문자열 비교 기법을 이용한 악성코드 탐지 및 분류 연구

이진경 임채태 정현철
한국인터넷진흥원
e-mail: neobug@gmail.com

A Study of Malware Detection and Classification by Comparing Extracted Strings

Jinkyung Lee, Chaetae Im, Hyuncheol Jeong
Korea Internet and Security Agency

요 약

최근 급격하게 증가하고 있는 악성코드에 비해 이들을 분석하기 위한 전문 인력은 매우 부족하다. 다행히 양산되는 악성코드의 대부분은 기존의 것을 수정한 변종이기 때문에 이들에 대해서는 자동분석 시스템을 활용해서 분석하는 것이 효율적이다. 악성코드 자동분석에는 동적 분석과 정적 분석 모두가 사용되지만 정적 분석은 여러 가지 한계점 때문에 아직까지도 개선된 연구를 필요로 한다.

본 논문은 문자열 비교를 통해 두 실행파일에 대한 유사도를 측정함으로써 악성코드 판별 및 분류를 도와주는 정적 분석기법을 제안한다. 제안된 방법은 비교 문자열의 수와 종류에 따라 그 성능이 결정되기 때문에 문자열들을 정제하는 과정이 선행된다. 또한 유사도 측정에 있어서 악성코드가 가지는 문자열들의 특성을 고려한 개선된 비교방법을 보인다.

1. 서 론

최근 수년간 악성코드의 수가 큰 폭으로 증가하고 있다. 시만텍의 '인터넷 보안 위협 보고서'에 따르면 2009년 한 해 동안 280만개 이상의 악성코드 시그니쳐가 생성되었고 이는 전년 대비 71퍼센트 가량 증가된 수치이다. 또한 이는 지금까지 생성된 모든 시그니쳐의 51퍼센트에 해당된다[1]. 이처럼 폭발적으로 증가하는 악성코드에 대응하기 위해서는 전문가의 양성도 중요하지만 분석 시스템의 자동화가 필수불가결하다.

악성코드 분석 시스템은 크게 동적 분석을 이용한 방법과 정적 분석을 이용한 방법으로 구분된다. 동적 분석은 파일을 수행시킴으로써 분석대상이 어떤 행위를 하는지 그리고 어떤 영향을 미치는지에 대한 정보를 얻을 수 있다. 이는 악성여부 탐지 및 분석대상의 행위 특성을 결정짓는데 도움이 된다. 반면 정적 분석은 파일을 수행시키지 않고 분석을 수행하기 때문에 분석 시스템에 적용하기에는 여러 가지 제약이 존재한다. 하지만 정적 분석은 분석된 악성코드와의 비교를 통해 특정 악성코드의 변종 여부를 판단할 수 있는 장점이 있다.

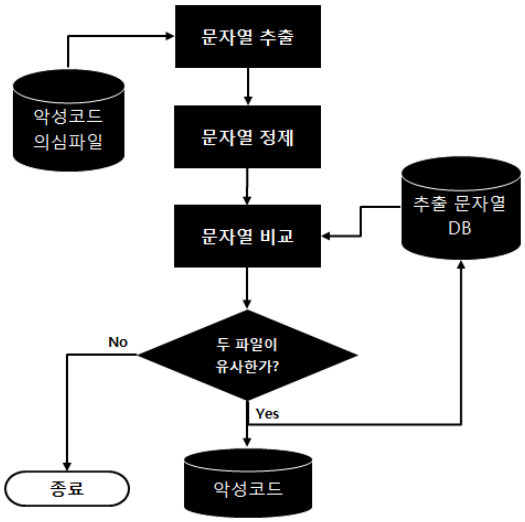
대표적인 악성코드 정적 분석 중 하나는 실행파일의 코드영역을 분석하여 프로그램의 분기점을 그래프로 표현하는 방법이 있다. 이 CFG(Control flow Graph)를 이용한 악성코드 분석은 두 실행파일에 대한 유사성 검증을 자동화하는데 적합하다[2]. 마찬가지로 실행파일로부터 추출한 문자열을 비교함으로써 두 실행파일에 대한 유사성을 검증하는 방법 역시 악성코드 자동분석 시스템에 충분히 유효하다. 특히 디스어셈블 기능을 방해하는 요소나 분기 난독화 기능이 포함된 실행파일에 대해서는 전자의 방법을 사용할 수 없기 때문에 후자와 같은 범용성 높은 정적 분석 기법연구가 필요하다.

본 논문은 문자열 비교를 통한 악성코드 탐지 및 분류 시스템을 제안하고 있다. 보편적으로 문자열 비교는 비교할 수가 많을수록 복잡도가 크게 증가한다. 또한 문자열의 종류에 따라 유사도 측정결과가 판이하게 달라지기도 한다. 따라서 악성코드 탐지 및 분류 시스템의 성능을 향상시키기 위해서는 추출된 문자열에 대한 적절한 정제과정이 필수적이다. 그리고 문자열 비교방법에 있어서 단순히 동일 문자열만을 찾기보다는 악성코드가 가지는 문자열 특성을 고려해서 상호유사성을 점검하는 방법이 추가될 필요가 있다.

2. 악성코드 탐지 및 분류 시스템

문자열 비교를 통한 악성코드 탐지 및 분류 시스템은 바이너리 파일로부터 추출 가능한 모든 문자열을 뽑아낸

후 각각의 문자열들을 비교함으로써 두 파일간의 유사도를 판단하는 시스템이다. 예를 들어, 두 파일의 유사도가 매우 높고 하나의 파일이 이미 분석된 악성코드라면 다른 한 파일은 이 악성코드의 변종일 가능성이 높은 것이다. 즉, 기존에 분석 완료된 악성코드를 비교기준으로 삼아 새롭게 입력되는 의심스런 바이너리 파일에 대한 악성여부 판단 및 변종정보를 결정해주는 시스템이다.



(그림 1) 문자열 비교를 통한 두 파일간의 유사성 판단 프로세스

(그림 1) 문자열 비교를 통한 두 파일간의 유사성 판단 프로세스는 크게 문자열 추출, 문자열 정제, 문자열 비교라는 3가지 과정으로 구성된다.

문자열 추출과정은 바이너리에 존재하는 모든 표현 가능한(Printable) 문자열을 추려내는 과정이다. 바이너리 데이터가 아스키 혹은 유니코드 표준에 정의된 문자영역 데이터를 연속으로 가진다면 그 데이터는 문자열로 판단할 수 있다. 일반적으로 문자열은 널(Null) 값을 종결자로 가지지만 실행파일 안에 존재하는 문자열은 항상 그런 것은 아니기 때문에 0x00으로 종료되지 않는 연속된 문자영역 데이터도 고려해야 한다. 최근 이슈가 되는 악성코드는 미국을 제외한 중국, 브라질, 인도 등에서 가장 활발하게 활동하고 있으므로 해당국가 고유의 문자영역을 문자열 추출 기준에 포함시키는 것도 좋은 방법이다[1].

문자열 정제과정은 추출된 문자열 중 악성코드 탐지 및 분류에 방해되는 요소들을 제거하는 과정이다. 바이너리로부터 추출된 문자열은 그 수가 많으면 많을수록 문자열을 비교하는데 소모되는 시간이 급격히 증가한다. 대량의 악성코드를 자동 분석해야 하는 시스템은 반드시 이러한 성능문제가 고려되어야 하기 때문에 추출된 문자열의 수를 줄이는 과정은 필수적이다. 또한 악성코드뿐만 아니라 일반 실행파일에서도 쉽게 찾을 수 있는 문자열의 경우는 악성코드 탐지 및 분류의 적중률을 낮추기 때문에 이러한 문자열 역시 제거되는 것이 좋다.

문자열 비교 과정은 정제과정을 거친 문자열 간에 비교를 통해서 하나의 바이너리가 다른 바이너리와 얼마나 유사한지를 판단하는 과정이다. 기본적으로 상호간에 부합되는 문자열의 수가 얼마나 되는지를 파악함으로써 두 바이너리 파일간의 유사성을 측정한다. 그리고 추가적으로 특정 문자집합을 포함하는 문자열의 경우 하나의 문자열이 다른 문자열에 포함되거나 혹은 각각의 문자열 유사도(Edit Distance)가 임계치(Threshold) 이상인지를 판단함으로써 전자의 측정 데이터를 보완한다. 이는 악성코드가 포함하고 있는 URL 문자열 데이터가 수시로 변경되어 재배포되는 특성 등을 고려한 것이다.

3. 문자열 정제 및 비교 방법

3.1 문자열 정제 방법

[표 1] 추출 문자열 구성 표

문자열 구분 기준	구성 률*	문자열 수*
10글자 이하의 모든 문자열	83%	86084
윈도우 DLL파일명 및 API 함수명	4%	4509
프로그램 언어에 종속적인 함수군	2%	2609
PE 파일 포맷의 기본 문자열	0.09%	103
나머지 문자열	10.91%	10464
Total	100 %	103769

* 실험을 위해 선택된 100개의 악성코드로부터 추출된 모든 문자열에 대한 수치

실험을 위해 선택된 100개의 악성코드로부터 문자열을 추출한 후 그 구성을 분석한 결과, 다음과 같이 악성코드 탐지 및 분류 시스템의 성능에 영향을 미치는 요소들을 분류할 수 있었다.

- 10글자 이하의 문자열
- 10글자 이상이지만 의미 없는 문자열
- 윈도우즈 DLL파일 및 API
- 프로그램 언어에 종속적인 함수군
- PE 파일 포맷이 기본적으로 가지는 문자열

실행파일로부터 추출된 문자열들은 [표 1]과같이 10글자 이하의 문자열들이 대부분을 이룬다. 이 문자열 집합은 특수문자와 숫자 등으로 구성된 의미 없는 문자열과 짧지만 의미가 있는 문자열로 구성되어 있는데, 이 역시 전자와 같은 문자열이 90% 이상을 차지하기 때문에 나머지 문자열들을 확보하기 위해 정제조건을 복잡하게 만들 필

요는 없다. 게다가 짧은 문자열은 각 문자열간의 유사도를 측정하는데 있어서 신뢰하기 어렵다.

10글자 이상의 문자열 중에서도 반복되는 특수문자와 숫자의 조합으로 구성된 의미 없는(Trash) 문자열이 나타난다. 그 수는 적은 편이지만 정제가 가능하다면 해주는 것이 좋다. 한 문자열의 문자 조합이 (그림 2)와 같은 논리식에 만족할 때 해당 문자열을 삭제한다.

```
IF ( 특수문자+숫자 > 소문자+대문자 )
    선택된 문자열 삭제
ELSE
    선택된 문자열 저장
```

(그림 2) Trash 문자열 정제수식

PE(Portable Executable)는 파일이 실행되는 순간 동적 라이브러리를 메모리에 로드(Load)시키기 위해 DLL 파일 명과 그 파일 안에 정의된 API 함수 명을 포함하고 있다. 따라서 실행파일로부터 문자열을 추출하면 다량의 DLL 파일명과 Windows API 함수 명이 출력된다. 두 요소는 실행파일간의 유사성을 측정하는데 있어서 효과적이긴 하지만 범용 적으로 사용되는 DLL 및 Windows API 함수 명은 악성코드 탐지 및 분류를 어렵게 만든다. 따라서 일반 실행파일에서는 잘 사용되지 않는 희귀한(Rare)한 Windows API 함수를 제외한 모든 DLL 파일명 및 함수 명은 제거되어야 한다.

악성코드는 다양한 언어로 작성되어 다양한 컴파일러로 생성된다. 대표적으로 악성코드는 C나 C++ 언어로 만들어지지만 역 공학(Reverse Engineering)을 어렵게 만들기 위해서 델파이 나 비주얼베이직 등의 언어로 작성되기도 한다. 이 때 각 언어에서 제공하는 라이브러리 함수를 이용해 악성코드를 작성하게 되면 최종적으로 이 함수들의 이름이 실행파일에 기록되기도 한다. 특히 비주얼베이직의 경우 컴포넌트 방식의 프로그래밍 언어이기 때문에 일반 실행파일이나 악성코드에 사용되는 함수들의 종류가 판이하게 다르지 않다. 따라서 '__vba'로 시작되거나 '_adj' 등의 접두어를 가진 문자열에 대해서는 삭제를 고려해봐야 한다.

PE는 Windows 운영체제의 실행파일 포맷이다. 파일이 악성코드이건 아니건 Windows 운영체제에서 수행되려면 PE 구조를 가져야 한다. 따라서 PE 실행파일 포맷이 기본적으로 가지는 문자열은 악성코드 탐지 및 분류에 의미가 없다. PE 헤더의 처음에 존재하는 '!This program cannot be run in DOS mode.' 나 'This program must be run under Win32' 등의 문자열은 항상 제거되어야 한다.

3.2 문자열 비교 방법

악성코드 탐지 및 분류 시스템에서 사용되는 문자열 비교 방법은 다음과 같이 크게 두 가지로 구분된다.

- 두 문자열 집합 간에 동일한 문자열 수 측정
- 특수한 문자를 포함하고 있을 경우 두 문자열 집합 간에 유사도 측정

두 문자열 집합 간에 동일한 문자열 수가 많으면 많을수록 상호 유사성은 높아진다. 문자열 데이터를 직접적으로 수정하지 않는 이상 변종 악성코드는 기존의 문자열 데이터를 포함하고 있을 가능성이 높다. 따라서 문자열 비교를 통한 악성코드 탐지 및 분류 시스템에서 이 같은 동일한 문자열의 존재여부를 점검하는 고려하지 않을 수 없다. 하지만 이런 문자열 비교를 통한 악성코드 탐지는 조금만 시간을 들여도 쉽게 피해갈 수 있다.

반면 악성코드가 데이터를 송수신하기 위해 접근하는 특정 호스트 문자열(URL)은 단순 문자열을 조작하는 일보다는 번거롭다. 호스트와의 통신을 위해 GET 혹은 POST로 전송하는 파라미터들은 그 이름이나 형태를 변경하기 위해서는 서버 프로그램 자체를 수정해야 하기 때문이다. 즉, 'http://', 'GET ', 'POST'와 같은 문자가 포함된 문자열의 경우 Levenshtein Distance의 결과를 (그림 3)과 같은 변형 Jaro - Winkler 수식으로 점수화해서 상호 문자열에 대한 유사성을 측정한다[3][4]. 그리고 일정 임계점 이상으로 유사한 문자열의 경우 각각을 동일한 문자열로 분류함으로써 문자열 비교 결과에 대한 정확도를 높인다.

$$dj = \frac{1}{2} * (m / [S1] + m / [S2])$$

$$dw = dj + 0.1 * 4 (1 - dj)$$

S1, S2 = 문자열
m = S1과 S2 간에 부합되는 문자들의 총 수

(그림 3) 변형된 Jaro-Winkler Distance

악성코드가 접근하는 호스트는 악성코드의 확산을 막기 위해서 빠르게 차단되는 게 최근 추세이다. 때문에 악성코드 제작자들은 악성코드가 접근하는 호스트 이름만 변경한 후 동일한 악성코드(변종)를 재배포한다.

4. 실험 결과

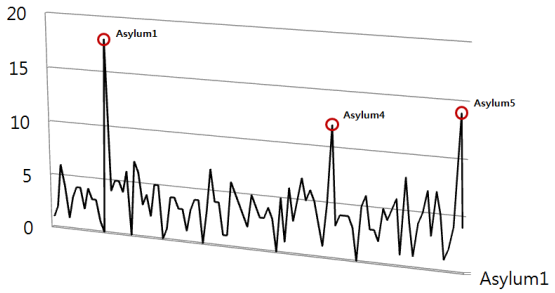
문자열 비교를 통한 악성코드 탐지 및 분류 시스템의 성능을 측정하기 위해 알려진 악성코드 1만개에서 100개의 테스트 집단을 구성하였다[5][6]. 그 중 시스템의 입력값으로 선택된 악성코드 Backdoor.Win32.Asylum의 변종

5가지를 포함시켜 실험에 임하였다. 선택된 Asylum의 분류명은 [표 2]와 같다. 표의 악성코드 분류명은 카스퍼스키 랩(Kaspersky Lab)의 것을 따르고 있으며 제출날짜는 바이러스 토탈(Virus Total)에 기록된 날짜를 의미한다[7].

[표 2] 실험 결과 값을 위해 선택된 악성코드

	Classification (Kaspersky)	Submission date
Asylum1	Backdoor.Win32.Asylum.013.c	2009-12-02 00:44:34 (UTC)
Asylum2	Backdoor.Win32.Asylum.Web.c	2009-12-19 16:12:20 (UTC)
Asylum3	Backdoor.Win32.Asylum.Web.a	2010-02-15 01:48:20 (UTC)
Asylum4	Backdoor.Win32.Asylum.012	2010-01-18 14:47:00 (UTC)
Asylum5	Backdoor.Win32.Asylum.013.e	2009-12-23 02:34:45 (UTC)

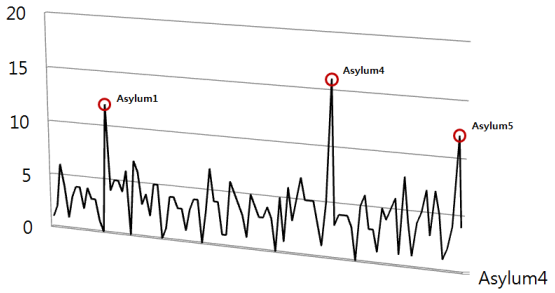
(그림 4)는 시스템에 악성코드 Asylum1을 입력 했을 때의 결과 그래프이며, (그림 5)과 (그림 6)은 각각 Asylum4와 Asylum5를 시스템에 입력했을 때의 결과 그래프이다. 그래프의 가로축은 실험에 사용된 100개의 악성코드를 의미하며 세로축은 시스템의 출력 값(높을수록 유사함)을 뜻한다. 이들 그래프에 따르면 Asylum1, Asylum4, Asylum5는 서로 유사하다는 것을 확인 할 수 있다. 반면 Asylum2와 Asylum3은 유사하지 않은 것으로 나타났는데 이는 오히려 정확한 결과이다. Asylum2와 Asylum3은 [표 2]에서 보는바와 같이 변종 중에서도 Web이라는 구분 명을 가진 Asylum1, Asylum4, Asylum5와는 다른 형태의 변종이기 때문이다.



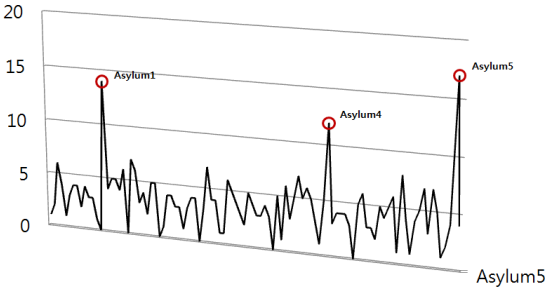
(그림 4) Asylum1에 대한 결과 그래프

5. 결 론

악성코드 탐지 및 분류를 자동화하기 위해서는 다양한 정적 분석 기법이 연구되어야 한다. 본 논문에서 제안된 문자열 비교를 통한 실행파일의 유사도 측정 기법은 하나의 정적 분석 기술로 사용될 수 있다. 하지만 더욱 정확하고 범용성 있는 시스템을 만들기 위해서는 다양한 정적 분석 기법을 병렬적으로 연결해 사용해야 하며 더 나아가 동적 분석기법과의 연동을 고려해야 할 것이다. 또한 각각의 알려진 세부적인 분석기법에 있어서도 지속적인 연구가 수행되어야 할 것이다.



(그림 5) Asylum4에 대한 결과 그래프



(그림 6) Asylum5에 대한 결과 그래프

참고문헌

[1] Symantec, "2009 Internet Security Threat Report", http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xv_04-2010.en-us.pdf

[2] C. Cifuentes and A. Fraboulet, "Intraprocedural static slicing of binary executables", Software Maintenance, International Conference, pages 188 - 195, 1997.

[3] Vladimir Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals", Soviet Physics Doklady, Vol. 10, p.707, 1966

[4] W.E. Winkler, "The State of Record Linkage and Current Research Problems", Technical Report RR/1999/04, Statistical Research Report Series, 1999.

[5] Offensive Computing, http://www.offensivecomputing.net

[6] VX Heavens, http://vx.netlux.org

[7] Virus Total, http://www.virustotal.com