J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr
Copyright ⓒ 2019 KSII

iOS 어플리케이션의 잠재적 취약점 분석을 위한 LLDB 모듈 개발☆

Development of LLDB module for potential vulnerability analysis in iOS Application

김 민 정[1] 류 재 철[1*]
Min-jeong Kim Jae-cheol Ryou

요 약

애플의 어플리케이션 마켓인 App Store에 어플리케이션을 등록하기 위해서는 애플 검증 센터를 통해 엄격한 검증 과정을 통과해야 한다. 그렇기 때문에 스파이웨어 어플리케이션의 유입이 까다롭다. 하지만 정상적인 어플리케이션의 취약점을 통해서도 악성코드가 실행될 수 있다. 이러한 공격을 방지하기 위해서는 어플리케이션에서 발생할 수 있는 잠재적 취약점을 패치하기 위해 조기에 발견하고 분석하는 연구가 필요하다. 잠재적 취약점을 증명하기 위해서는 취약점의 근본 원인을 파악하고 악용 가능성을 분석해야 한다. iOS 어플리케이션을 분석하는 도구로는 개발 도구인 Xcode에 내장되어 있는 LLDB라는 이름의 디버거를 활용할 수 있다. LLDB에는 다양한 기능이 존재하며 이 기능들은 API로도 제공되어 Python에서도 사용이 가능하다. 따라서 본 논문에서 LLDB API를 활용하여 iOS 어플리케이션의 잠재적 취약점을 효율적으로 분석하는 방법에 대해 제안한다.

☞ 주제어 : iOS, 취약점 분석, 디버거, LLDB

ABSTRACT

In order to register an application with Apple's App Store, it must pass a rigorous verification process through the Apple verification center. That's why spyware applications are difficult to get into the App Store. However, malicious code can also be executed through normal application vulnerabilities. To prevent such attacks, research is needed to detect and analyze early to patch potential vulnerabilities in applications. To prove a potential vulnerability, it is necessary to identify the root cause of the vulnerability and analyze the exploitability. A tool for analyzing iOS applications is the debugger named LLDB, which is built into Xcode, the development tool. There are various functions in the LLDB, and these functions are also available as APIs and are also available in Python. Therefore, in this paper, we propose a method to efficiently analyze potential vulnerabilities of iOS application by using LLDB API.

☞ keyword : iOS, vulnerability analysis, debugger, LLDB

1. 서 론

iOS는 PC용 운영체제인 MacOS를 기반으로 개발된 애플의 모바일 운영체제이며, Android에 비해 폐쇄적인 정책으로 보안성을 높이고 있다. iOS에서 동작하는 어플리케이션은 Objective C 또는 Swift로 작성된다. 개발된 어플리케이션은 애플의 어플리케이션 마켓인 App Store를 통해 자유롭게 공개하고 판매할 수 있지만 App Store에 등록하기 위해서는 애플 검증 센터의 검증 과정을 통과해야 한다. Android의 어플리케이션 마켓인 Google Play Store에 비해 엄격한 검증 과정을 거치기 때문에 Android 환경에 비해 스파이웨어의 유입이 힘들다. 하지만 스파이웨어가 아닌 정상적인 어플리케이션의 취약점을 통한 악성 코드 실행 가능성은 여전히 존재한다. 취약점이 발생한 대상 어플리케이션이 유명한 어플리케이션이거나 iOS 기본 어플리케이션일 경우는 그 파급력이 더욱 커지게 된다.

이러한 공격을 방지하기 위해서는 정상적인 어플리케이션에서 발생할 수 있는 잠재적 취약점을 조기에 발견하고 원인을 패치 하는 연구가 필요하다. 잠재적 취약점이란 프로그램 내에 존재하지만 알려지지 않은 취약점을 말한다. 잠재적 취약점을 찾는 연구는 다양하게 진행되고 있지만 그중에서도 단시간에 많은 크래시를 확보하기 위

1 Dept. of Computer Science & Engineering, Chungnam National Univ., Daejeon, 34134, Korea.
* Corresponding author (jcryou@cnu.ac.kr)
[Received 6 April 2019, Reviewed 25 April 2019(R2 12 June 2019), Accepted 1 July 2019]
☆ 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2014-6-00909, 운영체제 안전성 연구)
☆ 본 논문은 2018년도 한국인터넷정보학회 추계학술발표대회 우수 논문 추천에 따라 확장 및 수정된 논문임.

http://dx.doi.org/10.7472/jksii.2019.20.4.13

해서는 자동화 도구인 fuzzer가 사용된다. 하지만 fuzzing을 통해 찾은 크래시는 입력 값과 결과만 기록되므로 크래시 발생 원인은 알 수가 없다. 크래시의 근본적인 발생 원인을 파악하기 위해서는 수동적인 분석이 필요하며, 원인 분석을 마치면 공격 가능성 검증을 위해서 임의 코드 실행 등의 공격을 실제로 수행해보는 작업 또한 필요하다.

iOS 어플리케이션에서 발생한 크래시를 분석하기 위해서는 iOS 디바이스와 MacOS를 연결한 후 MacOS에서 동작하는 Xcode라는 개발 도구를 사용하여 분석할 수 있다. Xcode에는 LLDB라는 이름의 디버거가 기본적으로 내장되어 있어 이를 사용하게 된다. LLDB는 데스크탑, iOS 디바이스, 시뮬레이터에서 C, Objective-C, C++ 디버깅을 지원하며 Python 스크립트 사용이 가능하다. 또한 모든 LLDB API들은 SBI(Scripting Bridge Interface)를 통해서 Python에서도 사용이 가능하다. API를 활용하여 Python 스크립트를 작성하면 LLDB에서 기본적으로 제공하는 기능보다 세밀하게 조작할 수 있으며 취약점 분석에 필요한 다양한 정보를 추적해볼 수 있다.

따라서 본 논문은 iOS 어플리케이션의 잠재적 취약점 분석을 수월하게 진행하기 위한 LLDB API 활용 방법에 대해 제안하고 관련 모듈을 개발하였다.

2. 관련 연구

2.1 크래시 수집 방법

어플리케이션의 크래시를 수집하는 방법으로는 여러 가지 연구가 진행되어 왔다. 그중에서도 대표적인 방법이 퍼징(fuzzing)이다. 퍼징은 어플리케이션의 입력 값을 무작위로 변형하여 입력해봄으로써 비정상적인 동작을 유도하고 충돌을 탐지하는 기법이다. 입력 값을 랜덤으로 생성하고 결과를 확인해보는 과정을 자동화함으로써 단시간에 많은 크래시 수집이 가능하다. 모바일 어플리케이션뿐만 아니라 라우터 프로토콜 등 다양한 소프트웨어의 취약점을 찾는 방법으로 사용되고 있다[1][2][8].

퍼징은 주로 소스코드가 공개되어 있지 않은 블랙박스 환경에서 많이 사용된다. 소스코드가 공개된 화이트박스 환경일 경우에는 소스코드를 직접 살펴보며 취약점을 탐색하는 소스코드 오디팅 방법을 사용하기도 한다[9].

2.2 어플리케이션 분석 방법

정상적인 어플리케이션을 분석하는 방법 보다는 악의적인 행위를 하도록 만들어진 스파이웨어 어플리케이션을 분석하는 연구를 많이 찾아볼 수 있다[11][12]. 또한 모바일 분야에 대해서는 대부분 Android 환경에 대한 연구이고 iOS 환경에 대한 연구는 많지 않다[13].

어플리케이션을 분석하는 방법으로는 플랫폼을 수정하는 방법이나 후킹을 이용한 방법을 사용할 수 있다[3]. 플랫폼 수정은 안드로이드 어플리케이션이 실제로 동작하는 DVM(Dalvik Virtual Machine)의 코드를 수정하여 API를 추적하고 로그를 남기는 방법이다. 후킹을 이용한 분석 방법은 커널 API를 후킹하여 동적 분석을 수행하는 방법이다. 이러한 방법들을 통해 호출되는 API를 모니터링할 수 있고, 모니터링 결과를 기반으로 정적 분석과 동적 분석을 수행하게 된다. 이처럼 스파이웨어 분석에 대한 연구는 많이 이루어지고 있지만 정상적인 어플리케이션에 대한 비정상적인 입력 값으로 인해 발생하는 취약점 분석에 대한 연구는 많지 않다. 따라서 본 연구에서는 정상적인 iOS 어플리케이션에서의 취약점 분석에 대해 다룬다.

2.3 크래시 분석 방법

크래시를 분석하는 방법론에는 대표적으로 DBI(Dynamic Binary Instrumentation)와 Symbolic Execution이 있다. DBI는 런타임에 실행 코드를 삽입하여 프로그램의 동작을 분석하는 방법이며 취약점 원인 분석에도 사용된다[4]. 프로그램의 각 명령어 전, 후에 핸들러를 추가하고, 핸들러가 호출되면 명령어와 메모리에 대한 정보를 얻을 수 있다. 이러한 방법을 사용하면 프로그램의 구체적인 실행 상태를 관찰하고 분석할 수 있다. 하지만 실행이 실제로 이루어지는 경로의 코드에 대해서만 분석이 가능하다.

Symbolic Execution은 프로그램 변수에 특정 값을 넣지 않고 미지수를 사용하여 분석하는 방법이다[5][10]. 분기문을 만나면 양쪽 경로를 모두 따라가며 분석한다. 즉, 여러 분기문의 조건을 보고 프로그램의 실행 흐름이 바뀌기 위해 필요한 각 변수 값을 알아내어 모든 경로를 탐색하는 방법이다. 프로그램의 실행 가능한 경로를 꼼꼼히 살피며 여러 갈래로 분석을 수행하지만 구현이 어렵다. 또한 분기문 마다 발생하는 여러 실행 경로를 모두 따라가므로 시간과 메모리 소모가 심한 문제점이 있다.

본 연구에서는 LLDB만 사용하고도 분석에 도움이 되는 정보를 얻을 수 있도록 LLDB API를 사용하여 취약점 분석 모듈을 개발한다.

3. LLDB 모듈 설계 및 구현

잠재적 취약점을 분석하는데 있어서 크게 세 가지 문제점을 파악하고 그에 따른 방안을 모듈의 기능으로 구현했다. 첫 번째 문제점은 수백줄이 넘는 함수 내용 분석의 어려움이다. 이에 대해서는 실행 흐름 변경과 관련된 명령어를 눈에 띄게 표시해줌으로써 분석 포인트를 쉽게 찾아갈 수 있도록 구현했다. 두 번째 문제점은 크래시 원인에 영향을 준 데이터를 파악하기 어렵다는 점이다. 이에 대해서는 레지스터와 스택의 데이터를 추적하여 데이터 관계를 확인하는 기능을 구현했다. 세 번째 문제는 공격 가능성 검증 단계에서 실행 흐름 조작에 사용할 유용한 가젯 수집이 필요하다는 점이다. 이에 대해서는 로드된 모듈 내에서 유용한 가젯을 자동으로 탐색하여 파일로 저장하도록 기능을 구현했다.

3.1 함수 분석

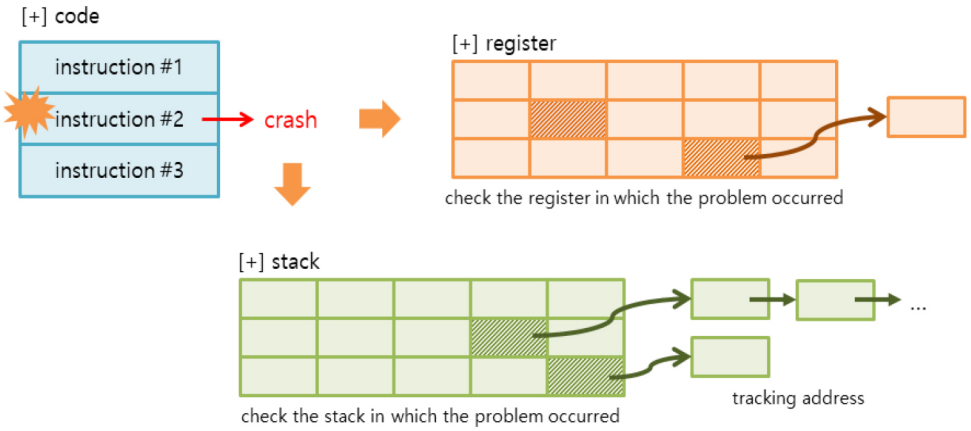
잠재적 취약점 분석을 위해서는 프로그램의 실행 흐름에 영향을 미치는 다양한 조건들을 분석해야 한다. 함수를 디스어셈블한 결과를 살펴보며 프로그램의 실행 흐름을 변경하는 다양한 인스트럭션에 대한 파악이 필요하다. 예시로 iOS 어플리케이션과 같이 ARM 프로세서에서 동작하는 프로그램에서는 blr, bl, b, br, b.ne, b.gt 등 branch 계열의 명령어들을 찾을 수 있다. branch 계열의 명령어는 다른 주소로 프로그램의 실행 흐름을 변경하는 기능을 수행하며 분기문에 해당한다.

branch 계열의 명령어와 같은 분기문은 cmp 명령어 처럼 비교문과 함께 쓰이는 경우가 많다. 프로그램이 실행 될 때 현재의 실행 조건, 입력 조건 등을 비교하여 서로 다른 실행 흐름을 갖도록 하는 경우이다. 잠재적 취약점이 발생하는 경로에 도달하기 위해서는 이러한 비교 조건에 따른 실행 흐름을 파악하는 것이 중요하다. 하지만 사용 프로그램의 경우 하나의 함수 안에도 수백 줄의 코드가 존재하기 때문에 필요한 명령어들을 빠르게 추적하기가 어렵다.

본 연구에서 개발한 모듈에서는 실행 흐름에 영향을 미칠 수 있는 분기문과 비교문의 위치를 알아보기 쉽게 표시하는 기능을 갖는다. 이 기능을 구현하기 위해 LLDB API 중 lldb.frame.Disassemble 등을 사용하여 분석할 대상 프레임을 가져오고 함수를 디스어셈블하였다. 그런 뒤에, 디스어셈블 결과 중 분기문과 비교문을 파싱할 수 있도록 구현하였다. 이를 통해 실행 흐름이 바뀔 수 있는 지점을 한눈에 알 수 있고 해당 지점에 브레이크 포인트를 설정하여 구체적인 분석을 진행할 수 있게 된다.

3.2 레지스터와 스택 분석

잠재적 취약점으로 인해 크래시가 발생했을 때, 해당 크래시가 발생한 근본적인 원인을 정확하게 파악할 수 있어야한다. 원인 분석을 마쳐야 분석 내용을 토대로 잠재적 취약점의 악용 가능성에 대해 연구하고 대처 방안을 생각할 수 있다. 원인 분석을 위해서는 크래시 발생 지점에서 각 레지스터의 정보들을 확인하고 해당 데이터가 어디에서 입력되고, 변경되어 왔는지 추적해야 한다. 하



(그림 1) 레지스터와 스택 분석
(Figure 1) analysis register and stack

지만 수십 개의 레지스터와 스택에 쌓인 무수한 값들을 하나씩 분석하려면 많은 시간이 필요하다.

본 연구에서 개발한 모듈에서는 각 레지스터의 정보와 함께 어떤 레지스터가 현재 프레임에서 사용중인지 표시해주어 분석 범위를 좁힐 수 있도록 하였다. 뿐만 아니라 그림 1과 같이 레지스터가 가리키는 값이 주소 값일 경우에는 해당 주소 값을 다시 추적하여 그 주소 값에 저장된 값을 보여준다. 추적된 값 또한 주소 값이라면 추적을 계속하도록 구현하였다. 스택의 정보도 마찬가지로 주소 값들에 대해서는 추적을 계속하여 보여주도록 하였다. 이 기능을 구현하기 위해 LLDB API 중 lldb.frame.FindRegister와 lldb.frame.Disassemble을 사용하여 레지스터 값과 함수의 디스어셈블 결과를 가져왔고, 디스어셈블 결과에 나타난 레지스터 이름을 파싱하여 현재 프레임에서 사용중인 레지스터 정보를 파악할 수 있었다. 레지스터의 값이 주소 값일 경우에는 lldb.process.GetMemoryRegions와 lldb.process.GetMemoryRegionInfo를 통해 해당 주소 값이 가리키는 영역에 대한 정보를 'heap', 'stack' 또는 라이브러리 이름으로 표시하였다. 또한 주소 값에 대한 추적을 계속하기 위해 ReadMemory를 사용하여 해당 주소를 읽고 값을 한 번 더 가져오도록 하였다. 이렇게 제공된 정보를 이용하면 분석 대상이 되는 데이터의 참조 흐름을 쉽게 파악할 수 있다.

3.3 가젯(Gadget) 검색

잠재적 취약점의 원인을 파악한 후에 악용 가능성을 검증하기 위해서는 프로그램의 실행 흐름을 바꿔보는 작업이 필요하다. 이때 주로 사용되는 공격 기법으로는 메모리상에 존재하는 실행 가능한 코드를 재사용하는 코드 재사용 공격(Code Reuse Attack)이 있다. 코드 재사용 공격을 사용할 경우, 스택 또는 힙 영역에 대한 실행 권한을 제거하는 DEP(Data Execution Preventation) 보호기법을 우회할 수 있기 때문에 공격 방법으로 널리 사용된다. DEP는 각 메모리 영역에 대해 읽기/쓰기/실행(r/w/x) 권한을 확인하고 실행 권한이 없는 메모리에 속한 코드는 실행되지 못하게 방지한다. DEP를 우회할 수 있는 코드 재사용 공격의 대표적인 종류로는 ROP(Return-Oriented Programing)와 JOP(Jump-Oriented Programing)가 있다[6][7]. ROP와 JOP의 경우, 실행 가능한 영역에 있는 코드를 호출하는 것이기 때문에 DEP와 관계 없이 정상적이 코드 호출이 가능하다.

ROP는 RET 명령어로 끝나는 짧은 코드 조각을 연결하는 방법이다. 이때 사용되는 코드 조각을 가젯이라고 한다. 프로그램의 실행 흐름을 조작하여 가젯을 실행하게 하면 짧은 명령어들을 수행한 후 RET 명령어를 실행하게 되는데, RET 명령어는 스택에 있는 값을 다음 실행할 명령어 주소로 설정하는 기능을 한다. 스택에 다음 실행할 가젯의 주소를 미리 설정해 놓으면 하나의 가젯을 실행한 후에 다음 가젯을 연결하여 실행할 수 있게 된다. 이러한 방법으로 짧은 명령어 조각을 여러 개 연결하여 원하는 기능을 수행하도록 실행 흐름을 조작할 수 있다. 스택에 원하는 가젯의 주소 값을 미리 설정해야하기 때문에 ROP는 스택을 제어할 수 있는 상황에서 사용가능한 공격 방법이다.

JOP는 ROP와 유사하지만 사용되는 가젯에 차이가 있다. JOP에 사용되는 가젯은 ROP에 사용되는 가젯과는 다르게 스택이 아닌 레지스터에 있는 값을 다음 실행할 명령어 주소로 설정한다. 레지스터 중 어느 하나라도 제어 가능한 값이 있다면 해당 레지스터에 다음 가젯의 주소를 설정해놓음으로써 여러 개의 가젯을 연결하여 원하는 실행 흐름을 만들 수 있다.

스택을 제어할 수 있는 경우 보다는 레지스터를 제어할 수 있는 경우가 더 자주 발생하기 때문에 본 연구에서는 ROP 가젯 보다는 JOP 가젯을 탐색하는 것에 집중하였다. 본 연구에서 개발한 모듈에서는 현재 프로세스와 관련 모듈을 모두 탐색하며 유용한 JOP 가젯들을 찾아주는 기능을 갖는다. 이 기능은 LLDB API 중 lldb.process.ReadMemory와 lldb.process.GetMemoryRegions, lldb.SBMemoryRegionInfo, lldb.GetRegionBase, lldb.GetRegionEnd 등을 사용하여 현재 프레임에서 로드한 메모리 정보들을 모두 읽어 들임으로써 구현하였다. 그 후, 정규식을 활용하여 유용한 가젯의 패턴을 검색하고 그 결과를 파일로 저장하도록 하였다.

4. 실험 결과

본 연구에서 개발한 모듈을 활용하여 잠재적 취약점 샘플을 분석하였다. 해당 샘플은 iOS 어플리케이션 중 mobile_safari에서 발생한 크래시이다. iPhone 디바이스와 맥을 연결한 후 Xcode를 이용하여 분석을 진행하였다. 먼저, 함수 분석 기능을 사용하면 그림 2와 같이 프로그램의 흐름이 바뀔 수 있는 지점을 한 눈에 파악할 수 있었다. 또한 어떠한 조건에 의해 실행 흐름이 바뀌는지도 쉽게 파악이 가능하였다. 또한 iOS가 사용하는 ARM 환경에서는 그림 3에서처럼 레지스터가 가진 값을 기반으로 실행 흐름이 바뀌는 경우가 빈번한데 이러한 경우도 눈

```
    0x18896facc <+120>: ldr    x9, [x9, x10, lsl #3]
    0x18896fad0 <+124>: ldr    x21, [x9, #0x18]
[>>]0x18896fad4 <+128>: cmp    w8, #0x1                    ; =0x1
[++]0x18896fad8 <+132>: b.ne   0x18896fb54                 ; <+256>
    0x18896fadc <+136>: ldr    x8, [sp, #0x10]
```

(그림 2) 함수 흐름 분석
(Figure 2) Function flow analysis

```
    0x18896fba0 <+332>: ldr    x8, [x0]
    0x18896fba4 <+336>: ldr    x8, [x8, #0x8]
[++]0x18896fba8 <+340>: blr    x8                          ============>register jump
[++]0x18896fbac <+344>: b      0x18896fbb8               ; <+356>
    0x18896fbb0 <+348>: ldr    x19, [x9]
    0x18896fbb4 <+352>: cbz    x19, 0x18896fb00          ; <+172>
```

(그림 3) 악용 가능성이 있는 흐름 분석
(Figure 3) Exeploitable flow analysis

```
(lldb) context register d
[*] register
 [x0  ]: 0x0000000101fbcb58(heap) --> 0x0000000101e9d180(heap) --> 0x0000000044444444(?)
 [x1  ]: 0x0000000103a70298(heap) --> 0x0000000000000000(?)
 [x2  ]: 0x000000016e3f88a0(stack) --> 0x00000001a995f528(WebCore) --> 0x0000000200000010(?)
 [x3  ]: 0x000000016e3f8908(stack) --> 0x0000000000000000(?)
 [x4  ]: 0x0000000103be22f0(heap) --> 0x0000000187c3cd68(JavaScriptCore) --> 0xb98012208b040f51(?)
 [x5  ]: 0x0000000000000010(?)
 [x6  ]: 0x0000000000000003(?)
 [x7  ]: 0x0000000000000000(?)
 [x8  ]: 0x0000000101e9d180(heap) --> 0x0000000044444444(?)
 [x9  ]: 0x0000000044444444(?)
```

(그림 4) 레지스터의 주소 값 추적
(Figure 4) Track address values in registers

```
(lldb) context stack
[*] stack
[00] 0x000000016e3f8880 :  --> 0x0000000000000000(?)
[01] 0x000000016e3f8888 :  --> 0x0000000188010001(StoreServices) --> 0x30510012301b036e(?)
[02] 0x000000016e3f8890 :  --> 0x0000000101fbcb58(heap) --> 0x0000000101e9d180(heap) --> 0x0000000044444444(?)
[03] 0x000000016e3f8898 :  --> 0x0000000100000001(heap)
[04] 0x000000016e3f88a0 :  --> 0x00000001a995f528(WebCore) --> 0x0000000200000010(?)
[05] 0x000000016e3f88a8 :  --> 0x0000000103b20380(heap) --> 0x0008150000000101(?)
[06] 0x000000016e3f88b0 :  --> 0x00000001a995f528(WebCore) --> 0x0000000200000010(?)
```

(그림 5) 스택의 주소 값 추적
(Figure 5) Track address values in the stack

에 띄게 표시를 해주고 있다. 레지스터 기반으로 실행 흐름이 바뀌는 경우는 해당 레지스터의 값이 컨트롤 가능한 값이라면 실행 흐름을 원하는 곳으로 바꿀 수 있으므로 중요 공격 벡터가 된다.

다음으로 그림 4, 그림 5와 같이 크래시가 발생한 지점에서의 각 레지스터와 스택의 정보를 분석하였다. 해당 크래시는 x9 레지스터가 가진 값을 주소 값으로 인식하여 데이터를 로드할 때, x9 레지스터에 0x44444444라는 잘못된 값이 들어가 있어서 문제가 발생하였다. 그런데 그림 4와 같이 다른 레지스터를 추적한 결과에서도 0x44444444라는 값을 확인할 수 있었다. 이를 통해 코드를 자세히 분석하지 않고도 데이터의 흐름을 파악할 수 있었다.

마지막으로, 공격 기법을 사용하기 위해 프로그램 내에 존재하는 가젯을 탐색하였다. 그림 6과 같이 탐색된 가젯들은 파일 형태로 저장되기 때문에 해당 파일에서 원하는 가젯을 검색해서 사용할 수 있다. 사전의 분석을 통해 컨트롤 가능한 레지스터와 메모리 공간을 알아냈다면, 해당 메모리 공간의 값을 레지스터로 로드하고 해당 레지스터로 실행 흐름을 변경하는 가젯을 사용하면 취약점을 악용할 수 있다.

```
jmpcall (without dup)
        > 0x00000001000f8808 : blr  x8;
        > 0x00000001000f8f7c : br   x3;
        > 0x00000001000f9030 : br   x1;
        > 0x0000000100119610 : ldr  x8, [x8, #0x90]; blr  x8;
        > 0x000000010011982c : ldr  x8, [x22]; ldr  x8, [x8, #0x38]; blr  x8;
        > 0x0000000100119c2c : blr  x21;
        > 0x000000010011bf94 : ldr  x0, [sp, #0x8]; ldr  x8, [x0]; ldr  x8, [x8, #0x1b0]; blr  x8;
        > 0x000000010011c16c : blr  x23;
        > 0x000000010011c6c8 : ldr  x0, [x23]; ldr  x8, [x0]; ldr  x8, [x8, #0x188]; blr  x8;
```

(그림 6) 가젯 탐색
(Figure 6) Find Gadget

5. 결론 및 향후 연구

본 연구에서는 잠재적 취약점을 효율적으로 분석하기 위한 LLDB API 사용에 대해 제안하였다. 함수 흐름에 대한 분석과 데이터 추적, 가젯 탐색 등의 기능을 제공한다. 함수 흐름 분석 기능은 수백줄이 넘는 코드 중 주의깊에 살펴야 하는 코드를 눈에 띄게 표시해줌으로써 가독성을 높여준다. 데이터 추적 기능에서는 보통 특정 레지스터에 어떠한 값이 들어가기까지 여러 레지스터가 영향을 주게 되는데, 자세한 분석을 하지 않고도 데이터의 흐름을 파악할 수 있다. 가젯 탐색 기능에서는 공격에 사용되는 유용한 가젯을 한번에 탐색하여 저장함으로 별도의 분석 시간이 필요하지 않다. 위의 세 가지 기능을 사용함으로써 잠재적 취약점 분석에 필요한 시간을 단축 시킬 수 있었다. 또한 이러한 방법은 잠재적 취약점 분석뿐만 아니라 악성코드 분석, CVE 분석 등에도 활용할 수 있다.

향후 연구에서는 함수의 호출 흐름인 콜 스택과 관련된 분석 방법을 자동화하고자 한다. 콜 스택에서는 함수의 호출 흐름에 대한 정보를 제공하기 때문에 이를 자동화하면 짧은 시간에 많은 정보를 획득할 수 있다.

참고문헌(Reference)

[1] Kwanghoon Choi, Myungpil Ko and Byeong-Mo Chang, "A Practical Intent Fuzzing Tool for Robustness of Inter-Component Communication in Android Apps", KSII Transactions on Internet and Information Systems, Vol.12, No.9, pp.4248-4270, 2018.
http://doi.org/10.3837/tiis.2018.09.008

[2] Huang, Xinyue, et al. "Fuzzing the Android Applications With HTTP/HTTPS Network Data", IEEE Access 7, pp.59951-59962, 2019.
https://doi.org/10.1109/ACCESS.2019.2915339

[3] Dong-Wook Kim, Kyung-Gi Na, Myung-Mook Han, Mijoo Kim, Woong Go, & Jun Hyung Park. "Malware Application Classification based on Feature Extraction and Machine Learning for Malicious Behavior Analysis in Android Platform", Journal of Internet Computing and Services, Vol.19, No.1, pp.27-36, 2018.
http://doi.org/10.7472/jksii.2018.19.1.27

[4] Peng, Jianshan, Mi Zhang, and Qingxian Wang. "Deduplication and Exploitability Determination of UAF Vulnerability Samples by Fast Clustering", KSII Transactions on Internet & Information Systems, Vol.10, No.10, pp.4933-4956, 2016.
https://doi.org/10.3837/tiis.2016.10.016

[5] Baldoni, Roberto, et al. "A survey of symbolic execution techniques", ACM Computing Surveys (CSUR) 51.3, 50, 2018.
https://doi.org/10.1145/3182657

[6] Marco Prandini, Marco Ramilli, "Return-Oriented Programming", IEEE Security & Privacy, Vol.10(6), pp.84-87, 2012.
https://doi.org/10.1109/MSP.2012.152

[7] Bletsch, Tyler, et al. "Jump-oriented programming: a new class of code-reuse attack", Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. ACM, pp.30-40, 2011.
https://doi.org/10.1145/1966913.1966919

[8] Rasthofer, Siegfried, et al. "Making malory behave maliciously: Targeted fuzzing of android execution environments", 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, pp.300-311, 2017.
https://doi.org/10.1109/ICSE.2017.35

[9] Padmanabhuni, Bindu Madhavi, and Hee Beng Kuan Tan. "Auditing buffer overflow vulnerabilities using

hybrid static–dynamic analysis", IET Software, Vol.10.2, pp.54-61, 2016. https://doi.org/10.1049/iet-sen.2014.0185

[10] Stephens, Nick, et al. "Driller: Augmenting Fuzzing Through Selective Symbolic Execution", NDSS, Vol. 16. No.2016, pp.1-16, 2016. http://dx.doi.org/10.14722/ndss.2016.23368

[11] García, Laura, and Ricardo J. Rodríguez. "A Peek under the Hood of iOS Malware", 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, pp.590-598, 2016. https://doi.org/10.1109/ARES.2016.15

[12] Tam, Kimberly, et al. "The evolution of android malware and android analysis techniques", ACM Computing Surveys (CSUR), Vol.49.4, No.76, 2017. https://doi.org/10.1145/3017427

[13] Machado, Pedro, José Campos, and Rui Abreu. "MZoltar: automatic debugging of Android applications." Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile. ACM, pp.9-16, 2013. https://doi.org/10.1145/2501553.2501556

◐ 저 자 소 개 ◑

김 민 정(Min-jeong Kim)
2018년 충남대학교 컴퓨터공학과(공학사)
2018년~현재 충남대학교 대학원 컴퓨터공학과(공학석사)
관심분야 : 정보보호, 시스템 보안, 스마트폰 보안, 취약점 분석, etc.
E-mail : rls1004@o.cnu.ac.kr

류 재 철(Jae-cheol Ryou)
1985년 한양대학교 산업공학과(공학사)
1988년 아이오와 주립대학교 대학원 전산학과(공학석사)
1990년 노스웨스턴 대학교 대학원 전산학과(이학박사)
1991년~현재 충남대학교 컴퓨터공학과 교수
관심분야 : 인터넷 보안, 전자 지불 시스템, 블록체인 etc.
E-mail : jcryou@cnu.ac.kr