ISSN(Print): 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.1.043

A Classification Method for Executable Files based on Comparison of Undocumented Information in the PE Header

Jung-Sun Kim† · Jung-Min Kang†† · Kang-San Kim†† · Wook Shin††

ABSTRACT

File identification and analysis is an important process of computer forensics, since the process determines which subjects are necessary to be collected and analyzed as digital evidence. An efficient file classification aids in the file identification, especially in case of copyright infringement where we often have huge amounts of files. A lot of file classification methods have been proposed by far, but they have mostly focused on classifying malicious behaviors based on known information. In copyright infringement cases, we need a different approach since our subject includes not only malicious codes, but also vast number of normal files. In this paper, we propose an efficient file classification method that relies on undocumented information in the header of the PE format files. Out method is useful in copyright infringement cases, being applied to any sort of PE format executable file whether the file is malicious, packed, mutated, transformed, virtualized, obfuscated, or not.

Keywords : PE(Portable Executable), File Identification, File Classification, Computer Forensics, Malicious Codes

실행파일 헤더내 문서화되지 않은 정보의 비교를 통한 실행파일 분류 방법

김 정 순† · 강 정 민†† · 김 강 산†† · 신 욱††

요 약

파일 식별과 분석은 컴퓨터 포렌식 수사과정에서 디지털증거 획득 및 증거분석에 중요한 요소이며 지금까지 많은 연구가 진행되었다. 그러나 실행파일의 식별과 분석은 주로 악성코드에 대해 연구되어 왔기 때문에, 저작권침해 사고와 같은 일반적인 실행파일을 세부적으로 분류하고 탐지해야 할 경우에는 기존의 악성코드 분류 방법은 적용되기 어렵다. 따라서, 본 논문에서는 실행파일 헤더내 문서화되지 않은 정보의 유사도 측정에 근거한 비교를 통해 실행파일을 세부적으로 분류할 수 있는 방법을 제시한다. 제안한 방법은 실행파일의 헤더에 포함된 정보를 이용하기 때문에 일반적인 실행파일뿐만 아니라 기존의 악성코드 및 새로운 악성코드와 변종 그리고 실행압축, 코드변형, 가상화 및 난독화된 실행파일 분류에도 활용이 가능하다.

키워드 : 실행파일, 파일 식별, 파일 분류, 컴퓨터 포렌식, 악성코드

1. 서 론

최근 IT 기술 발전에 따른 정보화 사회의 부작용으로 디지털 범죄가 증가하고 있으며, 컴퓨터 관련 범죄뿐만 아니라 일반 범죄에 대한 중요 증거 또는 단서를 컴퓨터와 스마트폰과 같은 디지털 정보기기 내에 보관하는 경우가 많아지고 있기 때문에 컴퓨터 포렌식 연구에 대한 중요성이 커지고 있다. 특히, 파일 식별과 분석은 컴퓨터 포렌식 수사과정에서 디지털 증거 획득 및 증거분석에 필요한 중요한 요소 중에 하나이다. 그러나 파일 식별과 분석에 대한 연구는 문서파일이나 이미지, 동영상 등 데이터 파일의 식별과 분석에 집중되어 왔으며, 상대적으로 실행파일에 대한 연구는 미진하고 대부분 악성코드의 탐지와 분류에 대한 연구가 집중되어 왔다. 그리고 기존의 악성코드 연구에 사용되었던 행위 모니터링을 통한 API콜 추적이나 악성행위 패턴에 의한 방법 그리고 시그니처 기반의 악성코드 탐지방법 등은 악성코드를 탐지하고 분류할 수 있지만 컴퓨터 포렌식 수사 목적으로 정상행위를 하는 실행파일을 분류하거나 식별하는 데 적용되기 어렵다. 또한, 소프트웨어 보호 목적으로 게임

† 정 회 원 : 한국전자통신연구원 부설연구소 연구원
†† 정 회 원 : 한국전자통신연구원 부설연구소 선임연구원
논문접수 : 2012년 9월 7일
수 정 일 : 1차 2012년 11월 15일, 2차 2012년 11월 19일
심사완료 : 2012년 11월 19일
* Corresponding Author : Jung-Sun Kim(cyberpia@ensec.re.kr)

프로그램이나 은행 업무 관련 프로그램 등 많은 정상적인 실행파일에 패킹과 난독화 기술을 적용하고 있으므로, 패킹과 난독화 기술에 취약한 기존의 악성코드 탐지 기법들을 사용하기에는 문제점이 있다.

따라서 본 논문에서는 실행파일 헤더내에 문서화되지 않은 정보의 유사도 비교를 통해 실행파일을 세부적으로 분류하는 방법을 제시하고, 제안한 방법으로 악성코드를 포함한 실행파일을 분류하고 식별할 수 있음을 보인다. 또한, 실험을 통해서 패킹과 난독화 기술이 적용된 실행파일에 대해서도 제안한 방법이 적용될 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 악성코드 탐지 및 분류에 관련된 기존 연구들을 소개하고, 3장에서는 제안한 방법을 기술한다. 그리고 4장에서는 제안한 방법의 분류 실험결과를 제시하고, 마지막 5장에서 결론을 내린다.

2. 관련 연구

실행파일의 분류 방법에 대한 연구는 주로 악성코드를 탐지하고 분류하기 위한 목적으로 진행되었다. 악성코드를 탐지하기 위해서는 기존에 발견된 악성코드의 실행파일을 실행하지 않고 디버거와 디스어셈블러를 이용하여 분석하는 정적 분석 방법과 악성코드를 가상머신과 같은 제한된 환경에서 실행시키고 악성코드의 행위를 분석하는 동적 분석 방법이 있다.

악성코드의 정적 분석 방법에는 플로우 차트 또는 플로우 그래프를 이용하여 악성코드를 분석하는 방법[5,9], 악성코드의 시스템콜 및 라이브러리 함수 호출을 기반으로 특정 악성 행위를 나타낼 수 있는 패턴을 생성하는 방법과 유사도 계산을 통한 코드 패턴 매칭 방법[4], ClamAV[19]를 확장하여 악성코드를 탐지하는 방법[10], 명령어 빈도수를 이용한 방법[3], 함수의 길이를 이용한 방법[6], 함수 호출 그래프를 이용한 방법[8], 바이트 레벨 파일 컨텐츠 분석 방법[7], API 조합을 이용한 윈도우 악성행위 분류 기법[11] 등이 제안되었다.

악성코드의 동적 분석 방법에는 실행 압축 우회기법을 이용한 악성코드에 대하여 행위 기반의 탐지 패턴을 생성하고 악성코드를 탐지하는 방법[2], 그래프 마이닝과 개념 분석을 이용하여 특정 악성 행위에 대한 특징들을 자동으로 추출하는 방법[14], 에뮬레이터를 기반으로 악성코드의 행위를 모니터링[12]하여 악성코드의 API콜을 추적하고 통계적 방법으로 분류하는 방법[13], 테인팅(Tainting) 기법을 이용한 분석 방법[1], Pin을 이용한 분석 방법[18] 등이 제안되었다. 그리고 정적방법과 동적방법을 혼합형태의 악성코드 탐지방법으로 문자열과 API의 순차적 특징을 이용한 악성코드 변종 분류 기법도 있다[16-17]. 기존에는 악성코드의 수집 및 분석에 수동적인 요소가 대부분이었으며, 이를 위해 시간 및 비용이 많이 소요되는 단점이 존재하였다. 이를 극복하기 위해 최근에는 악성코드를 자동으로 수집하고 분석하는 기술과 시스템이 지속적으로 연구되고 개발되는 추세이다[15].

그러나 기존의 악성코드 분석 및 탐지에 사용되었던 기법들은 악성코드 탐지에 특화된 방법들이므로 정상적인 실행파일의 세부 분류가 필요한 분야에는 사용되지 못한다. 물론, 정적분석이나 동적분석 방법을 사용하여 정상적인 실행파일을 분석한 다음 정상적인 실행파일에 대한 패턴을 생성하여 세부 분류를 할 수 있지만 시간 및 비용이 많이 소요된다. 소프트웨어 저작권 침해 여부 조사와 같은 컴퓨터 포렌식 수사의 경우 정상적인 실행파일들로부터 해당 실행파일이 어떤 종류의 프로그램인지 식별하고, 어떤 버전의 실행파일인지 또는 어떤 종류의 라이브러리들이 사용되어 작성되었는지 등에 대한 세부적인 분류에 대한 연구와 삭제된 저장매체에서 복구된 정보로부터 어떤 종류의 실행파일들이 있었는지 식별하고 분류하는 연구가 필요하다. 그리고 저작권 보호 및 프로그램 알고리즘 보호를 위해 사용되는 패킹, 코드변환, 가상화 등의 난독화 기법이 적용된 실행파일을 분석할 경우 기존의 방법들은 많은 노력과 시간이 필요하므로, 난독화된 경우의 실행파일에 대해서 효과적으로 식별하고 분류할 수 있는 연구가 필요하다. 또한, 파일 식별과 분류에 관한 연구는 특정 실행파일들의 가계도와 같은 정보를 분석할 수 있는 방법과 유사한 다른 변종의 프로그램들과의 상호 연관성을 분석할 수 있는 방법을 포함해야 한다.

3. 제안하는 방법

3.1 개요

본 논문에서는 윈도우 실행파일 포맷인 PE(Portable Executable) 헤더에 포함된 문서화되지 않은 정보를 이용하여 실행파일을 분류하는 방법을 제안한다. 제안한 방법은 Fig. 1과 같이 3 과정으로 구성된다. 첫 번째 과정은 윈도우 실행파일 헤더에서 유사도 계산에 사용될 정보를 추출하는 단계이고, 두 번째 과정은, 추출된 정보를 이용하여 유사도를 계산하는 단계이다. 마지막 세 번째 과정은 계산된 유사도를 이용하여 실행파일을 분류하는 단계이다.

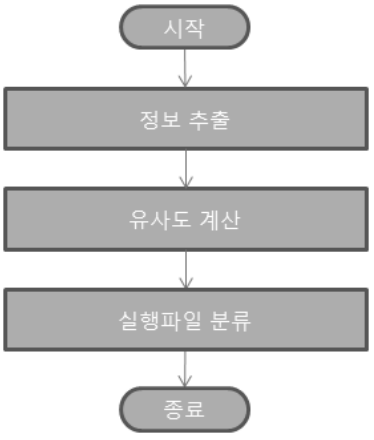

Fig. 1. The flowchart of the proposed method

3.2 정보 추출

본 논문에서 유사도 비교를 위해 사용되는 정보는 Daniel Pistelli에 의해 알려진 리치 시그니처(Rich Signature)이다. 리치 시그니처는 Microsoft Visual Studio와 같은 마이크로소프트사의 프로그램 개발도구를 사용하여 실행파일을 생성할 때 링커가 해당 실행파일의 헤더부분에 추가하는 정보이며, 마이크로소프트사에서는 리치 시그니처에 대한 정보를 공식적으로 제공하지 않는다[20].

1) 리치 시그니처 분석

리치 시그니처는 PE 헤더내의 Dos Stub와 NT Header 사이에 존재하며 리치 시그니처의 크기는 실행파일에 포함된 정적 라이브러리의 수에 따라 달라진다. Fig. 2는 한글 2010 실행파일 hwp.exe의 리치 시그니처 위치 예를 보여준다. 리치 시그니처는 각각의 라이브러리 정보를 저장하는 레코드들로 구성된다. 라이브러리 정보를 저장하는 레코드는 8바이트 크기이며, 상위 4바이트는 컴파일러 정보를 의미하고 하위 4바이트는 링커가 해당 라이브러리를 참조한 횟수를 의미한다. 리치 시그니처의 시작을 의미하는 문자열은 "DanS"이고, 끝을 의미하는 문자열은 "Rich"이다. 그리고 "Rich" 문자열 다음 4바이트는 마스크값을 의미한다. "DanS" 다음 12바이트는 패딩값이고, 라이브러리 정보가 저장되는 레코드들의 시작위치는 리치 시그니처의 시작주소부터 0x10 바이트 다음 위치이다. 또한, 리치 시그니처는 "Rich"를 제외한 전체 정보가 "Rich" 다음의 마스크값으로 XOR 인코딩되어 있다. 정보 추출 과정은 실행파일에서 인코딩된 리치 시그니처 정보와 마스크값을 추출하며, 마스크값을 이용하여 인코딩된 리치 시그니처를 디코딩하고 리치 시그니처의 유효성을 검사한다. 리치 시그니처의 유효성 검사는 디코딩된 리치 시그니처의 처음 4바이트 값이 "DanS" 문자열인지 여부로 확인한다.

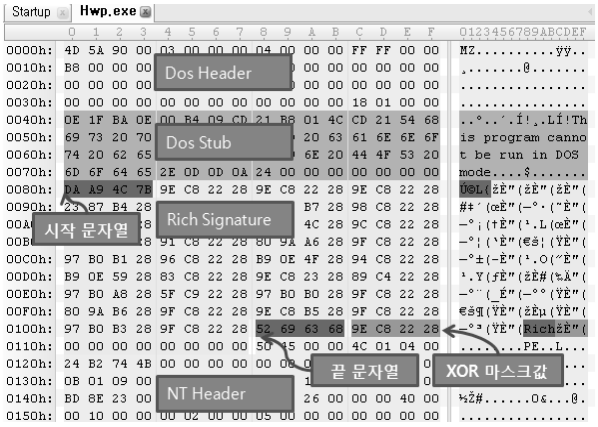

Fig. 2. The offset of the Rich Signature in PE Header

2) 리치 시그니처 추출 알고리즘

리치 시그니처 추출 알고리즘은 다음과 같다. 실행파일명을 입력값으로 받아서 실행파일 헤더내에 리치 시그니처가 있는지 검사한다. 리치 시그니처가 있는 경우는 인코딩된 리치 시그니처의 데이터를 XOR 마스크값으로 디코딩하여 리치 시그니처의 유효성을 검사한 다음 디코딩된 리치 시그니처를 반환한다[20].

알고리즘명: GetRichSignature
입력: 실행파일명
출력: 디코드된 리치 시그니처

1. 실행파일의 핸들을 가져온다.
2. 실행파일에서 Dos Header와 NT header의 정보를 얻어온다.
3. Dos Stub 다음 위치부터 NT header앞까지의 정보를 추출하여 리치 시그니처가 있는지 검사한다. ("Rich" 문자열이 있는지 검사한다.)
4. 리치 시그니처를 발견하지 못하면 리치 시그니처가 없다는 에러를 반환한다.
5. 리치 시그니처가 있는 경우 XOR 마스크 값을 추출한다.
6. 추출된 XOR 마스크 값을 이용하여 리치 시그니처를 디코딩한다. Dos Stub다음 위치(0x80)부터 NT Header 앞 "Rich" 문자열까지 XOR 마스크값을 이용하여 4바이트씩 XOR하여 디코딩한다.
7. 디코딩된 처음 4바이트 값이 "DanS"인지 검사한다.
8. "DanS"가 아니면 잘못된 리치 시그니처 형식이므로 리치 시그니처 형식 에러 값을 반환한다.
9. "DanS"인 경우는 정상적인 리치 시그니처이므로 디코딩된 리치 시그니처를 반환한다.

추출 알고리즘을 통해 디코딩된 리치 시그니처의 예는 Fig. 3과 같다. 디코딩된 리치 시그니처는 시작 문자열 "DanS"과 끝 문자열 "Rich"가 있으므로 추출된 결과는 유효한 리치 시그니처임을 확인할 수 있으며, 0으로 패딩된 12바이트와 총 15개의 레코드가 포함되어 있음을 확인할 수 있다. 실행파일에 저장된 값은 인텔계열 중앙처리장치에서 리틀 엔디안 방식으로 변환되어 계산된다. 즉, 문자열 "Rich"값은 0x68636952이고 문자열 "DanS" 값은 0x536E6144이다.

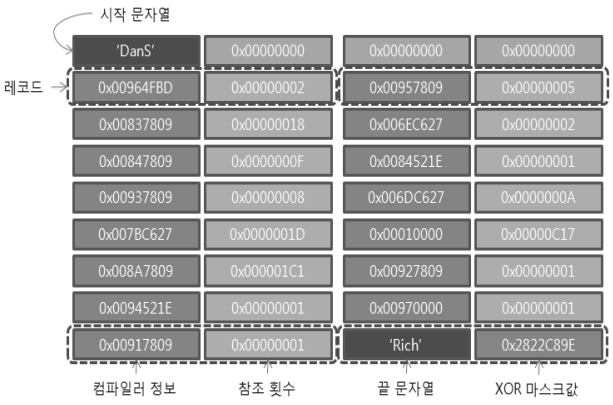

Fig. 3. An Example of a decoded Rich Signature

3.3 유사도 계산

본 논문에서 제안하는 방법은 실행파일을 분류하기 위해 실행파일에서 추출한 리치 시그니처의 정보를 이용하여 유사도를 계산한다. 리치 시그니처에 포함된 컴파일러 정보를 이용한 유사도 계산 수식 S_C는 Equation (1)로 정의하고, 참조횟수 정보를 이용한 유사도 계산 수식 S_R은 Equation (2)로 정의한다.

$$S_C(A,B) = \frac{n(A_C \cap B_C)}{n(A_C \cup B_C)} \quad (1)$$

$$S_R(A,B) = 1 - \frac{\sum_{a \in A_C \cap B_C} |A_R(a) - B_R(a)|}{\sum_{a \in A_C \cap B_C} A_R(a) + \sum_{a \in A_C \cap B_C} B_R(a)} \quad (2)$$

단, $n(A)$: 집합 A의 원소의 개수,
A_C : 파일 A의 컴파일러 정보의 집합,
$A_R(a)$: 파일 A의 컴파일러 정보 a에 대한 참조 횟수의 값

실행파일 분류에 사용되는 유사도는 S_C는 비교 대상 실행파일 A, B의 리치 시그니처에 포함된 컴파일러 정보 중에서 A, B에 공통으로 포함된 컴파일러 정보 수를 A, B의 컴파일러 정보의 합집합 수로 나눈 값이다. A와 B가 동일한 컴파일러 정보를 가진다면 S_C값은 1이 되고 A와 B에 동일한 컴파일러 정보가 하나도 없다면 S_C값은 0이 된다. A와 B에 공통으로 포함된 컴파일러 정보가 많을수록 유사도는 높으며, 이 값은 1에 가까운 값을 갖는다. S_R은 비교 대상 파일 A, B의 부분 유사도 값을 계산하는데 사용되며, 공통으로 포함된 컴파일러 정보에 대해 각각의 참조횟수 값이 얼마나 유사한지 계산한다. S_R은 S_C값이 1인 실행파일에 대한 버전별 세부분류에 사용된다. A와 B에 공통으로 포함된 컴파일러 정보가 없다면 S_R은 의미가 없으므로 이 경우의 값은 0으로 정의한다. A와 B에 공통으로 포함된 컴파일러 정보 각각에 대한 참조값들도 모두 같다면 S_R값은 1이 되며, 컴파일러 정보 각각에 대한 참조값들이 다르고 차이가 크다면 이 값은 0에 가까운 값을 갖는다. 만약 비교 대상 파일 A와 B의 리치 시그니처가 완전히 동일하다면 S_C값과 S_R값은 모두 1이 된다. 그리고 A와 B의 리치 시그니처에 포함된 컴파일러 정보만 동일하다면 S_C값은 1이 되고 S_R값은 0과 1 사이의 값이 된다.

3.4 실행파일 식별 및 분류

실행파일의 식별 및 분류는 3.3의 유사도 계산식과 분류 데이터베이스에 저장된 정보를 이용한다. 분류 데이터베이스에는 분류된 리치 시그니처 정보와 실행파일의 정보 및 컴파일러 정보를 저장한다.

알고리즘명: ClassificationOfExecutableFile
입력: 실행파일명
출력: 분류결과

1. GetRichSignature 알고리즘을 이용하여 실행파일의 디코드된 리치 시그니처 정보를 획득한다.
2. GetRichSignature 과정에 오류가 발생한 경우는 에러값을 반환하고 종료한다.
3. 분류 데이터베이스에 저장된 리치 시그니처 정보를 하나씩 가져온다.
4. 실행파일의 디코드된 리치 시그니처 정보와 분류 데이터베이스에서 가져온 리치 시그니처 정보를 이용하여 유사도 S_C를 계산한다.
5. 가장 큰 유사도 S_C값을 선택하여 정보를 저장한다.
6. 분류 데이터베이스에 저장된 모든 리치 시그니처 정보를 가져올때까지 과정 2 ~ 4를 반복한다.
7. 임계값과 선택된 유사도 값을 비교한다.
8. 선택된 유사도값이 임계값보다 작으면 분류 데이터베이스에 없는 새로운 파일이므로 새로운 실행파일 그룹으로 분류하고 분류 데이터베이스에 정보를 저장한 다음 종료한다.
9. 선택된 유사도값이 임계값보다 크거나 같으면 해당 실행파일을 선택된 유사도값의 리치 시그니처 그룹으로 분류한다.
10. 유사도 S_C값이 1이 아닌 경우는 세부 분류 없이 종료한다.
11. 유사도 S_C값이 1인 경우에는 유사도 S_R을 계산하여 세부분류를 실행한다.
12. 유사도 S_R값이 1인 경우는 분류된 것과 동일한 경우이므로 종료한다.
13. 유사도 S_R값이 1이 아닌 경우에는 세부 분류를 하고 정보를 저장한 다음 종료한다.

GetRichSignature 알고리즘을 이용하여 비교 대상 실행파일에서 디코드된 리치 시그니처 정보를 획득하고, 분류 데이터베이스에 저장된 리치 시그니처 정보를 얻어온다. 획득된 2가지 정보를 이용하여 유사도 S_C를 계산하고 설정된 임계값(Threshold)에 따라 대분류를 수행한다. 임계값은 0과 1사이의 값이다. 세부 분류가 필요한 경우에는 유사도 S_R를 계산하여 실행파일에 대한 세부 분류를 수행한다.

4. 실험 및 결과

4.1 실험 데이터

실험 데이터는 일반 실행파일과 악성코드 실행파일 2종류이며, 일반 실행파일은 한글 2010파일(hwp.exe), MS 파워포인트 2010(powerpnt.exe), 윈도우 7에 포함된 노트패드(notepad.exe), 계산기(calc.exe) 프로그램이고, 악성코드는 Stuxnet 계열의 Stuxnet 파일 샘플 파일 6개, Duqu 샘플 파일 2개, Flame 샘플 파일 11개, Gauss 샘플 파일 10개로 구성된다. 그리고, 악성코드의 분류 실험을 위해 CN시큐리

티[21]에서 제공하는 악성코드 데이터를 사용한다. 실험을 위해 분류 데이터베이스에는 미리 분류된 Duqu, Hwp 2010, PowerPoint 2010의 정보가 저장되어 있다. 제안한 방법의 실험은 다음 4가지 항목(실행파일 분류 실험, 악성코드 분류 실험, 압축, 코드변형 및 난독화된 실행파일 분류 실험, 삭제된 저장매체에서 복구된 실행파일의 분류 실험)에 대해 수행하였다.

4.2 실행파일 분류 실험

제안한 방법을 이용하여 실행파일 분류 실험을 수행한 결과는 Table 1과 같다. 분류 실험에 사용될 샘플 파일은 임의로 3개의 파일(hwp.exe, notepad.exe, ClasmAV.Trojan.Duqu.Infostealer.20111108)을 선정하여 구성하였으며, 임계값은 0.9로 설정하였다. 표안의 값은 유사도 S_C의 계산값이며, 유사도 S_C값이 1인 경우는 괄호 속에 유사도 S_R값을 계산하여 표시하였다.

Table 1. Experimental results of executable files

Files	G_DUQU_2011	G_HWP_2010	G_POWERPOINT_2010	Results(T=0.9)
Sample1	1.0(1.0)	0.15	0.14	G_DUQU_2011
Sampe2	0.15	1.0(1.0)	0.50	G_HWP_2010
Sampe3	0.15	0.35	0.70	G_UNKNOWN

실험결과 Sample1은 Duqu(G_DUQU_2011), Sample2는 한글 2010 (G_HWP_2010), Sample3은 알려지지 않은 그룹(G_UNKNOWN)으로 정상적으로 분류되었다. Sample1과 Sample2의 유사도 S_C와 유사도 S_R값이 모두 1이므로 분류 데이터베이스에 저장된 정보와 동일한 실행파일임을 확인할 수 있다. 그리고 G_UNKNOWN으로 분류된 Sample3 파일은 분류 데이터베이스에 없는 정보이므로 분류 데이터베이스에 새롭게 추가된다. 악성코드인 Duqu와 문서작성 프로그램들과의 유사도는 0.15로 낮게 나타나고, 프로그램 제작사는 다르지만 기능이 유사한 문서작성 프로그램인 한글 2010과 파워포인트 2010의 유사도는 0.50으로 높게 나타났다. 또한, 프로그램 제작사가 동일하고 제품 출시가 비슷한 파워포인트 2010과 노트패드인 Sample3의 유사도는 0.70으로 높게 나타났다. 따라서, 실행파일의 특성에 따라 임계값을 조정하여 Sample3과 파워포인트 2010은 동일한 제작사에서 개발 프로그램으로 분류가능하며, Sample2와 파워포인트 2010은 유사한 기능을 갖는 문서작성 프로그램으로 분류가 가능하다.

4.3 악성코드 분류 실험

Table 2는 CN시큐리티에서 제공하는 데이터를 이용한 악성코드 분류 실험 결과이다. 실험에 사용된 데이터는 2012년 3월에 수집된 악성코드에서 800개를 선정하여 사용하였다. 국내 백신프로그램으로 검사한 결과 600개를 악성코드로 탐지하였으며 제안된 방법으로 분류한 결과 512개의 파일을 분류하였다. 800개의 실험데이터 중 288개의 파일은 Rich Signature가 존재하지 않아서 제안된 방법으로는 분류할 수 없었다. 그리고 유사도 S_C와 악성코드 분류를 위한 임계값을 1.0, 0.9, 0.85로 설정하여 실험하였으며, 임계값이 1.0인 경우의 분류 결과를 정상적인 분류로 가정하였다.

Table 2. Experimental results of malicious codes

num	T = 1.0		T = 0.9		T = 0.85	
	Group	Files	Group	Error Files	Group	Error Files
1	150	150	142		113	
2	25	50	23		22	
3	7	21	8	3	13	18
4	3	12	5	8	6	12
5	6	30	5		4	
6	4	24	5	6	4	
8					1	8
9	1	9	1		1	
12	1	12	1		1	
14	2	28	2		1	
15					1	15
16	1	16	1		1	
20	1	20	1		1	
21	1	21	1		1	
23	1	23	1		1	
36	1	36	1		1	
60	1	60	1			
74					1	74
sum	205	512	198	17	173	127

임계값이 1.0인 경우 총 512개의 파일에 대해 그룹의 원소수가 1개인 그룹이 150개, 그룹의 원소수가 60개인 그룹이 1개 등으로 분류되어 총 205개의 그룹으로 분류되었다. 그러나 임계값에 따라 잘못 분류된 경우가 발생하였고, 임계값이 낮을수록 분류 그룹수는 적어지고 잘못 분류된 파일의 수는 많아졌다. 임계값이 0.9인 경우는 3.3% (17/512)의 오류가 발생하였으며, 임계값이 0.85인 경우는 24.8% (127/512)의 오류가 발생하였다. 실험결과 악성코드 분류를 위한 임계값은 0.9가 0.85보다 타당함을 확인할 수 있다. 그리고 같은 그룹으로 분류된 파일들은 모두 유사한 형태의 파일들이다. 예를 들어, 원소수가 60개인 그룹에 속한 파일들은 백신 프로그램에서 트로이목마로 분류되며, 분석결과 이 파일들은 모두 UPX로 패킹된 파일이다.

Table 3은 프로그램 제작 환경이나 제작자 또는 그룹별로 악성코드를 분류하는 실험결과이다. 4.2의 실행파일 분류 실험과 동일하게 미분류된 실험샘플들과 분류 데이터베이스에 저장된 분류된 실행파일 그룹들의 유사도를 비교하여 동일한 프로그램 제작환경 및 프로그램 제작자 또는 그룹별로 악성코드를 식별한다. 실험에 사용된 샘플은 실제 피해시스템에서 수집한 스턱스넷 계열의 샘플파일들이다. 피해시스템에

따라 수집된 파일명이 다를 수 있기 때문에 파일의 해쉬결과를 파일이름으로 표현한다. 유사도에 의한 그룹 분류 실험에서 사용될 임계값은 0.85으로 설정한다. 이 실험은 악성코드들이 얼마나 유사한지를 판단하기 위한 목적이므로 악성코드를 분류하는 경우의 임계값보다 낮게 설정하여 실험한다. 예를 들어 제안방법으로 Stuxnet과 Duqu의 유사성을 계산하여 비슷한 그룹으로 분류된다면 Stuxnet과 Duqu를 개발한 그룹간의 연관성을 판단하는 근거로 사용할 수 있다.

Table 3. Experimental results of Stuxnet

num	Files	G_DUQUE_2011	G_HWP_2010	G_POWERPOINT_2010	Results (T=0.85)
1	1e17d81979271cfa44d471430fe123a5	**0.87**	0.15	0.13	G_DUQU_2011
2	37fc7c5d89f1e5a96f54318df1a2b905	0.14	0.33	0.50	G_UNKNOWN
3	68eb6d3adc49da0a79aff2202bbb3bea	0.17	0.64	0.46	G_UNKNOWN
4	74ddc49a7c121a61b8d06c03f92d0c13	0.15	0.35	0.54	G_UNKNOWN
5	cc1db5360109de3b857654297d262ca1	X	X	X	NO_RICHSIGNATURE
6	f8153747bae8b4ae48837ee17172151e	**0.87**	0.15	0.13	G_DUQU_2011

실험결과 수집된 스턱스넷 악성코드 샘플 6개중 2개의 파일(1번, 6번 파일)과 Duqu 악성코드의 유사도가 높게 나타났으며, 스턱스넷 악성코드는 Duqu 악성코드와 유사한 악성코드 그룹으로 분류된다. 따라서, 스턱스넷을 제작한 프로그램 그룹과 Duqu를 제작한 프로그램 그룹은 유사한 기능을 갖는 파일을 제작하여 악성코드를 만들었으며, 소스코드가 공개되지 않는 악성코드의 특성상 Duqu를 제작한 그룹은 스턱스넷을 제작한 그룹과 동일하거나 두 그룹 사이에는 특별한 연관관계가 있다고 유추할 수 있다. 5번 파일은 실행파일의 헤더내에 유사도 측정에 사용되는 컴파일러 정보가 없기 때문에 유사도를 계산할 수 없으며 NO_RICHSIGNATURE 에러 메시지를 출력한다. 유사도가 높게 나타난 스턱스넷 샘플 파일(1번과 6번)은 실제로 암호화와 은폐에 사용되는 커널 레벨의 루트킷 드라이버 프로그램이며, 분류 데이터베이스에 저장된 Duqu의 리치 시그니처는 Duqu에서 사용되는 동일한 목적의 커널 레벨의 루트킷 프로그램이다.

Table 4. Experimental results of Stuxnet family

Files	Stuxnet(6)	Duqu(2)	Flame(11)	Gauss(10)
Stuxnet(6)	-	(0.87, 0.14)	(0.45, 0.09)	(0.45, 0.15)
Duqu(2)	(**0.87**, 0.14)	-	(0.36, 0.21)	(0.36, 0.33)
Flame(11)	(0.45, 0.09)	(0.36, 0.21)	-	(1.0, 0.58)
Gauss(10)	(0.45, 0.15)	(0.36, 0.33)	(**1.0**, 0.58)	-

Table 4는 제안한 방법으로 Stuxnet 계열의 악성코드들간의 유사도를 계산한 실험결과이다. 괄호안의 숫자는 샘플의 파일 수를 의미하며, 샘플별 유사도의 최대값과 최소값을 (최대값, 최소값)으로 표시한다. 실험결과 Stuxnet과 Duqu의 최대 유사도는 0.87이고 Falme과 Gauss의 최대 유사도는 1.0으로 나타났다. 그러나, Stuxnet과 Flame, Stuxnet과 Gauss의 유사도는 0.45로 낮게 나타났으며, Duqu와 Flame, Duqu와 Gauss의 유사도는 0.36으로 낮게 나타났다. 따라서, 제안한 방법으로 Stuxnet과 Duqu의 연관성, Flame과 Gauss의 연관성이 높음을 확인할 수 있다. 수집된 악성코드는 파일별로 수행하는 역할이 다르기 때문에 유사도가 다르게 나타난다. 유사도가 1.0으로 높게 나나타난 샘플 파일은 Gauss의 winshell.ocx, smdk.ocx, mcdmn.ocx와 Flame의 msglu32.ocx, adventcfg.ocx, adventcfg2.ocx 파일이다.

4.4 압축, 코드변형 및 난독화된 실행파일 분류 실험

Table 5는 압축, 코드변형 및 난독화된 실행파일에 대한 분류 실험 결과이다. 실험에 사용된 실행파일은 윈도우 7에 포함된 계산기 프로그램인 calc.exe 파일이고, 난독화 상용제품인 Themida와 VMProtect 프로그램을 이용하여 실험파일에 대해 서로 다른 옵션을 선택하여 각각 2개의 실험파일을 생성하였다. 실험파일 분류 데이터베이스에는 앞 실험의 실행파일 데이터베이스에 계산기 정보가 추가된 것을 사용하였다. 실험에서 사용될 임계값은 0.90으로 설정한다.

실험결과 Themida와 VMProtect를 이용하여 압축 또는 난독화시킨 경우에도 모두 유사도 1.0으로 원본과 동일한계산기 프로그램(G_CALC_W7)으로 분류된다. 리치 시그니처가 저장된 PE 헤더 부분을 변경하지 않는 경우에는 제안한 방법으로 정상적으로 분류가 가능하다. 그러나 리치 시그니처가 저장된 PE 헤더 부분을 변경하는 일부 악성코드에 대해서는 제안한 방법으로 분류가 불가능하다.

Table 5. Experimental results of packed, code modified and obfuscated executable files

Files	G_DUQU_2011	G_HWP_2010	G_POWERPOINT_2010	G_CALC_W7	Results (T=0.9)
calc.exe	0.07	0.05	0.06	**1.0(1.0)**	**G_CALC_W7**
calc_the1.exe	0.07	0.05	0.06	**1.0(1.0)**	**G_CALC_W7**
calc_the2.exe	0.07	0.05	0.06	**1.0(1.0)**	**G_CALC_W7**
calc_vm1.exe	0.07	0.05	0.06	**1.0(1.0)**	**G_CALC_W7**
calc_vm2.exe	0.07	0.05	0.06	**1.0(1.0)**	**G_CALC_W7**

4.5 삭제된 저장매체에서 복구된 실행파일의 분류 실험

Table 6은 삭제된 저장매체에서 복구된 파일들에 대한 실행파일 분류 실험결과이다. 실험에 사용된 샘플파일은 한글 2010패키지가 설치된 폴더의 파일을 삭제하고 컴퓨터 포렌식 프로그램을 사용하여 hwp.exe 파일을 포함한 7개의

파일을 복구한 샘플을 사용한다. 복구된 파일은 원래 파일명을 알 수 없으며 원본내용과 100% 동일하게 복구되었다는 보장은 없다. 실험파일 분류 데이터베이스에는 앞 실험과 동일하며, 실험에서 사용될 임계값은 0.90으로 설정한다.

Table 6. Experimental results of recovered executable files

num	Files	G_DUQUE_2011	G_HWP_2010	G_POWERPOINT_2010	Results (T=0.9)
1	_91333000443074801 66_$RQWNO0K.dll	0.21	0.66	0.18	G_DUQU_2011
2	_91333000443074801 67_$R5ZU37Y.enu	X	X	X	NOT_PE
3	_91333000443074801 68_$RD35MBR.kor	X	X	X	NOT_PE
4	_91333000443074016 9_$RT0I1X6.exe	0.15	**1.0(1.0)**	0.50	**G_HWP_2010**
5	_91333000443074801 70_$RMBAQG3.dll	0.16	**0.93**	0.53	**G_HWP_2010**
6	_91333000443074801 71_$R55ZN19.kor	X	X	X	NOT_PE
7	_91333000443074801 72_$R2EIKSY.tlb	X	X	X	NOT_PE

실험결과 복구된 파일에서 4번 샘플 파일은 유사도가 1.0으로 G_HWP_2010 실행파일로 정상적으로 분류하였다. NOT_PE로 분류된 파일들은 PE형식이 아니므로 유사도 계산 및 실행파일 분류 데이터베이스에는 정보가 저장되지 않는다. 5번 파일은 유사도가 0.93으로 한글 2010과 유사한 그룹(G_HWP_2010)으로 분류되었다. 1번 파일은 한글 2010 그룹으로 분류되지는 않았지만 유사도가 0.66으로 상대적 높아 한글 2010 실행파일과 비슷한 환경에서 제작되었다고 판단할 수 있다.

5. 결론 및 향후과제

본 논문에서는 윈도우 실행파일 포맷인 PE 헤더에 포함된 문서화되지 않은 정보를 이용하여 실행파일을 분류하는 방법을 제안한다. 제안하는 방법은 윈도우 실행파일 헤더에서 인코딩된 리치 시그니처를 추출하고, 추출된 정보를 디코딩하여 리치 시그니처에 포함된 컴파일러 정보와 참조횟수를 유사도 계산에 사용하여 실행파일을 분류한다. 제안한 방법은 실험을 통해 일반적인 실행파일 및 악성코드를 분류할 수 있음을 보였으며, 압축이나 코드변형, 가상화 및 난독화된 실행파일에 대한 분류와 삭제된 저장매체에서 복구된 실행파일의 분류에도 적용 가능함을 보였다. 따라서 제안한 방법은 실행파일 제작자 또는 그룹별로 실행파일을 분류할 수 있으므로 악성코드 탐지 및 보안관제 분야에 적용가능하며, 컴퓨터 포렌식 분야에 적용되어 소프트웨어 저작권 침해 수사 및 침입 도구 탐지에 적용 가능하다. 또한, 새로운 악성코드와 변종 및 난독화된 악성코드에 대한 탐지분야도 적용가능하다. 그러나 제안한 방법은 문서화되지 않은 리치 시그니처 정보를 이용하기 때문에 실행파일에 리치 시그니처가 없는 경우에는 실행파일 분류가 불가능하다는 단점이 있다. 향후 제안한 방법을 보완하여 실행파일을 분류할 수 있는 방법에 대한 연구를 지속해야 한다.

참 고 문 헌

[1] M. Gheorghescu, “An Automated virus classification system,” in Proceedings of the 2005 Virus Bulletin Conference, pp. 294-300, 2005.

[2] N. Y. Park, Y. M. Kim, and B. N. Noh, “A Behavior based Detection for Malicious Code Using Obfuscation Technique,” The Journal of KIISC: Vol.16, No.3, pp.17-28, 2006.

[3] A. Karnik, S. Goswami, and R. Guha, “Detecting Obfuscated Viruses Using Cosine Similarity Analysis,” in Proceedings of the 1th Asia International Conference on Modelling & Simulation, pp.165-170, March, 2007.

[4] Q. Zhang and D. S. Reeves, “MetaAware: Identifying Metamorphic Malware,” in Proceedings of the 23rd Annual Computer Security Applications Conference, pp.411-420, September, 2007.

[5] G. Bonfante, M. Kaczmarek, and J. Y. Marion, “Morphological Detection of Malware,” in Proceedings of the 3rd International Conference on Malicious and Unwanted Software, pp.1-8, October, 2008.

[6] R. Tian, L. M. Batten, and S. C. Versteeg, “Function Length as a Tool for Malware Classification,” in Proceedings of the 3rd International Conference on Malicious and Unwanted Software, pp.69-76, October, 2008.

[7] S. M. Tabish, M. Z. Shafiq, and M. Farooq, “Malware Detection using Statistical Analysis of Byte-Level File Content,” in Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, pp.23-31, June, 2009.

[8] J. Lee, K. Jeong, and H. Lee, “Detecting Metamorphic Malwares using Code Graphs,” in Proceedings of the 2010 ACM Symposium on Applied Computing, pp.1970-1977, March, 2010.

[9] S. Cesare and Y. Xiang, “A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost,” in Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, pp.721-728, April, 2010.

[10] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, “SplitScreen: Enabling Efficient, Distributed Malware Detection,” in Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, April, 2010.

[11] D. Y. Yang and S. C. Yeo, “Windows Abnormal Classification Based on API Call Group Conditions,” The Journal of KITCS: Vol.2, No.1, pp.13-20, 2010.
[12] Q. G. Miao, Y. Wang, Y. Cao, X. G. Zhang, and Z. L. Liu, “APICapture - a Tool for Monitoring the Behavior of Malware,” in Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering, pp.390-394, August, 2010.
[13] V. P. Nair, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi, “MEDUSA: MEtamorphic malware dynamic analysis using signature from API,” in Proceedings of the 3rd International Conference on Security of Information and Networks, pp.263-269, September, 2010.
[14] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors,” in Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp.45-60, November, 2010.
[15] K. S. Han, I. K. Kim, and E. G. Im, “A Proposal of Evaluation Index for Malware Collecting and Analyzing,” The Journal of Security Engineering: Vol.8, No.1, pp.77-88, 2011.
[16] K. S. Han, I. K. Kim, and E. G. Im, “Malware Family Classification Method using API Sequential Characteristic,” The Journal of Security Engineering: Vol.8, No.2, pp.319-335, 2011.
[17] J. W. Park, S. T. Moon, G. W. Son, I. K. Kim, K. S. Han, E. G. Im, and I. G. Kim, “An Automatic Malware Classification System using String List and APIs,” The Journal of Security Engineering: Vol.8, No.5, pp.611-626, 2011.
[18] S. H. Kim, S. B. Ji, and Y. S. Park, “Malware Analysis Method Using Pin,” in Proceedings of the 38th KIISE Fall Conference, Vol.38, No.2(C), pp.187-190, 2011.
[19] ClamAV, http://www.clamav.net/
[20] Rich Signature, http://www.ntcore.com/Files/richsign.htm
[21] CNSECURITY, http://www.cnsec.co.kr

김 정 순

e-mail : cyberpia@ensec.re.kr
1998년 전남대학교 전산학과(학사)
2000년 전남대학교 전산통계학과(석사)
2006년 전남대학교 전산학과(박사)
2010년~현 재 한국전자통신연구원 부설연구소 연구원
관심분야 : 정보보호

강 정 민

e-mail : jmkang@ensec.re.kr
2002년 광주과학기술원(GIST) 정보통신공학과(석사)
2002년~2003년 삼성 SDS
2011년 고려대학교 컴퓨터교육학과(박사수료)
2003년~현 재 한국전자통신연구원 부설연구소 선임연구원
관심분야 : 정보보호

김 강 산

e-mail : kahn@ensec.re.kr
2002년 경북대학교 전자공학과(학사)
2004년 한국과학기술원 전기 및 전자공학과(석사)
2004년~현 재 한국전자통신연구원 부설연구소 선임연구원
관심분야 : 정보보호

신 욱

e-mail : sunihill@geguri.kjist.ac.kr
2005년 광주과학기술원 정보통신공학과(박사)
2005년~2007년 UIUC Postdoc
2008년~2011년 (주)KDDI 연구소 연구원
2011년~현 재 한국전자통신연구원 부설연구소 선임연구원
관심분야 : 정보보호