정 보 보 호 학 회 지
제 32 권 제6호, 2022. 12

효율성 관점의 APK 분석 및 안티 디버깅 우회 방법

박 민 수*

요 약

안드로이드 환경은 디컴파일을 통해 개발자가 작성한 코드를 쉽게 획득할 수 있고, 해당 코드의 분석을 통해 유사 앱을 개발하거나 기능을 변조할 수 있다. 이러한 문제에 대응하기 위해 다양한 난독화, 안티 디버깅 기법을 개발하여 적용하고 있다. 하지만 해당 기법들은 충분한 시간과 노력을 투자한다면 분석 및 우회가 가능하므로 개발자는 분석을 원천적으로 막기보다 분석을 지연하는 목적으로 활용하고 있다. 안티 디버깅을 지원하는 대부분의 상용 SW 또한 이런 한계를 크게 벗어나지 않고 있으며 적용하는 안티 디버깅 기법의 종류와 수의 차이만 있다. 특히 기존의 연구는 안티 디버깅 기법의 우회에 집중하고 있으며 해당 연구는 분석을 위한 사전 단계의 개념이며 실제 분석 단계에서는 활용하지 못해 분석 효율을 올리지는 못한다. 본 연구에서는 분석의 관점에서 안티 디버깅을 무력화하고, 실제 분석 시 활용할 수 있는 방법을 소개한다.

Ⅰ. 서 론

안드로이드는 바이트코드 형태의 자바를 지원하고 있다. 자바는 바이너리코드와 달리 높은 생산성과 호환성을 가지지만 디컴파일을 통해 코드가 그대로 노출되는 문제를 가진다.

이는 자바의 구조적인 문제로 디컴파일을 통해 원본 코드, 리소스를 모드 획득할 수 있다. 공격자는 획득한 코드와 리소스를 이용하여 코드를 복제 및 변형하거나 치명적인 오류나 버그를 발견하여 이를 악용할 수 있다. 디컴파일 도구 또한 무료로 배포되고 지속적인 업데이트를 통해 강력한 기능을 지원한다. 따라서 누구나 원한다면 안드로이드 앱을 디컴파일하고 이를 분석할 수 있다.

개발자는 디컴파일을 방지하기 위해 다양한 난독화 기법 및 안티 디버깅 기법을 적용하고 있다. 하지만 이것은 단지 분석을 지연시키기 위한 방법으로 원천적으로 디컴파일을 통한 분석을 막을 수는 없다. 상용 보안 솔루션 또한 동일한 한계를 가지고 있으며 이를 보완하기 위해 여러 난독화 및 안티 디버깅 기법을 동시에 적용하여 분석을 지연시키거나 방해하는 형태를 취한다. 즉, 분석을 위해서는 해당 기법들을 하나씩 무력화시키거나 우회하여야 하며, 이는 분석에 소요되는 비용을 증가시켜 분석을 방해하는 효과를 가진다. 다만 안티 디버깅 기법이 발전함에 따라 디컴파일러와 같은 분석 도구와 기법도 발전하고 있어 큰 효과가 없다.

본 논문에서는 주로 사용되는 안티 디버깅 기법을 소개하고, 분석에 활용할 수 있는 상용 및 무료 도구의 활용 사례를 소개한다.

Ⅱ. 관련 연구

본 장에서는 주로 사용되는 안티 디버깅 기법과 디컴파일 도구에 대해 소개한다. 구글은 기본적으로 안드로이드 apk의 보호를 위해 Proguard를 제공하지만 해당 도구는 단순 난독화 도구로 안티 디버깅 및 디컴파을 방어하지는 못한다. 따라서 개발자는 직접 안티 디버깅 기법을 개발 및 적용하거나 상용 도구를 사용하여 이를 지원한다.

2.1. 안티 디버깅

난독화는 안티 디버깅 기법 중 가장 대표적인 기법으로 분석을 방해하기 위해 함수 및 변수 명을 임의의 의미없는 문자로 변경하는 방법이다. 난독화는 어디에 어떻게 적용할 것인지에 따라 레이아웃, 제어, 데이터 난독화로 구분할 수 있다[1].

레이아웃 난독화는 식별자, 주석과 같이 코드 실행과는 상관없는 요소를 임의로 생성된 값으로 변경하는 것으로 분석 시 함수 및 변수의 이름으로 동작을 추측

* LG전자 HE사업본부 (선임연구원, minsoon2@gmail.com)

하거나 예측할 수 없도록 하여 분석을 어렵게 만든다. 하지만 단순히 이름을 변경하는 정도의 수준으로 실제 동작을 변경하거나 숨길 수 없는 한계가 있다.

제어 난독화는 코드의 실행 흐름을 변형하여 분석을 어렵게 만드는 방법으로 코드를 임의로 분리하거나 다수의 분리된 코드를 하나로 합쳐 원래의 흐름을 알 수 없도록 한다. 이때 더미 코드를 추가하여 복잡성을 증가시킬 수 있다. 하지만 이 기법 또한 원래의 기능을 원천적으로 숨길 수는 없으며 단지 코드의 흐름을 복잡하게 만들어 분석을 방해하는 목적으로 사용한다. 이로 인해 속도가 느려지거나 의도치 않은 오류가 발생할 수 있다.

데이터 난독화는 해당 앱이 가지고 있는 데이터를 보호하기 위해 암호화나 인코딩 기법을 이용하는 것을 말한다. 하지만 해당 코드가 개발자의 의도대로 동작하기 위해서는 코드 실행 과정에서 원본 데이터로 처리 후 사용해야 하므로 분석을 통해 원본 데이터로 복원이 가능하다. 예를 들어 문자를 암호화 할 경우 앱 실행 단계에서 암호화 된 문자를 복호화하여 사용해야 한다. 즉, 분석을 통해 복호화 함수를 찾아내고 이를 통해 원본 데이터를 획득할 수 있다.

이와 같이 난독화는 단지 코드 분석을 방해하기 위한 용도로만 사용할 수 있으며, 원천적으로 분석을 능동적으로 막을 수는 없다. 따라서 각 보안 솔루션과 개발자는 분석을 원천적으로 막기 위해 디컴파일 및 디버깅 도구의 동작을 방해하거나 디버깅 상태를 탐지하여 코드의 실행을 강제로 종료시키는 방법을 사용한다. 상용 보안 솔루션에서는 실행 압축 기법을 사용한 loader의 개념을 도입하여 실제 동작이 일어나는 dex, so와 같은 파일은 앱 패키지 내에 보이지 않도록 숨기고, 실제 loader의 역할을 하는 코드를 작성하여 앱을 배포한다. 앱은 설치 후 실행 단계에서 실제 해당 앱의 동작을 담당하는 dex, so 파일을 동적으로 디코딩하거나 복호화하고 동적으로 로딩한다. 이때 loader를 분석한다면 숨겨둔 dex, so가 유출되어 보호의 의미가 사라지므로 loader를 보호하기 위한 다양한 안티 디버깅 기법이 또 다시 중첩으로 적용된다. 이러한 개념은 난독화와 다른 목적으로 개발되었다. 난독화는 실행 코드의 노출에는 관여하지 않고, 분석의 난이도를 증가시켜 분석을 방해하는데 목적을 둔다. 하지만 실행 압축 기법은 실행 코드의 노출을 원천적으로 막는 것을 목적으로 하며 실제 동작을 담당하는 dex, so 파일을

```
import android.app.Activity;

public class ɔ extends Activity {
    public ɔ() {
        super();
    }
}
```

〔그림 1〕 실행 압축이 적용된 코드

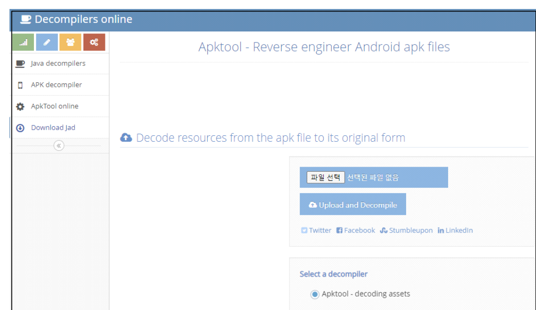
획득할 수 없다면 공격자는 실제 코드를 볼 수 없어 분석을 막을 수 있다.

그 외 다양한 안티 디버깅 기법이 공유되거나 개발되고 있으며 주로 루팅 여부 판단, 안드로이드의 프로세스 관리 체계를 활용하거나 에뮬레이터 여부 판단, API 또는 데이터 제약사항과 같은 요소를 이용하여 디버깅을 방해한다.

2.2. 디버깅 도구

효과적인 apk 분석을 위해서는 디컴파일이 필수적으로 요구된다. 현재 다양한 무료 및 유료 디컴파일 도구가 개발되어 배포되고 있으며 가장 널리 알려진 도구로는 apktool[2], dex2jar[3], jadx[4], jd-gui[5], JEB[6]가 있다. JEB를 제외한 나머지 도구는 모두 무료로 배포되고 있으며 각 디컴파일 도구의 성능 및 동작 방법, 지원하는 기능에 차이가 있다. 특히 최근에는 해당 도구를 직접 다운로드 하지 않고 웹 사이트에서 온라인 형태로 도구를 이용할 수도 있으며, 사용자는 해당 웹 페이지에 분석 대상 apk를 업로드 하는 것만으로 분석 결과를 다운로드 할 수 있다[7]. 대부분의 웹 사이트에서 제공하는 온라인 디컴파일 서비스는 apktool을 활용한다.

이중 jadx는 오픈소스 도구로 다른 도구에 비해 업



〔그림 2〕 온라인 디컴파일 도구

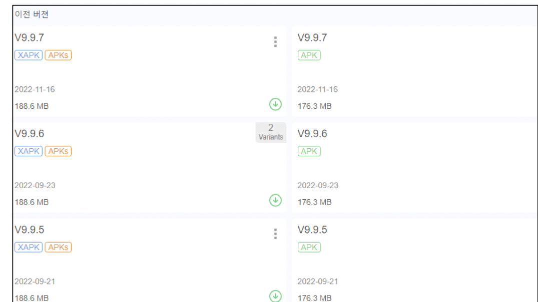
데이트 주기가 짧고 사용 중 불편한 부분이나 문의 사항이 생겼을 경우 공식 git을 통해 의견을 전달하거나 문의 사항을 해결할 수 있다. 또한 jadx만으로 디컴파일과 분석 및 편집을 모두 할 수 있어 분석 효율이 높다. JEB의 경우 상용 도구이지만 다른 디컴파일 도구에 비해 디컴파일 성능이 높아 다른 도구에서 제대로 분석하지 못하는 코드를 분석할 수 있다. 다만 분석 결과가 온전한 자바 코드의 형태가 아닌 실행 흐름을 goto로 별도로 표시하여 코드의 완성 형태에서는 성능이 떨어진다. 그 외 jd-gui나 apktool은 분석에 필요한 과정 중 일부 과정만을 지원하거나 특정 기능만을 제공하여 다른 도구와 함께 사용해야 하는 한계가 있다. 이와 같은 디버깅 도구를 활용하는 방법은 3장에서 설명한다.

Ⅲ. 디버깅 및 분석

안티 디버깅 우회 및 분석 방법에 대한 연구는 지금까지 많이 진행되었다. 하지만 현 시점에서도 기본적인 난독화가 적용되지 않은 앱이 대부분이며, 적용을 하더라도 일부만 적용되거나 단순히 여러 기법을 한꺼번에 적용한 수준을 벗어나지 않는다. 본 논문에서는 안티 디버깅 기법의 우회 보다 실제 분석을 진행하는데 효율을 높일 수 있는 부분에 집중한다. 본 논문에서 소개하는 방법은 대부분의 분석에 활용할 수 있으며, 간단한 도구의 활용 및 관점의 변화만으로도 분석 효율을 향상 시킬 수 있다.

3.1. Archive 활용

우선 보안 솔루션의 적용 및 활용은 앱의 성능, 개발 및 유지보수의 관점에서 비용을 발생시킨다. 의도치 않은 버그를 일으키거나 성능의 하락이 발생하고, 상용 보안 솔루션의 도입으로 인한 금전적 비용이 발생하기도 한다. 이로 인해 각 앱의 개발자는 초기에는 보안 솔루션을 적용하지 않고, 향후 해당 앱이 수익을 발생시키거나 해당 앱에 대한 공격 또는 변형이 발견될 때 보안 솔루션의 적용을 고민한다. 이러한 현실적인 문제로 인해 앱의 버전에 따라 보안 솔루션의 도입 여부가 달라짐을 이용할 수 있다. 즉, 각종 보안 수단이 도입되기 전의 버전을 활용한다면 분석범위를 대폭 줄일 수 있으며, 다양한 apk archive[8]를 이용하여 공격자는 손쉽게 해당 앱의 각 버전을 획득할 수 있다.



〔그림 3〕 archive를 이용한 이전 버전 apk 획득

기본적으로 안티 롤백에 대한 요구 사항은 임베디드 디바이스 SW의 기본적인 보안 요구 사항이지만 안드로이드와 같은 공개 플랫폼은 개방된 앱 생태계를 가지고 있어 이러한 보안 요구 사항을 적용할 수 없고, 이로 인해 공격자에게 낮은 버전을 이용한 분석 기회를 제공하게 된다.

archive를 이용하여 각 버전을 확인하면 보안 수단이 적용되기 전의 코드를 확인할 수 있다. 즉, 난독화 및 안티 디버깅 기법이 적용되기 전 코드를 손쉽게 획득할 수 있고, 이는 공격자가 번거로운 분석을 할 필요가 없음을 의미한다. 또한 대부분의 앱은 업데이트 시 일부 코드를 변경 또는 추가하며 이는 전체 코드 대비 실제 변경된 코드의 양이 극히 적다. 따라서 공격자는 손쉽게 대부분의 코드를 원본 형태로 획득할 수 있다.

3.2. 디버깅 도구의 선택

디버깅 시 가장 중요한 것은 어떤 디버깅 도구를 사용할지 결정하는 것이다. 특히 디컴파일러의 경우 각 도구의 노하우, 알고리즘, 제공하는 기능에 따라 각기 다른 성능을 가지고, 동일한 코드도 해석하는 방법에 따라 다른 결과를 보여준다. 만약 분석 대상 apk에 난독화 및 암호화와 같은 안티 디버깅이 적용되어 있다면 도구에 따라 특정 코드는 해석을 하지 못하는 경우가 발생한다. 하지만 다른 디컴파일러를 사용한다면 다른 코드 해석으로 분석된 코드를 획득할 수 있다. [그림 4]는 동일한 코드를 서로 다른 디컴파일러가 해석한 결과를 보여준다. 좌측의 도구는 코드 해석에 실패하였고, 우측의 도구는 온전한 코드를 획득하였다.

또한 디버깅 도구의 버전에 따라 분석 결과가 달라질 수 있다. 모든 디버깅 도구는 유지보수 및 신규 기능 대응을 위해 지속적으로 업데이트를 진행하고, 이

```
r0 = r0.getCheckPay();  Catch:{ Exception -> 0x006b }     .getCheckPay().equals("1") && Int
r1 = "1";   Catch:{ Exception -> 0x006b }
r0 = r0.equals(r1);    Catch:{ Exception -> 0x006b }
if (r0 != 0) goto L_0x006b; Catch:{ Exception -> 0x006l
0x0049:
```

〔그림 4〕 디컴파일러의 성능 차이

```
r0 = r0.getCheckPay();  Catch:{ Exception -> 0x006b }     .getCheckPay().equals("1")
r1 = "1";   Catch:{ Exception -> 0x006b }
r0 = r0.equals(r1);    Catch:{ Exception -> 0x006b }
if (r0 != 0) goto L_0x006b; Catch:{ Exception -> 0x006l
L_0x0049:
```

〔그림 5〕 동일한 디컴파일러의 버전 별 결과 차이

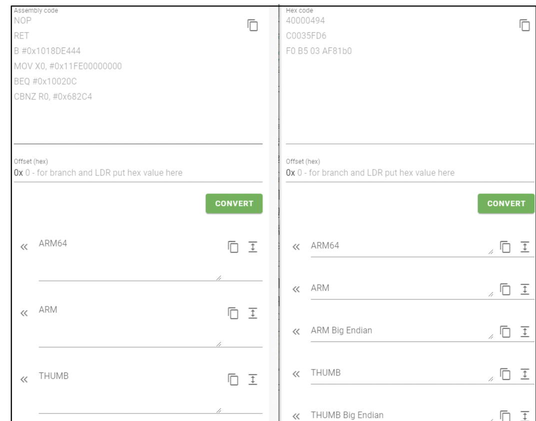
로 인해 기능에 변경이 발생하기도 한다. 특히 디컴파일러의 경우 버전에 따라 코드의 해석이 달라지기도 하며 이때 최신 버전이 항상 좋은 결과를 보여주지는 않는다. [그림 5]는 서로 다른 버전의 동일한 도구가 특정 코드에 대해 디컴파일에 성공 및 실패 한 것을 보여준다. 경우에 따라 낮은 버전의 디컴파일러가 코드 해석에 성공하거나 더 정확한 해석 결과를 보여주기도 하므로 다양한 버전의 도구를 활용하며 분석을 진행한다면 분석 정확도 및 효율성이 증가하므로 최신 버전의 도구만을 사용해서는 안된다.

3.3. so 변형 및 분석

안티 디버깅을 위해 자주 사용되는 기법 중 하나는 so를 통해 native API를 개발하고 이를 이용하여 안티 디버깅을 적용하는 것이다. native API는 c로 작성되어 디버깅이 자바에 비해 어렵고, 분석에 많은 시간이 소요된다. 또한 디컴파일 및 역공학을 진행하더라도 원본 코드가 아닌 도구가 자의적으로 해석한 코드 또는 어셈블리 형태의 코드만 획득할 수 있어 분석의 난이도가 올라간다. 특히 안티 디버깅 기법을 파악하고 이를 우회하려고 한다면 매번 메모리에 로드 된 코드를 변경해야 하는 번거로움이 있다. 이때 적용할 수 있는 방법은 so에서 원하는 부분의 코드를 변형하는 것이다. 특히 so에서 구현한 안티 디버깅 기법의 경우 특정 번지의 데이터를 확인하거나 해당 앱의 전자서명 값을 확인하는 것과 같이 특정 데이터의 변조 여부를 확인하는 경우가 많다. 코드의 분기만을 변조할 경우에도 디버깅으로 탐지하여 코드의 실행이 중단되거나 서버와의 통신이 중단될 수 있어 so를 변조하여 정상적인 동작의 형태로 동작 또는 데이터를 속이는 것이 효과적이다. so를 변조하기 위한 가장 효과적인 방법은 직접 어셈블리어를 작성하거나 변조하여 hex 에디터로 바이너리에 직접 입력하는 것이다. 특히 이 방법을 사용하면 간단한 코드 분기의 변조뿐만이 아니라 원하는 상수를 추가하고, 해당 상수의 메모리를 참조하거나 변형하는 것도 가능하여 안티 디버깅 기법에 대한 유연한 대처가 가능하다. 극단적으로 원본 so의 안티 디버깅 API가 무결성 검증을 위한 데이터 구조를 생성하고 입력할 때 원하는 형태로 데이터를 변조 또는 복제할 수 있어 client-server 구조의 안티 디버깅 및 무결성 검증 루틴을 원천적으로 무력화할 수 있다. [그림 6]은 Online ARM to HEX / HEX to ARM 에디터[9]이며, 이러한 도구를 이용하면 원하는 코드를 so 바이너리에 직접 작성할 수 있다.

또한 앱 분석의 효율성을 높이기 위해 분석 대상앱을 디컴파일하고 필요에 따라 코드를 변조한 후 앱을 재서명하여 설치하면 코드 단위의 디버깅이 가능하다. 이때 안드로이드는 패키지 이름을 기준으로 앱을 구분하고, 앱 설치 시 패키지 이름이 겹칠 경우 전자서명 값을 검증하여 동일한 앱 인지를 확인한다. 이때 so를 사용한다면 단순히 apk의 패키지 이름을 변경하는 식으로는 so와 심볼 충돌이 발생하여 실행이 되지 않으므로 so 심볼 변조를 함께 병행하여 분석을 진행하면 원본 앱과 변조 앱을 동시에 설치하여 분석 효율이 높아진다.

특히 so 파일의 경우 효과적인 분석을 위해 IDA[10]와 같은 도구를 이용하여 정적 분석을 수행할 수 있다. 하지만 정확한 분석을 위해서는 동적 분석을 진행해야 하며, 이때 원본 apk로부터 추출한 so 파일을 이용하여 동적 분석을 수행할 수 있다. 하지만 동적 분석을 막기 위해 주로 디버깅 도구의 attach를 막거나 apk의 디버깅 모드 확성화 여부를 확인하는 것과 같은



〔그림 6〕 온라인 ARM to HEX 에디터

안티 디버깅 기법을 적용한다. 뿐만 아니라 apk의 특정 offset을 읽거나 전자서명 값을 검증, 획득한 전자서명 값을 서버로 전송하여 정상적인 전자서명이 아니라면 필요한 데이터를 내려주지 않는 형태를 취하기도 한다. 이때 so 파일을 변조하여 정상적인 앱으로 위장한다면 간단히 이를 무력화 할 수 있으며, 이후 추가적인 분석을 진행하는 데에도 매번 이를 우회하지 않아도 되기 때문에 효율적으로 분석이 가능하다.

3.4. 안티 디버깅 우회

안티 디버깅 기법 및 이를 우회하기 위한 방법은 매우 다양하고, 대부분의 경우 어떤 안티 디버깅 기법이 적용되었는지 추측하고, 이에 대한 우회 방법을 시도하는 형태로 진행한다. 다만 최근 안티 디버깅 솔루션의 추세가 다양한 안티 디버깅 기법을 중첩으로 적용하는 것으로 하나씩 이를 우회하기 위해서는 많은 시간과 노력이 소요된다. 즉, 분석에 대한 비용을 증가시켜 분석을 포기하도록 유도한다.

특히 자바 코드 레벨의 안티 디버깅 뿐만 아니라 so를 이용한 안티 디버깅, 난독화, server-clietn 구조의 무결성 검증, 실행 압축 기법을 통한 so 및 dex 보호가 동시 다발적으로 적용되며, 그 외 다양한 형태의 안티 디버깅 기법이 추가로 적용된다. 이를 무력화하기 위해 모든 디버깅 기법을 하나씩 우회 하는 것은 시간적으로 낭비이며, 이러한 솔루션도 결국 앱 실행단계에서는 원본 so 및 dex를 동적으로 로드할 수 밖에 없다. 따라서 최종 실행 단계에서 메모리에 접근하여 이를 획득한다면 불필요한 안티 디버깅 기법 우회를 시도하지 않아도 된다. 특히 LiME[11]과 같은 커널 레벨 메모리 덤프 도구를 사용한다면 대부분의 앱이 무력화되어 원본 코드를 노출하게 된다. 커널 레벨 메모리 덤프를 수행하기 위해서는 기본적으로 루팅된 디바이스와 안드로이드 커널 재빌드 및 설치 과정이 필요하지만 커널 모듈이 직접 메모리 덤프를 수행하므로 안드로이드 전체 메모리에 대한 덤프가 가능하다.

또한 메모리 덤프를 통한 원본 코드의 획득과 앞에서 소개한 apk 디컴파일 및 재서명, so 코드 및 심볼 변조를 함께 시도한다면 자바 코드 레벨의 안티 디버깅과 so를 이용한 안티 디버깅을 모두 무력화할 수 있으며, 공격자가 원하는 형태로 앱의 변조 및 복제, 복원이 가능하다. 실제로 대부분의 앱이 이 방법을 통해 안티 디버깅 기법이 무력화되고 코드의 복제 및 변형을 통해 server-client 구조의 무결성 검증을 모두 우회하여 정상 사용자로 인식시키는 것이 가능함을 확인하였다. 뿐만 아니라 유료 안티 디버깅 솔루션이 적용된 앱을 복제, 변형한 후 원본 앱과 함께 설치하여 원본 앱과 복제 앱이 동시에 실행 가능함을 확인하였다.

Ⅳ. 결 론

현재 안드로이드는 자바의 구조적인 문제로 apk 분석 및 변조에 대한 보호에 어려움을 겪고 있다. 안드로이드는 자바를 선택함으로서 개발 편의성 및 생태계 확장을 극대화 하였지만 반대로 자바가 가진 코드 레벨에 대한 보안성이 낮음으로 인한 문제가 지속적으로 발생하고 있다. 개발자들은 개발 결과물을 보호하기 위해 다양한 안티 디버깅 기법을 개발하거나 솔루션을 구매하여 적용하고 있다. 하지만 대부분의 솔루션 또한 단순히 공격자를 귀찮게 하거나 번거롭게 하는 수준을 벗어나지 못하고 있으며, 강력한 방법을 적용할 수 록 성능의 하락 및 에러를 유발하여 사용성을 하락시키고 있다. 본 논문에서는 이러한 안티 디버깅 기법의 무분별한 적용은 실질적으로 앱을 보호하는데 큰 도움이 되지 못하므로 이를 보호하기 위한 근복적인 대책이 필요함을 보여준다. 특히 안티 디버깅이 적용된 분석 대상의 모든 앱은 메모리 덤프를 통해 원본 코드의 획득이 가능하였고, 메모리 덤프를 통해 중첩 적용된 안티 디버깅 기법이 한번에 우회 되는 것을 확인할 수 있었다. 뿐만 아니라 본 논문에서 소개한 so 변조를 활용할 경우 완전한 앱의 복제 및 변조가 가능하였으며, 적용된 보안 솔루션이 모두 무력화되어 이를 탐지하지 못하였다.

참 고 문 헌

[1] Christian Collberg, Clark David Thomborson, Douglas Low, “A Taxonomy of Obfuscating Transformations”, http://www.cs.auckland.ac.nz/staff-cgi-bin/mjd/csTRcgi.pl?serial, 1997

[2] https://ibotpeaches.github.io/Apktool/

[3] https://github.com/pxb1988/dex2jar/

[4] https://github.com/skylot/jadx/

[5] http://java-decompiler.github.io/

[6] https://www.pnfsoftware.com/
[7] http://www.javadecompilers.com/apk/
[8] https://www.apkmonk.com/
[9] https://armconverter.com/
[10] https://hex-rays.com/ida-pro/
[11] https://github.com/504ensicsLabs/LiME

〈저 자 소 개〉

박 민 수 (Minsu Park)
2013년 2월 : 고려대학교 정보보호대학원 석사
2018년 2월 : 고려대학교 정보보호대학원 박사
2018년 2월~현재 : LG전자 HE사업본부 선임연구원
<관심분야> 정보보호, 운영체제