Challenge 9: Mobile Malware

Submission Template

Submit your solution at http://www.honeynet.org/challenge2011/ by 17:00 EST, Wednesday, June 15th 2011. Results will be released around the third week of June.

Name (required):José LOPES ESTEVES	Email (required):jose.lopes-esteves@sogeti.com
Country (optional):France	Profession (optional): _ Security Professional

Question 1. Write an executive summary of this incident.	Possible Points: 3pts
Tools Used: **Microsoft word** ☺ Awarded Points:	

It seems that the user installed and ran an application which has a legit part and a malicious part. This application seems to be a currency converter.

The malicious payload then started to communicate with a server in an encrypted format, through HTTP POST requests. The malware sent the victim's IMEI, IMSI, operator and country, stored SMS, contacts, and freshly received SMS. It also made a phone call and sent a SMS to the following number: +33645324806.

After a in-deep analysis, it appears that the malware is a spy software that receives commands from a Command and Control server over HTTP.

The malware can spy on the SMS and the contacts, and is able to pass phone calls, send SMS, open URLs through the browser and to trigger the vibrator.

Question 2. Provide the phone brand, model, OS name and version.	Possible Points: 1pts
Tools Used:**Wireshark** Awarded Points:	

According to the several user-agent fields in HTTP requests captured :

The victim's phone is a **LG P970**
The running OS is **Android 2.2.2**

Question 3. Extract any suspicious application (if any). Detail your extraction	Possible Points: 4pts

method. Please provide name and SHA1 for each suspicious app.	
Tools Used: **testdisk** (linux disk recovery utility) Awarded Points:	
I found **ONE** suspicious application: **com.fc9.currencyguide-1.apk** (SHA1: **c630e3e9366c248a07287c2d72a7c02236ff92a5**) For extracting, I used the testdisk utility (open source tool by Christophe Grenier, available through linux aptitude) to correct the data.bin ext2/ext3 partition, parse the file system and extract the files and directories. In the app/ directory, all the installed applications apk files are available for further examination. My focus was turned to applications requesting strange permissions, and to their structure. The application I selected requests a lot of permissions, uses a "daemon" package, which is not very common, and its code is very obfuscated.. Furthermore, the "honeynet" string is in the manifest ☺	

Question 4. What permissions are requested by the malware(s)? Why it is suspicious?	Possible Points: 1pts
Tools Used: **android-apktool** Awarded Points:	
The malware requests a lot of permissions: • Internet • Read phone state • Access wifi state • Wake lock • Access network state • Receive boot completed • Camera • Vibrate • Access coarse location • Access fine location • Call phone • Send sms • Read contacts • Receive sms • Read sms It is suspicious because a currency converter or indicator (as the application seems to be at this step of the examination) does not need such intrusive permissions, like the camera, SMS read/write/send or contact reading, call phone etc…	

Question 5. Please provide a solution/s to quickly identify any suspicious API (please define your suspicious API according to your understanding).	Possible Points: 8pt
Tools Used: **android-apktool** Awarded Points:	
In fact in this case I think that the best way to identify suspicious APIs is to think about what the legit part of the application is supposed to do. By examining the AndroidManifest.xml, I got to the following conclusions:	

- The BootReceiver intent receiver is suspicious, because a currency converter does not need to run at each system startup.
- The activity "com.fc9.currencyguide.daemon" has no GUI (theme = "@android:style/Theme.NoDisplay"), which is very suspicious for an Activity.
- The service "com.fc9.currencyguide.daemon.CComService" is also suspicious, because it requires to be ran in another process. I don't understand why such an application needs a service that runs in a remote process….

Question 6. What is the malware's home server URL and where is it located? Where, in the code, is/are stored the command server(s) URL(s)	Possible Points: 4pt
Tools Used: **wireshark, dex2jar, jd, flagfox(firefox extension)** Awarded Points:	

The malware's home server is: **faeacdeadbeefada.zonbi.org**

A whois search tells that it is located at: **Gunzenhausen, Germany** (ip: **88.198.206.97**), the internet provider is **hetzner online**

The command server's url is DES encrypted, and is stored in the class **com.fc9.currencyguide.daemon.a**

Question 7. What can you say about the communications model between the malware and its C&C server?	Possible Points: 2pts
Tools Used: Awarded Points:	

It is a classical botnet client-server communication model.
The malware registers itself to the server, providing data used to derive a symmetric encryption key for ciphering the data sent in clear HTTP requests.

Then the malware periodically asks for commands, executes them and retrieves ciphered data in the POST "data" parameter.

The commands are sent by the server using an XML structure, containing the command and some parameters:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
        <rootElem>
                <cmd>getsms</cmd>
                <params>/01lO1110llOO.php</params>
        </rootElem>
```

Question 8. If encryption was used for the communication, which encryption algorithm was used? What was the key used? Explain how you found it.	Possible Points: 4pts
Tools Used: **dex2jar, jd, dedexer, eclipse, wireshark** Awarded Points:	

Encryption is user for ciphering the content of the clear HTTP dialog.

The algorithm is DES, with an 8 byte key.

The key used for communication encryption is [-60, -55, -105, 58, 69, -57, 0, 125]

To find the key, I first **decompiled the classes.dex** file of the APK using dex2jar and jd.
Then I exported the .class files to work with it in Eclipse, so that I could rename the classes, functions and packages.
Doing so, I've been able to **identifiy the variables used by the decryption**/encryption routines. But dex2jar is not perfect, and some parts of the code that initializes or instantiates those variables was not well translated into java code.
At this point I had to use dedexer to focus on the **com.fc9.currencyguide.daemon.b.h class** which contains the code that initiates the dialog with the server, and **sets the DES key.**.

A more precise description of the key exchange is done in the answer to question 12, section "Notes about cryptography"

Question 9. Please draw a graph of the decrypted communication flow, found in the pcap, between the malware and the C&C.	Possible Points: 4pts
Tools Used: **wireshark, eclipse** Awarded Points:	

For this answer, [C->S] means client to server and [S->C] means server to client. The numbers represent the packet number in wireshark, and the file names represent the destination files of the requests on the server.

Sorry if the answer is not really readable, but I was not sure if I was supposed to provide decrypted data or if it was enough to comment what happens, so I gave all the information I had.

First here is a summary of the communication flow. The detail of the data transmitted is given below:

1. **Key exchange**
2. **Registration to server**
3. Client waiting for command
4. **Command: Get all stored SMS**
5. Client waiting for command
6. **Command: Start SMS interception**
7. Client waiting for command
8. **Command: Get all contacts**
9. Client waiting for command
10. **Command: Call +33645324806**
11. Client waiting for command
12. **Command: Send an SMS to +33645324806**
13. Client waiting for command
14. **Command: Vibrate**
15. Client waiting for command
16. **Client sends incoming SMS to server**

Here is the detail of the deciphered communication:

KEY EXCHANGE

[C→S]--------2175--->>>>>>>>>>>>>>> /
p=00EABBE22F3A27C63780C932C76B351199
&g=00B7C0F36DF96306CC01FA6BE403E9A000
&x=678325046C23AA8838ACD869B29CBF50
&n=00ED706CC98EB0CAC407D020A1E91CC797

[S→C]------2183--<<<<<<<<<<<<<<<<<<

x=9915D4E8B2F342BEFC3E70C352D78F49
&n=291E97F5725C366482418E95BF39D4A9
&s=CCA2D1552FFEE88D933C5FBAF25E435803F1FFF9

REGISTRATION TO SERVER
[C->S] ---------2197----------------------------------->>>>>>>>>>>>>>> /reg.php
data= imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

[S->C] --<<<<<<<<<<<<<<<
HTTP 200 ok

WAITING FOR COMMANDS
[C->S] ----------2658----------------------------------->>>>>>>>>>>>>>> /data.php
data= imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

GET ALL STORED SMS
[S->C] -----------2680----------------------------------<<<<<<<<<<<<<<<

< ?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>getsms</cmd>
<params>/01lO1110llOO.php</params>
</rootElem>

[C->S] ------2860-->>>>>>>>>>>>>>> 01lo1110lloo.php
data= 21080=Votre mot de passe est strictement confidentiel : conservez-le précieusement et ne le communiquez pas à un tiers. Votre mot de passe est 3B6AT4
&+33666186296=Le 0645324806 remporte 1 chèque en euro!Composez le 0899650923 pr le retirer immédiatement. Jeu sous Controle d huissier (1E35/ap+0E34/mn)
&+33644066241=VocalMessenger: Vous avez recu 1 nouveau message vocal a 11:59. Pour le consulter, composez le 0899230625
Code confidentiel: 1701
(1e35+0e34-noSmsEnvStop)
&20904=Mobicarte - Compte Principal. Attention, il vous reste moins d'une journée pour utiliser votre crédit de 3.93 EUR.
&20904=Mobicarte - Compte Principal. Attention, il ne vous reste plus qu'une journée pour utiliser votre crédit de 4.05 EUR.
&20904=Mobicarte - Compte Principal. Attention, il vous reste moins d'une semaine pour utiliser votre crédit de 4.17 EUR.
&20904=Mobicarte - Compte Principal. Attention, il ne vous reste plus qu'une semaine pour utiliser votre crédit de 4.29 EUR.
&20904=mobicarte: votre ligne est identifiée, vous pouvez maintenant choisir votre Bonus et recharger votre compte au #123# (appel gratuit).
&20904=Mobicarte: Votre numero de telephone est le 0645324806, valide jusqu'au 05/01/12.
&20904=Bienvenue chez Orange,votre numéro mobicarte est le 0645324806. Vous bénéficiez d'un crédit de 5E de

communications valable jusqu'au 05/01/12 en Fce métro.&

[S->C] ----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
HTTP 200OK

WAITING FOR COMMANDS
[C->S] ------4484----------->>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

START SMS INTERCEPTION
[S->C] ------4488----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>smsspy</cmd>
<params></params>
</rootElem>

WAITING FOR COMMANDS
[C->S] ------4484----------->>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

GET ALL CONTACTS
[S->C] -------4693----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>getcontacts</cmd>
<params>/0S550SSSO5.php</params>
</rootElem>

[C->S] ------4700----------->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> /0S550SSSO5.php
data=Niobe=5553&Agent Smith=5551&Merovingian=5559&Trinity=5558&Seraph=5557&Neo=5555&Morpheus=5554&

[S->C] ----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

HTTP 200OK

WAITING FOR COMMANDS
[C->S] ------4738----------->>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

CALL +33645324806
[S->C] -------4745----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>call</cmd>
<params>('+33645324806',)</params>
</rootElem>

WAITING FOR COMMANDS
[C->S]------5990----------->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

SEND AN SMS
[S->C] ---------6050----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>sendsms</cmd>
<params>+33645324806;Bot can send SMS !</params>
</rootElem>

WAITING FOR COMMANDS
[C->S]------6269----------->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

VIBRATE
[S->C] ---------6272----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rootElem>
<cmd>vibrate</cmd>
<params>1000</params>
</rootElem>

WAITING FOR COMMANDS
[C->S]------6301----------->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

[S->C] ---------6306----------<<<<<<<<<<<<<<<<<<<<<<<<<<<<
HTTP 200 OK

(……)

WAITING FOR COMMANDS
[C->S]----------->>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> /data.php
data=imei=356772040481677&imsi=208013002954000&opname=Orange F&opcode=20801&opiso=fr

SMS RECEIVED C TO S
[C->S]--------13460----------->>>>>/s.php
data= 21080=Votre mot de passe est strictement confidentiel : conservez-le précieusement et ne le communiquez pas à un tiers. Votre mot de passe est Y0ucantS33iT&

Question 10. What personal information were leaked during this incident A special *secret* information was leaked, Explain how and what it was.	Possible Points: 2pts
Tools Used: Awarded Points:	

During the incident, the malware sent:

- **Names and phone numbers** of all **the contacts** who have a phone number
- All the SMS that are on the phone, some of them containing **user password for a voicemail service**, his **operator's client account access password**, his **phone credit** and his **phone number**
- An incoming SMS containing the user's **new password for his operator's client account access**.
- The phone's **IMEI** and the user's **IMSI**
- The **operator name and code**
- The **operator country**

I guess that the special secret information is the **password for the user's account** access on the operator's service platforms. This information was sent by **SMS** to the user, when he requested it because he forgot it or changed it.

Or maybe it is the fact that the user is using a **prepaid SIM card**, also deduced from an **SMS** from the operator, with his phone number**.**

Question 11. What particular techniques are used by the malware to harden analysis or to evade detection? What unusual behavior can be noticed?	Possible Points: 6pts
Tools Used: Awarded Points:	

To harden analysis and or evade detection, the malware used:

- An ugly **code obfuscation**
- A **clustering of the code** in a lot of packages and classes (some classes are empty…)
- All the critical/suspicious **strings are DES encrypted or SHA1hashed** (they do not appear in clear text in the code). A more precise description is done in the answer to question 12, section "Notes about cryptography"
- **The key** for deciphering those strings is **generated using the target's IMSI**, so that an analysis of the malware off the phone without the victim's IMSI and without network traffic dumps containing the IMSI is much harder.
- The malicious part of the malware is **not triggered if the application runs on the emulator** (e.g. the DEVICE ID, MODEL, BUILD are checked)
- The malicious part of the malware **is not triggered if the IMSI is not the target's IMSI**, so that only the target could possibly detect an unusual behavior
- The malware is a **part of a working regular application**, which is not suspicious and which justifies some of the permissions requested.
- The malicious part of the application is a service, hence it **has no GUI** and is not listed by the system as a running process through the "applications" menu.
- The C&C server can ask to **unregister the SMS intent receiver**
- **The communication** with the server is **DES encrypted**, and the key changes every time the malware restarts.
- The client and server exchange **a lot of junk data** (that is not used) during the key exchange.

What I find unusual is the fact that the IMSI is used both for encryption of strings and as a checked value for running the payload. This attack is directed to a single user, and directly related to a mobile operator's user account. The target is therefore a unique PERSON (or more precisely to a unique SIM card). By the way, I'm curious about how the attackers got the IMSI of the target (maybe with another application, or with an IMSI catcher attack)

Question 12. Provide a detail analysis of the malware behavior and features.	Possible Points: 10pts

Tools Used:
Awarded Points:

Infection vector:

The malware is part of an apparently legit application: a currency converter. The infection is caused by the install of the application.

Malware behavior:

When the application is installed, the malware registers:

- an invisible (no GUI) activity that is launched every time the application is launched ("com.fc9.currencyguide.daemon.fc9", an entry point of the malicious part)
- a BOOT_COMPLETE intent receiver to start the malicious part on system startup (another entry point of the malicious part)
- a service ("com.fc9.currencyguide.daemon.CComService", the malicious part) that runs in a separate process, that can be launched by a particular intent.

This means that the malicious part of the application is launched:

- each time the legit application is started
- on system startup;
- and runs silently in another process during system runtime

When the malicious part of the application runs:

- It checks if the IMSI corresponds to the target's IMSI, and if not, it does nothing
- It checks if it is running on the emulator, by checking the "Build.DEVICE_ID", "Build.MODEL" and "Build.PRODUCT" strings provided by the system. If so, it does nothing. Those strings are not in the code. Instead, the comparison is performed on SHA1 hashes (followed by a substitution). A more precise description is done in the section "Notes about cryptography"
- Then if these checks are passed, it starts communicating with the C&C server using the internet connection.

C&C server communication:

- The server is located at: faeacdeadbeefada.zonbi.org (the string is DES encrypted in the code, and decrypted on startup)
- The port used for the communication is 443(the string is encrypted in the code, and decrypted on startup), but without using HTTPS.

Then the following protocol is used:

Key Exchange:

- The malware first sends random numbers p, g, x, and n to the server.
- The server then sends numbers back: x, n and s.
- Client's p, g and x, and server's x are used to generate a DES key that will be used to cipher the data transmitted between the client and the C&C server, through clear HTTP POST requests and answers. The DES key for this session was [-60, -55, -105, 58, 69, -57, 0, 125]. A more precise description is done in the section "Notes about cryptography"

Registration:

- The client registers to the server (request to /reg.php), providing the IMEI, IMSI, the name of the operator, the code

of the operator and the country code

Command loop:

- The client requests for commands (request to /data.php), providing the IMEI, IMSI, the name of the operator, the code of the operator and the country code. This request is repeated each time the client asks for new commands.
- The server sends an HTTP response, which can be either empty (HTTP 200 OK) or which can contain an encrypted command (the structure will be described later)
- The client parses and executes the command, and sends the answer back, in a request to /data.php (default) or to another URL specified in the command (ex: /0S550SSSO5.php)

Command format:

- The server sends commands to the client using an XML string, containing a command tag and a parameter tag
- The parameter tag can contain several parameters, separated by a semi-colon

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
    <rootElem>
        <cmd>getcontacts</cmd>
        <params>/0S550SSSO5.php</params>
    </rootElem>
```

Supported commands:

Vibrate

- Cmd: vibrate
- Params: none
- Description: activates the phone's vibrator

Make phone call

- Cmd: call
- Params: phone number
- Description: call the provided phone number

Send SMS

- Cmd: sendsms
- Params: phone number ; SMS body
- Description: send a SMS to the provided phone number, which content is the SMS body

Retrieve the contacts

- Cmd: getcontacts
- Params: URL
- Description: send all name/phone number pairs of all the contacts that have a phone number to the provided URL

Open URL

- Cmd: goto
- Params: URL
- Description: go to the provided URL using the built in browser

Retrieve SMS

- Cmd: getsms
- Params: URL
- Description: send all stored SMS to the provided URL

Activate SMS interception

- Cmd: smsspy
- Params: none
- Description: register the intent receiver for incoming SMS. When a SMS is received, it is sent to /s.php

Disable SMS interception

- Cmd: smsunspy
- Params: none
- Description: unregister the intent receiver for incoming SMS

Notes about cryptography:

The malware uses cryptography to

- Protect hardcoded strings
- Protect the communication with the C&C server

The hardcoded strings are protected using two methods, depending on the needs:

- For strings used in comparisons: a substitution function is combined to a SHA1 hash, so that comparisons are made on the substituted hashes (which are available in the source code)
- For strings that need to be determined (a hash is not appropriate), DES encryption is used. The key is derived from the substituted SHA1 hash (same as above) of the IMSI. Hence, the key depends only on the IMSI, and in this case, it was [99, -78, 82, -10, 6, -12, 22, 127]

The communication with the C&C server is protected using a DES encryption. The key is chosen through a challenge-response key exchange protocol with the server:

The client generates some numbers:

- Pc a random number
- Gc a random number
- Nc a random number
- Cc a hardcoded number (0963485269741EF69AE45D69F23AA9)
- Xc = Gc.modPow(Cc, Pc)

The client sends substituted string values of Pc, Gc, Xc, and Nc (the substitution is the same as above)

The server sends back three numbers: Xs, Ss, Ns (but only Xs is used by the client)

The client parses Xs as a hexadecimal string value and derives the DES key:

- Ic = Xs.modPow(Cc, Pc)
- B = Ic.toByteArray()
- The DES key is the last 8 bytes of B (in this session it is [-60, -55, -105, 58, 69, -57, 0, 125])

This DES key is renegotiated every time the CComService service restarts.

This means that the server knows the value of Cc, and that the client could send only Pc to the server, and the server only Xs to the client, for being able to generate the key. All the other exchanged values are only junk values for protocol obfuscation.

Bonus Question. Please provide a method to block (or request permission from Android	Possible Points: 8pts

(similar to UAC concept)) when any suspicious call received from Android.	
Tools Used: Awarded Points:	

Obviously, a **user permission request** for accessing private data and performing actions that cost money to the user should be performed. In that case the user will be aware that the currency converter tries to make a phone call, or to send a SMS. Another mechanism that allows the user to always grant the permission and avoid the permission request prompt can be added, so that the user experience is not too bad due to security.

Additionally, some researchers proposed a mechanism that allows the user **to choose whether he wants to provide true information to an application or not**. The IMSI for example can be replaced by a random one if the user chooses so. This could be a good way to control the information that is sent from the phone.

I think using those two security measures, this particular malware would become harmless, and more generally as a Android user I really miss such features in my daily use.