Ghiribizzo's Cracking Tutorial

Improving the Standard Serial Protection Scheme

Improvements

I will present a simple improvement over the standard serial number protection scheme that every programmer should use if they already employ a serial number type of protection. For a little extra work, you end up with a much more user friendly protection.

PGP and Signed Tutorials

My tutorials and programs should be signed electronically using PGP. PGP 5 supports DSS/Diffie-Hellman keys. These keys are not supported by previous versions of PGP.
You should check the signature to make sure that the tutorial and especially its program files have not been tampered with. All cracks, tutorials and zip files I release will be signed. This will prevent tampering and will hopefully reduce the chances of viral infection.
My signature will also be the only way you can identify me as my email address will often change.

My Web Site:	http://Ghiribizzo.home.ml.org
My Email:	Ghiribizzo@geocities.com
My Backup Email:	Ghiribizzo@hotmail.com

Introduction

I read Aesculapius' strainer for the 1999 HCU entrance and Terminate caught my eye. Aesculapius mentions that the old version 4 serial numbers would work for version 5 but fake version 4 serial numbers would not.

This seems such an obvious improvement to the standard serial scheme I am surprised it has not been implemented more frequently. Some shareware programmers are notorious for their frequent changes to the serial scheme (are you reading this author of UltraEdit?) and other programs have not altered their serial scheme for a long time. Let's look at some advantages and disadvantages of the improved serial scheme.

Advantages:

Less hassle for the end user:
New keys can be demanded for each new version.
No need to send out new keys to each registered user when a new version is finished.
Useful lifetime of 'serial lists' will be much reduced.

Disadvantages:

A little extra work must be done when creating the initial serial scheme.
Doesn't make the actual scheme any more difficult to crack.

As you can see, the advantages far outweigh the disadvantages and there are some serious savings to be made both in time and money. So how do we go about implementing such a scheme? I'm sure most of the good programmers/protectors will already have many devious ideas going through their minds at this point. I'll illustrate a simple system for the brain dead authors.

A Crude Implementation

The system basically relies on the serial number holding information which the cracker can't get. Now, in a standard serial scheme, the cracker can examine the code used to check the serial and reverse the scheme to produce a serial which possesses the required characteristics. A crude form of the improved scheme could be to build a second serial scheme into the protection and simply concatenate the serial numbers.

e.g.

The standard scheme has an 6 byte serial XXX-XXX
We create a different scheme again with an 6 byte serial YYY-YYY
So for our improved scheme the serial would be XXX-XXX-YYY-YYY
Now this may seem no more secure. The trick is to NOT check the second serial number. This means that there is NO WAY that the cracker can know how the second serial number is derived. The check is only made in the next version of the program.

So we release version 1 of our program and a cracker cracks the protection, possibly making a key generator for the first 6 bytes of the key. The cracker possibly knows that the serial needs to be 12 bytes long, but the last 6 bytes don't seem to be checked and there's nothing more that he can do.

Then we decide we need more money so release version 1.2 of our program that fixes a few bugs and has some new features. We have already found that our program has been cracked and that key generators are freely available on the net. Worse still, various serial numbers have appeared in the huge 'serialz lists'. Do we completely rewrite our protection system and contact each registered user (by email if we are lucky, or by snail mail if we are not) and waste time and money? Do we moronically hardwire every single warez serial number into our new version? No! Of course not. We simply add the code which we had prepared all along which checks the last 6 digits of our serial scheme. We can release our new version on the web and registered users can simply install over the previous version and the program will find and recognise the key. The crackers will need to crack it again (admittedly this probably will be quite easy to do) and the lamers will need to wait for the next keygens and serial lists to be made.

Hmm. There are some extra features we want to add. Perhaps we should have made the serial number 18 bytes long…

So it wasn't the best way to implement the improved protection scheme, but I'm sure all shareware authors will understand why this is so much better. Concatenation is a crude way of doing it. As long as you let the serial have some hidden property that is only checked in later versions you can apply the same principles. We can estimate the number of revisions we intend to make and include enough special key properties to hopefully last the lifetime of the program. Some ideas for properties: serial number must be odd, 6^{th} digit must be a 9 etc. etc. use your imagination.

I was going to write a demonstration program, but the principle is so simple, I'm sure that you all understand it. If not, write to me and I'll work on one if I have the time. Better still, you write one and send it to me for inclusion!

Well, I hope some shareware authors are reading this and we begin to see some 2^{nd} generation serial schemes being developed.