랜섬웨어 암호기능 분석 보고서

- SimpleLocker -

2019. 07

차세대암호인증팀

KISA 한국인터넷진흥원
KOREA INTERNET & SECURITY AGENCY

Contents

1. 개요

SimpleLocker 랜섬웨어는 2015~2016년에 사이에 유포된 최초의 안드로이드 모바일 대상 랜섬웨어이다. 동유럽, 미주 지역을 기반으로 활동하였고, 2016년 후반까지 약 15만 회 정도 다운로드된 것으로 파악된다.[1] SimpleLocker 랜섬웨어는 스마트폰의 파일을 암호화하고, 암호화된 파일을 복구하기 위한 비용을 피해자에게 요구한다.

분석에 사용한 SimpleLocker 랜섬웨어 샘플의 해시 값은 [표 1]과 같다.

MD5	b98cac8f1ce9284f9882ba007878caf1
SHA1	72ec80b52ad38417240801dba1a730ab9804a2f9
SHA256	3c079b47a2129fe6c4b35774a02147de9dd9b28043570fff800318ee75beea7f

[표 1] SimpleLocker 랜섬웨어 샘플 정보

2. 랜섬웨어 분석

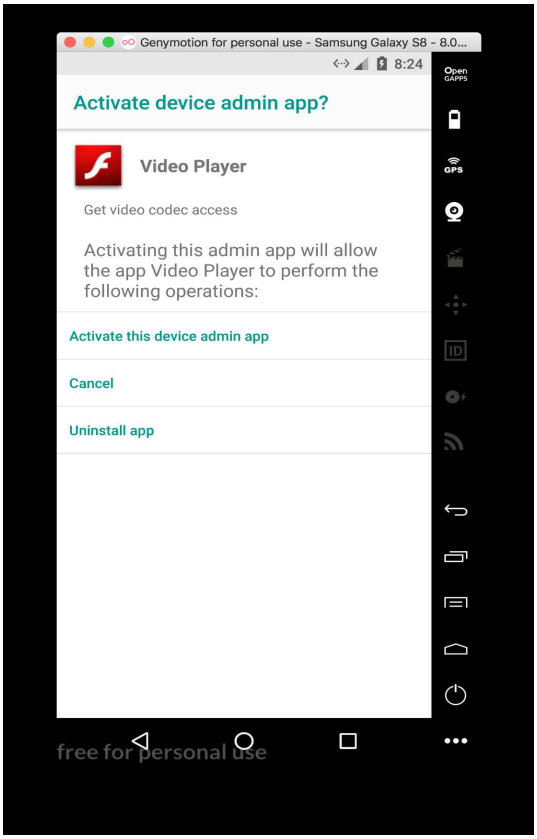
2.1 실행 과정

SimpleLocker 랜섬웨어를 설치하면 스마트폰의 애플리케이션 메뉴에 [그림 1]과 같은 아이콘이 생성된다.

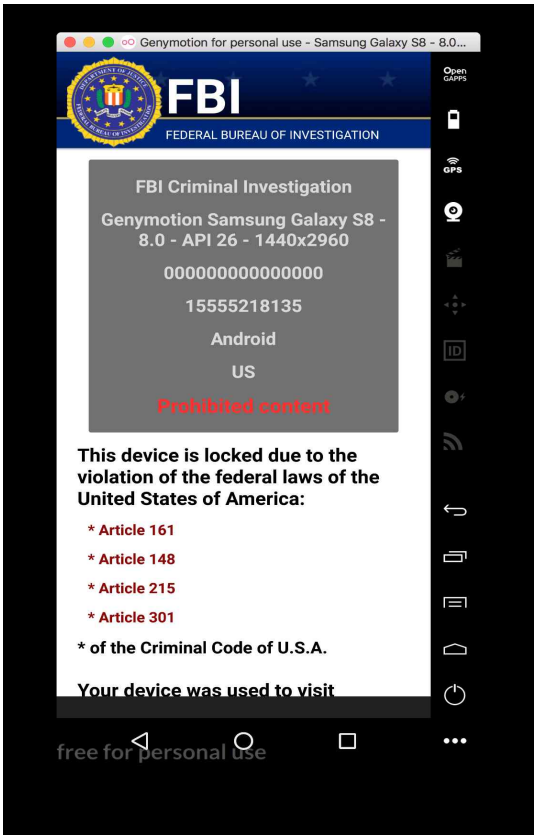
[그림 1] SimpleLocker 랜섬웨어 아이콘

생성된 아이콘은 SimpleLocker 랜섬웨어를 실행하는 아이콘이다. 이 아이콘을 실행하면 [그림 2], [그림 3]과 같은 화면이 나타난다 [그림 2]의 'Activate this device admin app'을 실행하면 [그림 3]의 'FBI Warning' 메시지가 나타난다.

1) The 5 biggest ransomware attacks of the last 5 years, www.csoonline.com/article/3212260/ransomware/the-5-biggest-ransomware-attacks-of-the-last-5-years.html



[그림 2] SimpleLocker 랜섬웨어 실행 화면



[그림 3] FBI Warning 메시지

SimpleLocker 랜섬웨어는 스마트폰에 설치된 후, 다양한 타입의 파일을 AES 암호화 키로 암호화하기 위해 스마트폰의 파일 시스템을 스캔한다. 스캔 후, [표 2]의 소스코드에 명시된 16종의 확장자를 가진 파일에 대해 암호화를 시도한다. 파일 암호화가 완료되면 암호화된 파일의 확장자를 '.encoded'로 변경한다.

```
public final class b
{
  public static final List a = Arrays.asList(new String[] { “jpeg”, “jpg”, “png”, “bmp”,
  “gif”, “pdf”, “doc”, “docx”, “txt”, “avi”, “mkv”, “3gp”, “mp4”, “zip”, “7z”, “rar”});
}
```

[표 2] 암호화 대상 파일 확장자

SimpleLocker 랜섬웨어는 감염 과정에서 IMEI[2] 번호, 디바이스 모델 등의 정보를 수집하고, 수집된 정보를 C&C 서버[3]로 전송한다.

2) International Mobile Equipment Identity, 국제 모바일 기기 식별코드로 휴대전화마다 부여되는 고유번호
3) 해커가 원하는 공격을 수행하도록 원격지에서 명령을 내리거나 악성코드를 제어하는 서버

[표 3]은 SimpleLocker 랜섬웨어가 스마트폰의 파일 시스템에 존재하는 파일들 중에서 암호화할 파일을 구분하기 위한 소스코드이다.

```
private ArrayList a = new ArrayList(); // 암호화 대상 파일 확장자를 가진 파일 저장
private ArrayList b = new ArrayList(); // 암호화 제외 대상 확장자를 가진 파일 저장
private final List c = Arrays.asList(new String[] { "encoded" });

private void a(File paramFile)
{
  File[] arrayOfFile = paramFile.listFiles();
  int i = 0;
  if (i >= arrayOfFile.length)
    return;
  File localFile = new File(paramFile.getAbsolutePath(), arrayOfFile[i].getName());
  if ((localFile.isDirectory()) && (localFile.listFiles() != null))
    a(localFile);
  while (true)
  {
    i += 1;
    break;
    String str = localFile.getAbsolutePath();
    str = str.substring(str.lastIndexOf(".") + 1); // 파일 확장자 구분
    if (this.c.contains(str))
    {
      this.b.add(localFile.getAbsolutePath());
      continue;
    }
    if (!b.a.contains(str)) // 암호화 대상 확장자
      continue;
    this.a.add(localFile.getAbsolutePath());
  }
}
```

[표 3] 암호화 대상 파일 구분

먼저, SimpleLocker 랜섬웨어는 listFiles 메소드를 사용하여 입력받은 경로 및 폴더의 파일들을 File 타입의 배열에 저장한다. getAbsolutePath 메소드를 사용하여 입력받은 경로의 절대 경로 값을 가져오고, getName 메소드를 사용하여 해당 경로의 파일 이름을 가져온다. 마지막으로, 이 두 가지 값을 사용하여 File 클래스 타입의 객체를 생성한다.

SimpleLocker 랜섬웨어는 반복문과 조건문을 사용하여 읽어온 파일 확장자가 암호화 대상 파일 확장자와 일치하는지 검사한다. 먼저, lastIndexOf 메소드를 사용하여 파일 경로의 가장 오른쪽부터 검사한 후, '.' 문자가 위치한 index 값을 반환한다. 반환받은 index 값에 숫자 1을 더하여 substring 메소드의 입력값으로 사용한다. 그 결과, 해당 파일의 확장자만 남게 되고 이를 str 변수에 저장한다.

str 변수의 값이 이미 암호화된 파일의 확장자인 'encoded'이거나, [표 2]에 해당하지 않는 파일 확장자인 경우 파일 암호화 제외 대상으로 분류한다. 그리고, 해당 파일의 절대 경로명을 배열 b에 저장한다. 반복문과 조건문의 실행이 완료되면 암호화 대상 확장자를 가진 파일의 절대 경로명만 배열 a에 저장된다.

SimpleLocker 랜섬웨어는 파일 암호화 대상 확장자를 구분하는 과정이 완료되면, [표 4]의 파일 암호화 관련 함수를 불러오는 소스코드를 실행한다.

```
private SharedPreferences d;

public a(String paramString)
{
  a locala;
  Iterator localIterator;
  if ((!this.d.getBoolean("FILES_WERE_ENCRYPTED", false)) && (c()))
  {
    locala = new a("32a12651860ccbd6f968bb9f479edffc"); // 암호화 키 생성 함수
    localIterator = this.a.iterator();

  while (true)
  {
    if (!localIterator.hasNext())
    {
      x.a(this.d, "FILES_WERE_ENCRYPTED", true);
      return;
    }
    String str = (String)localIterator.next();
    locala.a(str, str + ".encoded"); // 파일 암호화
    new File(str).delete();
  }
}
```

[표 4] 파일 암호화 관련 함수 로딩

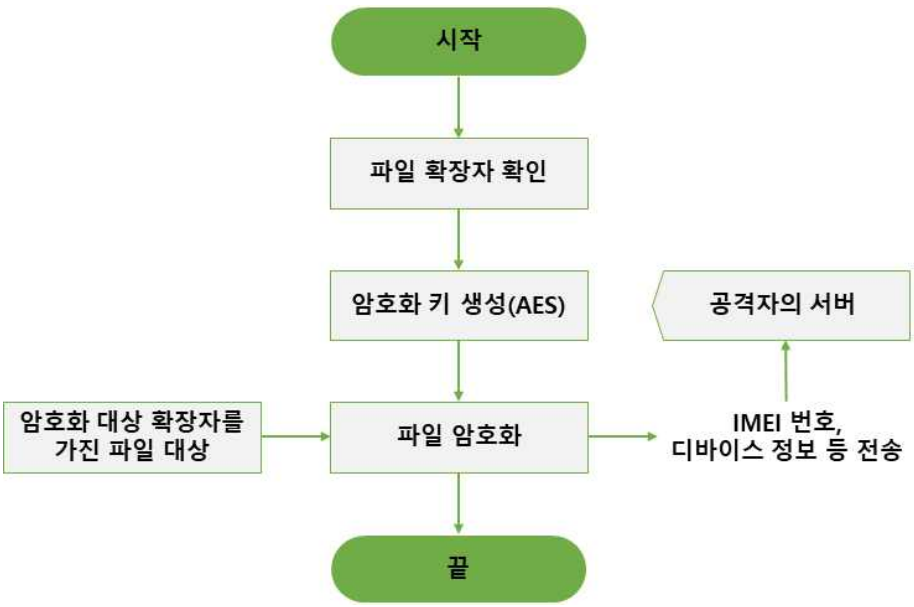
SimpleLocker 랜섬웨어는 안드로이드 초기 설정값을 설정하기 위해 SharedPreferences 인터페이스 값을 저장한 변수 d를 불러온다. getBoolean 메소드를 사용하여 d 변수의 값이 'FILES_WERE_ENCRYPTED'인지 확인하고, 아니면 파일 암호화를 수행한다.

d 변수의 값을 확인한 후에, a 함수는 고정된 문자열 '32a12651860ccbd6f968bb9f479edffc'를 사용하여 AES 암호화 키를 생성한다. 이는 고정된 값을 가진 암호화 키 사용을 의미한다. 키 생성이 완료된 후, 암호화 대상 확장자를 가진 파일의 절대 경로명이 저장된 배열 a의 값들을 읽어온다.

SimpleLocker 랜섬웨어는 반복문을 사용하여 안드로이드 초기 설정값을 변경하고, 각 파일에 대해 암호화를 수행하기 위한 입력값을 암호화 수행 함수로 전달한다. 먼저, x 클래스에 정의된 SharedPreferences.Editor 인터페이스의 putBoolean 메소드를 사용하여 'FILES_WERE_ENCRYPTED' 설정값(d 변수)의 상태를 true 값으로 변경한다.

값을 변경한 후, 암호화 대상에 해당하는 파일의 절대 경로명과 이 값에 '.encoded' 문자열을 추가한 값을 실제 파일 암호화를 수행하는 함수인 a의 입력값으로 전달한다. 마지막으로, File 클래스의 delete 메소드를 사용하여 암호화가 완료된 파일의 원본 파일을 삭제한다.

SimpleLocker 랜섬웨어의 실행 과정을 요약하면 [그림 4]와 같다.



[그림 4] SimpleLocker 랜섬웨어 실행 과정

2.2 암호화 키 생성 과정

SimpleLocker 랜섬웨어는 암호화 키를 생성하기 위해 javax.crypto API를 사용한다. 아래의 [표 5]는 SimpleLocker 랜섬웨어가 암호화 키를 생성하는 소스코드이다.

```
public a(String paramString) {
  ……(skip)
  this.a = Cipher.getInstance("AES/CBC/PKCS7Padding");
  this.b = new SecretKeySpec(paramString, "AES");
  this.c = new IvParameterSpec(new byte[16]);
  ……(skip)
}
```

[표 5] 암호화 키 생성 과정

SimpleLocker 랜섬웨어는 파일 암호화 방식을 사용하기 위해 Cipher 클래스의 getInstance 메소드를 사용한다. getInstance 메소드는 String 타입의 transformation 파라미터 값을 갖는다. 기본적으로 transformation 파라미터는 사용할 암호 알고리즘의 이름을 정의하고, 부가적으로 암호 운영 모드와 패딩 방법을 정의한다[표 6].

transformation 파라미터 : "암호 알고리즘" 또는 "암호 알고리즘/암호 운영모드/패딩 방법" 예시) Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");

[표 6] Cipher 클래스의 getInstance 메소드

[표 5]의 Cipher.getInstance("AES/CBC/PKCS7Padding") 소스코드에서 SimpleLocker 랜섬웨어는 파일 암호화 방식을 AES 암호 알고리즘, CBC 운영 모드 그리고 PKCS7Padding으로 정의한다.

SimpleLocker 랜섬웨어는 파일 암호화 방식을 정의한 뒤, 주어진 바이트 배열로부터 AES 암호화 키를 구성하기 위해 [표 7]에 설명된 SecretKeySpec 클래스의 SecretKeySpec 생성자를 사용한다. [표 5]의 'new SecretKeySpec(paramString, "AES")' 소스코드에 나타나 있다.

public SecretKeySpec(byte[] key, String algorithm)	
파라미터	**설명**
key	비밀 키 구성 요소
algorithm	주어진 비밀 키 구성 요소와 연관된 비밀 키 알고리즘 이름

[표 7] SecretKeySpec 생성자

첫 번째 파라미터인 paramString의 값은 고정된 문자열인 '32a12651860ccbd6f968bb9f479edffc'의 SHA256 해시 값으로 고정된 값이다. SimpleLocker 랜섬웨어는 이 값을 AES 암호화 키의 구성 요소로 사용한다.

두 번째 파라미터인 String 타입의 algorithm은 사전 정의된 키 구성 요소와 연관된 비밀 키 알고리즘의 이름을 나타내는데, SimpleLocker 랜섬웨어는 AES 암호 알고리즘으로 정의한다.

2.3 암호화 과정

SimpleLocker 랜섬웨어에서 파일 암호화를 수행하는 코드는 [표 8]과 같다.

```
private final Cipher a;
private final SecretKeySpec b;
private AlgorithmParameterSpec c;

public final void a(String paramString1, String paramString2)
{
  paramString1 = new FileInputStream(paramString1);
  paramString2 = new FileOutputStream(paramString2);
  this.a.init(1, this.b, this.c);
  paramString2 = new CipherOutputStream(paramString2, this.a);
  byte[] arrayOfByte = new byte[8];
  while (true)
  {
    int i = paramString1.read(arrayOfByte);
    if (i == -1)
    {
      paramString2.flush();
      paramString2.close();
      paramString1.close();
      return;
    }
    paramString2.write(arrayOfByte, 0, i);
  }
}
```

[표 8] 파일 암호화 수행

먼저, Cipher 클래스 타입의 변수 a를 init 메소드를 사용하여 초기화한다. 여기에서 init 메소드는 키와 일련의 알고리즘 파라미터를 사용하여 암호를 초기화하는데, 각 파라미터에 대한 설명은 아래 [표 9]와 같다.

public final void init (int opmode, Key key, AlgorithmParameterSpec params)	
파라미터	**설명**
opmode	암호 운영 모드 (ENCRYPT_MODE, DECRYPT_MODE, WRAP_MODE, UNWRAP_MODE)
key	암호화 키
params	알고리즘 파라미터

[표 9] Cipher 클래스의 init 메소드

SimpleLocker 랜섬웨어는 init 메소드의 첫 번째 파라미터의 입력 값을 1로 설정하였는데, 이는 'ENCRYPT_MODE'를 나타낸다. 두 번째 파라미터의 입력 값은 SecretKeySpec 타입의 변수 b로써, 암호화 키 생성 과정에서 AES 암호 알고리즘으로 구성한다. 세 번째 파라미터의 입력 값은 IV(초기화 벡터) 값으로, 크기는 16 bytes이다.

SimpleLocker 랜섬웨어는 초기화된 AES 암호화 키를 사용하여 파일을 암호화한다.

3. 복원 기술

SimpleLocker 랜섬웨어는 고정된 문자열('32a12651860ccbd6f968bb9f479edffc')의 SHA256 해시 값을 사용하여 AES 암호화 키를 생성한다. 그러므로, 고정된 문자열의 해시 값과 대칭키 알고리즘의 특성을 이용하여 복호화 키 생성이 가능하다. 아래의 [표 10]은 파이썬으로 개발한 SimpleLocker 랜섬웨어의 복구 소스코드이다.

```
from Crypto.Cipher import AES
from hashlib import sha256
def Decrypt_simplelocker(path_filename, filesize):
    readfile = open(path_filename, "rb")
    writefile = open(path_filename.split(".encoded")[0], "wb")
    IV = "\x00" * 16 // 복호화 시 사용할 IV 값
    KEY = sha256("32a12651860ccbd6f968bb9f479edffc").digest()
    cipher = readfile.read(filesize)
    encryptor = AES.new(KEY, AES.MODE_CBC, IV) // AES 복호화 키 생성
    plain = pkcs7_unpadding(encryptor.decrypt(cipher))
    writefile.write(plain)
    readfile.close()
    writefile.close()
```

[표 10] SimpleLocker 랜섬웨어 복구 소스코드

Module AES에서 제공하는 new 함수의 첫 번째 입력 값은 AES 복호화 키를 생성하기 위한 비밀키 값으로써, 고정된 문자열의 SHA256 해시 값을 사용한다. 두 번째 입력 값은 복호화 시 사용할 운영 모드로써, CBC 모드를 사용한다. 세 번째 입력 값은 복호화 시 사용할 IV 값으로써, \x00 * 16을 사용한다. new 함수는 세 가지 입력 값을 사용하여 AES 복호화 키를 생성한다.

SimpleLocker 랜섬웨어 복구 소스코드는 암호화된 파일 리스트를 읽어오고, 해당 파일의 암호화 시 추가된 '.encoded' 확장자를 제거한다. 그리고, new 함수로 생성한 AES 복호화 키로 암호화된 파일을 복호화 한다. 복호화 된 파일은 실행하였을 시 정상적으로 실행된다.

4. 결론

SimpleLocker 랜섬웨어는 AES 암호 알고리즘을 사용하여 파일 암호화를 수행한다. 고정된 문자열의 SHA256 해시 값을 사용하여 AES 암호화 키를 생성하기 때문에, 대칭키의 특성을 이용하면 복호화 키 생성이 가능하다. 그리고 모든 파일에 동일한 AES 암호화 키를 사용하여 암호화하기 때문에, 하나의 복호화 키를 사용하여 모든 파일을 복호화 할 수 있다. SimpleLocker 랜섬웨어 분석 결과는 아래 [표 11]과 같다.

상세 요소		분석 내용
해시함수		-
대칭키	알고리즘명	AES
	암호화 키	sha256('32a12651860ccbd6f968bb9f479edffc')
	IV	"\x00"* 16
	패딩여부	PKCS7Padding
	운영모드	CBC
암호화 API		javax.crypto API
대칭키 생성 함수		-
난수 생성기		-
엔트로피 수집		-
키 관리 방법	모든 파일 상이한 암호화 키	×
	모든 파일 동일한 암호화 키	O
	네트워크 연결에 따른 키 변화	×
할당 해제 수행 여부		-
제로화 수행 여부		-
운영체제 백업파일 삭제		-

[표 11] SimpleLocker 랜섬웨어 요소별 분석 결과

랜섬웨어 암호기능 분석 보고서 작성 공헌자

구 분	소 속	직 위	성 명
책임자	KISA	팀장	박해룡
작성자	KISA	선임	김기문
		주임	김대운
		주임	이영주

랜섬웨어 암호기능 분석 보고서 - SimpleLocker

2019년 7월 발행

발행처 KISA 한국인터넷진흥원
KOREA INTERNET & SECURITY AGENCY