Subject: Re: Protect package - A starting copy protection system
Posted by mdelfede on Mon, 20 Sep 2010 08:02:06 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 20 September 2010 00:50Hi Max!

This is a very interesting package. I couldn't resist and played with it for a while My conclusions: It works very well and it would definitely stop me from stealing the app.

Ehehehe.... no, it's not too difficult to defeat, IF you've the key

Quote:
Few observations:
.....
You can't declare variables inside the encrypted block, because 'jump to label __end crosses initialization of ...'. It is somewhat cryptic, so it should be probably mentioned in docs.

mhhhh... I haven't tried it enough, but I guess it can be solved
somehow.... I couldn't just resist to put on bazaar for testing
The main problem I found is that damned M$ inline assembler, I guess it was coded by some drunk people....

Quote:
Work around is to put another pair of '{}' between the calls to PROTECT_... macros to limit the scope of variables declared inside.

I'll look at this solution this night.... Maybe it's a good suggestion

Quote:
You can't have two encrypted blocks in one function, as it results into redeclaring variables. Even if the blocks are in different scopes, it fails on duplicate labels. This could be fixed easily, but it is probably not important for real-life usage.

Well, the macros are thought for single usage in every function. I guess they could be modified for multiple usages (maybe using the __LINE__ macro for labels, or something like that, but I guess it's better to use it just once per function.

Quote:
The 'return' in PROTECT_END_FUNC prevents using the macro in functions returning a value. Omitting it causes runtime error,

Well, that's a bigger problem.... I didn't think about it.
The ending return statement is just to avoid entering into data (garbage) part of the code.... But can be solved also with an assembler jmp, I guess. I'll try it this night too

Quote:
One question at the end: Do I understand it right, that the decryption is performed only on first call of the function? So it modifies only the program loaded in memory? If so, I'll seriously consider

calling it a voodoo

Yep, it's decrypting on the fly on first function call.... and that's the biggest flaw of the approach. With a good placed breakpoint you can have the decrypted code in memory, and looking for the call to decrypt routine is quite easy with a good debugger. You must, indeed, have the key handy to do that, without key you can't do anything.
I was thinking about obfuscating a bit more the stuff, I've just to think a bit about it. The "big" problem is to keep the process simple and portable between GCC and MSC; to do a better think we should parse executable headers which isn't easy.

Anyways, as you can see from macro code, the whole process isn't a big voodoo The macro just marks the executable with some known strings, easily found by encrypter code, which patch the executable on right places. The decrypter has just to open code memory for write access and reverse the process. It's easy if you use encrypt algorithm that doesn't change code size.
A better (and more secure) approach would be to encrypt/pack the code and to decrypt/unpack in memory allocated for the purpose.... But, I guess then ww would have do parse executables, object files and so on, which makes it a nightmare for portability sakes

Quote:
Good job!
Honza

Thank you

Ciao

Max