Shub-Nigurrath [ARTEAM]

A Symbian Symphony for 4 Crackmes and a Commercial Program

Version 1.0
Last Rev.: August 2007

Into this Tutorial

Forewords

This is my second issue on Symbian; the previous one [1] was a really big effort and contained a lot of issues to start reversing for Symbian S60. No 3rd edition stuffs still in my tutorials, due to the lack of good loaders from the scene, but that version of Symbian will completely substitute the old S60 not before few years, so there's still a lot of space for tutorials like this.

Moreover the techniques will remain even with new versions of Symbian OS.

This time thanks to argv (our new fellow) I will concentrate on the resolution of four crackmes he developed. These crackmes range from very simple to more complex, it will be the occasion to deeply look at some of the concepts I already underlined in the previous tutorial and the chance for you to train yourself on some new Symbian stuffs.

I also included into the distribution of this tutorial all the crackmes sis files.

I will conclude this tutorial with a small modification on a commercial application, well, actually a freeware application: FEExplorer.

Of course the understanding of the concepts already expressed in [1] is a pre-requisite.

The result of this tutorial is a symphony of patches, crackmes and "Notes", meant to improve your musical ear for Symbian reversing.. ♫♫

Have phun,
Shub

Author: Shub-Nigurrath

Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this document have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible for damages to the companies holding rights on those programs. The scope of this tutorial as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art and generally the comprehension of what happens under the hood. We are not releasing any cracked application. We are what we know..

Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: http://releases.accessroot.com

Table of Contents

1. Crackme 0x01

The job of this first crackme is very simple and consists in changing a string from a "Not Cracked :(" to "Cracked :)".

Solving the assignment will allow us to deeply analyze the structure of strings into Symbian programs and to better understand how to modify them.

Difficulty: 1 - Needs basic Symbian C++ knowledge and HEX Editing
Platform: Symbian OS [Series 60]
Language: Symbian C++

Note 1: The installation procedure for all these crackmes is always the same:

1. Install the crackme0x0*.sis distribution file on the phone.
2. Decompress the sis file on the PC to extract the *.app file or take it from the phone and load into IDA.
3. Be sure your IDA is equipped with the settings I already explained in [1] and with latest IDS files (you can find them in Symbian section of our forum).
4. Be sure to have a reliable method to transfer files on the phone directly into the installation folder (e.g. TotalCommander with SymbianFS extension).

1.1. Details and what we have to do

Figure 1 shows the original Crackme 0x01: when launched you can choose the menu on the left and the nags appears.

Figure 1 - Original Crackme 0x01

We will see in next sections that the voices of a menu and the corresponding functions of the program are handled into a Menu Handler function that identifies what each menu does (like for PC applications this function does the events-handlers mapping). One solution you would imagine is to erase the menu voice assigning to it a null function that simply does nothing. This option is generally speaking more complicated than one would expect because the menu handler is directly handled by the Avkon library; in next Crackme we will see a possible way to handle this situation. Generally speaking a not correct patch sends the application in Panic mode; the result is a Panic dialog and the exit from the program (this happens usually due to stack or registry corruptions)

Note 2: All the crackme and patches discussed here have been tested on Nokia phones 6600, 6630 and N70.

1.2. Approaching the enemy

Before going on it is required to view a little of theory on strings (called Descriptors) and Symbian programs.
From the book [2], "Developing Series 60 Applications - A Guide for Symbian OS C++ Developers" in the Chapter 3, in the section about "Descriptors" you can find:

All of the descriptors have an iLength member that stores the current length of the descriptor. This is how descriptors can function without the need for null termination. Modifiable API descriptors also have an iMaxLength member, since their length needs to be bounded

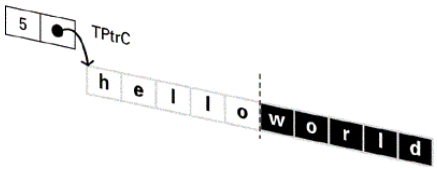


Figure 2 - General structure of Descriptors (TPtrC)

Figure 2 shows the concept and Figure 3 the hierarchy of all the possible Descriptors you may find, all inherits their properties from the base class TDesC, TPtrC represents a special type of descriptors, called Pointer Descriptors.

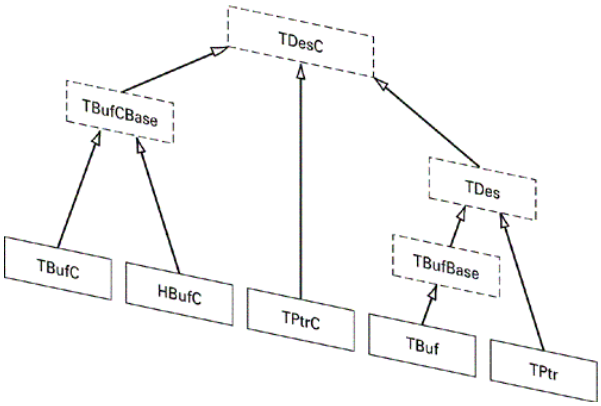


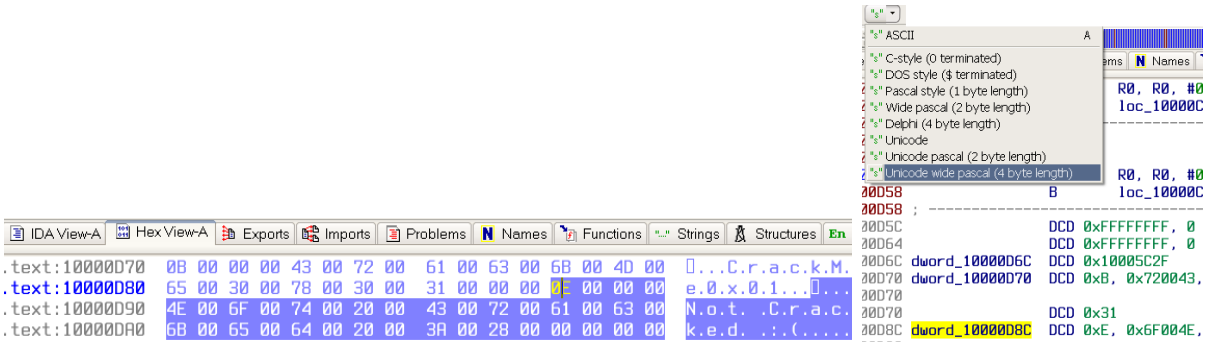
Figure 3 - Descriptor hierarchy

Now it's time to see using IDA how is stored the string we want to change. IDA doesn't recognize it at a first sight.

```
.text:10000D6C dword_10000D6C  DCD 0x10005C2F          ; DATA XREF: .text:off_10000054↑o
.text:10000D70 dword_10000D70  DCD 0xB, 0x720043, 0x630061, 0x4D006B, 0x300065, 0x300078
.text:10000D70                                         ; DATA XREF: .text:off_10000D04↑o
.text:10000D70                 DCD 0x31
.text:10000D8C dword_10000D8C  DCD 0xE, 0x6F004E, 0x200074, 0x720043, 0x630061, 0x65006B
.text:10000D8C                                         ; DATA XREF: sub_10000100:off_10000220↑o
```

This is due to the special strings structure I introduced before. IDA by default assumes the strings are stored in C-style (terminated by a 0 value); unless you specify that the default format is another one. You have then to use the IDA's Hex view and find where are the strings we want, and then explicitly define using the string definition button, and specifying the "Unicode Wide Pascal (4 bytes)" string format (in the hex view the real string boundaries are not correctly identified as you can see).

Note 3: All current Symbian OS smartphones are based on the ARM processor, which has two instruction sets: a 32-bit set (known as ARM) and a 16-bit set (called THUMB). Code compiled to one set can interoperate with the other. The ARM instruction set is fast but uses more memory per instruction (RISC); THUMB is more compact but slower, that is more instructions are required to perform the same work (semi RISC).
Usually programs are compiled using one of the instruction sets, and IDA allows switching from one mode to the other, for this tutorial I always used programs compiled using the ARM instruction set.



The string is then identified and correctly labeled as aNotCracked. Now you can see the structure I told before: 14d is the length of the string (equal to 0Eh). If you see the same portion of program in the Hex View you will see a value equal to 0Eh.

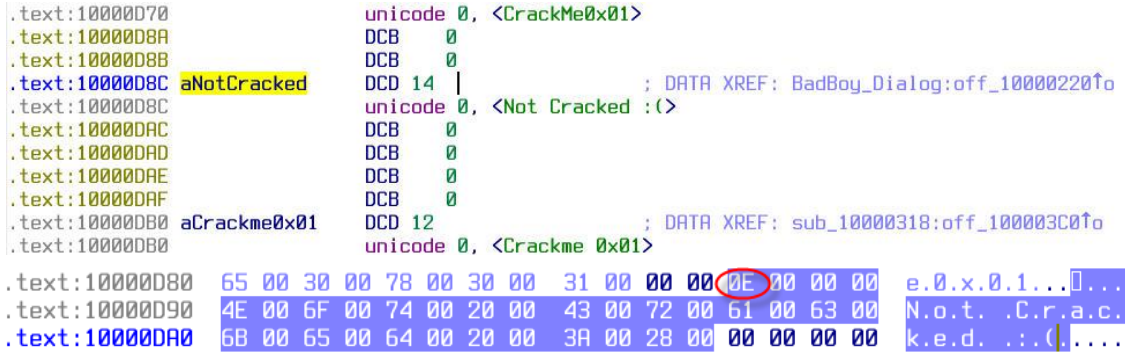


Figure 4 - location of string aNotCracked

Now press X to find all the references to this string and start climbing the rope..

The string is not directly accessed by the program but through an offset which contains the references to the real string. This offset (off_10000220) acts like a real proxy giving the real code a fixed entry point to the string, allowing the compiler or the program loader of the phone to relocate wherever it wants the strings but leaving the code with fixed references: actually the only reference to a string is indeed a reference to an offset which contains the actual address of the string.
If you go in the hex view of the address 10000220h you can see that the opcode of this "instruction" is: 8C 0D 00 10, this is the address (in LSB format) of the address where the string actually is located: 10000D8C, see Figure 4.
Pressing again X on the label off_10000220 you land where the string is actually used, see Figure 5.

Note 4: A panic occurs on any error that is not recoverable, at which time the thread exits immediately and the system displays a popup with information regarding the error (the SDK documentation contains a list of the system panics). In general, a panic occurs as a result of a programming error of some kind. An example is if you use an API improperly. The end result of a panic is either a reboot (Blue Screen of Death on the phone too) or an ''Application closed'' message box.
You can invoke a panic in your code in response to an error you detect by calling:

```
User::Panic(const TDes& aCategory, TInt aReason);
```

On Series 60, when a panic occurs a box that simply says 'Program Closed' is displayed. To cause the full panic information to appear you need to create a dummy file (which can be empty) in `\system\bootdata\errd` on the target system's `C:` drive. This works for both the emulator and the smartphone [4]

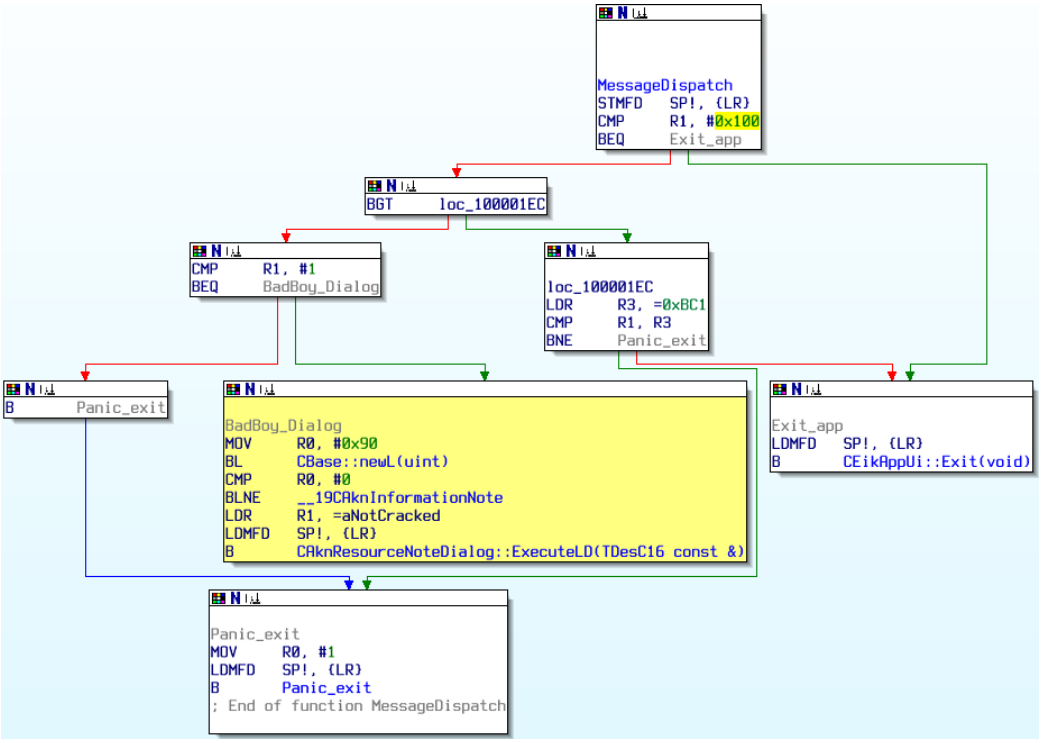


Figure 5 - Crackme 0x01 BadBoy_Dialog

I renamed it (pressing N) with the name MessageDispatch. As you can see, the function is a small switch-case, where the R1 value is compared to choose the correct action:

- R1=0x100 or R1=0xBC1 → CEikAppUi::Exit()
- R1=0x1 → Create the aNotCracked dialog
- All the other values of R1 → Panic_exit function

This is the messages dispatch function I mentioned before, the function that choose which action is performed for which menu. I also renamed the function sub_10000CF8 as "Panic_exit" and the labels to improve readability.

1.3. Solving the problem

There are two ways to solve the problem: changing the string into something different and adjusting to the new length or work on offsets.

1.3.1 First method: changing the string.

We change the string directly:

"Not Cracked :(" → length=14 -> 0x0E
"Cracked :)" → length=10 -> 0x0A

The patch is then the following one:

```
00000E00|7800 3000 3100 0000 0A00 0000 4300 7200 6100 6300 6B00 6500 6400 2000 3A00 2900 6400 2000 3A00 2800 0000 0000 0C00 0000 4300 7200 6100 6300|x.0.1.......C.r.a.c.k.e.d. .:.).d. .:.(
```

I changed the length and the string characters, leaving those extra, because are not seen by the application (they are seen as garbage).