New type of threat: Mobile botnets on Symbian

Cao Yang[1], Zou Shihong[1, 2], Li Wei[1]

1, NetQin Mobile Inc.

2, Beijing University of Posts and Telecommunications

About Author(s)

Cao Yang is a virus analyst in Threat Response Team, Research Center, NetQin Mobile Inc. Contact Details:caoyang@netqin.com

Zou Shihong is vice president of NetQin Mobile Inc.

Contact Details:zoushihong@netqin.com

Li Wei is director of Research Center, NetQin Mobile Inc

Contact Details:liwei@netqin.com

Mail Address: No.4 Building, Heping Li East Street 11,Dongcheng District, Beijing, China 100013

Fax: +86 10 85655518

Keywords

"Zombie", botnets, mobile security, Symbian worm, malicious server

New type of threat: Mobile botnets on Symbian

Abstract

In last June, sets of viruses broke out on Symbian phones in China. Within one week, more than 1 million phones were infected, according to CNCERT. The statistics of victim has been climbing since then. Compared to the viruses broke out previously, these viruses bear more resemblance to the "botnets" virus on PC, so they are given the name "Zombie". (The formal virus names are "FC.ThemeInstaller.A", "AVK.DuMusic.A" and their variants.)

This paper will firstly provide some background information about "Zombie", and then introduce the security mechanism on Symbian OS 9, the basic assembly code and reverse engineering techniques, all of which are essential to understand the latter part. Next, this paper will explain the basic features of "Zombie" from an implementation aspect, including how they propagate, how they protect themselves against anti-virus, and how they spread etc. These features are illustrated with assembly code and regenerated standard API on Symbian. Most importantly, this paper will explain the new feature of "Zombie", that remote malicious server plays an important, even vital role in the attack and spread of "Zombie". It will show how the server commands "Zombie" to conduct malicious behaviours the hacker wants. These commands can range widely, from uploading sensitive information of the victim to downloading new addresses of the remote server for protecting "Zombie" from operator, such as China Mobile's blocking. This paper will provide these commands already known, but there will always exist new commands, since the high expansibility of "Zombie" allowing them to accept whatever commands defined by hacker. By showing and explaining these "protocol" between remote malicious server and 'Zombie", this paper will provide an overview of the whole process of "Zombie" attack and the framework of this new type of mobile threat.

Finally, this paper will conclude on the importance of the "Zombie" and their influence to mobile security world widely.

Introduction

As is known to all, China has the largest smart-phone market around the world. Driven by potential huge profit, many hackers take risks producing mobile viruses. This situation is especially critical on Symbian, since it is the most popular platform currently in China. In 2010 alone, more than 1700 mobile viruses (NetQin, 2010a) have been captured, which exceeds the three previous years combined. Besides increasing in amount, the viruses are also developing in the ability to attack, defend and spread. By now, they have grown strong enough to cause havocs on mobile platforms.

In this paper, the "Zombie" viruses will be discussed. The word "Zombie" is an informal name given to viruses possessing the typical characteristic of botnets - all the victims are, at least partially, controlled by the hacker. In last June, "Zombie" broke out and infected more than one million phones within one week (CNCERT, 2010), according to CNCERT (China National Computer network Emergency Response Technical). Although China's largest operator-China Mobile, took several measures, such as blocking malicious servers, deactivating the phones which sends lots of featured short messages, the amount of victim was still climbing. Due to the strong transmissibility and robustness of "Zombie", the direct economic losses have totalled twenty million Yuan (NetQin 2010b). Moreover, the outbreak of "Zombie" also draws the attention of mainstream medias. For example, CCTV (China Central Television) gave a special coverage (CCTV, 2010) on this security incident. "Zombie" was firstly captured by us in last June, with the name “NmapPlug.A”. Till now, dozens of variants have been found. They were given names like “FC.ThemeInstaller.A” and “AVK.DuMusic.A” in our virus database. [1]

This paper will give a detailed analysis of ThemeInstaller.A which is a classic representation of “Zombie”. Firstly, it provides background information about Symbian’s security mechanism and introduces some reverse techniques on Symbian platform. Then, the analysing procedure begins. Each conclusion will be illustrated with assembly code and regenerated standard API. Moreover, the specific protocol between “Zombie” and remote server will be provided and explained, to reveal the framework of this threat. Finally, based on these findings, this paper concludes on the work done and predicts the impact “Zombie” brings to mobile security world widely.

Required Knowledge

In this section, we will introduce some basic knowledge which can help to understand the analysis in latter part. These knowledges include three aspects: Symbian’s security mechanism, ARM assembly code, and some reverse techniques.

Symbian’s security mechanism

Since the version 9, Symbian introduces a new security mechanism, which mainly falls into three parts: data caging, capabilities and Symbian Signed. In general, if the application wants to conduct certain behaviours, it must have the corresponding capability. The capability is granted by Symbian through the form of certificate. Before the grant of certain certificate, Symbian will do checks on the software. The scale of check depends on the level of capability required. Currently, most viruses get certificate through Express Sign, which is enough to cause havocs on Symbian. Data caging is the restrictions for directory’s access. For example, every application has a private directory which cannot be accessed by other applications if they don’t have the “AllFiles” capability. The system’s directories such as “sys\bin” also have restrictions for read/write operations. [2]

ARM assembly code

As we know, Symbian is based on ARM’s architecture, in other words, the form of code running on the phone is actually ARM assembly.[3] For this paper, we only need to know a few instructs.

- PUSH {r1, r2}: push the value of r1 and r2 ,to the stack of current function
- MOV r1, r2: set r1 with the value of r2.
- LDR r2, =off_794B0138: load the value of variable off_794B0138 to R2.
- LDR r2, [r1]: load the byte at the address indicated by r1
- BLX r1: call the function whose address is indicated by r1
- CMP r1, r2: r1 minus r2 and the result will affect certain bits in flag register.
- BLE loc_794AC48C: check the result of CMP, by accessing certain bits in flag register. For example, taking previous instruct into account, if the value is less than or the same to

[1] We have given a special coverage about “Zombie” on our website. Please refer to “http://www.netqin.com/market/jiangshi/” for more information.

[2] Detailed information can be found on Nokia’s wiki (Nokia, 2009).

[3] The languages such as Java and Python are not the same. They are runtime languages, and run on a virtual platform.

the value of r2, call the function loc_794AC48C.

- B: directly jump to an address without return
- STR r2, [r0] : store r2 to the address indicated by r0

Several reverse techniques

Descriptors

Actually, descriptor is the predefined format for strings on Symbian. The identification of plaintext is important during reverse engineering. There are mainly five kinds of descriptors, summarized in the table below.

TBuf	3
TBufC	0
HBufC	0
TPtr	4
TPtrC	1

Table 1: class code for descriptors

The number is the value of their first half byte, which is reserved to identify the class of descriptors. It is a little more difficult to read the string for TPtr and TPtrC. The structure of TPtr is shown below. (The structure of TPtrC doesn't have the max length field.)

4bits:type	28bits:length	32bits:max length	32bits:address of the real string

When the descriptor is TPtr or TPtrC, we should jump to the real address to get strings. The other three descriptors can be directly read, and will not be discussed here.

Function arguments

When the system calls functions, the storage of arguments have mainly two cases. For a non-static member function of a class, register r0 always stores the "this" pointer of this class, and the augments are stored one after another in r1, r2……. For example, when we call the function CTelephony::GetPhoneId(TRequestStatus &,TDes8 &), the register r0 stores "this" pointer of CTelephony's object, r1 stores the address of TRequestStatus's object, r2 stores the address of an descriptor. But when the function to be called is a static function, then r0 will be used as ordinary registers.

Class identification

The reverse engineering of a class is the most important part, especially on Symbian. Because there are callbacks and virtual functions used everywhere. Without the knowledge of the classes' structure, some virtual functions which are not directly called will be missed during analysis. Actually, they may also execute, through dynamic function calls.

The vtable is the key to know about class, and it will be visited during the class's construct period. Usually, system will allocate memories for a class before constructing it. On Symbian, the allocation method is usually "User::AllocZL". After allocation, the construct procedure begins. Actually, the construct is a recursion period. The pseudo code is shown below.

```
Alloc memory;
Call ObjectConstructor;
FUNC ObjectConstructor
    WHILE has next parent class
        Call ObjectConstructor of this parent class
    ENDWHILE
    Put offset 4 bytes of vtable address into first 4 bytes of object's memory
END FUNC
```

We can see that the vtable address can be got though the construction period.

Dynamic call

Dynamic call supports one of the three main features of C++: polymorphism. The steps implemented on ARM platform is shown below.

LDR R0, [R4]; R4 stores "this" pointer of current class. In this step, R0 will store the object's first byte.

LDR R1, [R0, #0xC]; 0xC is the offset of vtable, and this step is used to get address of related virtual function

MOVS R0, R4; R4 is "this" pointer, which stores the address of current class object

BLX R1; call the virtual function

The analysis of "Zombie"

Beyond all doubt, "Zombie" has earned its fame, with millions phones infected and a huge botnets created. Fortunately, the hacker's aim is not attacking but earning money, otherwise this botnets is large enough to disable current telecommunication network.

Symbian 9's security mechanism is more rigorous than other OS and was once thought a good solution to malware problem. Even so, "Zombie" has successfully created a botnets on this platform. It's believed that the special features of "Zombie", and an elaborately-designed protocol with remote server, make mobile botnets comes into reality.

The next sections are organized as follows. First, we will summarize the typical features of "Zombie". Second, the static structure of ThemeInstaller.A will be introduced. Then, the concrete features will be analysed, illustrated with related reversed assembly code and flowchart. Finally, the decrypted protocol between server and ThemeInstaller.A will be provided and explained. Based on these results, we will illustrate the framework of this new type of threat.

Typical features

The typical features of “Zombie” fall into three categories: concealment, defensiveness and transmissibility. [4]

- Concealment: strategies taken to hide or disguise.
  - Clear traces in system’s log after conduct communication activities, such as connecting to Internet or sending messages, etc.
  - Go into self-destruction period, when certain task finishes.
  - Only start malicious tasks when the phone is not being used.
  - Conduct activity, such as making new calls, sending messages, installing software and connecting to Internet, in a stealthy way.
  - Only release and install new malware, when there are suitable amount of software installed.
  - Clear records in system’s installation log after silent installation.
  - Hide from system’s task list.
- Defensiveness: strategies taken to defend itself against stop or elimination.
  - Attack security softwares.
  - Schedule task to launch itself.
  - Send messages via socket, which will escape the monitoring or control of other softwares. [5]
  - Daemon process is implemented to defend against process killing.
  - Daemon package is implemented to protect itself from uninstallation.
  - Anti-uninstallation in a violent way.
- Transmissibility: strategies taken to spread.
  - Send short messages with a download link in the text. The recipient and message text are customized by the hacker.
  - Download and install new malware from malicious server

Besides all these “local” features, “Zombie” is actually notorious for the botnets feature: all the viruses are controlled by the remote server. Through interactions with “Zombie”, the remote server can configure certain behaviours, such as messages, call, and self-protection, etc.

Structural analysis

This section mainly discusses the structure of ThemeInstaller.A[6]. Other variants have similar structure and features.

[4] These features are merged version of all variants, such as DuMusic.A and NmapPlug.A.

[5] This point rectifies previous opinion in (Axelle Apvrille, 2010), that Symbian has only two messaging methods.

[6] sha1: C091665B8D48D37EA4AC4BA0FC5FF82868BFD37C.

ThemeInstaller.A arrives as a simple package including four data files and one executable. Its Symbian signed and the certificate is issued to “Hangzhou Ruixi Technology Co., Ltd.” The defects of Symbian Signed will not be discussed here, since it has been explained in (Axelle Apvrille, 2010). The following script is extracted from the package using SisContents (SisContents 2010).

"C_System\Data\Theme\0.dat"-"C:\System\Data\Theme\0.dat"

“C_System\Data\Theme\1.dat"-"C:\System\Data\Theme\1.dat"

"C_System\Data\Theme\2.dat"-"C:\System\Data\Theme\2.dat"

"C_System\Data\Theme\3.dat"-"C:\System\Data\Theme\3.dat"

"C_Resource\Apps\OviUpdate_20031C43.rsc"-"C:\Resource\Apps\OviUpdate_20031C43.rsc"

"C_sys\bin\ThemeInstaller.exe"-"C:\sys\bin\ThemeInstaller.exe", FR, RI

OviUpdate_20031C43.rsc is a standard resource file.ThemeInstaller.exe is an executable file. The flag “FR, RI” indicates it will run immediately after the installation. The four data files are actually encrypted version of standard installation packages: sisx or jar. They will be decrypted and installed by ThemeInstaller.exe. The decrypted version of 0.dat is a safe jar file, which may be hacker’s trick to convince the user nothing but safe software installed. Other three data files are the real “Zombie” viruses and their names are “Ovi Update”, “Ovi Store Installer” and “OviStore”. (Certainly, these packages are also Symbian Signed. The certificate is issued to “Hangzhou Ruixi Technology Co., Ltd.”) In latter section, we will explain the relationship between these packages. The structure of “Ovi Update” is as follows, with unimportant files omitted.

"sys\bin\OviUpdate.exe"-"!:\sys\bin\OviUpdate.exe", FR, RI

"C_sys\bin\DebugSrv.exe"-"C:\sys\bin\DebugSrv.exe"

"C_sys\bin\TrafficD.exe"-"C:\sys\bin\TrafficD.exe"

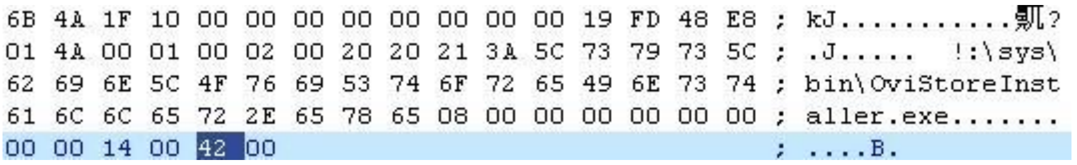
"C_sys\bin\OviStoreInstaller.exe"-"C:\sys\bin\OviStoreInstaller.exe"

"C_private\101f875a\import\[20031C45].rsc"-"C:\private\101f875a\import\[20031C45].rsc"

"C_sys\bin\AssistantProtect.exe"-"C:\sys\bin\AssistantProtect.exe"

"C_sys\bin\RunAssistant.exe"-"C:\sys\bin\RunAssistant.exe", FR, RR, RW

As the related flags indicate, OviUpdate.exe will run after installation and RunAssistant.exe will run during uninstallation. The file [20031c45].rsc will be copied into “c:\private\101f875a\import” on the phone, which is Symbian OS 9’s typical way to start applications on phone boot. The hex dump of this file shows that OviStoreInstaller.exe will be launched.



```
6B 4A 1F 10 00 00 00 00 00 00 00 00 19 FD 48 E8 ; kJ...........飘?
01 4A 00 01 00 02 00 20 20 21 3A 5C 73 79 73 5C ; .J.....  !:\sys\
62 69 6E 5C 4F 76 69 53 74 6F 72 65 49 6E 73 74 ; bin\OviStoreInst
61 6C 6C 65 72 2E 65 78 65 08 00 00 00 00 00 00 ; aller.exe.......
00 00 14 00 42 00                               ; ....B.
```

The structure of “Ovi Store Installer” is much simpler, and the extracted script is shown below. Here, we only need to know that RunAssistantProtect.exe will run during uninstallation. What it handles will be discussed later.

"sys\bin\Assistant.exe"-"!:\sys\bin\Assistant.exe"

"sys\bin\RunAssistantProtect.exe"-"!:\sys\bin\RunAssistantProtect.exe", FR, RR, RW

There is only one executable in Package “OviStore” and it also runs after installation.

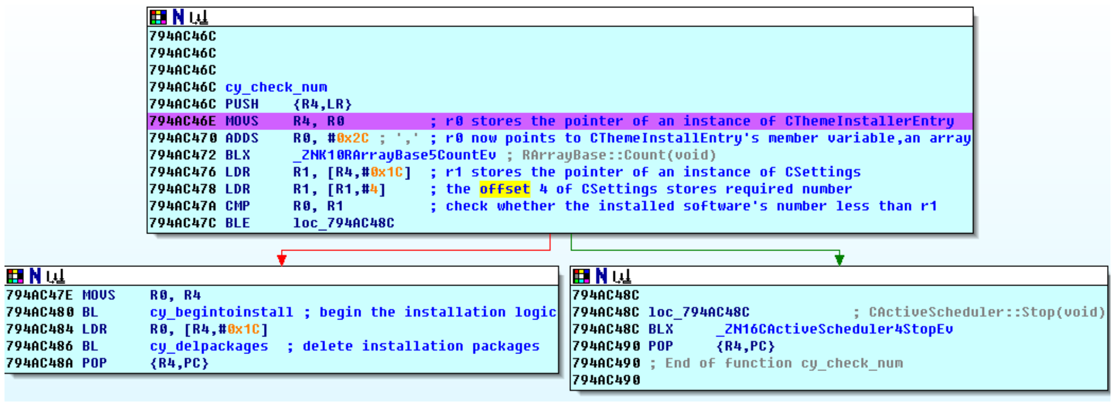
"sys\bin\OviStoreClient.exe"-"!:\sys\bin\OviStoreClient.exe", FR, RI

“Local” features

In previous section, we are acquainted with the main features of “Zombie”. Without any exaggeration, they can represent the most complicated strategies or technologies available currently.[7] We will discuss a subset of ThemeInstaller.A’s features. The discussion will focus on how these features are implemented. Some will be explained with illustration; some will be explained with assembly code. The features not discussed here are either explained before (in other papers), or just too simple to be mentioned.

Features during installation

According to the structural analysis, we know that ThemeInstaller.exe will run automatically during installation. Actually, its main task is decrypting the four data files and installing them. However, before that, this binary will check the number of installed software on the phone. In other words, if the victim’s phone installs less than certain (8 here) number, it won’t install the actual “Zombie” viruses. Good way to disguise.



```
794AC46C
794AC46C
794AC46C
794AC46C cy_check_num
794AC46C PUSH    {R4,LR}
794AC46E MOVS    R4, R0              ; r0 stores the pointer of an instance of CThemeInstallerEntry
794AC470 ADDS    R0, #0x2C ; ',' ; r0 now points to CThemeInstallEntry's member variable,an array
794AC472 BLX     _ZNK10RArrayBase5CountEv ; RArrayBase::Count(void)
794AC476 LDR     R1, [R4,#0x1C]  ; r1 stores the pointer of an instance of CSettings
794AC478 LDR     R1, [R1,#4]     ; the offset 4 of CSettings stores required number
794AC47A CMP     R0, R1          ; check whether the installed software's number less than r1
794AC47C BLE     loc_794AC48C
```

```
794AC47E MOVS    R0, R4
794AC480 BL      cy_begintoinstall ; begin the installation logic
794AC484 LDR     R0, [R4,#0x1C]
794AC486 BL      cy_delpackages   ; delete installation packages
794AC48A POP     {R4,PC}
```

```
794AC48C
794AC48C loc_794AC48C            ; CActiveScheduler::Stop(void)
794AC48C BLX     _ZN16CActiveScheduler4StopEv
794AC490 POP     {R4,PC}
794AC490 ; End of function cy_check_num
794AC490
```

Figure 1 IDA’s screenshot of ThemeInstaller.A check the num of installed software

As shown in Figure 1, the binary will check the number here. If there are enough software, it will begin the installation logic, otherwise it just directly stop the active scheduler. [8] The required number is 8, which is initialized in the constructor of class CSettings.[9]

```
.text:794AC5B6      STR     R1, [R0, #0x10]
.text:794AC5B8      MOVS    R1, #8              ; initialize the required number to 8
.text:794AC5BA      LDR     R2, =off_794B0138   ; load the address of virtual table
.text:794AC5BC      STR     R1, [R0, #4]    ; store r1 in offset 4
.text:794AC5BE      STR     R2, [R0]            ; initialize the first variable of class CSettings.
```

[7] What API can be used is restrained into the scope of Express Signed.

[8] Active scheduler is a part of Symbian’s featured active object framework. Stop the active scheduler usually equals to exiting the program.

[9] The name “CSettings” is got through the RTTI information right before its vtable.

Symbian has provided class RSisRegistrySession and RSisRegistryEntry to handle the information of installed softwares. Though RSisRegistrySession's function- InstalledPackagesL, the hacker gets an array of installed software's information. The length of that array is the number of software installed.

Remote debugging is a nice feature supported by recent version of IDA Pro. Though remote debugging, we can the fetch decrypted package directly, without the need to understand specific decryption algorism. All we need to do is just adding breakpoints to Symbian Installation API.

In this case, ThemeInstaller.exe will firstly decrypt the data files, then dump them into "C:\System\Cache\1\", and at last, silent-install the dumped file. (Latter dumped file will overwrite previous file, since they all have the same name: a1d54bc2.) When the installation is finished, the original data files will be deleted.

Package's relationship

Actually, ThemeInstaller.A's core includes three packages: "Ovi Update","Ovi Store" and "Ovi Store Installer". They work closely with each other in two aspects: anti-uninstallation and destruction of "Ovi Update" (See Figure 2).

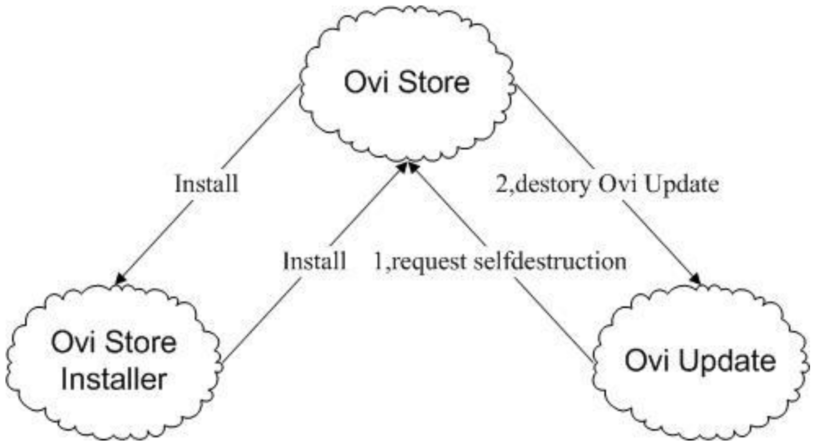


Figure 2 the interaction of ThemeInstaller.A's main packages

The first task is anti-uninstallation. "Ovi Store Installer" and "Ovi Store" are actually daemon packages. If one of them was uninstalled, the other one will help to reinstall. This realization is closely connected with these packages' structure. When the user uninstalls "Ovi Store Installer", RunAssistantProtect.exe will be launched. This executable will launch AssistantProtect.exe in package "Ovi Store". AssistantProtect.exe will handle the silent-installation of "Ovi Store Installer". This procedure is the same in reversed procedure, with RunAssistant.exe in "Ovi Store" and Assistant.exe in "Ovi Store Installer".

So, the actual effect will be this. First, the user removes "Ovi Store" or "Ovi Store Installer", and the system shows uninstallation successfully complete. But later, in system's application list, the uninstalled software magically appears, again! Why not remove both of them at the same time, someone may ask. Well, the hacker has already got precaution for that. Generally speaking, Symbian's installer is in charge of installing and uninstalling softwares. But, it always operates with a restriction: one operation at a time. That is to say, when the installation is in progress, and the user wants to uninstall software, he must wait until installation complete. So, when the uninstallation of "Ovi Store" (or "Ovi Store Installer") completes, the installer will be immediately occupied, which prevent the following uninstallation of other packages.

The second task is self-destruction. Actually, “Ovi Update” will detect phone’s idle state. If the detection shows user is beginning to using the phone, “Ovi Update” will destroy itself with the help of “Ovi Store”, as is shown in Figure3-2. First, OviStoreClient.exe in “Ovi Update” will launch DebugSrv.exe in “Ovi Store”. Then, DebugSrv.exe will silent-uninstall “Ovi Update”. This whole procedure is mainly realized in the assembly code below.

1, Launch DebugSrv.exe (in binary “OviStoreClient.exe)

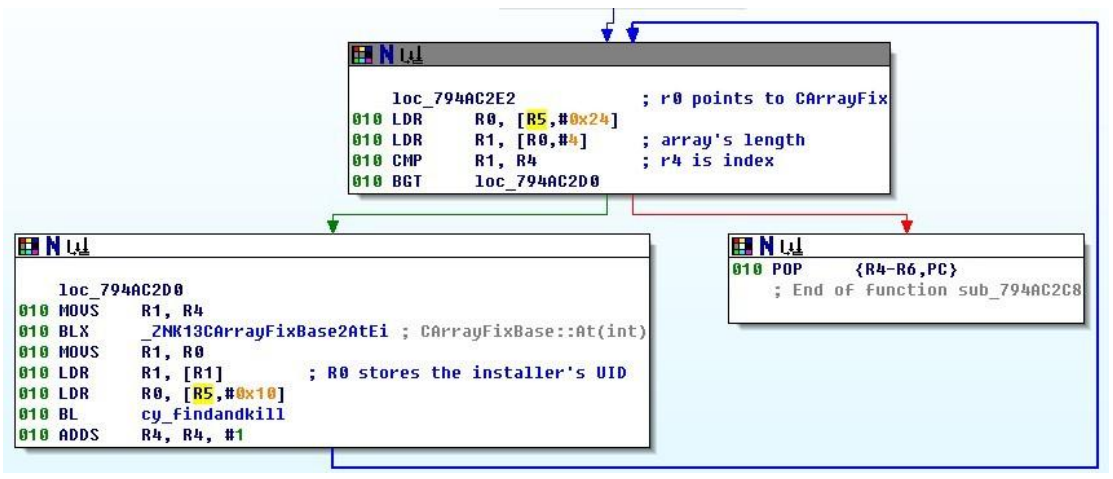
```
.text:792E1214        BLX    _ZNK13CArrayFixBase2AtEi      ; CArrayFixBase::At(int)
.text:792E1218        MOVS   R1, R0                ; R0 stores the name of “DebugSrv.exe”
.text:792E121A        MOVS   R3, #0
.text:792E121C        ADD    R0, SP, #0xA8+rprocess
.text:792E121E        ADD    R2, SP, #0xA8+tdes1
.text:792E1220        BLX    _ZN8RProcess6CreateERK7TDesC16S2_10TOwnerType
                      ; RProcess::Create(TDesC16  const&,TDesC16  const&,TOwnerType)
```

2, DebugSrv.exe stores system installer’s UID in an array.

```
.text:794AC112        BLX    _ZN13CArrayFixFlatI4TUidEC1Ei
                                        ;CArrayFixFlat<TUid>::CArrayFixFlat(int)
.text:794AC116        STR    R0, [R4, #0x24]
.text:794AC118        LDR    R0, =0x101F7295            ; one of system installer’s uid
……
.text:794AC120        BL     cy_AddtoArray
.text:794AC124        LDR    R0, =0x101F875A    ; one of system installer’s uid
……
.text:794AC12C        BL     cy_AddtoArray
```

Please notice, Symbian’s installer architecture includes two parts: UI and server. So, there are two UIDs, 0x101f7295 and 0x101f875a.

3, DebugSrv.exe’s logic to kill process.



```
    loc_794AC2E2               ; r0 points to CArrayFix
010 LDR     R0, [R5,#0x24]
010 LDR     R1, [R0,#4]        ; array's length
010 CMP     R1, R4             ; r4 is index
010 BGT     loc_794AC2D0
```

```
    loc_794AC2D0
010 MOVS    R1, R4
010 BLX     _ZNK13CArrayFixBase2AtEi ; CArrayFixBase::At(int)
010 MOVS    R1, R0
010 LDR     R1, [R1]           ; R0 stores the installer's UID
010 LDR     R0, [R5,#0x10]
010 BL      cy_findandkill
010 ADDS    R4, R4, #1
```

```
010 POP     {R4-R6,PC}
     ; End of function sub_794AC2C8
```

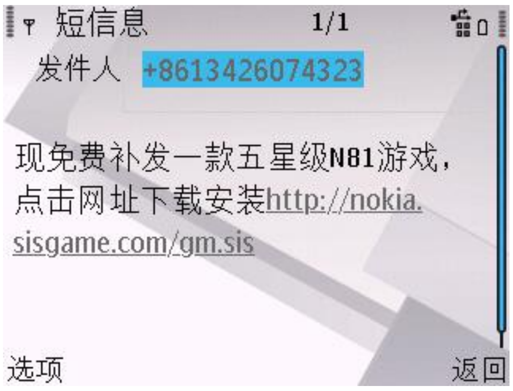
Figure 3 DebugSrv.exe's logic to kill process

The system's installer will be killed here. So, we can deduce that, the hacker also notices the unique restriction of Symbian's installer: one operation at a time.

4, DebugSrv.exe simply calls the silent uninstallation function to remove "Ovi Store". The UID of "Ovi Store" is hard-coded in its binary.

Send short messages

Sending short messages is ThemeInstaller.A's propagation method. The messages are sent with a link in the text. This link identifies software on remote server, see Figure 4. The message's content and recipient can be configured through remote server.



发件人 +8613426074323

现免费补发一款五星级N81游戏，点击网址下载安装http://nokia.sisgame.com/gm.sis

Figure 4 short messages sent by ThemeInstaller.A

Generally speaking, Symbian supports sending short messages in three levels. The upper level is easy to use, but hard to customize. The class SendAs, SendAppUi, etc are in this level. The middle level is MTM (message type modules), which is the most flexible way to operate messages. Currently, most softwares send messages via MTM framework. At the last level is, sending or receiving messages is actually operating on specific port of the phone. This is not widely used except some products with message-blocking feature.

Why does ThemeInstaller.A send messages at this level? It is because the port is a critical resource, thus cannot be occupied by two process at the same time. Currently, many security products occupy the same port to block messages. ThemeInstaller.A will firstly stop their process, and then occupies this port. The attacked software's blocking will be kept disabled, unless ThemeInstaller.A is recognized as a virus and killed.

Sending messages via socket includes four procedures: initialize the RSocket class, set and bind message address for a socket, create short message, and write the port to send message.

1, Initialize the RSocket class

```
.text:797E85FA   MOVS    R1, R4
.text:797E85FC   ADDS    R1, #0x34;
.text:797E85FE   MOVS    R2, #0x10    ; KSMSAddrFamily
.text:797E8600   MOVS    R0, R7              ; R0 stores address of RSocket's instance
.text:797E8602   MOVS    R3, #2              ; KSockDatagram
.text:797E8604   BLX    _ZN7RSocket4OpenER11RSocketServ;
```

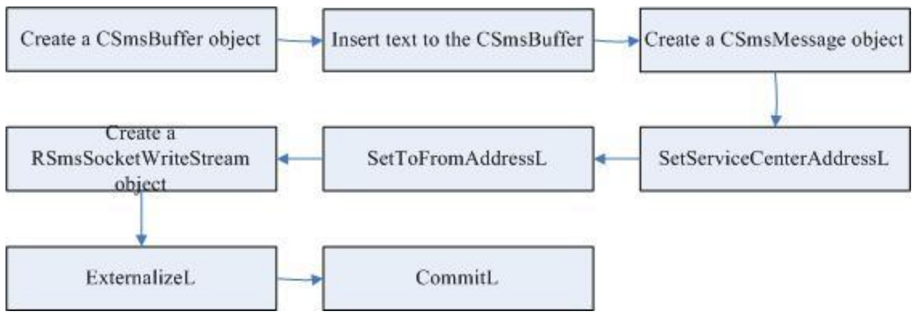
```
                                        ;RSocket::Open(RSocketServ &,uint,uint,uint)
```

2, Set and bind the message address for a socket

```
.text:797E860C   ADD     R0, SP, #0x198+cy_smsaddr;
                                        ; The address of TSmsAddr's instance
.text:797E860E   BLX     _ZN8TSmsAddrC1Ev; TSmsAddr::TSmsAddr(void)
.text:797E8612   MOVS    R1, #1                  ; ESmsAddrSendOnly
.text:797E8614   ADD     R0, SP, #0x198+cy_smsaddr
.text:797E8616   BLX     _ZN8TSmsAddr16SetSmsAddrFamilyE14TSmsAddrFamily
                                   ;TSmsAddr::SetSmsAddrFamily(TSmsAddrFamily)
.text:797E861A   MOVS    R0, R7          ; R0 stores the instance of RSocket
.text:797E861C   ADD     R1, SP, #0x198+cy_smsaddr
.text:797E861E   BLX   _ZN7RSocket4BindER9TSockAddr
                                   ;RSocket::Bind(TSockAddr &)
```

3, Create short message

This procedure is a little complicated, so the assembly code will not be provided. Instead, a flow chat is provided to illustrate these steps.



Create a CSmsBuffer object
Insert text to the CSmsBuffer
Create a CSmsMessage object
SetServiceCenterAddressL
SetToFromAddressL
Create a RSmsSocketWriteStream object
ExternalizeL
CommitL

Figure 5 the flow chart of creating short messages

The message text is set via CSmsBuffer and the recipient is set through SetToFromAddressL.

4, Write the port to send message

```
.text:797E86F4   ADDS    R2, R4, #4
          ; R4 points to an active object which is a member variable of CMainEntry
.text:797E86F6   LDR     R1, =0x306              ; KIoctlSendSmsMessage
.text:797E86F8   MOVS    R0, R7
.text:797E86FA   ADD     R3, SP, #0x198+addr_string
.text:797E86FC   BLX     _ZN7RSocket5IoctlEjR14TRequestStatusP5TDes8j
                                   ; RSocket::Ioctl(uint,TRequestStatus &,TDes8 *,uint)
```

Detect phone's idle state

OviStoreClient.exe and OviUpdate.exe will detect phone's idle state. The idle state mentioned here means whether the user is using the phone. Please separate two cases: the phone's used and the user is using the phone. For example, if the phone is playing music but without user's intervention, ThemeInstaller.A will continue conducting malicious behavior. As soon as the phone is picked by the user, this virus will stop and exit! (Don't worry about its stop, because it will restart automatically later. This feature will be discussed in section3.3.5.)

This trick includes two aspects, backlight and key lock. Almost all the Symbian phones have backlight, which can be turned on by pushes of the keyboard. If the keyboard isn't touched in certain time interval, Symbian will turn off the backlight to save power. Another aspect is key lock, which is a popular trick to avoid unconscious operations. For example, if the phone is in pocket, the bump of objects may activate certain operation on the phone. Similarly, the phone's keyboard will be locked if certain time eclipses (Its home screen should be on the foreground).

The backlight detection is implemented via class CHWRMLight. The virus initialize an instance of CHWRMLight using CHWRMLight::NewL(MHWRMLightObserver *). The argument is a callback function, which will be called if the backlight state has changed. This callback function offers two kind of information, the first is which part of the device has a changed event, the second is what event has happened (light on or light off). In this case, the virus mainly cares about the state of primary display of the device [10] . As to the keyboard lock, Symbian offers RAknKeyLock::IsKeyLockEnabled to check whether it has been locked.

Start automatically

ThemeInstaller.A doesn't choose daemon process as its protection method, because the effect is so easily to be noticed: process cannot be terminated. As an alternative, it uses Symbian's scheduler framework to achieve the same goal, which is stealthier.

Symbian has provided related interface: RScheduler. It is a client-side interface to the Task Scheduler, and can be used to scheduling a task running at regular interval of time. The following is ThemeInstaller.A's procedure.

1, connect to the task scheduler.

This is simply achieved by calling the Connect function of RScheduler.

2, register to the task scheduler.

```
.text:797E8C6E   BLX    _ZN10TBufBase16C1ERK7TDesC16
                                  ; TBufBase16::TBufBase16 (TDesC16 const&,int)
.text:797E8C72   MOVS    R1, R0; R1 stores a binary's full pathname.
.text:797E8C74   LDR     R0, [SP, #0x388+var_28]
                                       ;R0 stores instance of RScheduler
.text:797E8C76   MOVS    R2, #0
.text:797E8C78   BLX    _ZN10RScheduler8RegisterERK4TBufILi256EEi
```

[10] Other parts include primary keyboard of the device, secondary display of the device, secondary keyboard of the device, etc.

```
                                          ; RScheduler::Register(TBuf<256>  const&,int)
.text:797E8C7C   BLX    _ZN4User12LeaveIfErrorEi    ; User::LeaveIfError(int)
```

3, create a time based schedule i.e., information about the start and end time.

```
.text:797E8C82   BLX    _ZN7TTsTimeC1Ev          ; TTsTime::TTsTime(void)
.text:797E8C86   ADD    R0, SP, #0x388+ttime
.text:797E8C88   BLX    _ZN5TTime8HomeTimeEv      ; TTime::HomeTime(void)
……
.text:797E8C96   BLX    _ZNK5TTimeplE20TTimeIntervalMinutes
                                             ; TTime::operator+(TTimeIntervalMinutes)
……
.text:797E8C9E   BLX    _ZN7TTsTime12SetLocalTimeERK5TTime
                                             ; TTsTime::SetLocalTime(TTime  const&)
……
.text:797E8CB0   BLX
_ZN19TScheduleEntryInfo2C1ERK7TTsTime13TIntervalTypei20TTimeIntervalMinutes
                                       ;TScheduleEntryInfo2::TScheduleEntryInfo2(TTsTime
                                   const&,TIntervalType,int,TTimeIntervalMinutes)
  ……
 .text:797ECAD6 LDR    R1, =_ZN8CBufFlat4NewLEi       ; CBufFlat::NewL(int)
 .text:797ECAD8 ADDS    R2, #0xD
 .text:797ECADABLX     _ZN13CArrayFixBaseC1EPFP8CBufBaseiEii
                                    ; CArrayFixBase::CArrayFixBase(CBufBase * (*)(int),int,int)
  ……
  .text:797ECB74         BLX    _ZN13CArrayFixBase7InsertLEiPKv
                                   ; CArrayFixBase::InsertL(int,void  const*)
  ……
  .text:797E8CDC BLX           _ZN6TDes164CopyERK7TDesC16
                                             ; TDes16::Copy(TDesC16  const&)
 .text:797E8CE0  MOVS    R1, R4
 .text:797E8CE2  LDR     R2, [R4,#0xC]
 .text:797E8CE4  LDR     R0, [SP,#0x388+var_28]
 .text:797E8CE6  ADDS    R1, #0x10
 .text:797E8CE8
```

```
BLX_ZN10RScheduler24CreatePersistentScheduleER17TSchedulerItemRefRK13CArrayFixF
latI19TScheduleEntryInfo2E ; RScheduler::CreatePersistentSchedule(TSchedulerItemRef
&,CArrayFixFlat<TScheduleEntryInfo2>  const&)
```

4, schedule the task i.e., add this to the Schedule.

```
.text:797E8CFE BLX     _ZN10RScheduler12ScheduleTaskER9TTaskInfoR7HBufC16i
                        ; RScheduler::ScheduleTask(TTaskInfo &,HBufC16 &,int)
```

5, disconnect to the Task Scheduler.

Simply call the Close function of RScheduler.

Attack other softwares

OviStoreClient.exe and OviUpdate.exe will attack other softwares. The targets are usually security products in China. OviUpdate.exe attack in two aspects: silent-uninstallation and process killing, while OviStoreClient.exe only kills unwanted process. The information required to recognize the targets are provided in two sources: hard-coded in the binary or downloaded from the server. We will give analysis on the logic of its attack based on hard-coded data. Downloaded data will be provided in the Section 3.4.

The basic design of OviStoreClient.exe and OviUpdate.exe are the same, which is attacking others according to black or white list, so we will only give analysis of OviStoreClient.exe here. The attack of OviStoreClient.exe includes two procedures: initialize recognizing information and kill the target process. The code below shows that it inserts several predefined UIDs into an array. These UIDs are mainly ThemeInstaller.A's related UID, which can be thought as a white list.

```
.text:79AFCCE8  BLX     _ZN4User7AllocZLEi            ; User::AllocZL (int)
.text:79AFCCEC  MOVS    R1, #0xA     ; the size of CArrayFixFlat's object
.text:79AFCCEE  BLX     _ZN13CArrayFixFlatI4TUidEC1Ei
                                        ;CArrayFixFlat<TUid>::CArrayFixFlat(int)
.text:79AFCCF2  STR     R0, [R4, #0xC]
.text:79AFCCF4  LDR     R0, =0x20031C41              ; one of the UIDs
.text:79AFCCF6  STR     R0, [SP, #0x10+var_10]
.text:79AFCCF8  LDR     R0, [R4, #0xC]
.text:79AFCCFA  MOV     R1, SP
.text:79AFCCFC  BL      cy_AddtoArray          ;CArrayFixBase::InsertL(int,void  const*)
```

In this case, the final size of "white" list is 12. (The list's length is variable, because new data will be downloaded from remote server later.) When all data are prepared, OviStoreClient.exe can start its attacking logic.

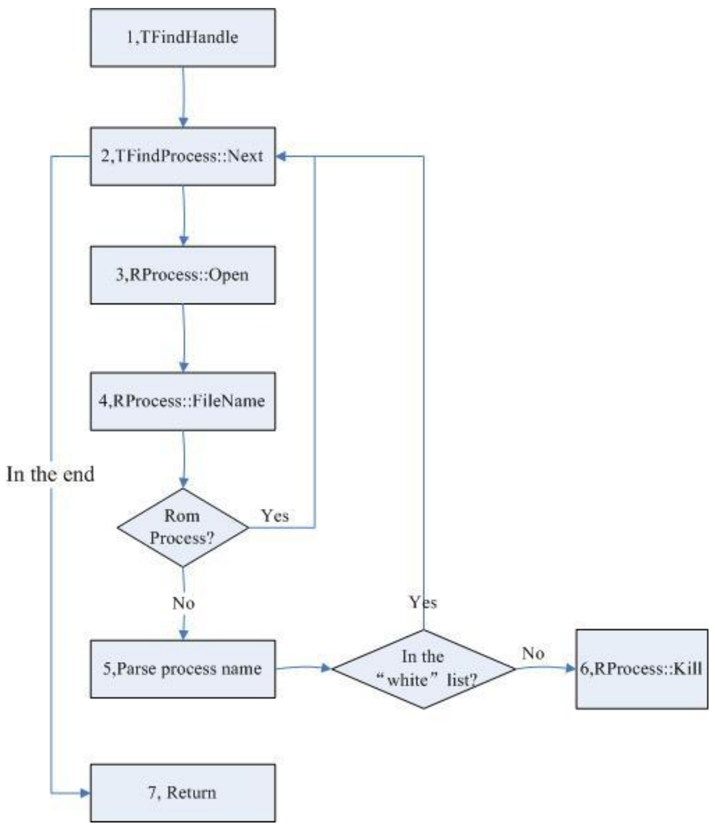


Figure 6 OviStoreClient.exe's attacking logic

From figure 6, we can clearly understand the attacking logic, but there are still a few details need to be specified. Firstly, not like Windows or Linux, Symbian's system executables are all burned into the phone's ROM (read only memory) chip. This ROM address is represented by letter "z", which is a unique drive in Symbian's file system. The system's executables actually run on ROM, so a practical way is getting the first letter of process' filename and check whether it's "z". Secondly, procedure 5 is actually a trick to get process related UID. On Symbian, the name of process is composed of three parts, filename, file's uid and instance's number. For example, the process name of OviStoreClient.exe is "OviStoreClient.exe [20031c41]0001".

Call to order services

Currently in China, there are many ways to order services from operator. Making calls are among them. First, user makes a call to the number of a service provider. Then, the provider offers options for user to choose. User pushes certain keys on the keyboard to interact with the remote provider. This procedure goes step by step, and finally the service is ordered. Obviously, this requires user's high involvement. Well, ThemeInstaller.A realizes an automatic and stealthy way to order services.

Actually, when user pushes keys during a call, a DTMF (Dual Tone Multi Frequency) tone will be generated and sent on the line. DTMF signal includes 16 coded identifications which corresponding to keys on the phone. The operator receives these DTMF tones and identifies the related number (key). In fact, certain services always correspond to fixed steps of key's push, which can be simulated to a sequence of numbers. (Besides, the time interval like user's operation time and the operator's voice prompt should be taken into account.)

Symbian provides class CTelephony to achieve these goals. The function DialNewCall is responsible to make calls to service provider and SendDTMFTones is used to simulate user's interaction. There are not complicated procedures, but idea of the hacker is noticeable.

Download malware

We have mentioned that there is a communication channel between "Zombie' and the server. The protocol between them is actually implemented in xml format, which will be provided in latter section. As to ThemeInstaller.A, its parsing capability cannot be extended, since the reversed result shows that all the xml's fields are hard coded in the binary. Currently, downloading new malware is its main method to extend protocol.

In this case, newly downloaded file will always be named "DB13DFD3.sis".The download procedure is simply implemented using HTTP protocol. (Axelle Apvrille, 2010) has explained the procedure of connecting to Internet stealthily so we will not discussed it here.

Botnets features

The word "Botnet" was firstly introduced from PC. Actually, "Botnet" is not the name for certain virus samples, but the floorboard for a structure with client side and server side. In this section, we will focus on this structure and reveal how the server controls these clients.

Related environment

Before the specific analysis, we will discuss the problem "Zombie" faces, and this can help to understand its botnets feature. As we know, botnets requires two sides: control server and "zombies". So, what's basic requirement for a botnets? First, the communication shouldn't be easily cut. Second, "zombie" should survive in complex situations. Third, to be botnets, spread method should be very effective. Ok, what is the actual environment to "Zombie"?

On the remote side: firstly, the communication channel is via GPRS (General Packet Radio Service) network, which is maintained by operators, such as China Mobile and China Unicom. Once the malicious server is spotted, these operators can block it, which will cut the channel between "zombie" and server. Secondly, short messages transmitted in the telecommunication network can also been blocked by operators. If the message's content doesn't change, the operator can be easily attracted to the statistic of such messages when the virus breaks out.

On the local side, the situation is more complex. Firstly, a method identifying each of the "zombie" should exist. Secondly, the phone types are various, for example, MMS (Multimedia Messaging Service) is not supported by all the phones. Finally, some phones may be equipped with security products, which at least, can block connections.

There are also many other problems. As the situation always changes, the protocol should be extensible. To each victim, the server has to learn its environment and private information as much as possible. Etc.

Protocol analysis

In our virus-analysis lab, we have captured their networking packages, but they are encrypted. [11]However, there are many ways to get this protocol's plaintext, such as remote debugging. During

[11] The encryption algorithm will be provided in the appendix.

the remote debugging, character set converting function and protocol parsing function should be added breakpoints. For example, ConvertFromUnicodeToUtf8L (TDesC16 const&) can convert Unicode to Utf8.Usually, the data should be converted to utf8 before sent out, and converted to Unicode after received.

ThemeInstaller.A has two versions of protocol, which are used by OviStoreClient.exe and OviUpdate.exe separately.

- Interaction between "OviStoreClient.exe" and the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<PostData>
<Task>3</Task>
<IMEI>359327035551369</IMEI>
<IMSI>460027016006646</IMSI>
<Edition>1</Edition>
</PostData>
```

This is a request of OviStoreClient.exe. As the plaintext shows, my phone's IMEI (International Mobile Equipment Identity) and IMSI (International Mobile Subscriber Identity) have been leaked to the server. IMEI can be used to identify the phone, while IMSI the subscriber. The combination of these two codes can uniquely locate one victim. The flag "Task" indicates what kind of service requested, and "Edition" shows the protocol's version.

```
<GetData>
<Task>3</Task>
<SafeTime>2</SafeTime>
<Type_Kill>
  <App uid="2002f8d2" Ename="Qh360Keeper_0x2002F8D2.exe"/>
</Type_Kill>
<TelTask Taskid="10005" number="12590649001" time="251">
  <DTMF value="1" time="12"/>
  <DTMF value="1" time="10"/>
  <DTMF value="2" time="13"/>
  <DTMF value="1" time="12"/>
  <DTMF value="1" time="10"/>
  <DTMF value="2" time="13"/>
  <DTMF value="1" time="12"/>
  <DTMF value="1" time="10"/>
  <DTMF value="2" time="13"/>
</TelTask>
```

```
</GetData>
```

This is the response from remote server. As the flag indicates, “Type_Kill” includes the binary name of a security product and it’s UID. This is a part of its attacking features, as we discussed before. The virus’s blacklist is kept updated, to defend itself against newly developed security product. “TelTask” is tricky, which confused us for a while. Actually, this part will be parsed and used to order services. The flag “number” indicates the number of service provider. The flag “DTMF” is actually the simulation of user’s selections. It has two properties, “value” and “time”. Flag “value” is simulation of keys on the keyboard, and “time” may be the latency required by service operator. So in this case, the virus will firstly call “12590649001”, and then “push” the keys one by one (“112112112”). The “SafeTime” indicates the latency before next connection.

- Interaction between “OviUpdate.exe” and the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
<Protocol>1.0.0</Protocol>
<Command>2</Command>
<IMEI>356044032022194</IMEI>
<IMSI>460027016006663</IMSI>
<SMSCenter>+8613800100500</SMSCenter>
<AllCalls>0</AllCalls>
<InstalledProductInfo>
<Product uid="E0000230" name="ActiveFile" />
<Product uid="200170BB" name="App TRK" />
<Product uid="EA1E2B6C" name="Log Example for Series 60 3rd" />
<Product uid="20030C77" name="Nokia Maps Plug" />
</InstalledProductInfo>
</Request>
```

This is a request of “OviUpdate.exe”. The flags “IMEI” and “IMSI” have been explained before. The flag “SMSCenter” is the number of message center on the phone. It will be required when user sends messages. The flag “InstalledProductInfo” contains the installed software’s information, such as software UID and software name. From this request, we know that at least these informations have been leaked.

```
<Reply>
<Protocol>1.0.0</Protocol>
<Command>2</Command>
<NextConInterval>9970</NextConInterval>
<MissionType>10</MissionType>
<SendSMSInfo id="1277630477863-356044032022194-1">
```

```
<SendSMSContent>现免费补发一款五星级 N81 游戏，点击网址下载安装
http://nokia.sisgame.com/gm.sis </SendSMSContent>
<SendSMSNumber>13500295087</SendSMSNumber>
<SendSMSNumber>13500297087</SendSMSNumber>
<SendSMSNumber>13500298155</SendSMSNumber>
</SendSMSInfo>
<ConnectProtect>
<ConnectProtectProduct>
<HandledProduct uid="2000AB0E" property="64578"
launchfile="NetQin_Anti_Virus_12345678.exe"/>
<HandledProduct uid="20028B0B" property="23793"/>
launchfile="NetQin_Communication_Mast.exe"/>
<HandledProduct uid="2002659F" property="57864"
launchfile="NetQin_PhoneGuard_PrivateStartup_0x20024FEB.exe"/>
</ConnectProtectProduct>
<JudgeProperty value="56496"/>
</ConnectProtect>
<ProxyList>
<Proxy url="http://zwe212.com/ms/MSServlet"/>
<Proxy url="http://98.126.64.130/ms/MSServlet"/>
<Proxy url="http://69.90.188.167/ms/MSServlet"/>
<Proxy url="http://69.90.188.169/ms/MSServlet"/>
<Proxy url="http://ddoay.com/ms/MSServlet"/>
</ProxyList>
</Reply>
```

This is a reduced version of server’s response (the redundant part has been omitted). Flag “ProxyList” includes the new addresses of malicious servers. These servers are back-up for each other. In other words, the operator has to block all the addresses of “Zombie” to stop the server’s control over “Zombie”. Flag “ConnectProtectProduct” serves as the data source of black list for “Zombie”. Here, the remote server response the related information of NetQin’s product. There is also a configuration of short messages. The flag “SendSMSContent” includes the text content of messages and “SendSMSNumber” includes the recipient’s numbers.

The framework of this threat

We have discussed the main features of “Zombie” in previous sections. Now, we will put these pieces together. The framework of this new type of threat is shown in figure 7.

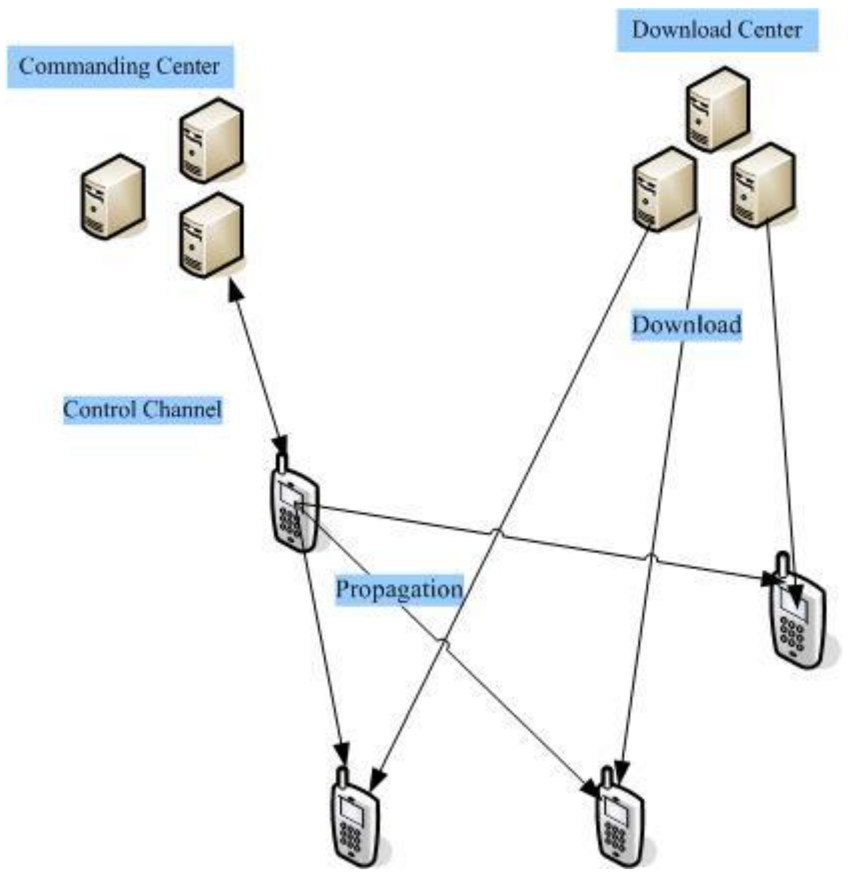


Figure 7 the framework of this threat

There are mainly four parts in this framework: commanding center, download center, the virus and the hacker.

Firstly, the hacker is the “boss”, the creator of this botnets. According to the investigation of police’s department, the hacker is not one single guy but several organizations. They make these viruses mainly for money. The profit comes in three ways: command the phone to send order messages to certain service providers; command the phone to make calls to order services; command the phone to send advertisement messages for certain companies. Besides, these hackers may also jeopardize phone user’s privacy. We have already known that they can upload phone’s IMEI, IMSI and the installation list. However, via installing new malware on the phone, they have the capability to steal more private information.

Secondly, the command center is the key to botnets. It helps to defend by updating new security software’s information, helps to avoid operator’s blocking by updating new malicious server’s address. It directly configures the scale of virus’ spread through short messages. The viruses will also download new malware from this server to extend the protocol. Besides, the server may accumulate huge amount of user’s private information, which may be utilized by hacker.

Thirdly, the download center is a relay of the infection path. Receipt of malicious messages doesn’t mean infection. People should click the link, download and install, to finish the whole infection path. So, there is also some social psychology applied in the attack of “Zombie”. But the download server may also be configured as a normal site, for advertisement of certain products.

Finally, the viruses are the leading role in this framework. They acquire many techniques to hide, protect and spread. As to ThemeInstaller.A, everything seems perfect except one thing: the protocol between commanding server and viruses aren't extensible. To extend the protocol, the alternative is downloading another malware automatically.

Conclusion

In this paper, we mainly discuss the features of "ThemeInstaller.A". However, the "Zombie" includes many variants, such as "NmapPlug.A", "NmapPlug.B"… "Dumusic.A"…,etc. Their protocol may be different, but the framework is the same. For example, the protocol of "Dumusic" can be directly recognized in the captured networking package. Their protocol is like: "04, http://uni.lyy.mobi/u.jsp?u=20912060;12, 20029080".In this case, "04" means an website should be added to the browser's bookmark; "12" means the software having this UID(0x20029080) should be uninstalled. This protocol is simpler, but can also be effective.

There are still some puzzles left about "ThemeInstaller.A". What's the whole set of its protocol? How does the server harvest phone numbers, since Symbian doesn't provide interface to retrieve phone number? [12]Besides, we also found functions such as hang-up calls, which haven't been activated yet.

Due to the involvement of remote server, these viruses' transmissibility and robustness have ascended to a new level. This framework is now a developing trend for mobile virus, not only on Symbian, but also on other platforms. On android, last November, we have captured the virus "Geinimi" which also has the characteristic of a botnets. Currently, the botnets is built merely for money. But, it can do more harm technically. In other words, botnets is just a framework, which can be easily added new malware. Just imagine, if the hacker downloads "Smspatch" and "Lanpackage" (NetQin, 2010c) to the victim, the damage will be more serious.

During the combat with "Zombie", we work closely with China Mobile and CNCERT, to help them block malicious servers and certain featured short messages. After the firstly week's crazy spread, the increasing speed is kept to a lower level.

[12] Actually, "Dumusic" will always connect to certain site via cmwap- an internet access point in China. The downloaded data includes the phone's number. But ThemeInstaller.A doesn't choose this way.

Appendix: The encryption algorithm

From previous analysis, we know the protocol between “ThemeInstaller.A” and remote server is encrypted. However, we can get the plaintext through remote debugging， without knowing its encryption algorithm. The encryption algorithm will be provided here, just for interested analyst.

1, Generate a key pool, which has 256 bytes and every byte is different.

2, Every byte in the original plaintext will be encrypted. First, get the value byte by byte, from the plaintext. Then, this value is used as an index to get the data in key pool, which is exactly the encrypted version.

3, The pseudo-code of the encryption algorithm is shown below.

```
TUint keyPool[64] =

{0xDDDC0A08,0x5C5BE4E3,0x5D636261,0x64605F5E,0x6C6B6A69,0x6766656D,0x75746E68,
0x706F7776, 0x78737271, 0x81807F7E, 0x7C7B7A79,0x8887827D,0x838B8A89, 0x8C868584,
0x95949392, 0xA2A1A08D, 0x1FA3A9A8,0x38331E21,0xE5E13A39, 0x1312DFDE,
0x26090100, 0x2D242322, 0x31302F2E,0x9F9E9D02,0x9B9A9997, 0xB071103, 0xC060504,
0x100F0E0D, 0x16201514,0x1A191817,0x271D1C1B, 0x59585756, 0x5A535251, 0x2825343B,
0xF22B2A29,0xFAF5F4F3,0x322CFFFE, 0x3C373635, 0x3E3D4241, 0x4443403F,
0x4D4C4645,0x4A494847,0x504F4E4B, 0x8F8E5554, 0x9C969190, 0xA6A5A498,
0xB0ABAAA7,0xACB3B2B1,0xB4AFAEAD, 0xB6BDBCB5, 0xBAB9B8B7, 0xC4BFBEBB,
0xC0C7C6C5,0xC8C3C2C1,0xD1D0CFC9, 0xCDCCCBCA, 0xD9D8D2CE, 0xD4D3DBDA,
0xE0D7D6D5,0xE7EFE6E2, 0xEEEDECEB,0xF0EAE9E8,0xF9F8F7F6,0xFDFCFBF};

HBufC8* EncryptL(const TDesC8& aData)

{

    TInt len = aData.Length();

    HBufC8* bufHeap = HBufC8::NewL(len);

    TPtr8 buf(bufHeap->Des());

    TUint8* pKeyPool = (TUint8*) keyPool;

    for (TInt i=0; i<len; i++)

    {

        TUint8 src = aData.AtC(i);

        TUint8 dst = pKeyPool[src];

        buf.Append(dst);

    }

    return bufHeap;

}
```

References

NetQin. (2010a). 2010 report for mobile security:

http://www.netqin.com/market/2010report/

CNCERT(2010). CNCERT’s alert for “Zombie”: http://www.cert.org.cn/articles/bulletin/common/2010091925129.shtml

NetQin.(2010b). Summarization of notorious viruses: http://www.netqin.com/security/securityinfo.jsp?id=3584&type=2

CCTV. (2010). CCTV’s report on “Zombie”: http://bugu.cntv.cn/news/talk/jiaodianfangtan/classpage/video/20101116/100907.shtml

Nokia. (2009). Nokia’s wiki about security mechanism:

http://wiki.forum.nokia.com/index.php/Platform_Security

Axelle Apvrille.(2010) Symbian worm Yxes: Towards mobile botnets? : EICAR 2010

SisContents (2010) Unpacking, editing and signing of Symbian SIS packages: http://cdtools.net/symbianandev/home.html

NetQin.(2010c). NetQin’s virus information center: http://virus.netqin.com/en/