Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.5, Oct. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.5.845

AppLock 정보 은닉 앱에 대한 취약점 분석*

홍 표 길,[1†] 김 도 현[2‡]
[1,2]부산가톨릭대학교 (대학원생, 교수)

Vulnerability analysis for AppLock Application*

Pyo-gil Hong,[1†] Dohyun Kim[2‡]
[1,2]Catholic University of Pusan (Graduate student, Professor)

요 약

스마트폰의 메모리 용량이 증가하면서 스마트폰에 저장된 개인 정보의 종류와 양도 증가하고 있다. 하지만 최근 악의적인 공격자의 악성 앱이나 수리기사 등의 타인으로 인해 스마트폰의 사진, 동영상 등의 다양한 개인 정보가 유출될 가능성이 증가하고 있기 때문에, 사용자의 이러한 개인 정보를 보호할 수 있는 다양한 정보 은닉 앱이 출시되고 있다. 본 논문은 이러한 정보 은닉 앱의 암호 알고리즘 및 데이터 보호 기능을 분석하여 안전성 및 취약점을 분석 및 연구했다. 이를 위해 우리는 Google Play에 등록된 정보 은닉 앱 중에서 전 세계적으로 가장 많이 다운로드 된 AppLock 3.3.2 버전(December 30, 2020)과, 5.3.7 버전(June 13, 2022)을 분석했다. 접근 제어 기능의 경우, 사용자가 입력한 패턴을 암호화하기 위한 값들이 소스 코드에 평문으로 하드코딩 되어있으며 암호 알고리즘이 적용된 패턴 값은 xml 파일에 저장한다는 취약점이 존재했다. 또한 금고 기능의 경우 금고에 저장하기 위한 파일과 로그 파일을 암호화하지 않는 취약점이 존재했다.

ABSTRACT

As the memory capacity of smartphone increases, the type and amount of privacy stored in the smartphone is also increasing. but recently there is an increasing possibility that various personal information such as photos and videos of smartphones may be leaked due to malicious apps by malicious attackers or other people such as repair technicians. This paper analyzed and studied the security and vulnerability of these vault apps by analyzing the cryptography algorithm and data protection function. We analyzed 5.3.7(June 13, 2022) and 3.3.2(December 30, 2020) versions of AppLock, the most downloaded information-hidding apps registered with Google Play, and found various vulnerabilities. In the case of access control, there was a vulnerability in that values for encrypting patterns entered by users were hardcoded into plain text in the source code, and encrypted pattern values were stored in xml files. In addition, in the case of the vault function, there was a vulnerability in that the files and log files for storing in the vault were not encrypted.

Keywords: AppLock, Vault Application, information hiding, vulnerability analysis

I. 서 론

스마트폰의 사용자와 기기 용량이 증가함에 따라 스마트폰 내부에 저장되는 개인 정보의 종류와 크기는 증가했다. 그러나 악의적인 공격자의 해킹, 랜섬웨어 등의 공격과 수리기사 등의 타인의 비인가 접근으로 인해 개인 정보가 유출될 가능성이 존재한다. 이러한 상황으로부터 스마트폰 내부의 개인 정보의

Received(07. 08. 2022), Modified(08. 17. 2022), Accepted(08. 18. 2022)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2021R1F1A1061926).
* 본 논문은 2021년도 한국정보보호학회 동계학술대회에 발표한 우수 논문을 개선 및 확장한 것임.
† 주저자, kinvyrlf1744@gmail.com
‡ 교신저자, dohyun@cup.ac.kr(Corresponding author)

유출 가능성을 줄이기 위한 정보 은닉 앱이 등장했다. 그러나 정보 은닉 앱 개발 과정에서 암호 알고리즘 오용 및 잘못된 데이터 보호 기능 구현으로 인해 암호화 및 은닉된 사용자의 개인 정보가 복호화될 가능성이 있다.

본 논문에서는 Google Play에서 전 세계적으로 가장 많은 1억 회 이상 다운로드 된 정보 은닉 앱인 AppLock(ver. 3.3.2, 5.3.7)의 접근 제어 기능과 주요 기능인 금고 기능을 정적 및 동적 분석을 통해 앱의 메커니즘 및 알고리즘을 분석하여 안전성 및 취약점을 연구했다. 그 결과 패턴락 접근 제어의 비밀번호 복호화, 은닉된 파일의 정보 및 실제 데이터 추출이 가능한 것을 확인했다.

2장에서는 정보 은닉 앱을 대상으로 연구한 관련 연구들을 소개한다. 3장에서는 AppLock의 접근 제어 기능을 분석한 내용을 설명하고 4장에서는 AppLock의 금고 기능을 분석한 내용을 설명한다. 5장에서는 3장과 4장에서 다루었던 내용을 바탕으로 패턴락 접근 제어 비밀번호 복호화와 은닉된 파일의 정보 및 실제 데이터 추출과 관련하여 설명한다. 6장에서는 본 논문에서 다루었던 이 연구의 결론에 대해 논의한다.

II. 관련 연구

다양한 정보 은닉 앱을 대상으로 안전성 및 취약점을 분석한 연구가 있었다. Xiaolu Zhang 외 2명은 Google Play에서 18개의 Vault Application 앱을 역공학하고 포렌식 아티팩트를 조사하여 분석한 사례 연구와 결과를 제시했다[1]. Peter Sabev 외 1명은 2개의 안드로이드 Vault Application 앱에 대해 앱이 실행 중일 때 메인 메모리에 로드된 데이터를 검사하여 암호화 여부를 조사했고 대부분 데이터가 평문으로 로드되는 것을 확인하였다[2]. Michaila Duncan 외 1명은 총 6개의 루팅된 안드로이드 기기에서 Vault Application 앱을 탐지하고 은닉되어 숨겨진 파일들을 추출하는 연구를 수행했다[3]. Tahira Rasul 외 2명은 Google Play 스토어에서 사용자의 데이터를 은닉하는 18개의 Vault Application의 작동 방식, 내부 코드를 분석하여 데이터가 은닉 기법과 데이터 복구 가능성을 확인, 분석 방안을 제시했다. 분석한 결과 대부분의 Vault Application은 사용자의 데이터 은닉을 지원하지만, 프로그래밍적으로 취약점이 존재하였으며 데이터와 로그인 자격을 쉽게 획득할 수 있었다는 것을 발견했다[4].

Nannan Xie 외 2명은 SVM(Support Vector Machine) 기반 탐지 기법을 사용하여 안드로이드 Vault Application 앱의 설치 여부를 탐지하는 연구를 하여 93.33% 정확도를 달성했고 이것을 자동화 도구로 개발했다[5]. Dae-gyu Kim 외 1명은 은닉형 Vault Application 앱과 비 은닉형 Vault Application 앱 각각 15개를 대상으로 앱에 포함된 strings.xml 파일을 텍스트마이닝 기법을 활용하여 XML 구문을 비교 분석했다. 분석한 결과 은닉형 Vault Application 앱에서는 은닉 관련 단어를 높은 빈도로 발견할 수 있었으며 공학 기술적인 관점에서 분석한 것이 아닌 인문 사회학적인 관점으로 접근하여 분석했다는 의의가 있다[6]. Mingming Peng 외 5명은 Google Play에서 Android Vault Application 앱들을 머신 러닝과 딥 러닝 모델들을 이용하여 완전히 자동으로 탐지하는 프레임워크인 DECADE를 제안했다. 설명 및 응용 프로그램 제목에서 문자열 데이터를 추출하여 텍스트 기반 분류 모델링을 통해 실험한 결과 SVM(Support Vector Machine)이 BV, Doc2vec, FastText, Sentence Bert 4가지 중에서 Sentence Bert를 제외하고 정확도가 가장 높았다. 또한 텍스트 기반, 이미지 기반 및 텍스트 + 이미지 기반 모델 중에서 가장 정확도가 높은 것은 텍스트 기반 모델이 가장 정확도가 높은 것을 발견했다[7].

특히 Xiaolu Zhang 외 2명이 연구한 AppLock(ver. 2.16.3)을 제외하고 연구된 바가 없었다. 또한 본 논문에서 소개한 AppLock(ver. 3.3.2, 5.3.7)과 취약점이 거의 같지만 접근 제어 기능의 경우 암호키 관리 및 암호키 값에서 차이가 존재하며 금고 기능의 경우 금고에 저장된 파일 경로와 로그 파일의 위치 및 이름, 내부 구조에서 차이가 존재한다[1].

III. AppLock의 접근 제어 기능 분석

본 논문에서는 정보 은닉 앱의 안전성 분석을 위해 Google Play 기준 1억 회 이상 다운로드 된 AppLock의 2가지 버전을 분석했다. 정적 분석을 위해 JADX-GUI와 JEB-Decompiler를 사용했고, 동적 분석을 위해 Frida의 함수 후킹 기능을 통해 2가지 버전의 차이점을 확인했다. 동적 분석을 위한 환

Table 1. Dynamic analysis environment

Name	Description
Device Name	Galaxy Note5
Model Number	SM-N920K
Android Version	7.0
Root Option	Rooted

경은 〔표 1〕과 같다.

3.1 AppLock(ver. 3.3.2)

AppLock은 처음 실행하면 접근 제어를 위해 패턴을 설정한다. 사용자가 입력한 패턴은 이 앱의 접근 제어 해제에 필요한 패턴이며 AppLock은 이 패턴을 암호화하여 저장한다. 이를 위해 Java의 Cipher 라이브러리의 AES-256/CBC/PKCS7Padding을 사용하고 해시 알고리즘은 MD5, SHA1, SHA256을 사용한다. 암호키는 랜덤한 문자열을 생성하여 사용하고 IV(Initialization Vector)는 하드코딩 된 '20110711 20170711' 문자열을 바이트 배열로 변환하여 사용한다. 전체적인 알고리즘은 〔수식 1〕과 같다.

$$\begin{aligned} P &= Base64(SHA1(Arr(Pattern))) \\ Key &= Random(32) \\ IV &= 2011071120170711 \\ image\_lock\_pattern &= \\ Base64(&AES\_CBC(P,\ Key,\ IV)) \end{aligned} \quad (1)$$

● 접근 제어 패턴값

사용자가 입력한 패턴은 〔그림 1〕과 같은 총 9개의 점을 통해 (x, y) 좌표로 관리되고 이것은 패턴 입력 순서(변수 i)에 따라 (x[i] * 3) + y[i] 식을 통해 바이트 배열로 변환된다. 이 바이트 배열에 SHA1 해시 알고리즘과 Base64 Encoding을 차례로 적용한다. 이것이 암호화 대상으로 〔수식 1〕의 'Base64(SHA1(Arr(Pattern)))' 값이다.

만약 임의의 사용자가 〔그림 2〕와 같이 'ㄱ'자 모양으로 패턴을 설정할 경우, 〔그림 3〕과 같이 변환한다.

● 암호키 랜덤 문자열

암호키 생성을 위해 알파벳 대소문자 52개(A-Z, a-z), 숫자 10개(0-9), 특수 기호 15개(~!@#$%^&*_-+=|?)들을 사용하여 랜덤한 32 Bytes 길이

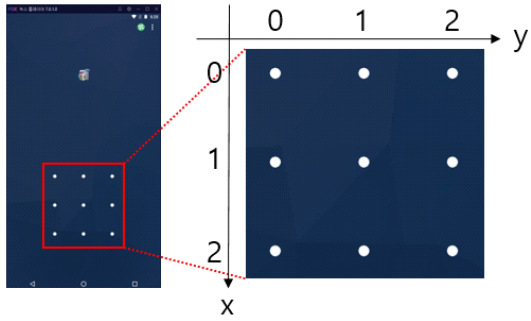

Fig. 1. Pattern Values

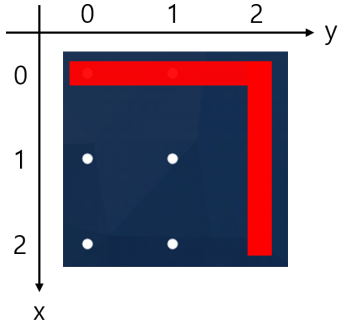

Fig. 2. Pattern Example

```
i -> (x, y)        bArr[i] = (x[i] * 3) + y[i]
0 -> (0, 0)        bArr[0] -> 0 = 0 * 3 + 0
1 -> (0, 1)        bArr[1] -> 1 = 0 * 3 + 1
2 -> (0, 2)        bArr[2] -> 2 = 0 * 3 + 2
3 -> (1, 2)        bArr[3] -> 5 = 1 * 3 + 2
4 -> (2, 2)        bArr[4] -> 8 = 2 * 3 + 2

SHA1 Apply -> ffec1e70d113b0b96c7e6cb2b33460e24407a96e
Base64 Apply -> /+wecNETsLlsfmyyszRg4kQHqW4=
```

Fig. 3. Base64(SHA1(Arr(Pattern))) Example

의 문자열을 생성하여 암호키로 사용한다. 이것이 〔수식 1〕의 'Random(32)' 값이다.

AppLock은 이 암호키를 특정 파일에 저장하여 관리하는데, 이를 위해 처음 사용자 패턴을 입력받은 후에 사용자 uuid를 생성하고 MD5 해시를 적용하여 '/data/data/com.domobile.AppLockwatcher/files/〔uuid의 MD5〕 파일을 생성한다. 이 파일 안에 암호키인 랜덤한 32 Bytes 길이의 문자열 값에 〔수식 2〕의 암호 알고리즘을 적용하여 저장한다.

이 과정에서 암호키 랜덤 문자열을 암호화하기 위한 암호키인 'domobile' 문자열과 IV 값인 '20110711120170711' 문자열은 소스 코드에 하드코딩 되어

있다. 이 파일 경로를 위해 생성된 uuid는 '/data/data/com.domobile.AppLockwatcher/shared_pref/com.domobile.AppLockwatcher_preferences.xml' 파일의 'pk_AppLock_uuid' 키를 통해 저장된다.

$$\begin{aligned} &P = Random(32)\\ &Key = domobile\\ &IV = 2011071120170711\\ &Encryption\ Key = \\ &Base64(AES\_CBC(P,\ Key,\ IV)) \end{aligned} \quad (2)$$

● 암호화된 접근 제어 패턴값 관리

AppLock은 〔수식 1〕로 생성된 암호화된 접근 제어 패턴값을 '/data/data/com.domobileAppLockwatcher/shared_pref/com.domobile.AppLockwatcher_preferences.xml' 파일 안에 'image_lock_pattern' 키를 통해 저장한다.

3.2 AppLock(ver. 5.3.7)

AppLock에서는 AppLock(ver. 3.3.2)과 암호화 과정이 비슷하지만, 암호키 생성에 대해 차이점이 존재한다.

● 암호키 생성

AppLock에서는 getFilesDir() 함수를 통해 '/data/data/com.domobile.AppLockwatcher/' 경로에 files 폴더가 있는지 확인한다. 만약 존재하지 않는다면 'domobile2011' 문자열을 반환한다. 그러나 해당 경로에 files 폴더가 존재한다면 '/data/data/com.domobile.AppLockwatcher/files/〔uuid(MD5)〕' 경로에 SSID(MD5) 파일이 있는지 검사한다. 만약 존재할 경우 파일 안의 내용(file_content)을 읽어 복호화를 진행한다. 복호화키는 'domobile'이며 파일의 내용을 복호화한 값을 반환한다. 복호화 수식은 〔수식 3〕과 같다.

$$\begin{aligned} &P = file\_content\\ &Key = domobile\\ &IV = 2011071120170711\\ &Decryption\ Key = \\ &Base64(AES\_CBC(P,\ Key,\ IV)) \end{aligned} \quad (3)$$

AppLock의 암호 알고리즘의 수식은 〔수식 4〕와 같으며 사용자가 입력한 패턴값에 해당 암호 알고리즘을 적용하여 '/data/data/com.domobile.AppLockwatcher/shared_pref/com.domobile.AppLockwatcher_preferences.xml' 파일 안의 'image_lock_pattern' 키에 암호화된 패턴값을 저장한다.

$$\begin{aligned} &P = Base64(SHA1(Arr(Pattern)))\\ &Key = domobile2011\\ &IV = 2011071120170711\\ &image\_lock\_pattern = \\ &Base64(AES\_CBC(P,\ Key,\ IV)) \end{aligned} \quad (4)$$

IV. AppLock의 금고 기능 분석

AppLock의 주요 기능인 금고 기능은 파일 형식에 따라 5가지 카테고리(이미지, 비디오, 오디오, 파일, APK)로 나뉜다. 금고에 파일을 저장할 때는 원본 파일을 사용자가 인식하기 힘든 경로를 생성하여 확장자 제거 및 파일 이름을 변경하여 이동시킨다. 이렇게 금고에 저장한 파일 내역은 AppLock이 관리하는 특정 경로의 로그에 저장된다.

4.1 AppLock(ver. 3.3.2)

● 금고에 파일 저장

특정 파일을 금고에 저장하면 AppLock은 그 파일을 이동시킬 임의의 경로를 생성한다. 그 후 원본 파일을 생성된 경로로 암호화하지 않고 단순 이동시킨다. 이를 통해 일반 사용자는 알기 어려운 특별한 경로로 파일을 보관한다.

금고의 기본 경로는 '/storage/emulated/0/.dom0o7b1i1le/dont_remove/'로 이 경로는 소스 코드에 하드코딩되어 있다. 이 하위 경로에 0부터 99까지 순차적으로 MD5 해시 알고리즘을 적용한 값으로 총 100개의 디렉터리를 만들고 그 하위에 금고에 저장할 파일의 포맷에 따라 디렉터리를 생성한다(〔.image|.video|.file|.thumb〕). 그 후 특정 파일을 금고로 이동할 때는 총 100개의 디렉터리 중에 랜덤으로 선택한다. 그리고 하위 경로에 이동하는 파일 형식에 따라 4개의 폴더〔.image|.video|.file|.thumb〕 중에 1개의 폴더에 금고로 이동할 파일의 이름을 금고 이동 시점의 UnixTime 값으로 변경하여 확장자를 제거하고 Java의 renameTo() 메소드를 통해 단순 이동시킨다. 〔그림 4〕는 금고에 저장된 파일들의 경로를 도식화한 것이다.

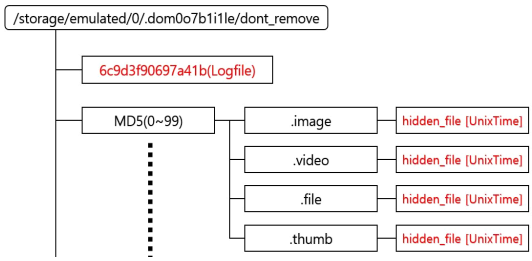

Fig. 4. Vault Structure

● 금고 기록 로깅

AppLock은 금고로 원본 파일을 이동시킨 후 SQLite Database 파일 포맷의 특정한 로그 파일을 통해 그 기록을 저장한다. 이 로그 파일의 경로는 '/data/data/com.domobile.AppLockwatcher/files/Media/6c9d3f90697a41b'다. 이 로그 파일은 외부 저장소('/storage' 파티션) 영역인 '/storage/emulated/0/.dom0o7b1i1le/dont_remove/6c9d3f90697a41b'로 복사된다. 〔그림 5〕은 파일의 스키마 정보, 〔표 2〕은 금고 기록을 로깅하는 '6c9d3f90697a41b' 의 주요 컬럼 정보이다.

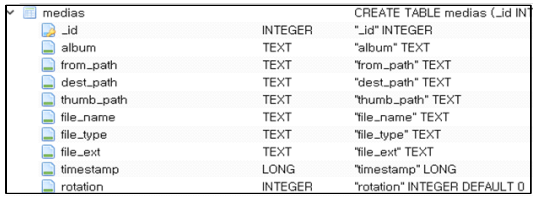

Fig. 5. 6c9d3f90697a41b schema information

Table 2. Important columns of 6c9d3f90697a41b

Name	Description
from_path	The full path of original file
dest_path	The full path of hidden file
thumb_path	The full path of thumb file
file_name	The name of hidden file
file_type	The type of original file
file_ext	The extension of original file
timestamp	The Last time put in Vault

4.2 AppLock(ver. 5.3.7)

● 금고에 파일 저장

AppLock은 특정 파일을 금고에 저장할 때 임의의 경로를 생성한다. 그리고 원본 파일의 이름과 내용을 암호화한 후에 생성된 경로에 저장한다.

외부 저장소(/storage 파티션) 영역의 금고 기본 경로는 '/storage/emulated/0/.do0mo7bi1le1/medias'이며 저장할 파일의 포맷과 관계없이 이 경로에 전부 저장한다. 이 하위에 생성되는 파일의 이름은 자바의 라이브러리인 'java.util.UUID'의 내부 함수인 randomUUID()의 반환 값이다. 〔그림 6〕는 외부 저장소(/storage 파티션) 영역의 금고 구조이다.

내부 저장소(/data 파티션) 영역의 금고에 저장하는 파일은 '/data/data/com.domobile.AppLockwatcher/files'의 하위 경로에 저장할 파일의 포맷에 따라 디렉터리(Photos|Videos|Audios|Files|Apks)를 생성하고 이 하위 경로에 금고에 저장했던 파일의 원본 파일을 저장한다. 저장할 때 파일의 이름은 '/storage' 파티션 영역의 금고에 저장한 파일의 이름과 동일하다. 〔그림 7〕는 내부 저장소 영역(/data 파티션)의 금고 구조이다.

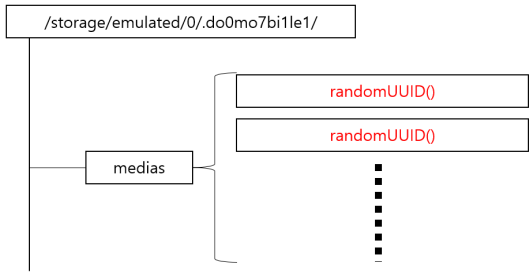

Fig. 6. Vault Structure in '/storage' Partition

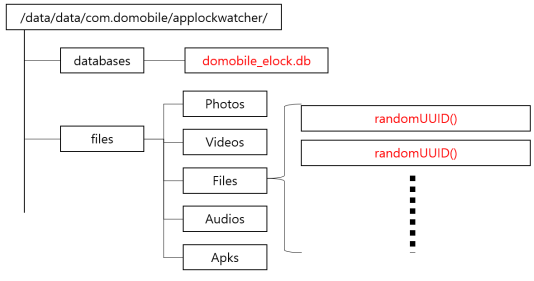

Fig. 7. Vault Structure in '/data' Partition

● 금고 기록 로깅

AppLock은 금고로 이동시킨 파일의 메타데이터 및 금고 입출력 로그를 SQLite3 DataBase 파일 포맷으로 관리한다. 해당 로그 파일의 경로는 '/data/data/com.domobile.AppLockwatcher/data

SMediaTable		CREATE TABLE SMedia
_id	INTEGER	"_id" INTEGER
driveId	TEXT	"driveId" TEXT
driveSize	INTEGER	"driveSize" INTEGER
uid	TEXT	"uid" TEXT
albumId	TEXT	"albumId" TEXT
albumName	TEXT	"albumName" TEXT
mimeType	TEXT	"mimeType" TEXT
name	TEXT	"name" TEXT
filePath	TEXT	"filePath" TEXT
fileSize	INTEGER	"fileSize" INTEGER
orientation	INTEGER	"orientation" INTEGER
width	INTEGER	"width" INTEGER
height	INTEGER	"height" INTEGER
duration	INTEGER	"duration" INTEGER
srcPath	TEXT	"srcPath" TEXT
srcMd5	TEXT	"srcMd5" TEXT
dateToken	INTEGER	"dateToken" INTEGER
lastTime	INTEGER	"lastTime" INTEGER
sortId	TEXT	"sortId" TEXT
delState	INTEGER	"delState" INTEGER
synState	INTEGER	"synState" INTEGER
bkpState	INTEGER	"bkpState" INTEGER
keepDevice	INTEGER	"keepDevice" INTEGER
fitState	INTEGER	"fitState" INTEGER

Fig. 8. SMeidaTable schema information in domobile_elock.db

Table 3. Important columns of 'SMeidaTable'

Name	Description
uid	The name of hidden file
mimeType	The extension of original file
name	The name of original file
fileSize	The size of file
srcPath	The location of original file
srcMd5	The MD5 hash value of original file
lastTime	The Last time put in Vault

bases/domobile_elock.db'이며 로그 데이터는 'SMediaTable' 테이블에 존재한다. 〔표 3〕는 'SMediaTable'에 저장된 은닉 파일들을 추출할 때 필요한 주요 컬럼들의 정보이다. 〔그림 8〕은 'domobile_elock.db'의 'SMediaTable'의 스키마 정보이다.

V. AppLock 앱의 취약점 분석

본 절에서는 AppLock의 암호 알고리즘 및 데이터 보호 기능에 대한 취약점을 분석한다. 분석 대상은 접근 제어를 위해 설정한 패턴 값 복호화에 대한 가능성과 금고에 저장된 은닉 파일들의 복구 가능성에 대한 것이다. 취약점 분석에 사용한 테스트 환경은 〔표 1〕과 같다.

5.1 AppLock(ver. 3.3.2)

5.1.1 접근 제어 패턴값 복호화

AppLock의 암호화된 접근 제어 패턴값을 복호화하기 위해서는 암호화에 사용된 암호 알고리즘과 암호화된 접근 제어 패턴값, 암호키로 사용된 랜덤 문자열, IV 값들이 필요하다.

● 암호키 복호화

위 분석을 통해 암호화된 암호키 랜덤 문자열은 특정 파일에 저장됨을 알아냈다. 따라서 이 파일을 찾아내기 위해 '/data/data/com.domobile.AppLockwatcher/shared_prefs/com.domobile.AppLockwatcher_preferences.xml' 파일의 'pk_AppLock_uuid' 값을 추출하고 이것의 MD5 해시값을 구한다. 그 결과 암호화된 암호키 랜덤 문자열을 '/data/data/com.domobile.AppLockwatcher/files/〔uuid의 MD5〕' 파일에서 추출할 수 있다. 이 값과 복호화 키('domobile'), IV('2011071120170711') 값을 이용하여 AES 복호화를 하면, AppLock에 설정된 접근 제어 패턴값을 암호화하기 위해 사용된 암호키를 추출할 수 있다.

● 접근 제어 패턴값 복호화

암호화된 접근 제어 패턴값은 '/data/data/com.domobile.AppLockwatcher/shared_prefs/com.domobile.AppLockwatcher_preferences.xml'의 'image_lock_pattern' 키를 통해 획득할 수 있다. 이 값과 앞에서 구한 암호키 랜덤 문자열, IV 값('2011071120170711')을 이용해 암호화된 접근 패턴값을 복호화할 수 있다.

그러나 접근 패턴값에는 Base64와 SHA1이 적용되어 있기 때문에 SHA1(Arr(Pattern)) 값에 대한 해시테이블 공격을 통해 사용자가 실제 입력한 패턴 값의 좌표를 구할 수 있다. AppLock은 패턴의 수를 최소 4개에서 9개까지 설정할 수 있는데, 이에 대한 모든 해시테이블을 만들어서 실제 패턴값을 획득할 수 있다.

5.1.2 금고의 은닉 파일 복구

AppLock의 금고 기능으로 저장된 파일들은 앞에서 살펴봤듯이 소스 코드 역공학을 통해 하드코딩된 경로인 '/storage/emulated/0/.dom0o7b1i1le/dont_

remove/' 하위에 존재하고, 금고 기록에 대한 로그는 'data/data/com.domobile.AppLockwatcher/files/Media/6c9d3f90697a41b', '/storage/emulated/0/.dom0o7b1i1le/dont_remove/6c9d3f90697a41b' 두 개의 동일한 파일에 존재함을 알 수 있다.

이 파일은 〔그림 5〕와 같이 다양한 원본 파일과 금고로 이동된 파일의 정보 등을 포함하고 있는데, 이러한 파일을 사용자 권한으로 접근 가능한 SDcard 영역에 관리하는 것은 큰 취약점이다. 따라서 이 파일을 통해 금고에 저장된 파일들의 원본 파일을 획득할 수 있다.

하지만, AppLock은 금고에서 파일을 꺼내면 로그 파일에서 해당 파일에 대한 기록은 지우기 때문에 이전에 금고에 저장했던 파일의 기록을 분석하기 위해서는 'data/data/com.domobile.AppLockwatcher/files/Media/6c9d3f90697a41b' 파일의 미할당 페이지, 저널 파일들을 분석해야 한다.

5.2 AppLock(ver. 5.3.7)

5.2.1 접근 제어 패턴값 복호화

AppLock에서 암호화된 접근 제어 패턴값을 복호화하기 위한 방안은 3.3.2 버전의 방식과 동일하다. 그러나 암호키 복호화 방식에 차이가 존재한다.

● 암호키 복호화

위 분석을 통해 특정 파일에 암호키를 암호화하여 저장하는 경우와 저장하지 않는 경우가 존재한다. 만약 암호화된 암호키가 특정 파일에 존재하는 경우에는 복호화가 필요하다.

암호화된 암호키가 저장된 특정 파일에 존재할 경우 '/data/data/com.domobile.AppLockwatcher/shared_prefs/com.domobile.AppLockwatcher_preferences.xml' 파일의 'pk_AppLock_uuid' 값을 추출하고 이것의 MD5 해시값을 구하여 암호키가 저장된 특정 파일을 찾고 기존의 암호키 복호화 방법을 통해 복호화가 가능하다. 암호화된 암호키를 저장한 특정 파일이 존재하지 않을 때 암호키 값은 'domobile2011'이다.

● 접근 제어 패턴값 복호화

암호화된 접근 제어 패턴값은 '/data/data/com.domobile.AppLockwatcher/shared_prefs/com.domobile.AppLockwatcher_preferences.xml'의 'image_lock_pattern' 키를 통해 획득할 수 있다. 이 값과 앞에서 구한 암호키(랜덤 32 Bytes 문자열|'domobile2011'), IV 값('2011071120170711')을 이용해 암호화된 접근 패턴값을 복호화할 수 있다.

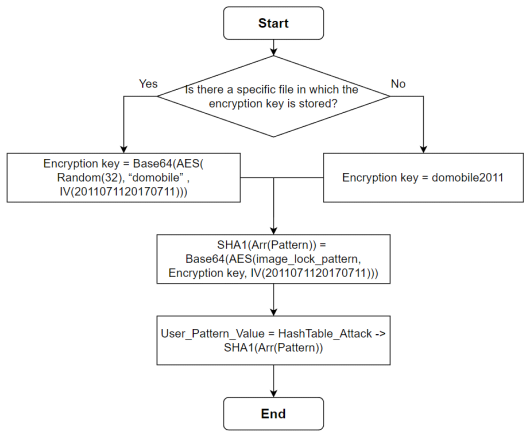

Fig. 9. PatternLock Decryption Algorithm

그리고 기존의 방안과 동일한 방식으로 SHA1(Arr(Pattern)) 값에 대한 해시테이블 공격을 통해 사용자가 입력한 패턴 값의 좌표를 구할 수 있다. 〔그림 9〕은 AppLock의 암호화된 패턴락 복호화 알고리즘이다.

5.2.2 금고의 은닉 파일 복구

AppLock의 금고 기능으로 저장된 파일들은 앞에서 살펴봤듯이 소스 코드 역공학을 통해 하드코딩된 경로인 '/storage/emulated/0/.do0mo7bi1le1/medias' 하위에 존재하고, 금고 기록에 대한 로그는 '/data/data/com.domobile.AppLockwatcher/databases/domobile_elock.db' 파일의 'SMeidaTable' 테이블에 존재한다. 그러나 금고에 저장한 파일의 원본을 '/data/data/com.domobile.AppLockwatcher/files' 경로에 저장하기 때문에 복호화할 필요 없이 원본 파일을 획득할 수 있다는 취약점이 존재한다.

하지만, AppLock은 금고에서 파일을 꺼내면 로그 파일에서 해당 파일에 대한 기록은 지우기 때문에 이전에 금고에 저장했던 파일의 기록을 분석하기 위해서는 '/data/data/com.domobile.AppLockwatcher/databases/domobile_elock.db-journal' 파일의 미할당 페이지, 저널 파일들을 분석해야 한다.

Ⅵ. 결 론

본 논문에서는 Google Play 기준 1억 회 이상 다운로드 된 AppLock을 정적 및 동적 분석하여 이 앱의 주요 기능인 접근 제어 기능과 금고 기능의 안전성 및 취약점을 분석했다.

접근 제어 기능 분석 결과 암복호화에 관련된 데이터들은 특정 xml 파일에 기록되어 있으며 IV 및 암호키 값의 경우 평문으로 표기돼 있어 악의적인 공격자에 의해 쉽게 무력화될 가능성이 존재한다. 또한 AppLock(ver. 3.3.2)에서는 암호키를 특정 파일에 저장하여 관리하는 반면에 AppLock(ver. 5.3.7)에서는 암호키가 하드코딩 돼 있어서 오히려 취약점이 개선되지 않은 것을 발견했다.

금고 기능을 분석한 결과 AppLock(ver. 3.3.2)에서는 원본 파일을 확장자를 제거한 후 파일의 이름을 변경하여 사용자가 인식하기 힘든 경로로 단순 이동시킨다. 하지만 AppLock(ver. 5.3.7)에서는 원본 파일의 암호화를 진행한 후 확장자를 제거하고 파일 이름을 변경하여 금고로 이동시킨다. 그러나 AppLock(ver. 5.3.7) 또한 금고에 저장된 은닉 파일의 원본을 '/data' 파티션에 그대로 보관하기 때문에 취약점이 존재하는 것을 발견했다.

우리는 분석한 결과를 통해 AppLock의 취약점 보안 방안을 제안한다. 접근 제어 기능의 경우 소스코드에 하드코딩된 암호키 값 및 IV 값을 분석가가 찾기 어렵게 난독화 하거나 DexGuard와 같은 상용 난독화 도구를 이용해야 한다[8]. 금고 기능의 경우 금고에 저장하는 파일을 암호화해야 하며 금고 입출력 로그가 저장된 DB 파일을 SQLCipher와 같은 암호화 도구를 활용하여 암호화해야 한다[9]. 또는 LIAPP과 같은 상용 보안 앱 솔루션을 사용하여 앱 자체를 보호해야 한다[10]. 우리는 향후 금고 기능이 포함된 DropBox와 같은 다른 정보 은닉 앱들의 취약점도 분석할 예정이다.

References

[1] Zhang, Xiaolu, Ibrahim Baggili, and Frank Breitinger. "Breaking into the vault: Privacy, security and forensic analysis of Android vault applications." Computers & Security, vol. 70, pp. 516-531, Sep. 2017.

[2] Petrov, Peter Sabevand Milen. "Android Password Managers and Vault Applications: An Investigation on Data Remanence in Main Memory.", 2021.

[3] Duncan, Michaila, and Umit Karabiyik. "Detection and recovery of anti-forensic (vault) applications on android devices.",Annual ADFSL Conference on Digital Forensics, Security and Law. no. 6, May. 2018.

[4] Tahira Rasul, Rabia Latif, Nor Shahida Mohd Jamail. "A computational forensic framework for detection of hidden applications on Android." Indonesian Journal of Electrical Engineering and Computer Science, vol. 20, no. 1, pp. 353-360, Apr. 2020.

[5] Xie, Nannan, et al. "Android Vault Application Behavior Analysis and Detection.", ICPCSEE 2020:Data Science, vol. 1257, pp. 428-439, Aug. 2020.

[6] Dae-gyu Kim, Chang-soo Kim. "A Study on the Feature Point Extraction Methodology Based on XML for Searching Hidden Vault Anti-Forensics Apps." Journal of Internet Computing and Services, vol. 23, no. 2, pp. 61 - 70, Jan. 2022.

[7] PENG, Mingming, et al. "DECADE-Deep Learning Based Content-hiding Application Detection System for Android." 2021 IEEE International Conference on Big Data (Big Data), pp. 5430-5440, Dec. 2021.

[8] GUARDSQUARE, "DexGuard", https://www.guardsquare.com/dexguard, Sep. 2022.

[9] Zetetic, "SQLCipher", https://www.zetetic.net/sqlcipher/, Sep. 2022.

[10] LOCKIN, "liapp", https://liapp.lockincomp.com/ko/, Sep. 2022.

〈저자소개〉

홍 표 길 (Pyo-gil Hong) 정회원
2021년 2월: 부산가톨릭대학교 컴퓨터공학과 학사
2021년 3월~현재: 부산가톨릭대학교 일반대학원 컴퓨터공학과 석사과정
〈관심분야〉 디지털 포렌식, 취약점 분석

김 도 현 (Dohyun Kim) 종신회원
2019년 8월: 고려대학교 정보보호대학원 공학박사
2017년 7월~2019년 8월: 한국전자통신연구원 정보보호연구본부 연구원
2019년 9월~2020년 3월: 고려대학교 정보보호연구원 연구교수
2020년 4월~2021년 9월: 부산가톨릭대학교 컴퓨터공학과 조교수
2021년 10월~현재: 부산가톨릭대학교 컴퓨터정보공학과 조교수
2020년 7월~현재: 부산가톨릭대학교 융합보안공학센터 센터장
〈관심분야〉 사이버 보안, 디지털 포렌식, 취약점 분석