바이너리 취약점 분석을 위한 파일 포맷 분석기 구현

오동엽[O], 류재철[*]
[*]충남대학교 컴퓨터공학과
[O]한국과학기술원 사이버보안연구센터
e-mail: oh51dy@kaist.ac.kr[O], jcryou@cnu.ac.kr[*]

Implementation of file format analyzer for binary vulnerability analysis

DongYeop Oh[O], Jea-cheol Ryu[*]
[*]Dept. of Computer Engineering, Chung-Nam National University
[O]Cyber Security Research Center, KAIST

● 요 약 ●

최근 PC를 비롯한 모바일, IOT 기기 등 다양한 환경에서의 사이버 공격이 기승을 부리고 있으며, 그 방법 또한 나날이 발전하고 있다. 이러한 사이버위협으로부터 개인 및 기업의 자산을 지키기 위한 근본적인 대안이 없이는 매번 반복적인 피해를 피하기 어려운 현실이다. 다양한 환경이라고 함은, 다양한 OS(Operation System), 다양한 ISA (Instruction Set Architecture)의 조합으로 이루어지는 사이버환경을 의미한다. 이러한 조합들은 일반 사용자들에게 가장 많이 쓰이는 Windows & Intel 조합의 환경과, Linux & Intel 또는 Linux & ARM 등 기업에서 서비스를 위해 쓰이는 서버 환경 등을 예로 들 수 있다. 그밖에 최근 IOT기기나 모바일 기기와 같은 환경도 있을 수 있다. 바이너리 파일에 대한 보안은 다양한 연구가 진행되고 있지만 그 범위가 방대하고, 깊이가 필요한 영역이라 진입 장벽이 높은 실정이다. 본 논문에서는 이러한 바이너리의 취약점을 분석하기 위한 첫 번째 단계로써 다양한 바이너리 파일을 하나의 정형화된 자료구조로 변환하는 바이너리 포맷 분석기의 한 방법을 제시하고자 한다. 다양한 OS와 다양한 ISA환경에서 사용되는 바이너리들에서 공통적으로 존재하는 정보들 중, 바이너리의 취약점 분석을 위해 필요한 데이터를 보다 효율적으로 수집하고, 관리하는 것이 바이너리를 통한 사이버 위협을 탐지하는 연구에서 기초가 된다고 할 수 있기 때문이다.

키워드: 파일포맷(file format), 바이너리(binary), 취약점(vulnerability), 운영체제(OS)

I. Introduction

최근의 사이버 보안 위협은 다양한 환경에서 다양한 형태로 이루어지고 있으며, 그에 따른 대응 방안 또한 다양해지고 있는 추세이다. 그렇기 때문에 보다 많은 보안 인력이 필요하며 다양한 방식의 방어기법들이 등장을 하게 되었다.

현재는 소스코드 분석을 통한 취약점 탐지, 바이너리 기반의 패턴매칭을 이용한 정적 분석기법이 있고, 실제 바이너리 파일을 실행시켜서 취약점을 찾는 퍼징 기법 등의 동적 분석 방법 등이 바이너리를 통한 사이버위협의 방어 수단으로 널리 사용되고 있는 상태이다.

현재 오픈소스를 통해 많은 수의 바이너리 파일들의 소스가 공개되어져 있어 소스코드를 통한 취약점 분석이 용이해진 부분이 있다. 그러나 바이너리파일이 아닌 소스코드를 통한 취약점 분석 기법은 분명히 한계 점을 가지고 있다.[1]

그 첫 번째로 아직도 소스코드가 공개되지 않는 바이너리파일들이 많이 사용되고 있다. 소스코드가 확보된 바이너리들로 그렇지 못한 케이스의 바이너리를 전부 대체하는 것은 사실상 불가능하다고 여겨지며, 혹여 소스코드가 확보되는 케이스가 많아진다고 하더라도 여전히 소스코드를 모르는 바이너리는 계속해서 생성될 것이기 때문이다. 실 예로 군사 목적 등으로 개발되는 소프트웨어나 악의적인 목적으로 생성되는 악성바이너리의 경우는 해당 바이너리의 개발자가 소스코드를 직접 공개하기 전에는 이를 확보하고 분석하는 것이 불가능에 가깝기 때문이다.

또한, 실재로 바이너리파일의 실행 결과는 소스코드 분석으로 예측되는 실행 결과가 다른 경우가 존재한다. 이는 소스코드 상태에서 컴파일러를 통해 기계어로 변환되는 과정에 악의적인 이유로 취약점을 일으키는 코드를 삽입하거나 혹은 최적화과정 중에 악의적이지 않은 이유로 코드를 수정하는 경우가 발생되고, 그에 따라 예상하지

못한 실행 결과를 보여줄 수 있기 때문이다.[2] 따라서 소스코드 분석 보다는 실재로 실행을 하는 주체인 바이너리 파일을 분석하는 것이 중요하다.

II. Preliminaries

본 논문에서는 다양한 OS 다양한 바이너리 파일 포맷 중 Linux, Windows, MacOS에서 기본 실행파일인 ELF, PE, MACH-O을 분석 대상 바이너리로 선정하였다.

1. Related works

1.1 ELF

ELF(Executable and Linkable Format)는 실행 파일, 목적 파일, 공유 라이브러리 그리고 코어 덤프를 위한 표준 파일 형식이다. 1999년 86open 프로젝트에 의해 X86 기반 유닉스, 유닉스 계열 시스템의 표준 바이너리 파일 형식으로 선택 되었다. ELF 형식은 다양한 환경에서 오래된 실행 파일 포맷들을 대체 하였으며, 현재 Linux, FreeBSD, 솔라리스등 다양한 범용 운영체제에서 표준바이너리로 쓰이고 있을 뿐 아니라, 플레이스테이션등 몇몇 게임 콘솔에서도 사용이 된다. ELF는 ELF Header와 파일 데이터로 이루어지며, 32Bit 64Bit에 따라 Header의 형태가 일부 상이하다. [3]

ELF는 다양한 운영체제에서 널리 사용되는 포맷인 만큼 분석 유틸리티 또한 다양한 형태로 존재한다. 대표적으로 유닉스 바이너리 유틸리티 중에서는 ELF 파일헤더의 정보를 보여주는 readelf가 있으며, objdump의 경우 ELF 바이너리의 역어셈블까지 지원한다.

1.2 PE

PE(Portable Executable)는 우리가 주로 사용하고 있는 윈도우 환경에서의 실행 파일 포맷이며, exe을 비롯하여, sys, dll등의 확장자로 생성 된 파일들을 포함한다. PE 파일 역시 ELF파일과 같이 PE Header영역이 존재하며, 차이점으로는 PE Header 보다 앞서서 Dos Header Stub Code 영역이 존재한다. 그리고 PE Header이후 Section Header 들이 나오고 이어서 Section들이 이어진다.[4]

PE파일은 ELF나 Mach-O 바이너리와 다르게 Symbol에 대한 정보를 포함하고 있지 않는다. PE파일의 경우는 PDB(Program database file)이라고 하는 Debug Info 파일이 별도로 존재하며, 해당 파일을 같이 분석해야 해당 바이너리의 Symbol 정보를 얻을 수 있는 구조이다. 그러나 일반적으로 PE 파일들은 PDB 파일 없이 홀로 배포되는 경우가 대부분이다.

1.3 MACH-O

Mach-O는 Mach 오브젝트 파일형식의 줄임말로써, 실행파일, 오브젝트 코드, 라이브러리 등의 형식을 이야기한다. Mach-O는 Mach 커널을 기반으로 하는 대부분의 시스템에서 사용된다.

Mach-O의 구성은 하나의 Mach-O header와 SymTab Command, Main Command, Segment Command등 복수개의 Command로 구성되고, Segment Command 아래 __TEXT, __DATA 등의 이름으로 Section들이 존재한다.[5] Mach-O 바이너리의 경우 otool 이라고 하는 툴을 이용하여 바이너리 분석이 가능하다.

III. The Proposed Scheme

이번 장에서는 다양한 바이너리 포맷에서 바이너리 분석을 위해 필요한 정보들을 효율적으로 수집하고, 관리하는 방법에 대해 제안하고자 한다. 대상은 Intel, ARM, MIPS 3개의 ISA와 ELF, PE, MACH-O 3개의 OS별 파일 포맷 중 Intel, ARM과 ELF, PE, MACH-O의 조합으로 한다.

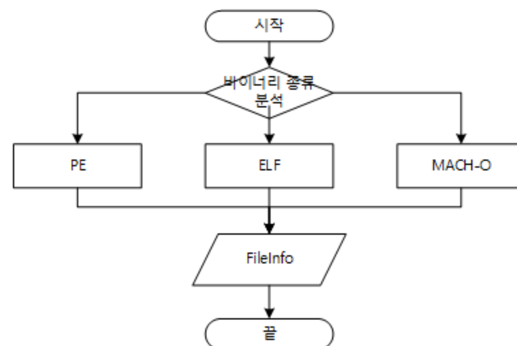

Fig. 1. Model WorkFlow

각각의 다른 입력 바이너리 파일의 종류를 구분한 후, 하나의 통합된 FileInfo 자료 구조로 변환을 하고, 해당 자료 구조를 활용하여 역공학을 진행하는 순서로 진행된다.

3.1 개발환경

바이너리 파일 형식들은 각각 다른 OS 환경에서 실행이 된다. 그러나 정적분석기법을 통해서 바이너리를 분석 할 경우 실제로 실행을 하지 않기 때문에 OS에 종속적이지 않게 분석이 가능하다. 또한 분석가의 작업환경에 따라 각각의 OS환경에서 분석이 가능하도록 하는 것이 필요하다. 이를 위하여, 본 논문에서 구현한 바이너리 포맷 분석기는 MS사에서 만든 F#을 이용하여 구현하였다. F#을 이용하는 이유로 .net 플랫폼을 이용하여 다양한 OS를 지원하는 언어이기도 하며, side effect를 최소화 하는 것이 중요하기 때문이다. 또한 matching 함수가 빈번하게 발생하는 프로그램을 구현해야 하는 부분과 함수를 단순화하여 코드 재사용에도 유리한 장점이 있는 언어이기도 하기때문이다. 이를 위하여 함수형 언어인 F#을 이용하여 구현하였다.

3.2 Binary Format

바이너리 파일은 각 OS환경에 따라 다양한 포맷을 가지고 있다. 물론 각각의 바이너리 파일에서 Text Section 만을 추출하여 Raw Binary 형태로 역어셈블하여 바이너리를 분석 할 수도 있다. 그러나

보다 정확하고 디테일한 분석을 위해서는 바이너리 파일의 Header에 있는 정보를 같이 활용하는 것이 중요하다. Section과 Symbol의 경우 Key를 Address Range로 하고 Value를 각각 Section, Symbol로 하는 Map 형태로 저장하는 것이 좋다. 각각의 정보는 해당 Section, Symbol의 시작과 끝의 주소 범위를 같이 확인하는 형태로 활용되기 때문이다. 바이너리의 Stripped 정보 역시 Symbol 정보를 찾는 방식에 있어 중요한 확인 요소로 활용된다.

다음은 바이너리 파일의 Header 정보 중 취약점 분석을 위하여 필요한 주요 데이터들 이다.

·Entry Point – 프로그램의 시작 주소
·TextStartAddr – text 섹션의 시작 주소
·Section Number – 섹션의 개수
·Symbol Number – 심볼의 개수
·Stripped – symbols 제거 여부
·WordSize – 워드 사이즈 (ex 32bit, 64bit)
·NX Bit – NX 정보

3.3 API

수집된 각각의 바이너리 데이터들을 분석과정에서 효율적으로 활용할 수 있도록 다양한 API를 제공해야한다. 기본적으로 필요한 Entry Point, WordSize와 그 자체로 의미가 있는 데이터들도 있는 반면, Section Range, Symbol Range와 같이 각각의 상호관계를 분석 및 가공해야 얻을 수 있는 데이터들이 존재하기 때문이다. 또한 ARM ISA에서 쓰이는 mapping Symbol과 같이 일부 데이터는 특정 환경 및 특정 파일포맷에서만 필요한 경우도 있다.

예를 들어 ELF 포맷은 ISA에 따라서 특정되는 속성들이 존재한다. 특히 ARM의 경우 mapping symbols이라고 불리는 ARM 전용 Symbol 형태가 존재 하는데 다른 symbol들과 다르게 $a, $t, $d와 같은 이름을 가지고 있으며 이는 바이너리를 역어셈블링할 때 해당 Symbol 영역의 데이터의 해석 방법을 명시하고 있다.[6]

다음은 바이너리파일 분석에 필요한 주요 API들 이다.
·TransAddr – FIle상의 offset과 Address의 변환
·IsValidAddr – 입력 주소 값이 유효한지 검증
·GetSymChunkMode – MappingSymbol 값
·GetSymChunkRange – MappingSymbol 적용범위
·GetFuncAddrs – Symbols의 시작 주소리스트
·GetSecName – 현재 주소에 맞는 Section 이름
·FindSymName – 현재 주소에 맞는 Symbol 이름
·GetLinearTargetSecRanges - linearSweep 방식으로 리버싱을 진행할 때 타깃 되는 Section의 범위

III. Reault and Discussion

본 논문에서 제안한 분석기를 검증하기 위해서 기존에 각각의 환경에서 제공되는 분석 툴들과 비교 검증을 진행하였다. 단순 바이너리 정보 추출 테스트와 실제 리버싱 테스트를 진행하였다. 바이너리 정보 추출 테스트에서의 검증은 ELF의 readelf, PE의 dumpbin, Mach-O의 경우 otool을 비교대상 툴로 선정하였다. 바이너리 정보 추출관련 비교 항목은 3.2절에 언급된 주요 데이터들을 그 대상으로 한다. 검증에 활용된 바이너리의 수가 다소 부족한 면이 있으나 이는 추후 계속해서 추가 검증을 통해 비교하고, 수정 보완해 나갈 예정이다.

Table 1. Test data and result

포맷(32bit & 64bit)	바이너리	결과
PE	C:₩Windows₩System32₩calc.exe 외 10종	일치
ELF	bin/ls 외 10종	일치
MACH-O	bin/ls 외 10종	일치

리버싱 테스트의 경우 IDA Pro, ObjDump의 결과와 본 논문에서 구현한 바이너리 포맷 분석기를 활용하여 별도로 진행 중인 역공학 연구 프로젝트의 리버싱 툴을 비교 검증하였다. 비교 대상은 instruction 별 Address 정보와 Symbol정보가 정확히 리프팅 되는지 태스트를 진행하였다. 특히 Section별 Address의 매칭정보와 instrction 의 해석이 맞는지 체크하였다. Intruction 해석부분파트에서 중점적으로 체크 한 부분은 ARM Mode와 Thumb Mode가 섞여있는 ARM32Bit(Thumb Mode) 바이너리에서 Section별 혹은 Mapping Symbol 이 변경되는 Address마다 정확한 Type의 instruction으로 해석하는지 여부이다.

Symbol의 경우 Static Symbol의 정보와 Dynamic Symbol의 정보가 다른 툴과 일치하는지 검증 하였다. 이 테스트 또한 대상 바이너리는 위의 표1에서와 동일한 바이너리들을 대상으로 진행하였으며, 실험 결과는 정상적으로 확인 되었다.

IV. Conclusions

본 논문은 바이너리 취약점 분석을 보다 효율적으로 진행하기 위한 첫 번째 단계로써 다양한 환경에서 동작하는 바이너리를 하나의 통합된 환경과 자료구조로 1차 가공을 하는 것을 제안하고 있다. 바이너리 포맷 분석은 정확한 바이너리 파일을 분석하기 위한 시작점이다. 분석 대상 바이너리로부터 다양한 바이너리 취약점 분석 기법 방법들에서 활용 할 수 있는 정확한 데이터를 추출하는 것이 중요하다, 특히 각 Section, Symbol들의 정확한 해석과 NXbit, ASLR과 같은 메모리 보호기능의 설정 상태 등 각각의 환경에 맞는 데이터를 추출해야 리버싱을 제대로 진행할 수 있다. 본 논문에 제시된 ELF 바이너리를 비롯하여 PE, MACH-O 바이너리의 통합 포맷분석기를 활용하여 바이너리 취약점 분석연구에 활용 하고 있으며, 지속적으로 수정 보완을 진행 할 예정이다.

ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded

by the Korea government (MSIT) (No.2016-0-00102, Building a Platform for Automated Reverse Engineering and Vulnerability Detection with Binary Code Analysis).

REFERENCES

[1] B. Schwarz, S. K. Debray, and G. R. Andrews, "Disassembly of Executable Code Revisited," In 9th Working Conference on Reverse Engineering(WCRE), pp.45-54, IEEE Computer Society, 2002.
[2] Christopher Kruegel, William Robertson, Fredrik Valeur and Giovanni Vigna, "Static Disassembly of Obfuscated Binaries", the 13th conference on USENIX Security Symposium, pp.18-18, 2004.
[3] http://www.sco.com/developers/gabi/2000-07-17/ch4. eheader.html
[4] https://docs.microsoft.com/ko-kr/windows/desktop/De bug/pe-format
[5] http://developer.apple.com/mac/library/documentation/ DeveloperTools/Conceptual/MachORuntime/Reference/reference.html
[6] Jiunn-Yeu Chen ; Bor-Yeh Shen ; Quan-Huei Ou ; Wuu Yang; Wei-Chung Hsu, "Effective code discovery for ARM/Thumb mixed ISA binaries in a static binary translator", Published in: 2013 International Conferen ce on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013