情報保護學會論文誌
第16卷 第6號, 2006. 12

삭제된 파일 조각에서 기계어 코드 유사도를 이용한 악의적인 파일 탐지에 대한 연구*

류 동 주[1†], 이 석 봉[2], 김 민 수[3‡]
[1](주)IT Bank, [2]전남대학교, [3]목포대학교

A Study of Detecting Malicious Files using Similarity between Machine Code in Deleted File Slices

Dongju Lee[1†], SukBong Lee[2], Minsoo Kim[3‡]
[1]IT Bank, [2]Chonnam National University, [3]Mokpo National University

요 약

컴퓨터 포렌식스에서 파일 시스템은 사이버범죄의 증거를 수집할 수 있는 대상이다. 이에 따라 파일 시스템을 복구하고 중요한 정보를 찾는 방법은 많이 제시되고 있다. 그러나 조각난 파일이나 파일 지스러기 공간에서 악성 파일을 찾는 방법은 제시되고 있지 않다. 본 논문에서는 파일 조각이 악의적인 파일인지를 조사하는 방법을 제시한다. 조각 파일 내의 기계어 코드 비율을 검사하여 실행파일인지를 판단하고, 명령어 시퀀스 유사도를 비교하여 악성파일인지를 판단한다. 명령어 시퀀스 유사도를 검사하기 위해, HMM을 이용하여 악성 파일을 프로파일링하고 연속된 평가값을 비교하는 방법을 제시한다. 이러한 방법을 적용하여 적절한 임계 수준에서 버퍼오버플로우 공격 특성을 갖는 악의적인 실행 파일을 정확히 가려낼 수 있었다.

ABSTRACT

A file system is an evidence resource of cyber crime in computer forensics. Therefore the methods of recovering the file system and searching important information have been offered. However, the methods for finding a malicious file in free blocks or slack spaces have not been suggested. In this paper, we propose an investigation method to find a maliciously executable fragmented file. After estimating if a file is executable with a machine code rate, we conclude it could be malicious by comparing a similarity of instruction sequences. To examine instruction sequences, we also propose a method of profiling malicious files using HMM and a method of comparing the continued scores. As the results, we could exactly pick out the malicious execution files, such as buffer overflow attack program, at fitting threshold level.

Keywords : *컴퓨터 포렌식스 (computer forensics), 파일 복구 (file recovery), 해킹(hacking), 명령어 시퀀스(instruction sequence)*

접수일: 2006년 8월 28일 ; 채택일: 2006년 10월 11일
* 이 논문은 2005년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2005-003-D00365).

† 주저자, ryu@gwangju.ac.kr
‡ 교신저자, phoenix@mokpo.ac.kr

Ⅰ. 서 론

최근 컴퓨터 기술의 급속한 발전으로 인해 기존의 텍스트 위주의 사용자 환경에서 벗어나 이미지, 그래픽, 오디오 및 비디오 데이터 등을 제공하는 멀티미디어 사용자 환경으로 변환하고 있다. 정보통신의 발달과 인터넷을 이용한 전자 상거래 및 금융 서비스 등 다양한 서비스와 컴퓨터의 활용도가 높은 사회에서 이를 악용하는 사례가 발생하고 있다. 실생활에서의 범죄가 발생하면 범죄에 사용된 증거를 수집하여 범행 행위를 입증하고 범인을 찾아낸다. 컴퓨터를 활용한 사이버 범죄의 경우에도 동일하게 증거를 확보하고 범죄 행위를 입증하여 범인을 체포할 수 있는 방안이 필요하다. 그러므로 컴퓨터에서 증거를 확보하는 연구 분야로 컴퓨터 포렌식스 분야가 발전하고 있다.[1]

사이버 범죄에서 조각난 파일이나 지스러기 공간(slack space)은 해커에 의해 컴파일된 데이터, 추후에 범죄에 이용하기 위해 남겨둔 이진 파일, 해커가 설치 후에 지웠던 파일 등의 정보를 찾아볼 수 있는 곳이다.[2] 이러한 정보는 해커의 침투 의도를 파악하고 피해상황을 분석할 수 있으며, 컴퓨터 포렌식스 절차상 필요한 증거로 제시될 수 있다. 본 논문에서는 파일 조각의 특성을 파악하고 그 의도를 찾아내기 위한 방안을 제안한다.

본 논문에서는 위와 같은 목적으로 다음과 같은 세단계로 연구를 수행한다. 먼저, 어셈블리 명령어 검출을 통해 실행파일 조각을 추출하는 방법을 연구한다. 두 번째로 실행파일 조각의 의도를 파악하기 위하여 악의적인 실행파일에 대한 프로파일링을 방법을 연구한다. 마지막으로 실행파일 조각이 어떠한 의도를 가졌는지를 평가하기 위해 프로파일링 정보와의 유사도를 산출하는 방법을 연구한다.

첫 번째로, 파일 조각이 실행파일임을 판단하기 위하여 명령어 코드 비중 검사를 수행한다. 텍스트 파일은 문자열 검사로 쉽게 구분할 수 있지만 실행파일은 이진코드로 되어 있어서 구분이 쉽지 않다. 단지 실행파일의 경우 역어셈블을 수행하면 명령어 코드로 변환할 수 있다는 특징을 갖는다. 조각 파일이라도 명령어 코드 분석을 수행할 경우 일반적인 이진파일인지 실행파일인지를 어느 정도 구분할 수 있다. 본 논문에서는 일차적으로 조각 파일에 명령어 코드가 어느 정도 포함되어 있는지를 통해 실행파일 여부를 판단한다.

두 번째로, 악의적인 실행파일을 검출하기 위해 HMM(Hidden Markov Model)을 사용하여 프로파일링을 한다. HMM은 표본모델의 반복적 수행을 통하여 입력된 데이터를 비교, 추출값을 제공하지만 정확한 표본 모델 데이터가 선행되어야만 하고 정상적 모델을 구축해야 하는 어려움을 가지고 있다. 일반적인 실행파일의 경우 일정한 명령어 코드가 존재한다. 본 논문에서는 역어셈블링이 수행 가능한 라이브러리 Libdisasm을 사용하여 조각파일을 역어셈블링 한다. 모든 명령어 코드를 사용할 경우 HMM 학습시간에 큰 영향을 주기 때문에 빈번하게 사용되는 명령어 코드를 분석하여 25개의 명령어 코드만을 사용할 수 있도록 한다.

마지막으로, 파일 조각이 악성파일인지를 검사하기 위해 공격용 파일에 대하여 HMM 학습을 수행하여 악성파일에 대한 프로파일을 생성한다. 본 논문에서는 실험용 악성파일로 알려진 버퍼오버플로우(Buffer Overflow) 공격 프로그램을 사용하였으며, 이러한 프로파일을 적용하여 정상적인 파일과 버퍼오버플로우 공격 파일에 대한 탐지 성능을 실험하였다. 특히 HMM 프로파일의 평가값을 직접 이용하는 것보다 연속된 평가값을 누적 적용함으로써 오류를 줄일 수 있었다. 이러한 판단의 정확성을 높이기 위해 여러 파일을 대상으로 실험한 결과를 토대로 HMM 평가값에 대한 판단 기준을 제시하였다.

Ⅱ. 관련 연구

2.1 컴퓨터 포렌식스 분석 기법

컴퓨터 보안사고의 증가로 인하여 체계적으로 범죄를 수사하는 컴퓨터 포렌식스의 필요성이 강조되고 있다. 잠재적인 증거 손실을 막고 발생한 사이버 범죄에 대한 책임의 소지를 규명해야 함에 따라 체계적인 수사 방법이 요구되고 있다.[3] 현재의 컴퓨터 포렌식스 기술은 침입탐지 기술을 이용하는 수준이며 가독성 있는 텍스트 정보의 검색 방법에 의존하고 있다. 특히 해커가 공격에 이용한 실행파일을 발견하더라도 실행하지 않고 판단하는 방법이 제시되고 있지 않다.

현재 알려진 포렌식 도구로는 TCT(The Corner's Toolkit)나 Sleuth Kit이 공개용으로 제공되고 있고 사건이 발생한 시스템의 상태 정보를 스냅샷의 형태로 생성하며 백업된 디스크 이미지 분

석 및 파일 복구에도 매우 유용하게 사용될 수 있다.[1,4] 상업용으로 알려진 인케이스(EnCase)는 증거 보존과 분석 기능을 모두 갖추고 있으며 삭제된 파일, 폴더, 비할당 클러스터에 대한 검색 및 복구 기능을 지원하고 있다.[5] 이러한 도구들은 복구된 블록의 연관성을 분석하여 블록의 유형을 결정하지만, 파일의 헤더 정보가 없으면 연관성을 찾지 못한다.[6]

피해 시스템의 파일시스템이 포렌식스 검사시스템에 마운트 되었다면 가장 먼저 파일시스템에 의존한 검사를 통하여 범죄와 관련되어 있을 가능성이 있는 모든 정보를 수집하여야 한다. 특히 추적회피를 위한 안티 포렌식(anti-forensic)에 대비하여 파일시스템에 의존하여 검사할 수 없는 영역인 파일 지스러기 영역 및 자유 데이터 블록의 정보를 복구하고, 해당 정보가 발생한 보안 침해 사고와 관련이 있는지를 검사한다. 마지막으로 단일 개체로는 판단할 수 없는 정보의 경우, 정보간의 상호 관계를 파악하고 블록정보간의 연관성을 근거로 판단하여야 한다.[3]

현재까지 확인된 컴퓨터 포렌식스 수사도구들은 파일 복구기능과 파일의 링크가 깨진 부분을 복구하는 기능만을 수행하고 있다.[7] 따라서 악의적 파일이 분산되어 저장된 경우나 조각난 파일에서의 실행코드 여부의 판단가능성은 매우 희박하다. Schultz는 GNU *strings*를 이용하여 검출된 문자열을 이용하여 악성파일을 구분하는 방법을 제시하였다.[8] 문자열을 이용한 구분 방법은 쉬운 접근방법이지만, 악성파일에서 숨기는 경우가 많이 있으므로 정확한 판단근거가 되기는 어렵다. 이를 보완하기 위해 Schultz는 *hexdump*를 이용하여 추출된 16진수 코드값에 데이터마이닝을 적용하여 생성된 시퀀스를 이용하는 방법도 제시하였다.[8] 그러나 이렇게 생성된 시퀀스는 실행파일의 특성이 반영되지 않는 데이터로 일반적인 이진(binary) 파일의 16진수 코드값과 비교하기에는 그 정확성이 떨어진다. 이에 비해 어셈블리 명령어의 루프 구간에 대한 특성을 프로파일링하여 악성파일을 찾는 방법이 연구되었다.[9] 그러나 프로파일링된 루프 구간 특성이 악성파일을 대표한다고 말하기 어렵다는 문제점이 존재한다.

본 논문에서는 삭제된 파일의 명령어 코드를 확인하고 이를 가공하기전의 데이터와 깨끗한 실행파일과의 유사도 추출과정을 분석하여 탐지해내는 방법을 연구한다. 리눅스에서는 역어셈블링을 이용하는 Libdisasm이라는 모듈을 사용하는 방법이 활발히 연구 중이다.[10] 이러한 리눅스 기반의 역어셈블링 기능을 응용함으로써 실험에 사용할 수 있는 명령어 코드를 쉽게 선택할 수 있다. 본 논문에서는 파일시스템에 존재하는 모든 형태의 정보에 대한 연관성을 탐지하기 보다는 검출된 이진 데이터가 실행파일의 조각인지, 만약 실행파일의 일부분이라면 원래의 실행파일이 악의적으로 설치되었는지에 대한 판단 방법을 제시한다.

2.2 HMM 응용 제안 기법

운용중인 시스템 상에서 어떤 파일들을 실행시킴으로써, 실질적인 사용자 허가권을 습득할 수도 있으며 새로운 프로세스를 생성함으로써 운영체제의 메모리 부분을 덮어씀으로서 버퍼오버플로우 공격이 가능하게 하기도 한다. 실제 커널에서의 메모리 관리 시스템은 메인 메모리 안에 데이터를 배치하게 되고 그러면 메인 메모리 또는 스왑 파일시스템상의 다른 배치되지 않은 데이터를 덮어쓴다. 이렇게 수행되는 리눅스 커널에서의 메모리 역할 상 다시 덮어쓴 부분을 컴퓨터 포렌식스 도구를 이용하여 복구한다고 해도 실질적인 파일정보를 담고 있는 헤더의 정보가 사라질 수도 있다. 또한 디스크 구조상 아이노드(i-node)에서의 링크 플래그 값이 변한 것이라면 복구가 가능하겠지만 악의적 파일의 경우 단편화(fragmentation)시켜 조각난 파일이 무슨 파일인지 파악하기에는 상당히 난해할 수밖에 없는 것이 현재 컴퓨터 포렌식스 수사의 현실이다.[11] 그러므로 파일정보가 사라지거나 악의적 조작에 의해 변형된 정보를 일정한 사전 정의된 프로파일 패턴 모델을 기준으로 삼아 유사성을 정해진 규칙에 따라 순환반복 검사함으로써 완전한 정보를 복구하기는 불가능해도 숨겨진 파일의 기본 형태 즉, 악의적 파일인지 단순 파일인지의 판단유무는 가능할 수 있어야 한다. 본 논문에서는 HMM을 이용하여 일정비율의 코드를 사용하는 악의적 파일에 대한 정보를 사전 모델로 구성하여 상대적이기는 하나 비슷한 파일의 유사도를 검사하여 악의적 파일 유무를 판단하는 방법을 제안 한다.

HMM은 초기상태 확률분포 π와 상태전이 확률분포 A, 관측기호 확률분포 B로 구성된다. HMM은 정의된 행렬에만 의존해서 확률적으로 대상을 모델링하여 특정 관찰 열이 구축된 모델로부터 생성되었을 확률 및 최적의 상태전이를 구할 수 있다. 이러

한 HMM의 특성은 시스템에서 생성되는 관찰결과로부터 정상행위를 구축하고 현재의 행동이 정상행위인지를 평가할 수 있는 자연스러운 도구를 제공한다.[12] 또한, HMM의 모델링과 평가를 위한 절차들은 잘 정립된 수학적 배경을 가지고 있어서, 음성인식을 포함한 여러 분야에서 소스가 알려지지 않은 대상을 모델링하는데 널리 유용성을 인정받고 있다.[13]

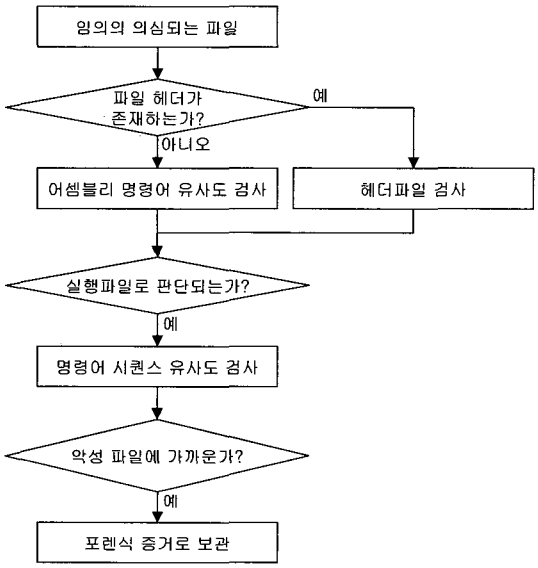
행위 모델링은 설계 모델로부터 주어진 시퀀스 O가 나올 수 있는 확률 값인 $P(O|\lambda)$가 최대가 되도록 HMM의 구성요소들을 조정해 나가는 과정인데 이렇게 생성된 모델에 실험에 사용될 조각난 파일을 입력하고 일반적 실행파일의 모델에서 입력된 명령어 코드가 나올 확률 값을 구해 평가한다. 이러한 과정은 HMM의 포워드-백워드 알고리즘을 이용하여 교차 검사를 수행하게 된다. 행위 모델링은 모델링 데이터에 따라 정상행위 프로파일링과 비정상행위 프로파일링 방법으로 구분된다.[12] 본 논문에서 사용하는 대상은 명령어 파일 조각이므로 다양한 정상 실행파일보다 악의적인 실행파일만을 대상으로 모델링하여 판정하기로 한다.

III. 파일 조각에서 어셈블리 명령어를 이용한 실행파일 판정

어셈블리 명령어 코드가 유사하다는 것을 판정하는 방법으로 일부 코드를 직접 비교하는 방법을 생각할 수 있다. 이러한 방법은 두 코드가 같다는 것을 판정하는 직접적인 방법이다. 그러나 명령어 코드는 컴파일 환경이나 방법에 따라 달라지고, 공격자가 소스를 일부 수정한 경우도 많아 실제로 적용하기는 어렵다. 직접적인 증명이 어렵다면, 간접적으로 통계적인 방법을 사용하거나 인공지능적인 방법을 사용할 수 있다. 본 논문에서는 [그림 1]과 같은 과정을 통해 어셈블리 명령어 코드의 비중을 검사하는 방법과 어셈블리 명령어 시퀀스의 특성 비교 방법을 제안한다.

3.1 어셈블리 명령어 검출을 통한 실행파일 조각을 추출

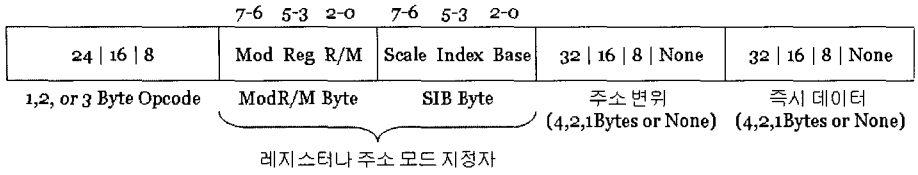
실행 파일은 일반적으로 헤더 정보, 어셈블리 명령어, 라이브러리 등으로 구성되어 있다. 실행파일에서 어셈블리 명령어들을 모두 추출해 낼 수 있다면, 실행파일을 소스 프로그램으로 복원이 가능하게 될 것이다. 그런데, 실행파일의 코드 세그먼트 외에는 판단하기 어려워 어셈블리 명령어만으로 실행파일인지를 판단하는 것은 쉽지 않다. 또한 헤더 정보가 없다면 어셈블리 명령어가 포함되는 코드 세그먼트 위치를 정확히 찾는 것조차 힘들다. 파일 형식에서 로그파일은 패턴이 일정하여 파일 구분이 쉬우나 실행파일의 경우 이미지나 압축파일과 같은 다른 이진파일과의 차이점을 찾기는 쉽지 않다.



[그림 1] 파일 조각이 악성 실행파일 검사 흐름도

ELF 형식으로 작성된 실행파일의 검출은 리눅스 시스템의 기본 명령어 중 하나인 file 프로그램을 이용할 수 있다. file 명령어는 파일의 헤더부분이나 작성 형식을 검사하여 어떠한 형식의 파일인지 알려준다. 텍스트, 이진파일, 또는 장치 등의 모든 파일 형태가 가능하다. 그러나 file 명령어는 각 파일의 특성 정보를 담고 있는 매직 파일을 갖고 검사하는 파일이 어떠한 형태에 부합하는지 검색하는 방법을 이용하고 있으므로 알려지지 않은 새로운 방식으로 기록되었거나 이진 파일의 헤더부분이 손실되었다면 형식을 알아낼 수 없다. 본 논문에서는 여러 실행파일 형식 중, 다양한 운영체제를 지원하고 32비트 인텔 아키텍처[14]에서 동작하며, 동계열의 어셈블리 명령어 코드로 작성된 ELF 형식의 실행파일 탐지 방법을 연구하였다.

본 논문에서 실행파일 여부를 판단하는 대상은 완전한 포맷이 아니라 파일의 단편이기 때문에 헤더



〔그림 2〕 기계어 명령 형식

정보를 이용한 기존의 탐지방법으로 판단이 불가능하다. 본 논문에서 제시하는 방법은 파일시스템에서 검출될 수 있는 실행파일은 사용자가 접근할 수 있는 일반적인 실행파일, 파일이 이미 삭제되어 다른 파일을 기록할 때 할당할 수 있는 데이터 블록에 기록되어 있는 경우, 그리고 해당 파일이 이미 삭제되었으나 그 전에 존재했던 데이터 블록이 다른 파일에 할당되었으나 지스러기 공간에 파일의 일부가 남아 있는 경우에서 적용 가능하다.

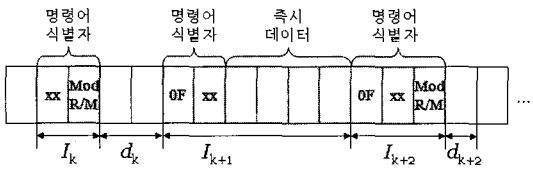
3.1.1 기계어 명령 구조

Interl 계열의 기계어 명령어는 〔그림 2〕와 같은 구조를 가진다.[14] 1~3바이트의 Opcode가 있고, Opcode에 따라 선택적으로 ModR/M 바이트와 SIB 바이트가 뒤따른다. 경우에 따라서 확장주소를 위해 주소 변위 (address displacement)가 쓰일 수 있고, 직접 데이터 주소를 지정하는 즉시 데이터 (immediate data)가 쓰일 수 있다. Opcode는 1 ~ 3바이트의 크기를 가진다. 2바이트 Opcode는 첫 바이트 값이 0Fh 값을 가진다. 3바이트 Opcode는 2바이트 값이 0F38h와 0F3Ah 이어야 한다. 오퍼랜드(operand)를 사용하는 명령에서는 레지스터나 주소모드 지정자(ModR/M이나 SIB 바이트)를 이용하여 오퍼랜드를 표시한다. 시스템에서 사용하는 어셈블리 명령어는 Opcode로 기본 특성이 분류되고 ModR/M에 따라 다양하게 제공된다.

본 논문에서는 명령어 식별자(identifier)를 Opcode + ModR/M 값으로 사용한다. 실제로 기계어 명령어에 대응하는 어셈블리 명령어는 Opcode 뿐만 아니라 ModR/M 값에 따라 다르게 표현되기 때문이다. 일반적인 목적으로 사용되는 32비트 명령어 코드를 대상으로 1~3바이트 명령어 식별자를 구분하여 보았다.[14] 1바이트만으로 사용가능한 Opcode의 수는 167개이며, ModR/M을 붙여서 사용가능한 Opcode의 수는 70개이다. 2바이트 Opcode 중에서 ModR/M을 붙여서 사용가능한 Opcode 수는 48개이며, 그렇지 않는 Opcode 수는 33개이다. ModR/M 바이트가 모두 사용가능하다고 할 때, 2바이트 명령어 식별자 수는 $70 \times 256 + 33 = 18,023$개이며, 3바이트 명령어 식별자 수는 $48 \times 256 = 12,288$개이다. 따라서 1바이트를 선택하였을 때 명령어 식별자라고 말할 수 있는 확률은 $167 / 2^8 = 65.23\%$이다. 그에 비하여, 2바이트의 경우 명령어 식별자로 구분할 수 있는 비율은 $18,023 / (2^8 \times 2^8) = 27.50\%$이고, 3바이트의 경우 명령어 식별자로 구분할 수 있는 비율은 $12,288 / (2^8 \times 2^8 \times 2^8) = 0.07\%$이다. 결국 1바이트 명령어 식별자로는 명령어 코드를 구분하기 어렵다고 볼 수 있다.

3.1.2 명령어 식별자 추출

기계어 명령은 종류가 매우 다양하고 오퍼랜드를 사용하는 형식도 다양하여 기계어 명령을 추출하는 것은 복잡한 구조에 대한 이해와 많은 계산을 필요로 한다. 이에 본 논문에서는 기계어 명령 부분에 대한 비중을 검사하여 기계어 명령에 대한 코드 분석이 없이도 쉽게 실행파일 여부를 판단할 수 있는 방법을 제시한다.



〔그림 3〕 명령어 식별자와 명령어 크기 계산

실행파일에서 어셈블리 명령어 부분의 비중을 검사하는 방법은 명령어 코드에 대한 분석이 없이도 쉽게 실행파일 여부를 판단할 수 있게 해준다. 다음은 실행파일의 비중을 검사하는 절차이다.

단계 1. 명령어 식별자 테이블 생성 - 명령코드와

ModR/M의 크기가 2~3바이트인 경우에 해당하는 모든 조합을 테이블로 기록

단계 2. 명령어 추출 - 이진 조각파일에서 2~3바이트를 읽어 들여, 2~3바이트 명령어 식별자인지를 비교한다. 명령어 식별자 테이블에 없는 명령어의 경우는 1바이트만 검사하고 단계 2를 다시 수행한다.

단계 3. 명령어의 크기 계산([그림 3]) - 발견된 명령어가 즉시데이터나 주소변위가 존재하면, 해당 크기만큼을 명령어크기(바이트)를 누적한다.
명령어크기(I_k) = 명령어식별자크기 + 주소변위(바이트) + 즉시데이터(바이트)

단계 4. 명령어 구간 거리 계산 - 명령어크기가 지정된 명령어 구간거리보다 작으면 구간거리와 명령어크기의 차이(d_k)만큼 파일 포인터를 이동. 파일 끝을 만나면 단계 5로 가고 그렇지 않으면 단계 2로 돌아감.

단계 5. 명령어 비율 계산 - 발견된 명령어크기의 누적이 전체 데이터에서 차지하는 비율을 계산하여, 실험에 의해 책정된 임계값(δ)보다 높으면 ELF 실행파일로 판단한다. 이에 대한 수식은 아래와 같으며, 여기서 N은 탐색된 명령어의 개수이고 S는 테스트 이진데이터의 바이트 크기이다.

$$\frac{\sum_{k=1}^{N} I_k}{S} > \delta.$$

본 논문에서 제안하는 실행파일 탐지 기법은 임의의 이진파일 단편에서 어셈블리 명령어가 차지하는 비율을 계산하여 임계값 이상을 차지한다면 실행 파일로 판단한다.

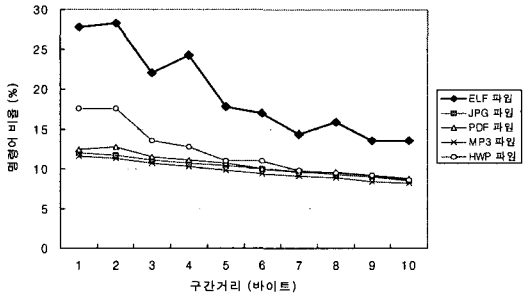
3.2 어셈블리 명령어의 유사도 검사 결과 분석

위의 명령어 크기를 계산하는 부분에서 1바이트로 이루어진 Opcode, SIB 바이트, 그리고 Prefix 바이트는 제외되었으며 가변적인 주소변위나 즉시데이터는 고정된 크기로 계산하였다. 그러나 이러한 바이트로 인하여 명령어 식별자를 추출하는 위치에 오류가 발생할 수 있다. 즉, 명령어 구간거리를 사용하는 이유는 명령어 크기 계산에서 제외된 부분이나 가변적인 바이트로 인한 오류를 줄이기 위한 것이다.

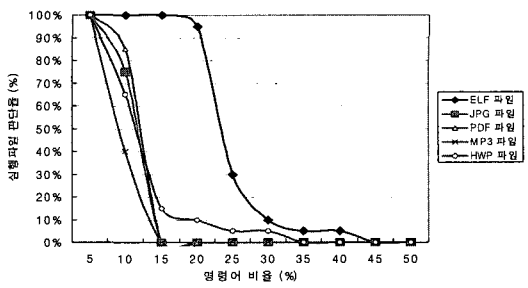
본 논문에서 3.1.2에서 제시한 방법을 여러 종류의 파일에 적용하여 명령어 비율을 계산하여 보았다. [표 1]은 명령어 구간거리를 1바이트에서 10바이트까지 변경하며 ELF 실행파일, JPG 파일, PDF 파일, MP3 파일, 그리고 HWP 파일 각각 20개에 대하여 명령어 비율을 계산한 것을 보여준다. 전체적으로 구간거리를 늘릴수록 명령어 비율값이 줄어듦을 알 수 있으며 ELF 파일과의 차이값도 줄어듦을 알 수 있다. 이러한 결과로 구간거리를 적게 잡을수록 명령어 비율이 높고 기타 파일과의 차이도 더 커진다. 그런데 JPG, PDF, 그리고 MP3와 같이 전체적인 형식이 일정한 파일의 경우 명령어 비율이 일정하게 나타나지만, HWP와 같이 여러 형식이 들어갈 수 있는 문서파일은 구간거리가 짧았을 때 명령어 비율이 매우 높게 측정되는 경우가 일부 존재하므로 이러한 부분은 오류를 발생시킬 수 있다. 또한 일반적인 유닉스나 리눅스 시스템의 데이터는 2바이트나 4바이트 단위로 분리되는 특성이 있다. [표 1]에서도 2바이트와 4바이트 부분이 다른 부분에 비해 차이값이 더 크게 나왔으며 [그림 4]와 같이 그래프로 보더라도 4바이트에서 특징이 도드라짐을 알 수 있다. 따라서 본 논문에서는 적절한 구간거리로 4바이트를 적용하기로 한다.

[표 1] 구간거리에 따른 명령어 비율 차이 조사

	파일 종류		구간거리 (bytes)									
			1	2	3	4	5	6	7	8	9	10
명령어 비율 (%)	ELF 실행파일		27.78	28.27	22.03	24.28	17.78	17.01	14.36	15.83	13.58	13.50
	기타 파일	JPG	12.04	11.68	11.15	10.68	10.40	9.96	9.69	9.26	9.00	8.53
		PDF	12.37	12.71	11.55	11.11	10.72	10.05	9.60	9.63	9.20	8.75
		MP3	11.58	11.36	10.69	10.27	9.79	9.42	9.13	8.86	8.43	8.21
		HWP	17.53	17.63	13.58	12.74	11.00	11.03	9.78	9.53	9.22	8.60
	차이값		14.40	14.93	10.29	13.08	7.31	6.90	4.81	6.51	4.62	4.98



〔그림 4〕 구간거리에 따른 명령어 비율 측정



〔그림 5〕 명령어 비율에 따른 판단의 정확도 비교

〔표 1〕에서의 결과처럼 실제로 명령어 비율은 그렇게 높지 않게 나타난다. 그것은 실행파일에서 데이터세그먼트가 존재하고, 코드 부분에서도 직접데이터가 사용될 수 있으며, 1바이트 명령어 식별자가 많이 쓰이기 때문일 것이다. 따라서 실행파일임을 판단하기 위해서는 명령어 비율에 대한 적절한 임계값을 찾아야 한다. 〔그림 5〕는 구간거리를 4로 하여 다섯 가지 종류의 파일에 대하여 명령어 비율을 5%에서 50%까지 변경하며 실행파일 판단율을 비교해본 것이다. 15%일 때 JPG, PDF, MP3 파일들은 정확히 구분되고 있으나 HWP 파일들은 오류가 약간 존재함을 알 수 있다. 이러한 오류는 판단 기준을 30%까지 늘려도 여전히 존재한다. ELF 실행파일들은 명령어 비율을 20%로 변경할 때부터 정확성을 잃어간다. 이러한 결과로 임계값을 설정할 수 있는 명령어 비율은 15%~20% 사이가 적절함을 알 수 있다. 비록 ELF 실행파일이 아닌 파일이 실행파일로 판단되었다 하더라도 그 오류율은 10%를 넘지 않을 것이며, 다음 장의 명령어 시퀀스에 의한 방법으로 그 오류가 줄어들 것으로 판단된다.

IV. 명령어 시퀀스의 유사도 비교를 통한 악성 파일 검출

HMM은 순차적 특성을 갖는 데이터에 대한 정보화에 적합한 방법이다. 임의의 파일이 악의적인 실행파일임을 판단하기 위해서는 어셈블리 명령어 수준에서 분석이 이루어져야 한다. 그리고 일반 실행파일과 비교하여 악성인지 아닌지를 판단하기 위해서는 특징을 추출하는 방식이나 학습을 통한 프로파일링 방법이 적용될 수 있다. 먼저 바이러스 백신에 자주 사용되는 특징 추출 방식은 악성 실행파일에서 특정 문자열을 사용하지 않는 한 알 수 없게 된다.

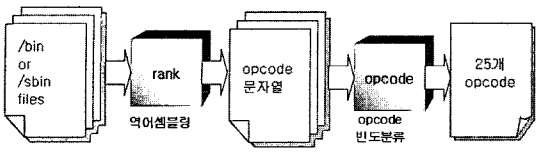
악성 실행파일에 대한 판단을 위해 프로파일링 방법을 사용하기 위해서는 여러 가지 문제점이 존재한다. 정상 실행파일에 대한 프로파일링 방법은 알려지지 않은 악성 실행파일을 탐지할 가능성은 존재하나 시스템 내에는 수많은 정상 실행파일이 존재하므로 적절한 학습 조건을 찾기가 쉽지 않다. 반면에, 악성 실행파일에 대한 프로파일링 방법은 특징 추출 방식과 유사하게 정해진 악성 실행파일에서 순차적 특징을 추출할 수 있다.

4.1 어셈블리 명령어 시퀀스의 프로파일링

HMM을 이용하여 명령어 시퀀스의 유사도를 검사하기 위해서는 HMM을 통한 프로파일링 과정이 필요하다. 시스템에는 정상적인 명령어가 많고 그 특성도 다양하기 때문에, 일반 명령어에 대한 학습은 어렵다고 판단된다. 그러나 공격 프로그램은 그 특성에 따라 어느 정도 분류되어 있다. 이렇게 분류된 공격 프로그램에 대한 학습은 그 프로그램들의 특성을 찾아내는데 효과적이 될 것으로 예상된다. 본 논문에서는 버퍼오버플로우 공격 프로그램 몇 개를 선별하여 HMM을 통해 학습을 시키고자 한다. 이렇게 학습된 HMM 데이터는 실행파일이 버퍼오버플로우 공격 프로그램인지 아닌지를 판별하는 중요한 요소가 될 것이다.

HMM 학습을 하기 위해 사용되는 시퀀스 데이터는 어셈블리 명령어이다. 그런데, 어셈블리 명령어의 수는 너무 많아서 그 명령어를 모두 학습 아이템으로 사용할 경우 HMM 학습 부하가 너무 클 것으로 예상된다. 이에 따라 HMM 학습 이전에 어셈블리 명령어에 대한 분류작업이 필요하게 되었다. 또

한, 파일 조각에서 어셈블리 명령어를 추출해야 하므로 파일 조각에 대한 역어셈블링 작업이 필요하다.



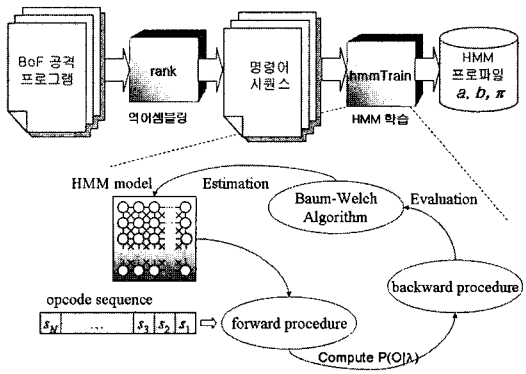
〔그림 6〕 명령어 빈도 분류 과정

실험 환경에서 첫 번째 분류하는 것은 사용되는 어셈블리 명령어의 빈도수를 랭크(rank) 순위로 처리하는 과정이다. 이 과정에서 많은 프로그램에서 사용하는 명령어를 빈도별로 구성하여 데이터베이스화 한다. HMM의 학습부하를 줄이고 어셈블리 명령어 시퀀스의 특징을 정확히 분류하기 위해 어셈블리 명령어의 빈도수에 따라 HMM에서 사용되는 명령어 코드(opcode)를 선택한다. 오류율은 다소 존재하지만 랭크 작업은 다양한 어셈블리 명령어를 빈도에 따라 선택하여 대표 명령어 코드를 찾기 위한 것이다. 〔그림 6〕은 명령어 빈도 분류를 위한 랭크과정을 보여준다.

실행코드를 분류하기 전에 데이터의 역어셈블링 작업이 선행되어야 한다. 이를 위해 데이터를 일정한 스트림 형태에서 입력이 가능한 포맷으로 변환을 해야 하며 25개 단위의 입력버퍼로 설계하여 크기단위를 조절하였다. 실제 입력버퍼 사이즈는 가변적이므로 측량 시간을 위해서 명령어 코드의 기본 지향 사이즈인 25개를 기준으로 구현하였다. 명령어 코드 개수를 30개부터 5씩 증가시켜보았지만 실험데이터에는 큰 변화가 없고 측정시간만 지연되어, 25를 명령어 코드 입력단위 크기로 설정하였다. 따라서 실험에 사용되는 패턴 유사도를 위하여 메모리 최소크기는 25개 단위로 사용하였다.

본 논문에서는 랭크과정을 통하여 사용 빈도수가 높은 24개를 측정대상으로 선택하였다. 선택된 24개의 어셈블리 명령어는 AAA, ADD, AND, CALL, CMP, DEC, IMUL, INC, JC, JMP, JNC, JNZ, JZ, LEA, MOV, OR, OUTSB, POP, POPAD, PUSH, RET, SUB, TEST, 그리고 XOR이다. 이 외의 빈도수가 낮은 어셈블리 명령어들은 25번째 명령어로 묶어서 예외로 처리하도록 한다. 아래는 /sbin 아래의 파일에 대한 명령어 빈도수를 검사하고 25개를 선택하는 과정을 보여준다.

```
[ryu@wargame test]$ ./rank
Usage: ./rank dir_name size_of_M out_filename
dir_name: Directory name that contains files
to be read
size_of_M: Size of M
[ryu@wargame test]$ ./rank /sbin 25
output filename : rank_idx.sbin.25
Working: /sbin is directory
Working: FileInfo : /sbin/rdump, fd:4, size:643552
Working: filename : /sbin/rdump
...
```

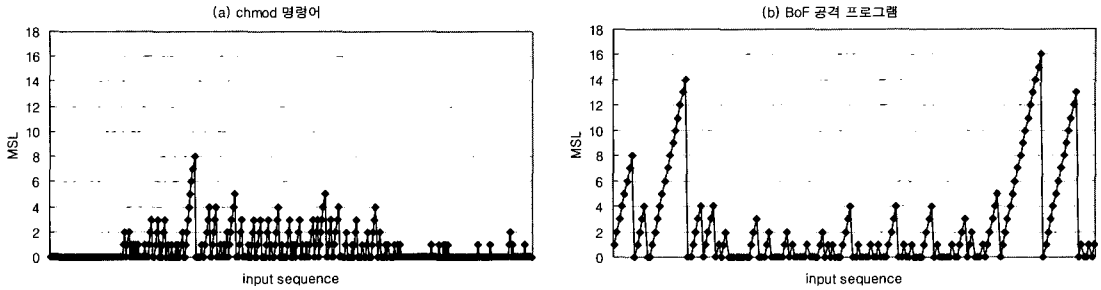


〔그림 7〕 HMM 프로파일링

본 논문은 〔그림 7〕과 같은 절차에 따라 명령어 시퀀스에 대한 프로파일링을 수행한다. 삭제된 파일에서 추출한 명령어 시퀀스(opcode sequence)는 HMM 학습 절차에 따라 포워드 프로시저(forward procedure)와 백워드 프로시저(backward procedure)를 거쳐 나온 HMM 구성 확률(π, A, B)에 Baum-Welch 알고리즘으로 재평가 하는 과정을 반복함으로써 프로파일링이 수행된다. 이렇게 생성된 최종 HMM 구성 확률은 추후에 사용될 HMM 프로파일로 확정된다.

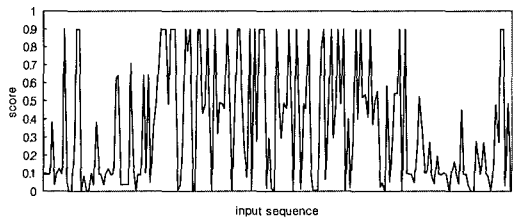
4.2 어셈블리 명령어 시퀀스의 유사도 비교

명령어 시퀀스를 프로파일링한 HMM 모델에 데이터를 적용하여 악성 파일 여부를 판단하기 위해서는 적절한 임계값의 설정과 판단 척도가 정의되어야 한다. HMM 모델에 시퀀스 데이터를 적용하여 나온 평가값(score)은 0에서 1사이의 확률값으로 유사도를 나타낸다. 본 논문에서는 평가값이 0에 가까울수록 프로파일링된 모델에 근사하게 나타나도록 하고 있다. 그런데, HMM은 확률값을 기반으로 하고 있어서 많은 학습시간을 필요로 하고 학습된 모델이라



〔그림 9〕 일반 프로그램과 공격 프로그램의 MSL 차이 비교

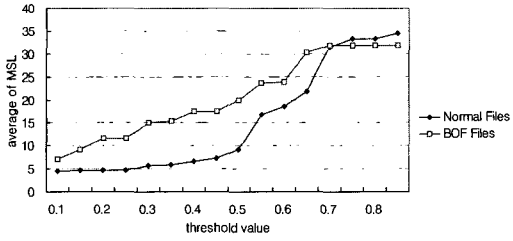
도 오류 가능성을 많이 포함하고 있다. 〔그림 8〕은 학습에 포함된 데이터셋 중 하나에 대한 유사도를 비교해 본 것이다. 〔그림 8〕에서 볼 수 있듯이 명령어 번호를 입력으로 명령어 시퀀스에 따른 평가값이 1에 가까운 경우가 상당히 많음을 알 수 있다. 즉, 평가값이나 평균값으로 유사도를 판정하는 것은 정확성이 떨어질 수밖에 없다.



〔그림 8〕 명령어 시퀀스에 대한 유사도 평가값 비교

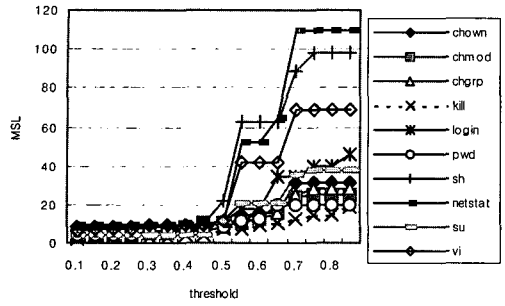
학습용 데이터가 유사한 특성을 가지고 있다면, 그 사이에는 공통된 시퀀스 특성이 존재할 것이다. 이러한 특성은 HMM 학습과정에서 프로파일링 모델이 생성되는데 중요한 역할을 할 것이다. 이러한 특성이 존재하는 부분을 비교한다면 정확성이 높은 판정결과를 얻을 수 있을 것으로 예상된다. 본 논문에서는 단일 평가값보다 연속된 평가값이 신뢰도가 더 높다고 판단하여, 유사하다고 판정되는 평가값이 연속된 값으로 출력될 때 판정에 적용하도록 하였다. 즉, 평가값이 0에 가까운 경우가 연속적으로 나오면 그 횟수를 일치된 시퀀스 길이(MSL, Matched Sequence Length)로 저장한다. 그리고 이 길이가 일정한 값 이상이면 프로파일링된 데이터와 유사하다고 판정한다.

〔그림 9〕는 명령어 시퀀스에 대한 HMM 평가값을 기반으로 임계값보다 큰 경우 MSL 값을 증가시켜 나온 결과 그래프이며, chmod 명령어와 BoF (버퍼오버플로우) 공격 프로그램을 실험대상 파일로 하였다. 〔그림 9〕(b)에서 처음부분에 MSL이 14인 부분이 있고 중간에 작은 값의 MSL로 채워지며 마지막 부분에 MSL이 16인 부분이 있다. 〔그림 9〕(a)와 〔그림 9〕(b)는 눈으로 보더라도 확연히 다르게 나타난다. 이러한 결과에 의하여 MSL로 악성 프로그램을 구분하는 것이 가능함을 알 수 있다.



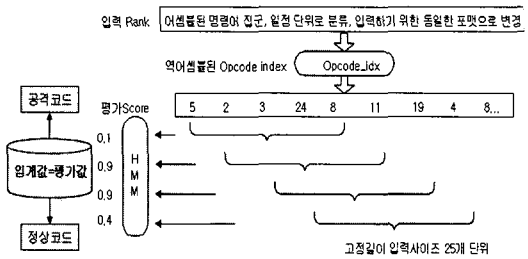
〔그림 10〕 MSL의 평균값 비교

MSL이 악성프로그램을 구분할 수 있는 척도가 될 수 있는지를 확인하고 적절한 임계값을 찾기 위하여, /bin 아래에 있는 파일들에 대한 MSL과 BoF 공격 프로그램들에 대한 MSL을 여러번 테스트하여 비교하여 보았다. 〔그림 10〕은 정상적인 파일들과 BoF 공격 파일들에 대하여 임계값을 변경하면서 MSL을 측정한 결과이다. 임계값이 0.7 이상에서는 오히려 정상파일의 MSL이 더 커지는데, 이것은 실행 파일의 크기에 영향을 받기 때문이다. 두 종류의 파일에 대한 MSL 평균값의 차이가 가장 많은 부분은 임계값이 0.30 ~ 0.55로 보이며, 0.4 정도에서 MSL 값이 가장 차이가 남을 알 수 있다.



〔그림 11〕 임계값에 따른 정상파일의 MSL 변화

〔그림 11〕에서는 /bin 디렉터리 아래의 정상파일 10개에 대한 MSL 값의 변화를 보여준다. 그림 11에서도 임계값이 0.45 정도까지는 MSL 변화가 거의 없다가 0.5일 때부터 급격히 증가함을 알 수 있다. 즉 정상적인 파일이라도 임계값이 0.5이상이 되면 악성파일과 구분하기 어려울 정도의 결과가 나타날 수 있음을 보여준다. 이에 따라 MSL 값을 구하기 위한 평가값에 대한 적절한 임계값은 0.4로 판명된다. 악성 프로그램인지 아닌지를 판정하기 위한 적절한 MSL 값은 10 ~ 15로 보이지만, 임계값이 0.3일 때부터 구분 가능한 10으로 선택하여 사용하면 오류가 가장 적을 것으로 예측된다.



〔그림 12〕 악의적 코드 분류 유사도 추출 과정

4.3 악의적인 실행파일에 대한 탐지 성능

본 논문에서는 악의적 코드를 감지하기 위해 HMM 기법을 이용하였다. 악의적 파일은 크게 BoF(버퍼 오버플로우)와 같은 공격기법을 주목표로 하여 프로파일링 작업을 수행하였다. 기본적인 프로파일링 모델은 BoF로서 시스템상의 메모리 번지를 추측하거나 메모리 덤프를 시도하는 코드에 역어셈블링 작업을 수행하여 모델링 하였다. 〔그림 12〕는 악의적 코드를 판정하기 위한 유사도 추출 관련 구현 시스템을 프로세스 흐름으로 표현한 것이다.

〔표 2〕 MSL에 따른 악성 실행파일 탐지에 대한 FP와 TP

MSL / 임계값	8		10		12		14	
	FP	TP	FP	TP	FP	TP	FP	TP
0.10	0.18	0.22	0.00	0.11	0.00	0.11	0.00	0.00
0.15	0.18	0.67	0.00	0.33	0.00	0.11	0.00	0.11
0.20	0.22	1.00	0.00	0.78	0.00	0.44	0.00	0.33
0.25	0.22	1.00	0.00	0.78	0.00	0.44	0.00	0.33
0.30	0.36	1.00	0.00	1.00	0.00	0.89	0.00	0.78
0.35	0.37	1.00	0.00	1.00	0.00	0.89	0.00	0.78
0.40	0.46	1.00	0.00	1.00	0.00	0.89	0.00	0.89
0.45	0.55	1.00	0.20	1.00	0.12	0.89	0.08	0.89
0.50	0.82	1.00	0.48	1.00	0.29	1.00	0.20	1.00
0.55	0.82	1.00	0.82	1.00	0.64	1.00	0.55	1.00
0.60	0.88	1.00	0.91	1.00	0.91	1.00	0.73	1.00
0.65	0.98	1.00	0.98	1.00	0.91	1.00	0.88	1.00
0.70	1.00	1.00	1.00	1.00	0.98	1.00	0.93	1.00
0.75	1.00	1.00	1.00	1.00	1.00	1.00	0.98	1.00
0.80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.85	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

본 논문의 실험에 사용된 파일로는 /bin 디렉터리의 97개 정상파일, 9개의 BoF 공격 프로그램이다. 실험에 사용한 BoF 공격 프로그램은 스택 오버플로우 공격이 5개, 힙 오버플로우 공격이 2개, 그리고 포맷스트링 공격이 2개로 해킹 테스트에 일반적으로 사용되는 코드이다. 실험에서 악성코드에 대한 학습을 위해서 BoF 공격 프로그램에서 7개를 임의로 선택하여 학습 데이터로 사용하였다. 이렇게 학습된 HMM 프로파일을 이용하여 MSL 값을 변경시키며 탐지 성능을 측정하여 보았다. [표 2]는 MSL 값을 8, 10, 12, 14로 적용하여 악성파일 탐지의 정확도를 FP(False Positive)와 TP(True Positive)로 표현한 것이다. TP가 1에 가까울수록 탐지율이 높고, FP가 0에 가까울수록 오탐율이 줄어든다. 이전 실험 결과에서 나타난 것처럼 임계값이 0.3~0.4이고 MSL가 10일 때 정확성이 가장 높게 나타난다([표 2]의 음영된 부분 참조). 이러한 결과를 바탕으로 HMM을 이용하여 명령어 시퀀스 유사도를 측정하는 방식으로 조각난 파일이 악성파일인지를 검사하는 방법이 효과적임을 알 수 있다.

V. 결 론

본 논문에서는 컴퓨터 포렌식스 조사관이 잠재적인 증거의 손실 없이 파일시스템을 검사하는 방법과 ELF 실행파일의 주요 구성요소인 어셈블리 명령어 코드의 특성을 이용하여 이진 파일 조각이 실행파일의 일부분인지를 판단하고, 검출된 조각 파일에서 명령어 부분을 추출하여 악의적인 목적의 프로그램의 일부인지를 판단하는 방법을 제시하였다. 컴퓨터 범죄 수사를 위하여 반드시 필요한 삭제된 데이터나 이진 변형된 실행파일의 영역에서 잠재적인 증거들을 손실 없이 수집하기 위해서는 무엇보다도 신뢰할 수 있는 원본 데이터의 보존이 필수적이다. 그러나 신뢰할 수 있는 원본 데이터의 수집과 보존 절차에도 불구하고 현재 조사 대상에서 제외되어 있는 삭제 및 변형된 이진 실행파일을 체계적으로 검사하기 위한 방법을 제안하였다.

어셈블리 명령어 코드가 유사하다는 것을 판정하는 방법으로 일부 코드를 직접 비교하는 방법을 생각할 수 있다. 이러한 방법은 두 코드가 같다는 것을 판정하는 직접적인 방법이다. 그러나 명령어 코드는 컴파일 환경이나 방법에 따라 달라지고, 공격자가 소스를 일부 수정한 경우도 많아 실제로 적용하기는 어렵다. 직접적인 증명이 어렵다면, 간접적으로 통계적인 방법을 사용하거나 인공지능적인 방법을 사용할 수 있다. 본 논문에서는 어셈블리 명령어 코드의 비중을 검사하는 방법과 어셈블리 명령어 시퀀스의 특성 비교 방법을 제안하였다.

본 논문에서는 수집된 파일 조각이 악성 실행파일인지를 검사하기 위해서 두 단계 절차를 수행한다. 먼저 파일 조각 내에 명령어 크기를 구해 누적한 후 비율을 검사하여 그 조각이 실행파일인지를 검사한다. 실험을 통해서 명령어 식별자 사이의 구간거리를 4로 설정하고 실행파일 여부를 판단하는 기준으로 임계 비율을 15%~25%로 설정하는 것이 정확도가 높다는 것을 알 수 있었다. 이 결과에 따라 실행파일로 판정이 되면, 악성 프로그램에 대한 HMM 프로파일에 적용하여 그 평가값으로 악성 프로그램 여부를 판정한다. HMM 학습을 수행하기 위한 입력 데이터로 악성 파일의 명령어 시퀀스를 사용했으며, HMM의 포워드-백워드 프로시저와 Baum-Welch 알고리즘을 사용하여 학습을 진행하였다. 이 결과로 구해진 악성 파일에 대한 HMM 프로파일을 이용하여 정상 파일과 악성 파일에 대한 성능 테스트를 수행하였다. 본 논문에서 사용한 악성 파일은 BoF 공격 프로그램이다.

사용한 명령어 시퀀스 유사도 검사 방법으로 HMM 평가값을 직접 구하는 방법을 취하지 않았다. 본 논문에서 사용한 방법은 HMM 평가값이 연속으로 악성파일과 일치한다는 판단을 내릴 때 MSL을 증가시키고, MSL이 일정한 값보다 커질 때 악성파일이라고 판정하는 것이다. 이러한 판단의 정확성을 높이기 위해 여러 파일을 대상으로 실험한 결과, HMM 평가값에 대한 임계값을 0.3 ~ 0.4로 설정하고 MSL 판단 기준을 10으로 하였을 때 가장 성능이 좋음을 알 수 있었다.

제안하는 악의적 코드 분류 방법은 삭제되거나 손실 데이터에서 실행 파일의 판단유무를 좌우하는 표준 모델을 제작하여 악의적 데이터를 선별해 내는 핵심 기술로 사용가능하며, 손상된 데이터나 혹은 물리적 매체에서 증거 불충분 될 제약조건에서 효과적인 모델링도구로 사용될 수 있을 것으로 판단된다. 따라서 본 논문에서 제시된 방법을 통하여 해커의 침입이후 잠복하고 있을 파일이나 백도어 등을 파악하고 해커의 증거인멸 이후 복원하는 과정에서의 부분 데이터를 분석하여 침입 증거로 제시할 수 있을

것으로 판단된다. 또한 최신 기법으로 응용하는 다양한 기법의 모델링 과정을 통하여 유사도 추출 과정을 광범위하게 설계하여 자동적으로 분석해 내는 과정을 추가한다면 완전한 데이터 복구가 어려운 상황에서 부분적 데이터만으로도 충분히 악의적 파일이나 실행 파일의 유무를 확인할 수 있는 계기가 될 것으로 전망된다.

참 고 문 헌

[1] B. Carrier. "Defining Digital Forensics Examination and Analysis Tools," *In Digital Research Workshop II*, 2002.

[2] 이석봉, 박준형, 김민수, 노봉남, "컴퓨터 포렌식스 관점에서 파일 지스러기 영역의 활용방법 연구," *한국정보과학회 학술발표논문집*, pp. 859-861, Oct. 2003.

[3] D. Farmer and W. Venema, "Forensic Computer Analysis: An Introduction," *Dr. Dobbs Journal*, Jul. 2001.

[4] B. Carrier, *The Sleuth Kit, http://www.sleuthkit.org/sleuthkit*, 2006.

[5] L. Garber, "EnCase: A Case Study in Computer-Forensic Technology," *IEEE Computer Magazine*, Jan., 2001.

[6] B. Carrier. "Performing an autopsy examination on FFS and EXT2FS partition images: An Introduction to TCTUTILs and the Autopsy Forensic Browser," *In Digital Research Workshop II*, 2002.

[7] W. Venema, "File Discovery Techniques," *Dr. Dobbs Journal*, Jul. 2001.

[8] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," *Proc. of IEEE Symposium on Security and Privacy*, pp. 38-49, 2001.

[9] J. Park, M. Kim, B. Noh, J. Joshi, "A Similarity based Technique for Detecting Malicious Executable files for Computer Forensics," *IEEE International Conf. on IRI*, Sep. 2006.

[10] *libdisasm x86 Disassembler Library, http://bastard.sourceforge.net/libdisasm.html.*

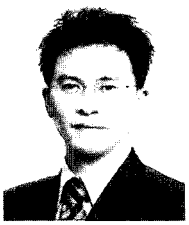
[11] K. McLaughlin, "Hacker Sophistication Outpacing Forensics," *Dr. Dobbs Journel*, Aug. 2006.

[12] 최종호, 조성배 "침입탐지 시스템을 위한 은닉 마르코프 모델의 적용," *한국정보과학회논문지*, 28(6), pp. 429-438, 2001

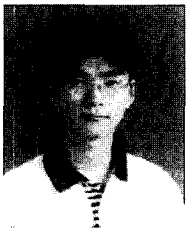
[13] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[14] Intel Corporation, *Intel® 64 and IA-32 Architecture Software Developer's Manual, http://www.intel.com/design/pentium4/manuals/index_new.htm*, 2006.

〈著者紹介〉



류 동 주 (Dongju Ryu) 정회원
1999년 2월: 광주대학교 컴퓨터학과 졸업
2002년 2월: 광주대학교 정보통신공학과 석사
2005년 2월: 전남대학교 정보보호협동과정 박사수료
2006년 현재 (주)IT Bank 멀티캠퍼스 보안 컨설턴트
<관심분야> 네트워크 보안, 차세대네트워크보안, 보안 키 알고리즘, 사이버 포렌식



이 석 봉 (SukBong Lee)
2001년 2월: 전남대학교 기계공학과 졸업
2004년 2월: 전남대학교 정보보호협동과정 석사
2006년 3월~현재: 전남대학교 정보보호협동과정 박사과정
<관심분야> DRM, 정보보안, Privacy Enhanced Technology

김 민 수 (Minsoo Kim) 종신회원
1993년 2월: 전남대학교 전산통계학과 졸업
1995년 2월: 전남대학교 전산통계학과 석사
2000년 2월: 전남대학교 전산통계학과 박사
2000년 3월~2001년 2월: 한국정보보호진흥원 선임연구원
2001년 3월~2005년 2월: 전남대학교 리눅스보안연구센터 연구교수
2005년 3월~현재: 목포대학교 정보공학부 정보보호전공 전임강사
<관심분야> 컴퓨터 포렌식스, 운영체제 보안, 침입탐지시스템, 데이터마이닝