

# 프로그램 변조를 통한 APK 타입 랜섬웨어 복호화 기법

최재민<sup>1</sup>, 최상훈<sup>2</sup>, 박기웅<sup>3†</sup>

<sup>1</sup>세종대학교 SysCore Lab. 석사과정

<sup>2</sup>세종대학교 SysCore Lab. 연구교수

<sup>3</sup>세종대학교 정보보호학과 교수

c.jaem7532@gmail.com, csh0052@gmail.com, woongbak@sejong.ac.kr

## Extraction Scheme for APK-type Ransomware via Software Tampering

Jae-Min Choi<sup>1</sup>, Sang-Hoon Choi<sup>2</sup>, Ki-Woong Park<sup>3†</sup>

<sup>1,2</sup> SysCore Lab., Sejong University

<sup>3</sup>Dept. of Computer and Information Security, Sejong University

### 요 약

랜섬웨어는 등장 이후 지속적으로 진화하여, 현재는 RaaS(Ransomware-as-a-Service) 형태로 배포되고 있으며, PC뿐만 아니라 모바일 기기까지 공격대상으로 삼고 있다. 이와 같이 고도화된 랜섬웨어에 대응하기 위해 탐지, 차단, 복구 기술 또한 발전해 왔다. 특히 복구와 관련해서는 메모리 덤프를 통해 복호화 키를 추출하거나, 취약한 암호 알고리즘을 이용해 키를 추출하는 연구가 활발히 진행되고 있다. 그러나 여전히 랜섬웨어 복호화는 어려운 과제로 남아 있다. 메모리 덤프나 공격자의 실수로 키를 추출할 수 있는 경우도 있지만, 기기 재부팅이나 강력한 암호화 알고리즘이 사용되면 복호화가 어렵다. 본 논문에서는 안드로이드 랜섬웨어를 대상으로 프로그램 변조를 통해 복호화하는 방법을 제안한다. 우리는 두 가지의 실험을 통해 안드로이드 랜섬웨어의 내부 로직을 수정하였고, 복호화가 가능함을 확인하였다.

### 1. 서론

랜섬웨어는 등장 이후 지속적으로 진화하며 점점 더 복잡한 형태로 발전하고 있다. 현재는 RaaS(Ransomware-as-a-Service) 형태로 서비스화되어 배포되고 있다. 현대의 랜섬웨어는 PC 환경뿐만 아니라 모바일 기기까지도 감염시키고 있다 [1]. 랜섬웨어의 고도화에 따라 이를 방어하기 위한 기술도 함께 발전하고 있으며, 다양한 탐지, 차단, 복구 방법에 관한 연구가 활발히 진행되고 있다. 그중 메모리 덤프를 활용하여 복호화 키를 추출하는 연구나, 공격자가 사용하는 취약한 암호 알고리즘을 이용한 키 추출 및 복호화 방법들이 제안되어 왔다 [4, 5]. 그러나 여전히 최신 랜섬웨어의 복호화에는 여전히 어려움이 존재하며, 일부 기업은 결국 비용을 지불하고 데이터를 복구하는 사례도 있다 [2]. 기존 연구에서는 난수를 기반으로 키를 생성하는 랜섬웨어에

대해 메모리 덤프를 통해 암호화 키를 찾아내는 방법이나, 공격자의 실수를 이용해 취약점을 공략하는 방법이 비교적 효과적이었다. 하지만 감염된 기기 재부팅되거나 공격자가 강력한 암호화 알고리즘을 사용할 경우, 복호화는 여전히 어렵다. 우리는 이와 같은 한계점을 해결하기 위해 모바일 기기 랜섬웨어에 대해 프로그램 변조를 통한 복호화 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 랜섬웨어 복호화에 필요한 배경지식 및 관련 연구에 관해 서술한다. 3장에서는 프로그램 변조 기법을 통한 안드로이드 랜섬웨어 복호화 방법에 관해 서술한다. 4장에서는 실험 결과에 관해 서술하고, 5장에서는 결론 및 향후 연구를 기술한다.

### 2. 배경지식 및 관련 연구

#### 2.1. 안드로이드 Dalvik과 Smali

안드로이드 운영체제는 Dalvik VM을 사용하여

†교신저자: 박기웅 (세종대학교 정보보호학과 교수)



```

>append(Ljava/lang/String;LJava/lang/StringBuffer;
    move-result-object v6
    move-object v7, v8
    lget-object v7, v7, Lcom/android/tencent/zdeve/bah/MainActivity;
    invoke-static {v7}, Ljava/lang/Integer;
    move-result v7
    const/4 v8, 0x4
    mul-int/lit8 v7, v7, 0x4
    const/4 v8, 0x3
    add-int/lit8 v7, v7, 0x3
    invoke-virtual {v6, v7}, Ljava/lang/StringBuffer;
    move-result-object v6
    invoke-virtual {v6, Ljava/lang/StringBuffer;

```

(그림 4-a) *xh* 연산 Smali 코드 및 변조 코드

**Origin Code**

```

.line 110
:cond_1
const-string v7, "Please do not quit the software, or the file may never be recovered!"
move-object v4, v7
goto :goto_0

```

**Code Tampering**

**Refactored Code**

```

.line 110
:cond_1
const-string v7, ""
move-object v7, v8
lget-object v7, v7, Lcom/android/tencent/zdeve/bah/MainActivity;
invoke-static {v7}, Ljava/lang/Integer;
move-result v8
mul-int/lit8 v8, v8, 0x4
add-int/lit8 v8, v8, 0x3
invoke-static {v8}, Ljava/lang/Integer;
move-result-object v7
move-object v4, v7
goto :goto_0

```

(그림 4-b) *str* 대입 Smali 코드 및 변조 코드

각각 Smali 코드는 기존의 코드에서 그림 4-a,b의 코드로 변경하였다. 이를 다시 리패키징한 뒤 확인하면, 그림 5-a,b 와 같이 Java 코드 또한 올바르게 변경된 것을 볼 수 있다.

**Origin Code**

```

const/4 v8, 0x4
mul-int/lit8 v7, v7, 0x4
const/4 v8, 0x3
add-int/lit8 v7, v7, 0x3

```

**Refactored Code**

```

const/4 v8, 0x1
mul-int/lit8 v7, v7, 0x1
const/4 v8, 0x0
add-int/lit8 v7, v7, 0x0

```

||

```

m = "" + (Integer.parseInt(this.xh) * 4 + 3);
m = "" + (Integer.parseInt(this.xh) * 1 + 0);

```

(그림 5-a) Smali 변조 코드의 Java 코드

**Origin Code**

```

String str;
if (getSupportFragmentManager().findFragmentById(2131099760) instanceof bah) {
    str = "";
} else {
    str = "Please do not quit the software, or the file may never be recovered!";
}
Toast.makeText((Context)this, str, 1).show();

```

**Refactored Code**

```

String str;
if (getSupportFragmentManager().findFragmentById(2131099760) instanceof bah) {
    str = "";
} else {
    str = Integer.toString(Integer.parseInt(this.xh) * 4 + 3);
}
Toast.makeText((Context)this, str, 1).show();

```

(그림 5-b) Smali 변조 코드의 Java 코드

## 4. 실험 결과

### 4.1 CASE 1: 연산 구문 변조

Smali 코드 변조를 통하여 MainActivity 클래스 .line 51에서 진행하는 계산식을 변조하였다. mul-int/lit8과 add-int/lit8은 Smali 코드에서 각각 곱셈과 덧셈을 의미하는 코드이고 3번째 매개변수의 Hex값을 수정하는 방식으로 진행하였다. 이후 해당 샘플을 apktool 을 이용하여 리패키징한 뒤 실행을 한 뒤, 그림 6 의 GUI에 표기되는 *xh*값과 동일한 값을 키 입력창에 작성하면 복호화가 올바르게 이뤄진다.



(그림 6) 연산 구문 변조를 통한 복호화 GUI

### 4.2 CASE 2: 코드 변조를 통한 복호화 키 출력

Smali 코드 변조를 통하여 MainActivity 클래스 .line 110에서 진행되는 Toast 메시지 *str* 변수에 문자열을 저장하는 코드를 변조하였다. 기존의 *v4* 레지스터에 “Please do not quit the software, or the file may never be recovered!”을 저장하던 Smali 코드에서 *xh*값을 *v7* 레지스터에 저장하고 해당 *v7* 레지스터에 parseInt 형변환 후 *v7*\*4+3 연산을 수행하였다. 이후, toString 형변환 하여 해당 값을 *v4* 레지스터에 저장하도록 변조하였다. 이후 해당 샘플을 CASE 1 과 동일하게 리패키징 한 뒤 실행을 하면 그림 7 과 같이 ‘뒤로가기’ 버튼을 터치하면 사용자에게 Toast 메시지를 이용하여 *xh*\*4+3 값을 출력한다. 이후 해당 값을 키 입력창에 작성하면 복호화가 올바르게 이뤄진다.



(그림 7) 코드 변조를 통한 복호화 GUI

### 4.3 WannaLocker 변조를 통한 복호화 평가

우리는 CASE 1과 CASE 2를 통하여 프로그램 변조를 통한 복호화가 가능함을 확인하였다. 해당 랜섬웨어는  $xh$  값을 임의의 난수로 받아온다. 하지만, 위처럼 프로그램 변조를 통해 프로그램 로직을 변경하여 난수로 키 값을 생성하는 랜섬웨어를 대상으로 달라지는 키 값에도 동일하게 복호화가 가능하다.

### 5. 결론 및 향후 연구

랜섬웨어는 RaaS 형태로 진화해 PC와 모바일 기기를 감염시키고 있으며, 이를 방어하기 위한 탐지, 차단, 복구 기술도 발전하고 있다. 메모리 덤프를 통한 복호화 키 추출이나 취약한 암호 알고리즘을 활용한 방법들이 제안되었으나, 강력한 암호화나 기기 재부팅 시 복호화는 여전히 어려운 과제로 남아 있다. 본 논문에서는 프로그램 변조 기법을 통한 안드로이드 랜섬웨어 복호화 방법을 제안한다. 우리는 실험을 통해 WannaLocker 패밀리에 대하여 복호화가 가능함을 증명하였다. 향후 연구에서는 EXE, ELF 등 PC 환경에서 실행되는 랜섬웨어에 대한 연구로 확장하여, 해당 방법이 안드로이드에 국한되는 것이 아님을 증명하고자 한다.

### Acknowledgement

본 논문은 과학기술정보통신부의 재원으로 정보통신기획평가원(IITP)의 국방ICT융합연구(Project No. 2022-11220701, 50%), 정보통신방송기술 국제공동연구(Project No. RS-2022-00165794, 30%), 대학 ICT연구센터사업(ITRC) 실감콘텐츠핵심기술개발(Project No. RS-2023-00228996, 20%)의 지원을 받아 수행된 연구임.

### 참고문헌

[1] Distribution of mobile malware worldwide in 2nd quarter 2023 and 1st quarter 2024, by type  
 [2] Share of infected organizations worldwide agreeing to pay ransom from 2021 to 2023  
 [3] Ehringer, David. "The dalvik virtual machine architecture." Techn. report (March 2010) 4.8 (2010): 72.  
 [4] Fernández-Fuentes, X., F. Pena, T., Cabaleiro, J.C., Recovering from Memory the Encryption Keys Used by Ransomware Targeting Windows

and Linux Systems, SAI : Intelligent Computing, London, United Kingdom, 2024, 1149-1166

[5] S. Kang, S. Lee, S. Kim, D. Kim, K. Kim, and J. Kim, "A Study on Decryption of Files Infected by Ragnar Locker Ransomware through Key Reuse Attack and Its Applications," Journal of the Korea Institute of Information Security & Cryptology, vol. 31, no. 2, pp. 221-231, 2021.  
 [6] MalwareBazaar : Malware dataset (n.d.), <https://bazaar.abuse.ch>  
 [7] dex2jar, <https://github.com/pxb1988/dex2jar>  
 [8] jda-gui, <https://java-decompiler.github.io>  
 [9] apktool, <https://apktool.org>