Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.4, Aug. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.4.721

악성코드의 효율적인 분석을 위한 안전한 오픈소스 함수에 대한 시그니처 기반 식별 도구*

이 석 수,† 양 종 환, 정 우 식, 김 영 철, 조 은 선‡
충남대학교

A Tool for Signature-Based Identification of Safe Open-Source Functions Toward Efficient Malware Analysis*

Seoksu Lee,† Jonghwan Yang, Woosik Jung, Yeongcheol Kim, Eun-Sun Cho‡
Chungnam National University

요 약

악성코드에 대한 빠른 대응을 위해서는 악성코드에 대한 효율적인 분석이 필요하다. 그 중 하나로, 오픈 소스 함수들과 같이 안전한 것으로 확인된 부분을 분석 대상에서 제외하여 방대한 분석 대상을 줄이는 방법이 도움이 될 수 있다. 본 논문은 여러 오픈소스의 동적 링크 라이브러리 파일을 윈도우 환경에서 생성하여 오픈소스의 함수 정보들을 버전별, 컴파일러별로 시그니처 정보를 추출하고 비교하여 변경이 의심스러운 함수를 찾을 수 있는 자동화 도구를 제시한다. 또한 해당 도구는 비교에 사용된 정보들을 DB에 저장, 추후에 사용할 수 있어 분석 시간 오버헤드를 줄일 수 있다.

ABSTRACT

In order to take rapid action against malware, efficiency in malware analysis is essential. For instance, it would be helpful to identify and eliminate open-source function bodies or other safe portions out of the target binary codes. In this paper, we propose an tool to create open source dynamic link library files in Windows environment, extract signature information by opensource and compiler version, and compare open source function information to find suspicious function. In addition, the tool can save the information used in the comparison to the DB and use it later, reducing the analysis time overhead.

Keywords: Open Source, Signature-based Analysis, Opensource Safety Identification

I. 서 론

최근에는 무료로 사용할 수 있고, 개발이 용이한 오픈소스 소프트웨어의 사용이 증가되고 있다. 이에 따라 악성코드에 관련되어서 오픈소스를 활용하는 경우도 증가하고 있다. 분석시간을 지연시키기 위해 악성 행위를 암호화하여 감추는 데에 오픈소스 암호화 관련 소프트웨어를 사용하는 경우가 많이 있다[1]. 이러한 오픈소스 소프트웨어들은 보안 취약성 등의 문제가 있을 수는 있지만 의도적인 악성행위를 담고 있지는 않다. 따라서 이러한 오픈소스 함수의 정의 부분에 대해서는 상대적으로 안전한 코드로 간주하여 악성코드 분석 대상 중에서 우선순위를 낮추어 분석할 수 있다. 현재 활용되고 있는 분석 도구 및 알고리즘은 오픈소스의 버전업과 컴파일러 버전별에 따라서 해당 오픈소스 코드의 안전성을 판단하기에는 최신 정보로 업데이트 하거나 비교하는 과정에서 시간이 많이 걸린다는 단점이 있다.

Received(03. 15 .2017), Modified(05. 10. 2017), Accepted(05. 18. 2017)
* 본 연구는 충남대학교 학술연구비에 의해 지원되었음.
* 본 논문은 2016년도 동계 학술대회에 발표한 우수논문을 개선 및 확장한 것임
† 주저자, 201302452@cs-cnu.org
‡ 교신저자, eschough@cnu.ac.kr(Corresponding author)

본 논문은 안전한 오픈소스 함수를 찾기까지의 분석 시간을 줄여 전체적인 분석 시간을 줄이는 것을 궁극적인 목표로 한다. 기존의 바이너리 분석을 위해 활용되고 있는 도구 및 알고리즘은 위에서 말했듯이 오픈소스의 버전업과 컴파일러 버전별에 따라서 해당 코드의 안전성을 판단하기에는 불충분하거나 비교하는 과정에서 시간이 많이 걸린다는 단점이 있다[2]. 이를 극복하기 위해 본 논문에서 제안하는 도구는 오픈소스 함수의 시그니처를 손쉽게 등록하고, 악성코드 분석 과정에서 시그니처 비교를 편리하게 수행할 수 있도록 하고자 한다. 또한 버전이나 컴파일러 등이 다른 경우에 대해서도 시그니처 기반 오픈소스 소프트웨어 식별이 최대한 올바르게 동작하는 방법을 찾기 위해 다양한 기준으로 비교하였다.

II. 관련 연구

기존의 바이너리를 기반으로 함수를 비교하는 연구는 다양하게 진행되었다[3]. 프로그램 특징을 이용하여 함수를 비교하는 연구는 Graph-edit distance와 Hungarian algorithm 및 기타 비교 알고리즘을 이용하였다[4][5]. 하지만 이러한 비교 알고리즘은 프로그램의 사이즈가 큰 경우 비교 속도가 느리다는 단점이 있다. 이러한 부분을 보완하기 위하여 프로그램 분석 시 사용하는 여러 종류가 있다. 도구 중 BinDiff[6]는 Hex-Rays에서 개발한 IDA Pro(Interactive DisAssembler)[7]의 플러그인으로, 바이너리코드간의 차이점과 유사점을 찾는 도구이다. 주로 악성코드 분석 연구에 사용된다. 그러나 새로운 버전으로 함수가 업데이트 되면 처음부터 새로 비교분석을 수행해야 하고, 프로그램 흐름을 파악하기 어려운 단점이 있다. Fig 1는 BinDiff를 이용하여 OpenSSL[8]을 다른 버전의 컴파일러로 컴파일 한 비교 결과이다. 해당 그림에서 볼 수 있듯이 컴파일러만 다른 2개 파일의 유사도는 10%로, 일치하다고 볼 수 없다는 결론을 내어준다.

Fig. 1. BinDiff Example

FLIRT(Fast Library Identification and Recognition Technology)[9]는 IDA Pro에 내장된 기능으로 IDA Pro를 이용하여 분석된 바이너리 내부의 함수들을 시그니처를 이용하여 매칭 시켜주는 기능을 한다. FLIRT의 기능을 분석자의 마음대로 사용하기 위해서는 시그니처 파일을 생성해야 한다. 시그니처란 해당 파일만이 가지고 있는 고유한 특징으로 사용자가 그 특징을 보는 관점에 따라서 달라 질 수 있다. 일반적으로 사용하는 시그니처로는 다운로드 시에 안전성 여부를 판단하기 위해 사용하는 해시값이 있다. FLIRT에서는 시그니처가 될 라이브러리를 선택하고 선택된 라이브러리에서 패턴을 추출하여 시그니처를 생성, 일치여부를 판단한다. 그러나 사용자가 시그니처 파일을 직접 생성하고 업데이트해야 하는 단점이 있다. 또한 FLIRT는 분석하려는 프로그램 구조가 단순할 경우 오류가 발생하게 된다. Fig 2은 OpenSSL 시그니처를 생성하여 FLIRT도구를 이용하여 비교한 결과와 부분 확대된 그림이다. 시그니처 내용과 비교하여 일치된 함수는 하늘색으로 표시된다.

본 논문에서는 BinDiff와 FLIRT의 단점을 보완하여 비교 대상의 버전과 컴파일러의 변경에도 바이너리 분석의 오버헤드를 최소화 할 수 있도록 도구를 제작하였다.

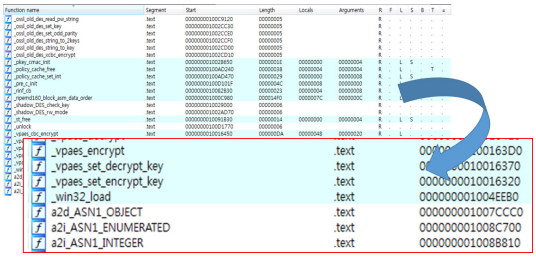

Fig. 2. FLIRT Example

III. 제안 도구

본 논문에서 제안하는 도구는 아래 Fig 3과 같이 오픈소스 DLL(Dynamic-Link Library)추출 도구 하나와 DLL을 통한 함수 시그니처 생성, DB 저장 및 비교해주는 2개의 IDA Pro플러그인으로 구성되어있다. IDA Pro는 Hex-Rays에서 만든 디

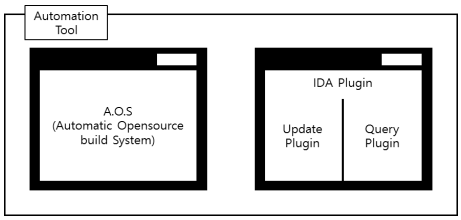

Fig. 3. Automation tool configuration

스어셈블리 도구로 다양한 환경에서의 실행 파일 및 DLL파일을 분석할 때 주로 사용되는 도구이다.

오픈소스 DLL 추출 도구는 일반적으로 다양한 환경에서 사용되는 오픈소스의 DLL파일을 생성하기 위한 도구로 해당 도구를 통해서 오픈소스 DLL파일을 버전별 또는 컴파일러별로 추출 할 수 있다. 함수 시그니처 비교 도구는 디스 어셈블러 프로그램인 IDA Pro의 API를 이용하여 다양한 오픈소스의 버전별, 컴파일러 별 각 함수 정보를 추출하고, DB에 저장하는 플러그인과 저장한 시그니처 리스트를 바탕으로 오픈소스 함수의 안전성을 검사하는 플러그인 2가지로 이루어져있다.

IV. 오픈소스 비교 방법

4.1 비교 실험에 사용된 오픈소스

본 논문의 실험에서 사용된 오픈소스는 OpenSSL, Zlib[10], Crypto++[11]이다. 해당 오픈소스들은 현재 보안과 관련된 다양한 암호 함수 및 네트워크상에서 안전한 연결 서비스 이용할 수 있는 함수들을 제공하고 있다. Zlib은 de facto standard로 인정받는 기본 라이브러리로 사용되며[12], OpenSSL의 경우 악성코드 삽입과 암호화 함수 취약점에 대한 연구가 진행되고 있다[13]. Crypto++의 경우 C++로 제작된 바이너리의 특수성을 확인하기 위한 좋은 오픈소스이다. 따라서 해당 오픈소스들을 실험 대상으로 정하였다.

• OpenSSL

OpenSSL은 네트워크를 통한 데이터 통신에 쓰이는 프로토콜 및 통신 암호화와 관련된 기능을 제공하는 오픈소스이다. C로 구성되어 있으며 현재 최신 버전은 2015년 1월 22일 배포된 1.0.2e버전이다.

• Crypto++

Crypto++는 C++기반의 암호화 라이브러리로 DES(Data Encryption Standard), AES(Advanced Encryption Standard)등의 다양한 암호 알고리즘이 적용된 암호화 함수를 제공한다. 현재 최신버전은 5.6.5버전이다.

• Zlib

Zlib는 압축과 관련된 함수를 제공하는 오픈소스로 리눅스 커널, 서버, 클라이언트 등 다양한 분야에서 활용되고 있다. 버전마다 큰 차이는 없으나 버전이 올라갈수록 상세 옵션에 대한 변경 점과 다양한 플랫폼에 지원될 수 있도록 바뀌고 있다. 현재 최신 버전은 1.2.11이다.

4.2 DLL 비교 방법

본 논문에서는 DLL파일을 IDA Pro플러그인의 형태인 비교도구를 이용하여 각 함수의 시그니처를 추출 및 비교한다. 3가지의 오픈소스의 DLL마다 각 함수의 시그니처를 만들 때 3가지의 정보를 이용하여 시그니처를 만든다. 시그니처는 크게 HEX, Assembly, Mnemonic을 이용하여 만든다. 각각의 함수에 관련된 정보를 위의 3가지 값을 이용하여 시그니처를 뽑아서 비교한다. 각각의 장점과 단점은 아래와 같다.

• Hexadecimal code

HEX(Hexadecimal code)는 바이너리 그대로의 값으로 프로그램의 변화에 큰 영향을 받는 값이다. 따라서 HEX를 기반으로 생성된 시그니처가 일치된다면 변경점이 없는 함수로 판단할 수 있다. 따라서 HEX를 이용한 함수의 시그니처는 DLL이 제공되는 오픈소스의 함수의 안전성 비교에 유용하게 사용 된다.

• Assembly code

Assembly code는 HEX보다 넓은 범위로 함수의 일치여부를 판단해 준다. 그러나 컴파일러의 차이에 의해서 call, jmp 부분의 순서가 뒤바뀌는 경우가 존재하기 때문에 해당 부분을 보완하기 위해 본 논문에서 제시한 도구는 Assembly code에서 call, jmp명령어를 다른 명령어로 치환하여 시그니처를 추출했다.

• Mnemonic code

Mnemonic code은 Assembly code에서 명령어 부분만 가지고 있는 데이터로 해당 부분의 정보를 이용하여 시그니처를 추출할 경우 Assembly code보다 더욱 더 넓은 범위의 일치여부를 조사할 수 있다. 하지만 세부적인 Op code(Opearation Code)의 변경 점에 대해서는 비교할 수 없다. 따라서 세부적인 변경을 알기 위한 목적으로는 사용하기 어렵다.

위의 3가지 정보를 가지고 오픈소스 각각의 함수 시그니처를 뽑아 해당 시그니처를 MD5로 해싱한다. MD5로 해싱함으로 프로그램마다 함수의 길이가 다양하고 함수의 길이가 길어질 경우 분석 데이터가 증가하여 분석 오버헤드가 증가되는 문제를 해결 할 수 있다. 이렇게 생성된 해시값은 DB를 이용하여 저장해 놓고 추후 함수 시그니처 비교에 사용한다.

V. 구 현

5.1 AOS(Automatic Opensource build System)

위에서 제시한 3가지 오픈소스에서 DLL을 추출하기 위하여 제작된 도구는 먼저 오픈소스의 최신버전의 DLL을 제공해주는지를 판단하여 제공한다면 해당 파일을 자동으로 다운로드 한다. 먼저 다운로드 과정에서 해당 파일의 SHA1 해시 값을 비교하여 다운로드 받을 파일의 변조 여부를 판단하였다. 하지만 위와 같이 DLL파일을 제공하지 않고 압축파일의 형식으로 제공하는 오픈소스의 경우에는 직접 컴파일을 하여 DLL파일을 생성하였다.

직접 컴파일하여 DLL을 추출하는 과정은 자동으로 이루어지며 batch코드 언어를 사용하여 도구를 제작하였다. 도구는 Fig 4와 같은 과정을 통해 사용자에게 DLL파일을 제공한다. 먼저 curl프로그램을 이용하여 해당 오픈소스 공식 홈페이지에서 zip또는 tar.gz 확장자로 된 오픈소스 압축 파일을 다운로드 받는다. 그 다음 unzip, 7zip등의 프로그램을 이용, 압축을 풀어서 컴파일 환경을 만든다. 일반적인 Windows환경은 CL.exe라는 Visual Studio 컴파일러가 설치되어있다. 해당 프로그램을 이용하여 압축을 푼 오픈소스 파일들을 컴파일 하면 DLL파일이 생성된다. 이 때 컴파일 버전은 해당 Visual Studio의 버전을 따라가게 된다. 컴파일을 완료한 DLL파일은 도구에서 따로 생성한 DLL파일을 모아두는 폴더로 복사되어 사용자에게 제공한다. 이렇게 생성된

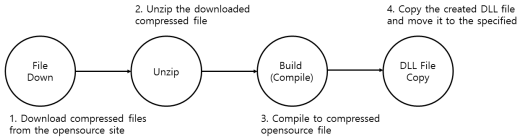

Fig. 4. AOS DLL file create process

DLL파일을 이용하여 함수의 비교 및 분석에 들어가게 된다.

5.2 BSC(Binary Signature Compare plugin)

각 오픈소스 DLL파일에서 함수 시그니처를 생성하고 비교하기 위해 우리는 IDA Pro API[14]를 이용하여 IDA Pro에서 사용 할 수 있는 2개의 플러그인으로 제작했다. 먼저 Update plugin은 DLL파일에서 각 함수에 대한 정보를 받는다. 함수 시그니처는 위의 장에서 설명한 HEX, Assembly, Mnemonic 3가지를 이용한 함수정보를 이용하여 생성, MD5 로 해싱한다. 해싱하는 이유는 각 함수마다 함수의 길이가 다르고 또 함수 시그니처를 생성하는 정보 중 Assembly의 경우 위의 장에서 설명한 부분을 고려하여 정보를 얻은 후의 비교 데이터들을 비교하기 편하게 동일한 형식으로 통일해주어야 되기 때문이다. 또 생성된 시그니처를 다른 시그니처와 비교 시 일정한 길이의 두 값을 비교하는 것이 비교시간을 줄이는 것에 효과적이기 때문에 시그니처를 MD5 해싱하여 사용했다. 아래 Fig 5는 함수 정보를 받은 후의 해시값을 생성하는 함수의 의사코드이다. 해당 함수는 분석하고자 하는 함수의 시작 주소와 끝 주소를 읽어와 시작 주소부터 하여 한 주소씩 증가하여 함수의 정보를 받아온다. 해당 정보를 바탕으로 시그니처를 생성한 후 마지막으로 MD5로 해싱하여 해싱 값을 반환하게 된다.

이와 같은 방법으로 생성된 각 함수의 해시값은

```
def getHash(funcea,endea):
    i = 0
    mnemString = ""
    hexString = ""

    #get Strings from idb
    while funcea + i < endea :
        #gather current function's mnemonic & hex value
        mnemString += getMnemString(funcea+i)+ " "
        hexString += getHexStirng(funcea+i)
        i += 1

    #MD5 incoding
    input_mnemonic_hash = md5.md5(mnemString).hexdigest()
    input_hex_hash = md5.md5(hexString).hexdigest()

    return input_mnemonic_hash,input_hex_hash
```

Fig. 5. getHash() pseudo code

DB에 업로드된다. 본 논문에서 제시하는 도구는 설치 및 이용이 쉬운 MongoDB[15]를 활용하였다. 각 함수의 정보는 업로드 전에 JSON(JavaScript Object Notation)으로 변환된다. Fig 6은 이러한 Update plugin의 동작과정을 단순화시켜 표현한 것이다.

BSC의 두 번째 플러그인인 Query plugin은 비교하고자 하는 대상 바이너리 파일을 입력받아 해당 파일의 함수를 위와 같은 방법으로 해시하여 기존의 DB에 존재하는 해시값과 비교, 함수의 안전성을 판단한다. Fig 7은 이러한 과정을 보여준다.

함수의 안전성을 판단한 후, 본 논문의 궁극적인 목적인 분석 시간 오버헤드를 줄이기 위해 지속적으로 업데이트된 DB의 값과 의심되는 함수와의 1:1 비교도 가능하다. 생성된 DB의 경우 DB 쿼리문을 이용하여 접근과 수정 및 삭제가 가능하다. 또한, 앞서 Update plugin에서 DB를 사용이 용이하기 위해 Json파일로 생성된 시그니처를 저장하였으며 해당 저장 정보들은 각 테이블마다 컴파일러와 버전정보도 같이 저장하였기 때문에 다양한 질의를 통해 원하는 정보를 얻기가 수월하다.

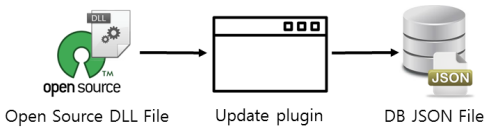

Fig. 6. Update plugin operating process

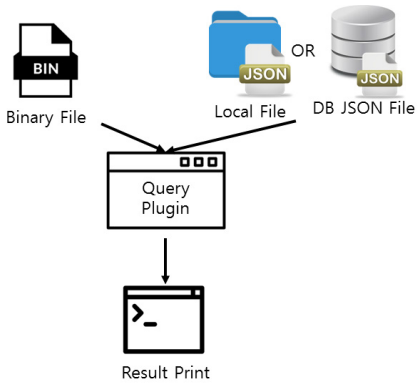

Fig. 7. Query plugin operating process

VI. 실 험

6.1 AOS 실험 결과

아래 Fig 8은 본 논문에서 제시한 DLL생성 도구(AOS)를 통해 Zlib-1.2.8의 DLL파일을 생성이 완료된 결과 화면이다. 해당 폴더의 DLL폴더에서 Zlib와 더불어서 다른 DLL파일도 들어가게 된다. 그림에서는 zlib128.dll, zlib1.dll이 zlib와 관련되어 DLL파일이 생성됨을 확인 할 수 있다.

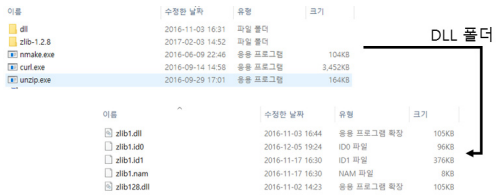

Fig. 8. AOS Building sample of Zlib

6.2 오픈소스 함수 비교 도구 실험

먼저 Update plugin으로 안전한 함수의 함수 시그니처를 생성하고 DB에 저장하였다. Fig 9에서 볼 수 있듯이 한 함수에 대해서 일정한 형식으로 각 함수의 정보를 저장하고 이를 DB를 이용하여 직접 확인 할 수 있다. 이 때 각각의 함수에 대해 Object id, Mnemonic code 해시값, HEX 해시값, 함수 이름, Assembly code 해시값 순서로 값이 들어있다.

이 후 IDA Pro플러그인으로 제작된 비교도구를 이용하여 각 함수를 비교하였다. Fig 9와 같이 DB에 저장된 데이터는 추후 다른 함수 또는 바이너리와의 비교 시 사용된다. 비교한 결과는 아래 Fig 10에서 볼 수 있듯이 IDA 결과 콘솔 창에서 확인 할 수

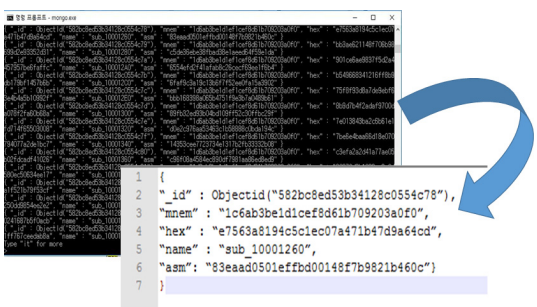

Fig. 9. Update plugin execution result

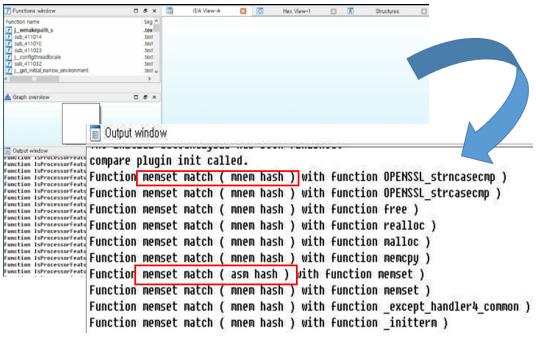

Fig. 10. Query plugin execution result

있다.

Fig 10에서 보여 지는 화면에서 아래쪽 Query plugin의결과를 확대한 그림이다. 예를 들어 7번째와 8번째 줄에서는 memset()이라는 함수와 Update plugin을 통해 DB에 저장되어 있는 OpenSSL의 각각의 함수와의 비교 결과를 보여준다. 분석하려는 memset()함수는 OpenSSL의 여러 함수와 Mnemonic code 해시값과 일치함을 볼 수 있다. 또한 Assembly code 해시값과 일치함을 볼 수 있는데, Mnemonic code 해시값은 위에서 설명 했듯이 여러 함수와의 폭 넓은 일치를 보여주기 때문에 정확성을 기대하기 힘들다. 따라서 정확도가 상대적으로 높은 Aseembly code 해시값과 일치함을 보아 해당 함수는 OpenSSL의 memset()함수일 것이라는 판단을 할 수 있다.

6.3 각 함수 비교 실험 결과

비교 도구를 이용하여 각 함수의 해시값을 추출하여 비교한 결과는 아래 표와 같다. 3가지 오픈 소스를 모두 실험한 결과 Crypto++, Zlib, OpenSSL 모두 비슷한 결과를 보여줬다. 그 중 특이한 경우가 존재하는 OpenSSL을 대표적결과로 올린다.

Table1의 상단부분은 HEX코드에 대한 시그니처 해시값의 결과이다. HEX값을 이용한 시그니처 값을 비교한 결과 각 함수에 대해 컴파일러 버전 및 오픈소스 버전에 따라 변화가 있는 것을 확인 할 수 있다. 그러나 RSA_public_encrypt함수와 RSA_public_decrypt 함수에서는 버전의 변화에만 값이 변하는 것을 확인 할 수 있는데 해당 이유는 OpenSSL DLL

Table 1. OpenSSL function signature MD5 hasing result

Compiler version	Function information	OpenSource Version	Rand_byte()	Rand_screen()	RSA_public_encrypt()	RSA_public_decrypt()
VS 2015	HEX decimal code	1.0.1f	361c782a918347676e0f6ff226f9d790	1e5477cc1a80247d1022601bdbd7bfe5	8886ffa1651084d8600bc6332831074e	632dd603d16df435ad78244d897b92b2
		1.0.2h	f1aab349b8f310a4c3dce79db9aa08c5	942ef16ec5bc06e59cabadb5022e7b7b	8886ffa1651084d8600bc6332831074e	632dd603d16df435ad78244d897b92b2
VS 6.0		1.0.1f	3830edef6dbb6602e174e8944088f108	5d82205b4c62dec4c6180fbc18e6d78a	36a2a9ec85bd1d9cb1246397767fc3ff	9e8afafd6b29f6f1dd61924fe2c8fcb6
		1.0.2h	8bed95d6d40f757bfa6d54b2f9fde638	1d83b1b5d5ef6fac242057887d068315	36a2a9ec85bd1d9cb1246397767fc3ff	9e8afafd6b29f6f1dd61924fe2c8fcb6
VS 2015	Assembly code	1.0.1f	2e1e62e64faf363b16c0b0047a00a147	9714585fb63a70d104da67c40817675a	2f5d945402c5537f8fc692af6e9b8e8f	4194fa6a7d746133d483b73440d2bdc8
		1.0.2h	722c20bbefdbadc3c5a2708c97884a91	fb2b8ba0544bcd374089ac4bba382fcf	2f5d945402c5537f8fc692af6e9b8e8f	4194fa6a7d746133d483b73440d2bdc8
VS 6.0		1.0.1f	6b43bb748612784f5808b4f265a918e5	5f2ddc71332097b321bd7505d8b152d5	a41ba799f106368f7466d62dfcda4856	2c2b01c6ed42168bab5b51bdde3bfb3d
		1.0.2h	0e4a3b2487c14ac0fd5852335175d461	1fec0345713b00f6173755a8a766092a	a41ba799f106368f7466d62dfcda4856	2c2b01c6ed42168bab5b51bdde3bfb3d
VS 2015	Mnemonic code	1.0.1f	b26ab659851ee90252e49273d9a81d45	1099ade2f71a76ba328f698fcffe6ba0	1d59982f186964f61807bdf86a3a7618	1d59982f186964f61807bdf86a3a7618
		1.0.2h	b26ab659851ee90252e49273d9a81d45	1099ade2f71a76ba328f698fcffe6ba0	1d59982f186964f61807bdf86a3a7618	1d59982f186964f61807bdf86a3a7618
VS 6.0		1.0.1f	787801fc257bf918d2927744056534a8	32403901d1b81711ba799da8aa78ae0c	9c45cece81dba0f31fb6d293dbfbe1b2	9c45cece81dba0f31fb6d293dbfbe1b2
		1.0.2h	787801fc257bf918d2927744056534a8	32403901d1b81711ba799da8aa78ae0c	9c45cece81dba0f31fb6d293dbfbe1b2	9c45cece81dba0f31fb6d293dbfbe1b2

파일에서 RSA encrypt와 decrypt의 함수가 똑같기 때문이다.

Table1의 중간 부분은 위와 같은 함수를 어셈블리를 이용한 함수 시그니처의 해시값이다. RAND _ byte함수의 해시값을 보면 버전 그리고 컴파일러에 따라서 해당 값이 변하는 것을 알 수 있다. 그러나 RSA_public_encrypt함수는 컴파일러 버전에 따라서만 값의 변화가 일어나는 것을 알 수 있다. 이를 통해 Rand_byte(), Rand_Screen()에 대한 함수가 OpenSSL버전이 업데이트 되면서 내용이 변했음을 확인할 수 있다. 실제로 홈페이지에서 그 변화에 대한 정보도 확인 할 수 있다.

Table1의 하단 부분은 Mnemonic에 대한 시그니처 해시값 결과이다. 해당 결과를 볼 경우 오픈소스 버전에 대한 차이는 없지만 컴파일러의 버전에 따라 차이가 있었다. 이는 넓은 범위로 함수의 일치여부를 판단할 수 있다. 위의 결과를 바탕으로 일반적으로 사용되는 오픈소스의 함수들은 컴파일러 버전에 따라서 큰 해시값 차이가 발생하였다. 또한 HEX 값을 이용한 해시값 비교 결과는 컴파일러 버전뿐만 아니라 오픈소스의 버전에서도 차이가 발생했다.

VII. 결 론

본 논문에서는 오픈소스 함수의 안전성을 판단하기 위해 2개의 도구를 제안 및 제작하였다. 먼저 안전성을 비교하기 위해 DLL파일을 자동으로 생성하기 위한 오픈소스 DLL자동화 도구와 디스어셈블러 프로그램을 이용하여 다양한 오픈소스의 버전별, 컴파일러별로 각 함수의 시그니처를 추출하고 DB에 저장하는 도구. 이 2개의 도구를 이용, 오픈소스 함수의 시그니처를 바탕으로 오픈소스 함수의 안전성을 검사하였다. 해당 도구는 기존의 다른 비교 도구들과 달리 향후 더욱더 많이 사용되고 있는 다양한 오픈소스에 대해 버전이 달라지거나 다양한 컴파일러를 통해서 빌드 된 각 함수의 정보를 해시하여 DB에 지속적으로 업로드 할 수 있으므로 프로그램에 사용되는 오픈소스 함수들의 안전성을 빠른 시간에 판단 할 수 있을 것이다.

그러나 해당 도구는 버전업데이트와 컴파일러버전에 따른 데이터를 수동적으로 업데이트해야 하는 단점이 존재한다. 따라서 추후 해당 부분에 대해서 자동으로 업데이트가 가능하도록 연구할 예정이다. 또한 본 논문에서는 보안적인 부분을 강조하여 암호화와 관련된 오픈소스들을 실험 대상으로 선정하여 실험하였지만 일반적인 오픈소스 함수들에 대해서는 다르게 함수 정보를 얻어야 할 경우가 발생할 수 있어 해당 부분에 대한 추가 연구도 필요하다.

궁극적으로 본 논문에서는 하나이상의 의심이 되거나 비슷한 한 함수만의 결과 비교를 통해 분석자의 오버헤드를 줄이는 것을 목표로 했다. 실생활에 본 논문에서 제안한 도구는 이전에 배포된 오프소스 함수들의 악성코드 삽입 여부 및 안전성을 판단할 수 있고, 새롭게 배포되는 오픈소스 함수를 DB에 저장된 데이터를 이용하여 쉽고 빠르게 검사 할 수 있다. 또한 악성코드와 관련된 오픈소스에 대해 분석할 경우 특정 오픈소스의 함수의 버전별 컴파일러 별 비교를 빠르게 진행하여 분석시간을 줄일 수 있다.

References

[1] HyeyuKwon, “Strengthen the security of applications by using the open source security framework”, Proc. of the KIISE Korea Computer Congress, pp. 1104-1106 June. 2016

[2] Yeongcheol Kim, Eun-Sun Cho, “Similarity Analysis on Different Versions of Same Functions”, Proc. of the KIISE Korea Computer Congress, pp. 760-762, Dec. 2016

[3] Yongsuk Choi, Jongmoo Choi, “Binary based Software Similarity Analysis Tool”. KIISE : Communioations of the Korean Institute of Information Scientistes and Engineers. 34(1), pp. 37-44, Jan. 2016

[4] JeongHyeok Park, YongSuk Choi, JongMoo Choi, “Software Similarity Analysis via Stack Usage Pattern”. Journal of KIISE : Computing Practices and Letters, 20(6), pp. 349-353, June. 2014

[5] Xin Hu, Tzi-cker Chiudo, Kang G. Shin, “Large-Scale Malware Indexing Using Function-Call Graphs”, ACM CCS 2009

[6] Bindiff, https://www.zynamics.com/bindiff.html

[7] IDA,https://www.hex-rays.com/produc

ts/ida
〔8〕 OpenSSL, https://www.openssl.org/
〔9〕 IDA F.L.I.R.T. Technology: In-Depth https://www.hex-rays.com/products/ida/tech/FLIRT/in_depth.shtml
〔10〕 Zlib, http://www.zlib.net/
〔11〕 Crypto++, https://www.cryptopp.com/
〔12〕 Zlib, https://ko.wikipedia.org/wiki/Zlib
〔13〕 Woo Hyun Ahn, Hyungsu Kim. "Attacking OpenSSL Shared Library Using Code Injection". Journal of KIISE : Computer System and Theory. 37(4), pp. 226-238, Aug. 2010
〔14〕 IDA api, https://www.hex-rays.com/products/ida/support/idapython_docs/
〔15〕 MongoDB, https://www.mongodb.com/

〈저자소개〉

이 석 수 (Seok-su Lee) 학생회원
2017년 2월: 충남대학교 컴퓨터공학과 졸업
2017년 3월~현재: 충남대학교 컴퓨터 석사과정
〈관심분야〉 프로그래밍언어, 역공학

양 종 환 (Jong-hwan Yang) 학생회원
2017년 2월: 충남대학교 컴퓨터공학과 졸업
〈관심분야〉 프로그래밍언어

정 우 식 (Woo-sik Jung) 학생회원
2017년 2월: 충남대학교 컴퓨터공학과 졸업
2017년 3월~2019년 2월: 충남대학교 컴퓨터 석사과정
〈관심분야〉 블록체인, 비트코인

김 영 철 (Yeongcheol Kim) 학생회원
2016년 2월: 충남대학교 컴퓨터공학과 졸업
2016년 3월~현재: 충남대학교 컴퓨터 석사과정
〈관심분야〉 프로그래밍언어, 역공학

조 은 선 (Eun-sun Cho) 정회원
1991년 2월: 서울대학교 계산통계학과 졸업
1993년 2월: 서울대학교 전산과학과 석사
1998년 3월: 서울대학교 전산과학과 박사
1999년~20100년: 한국과학기술원 연구원
2000년~2001년: 아주대학교 정보통신전문대학원 조교수 대우
2002년~2006년: 충북대학교 조교수
2006년~현재: 충남대학교 컴퓨터공학과 교수
〈관심분야〉 프로그래밍언어, 프로그램 분석, 이벤트 처리