Ghiribizzo's Cracking Tutorial

How to Protect Better : A Strategy

How to Protect Better

Becoming a good cracker requires a holistic approach. I have always recommended learning programming (esp. assembly) and virus writing as a complement to cracking. It is also fitting that we learn how to protect better in order that we can crack better.

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://www.geocities.com/Athens/3407
My Email:	Ghiribizzo@geocities.com
My Backup Email:	Ghiribizzo@hotmail.com

As a programmer, I have always resigned myself to the fact that no matter how good protection is, it can ALWAYS be defeated. The only sure way of securing a demo of a commercial product is to release the demo WITHOUT the code for the 'registered' features. There should be no mechanism for registering using name/serial as these can always be broken (as can ANY form of protection which re-enables features) and name/serial pairs are rife on the internet.

Leaving aside the search for a perfect system. Let's consider a program which has been 'crippled' and can be reactivated in some manner. As the program CAN be activated, it can also be cracked. The protection should therefore not aim to make the software uncrackable, but to make it as difficult and nasty as possible for the cracker. Now let us consider strategy. How do crackers work? 'Dead Listing' and 'Live Debugging' are the main techniques and it is on these two techniques that the protection will focus on. The protection must focus on both, as the cracker can use both to crack the program.

Remember: as well as making the program difficult to crack, we want to make it nasty to crack. To do this we want to crack the cracker. We want the crackers life to be a living hell as he tries to crack the program. We want him to hate cracking it so much that he will give up. I don't think this form of protection has been considered by others before. So bearing this in mind, let's see what we can do with the two techniques.

Live Debugging

An important part of cracking is to locate the protection code. This makes it a target for our 'psychological protection' as finding the code can be considered a 'chore' for crackers - the good fun being trying to circumvent the code and find creative ways of doing it - it also becomes a good point to make the cracker give up. After all, finding a way *around* a piece of code is a mental challenge (an we never give up on those, right?) but finding the code is always less fun. If we let a cracker step though our code at will and quickly locate the relevant sections, then this chore will be over quickly.

How can we stop this? Liberally scatter anti-debugger tricks throughout the code. Not just at the beginning (but make sure there is a lot at the beginning just to scare off the newbies and also to quickly demoralise the cracker) but *deep* inside the code - as deep as possible. Don't let the cracker step over a call, make him painfully trace though every single one by putting in code which me *must* crack to continue. If he doesn't punish him. You can warn him a few times first by deleting the registry etc. But if he doesn't take the hint, *reformat the hard disk!* That should get his attention. Even a corny

```
call protection routine
cmp  al,00
jne  beggar off section
call another routine
```

We come across this stupid protection again and again. Easy, we've run the beggar off section during a dry run, so now let's skip over it and go straight to the 'good guy' call. Oops. In fact, this was a fake protection check. The cracker has just inadvertently run a call to a virus (learn to write viruses - you'll learn tight efficient assembly and your own viruses, if well written, will not be able to be detected by current virus scanners - of course you cannot put this in commercial programs as it is illegal).

Don't use standard DOS interrupts or API calls. Write your own custom code. This stops crackers from just breaking on them. Which reminds me, remember to detect SoftICE being loaded in memory and do something about it!

Now that the cracker knows to tread lightly, we make his task seem dauntingly impossible. For those of us who have carefully written an assembly program... They just don't lend themselves to protection the code is too small and can easily be searched though. You need to junk the code. Write it in a high level language, write it for Windows 95 bloat the code so that the cracker must trace for hours. It will also make dead listing difficult as the cracker will need to examine hundreds of pages.

Now that all this preparation has been done, bury the protection routine deep in the code at several places - make them inline so that a single call can't be found and references targeted. *Protect the protection!* Look for the weak spots - a JNE which can be changed to a JMP. Target the opcodes and do a compare elsewhere to check for this tampering. Place this second check *after* the protection and he'll know it at debug time. But place it *before* and then when he patches it... bye-bye hard disk. Let's hope he checks his work before distributing the crack. This leads onto my next point (thinking about this I must once again advise you all to learn to program and write viruses as many tricks can be taken from viruses) put a random counter in the protection's protection. Make it so that the hard disk format occurs on a 1 in 8 chance. Place the protection before so that it is not caught at debug time. When the patch is made, the cracker will not know until hundreds of lamers write to complain... (these are nasty techniques - as a cracker, I am cringing at the thought of them - let's just hope this stays among crackers ;-)

Dead Listing

Now that you've made the 'Live' technique difficult, let's do the same for dead listing. Make sure you've junked the code to make the dead listing huge. Don't be stupid and have strings like 'invalid registration number' hanging around. In fact have nothing linked to the protection hanging around like that.

Spread the protection throughout the code don't have it all in one call etc. Piecing single instructions spread over 10s of pages is difficult.

Encrypt the code. Using xor is OK but use other techniques for variety, you don't want the encryption to be pinned down on searching for xor instructions (again spreading the decryption will camouflage it). Encrypt the code and data in different sections using different encryption. This means the defeat of one encryption does not give the code. So once you've encrypted the whole of your code, in 10 sections, say, with 10 different 'algorithms', the cracker must decrypt them at 10 different points - most of them buried deep into different parts of the code which will give him the problems of the 'Live' technique. You can also encrypt the whole lot once more near the beginning using all the tricks and peppering the decryption routine with anti-debug code. Use all the tricks to make decrypting difficult (learn assembly - you come up with some really cruel ones! It must be where I get the ideas for this document too!) and remember - punish the cracker by damaging his computer.

Summary

My strategy should be clear: make the 'Live' approach dangerous and difficult and make 'dead listing' impossible to force the cracker to use the 'Live' approach. I have introduced the idea of attacking the cracker as a means of protection (the best defence...). I may write a program to demonstrate some of my ideas. Any takers?

As an aside, writing your own packer can be an effective way to protect your code. But even as I write this, I hear that there is an unpacker which uses a heuristic technique to defeat unknown packers. I have not studied this unpacker yet - I believe it is called Tron - but I'm sure that should it be relied upon, then measures can be taken to actively defeat it (much as there are many tricks against SoftICE).

I hope that this has given you some food for thought. Keep on cracking... only now, be a bit more careful! ;-)