Journal of The Korea Institute of Information Security & Cryptology
VOL.28, NO.3, Jun. 2018
ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.3.643

자마린으로 개발된 안드로이드 앱의 정적 분석 연구*

임 경 환,† 김 규 식, 심 재 우, 조 성 제‡
단국대학교 컴퓨터학과

A Static Analysis Technique for Android Apps Written with Xamarin*

Kyeong-hwan Lim,† Gyu-sik Kim, Jae-woo Shim, Seong-je Cho‡
Dept. of Computer Science and Engineering, Dankook University

요 약

자마린은 대표적인 크로스 플랫폼 개발 프레임워크로, 안드로이드, iOS, 또는 Windows Phone등의 여러 플랫폼을 위한 모바일 앱을 C#으로 작성하게 해준다. 모바일 앱 개발자들은 기존의 C# 코드를 재사용하고 여러 플랫폼 간에 상당한 코드를 공유할 수 있어 개발 시간과 유지보수 비용을 줄일 수 있다. 한편, 멀웨어 작성자들 또한 자마린을 이용하여 악성 앱 제작 시간과 비용을 최소화하면서 더 많은 플랫폼에 악성 앱을 전파할 수 있다. 이에 대응하기 위해서 자마린으로 작성된 멀웨어를 분석하고 탐지하는 방안이 필요하다. 그러나 현재 자마린으로 작성된 앱에 대한 분석 방법에 대한 연구가 많이 이루어지고 있지 않다. 이에 본 논문에서는 자마린으로 개발된 안드로이드 앱의 구조를 파악하고 앱 코드를 정적으로 분석하는 기법을 제안한다. 또한, 코드 난독화가 적용된 앱에 대해서도 정적으로 역공학하는 방법을 보인다. 자마린으로 개발된 앱은 자바 바이트코드, C# 기반의 DLL 라이브러리, C/C++ 기반의 네이티브 라이브러리로 구성되어 있으며, 이들 서로 다른 유형의 코드들에 대한 정적 역공학 기법에 대해서 연구하였다.

ABSTRACT

Xamarin is a representative cross-platform development framework that allows developers to write mobile apps in C# for multiple mobile platforms, such as Android, iOS, or Windows Phone. Using Xamarin, mobile app developers can reuse existing C# code and share significant code across multiple platforms, reducing development time and maintenance costs. Meanwhile, malware authors can also use Xamarin to spread malicious apps on more platforms, minimizing the time and cost of malicious app creation. In order to cope with this problem, it is necessary to analyze and detect malware written with Xamarin. However, little studies have been conducted on static analysis methods of the apps written in Xamarin. In this paper, we examine the structure of Android apps written with Xamarin and propose a static analysis technique for the apps. We also demonstrate how to statically reverse-engineer apps that have been transformed using code obfuscation. Because the Android apps written with Xamarin consists of Java bytecode, C# based DLL libraries, and C/C++ based native libraries, we have studied static reverse engineering techniques for these different types of code.

Keywords: Xamarin framework, Android app, Cross-platform, Static analysis, Intermediate Language

Received(03. 14. 2018), Modified(05. 31. 2018), Accepted(06. 01. 2018)

* 본 논문은 2017년도 동계학술대회에 발표한 우수논문을 개선 및 확장한 것으로 2017년도 정부(미래창조과학부)의 재원으로 한국연구재단 기초연구사업의 지원을 받아 수행된 연구임(No. 2015R1A2A1A15053738) 그리고 산업통상자원부(MOTIE)와 한국에너지기술평가원(KETEP)의 지원을 받아 수행한 연구 과제임(NO. 20171510102080)
† 주저자, limkh120@dankook.ac.kr
‡ 교신저자, sjcho@dankook.ac.kr(Corresponding author)

I. 서 론

최근 하나의 프로그램을 개발하여 여러 플랫폼에서 수행할 수 있게 해 주는, 크로스 플랫폼 개발 프레임워크(cross-platform development framework)의 사용이 증가하고 있다[1-4]. 대표적인 크로스 플랫폼 개발 프레임워크로는 자마린(Xamarin), 유니티(Unity), PhoneGap, Appcelerator Titanium, Sencha 등이 있다. 이러한 프레임워크는 모바일 앱의 개발 생명주기 동안 개발 비용 및 시간, 유지보수 등의 면에서 효율성을 증대시켜 준다. 즉, 안드로이드, iOS, 윈도우 폰(Windows Phone), 타이젠(Tizen) 등의 여러 모바일 플랫폼들이 경쟁하고 있는 현재, 특정 플랫폼을 위한 네이티브 앱(native app)을 개별적으로 개발하는 것은 개발 비용을 크게 증가시킨다. 안드로이드는 Java, iOS는 Objective-C와 같이 특정 플랫폼은 전용 프로그래밍 언어 및 개발도구를 필요로 하며, 업데이트나 버그 수정과 같은 유지보수 비용도 매우 비싸다. 특정 모바일 플랫폼의 프로그래밍 언어와 API로 개발된 네이티브 앱은 해당 플랫폼의 기능을 충분히 활용할 수 있는 반면, 다른 플랫폼에서는 수행되기 어렵다. 이러한 상황에서 특정 플랫폼에 독립적인 모바일 크로스 플랫폼 도구가 주목받고 있다.

자마린(Xamarin)은 대표적인 모바일 크로스 플랫폼 프레임워크로, 안드로이드, iOS, 또는 윈도우 폰 등의 여러 플랫폼을 위한 모바일 앱을 C#으로 작성하게 해준다[1-4]. 자마린을 이용할 경우, C# 프로그래머들은 특정 플랫폼용 프로그래밍 언어를 익힐 필요 없이 모바일 앱을 즉시 개발할 수 있으며, 기존의 C# 코드를 재사용하고 다른 플랫폼 간에 상당한 코드를 공유할 수 있다. 2015년 기준 자마린 웹 사이트에 의하면, 90만 명의 개발자들이 크로스 플랫폼 앱들을 개발하기 위해 자마린 도구를 사용하고 있다[1].

한편, 공격자들도 멀웨어 제작시간을 최소화하면서 최대의 공격 효과를 내려고 자마린을 악용한다고 알려져 있다[5]. C#으로 악성 앱들을 더 빨리 제작하거나 기존의 악성 코드를 재활용하여 더 많은 플랫폼에 효과적으로 전파할 수 있기 때문이다. 백신 우회용 악성 실행파일을 제작하는 공격도구들의 집합인 Veil-Framework에서는, 악성 페이로드를 C#으로 개발하였다. 실제 인터넷을 검색하여 보면 C#으로 키로거(keylogger)를 작성하는 방법, C# Crypto-Ransomware 작성 및 DDoS 멀웨어 관련 유투브 영상, C#으로 콘솔 기반의 트로이목마를 작성하는 튜토리얼, C# Nemesis.Worm 등을 접할 수 있다. Mylonas[6, 7] 등은 실험을 통해 학생들이 C#을 이용하여 윈도우 모바일(Windows Mobile)에서는 이틀 만에, 윈도우 폰에서는 하루 만에 악성 앱을 제작할 수 있음을 보였다.

이처럼 C# 코드 기반의 악성 앱들이 도입되고 있지만, C# 코드로 제작된 악성 앱에 대한 효과적 분석 기법은 연구된 바가 없다. 이에 본 논문에서는 자마린으로 개발된 안드로이드 앱을 정적으로 역공학하는 방법을 제안한다. 또한 실험에서는, 난독화 기법을 포함시켜 C# 앱을 재패키징한 후, 이를 역공학해 보임으로써 제안 방법의 실효성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 기술한다. 3장에서는 자마린 프레임워크 및 자마린으로 개발된 앱의 실행 환경인 모노 런타임(Mono runtime)에 설명한다. 4장에서는 자마린으로 개발된 안드로이드 앱에 대한 정적 분석 기법을 기술하면서, 네이티브 앱을 정적 분석하는 방법과의 차이점에 대해 살펴본다. 5장에서는 역공학 방해 요소가 적용된 앱을 분석하는 방법에 대해 논의하고 6장에서 결론을 맺는다.

II. 관련 연구

자마린이 지원하는 모노(Mono)는 크로스 플랫폼 앱을 개발하도록 해 주는 .NET 프레임워크용 최초의 오픈 소스다[5]. 이 프로젝트는 자마린의 후원을 받아 리눅스, 윈도우, Mac OS X에 설치될 수 있다. 모노에서 개발된 프로젝트는 MoMA (Mono Migration Analyzer)을 사용하여 한 플랫폼에서 다른 플랫폼으로 이식될 수 있으며, MoMA는 자동화된 도구로 자마린 앱들의 생산성을 증대시켜 줄 수 있다. 자마린을 이용하면, C#으로 작성된 앱들이 안드로이드, iOS, Mac OS X 상에서 동일 코드를 공유할 수 있다.

Mylonas[6, 7] 등은 대표적인 스마트폰 플랫폼들의 보안 수준을 비교 평가하기 위해, 대상 플랫폼들에서 일반 프로그래머들이 멀웨어를 제작할 수 있는지, 제작할 수 있다면 얼마나 작성이 용이한 지를 실험하였다. 실험에서 프로그래머들은 스마트폰 플랫폼에서 제공된 개발 도구와 프로그래밍 라이브러리를

사용하여, 안드로이드에서는 Java 언어를 사용하여 12시간 만에 멀웨어를 제작할 수 있었다. iOS에서는 Objective-C를 사용하여 7일 만에 멀웨어를 제작하였고, Windows Mobile 6에서는 C#을 이용해 이틀 만에 멀웨어를 제작하였다. Windows Phone 7에서는 C# 프로그래밍 경험을 가진 학부생이 하루 만에 멀웨어를 제작하였다.

Boushehrinejadmoradi[2] 등은 X-Checker라는 테스팅 도구를 개발하여, 홈 플랫폼 API를 가지고 동일 기능을 제공하는 타깃 플랫폼 API로 변환할 때 발생할 수 있는 비일치 오류를 조사하였다. 실험결과, 자마린에서 Windows Phone API와 안드로이드 API의 구문 변환 시에 47개의 버그를 탐지하였다. Martinez[3] 등은 크로스 플랫폼 모바일 앱 개발 프레임워크를 사용하여 개발된 모바일 앱들의 품질을 분석하였다. 즉, 자마린과 같은 크로스 컴파일 프레임워크를 사용해 작성된 앱들의 개발 및 유지보수 과정을, 버그 탐지 및 정정에 초점을 두고 평가하였다.

여러 논문들이 악성 앱 개발 용이성과 버그 탐지·정정에 대해 연구하였지만, 자마린에서 C#으로 개발된 앱에 대한 체계적인 역공학 분석 연구는 현재까지 수행된 바가 없다. 본 연구는 자마린에서 C#으로 개발된 안드로이드 앱을 대상으로 정적 역공학 분석 연구를 진행한다.

III. 자마린 및 모노 런타임 환경

3.1 자마린 개요

2016년 마이크로소프트는 닷넷 프레임워크를 다른 외부 플랫폼으로 이식하기 위해, 모노 프로젝트(Mono Project) 기반으로 개발된 자마린 크로스 플랫폼 개발 환경을 무료로 제공하고 있다. 자마린은 닷넷 프레임워크와 C# 언어를 이용하여 안드로이드, 윈도우 폰, iOS와 같은 다양한 모바일 플랫폼에서 동작할 수 있는 앱을 개발하게 해 준다. 자마린은 크로스 플랫폼 개발 프레임워크[2] 또는 소스코드 번역기(Source code translator)[1]로, 주어진 소스코드를 크로스 컴파일하여, 컴파일된 코드를 다른 플랫폼들에서 수행하게 해 준다. 소스코드는 플랫폼의 네이티브 언어(native language) 또는 실행 가능한 바이트 코드로 변환된다. 소스코드가 런타임 환경(Runtime environment)에 의해 실행될 수 있는 코드로 번역된다면, 소스코드 번역기는 런타임 구성요소(Runtime element)와 결합될 수 있다. 자마린으로 개발된 앱의 개요가 Fig. 1.에 나타나 있다. 자마린으로 안드로이드 앱을 개발할 경우, C#이 번역되어 IL(Intermediate Language) 코드가 생성되며 해당 IL 코드를 실행하기 위한 모노 환경과 IL 코드를 위한 JIT(Just-In-Time) 컴파일 기능을 포함하여 앱을 생성하게 된다.

자마린으로 개발된 앱을 역공학하기 위해서는 컴파일 과정과 실행파일 구조, 런타임 동작 원리를 파악할 필요가 있다. 자마린의 C# 컴파일러를 통해 생성된 IL 코드는 Java의 바이트코드에 대응되며, CLR (Common Language Runtime)의 JIT 컴파일을 통해 기계어로 변환된다[5]. IL 코드는 .NET PE 파일의 포맷을 갖고 있으며 PE 포맷의 구성은 Fig. 2.와 같다. 이러한 앱의 기능 및 동작 과정을 파악하기 위해서 PE 파일 포맷을 갖는 DLL 파일들을 정적 역공학해야 한다.

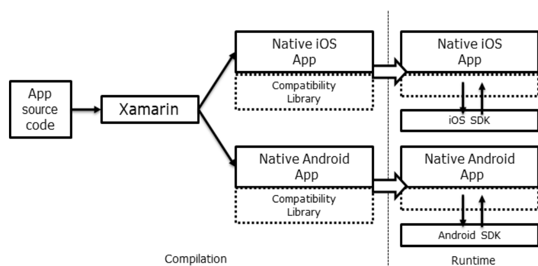

Fig. 1. Procedure of Xamarin app development [2]

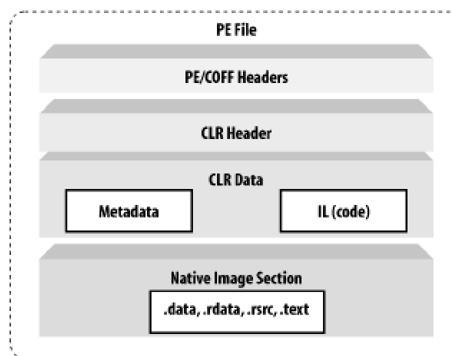

Fig. 2. Format of .NET PE files[5]

3.2 모노 런타임 환경

자마린으로 개발된 앱들의 경우, 소스코드로부터 IL 코드가 생성되고, 생성된 IL 코드는 CLR 환경에서 특정 플랫폼에 맞는 네이티브 코드로 변환되어 실행된다. 자마린을 통해 개발된 C# 기반의 안드로이드 앱도 컴파일 후 IL 코드 형태로 DLL 파일에 존재한다. 이 안드로이드 앱은 CLR 역할을 하는 모노 런타임을 통해 실행되며 기존의 안드로이드 실행 환경인 DVM(Dalvik Virtual Machine) 또는 ART(Android Runtime)과 함께 상호작용한다.

모노 런타임이 포함된 안드로이드 실행 구조가 Fig. 3.에 나타나 있다[8]. 모노 런타임에서 제공되는 .NET API는 XML, DB, 직렬화(Serialization), 입출력, 스트링(String), 네트워킹(Networking) 기능들을 제공한다[9]. 하지만 안드로이드 OS가 제공하는 Audio, Telephony, OpenGL 기능 등은 안드로이드 SDK 또는 자바 API로만 접근이 가능하다. 따라서 모노 런타임과 안드로이드 실행환경이 서로 통신해야 할 필요가 있다.

이를 위해, 자마린은 Managed Callable Wrappers(MCW)를 제공한다. MCW는 C# 코드에서 Java 코드를 호출해야 할 때 사용되는 자바 네이티브 인터페이스(JNI) 브릿지로 Binding Library형태로 제공하고 있다. Binding Library는 C# 코드에서 자바 코드를 호출할 수 있도록 자바 라이브러리에 대한 C# Wrapper를 생성하고 이를 C#에서 호출하는 방식을 이용하고 있다. 반대로 안드로이드 런타임에서 C#코드를 호출하기 위해서 Android Callable Wrappers(ACW)를 제공하고 있다.

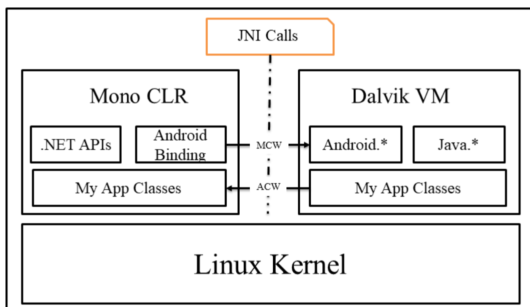

Fig. 3. Android execution environment including Mono runtime [8]

IV. 자마린 앱에 대한 정적 역공학 방법

이장에서는 자마린으로 개발된 안드로이드 앱의 구조를 분석하고, 정적으로 역공학하는 방법을 기술한다. 본 논문에서는 자마린에서 C#으로 개발된 안드로이드 앱을 '**자마린 앱**'이라고 부른다.

4.1 자마린 앱의 구조

기존의 Java 언어로 작성된 안드로이드 네이티브 앱과 비교하여 보면, 자마린 앱에는 assemblies 폴더와 lib 폴더가 추가로 구성된다. Fig. 4.는 마켓에 배포되고 있는 Snap Attack 이라는 자마린 앱의 구조를 보여준다. 해당 앱을 압축해체 후 살펴보면 assemblies 폴더에는 DLL 파일들이 존재하는데, 이 DLL은 C# 코드로부터 번역된 IL 코드가 존재한다. lib 폴더에는 C/C++ 기반의 네이티브 라이브러리인 so 파일들, 즉 모노 런타임 환경을 제공하기 위한 라이브러리들이 존재한다.

assemblies 폴더의 파일들은 안드로이드 MainActivity, Java 코드에서 C# 코드를 호출할 수 있게 해 주는 JNI, 안드로이드 API에 대한 C# 바인딩, 암호화 관련 API, 파일 입출력 기능 등을 제공하며, lib는 모노 런타임 환경과 JIT 컴파일 기능 등을 제공한다. 각 파일들의 주요 기능은 Table 1.과 같다.

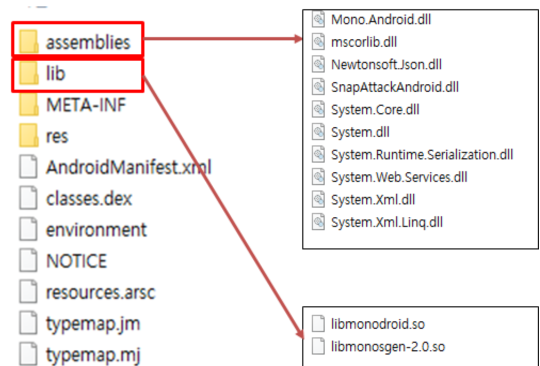

Fig. 4. The structure of the xamarin app (Snap Attack)

Table 1. Main features of files included in Xamarin app

Folder name	File name	Description
/Assemblies	SnapAttackAndroid.dll	Compilation results of C # developer code
	Mono.Android.dll	- Providing C # binding functionality for the Android API - Supporting for namespaces such as Java.Lang and Java.Net
	Mono.security.dll	Providing Cryptographic API
	Java.Interop.dll	Providing JNI functionality to call C # code from Java
	System.*.dll	File I / O function
/lib	libmonodroid.so	Providing a mono runtime environment
	libmonos-gen-2.0.so	Providing JIT function

4.2 자마린 앱에 대한 정적 분석 기법

정적 역공학은 실행코드를 역어셈블(disassemble)하거나 역컴파일(decompile)하는 과정을 포함한다. 자마린 앱의 경우, 실행파일은 Java 바이트코드, .NET 프레임워크의 IL 코드를 포함한 DLL 파일, 네이티브 라이브러리인 so 모듈(C/C++ 기반 바이너리 코드)들로 구성된다. 서로 다른 언어로 개발된 코드를 역공학하기 위해서 각 실행코드에 적합한 역공학 도구를 사용해야 한다.

앱의 바이트코드를 자바 소스코드로 역컴파일하기 위해 무료 도구인 Jadx[10]나 상용 도구인 JEB[11]를 사용할 수 있다. 또한 IL코드로 구성된 DLL 파일을 .NET 기반 C# 코드로 역컴파일하기 위해서는 무료 도구인 ILSpy[12]나 상용 도구인 .NET Reflector[13]를 사용할 수 있다. 라이브러리인 so 파일을 어셈블리나 C 기반 의사코드로 역공학하기 위해서는 IDA pro[14]를 사용할 수 있다. 자마린 앱의 구성 모듈별 필요한 정적 역공학 도구는 Fig. 5.와 같다.

자마린 앱을 정적으로 분석하기 위한 절차는 Fig. 6.과 같이 크게 4단계로 나눌 수 있다. 가장 먼저 분석 대상 DLL 파일을 파악해야 한다. 이를 위해서 자마린 앱 내에 있는 DEX 파일을 역공학하여 DEX 파일 내 존재하는 MonoPackageManager_Resources 클래스를 분석해야 한다. 해당 클래스는 개발자가 작성한 DLL 파일과 서드파티 라이브러리 리스트들을 Assemblies 배열에 저장하고 있다. 배열에 저장된 DLL 파일 이름들을 확인하여 분석 대상 DLL 파일

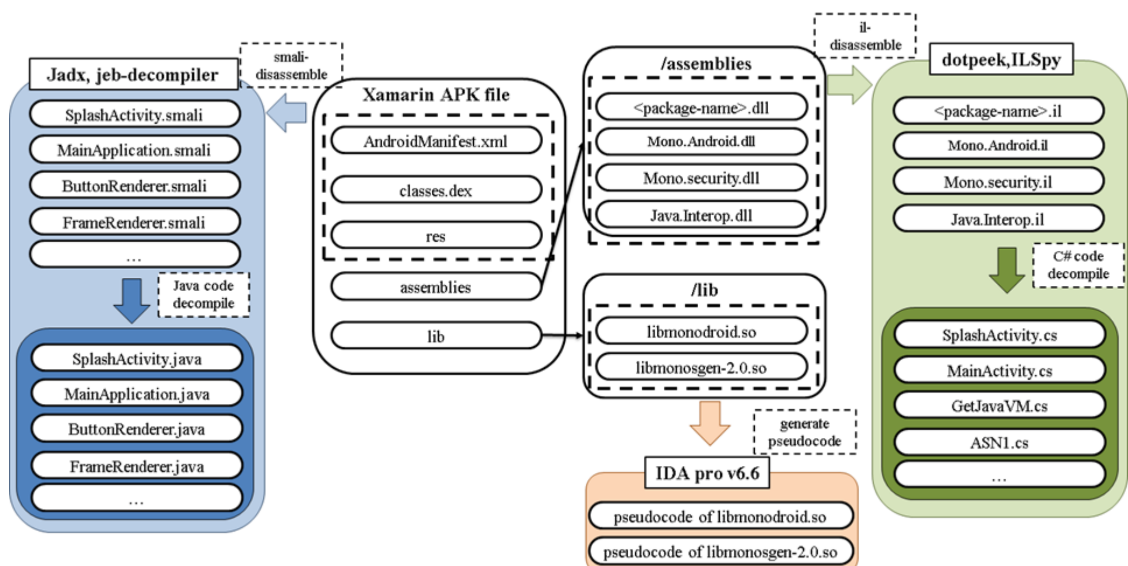

Fig. 5. File structure and static analysis tools of a Xamarin app

들을 파악할 수 있다. Fig. 4.에 나타난 Snap Attack 앱의 경우 Assemblies 배열 첫 번째 값으로 SnapAttackAndroid.dll의 이름을 갖고 있다. 해당 파일은 모노 런타임 상에서 가장 먼저 실행되는 진입점(보통은 특정 액티비티)을 포함하고 있다.

다음으로 SnapAttackAndroid.dll에서 진입점을 파악하기 위해서 AndroidManifest.xml을 분석해야 한다. 일반적인 안드로이드 앱의 진입점은 AndroidManifest.xml 상에서 〈action android:name="android.intent.action.MAIN"/〉 속성을 가진 액티비티(Activity)라고 할 수 있다. Snap Attack 앱의 경우 해당 속성을 갖는 액티비티는 DEX 파일 상의 RootActivity로 자바 코드 상에서의 가장 먼저 실행되는 진입점이다.

RootActivity에는 (C/C++로 작성된) 네이티브 함수인 Runtime.register( )함수를 호출하고 있다. 해당 코드는 C#코드로 작성된 RootActivity를 등록하는 코드로, 등록이 완료 된 후 SnapAttackAndroid.dll 상의 RootActivity가 실행된다. 해당 코드는 libmonodroid.so 파일 내에 네이티브 함수인 Java_mono_android_Runtime_register( ) 함수로 선언되어 있다. 해당 함수는 호출 과정 중에 monodroid_runtime_invoke( )를 호출한다. invoke( )함수는 인자 값으로 어셈블리 이름, 실행할 클래스 및 메소드 이름, 파라미터를 요구하며 인자 값에 맞

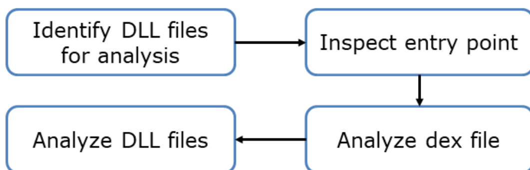

Fig. 6. Static reverse engineering procedures for Xamarin apps

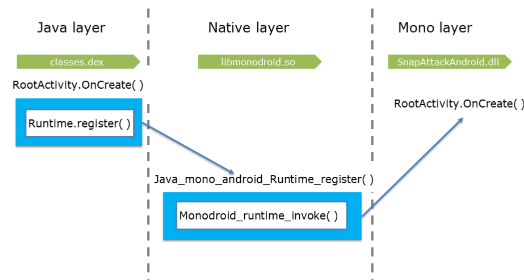

Fig. 7. Main control flow in a Xamarin app

는 어셈블리의 코드를 실행한다. libmonodroid.so 파일은 IDA Pro를 통해서 역공학을 진행하였다. Fig. 7.은 Snap Attack 앱의 정적 역공학 과정을 그림으로 표현한 것이다.

해당 앱은 자바영역에서 Runtime.register( ) 함수와 네이티브 영역에서 monodroid_runtime_invoke( ) 함수를 차례로 거쳐 모노 영역의 진입점인 RootActivity.OnCreate( ) 함수가 실행된다.

이와 같이 안드로이드 플랫폼에서 자마린 앱을 분석하여 본 결과, 기존의 Java로 작성된 네이티브 앱과 비교하여 차이점은 다음 Table 2.와 같다.

자마린 앱은 네이티브 앱과 달리 C# 기반 DLL 파일을 모노 런타임에서 실행하면서 기존 안드로이드 런타임과 상호작용하는 형태로 동작된다. 그리고 모노 런타임을 제공하기 위해 별도의 so 파일을 포함하고 있다. 따라서 자마린 앱을 정적으로 분석하기 위해서는 DLL 파일과 모노 런타임을 위한 so 파일을 분석하기 위한 과정이 기본적으로 필요하다.

Table 2. Differences between native and Xamarin apps

	Native apps	Xamarin apps
Files to be analyzed	- DEX file, - so file (exists if created by the developer)	- DLL files, - DEX file, - so file (exists by default for mono runtime)
Static analysis tools	JEB, Jadx, IDA Pro	ILspy, Reflector, JEB, Jadx, IDA Pro

V. 정적 역공학 방해 기법 및 극복

이 장에서는 자마린 앱의 구성 파일 중 C# 언어와 관련된 DLL 파일에 대한 코드 난독화(code obfuscation) 기법 및 이에 대한 분석 기법에 대해 설명한다. 닷넷 프레임워크에서 실행되는 프로그램을 대상으로 하는 코드 난독화 도구는 공통 중간 언어(Common Intermediate Language, CIL) 기반으로 적용된다. 관련 난독화 도구로는 Babel Obfuscator, Aldaray Rummage, Crypto Obfuscator For .Net,.Net Anti-Decompiler, Dotfuscator 등의 상용도구와, ConfuserEx, DotRefiner 등의 오픈소스 프로젝트가 있다. 이러

한 난독화 도구들은 IL 코드 분석을 방해·지연시키기 위해 다양한 데이터/코드 변환 기법들을 적용하고 있다. 대표적인 기법으로 덤프 방지(anti-dump), 제어흐름 보호 기법, 문자열 난독화, 리소스 암호화 등이 존재한다.

본 논문에서는 Crypto Obfuscator For .Net 도구와 ConfuserEx 도구의 난독화 기법을 자마린 앱의 DLL 파일에 적용가능한 지 확인하였다. Table 3은 자마린 앱에 적용 가능한 Crypto Obfuscator For .Net과 ConfuserEx의 정적 분석 방해 기법을 나타낸다. 두 도구들이 제공하는 정적 분석 방해기법을, Snap Attack 앱의 핵심 DLL 파일인 SnapAttackAndroid.dll에 각 기법별로 적용하고, APKtool을 이용해 재패키징(repackaging)하였다. 재패키징한 앱을 실행한 결과, ildasm[1) 사용 방지(Anti ildasm Protection), 심볼 이름 변경 (Symbol renaming), 제어 흐름 난독화(Control flow obfuscation), 문자열 암호화(String encryption) 기법 등 네 가지 기법은 적용 가능했으나, 이외에 기법들은 적용되지 않았다.[2)

자마린 앱에 적용되지 않는 기법들의 원인을 조사해 보니 Fig. 8.과 같이 공통적으로, DLL 파일 내에 존재하는 클래스인 *internal class 〈Module〉*에 난독화 기법과 관련된 코드가 포함되어 있었다. ConfuserEx에 의해 *internal class 〈Module〉*에 추가되는 코드들은 윈도우 환경에서만 동작하도록 고안되어 있어, 안드로이드 환경에서는 해당 IL 코드를 JIT 컴파일을 제대로 수행하지 못해, 실행이 불가능한 것으로 보인다. 이에 반해, 제어 흐름 난독화 기법 등 자마린 앱에 적용 가능한 기법들은 *internal class 〈Module〉* 영역에 관련 코드를 추가하지 않아 안드로이드 환경에서도 정상적으로 실행됨을 확인했다.

```
using [...]

internal class <Module>
{
    static <Module>()
    {
        <Module>.();
    }

    [DllImport("kernel32.dll", EntryPoint = "VirtualProtect")]
    internal static extern bool (IntPtr, uint, uint, ref uint);

    internal unsafe static void ()
    {
        Module module = typeof(<Module>).get_Module();
        string fullyQualifiedName = module.get_FullyQualifiedName();
        bool flag = fullyQualifiedName.get_Length() > 0 && fullyQualifiedName.get_Chars(0) == '<';
        byte* ptr = (byte*)((void*)Marshal.GetHINSTANCE(module));
        byte* ptr2 = ptr + *(uint*)(ptr + 60);
```

Fig. 8. Inserted Code after Applying DLL Tamper Protection

1) IL disassembler를 말함

2) Table 3의 '문자열 암호화' 난독화는 Crypto Obfuscator for .Net 도구에서만 지원되는 기능임

Table 3. Anti-Static Analysis Techniques of ConfuserEx and Crypto Obfuscator For .Net

Function	Description
Anti ildasm Protection	prevents the use of iasm.exe, a tool that disassembles DLL files into IL code
Symbol renaming	converts the name of the symbol (symbol) to make it difficult to read the decompiled code
Control Flow Protection	modifies the execution flow by changing the code in the method
Anti Tamper Protection	ensures the integrity of your app
Anti Debug Protection	prevents debugging
String Encryption	encrypts all literal strings

이에 자마린 앱에 적용 가능한 4가지 난독화 기법들을 적용하고 이를 분석하여 정적 분석하는 방법을 실험하였다.

5.1 ILDASM 사용방지 및 극복 방법

IL Disassembler (ILdasm)는 IL Assembler의 보조 도구로서 DLL 파일을 역어셈블하여 분석가가 알아보기 쉬운 IL 코드를 생성한다. 이 도구는 윈도우 환경 개발 도구인 비주얼 스튜디오에서 닷넷 응용프로그램 개발을 지원하기 위해 기본적으로 제공된다. ConfuserEx는 ildasm의 사용을 방지하는 기법을 제공한다. ildasm 사용을 방지하는 난독화 기법이 적용된 어셈블리를, ildasm 도구의 입력으로 제공했을 때 Fig. 9.와 같은 오류 문구를 출력하게 된다. 이 난독화 기법을 확인해 보면, Fig. 10.과 같이 DLL 파일 내에 ildasm 방지 속성인 SuppressIldasmAttribute 속성값을 추가하는 기능으로만 구성되어 있다. ildasm 도구는 DLL 파일 내에 SuppressIldasmAttribute 속성이 존재할 경우 해당 어셈블리를 분석하지 않는다. 이는 높은 수준의 난독화 기법이 아니어서, 이를 우회하는 것이

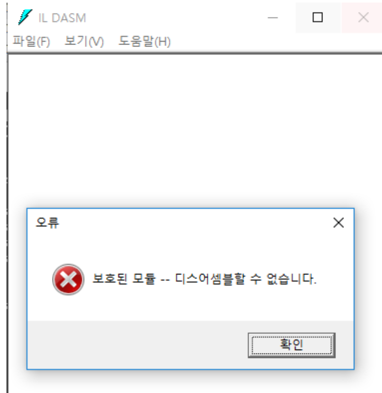

Fig. 9. Applying the ildasm prevention technique

```
[assembly: ExportEffect(typeof(ListViewSelectionOnTopEffect), "ListViewSelectionOnT
[assembly: ExportRenderer(typeof(NavigationView), typeof(NavigationViewRenderer))]
[assembly: ExportRenderer(typeof(SearchBar), typeof(EvolveSearchBarRenderer))]
[assembly: ResolutionGroupName("Xamarin")]
[module: ConfusedBy("ConfuserEx v1.0.0")]
[module: SuppressIldasm]
```

Fig. 10. Attribute of SuppressIldasmAttriubute

어렵지 않다.

SuppressIldasmAttribute 속성이 존재하는 DLL 파일을 역어셈블하기 위하여, 본 논문에서 SuppressIldasmAttribute를 무시하고, ildasm이 수행되도록 하는 두 가지 방법을 찾았다.

첫 번째 방법은 SuppressIldasmAttribute 속성이 적용된 어셈블리를 Mono.Cecil 라이브러리를 이용해 해당 속성을 제거한 어셈블리를 재생성하는 방법이다. 하지만 본 방법은 윈도우 환경에서 구동하는 어셈블리에는 적용이 가능하지만, 자마린 앱의 어셈블리에서는 Mono.Cecil. AssemblyResolutio-nException 오류로 인해 어셈블리 재생성이 불가능하다.

다른 방법은 ildasm 도구에 대한 실행파일을 수정하는 것이다. ildasm 실행파일은 어셈블리를 입력 값으로 하며, 해당 어셈블리를 IL 코드로 변환해주는 기능을 하는데 Fig. 11.과 같이 어셈블리 내에 SuppressIldasmAttribute가 존재할 경우 이를 Fig. 9.의 오류 문구와 함께 IL 변환 기능을 수행하지 않는다. 이러한 루틴을 우회하기 위해 ildasm 실행파일 내에 존재하는 SuppressIldasmAttrib-ute의 텍스트를 검색하고 이를 임의의 다른 텍스트로 수정할 경우 어셈블리 내의 SuppressIldasm-Attribute 속성이 존재하더라도 어셈블리 파일들을 IL 코드로 변환할 수 있다.

```
<Module>....Syst              <Module>....Syst
em.Runtime.Compi              em.Runtime.Compi
lerServices.Supp      →       lerServices.Supp
ressIldasmAttrib              Dankook Univ.  b
ute.m.s.c.o.r.l.              ute.m.s.c.o.r.l.
i.b.....%s %s %s              i.b.....%s %s %s
```

Fig. 11. Bypassing the SuppressIldasmAttribute attribute

5.2 심볼 이름 변경 및 제어흐름 난독화의 극복

심볼 이름 변경 기법은 타입, 메소드, 특성, 이벤트, 필드, 네임 스페이스 및 메소드 매개 변수의 이름을 분석하기 어렵게 변경하는 것이다. 예로, 오버로딩을 이용한 메소드 이름 변경 기법, 멤버 이름을 난독화하면서 사용자 지정 유니코드 문자로 변경하는 기법 등이 있다. 이 기법 적용 후에는 심볼 이름이 다르게 변경되어 원래대로 역공학되지 않는다. 따라서 난독화된 코드를 이해하기 어렵게 하는 효과를 가져 온다. Fig. 12.는 기존 Snap Attack 앱의 어셈블리를 난독화한 결과이며, 심볼 이름들이 알아보기 힘들게 바뀌었다는 것을 확인할 수 있었다.

그러나 심볼 이름 변경 기법이 적용된 앱을 역공학하여 보니, 해당 기법은 분석가로 하여금 정적분석을 어렵게 만들긴 하지만, 실제로는 분석이 가능하여 해당 앱의 구조나 실행 흐름을 파악할 수 있었다.

자마린 앱에 적용 가능한 다른 난독화 기법으로 제어 흐름 변경 기법이 있다. 제어 흐름 변경 기법은 기존 메소드 코드에 더미 코드를 삽입하거나, if문, switch문, for문 등을 추가하여 메소드의 실행 흐름을 변경한다. 이2처럼 제어 흐름이 변경되면 코드의 양이 많아지며 의미 없는 제어흐름이 추가되어 분석

```
public override void OnWindowFocusChanged(bool hasFocus)
{
    base.OnWindowFocusChanged(hasFocus);
    if (hasFocus)
    {
        while (true)
        {
            switch (4)
            {
            case 0:
                continue;
            }
            break;
        }
        if (!true)
        {
            RuntimeMethodHandle arg_1C_0 = methodof(RootActivity.OnW
        }
        if (this.isSleeping)
        {
            while (true)
            {
                switch (6)
                {
                case 0:
                    continue;
                }
```

Fig. 12. Snap Attack app with symbol renaming

가나 공격자로 하여금 역공학을 어렵게 할 수 있다. Snap Attack 앱의 어셈블리를 대상으로 제어흐름 난독화를 적용한 결과, 분석 코드의 양이 원본에 비해 많아지며 제어 흐름 또한 변경되어 분석이 상대적으로 어려워지는 것을 확인하였다. 그러나 난독화가 적용되어 있음에도 불구하고, 결국 높은 난이도는 아니기에 조금의 수고가 더해진다면 여전히 분석가능한 것을 확인하였다.

5.3 문자열 암호화 극복 방법

문자열 암호화(string encryption) 기법은 암호화 알고리즘을 이용해 소스코드 내 존재하는 문자열들을 암호화하여 분석하기 어려운 형태로 변경하여 준다. 해당 기법은 Crypto Obfuscator for .Net 도구에서 제공되고 있다. Fig. 13은 문자열 암호화 적용 전/후의 소스코드를 나타낸다.

문자열 암호화가 적용된 코드를 분석하기 위해서 복호화 루틴을 통해 복호화 과정을 수행해야 한다. Crypto Obfuscator for .Net 도구는 Fig. 13(b)의 표시된 해시값과 동일한 클래스를 생성하여 복호화 루틴을 저장하고 있는 것을 확인하였다. 문자열 암호화라는 난독화 방법을 극복하기 위해서 인터넷에 공개된 도구를 이용하여 역난독화(de-obfuscation)하였다. de4dot[14]는 오픈소스 .Net 역난독화 도구로 Crypto Obfuscator for .Net로 암호화된 문자열을 복호화하여 준다. 따라서 해당 도구를 이용하여 문자열 암호화가 적용된 코드에서 난독화가 적용되기 전의 원본 코드를 획득 할 수 있었다.

```
(this.GetType(), "(Lbolts/AppLink;Landroid/os/Bundle;Landroid/os/Bundle;)V",
"(Lbolts/AppLink;Landroid/os/Bundle;Landroid/os/Bundle;)V", constructorParam
```

(a) Before applying string encryption

```
c6c108bb6ae2823b2f11f919bde093b91.cfa38a6c72c62e03f6467cab021cec57c(2160),
919bde093b91.cfa38a6c72c62e03f6467cab021cec57c(2160), jvaluePtr);
```

(b) After applying string encryption

Fig. 13. Source code before and after applying string encryption

VI. 결 론

본 논문에서는 “자마린 프레임워크에서 C#으로 개발된 안드로이드 앱”(자마린 앱)의 구조를 분석하고 C# 코드를 실행하기 위한 모노 런타임 환경 동작 원리를 분석하였다. 또한 자마린으로 개발된 안드로이드 앱을 체계적으로 정적 분석하는 기법에 대해 제안하였다. 제안 기법은 소스코드가 제공되지 않는 경우에도 적용할 수 있는 기법으로 악성 자마린 앱을 분석하는데 활용될 수 있다. 정적 분석은 주어진 코드를 실행하지 않고 제어·데이터 흐름이나 특정한 코드 패턴을 조사하는 것으로, 전체 코드를 커버하면서 분석할 수 있다. 또한, 실행 오버헤드도 발생하지 않는다. 본 논문에서는 난독화가 적용된 앱에 대해서도 코드를 정적으로 분석하는 실험을 수행하였다. 단점으로는 패킹되었거나 암호화된 코드를 분석하기 어렵다는 점이다.

현재 자마린 앱을 대상으로 분석 기법에 대해서는 거의 연구된 것이 없다. 본 논문에서 제시한 기법은 자마린으로 제작된 악성 앱을 분석하기 위한 체계적인 방법으로 활용될 수 있다. 향후에는 패킹된 자마린 앱을 분석할 수 있는 동적 분석 기법, 그리고 자마린에서 안드로이드 바인딩(Android binding)을 거치지 않고 악성행위를 할 수 있는 요소를 도출하고 이에 대한 분석 방법에 대해서 연구할 계획이다.

References

[1] M. Willocx, J. Vossaert, and V. Naessens, “A quantitative assessment of performance in mobile app development tools,” Mobile Services (MS), 2015 IEEE International Conference on. IEEE, 2015.

[2] N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode, “Testing cross-platform mobile app development frameworks (t),” In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on IEEE, pp. 441-451, 2015.

[3] M. Martinez, and S. Lecomte, “Towards the quality improvement of cross-

platform mobile applications," In Mobile Software Engineering and Systems (MOBILESoft), 2017 IEEE/ACM 4th International Conference on IEEE, pp. 184-188, 2017.

〔4〕 P. Marius, "Considerations Regarding the Cross-Platform Mobile Application Development Process," Academy of Economic Studies. Economy Informatics Vol. 13, no. 1, 2013.

〔5〕 S. M. Pontiroli and F. R. Martinez, "The Tao of .NET and PowerShell Malware Analysis," Virus Bulletin Conference, 2015.

〔6〕 A. Mylonas, S. Dritsas, B. Tsoumas and D. Gritzalis, "On the feasibility of malware attacks in smartphone platforms," International Conference on E-Business and Telecommunications, 2011.

〔7〕 A. Mylonas, S. Dritsas, B. Tsoumas and D. Gritzalis, "Smartphone security evaluation The malware attack case," Security and Cryptography (SECRYPT), 2011 Proceedings of the International Conference on. IEEE, 2011.

〔8〕 R. Mark. Xamarin mobile application development for Android. Packt Publishing Ltd, 2014.

〔9〕 jadx, https://github.com/skylot/jadx

〔10〕 JEB, https://www.pnfsoftware.com/

〔11〕 ILspy, https://sourceforge.net/projects/ilspyportable/

〔12〕 .NET Reflector, https://www.red-gate.com/products/dotnet-development/reflector/

〔13〕 IDA Pro, https://www.hex-rays.com/products/ida/

〔14〕 de4dot, https://github.com/0xd4d/de4dot

〈저자소개〉

임 경 환 (Kyeonghwan Lim) 학생회원
2015년 2월: 단국대학교 컴퓨터학과 졸업
2016년 8월: 단국대학교 컴퓨터학과 석사
2016년 9월~현재: 단국대학교 컴퓨터학과 박사과정
〈관심분야〉 모바일 보안, 시스템 보안

김 규 식 (Gyoosik Hong) 학생회원
2016년 2월: 단국대학교 응용컴퓨터공학과 졸업
2018년 2월: 한국대학교 컴퓨터학과 석사
〈관심분야〉 모바일 보안, 시스템 보안

심 재 우 (Jaewoo Shim) 학생회원
2017년 2월: 단국대학교 소프트웨어학과 졸업
2017년 3월~현재: 단국대학교 컴퓨터학과 석사과정
〈관심분야〉 시스템 보안, 역공학

조 성 제 (Seong-je Cho) 종신회원
1989년: 서울대학교 컴퓨터공학과 졸업
1991년: 서울대학교 컴퓨터공학과 공학석사
1996년: 서울대학교 컴퓨터공학과 공학박사
1997년 3월~현재: 단국대학교 컴퓨터학과/소프트웨어학과 교수
〈관심분야〉 시스템 보안 및 악성코드 분석, 소프트웨어 보증, 시스템소프트웨어, 임베디드 소프트웨어 등