Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.25, NO.5, Oct. 2015 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2015.25.5.1021

코드 필터링 기법을 이용한 iOS 환경에서의 패치 분석 방법론*

조 제 경,† 류 재 철‡
충남대학교

Efficient method for finding patched vulnerability with code filtering in Apple iOS*

Je-gyeong Jo,† Jae-cheol Ryou‡
Chung-Nam National University

요 약

피싱 피해의 확대에 따라 정부 및 기관의 대응이 빨라지면서 피싱 공격은 더욱 발전하여 악성코드 및 취약점 활용에까지 이어지고 있다. 최근 마이크로 소프트의 패치가 발표될 때 마다 해당 취약점을 이용한 공격 기술이 공개되고 이를 악용하는 사례가 늘어나고 있다. 따라서 방어하는 입장에서도 패치를 분석하고 대응하기 위한 기술의 수요가 증가하였으며, 이로 인하여 다양한 패치 분석 방법론이 등장하였다. 하지만 이는 마이크로 소프트 제품에 맞춘 경우가 많으며 모바일 환경을 대상으로 이루어지지는 않았다. iOS와 같은 모바일 장치의 경우 운영체제를 구성하는 파일의 크기가 작고 개수가 많기 때문에 빠르게 비교해주는 기능이 필요하다. 따라서 본 연구는 기존의 연구에서 사용하던 Control Flow Graph나 Abstract Syntax Tree 방법이 아닌 기계어 코드에 최적화된 코드 필터링 방법을 이용한 패치 비교 방법론을 연구하여 안전한 모바일 환경을 구축하기 위한 기반을 마련하고자 한다.

ABSTRACT

Increasing of damage by phishing, government and organization response more rapidly. So phishing use malware and vulnerability for attack. Recently attack that use patch analysis is increased when Microsoft announce patches. Cause of that, researcher for security on defense need technology of patch analysis. But most patch analysis are develop for Microsoft's product. Increasing of mobile environment, necessary of patch analysis on mobile is increased. But ordinary patch analysis can not use mobile environment that there is many file and small size. So we suggest this research that use code filtering instead of Control Flow Graph and Abstract Syntax Tree.

Keywords : Smart Phone, iOS, Patch, Vulnerability, Code Filtering

I. 서 론

시스템 및 응용 소프트웨어는 초기에 개발된 제품 그대로를 유지하는 경우는 드물다. 해킹 및 바이러스의 공격에 대비하여 보안 패치가 이루어지게 되며 이로 인하여 추가적인 파일이 생성되거나 디스크/시스템 이미지가 만들어지게 된다. 대표적인 경우로 PC형 운영체제의 Microsoft Windows와 모바일 운영체제의 Android, iOS가 있다. 대부분의 개발사가 패치로 인한 변경을 공지하고 자동 업데이트를 권고하고 있다. 하지만 다수의 사용자가 성능, 안전성,

접수일(2015년 5월 26일), 수정일(1차: 2015년 7월 30일), 게재확정일(2015년 8월 19일)
* 본 연구는 한국연구재단 운영체제 안전성 연구과제(NRF-2014M3C4A7030648)의 일환으로 수행하였습니다.
† 주저자, oroi@cnu.ac.kr
‡ 교신저자, jcryou@home.cnu.ac.kr(Corresponding author)

루팅 및 탈옥 등의 이유로 자동 업데이트를 수행하지 않는다. 따라서 패치가 발표된 후 악의적인 공격자는 패치를 분석하여 공격 기술로 활용한다. Windows 제품군의 경우에는 이러한 사건이 자주 발생하여 Windows 제품군이 패치되는 화요일의 하루 뒤인 수요일에 악의적인 공격이 발생한다하여 Patch Tuesday, Exploit Wednesday 라는 말까지 등장하였다[1]. 악의적인 공격이 발생할 경우 보안팀은 트래픽을 차단하는 "소 잃고 외양간 고치기" 방식을 반복하고 있다. 따라서 최근 패치 정보를 분석하여 선방어 시스템을 구축하는 기술이 발전하기 시작하였다. 대표적인 경우로 Windows에 대한 취약점 리포트 서비스(Vupen, Telus 등)를 구매하여 해당 보고서의 공격기술을 차단하는 등으로 발전하고 있다. 하지만 PC 뿐만 아니라 사용자층이 급증하고 있는 모바일 환경에서도 이러한 선방어 시스템 구축이 시급하다. 구글의 안드로이드는 소스코드가 공개되어 있기에 패치 내용에 대한 비교가 수월한 반면 애플의 iOS는 코드가 공개되어 있지 않아 패치 내용에 대한 비교가 수월하지 않다. 특히 iOS의 경우 패치가 되지 않은 파일의 경우에도 재컴파일을 통해 주소 값이나 상수 값이 변경된 상태에서 배포된다. 따라서 소스코드가 공개되어 있지 않은 실행파일에 대한 코드 유사도 연구가 필요하다. 이에 따라 기존의 소스코드가 공개되지 않은 Windows를 대상으로 이루어진 코드 유사도 연구를 분석하고 iOS 환경에 최적화된 코드 유사도 연구를 수행하고자 한다.

II. 관련 연구

기존 소프트웨어 패치에 따른 코드 유사도 연구는 다른그림, Bindiff, Diaphora 등 다양하게 진행되고 있다. 기존의 단순 해쉬(CRC32, MD5)값 비교 연구에서부터 시작하여 소프트웨어 공학에서 저작권을 보호하기 위한 코드 재사용(Code Reuse) 탐지 연구를 이용하는 등 다양하게 발전하고 있다. 최근 관심을 받고 있는 코드 유사도 연구 기술로는 실행 코드의 흐름이나 함수간의 흐름(Control Flow Graph) 등을 분석하는 방법과 함수 내의 명령어나 주로 사용되는 커널의 기능 등을 나열하여 트리를 만들어 비교하는 Abstract Syntax Tree 기술 등이 있다. 본 연구에서는 앞서 설명한 알고리즘을 이용한 기존 솔루션을 분석하고 발전 방향을 연구하고자 한다.

2.1 다른 그림

"다른 그림"은 Microsoft 제품군이나 Windows 환경에서 작동하는 소프트웨어의 패치를 분석하여 바뀐 함수를 찾아내는 연구로 현재 "다른 그림 4"까지 공개하였다[2]. IDA Pro라는 디스어셈블 프로그램의 플러그인을 이용하여 작동한다. 실행 파일의 디스어셈블은 IDA Pro를 이용하고 유사도 비교는 자체 개발한 프로그램을 이용하여 수행한다. 유사도 비교를 위하여 Control Flow Graph(이하 CFG) 비교 방법론을 사용하고 있으며 본 연구의 방향인 함수의 해쉬(Hash)를 이용한 방법과는 방법론적 측면에서 속도 및 정확성에 많은 차이를 보여줄 수 있다.

2.2 Bindiff

Bindiff는 Zynamics 회사에서 개발한 솔루션으로 IDA Pro와 Java를 이용하여 작동하는 형태를 가지고 있다[3]. Bindiff는 유료 솔루션으로 그래프(Graph) 기반의 비교를 수행하는 것으로 알려져 있다. 디스어셈블은 IDA Pro를 이용하며 사용자에게 보여주는 형태는 Java를 이용하한 GUI 형태로 보여주고 있다. Bindiff 역시 그래프 기반의 유사도 비교 연구로써 속도가 느리다는 단점을 가지고 있다.

2.3 Diaphora

Diaphora는 인터넷에 공개된 코드 유사도 비교 도구로써 이 역시 IDA Pro를 이용하는 플러그인 형태를 가지고 있다[4]. 최근에 공개된 기술로써 Sqlite에 비교를 원하는 파일의 정보를 저장하고 비교하는 형태를 가지고 있기에 본 연구와 유사하다고 보여질 수 있으나 비교하기 위한 함수의 정보를 추출하는 방식으로 Abstract Syntax Tree(이하 AST) 알고리즘을 사용하는 것으로 알려져 있다. 일반적인 어셈블리 비교에서는 제안하는 연구가 향상된 결과를 나타내었으며 Hexray라 불리는 디컴파일러 플러그인 기능과 AST 알고리즘을 이용할 경우 제안하는 연구보다 10%정도 나아진 결과를 보여주었다. 하지만 AST와 Hexray를 사용하는 것은 속도가 느려질 수 있다는 단점을 가지고 있기에 임베디드 환경에 적합하지 않은 문제점이 있다.

앞선 기존 연구의 대부분이 CFG 기반의 연구로써 비교 결과는 우수하지만 흐름만 유사할 뿐 실제

코드상의 내용이 틀린 경우가 발생하거나 속도가 느리다는 단점이 있다. iOS 시스템에는 3~400여개의 파일의 존재하고 있기에 속도가 느릴 경우 비교 결과를 추출하는데 많은 시간을 소모해야 한다는 단점이 있다. 따라서 본 연구에서는 기존의 대용량 악성코드 그룹화를 위하여 연구개발한 코드 필터링 기반의 코드 유사도 기법 연구[5]를 응용하여 iOS 환경에 최적화된 코드 유사도 연구를 수행하고자 한다.

III. 패치 파일 수집 방법 연구

소프트웨어 라이센스(Software License) 보호를 위한 유사도 비교대상 파일 수집에는 소프트웨어를 구매하거나 인터넷에 있는 소프트웨어를 다운로드 하는 형태로 수집이 가능하며, 악성코드 그룹화를 위한 비교에 사용될 파일의 수집에는 바이러스 샘플을 제공하는 사이트인 Virustotal, Virusshare 등을 이용하여 수집한다. 하지만 본 연구를 위한 iOS 장치의 파일을 수집하기 위해서는 Firmware나 디스크 이미지 덤프를 통하여 수집을 수행하여야 한다.

3.1 Firmware를 통한 패치 파일 수집

PC에 설치하는 Windows나 Linux에서는 사용자가 원하는 프로그램을 설치하거나 실행하는데 제약이 없기에 패치된 파일의 수집에 문제가 없다. 하지만 iOS 장치는 탈옥하지 않는 이상 수집 프로그램을 설치할 수 없기에 이를 해결하기 위한 방법으로 펌웨어(Firmware) 분석을 수행하여야 한다.

iOS의 경우 펌웨어 파일이 ZIP 포맷으로 이루어져 있으며 펌웨어 파일 안에는 암호화된 시스템 파일이 저장되어 있다. 시스템 파일은 AES256 알고리즘으로 암호화 되어 있으며 복호화를 위한 키(Key)는 공개되어 있지 않다. 하지만 부트롬(Bootrom) 취약점에 의해 아이폰 4s까지의 키는 공개되어 있으며 인터넷에 각종 아이폰 연구 사이트에 공개되어 있다. 따라서 펌웨어 파일을 ZIP 알고리즘을 통해 압축해제 한 후 AES256 알고리즘과 부트롬 취약점에 의해 추출한 키를 이용하여 시스템 파일을 추출할 수 있다.

시스템 파일의 경우 OSX에서 사용하는 HFS 파일 시스템 형태를 가지고 있으며 OSX에서 지원하는 DMG 마운트(Mount) 도구를 이용하여 실제 패치된 파일을 추출할 수 있다.

하지만 펌웨어 파일의 복호화 방법은 현재까지 부트롬 취약점이 알려진 4s 이하의 장치에 대해서만 사용이 가능하다. 4s 이후에 출시된 제품에 대해서는 새로운 부트롬 취약점이 발견되지 않는 이상 탈옥을 통한 디스크 이미지 덤프에 의존할 수 밖에 없다.

3.2 디스크 이미지 덤프를 통한 패치 파일 수집

펌웨어 암호화 키가 추출이 되지 않는 장치에 대해서는 탈옥을 이용한 디스크 장치 파일의 접근 방법을 이용하여야 한다. 방어를 위해 공격기술을 이용하여야 한다는 역설적 부분이 존재하나 애플(Apple)의 폐쇄적 정책상 소스 코드가 공개되지 않고, 시스템 이미지 역시 공개하지 않기에 보안정책을 우회하여 시스템 정보를 추출하여야 한다. 보안정책을 우회한 탈옥 기술을 이용할 경우 사용자가 원하는 도구를 설치할 수 있으며 이는 Linux/BSD 계열에서 작동하는 디스크 덤프(DiskDump) 도구를 iOS 장치에 맞추어 컴파일 함으로써 해결할 수 있다. iOS 장치의 경우 시스템 파일이 저장되는 영역은 Read Only로 마운트되며 작동 중인 장치에서 파일을 추출하여도 변경되지 않은 순수한 파일을 추출 할 수 있다.

3.3 패치 파일 정보의 저장

iOS 장치의 펌웨어 파일을 복호화 하거나 탈옥을 통한 디스크 이미지 추출 및 DMG 마운트를 통하여 수집한 파일은 데이터베이스에 1차적으로 파일 정보가 저장된다. 저장되는 파일의 정보는 경로, 파일 전체 해쉬(MD5), 버전 정보이며 추후 파일간의 비교를 위하여 사용하게 된다.

IV. 코드 필터링 및 비교 모델 연구

앞서 설명한 패치 파일 수집 단계에서는 기존 도구 및 기존 개념을 수행하였지만 이는 본 연구단계를 위한 필수요소로써 진행한 것이다. 본 연구의 핵심 부분으로 패치된 파일의 함수간 비교를 위한 코드 필터링(Code Filtering) 단계를 거치고 비교하고자 한다.

4.1 코드 필터링 모델

iOS의 시스템 및 응용 소프트웨어는 실행 가능한

형태의 파일이며 기계어 코드로 구성되어 있다. 이러한 기계어 코드는 단순한 재 컴파일만으로도 위치 값(상수)이 바뀌며 변수 및 조건의 변경으로도 코드의 위치가 바뀌면서 코드를 가르키는 상수 값이 바뀐다. 따라서 본 연구에서는 기계어 코드내의 상수 값으로 표현되는 부분을 변수로 지정하였다. 변수의 대부분은 오퍼랜드라 불리는 기계어 코드의 연산자 다음에 위치하거나 레지스터 옆에 위치하게 된다. 이러한 변수 값을 삭제하여 함수의 코드 형태와 흐름에 대한 데이터를 유지한 채 해쉬(MD5) 값을 생성한다.

코드의 흐름을 유지하기 위해서는 단순히 기계어 코드의 변수 값을 전부 삭제하는 것이 아니라 변수의 저장 공간으로 이용되는 Register(이하 레지스터)는 유지하는 형태를 가지도록 하였다. 숫자로 이루어진 변수는 삭제를 하되, 레지스터는 유지하는 형태를 가지도록 한 것이다. 레지스터가 변수를 저장하는 용도로 사용되나 변경되는 경우는 매우 적기에 삭제될 경우 기계어 코드의 오퍼랜드만 남아 중복성이 다수 나타난다.

이렇게 상수 값이 제거된(필터링) 함수내용을 MD5 해쉬 값으로 연산하여 저장한다. 이 과정에서 함수의 이름이 존재하는 경우 함수의 이름도 함께 저장하게 된다.

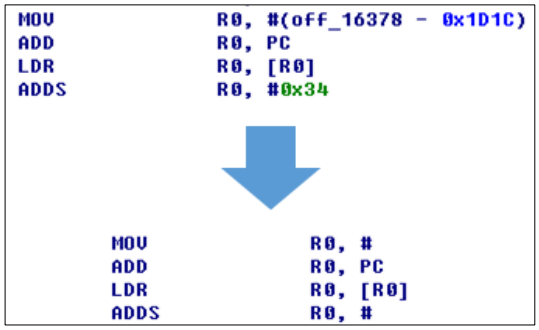

Fig. 1. Example of Code Filtering

4.2 코드 비교 모델

비교 하는 부분은 3가지 기준을 가지고 비교를 하게 된다. 우선 설치된 장치의 버전과 시스템 및 응용 소프트웨어의 버전을 확인하게 된다. 그리고 두번째로 파일 이름이 포함된 경로를 비교하게 된다. 파일 이름은 다른 소프트웨어 간에 동일할 수 있기에 경로를 포함하게 된다. 그리고 해당 파일에 맞추어져 있는 함수의 해쉬 리스트를 추출하여 비교하는 과정을 가지게 된다. 앞의 코드 필터링 모델에서 해쉬 값을 저장할 때 함수의 이름을 저장할 수 있도록 하였기 때문에 서로 다른 해쉬 리스트를 추출했을 경우 함수 이름을 기준으로 한번 더 비교를 수행할 수 있다.

함수의 해쉬 값이 다르면서 함수 이름이 같을 경우에는 패치된 함수로 확신할 수 있으며, 이름이 없거나 매치되지 않을 경우 코드를 실제 비교하기 위한 작업을 수행하게 된다.

V. 실 험

제안하는 연구 방법을 iOS 7.1.2와 iOS 8.0을 비교하는데 적용하였다. iOS 7.1.2 에서는 320개의 실행파일을 iOS 8.0에서는 415개의 실행파일을 추출하였다. 하지만 iOS 8.0의 415개 파일 중 iOS 7.1.2와 동일한 파일은 단 7개만 존재하였으며 기존 파일의 변경된 내용만 추출하기 위한 기법은 적용이 불가능하다는 것을 확인할 수 있다.

iOS 7.1.2와 8.0의 Syslogd 파일에 대해 함수 비교를 실제 수행해 본 결과 총 574개(8.0은 577개)의 함수가 있으며 파일 전체의 해쉬 값도 다르지만 함수 간의 해쉬 값 또한 다른 것으로 확인하였다.

Table 1. File hash compare result

File of iOS 7.1.2	File of iOS 8.0	Same File of iOS 7.1.2 & 8.0
320	415	7

Table 2. Function hash compare result

Version	Compare Type	Similar Function
7.1.2 vs 8.0	File Hash Compare	ALL Different
7.1.2 vs 8.0 without CodeFiltering	Func Hash Compare (Syslogd)	199 / 574
7.1.2 vs 8.0 with CodeFiltering	Func Hash Compare (Syslogd)	400 / 574
7.1.2 vs 8.0 without CodeFiltering	Func Hash Compare (ALL Files)	1,668,306 / 2,087,417
7.1.2 vs 8.0 with CodeFiltering	Func Hash Compare (ALL Files)	1,818,424 / 2,087,417

Table 3. Diaphora vs Our Solution

Solution	Similar Function (Syslogd)
Diaphora (Equal Assembly)	210 / 574
Diaphora (With Hexray)	375 / 574
Our Solution	400 / 574

필터링 과정이 개입되지 않은 경우 574개 함수 중 375개의 함수가 서로 다른 것으로 나타났으며 필터링 과정이 개입되었을 경우 174개의 함수가 서로 다른 것으로 나타났다.

기존 연구와의 비교를 위해 가장 최근에 공개된 Diaphora와 비교하였을 경우에 다음과 같은 결과가 나타났으며 본 연구의 긍정적 가능성을 보여주었다.

VI. 결 론

본 연구의 방법론은 기존의 다양한 악성코드를 이용하여 검증을 1차적으로 수행한 만큼 중복성의 문제는 많이 줄어든 장점이 있다. 하지만 악성코드가 아닌 시스템 파일의 경우 함수의 개수가 더 많고, 짧은 코드로 이루어진 함수가 더 많은 것을 확인하였다. 따라서 기존의 악성코드 유사도 연구에서 사용하던 코드 필터링 기법을 발전시켜 함수의 이름, 위치 정보 등을 이용하여 iOS에 최적화된 방법론을 찾아낼 수 있게 되었다.

본 연구는 기존의 Diaphora와 같은 CFG 및 AST 기반의 유사도 비교 연구보다 함수의 변경된 부분만을 표현하기에는 부족한 부분이 있다. 하지만 코드 유사도 비교 자체만으로는 〔Table 3.〕과 같이 보다 나은 성능을 보여주고 있다. 따라서 본 연구를 통하여 임베디드 및 Closed Platform 에서의 취약점 연구를 위한 기반을 마련하였다고 볼 수 있다. 향후 기존의 도구의 함수 내 변경된 부분을 표현하거나 유사도 비교 탐지율을 더욱 발전시켜 iOS 뿐만 아니라 임베디드 및 Closed Platform에 최적화된 코드 유사도 연구를 수행하고자 한다.

References

〔1〕 Dale G Peterson. "Patch Tuesday leads to exploit Wednesday," Digital Bond, Oct. 2009

〔2〕 Matt Oh, "Exploit Spotting : Locating Vulnerabilities Out Of Vendor Patches Automatically," Blackhat USA, Jul. 2010

〔3〕 Dullien, Thomas, and Rolf Rolles. "Graph-based comparison of executable objects," SSTIC 5, 2005

〔4〕 Diaphora, "https://github.com/joxeankoret/diaphora"

〔5〕 Chan-Kyu Park, Hyong-Shik Kim, Taejin Lee, Jae-Cheol Ryou, "Function partitioning methods for malware variant similarity comparison," Journal of The Korea Institute of Information Security & Cryptology, pp.321-330, Apr. 2015

〈저 자 소 개〉

조 제 경 (Je-gyeong Jo) 학생회원
2006년 2월: 한신대학교 정보시스템공학 졸업
2008년 8월: 한신대학교 컴퓨터정보학 석사
2014년 3월~현재: 충남대학교 컴퓨터공학 박사과정
〈관심분야〉 정보보호, 시스템보안, 네트워크보안

류 재 철 (Jae-cheol Ryou) 종신회원
1985년 2월: 한양대학교 산업공학과 졸업
1988년 5월: Iowa State University 전산학 석사
1990년 12월: Northwestern University 전산학 박사
1991년 2월~현재: 충남대학교 컴퓨터공학과 교수
〈관심분야〉 정보보호, 네트워크보안, 암호학, 보안프로토콜