Symbian Malware

OPL32 Infector Detection
Version 0.4
by Jimmy Shah

Note: No viral or malicious code was created during the research of this article.

OPL32 is a Bytecode compiled language for the Symbian OS. It provides system level access through built in functions and through extension DLLs called OPXs. Many shareware applications are written in OPL32.

The primary hardware that the Symbian OS ran on, used to be handheld computers from Psion. Psion is no longer in the consumer PDA/handheld business. Fortunately there is now a larger, expanding market in the form of smartphones from Nokia and Erricson. Unfortunately, a larger market is also something which is looked for by virus writers. Being connected to a huge pool of unaware users on a nearly global network is also helpful.

There is a low cost of entry for virus writers. The programming environment is either included with an installation of the OS or readily downloadable from the Symbian developer website. Relatively unskilled virus writers can take advantage of a number of tools to protect their code.

Infection types

Viruses add their entire body to the victim program in a number of ways. This can be at the top of the victim, at the bottom or mixed together with the victim. Regardless of what method is used, the virus will alter the victim program so that the virus code runs first. This is a description of how a simple virus operates. Most of the complexity in viruses comes from attempts to avoid detection.

Trojan horse programs are designed to look like legitimate programs but carry a destructive payload. They do not usually spread by themselves.

In order to detect infections it is important to understand the format of the file that will be infected. A description of the OPL32 file format is in the appendix.

OPL32 provides functions to open, read, and write any file on disk. Coupled with a file format description and an understanding of execution order virus creation is straightforward for the virus writer. Fortunately, these same functions allow one to detect and potentially repair the virus infected file.

Execution of a program starts at the first procedure in the procedure table. The procedure table contains a list of all the procedures in the file along with their offset within the file.

It is possible to add a procedure to the end of the file and replace the address for the first procedure with the viral procedure's address. To ensure the original file still runs a new entry could be added to the procedure table with the original address of the first procedure. The viral procedure would then contain a call to the original procedure.

This is the simplest method to infect a file. No attempt is made to disguise the viral code. Most virus writers would like to make it harder for their virus code to be discovered.

Virus code protection

Virus writers trying to avoid detection try to use "stealth" methods. Modifying the victim file will change the file's last modified date/time stamp. By using functions included in the System and Date OPXs it is possible to modify the victim file and then replace the old date and time. Hiding file size changes is more difficult as it would require altering behavior of the system shell program.

The LOADM command can be used to load a precompiled OPL32 module. Viruses can alter code within the victim's first procedure so that it becomes a LOADM call to their viral module. This method would have all infected programs using one common module.

None of this will protect someone from running an infected program through RevTran and seeing the unexpected addition. Legitimate developers have also wanted to protect their code from reverse engineering. A number of products have been released that are designed to either crash RevTran or make decompiled code meaningless. Virus writers can use these same tools to hide their code.

Virus Detection

There are two tools one should have in their virus detection toolbox:

- a hex editor
- Mike Rudin' s RevTran (Freeware)

RevTran

This program is one of the fastest ways to gather basic information on a suspected program. Basic info provided:

- Name of the first procedure
- Reverse translation of first procedure
- Source file name
- List of all the procedures in the file

The Source file name can be useful for discovering dropper programs for viruses. This is especially true if the source file is named something like "C:\MynewVirus\Haxor1". In the same fashion the name of the first procedure is useful.

The procedure list can also be helpful in detecting a viral procedure. A first procedure name that is in a different case from all the other procedures is suspicious.

Examining the code in the first procedure will usually reveal suspicious activity. Suspicious activity can be:

- searching for all OPL apps on the machine
- opening another file and copying the current procedure into it
- LOADM' ing a module named Haxor1.opo
- Printing a message on the screen like "You' ve been hit by Haxor1"

Of course if the virus writer has used anti-RevTran techniques it is necessary to use a hex editor.

Standard anti-RevTran techniques involve replacing all identifiers in the file with meaningless names or illegal characters. Mangling pointers at the end of unexecuted conditional statements is also used to confuse RevTran. There are also many techniques that utilize byte level access to create OPL illegal structures.

All these types of methods are designed to raise the cost of reverse engineering a mid to large size application. For an application with 50-100 procedures techniques such as these can push analysis time past all reasonable limits. Fortunately for the purpose of virus detection, except in pathological cases, one deals with one procedure or one small module.

Using a hex editor one can examine the procedure table directly. If the address of the first procedure comes after the address of the second procedure in the list, something is very wrong. The translator starts from the top of the source file and goes down. There is no pass that reorganizes the procedure table.

If the viral procedure is not reversible by RevTran it must be done manually. All OPL functions will never be altered by any protection scheme. OPL functions have a roughly one to one correspondence with their bytecode similar to assembly language. One can using a bytecode reference decode the layout of the virus procedure. Meaningless IF comparison blocks can be skipped.

References

OPL32 Programming

Symbian LTD., OPL reference database (version 1.0)

Tasker, Martin, Professional Symbian Programming

OPL Internals

Rudin, Mike Qcode.txt

Rudin, Mike RevTran 6.4

Feather, Clive D.W. OPO.FMT

Appendix: OPL File Format

WORD is 16 bytes.
LONG is 32 bytes.
Pascal strings start with a length byte followed by length number of characters.

Offset	Description
0	LONG UID 1
4	LONG UID 2
8	LONG UID 3
C	LONG Checksum of UID 1, UID 2, and UID 3
10	LONG Offset of second header
14	Pascal STRING Source Filename

|

Procedure Body (1)

|

Procedure Body (n)

|

Procedure Table

|

Second Header

Offset	Description
0	LONG &10000168 Stream ID?
4	WORD Required Translator Version
6	WORD Required Interpreter Version
8	Unidentified
C	LONG Offset of Procedure table
10	LONG Offset of included OPX list

Procedure table format

One table entry for each procedure in the program.

Table entry format

Offset	Description
0	Pascal STRING Procedure name (without terminating colon) LONG Offset of Procedure Body WORD Source line number minus 1

Procedure body format

Space control section

Offset	Description
0	WORD Data stack frame size
2	WORD Bytecode length
4	WORD Dynamic stack used

Parameter section

0	BYTE Number of parameters

Bitmapped type codes for each parameter.

Type code format

	Value
bits 0-6	0=Integer, 1=Long Integer, 2=Floating Point, 3=String
bit 7	1=Array, 0=Single variable

Global declaration section

Offset	Description
0	WORD Section size

One entry for each global variable in the procedure.

Global entry format

0	Pascal STRING Global variable name
	BYTE Type code as above WORD Offset within stack frame

Called procedure section

Offset Description

0 WORD Section size

One entry for each procedure called.

Called procedure format

0 Pascal STRING Procedure name
BYTE Number of arguments

Global references section

Lists globals referenced but not declared in procedure.
One entry for each reference.

Globals reference entry

Offset Description

0 Pascal STRING Global variable name
BYTE Type Code

Section terminated by zero BYTE.

String control section

One entry for each string.

String entry

Offset Description

0 WORD Data stack frame location
2 BYTE Maximum length of string

Section terminated by zero WORD.

Array control section

One entry for each array.

Array entry

Offset Description

0 WORD Data stack frame location
2 WORD Number of elements

Section terminated by zero WORD.