션의 'WRITE'속성 기반 패킹 탐지 방법은 PE 파일의 진입점 섹션에 'WRITE' 속성이 있는지 확인하여 패킹 여부를 탐지하는 방법이다. 패킹된 PE 파일은 패킹을 해제하는 코드와 패킹된 데이터를 쓰는 권한이 필요하므로 진입점 섹션에 'WRITE' 속성을 갖는 특징이 있다[18]. 본 시스템은 이러한 두 가지 특징을 모두 이용하여 패킹 여부를 탐지한다. 즉, 그림 5와 같이 PE 파일에서 진입점 섹션의 엔트로피 값과 속성 값을 모두 검사하여 엔트로피 값이 6.85를 초과하거나 'WRITE' 속성값이 존재하면 패킹된 파일로 탐지한다.

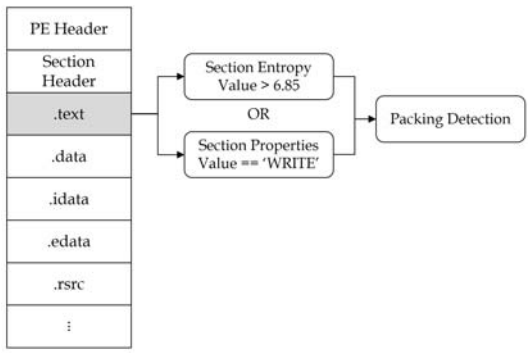


그림 5. 패킹 탐지 프로세스
Fig. 5. Packing Detection Process.

그림 6은 본 시스템에서 사용하는 패킹 여부 탐지 알고리즘이다. 먼저, PE 파일을 입력으로 받고 패킹 여부 탐지 결과를 출력한다. PE 파일을 입력으로 받으면 PE 파일을 분석한다(라인 1). 진입점 섹션의 속성 값이 'WRITE'라면 패킹 탐지 여부를 TRUE로 설정한다(라인 2-4). 그리고 진입점 섹션의 엔트로피 값을 계산한다(라인 5-10). 진입점 섹션의 속성 값이 'WRITE' 이거나 계산된 엔트로피 값이 6.85보다 크면 PE 파일이 패킹되었다고 판단한다(라인 11-12).

```
Procedure Packing Detect
Input File: Malware, PE file
Output PD: Packing Detection Result
1.  Analysis PE File;
2.  if (EPS_Characteristics == 'WRITE' ) then
3.        PES_Char = TRUE;
4.  end if
5.  for e:=0 to length in matrix[]
6.        if (matrix[i] != 0 ) then
7.              t = matrix[i] / length;
8.              EPS_Entropy = EPS_Entropy + t *  log2 1/t
9.        end if
10. end for
11. if (EPS_Entropy > 6.85 || EPS_Char == TURE ) then
12.       PD = TRUE;
13. return PD
```

그림 6. 패킹 탐지 알고리즘.
Fig. 6. Packing Detection Algorithm.

다음으로 본 시스템에서 사용하는 패커 종류 탐지 기법을 설명한다. 본 시스템에서는 시그니처 기반으로 패커 종류를 탐지한다. 대부분의 PE 분석 툴에서 사용하는 각 패커의 시그니처는 userdb.txt 파일에 들어 있으며, 본 시스템에서는 userdb.txt를 업데이트하여 약 6,000개의 시그니처를 확인한다.

Ⅳ. 구 현

본 절에서는 제안하는 시스템의 구현 내용을 설명한다. 시스템 구현 환경은 CPU i5-760, RAM 4GB이며 운영체제는 Window 7이다. Python 2.7로 구현하였으며 PE 파일 헤더를 쉽게 볼 수 있는 모듈인 'pefile'을 사용하였다. 또한, 언패킹 결과를 저장하는 데이터베이스는 MariaDB를 사용하였다. 그림 7은 시스템 UI로 첫 실행 화면이다. File 입력란 ⓐ에 PE 파일을 입력하면 추출한 기본 PE 정보를 ⓑ와 같이 제공한다. 이 단계에서 추출되는 정보는 엔트리 포인트, 엔트리포인트 섹션, PE 파일의 첫 바이트, 파일 오프셋, Anti-VM/Debugging 탐지 여부, 패커 종류이다. 실행 버튼 ⓒ를 누르면 PE 파일 분석이 시작된다.

그림 8은 PE 파일 분석 수행 화면이다. ⓐ에서 PE 파일 분석이 수행되어 파일 오프셋, 엔트리 포인트, 엔트리포인트 섹션, 첫 바이트, 전체 파일의 엔트로피, 파일 크기 등 기본 정보를 추출한다. 다음으로 ⓑ에서 진입점 섹션의 엔트로피와 'WRITE' 속성 여부로 패킹 여부를 탐지한다. 현재 진입점 섹션의 엔트로피 값이 6.85보다 크고 'WRITE' 속성이 있으므로 패킹 파일이라고 탐지한다. 다음으로 ⓒ에서 Anti-VM과 Anti-Debugging을 탐지한다. 그림에서는 Anti-VM/Debugging 모두 탐지되지 않았다.

그림 9는 PE 파일 분석 결과이다. 그림을 보면 ⓐ에서 UPX0, UPX1, .rsrc로 나뉘어진 섹션별 PE 파일의 정보를 확인할 수 있다. ⓑ에서는 그림 7보다 더 자세한 PE 파일 추출 정보를 확인할 수 있다. 이 때

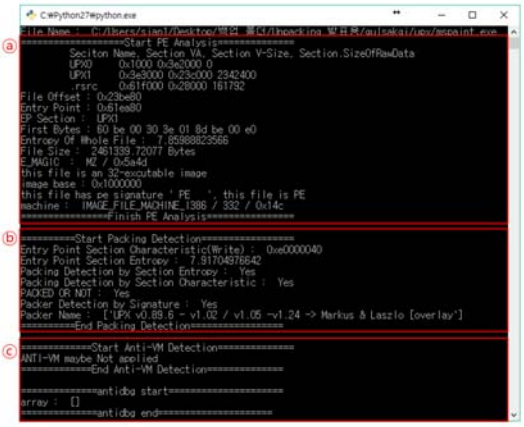


그림 8. 시스템 실행 화면.
Fig. 8. System Execution Screen.

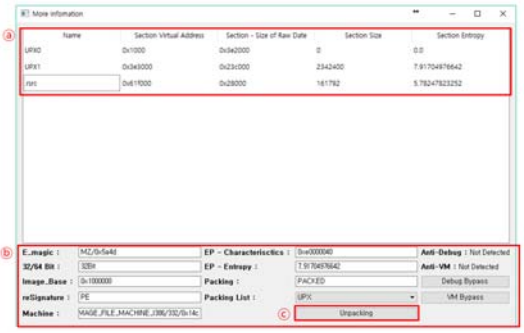


그림 9. PE 파일 정보 추출 결과.
Fig. 9. PE file data extraction result.

탐지된 패커 종류도 확인 가능하며, well-known 패커인 경우, 언패킹 버튼 ⓒ가 생성된다. 언패킹 버튼을 누르면 PE 파일이 언패킹되어 지정된 파일 경로에 저장된다. 그림 10은 well-known 패커로 패킹된 PE 파일을 언패킹한 결과이다. 지정된 경로에 언패킹된 파일이 저장되어 있는 것을 확인할 수 있다. 만약 custom 패커로 패킹된 경우 엔트로피기반 동적 분석을 수행하여 언패킹한다. 엔트로피 기반 언패킹은 본

그림 7. 시스템 UI.
Fig. 7. System UI.

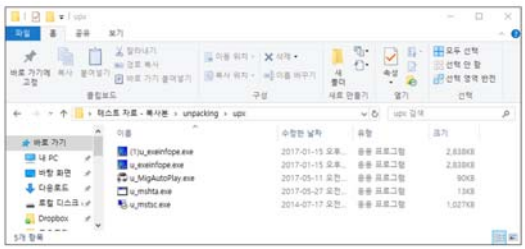


그림 10. 언패킹 결과.
Fig. 10. Unpacking result.

논문에서는 다루지 않고 향후 연구로 수행한다.

V. 검 증

본 절에서는 악성코드 분석을 위한 PE 파일 언패킹 자동 시스템의 성능을 검증한다. 검증은 총 다섯 가지로, PE 정보 추출 검증, Anti-VM 우회 검증, 패킹 여부 탐지 검증, 패커 종류 탐지 검증, well-known 패커로 패킹된 파일의 언패킹 검증을 수행한다.

5.1 PE 정보 추출 검증

먼저 PE 정보 추출 검증에 대해 설명한다. 기존 분석 툴에서 추출하는 PE 정보와 본 시스템에서 추출하는 PE 정보를 비교하여 검증한다. 이를 위해 웹에서 다운 받을 수 있는 파일 10개 선정하여 임의로 UPX, Aspack, Nspack, Upack, Yoda's Protector 패커를 선택하여 2개씩 패킹 후, Exeinfope로 추출한 PE 정보와 본 시스템으로 추출한 PE 정보를 비교한다. 그림 11은 Exeinfope로 추출한 PE 정보이고, 그림 12는 본 시스템으로 추출한 PE 정보이다. 검증 결과 File_Offset과 File_Size를 제외한 정보가 동일하게 추출되었다. File_Size의 경우 근소한 차이를 보이며, File_Offset의 경우 같은 경우와 다른 경우가 존재한다. 이는 Nspack의 패킹 알고리즘 특성에 따른 것으로 사료된다. Upack으로 298개의 파일을 패킹하여 PE 정보 추출을 수행한 결과 확인한 결과 6개의 파일에서 File_Offset 필드가 다른 값으로 추출되었다. Nspack과 Upack의 몇 개의 File_Offset만을 제외하면

File_Name	File_Offset	File_Size	File_Firstbytes	File_Imagebase	File_Machine	EPS_Name	EPS_Address
Anaconda2-4.3.1-Windows-x86.exe	0x28	986576	be b0 11 40 00 ad 50 ff 76 34	0x400000	IMAGE_FILE_MACHINE_I386	PSHeffHedS#0x1018	
DropboxInstaller.exe	0x9ae80	632192	60 e8 09 00 00 00 fe 3a 14 00	0x400000	IMAGE_FILE_MACHINE_I386	rsrc#x00#x00#0x243c80	
ftp.exe	0x10f	197685	e9 41 19 01 00 b4 09 ba 0b 01	0x10000000	IMAGE_FILE_MACHINE_I386	nsp0	0x101b
npp.7.4.1.Installer.exe	0x6601	129903	60 e8 03 00 00 00 e9 eb 04 5d	0x400000	IMAGE_FILE_MACHINE_I386	.aspack	0x6a001
PIL-1.1.7.win32-py2.7.exe	0x17801	100199	60 e8 03 00 00 00 e9 eb 04 5d	0x400000	IMAGE_FILE_MACHINE_I386	.aspack	0x26001
ResTuner_setup.exe	0xe2	898574	e9 1b fd 03 00 b4 09 ba 0b 01	0x400000	IMAGE_FILE_MACHINE_I386	nsp0	0x101b
Universal-USB-Installer-1.9.7.8.exe	0x22b49	111050	e8 03 00 00 00 eb 02 c2 bb 55	0x400000	IMAGE_FILE_MACHINE_I386	.yP	0x63549
Wireshark-win32-2.2.6.exe	0x4a949	229631	e8 03 00 00 00 eb 02 e8 bb 55	0x400000	IMAGE_FILE_MACHINE_I386	.yP	0x7a549
ImmunityDebugger_1_85_setup.exe	0x5980	22724600	60 be 15 f0 43 00 8d be eb 1f	0x400000	IMAGE_FILE_MACHINE_I386	UPX1	0x44780
NEW_GOMPLAYERSETUP.EXE	0x51b0	28581900	60 be 00 40 48 00 8d be 00 d0	0x400000	IMAGE_FILE_MACHINE_I386	UPX1	0x88db0

그림 11. Exeinfope로 추출한 PE 정보.
Fig. 11. PE data extracted from Exeinfope.

File_Name	File_Offset	File_Size	File_Firstbytes	File_Imagebase	File_Machine	EPS_Name	EPS_Address
Anaconda2-4.3.1-Windows-x86.exe	"00000018"	392052	BE B0 11 40 00 AD 50 FF 76 34	"00400000"	0x14c Intel 386	PS 錯誤!	"00001018"
DropboxInstaller.exe	"00098480"	635228	60 E8 09 00 00 00 FE 3A 14 00	"00400000"	0x14c Intel 386	.rsrc	"00143C80"
ftp.exe	"00000018"	20467	E9 41 19 01 00 B4 09 BA 0B 01	"01000000"	0x14c Intel 386	nsp0	"00001018"
npp.7.4.1.Installer.exe	"00006601"	165888	60 E8 03 00 00 00 E9 EB 04 5D	"00400000"	0x14c Intel 386	.aspack	"0006A001"
PIL-1.1.7.win32-py2.7.exe	"00017801"	101376	60 E8 03 00 00 00 E9 EB 04 5D	"00400000"	0x14c Intel 386	.aspack	"00026001"
ResTuner_setup.exe	"00000018"	96418	E9 1B FD 03 00 B4 09 BA 0B 01	"00400000"	0x14c Intel 386	nsp0	"00001018"
Universal-USB-Installer-1.9.7.8.exe	"00022B49"	155136	E8 03 00 00 00 E8 01 C2 BB 55	"00400000"	0x14c Intel 386	.yP	"00063549"
Wireshark-win32-2.2.6.exe	"0004AD49"	331776	E8 03 00 00 00 E8 01 E8 BB 55	"00400000"	0x14c Intel 386	.yP	"0007A549"
ImmunityDebugger_1_85_setup.exe	"00005980"	22725948	60 BE 15 F0 43 00 8D BE EB 1F	"00400000"	0x14c Intel 386	UPX1	"00044780"
NEW_GOMPLAYERSETUP.EXE	"00005180"	28582032	60 BE 00 40 48 00 8D BE 00 D0	"00400000"	0x14c Intel 386	UPX1	"00088D80"

그림 12. 제안하는 시스템으로 추출한 PE 정보.
Fig. 12. PE data extracted from proposed system.

PE 정보 추출의 검증이 잘되었다고 볼 수 있다.

5.2 Anti-VM/Debugging 무력화 검증

다음으로 Anti-VM/Debugging 무력화 검증에 대해 설명한다. 제3.3절에서 설명한 바와 같이, 본 시스템은 레지스트리 기반으로 레지스트리의 데이터 값을 변경하여 Anti-VM을 우회한다[23]. 따라서 'VMware', 'Virtual Box', 'VBox'와 같은 가상 머신 프로그램의 문자열을 수정하면 레지스트리에 의한 Anti-VM 무력화가 가능하다. 하지만 악성코드 제작자가 어떤 경로의 데이터 값을 비교하는지 알 수 없기 때문에 본 시스템은 레지스트리에 있는 모든 문자열을 바꿔 Anti-VM을 우회한다. 그림 13(a)는 수정 전 Virtual Box와 VMware의 윈도우 레지스트리 정보, 그림 13(b)는 문자열을 바꾼 Virtual Box와 VMware의 윈도우 레지스트리 정보이다. 그림 13(a)에서 'SystemBiosVersion', 'VideoBiosVersion' 두 개의 파일에서 각각 'VBOX', 'VM VirtualBox'의 문자열이 검출되어 그림 13(b)와 같이 해당 문자열을 'EDIT_Real Box', 'Eidt_Ver Real Machine' 등의 임의의 문자열로 바꾼 것을 확인할 수 있다. 또한 그림 13(a)에서 'SystemManufaction', 'SystemProduct' 파일에서 'VMware' 문자열이 검출되어 그림 13(b)와 같이 해당 문자열이 'EDIT. Inc', 'EDIT Platform'으로 교체된 것을 확인할 수 있다.

다음으로 Anti-Debugging 무력화 검증을 설명한다. 먼저, 앞서 설명한 바와 같이, Static Anti-Debugging은 PEB 구조체 정보를 이용하여 우회한다. 특히 Windows 7 이상의 OS에서는 CMD에서 'bcdedit /debug off' 명령어를 통해 일부 Static Anti-Debugging을 우회할 수 있다. 그림 14는 Window 7의 cmd에서 해당 명령어를 사용하여 Anti-Debugging을 적용한 것이다.

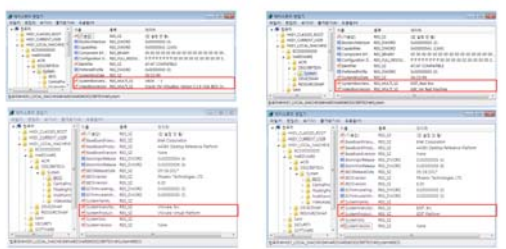

(a) Virtual Box and Vmware's registry information before change. (b) Virtual Box and Vmware's registry information after change.

그림 13. 수정 전, 후의 Virtual Box, VMware의 윈도우 레지스트리 정보.
Fig. 13. Registry data in Vitual Box, VMware before and after change.

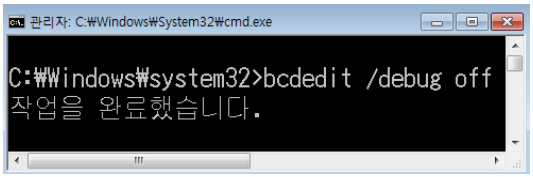


그림 14. Windows에서 제공하는 Static Anti-Debugging 우회 설정.
Fig. 14. Static Anti-Debugging bypass.

Dynamic Anti-Debugging은 정상 프로세스와 디버기의 예외 처리 방법이 다른 특징을 이용해 디버깅 여부를 탐지할 수 있다. 이를 무력화시키는 대표적인 방법은 'INT 3' 예외를 무시하여 Anti-Debugging을 우회하는 것이다. 그림 15는 Anti-Debugging이 적용된 프로그램을 디버거로 분석한 결과이고 그림 16은 디버거에서 'INT3' 예외를 무시하고 프로세스를 진행하여 Anti-Debugging을 우회한 결과이다. 그림 15에서는 디버깅이 탐지된 반면, 그림 16은 디버깅이 탐지되지 않았음을 확인할 수 있다.

그림 15. Dynamic Anti-Debugging이 탑재된 프로그램의 디버깅 탐지 결과.
Fig. 15. Debugging detection result of program with Dynamic Anti-Debugging.

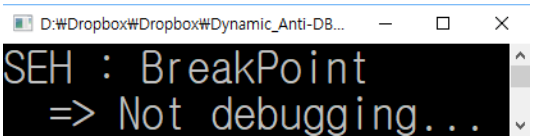


그림 16. Dynamic Anti-Debugging을 우회한 프로그램의 디버깅 탐지 결과.
Fig. 16. Debugging detection result of program that avoids Dynamic Anti-Debugging.

5.3 패킹 여부 탐지 검증

다음으로 패킹 여부 탐지 검증에 대해 설명한다. 패킹 여부는 미리 패킹 해놓은 파일의 데이터와 본 시스템의 패킹 여부 탐지 결과를 비교하여 검증한다. Windows 폴더에서 300개의 PE 파일을 무작위로 선택하여 UPX, Aspack, Nspack, Upack, Yoda's Protector 패커로 패킹한 파일을 실험에 사용한다. 총 UPX 276개, Aspack 300개, Nspack 298개, Upack 289개, Yoda's Protector 267개의 파일을 대상으로 패킹 여부 탐지를 검증하였다. 그림 17은 각 패커별 패킹 여부 탐지 결과이다. 그림을 보면, 5개 패커로 패킹한 데이터로 실험한 경우 모든 패커를 탐지하여 탐지율이 100%로 나타났다. 따라서, 본 시스템의 패킹 여부 탐지 기능은 정상적으로 동작함을 확인하였다.

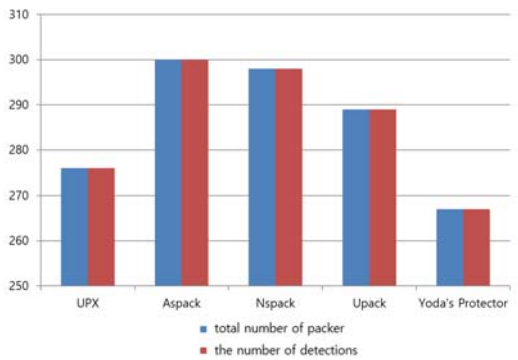


그림 17. 패커별 패킹 여부 탐지율.
Fig. 17. Detection rate of packing per each packer.

5.4 패커 종류 탐지 검증

다음으로 본 시스템의 패커 종류 탐지를 검증한다. 이를 위해 미리 패킹 해놓은 파일들의 패커 종류와 본 시스템의 패커 종류 탐지 결과를 비교한다. 제4.2절에서 사용한 데이터와 동일한 데이터에서 각 패커로 패킹한 파일을 20개씩 무작위로 선택하여 100개의 파일을 실험데이터로 사용한다. 그림 18은 패커 종류 탐지 결과를 나타낸다. 그림을 보면, Nspack을 제외한 다른 패커들로 패킹된 파일의 패커 종류를 100% 탐지하는 것으로 나타났다. Nspack의 경우 시그니처 기반으로 탐지가 되지 않았는데, 이는 현재 패킹한 Nspack의 버전이 userdb.txt에 등록 되어있지 않기 때문이다.

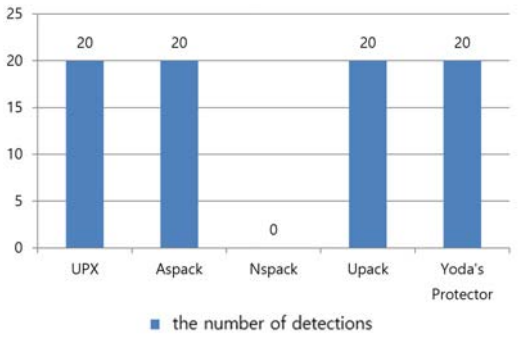
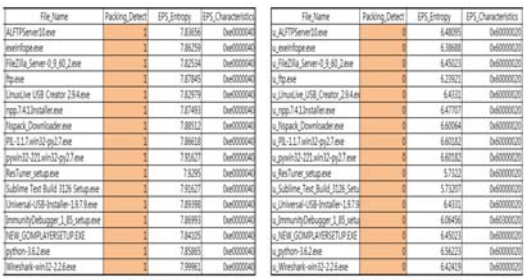


그림 18. 패커 종류 탐지 결과.
Fig. 18. Packer type detection result.

5.5 Well-known 패커의 언패킹 검증

마지막으로 well-known 패커로 탐지된 파일을 자동으로 언패킹하여 본 시스템의 언패킹이 올바르게 수행되는지 검증한다. 언패킹 검증은 상용 프로그램 파일을 UPX로 패킹하고, 제안하는 시스템으로 언패킹을 수행하여 언패킹 전후 패킹 데이터 결과가 동일한지 확인한다. 그림 19는 언패킹 전과 후의 패킹 데이터 비교 결과이다.

그림 19(a)가 언패킹 전, 그림 19(b)가 언패킹 후의 파일 데이터이다. 그림 19(a)에서 진입점 섹션의 엔트로피 값 'EPS_Entropy'가 모두 6.85보다 크고, 패킹 여부를 나타내는 'Packing_Detect' 컬럼 값은 모두 1이다. 따라서 모든 파일이 패킹되어 있다는 것을 확인할 수 있다. 반면 그림 19(b)는 진입점 섹션의 엔트로피 값이 모두 6.85 이하이고, 'Packing_Detect' 컬럼 값이 모두 0으로 파일의 언패킹이 모두 잘 수행되었음을 확인할 수 있다.



(a) Data before unpacking (b) Data after unpacking

그림 19. 언패킹 전후 패킹 데이터 비교 결과.
Fig. 19. Comparison of packing data before and after unpacking.

Ⅵ. 결 론

본 논문에서는 악성코드 탐지 및 분석을 위해 Anti-VM 무력화 환경에서 악성코드 언패킹을 자동으로 수행하는 시스템을 설계 및 구현하였다. 제안하는 시스템은 악성코드의 PE 정보를 분석하는 초기 분석과 초기 분석 결과를 토대로 언패킹을 수행하는 정적/동적 분석 단계로 나뉜다. 먼저 초기 분석 단계에서는 악성코드(PE 파일)가 시스템에 들어오면 PE 파일 데이터를 추출하고 추출된 PE 파일 값을 기반으로 Anti-VM/Debugging 여부를 탐지한다. 다음으로 패킹 여부와 패커 종류를 탐지한다. 이 때, 탐지된 패커가 well-known 패커일 경우 정적 분석 방법으로 기존 언패킹툴을 사용하여 언패킹하고, custom 패커일 경우 동적 분석 방법으로 엔트로피 기반으로 언패킹한다.

Python에서 제공하는 PE 파일 헤더를 쉽게 볼 수 있는 모듈인 'pefile'을 사용하여 구현하였고, 검증을 통해 각 기능이 제대로 동작함을 실험을 통해 확인하였다. 검증 결과, PE 정보 추출의 경우, Exeinfope와 비교하여 파일 크기에서 근소한 차이를 보였으나 파일 크기와 파일 오프셋을 제외한 모든 정보는 다른 PE 분석 툴과 동일한 정보를 추출하였다. 패킹 여부 탐지의 경우 well-known 5개의 패커에 대해 100% 탐지율을 나타냈다. 패커 종류 탐지는 시그니쳐에 존재하는 패커를 잘 탐지하는 것을 확인하였다. well-known 패커의 언패킹을 자동으로 수행하는 기능은 현재 UPX의 경우에 대해서 100% 언패킹을 수행한 결과를 도출하였다.

향후 연구로는 현재 well-known 패커로 사용한 5개의 패커 이외의 다른 패커들을 사용하여 검증할 계획이다. 또한, 본 실험에서 사용하지 않은 패커들에 대해서도 언패킹 방법을 연구하여 본 시스템과 연동하며, Custom 패커를 언패킹 하기 위해 엔트로피 값 변화 기반 언패킹 기능을 구현할 예정이다.

References

[1] AV-Test, *Security Report* (2017), May 3, 2018, from https://www.av-test.org/en/publications.

[2] J. Yu, M. Shin, and T. Kwon, "Analysis of research trend on machine learning based malware mutant identification," *Rev. KIISC*, vol. 27, no. 3, pp. 12-19, Jun. 2017.

[3] C. T. Lim, J. H. Oh, and H. C. Jeong, "Study of technical trends and analysis method of recent malware," *J. KIISE*, vol. 28, no. 11, pp. 117-126, Nov. 2010.

[4] H.-Y. Lim, W.-J. Kim, H.-J. Noh, and J.-S. Lim, "Research on malware classification with network activity for classification and attack prediction of attack groups," *J. KICS*, vol. 42, no. 1, pp. 193-204, Jan. 2017.

[5] S.-K. Yoon and C. Kim, "Machine learning based malware code classifier attack using hostile data," *J. KICS*, vol. 43, no. 1, pp. 77-80, Jan. 2018.

[6] H.-H. Kim and M.-J. Choi, "Android malware detection using auto-regressive moving-average model," *J. KICS*, vol. 40, no. 8, pp. 145-146, Aug. 2015.

[7] T. Brosch and M. Morgenstern, *Runtime Packers: The Hidden Problem* (2006), May 1, 2018, from https://www.av-test.org/fileadmin/pdf/publications.

[8] R. R. Branco, M. B. Gabriel, and D. N. Pedro, "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-VM technologies," in *Black Hat Technical Secur. Conf.*, pp. 1-27, Las Vegas, USA Jul. 2012.

[9] F. Guo, P. Ferrie, and T. C. Chiueh, "A study of the packer problem and its solutions," *Int. Workshop on Recent Advances in Intrusion Detection*, vol. 5320, pp. 98-115, Sep. 2008.

[10] M. Pietrek, *An in-depth Look into the Win32 Portable Executable File Format* (2002), Apr., 25, 2018, from https://msdn.microsoft.com/en-us/magazine/bb985992.aspx.

[11] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. Sixth ACM Conf. Data and Appl. Secur. and Privacy*, pp. 183-194, New Orleans, USA, Mar. 2016.

[12] A. Mohaisen, A. Omar, and M. Manar, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *Computers and Secur.*, vol. 52, pp. 251-266, 2015.

[13] J.-W. Park, et al., "An automatic malware classification system using string list and APIs," *J. Secur. Eng.*, vol. 8, no. 5, Oct. 2011.

[14] N. Idika and P. M. Aditya, *A Survey of Malware Detection Techniques*, *Purdue University*, 2007.

[15] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proc. ACM SIGKDD Workshop on CyberSecurity and Intell. Informatics*, pp. 23-31, Paris, France, Jun. 2009.

[16] G. Jeong, et al., "Generic unpacking using entropy analysis," *JAITC*, vol. 7, no. 1, pp. 232-238, Feb. 2009.

[17] S. Cesare and X. Yang, "Classification of malware using structured control flow," in *Proc. Eighth Australasian Symp. Parall. and Distrib. Comput.*, vol. 107, pp. 61-70, Brisbane, Australia, Jan. 2010.

[18] Y.-H. Lee, et al., "A study on generic unpacking using entropy Variation Analysis," *Journal of the Korea Institute of Information Security and Cryptology*, Vol. 22, No. 2, pp. 179-188, Apr. 2012.

[19] R. Lyda and J. Hamrock, "Using entropy analysis to find encrypted and packed malware," *IEEE Secur. and Privacy*, vol. 5, no. 2, pp. 40-45, Apr. 2007.

[20] Y.-S. Choi, et al., "PE file header analysis-based packed PE file detection technique (PHAD)," *IEEE CSA'08,* doi: 10.1109/CSA.2008.28, pp. 28-31, Oct. 2008.

[21] S. Han and S. Lee, "Packed PE file detection for malware forensics," *KIPSTC*, vol. 16, no. 5, pp. 555-562, Oct. 2009.

[22] Microsoft MSDN Library, *About the Registry*, Apr., 18, 2018, from https://msdn.microsoft.com/ko-kr/library/windows/desktop/ms724182(v=vs.85).aspx.

[23] A. Ortega, *Your Malware Shall Not Fool Us with Those Anti Analysis Tricks* (2012), Apr., 18, 2018, from https://www.alienvault.com/blogs/labs-research.

김 선 균 (Sun-Kyun Kim)

2016년 2월 : 강원대학교 컴퓨터과학과 졸업

2018년 2월 : 강원대학교 컴퓨터과학과 이학석사

<관심분야> 네트워크 관리, 악성코드 언패킹, Anti-VM, Anti-Debugging

김 하 진 (Hajin Kim)

2016년 2월 : 강원대학교 사회학과 졸업
2018년 2월 : 강원대학교 컴퓨터과학과 이학석사
<관심분야> 데이터 마이닝, 악성코드 언패킹

최 미 정 (Mi-Jung Choi)

1998년 2월 : 이화여자대학교 공학사
2000년 2월 : 포항공과대학교 공학석사
2004년 2월 : 포항공과대학교 공학박사
2004년~2005년 : 프랑스 INRIA 연구소 박사후 연구원
2005년~2006년 : 캐나다 워터루대학 박사후 연구원
2006년~2008년 : 포항공대 컴퓨터공학과 연구 조교수
2008년~현재 : 강원대학교 컴퓨터학부 컴퓨터과학전공 교수
<관심분야> 네트워크 관리, 정보보안, 악성코드 언패킹