Ghiribizzo's Cracking Tutorial

Walk-Through 1 : Cracking PSedit

The Walk-Through Series

There seems to be a growing number of cracking tutorials out on the net though only a few give a complete 'walk-through' which can be of great help to the beginner. The 'walk-though' series will aim to fill this gap and hopefully help to create the next generation of crackers.

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. Remember that Word files can contain macro viruses.
My signature will also be the only way you can identify me as my email address will often change.
The tutorials will hopefully be released in Word 7, PDF and Text formats. The format of future tutorials will depend upon the feedback I get. So write to me and tell me what you would like.

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

Cracking PSedit

I’ve chosen PSedit for a few reasons:

- It’s shareware and so is freely distributable
- It’s quite easy to crack
- It’s a good example of a ‘nag screen’
- You end up with a program you will find useful for future cracking

As this is the first cracking tutorial I am writing, I will briefly go over some of the basics.

The Debugger

The tool you will be using to crack is the debugger. There are other ways to crack but the easiest way for the beginner is the ‘live’ approach. Basically you will debug the program step by step. Executing the instuctions one at a time until you find the one which gives an undesirable effect (e.g. nag screens, password request etc.) and then finding a way around it.

There are several debuggers available for your use:

- Debug/Symdeb - debug comes with some if not all versions of MSDOS. However, it is not full screen and can therefore be daunting for the beginner to use. However it is still a very useful tool and can be used in a variety of different ways and has many uses. It is also very small i.e. takes up very little memory - a property which can be very handy…
- Freeware Full-Screen Debuggers - address one of the shortcomings of debug. I’ve tried a few but they are sometimes unreliable and there are better debuggers anyway.
- Turbo Debug (TD) - a favourite among many crackers. Simple and quick. Is useful for easy cracks that have little or no ‘tricks’ in them. Note there is another version of TD for 32bit programs called TD32. TD comes with a lot of the software development packages from Borland and can also be found on the internet for free.
- SoftIce 2.80 for Dos - An excellent DOS debugger from NuMega. This is used and loved by many crackers and can step through some anti-debug tricks as if they weren’t there. The downside is that you will need to create a boot disk to load the program as it often cannot be loaded from a Dos prompt. Also SoftIce has become so popular among crackers that many programmers have written code in their programs to detect if SoftIce is loaded so beware!
- SoftIce for Windows - A version of SoftIce for windows. Above comments apply.
- SoftIce for 95 - A windows 95 version. The latest version also contains many goodies including two monitor support which is a godsend.
- SoftIce for blah blah blah - Rather than go on and on, I’ll just mention that SoftIce is also available for Windows NT and I think also for OS/2 check NuMega’s website. They also occasionally dish out free- ahem… *trial* copies of their products.

The Unpacker

Sometimes executables are ‘packed’ or compressed to save disk space using programs like EXEPACK, LZEXE and PKlite. As well as compressing the program, it also messes up the code making cracking attempt more difficult (and for the beginner, often impossible). The solution is to unpack it with one of several programs available to us. Try doing an Archie search for these programs. I will recommend UNP which can unpack all of the above mentioned packing programs. UNP can be found using Archie and also on my website at: http://www.geocities.com/Athens/3407.

The HexEditor

You should all know what a hexeditor is and I'm not about to explain it. A hexeditor is required to alter the program once you've discovered how to bypass the protection/nag code. In this tutorial you will crack PSedit which is a hexeditor and so you can use PSedit to patch itself! Later on you may get confused as you will actually run the program and bypass the nag screen without using the hexeditor. This is because the method I teach you will involve modifying the code in memory and running the 'memory-patched' version. You want to be able to alter the program which resides on your hard disk so that the patch is always there. The hexeditor allows you to do this. If you don't understand this yet, don't worry because it will become clear later on.

OK, Let's Crack it

Right. Let's copy the file we want to crack into a temporary cracking directory. I've renamed mine p.exe to save me some typing. Now check the file; is it packed? Just run UNP on it, if it's packed then UNP will probably be able to deal with it. Occasionally, files can be packed a multiple of times so you can try running UNP once more if it is packed.

Before we dive into cracking. Let's take a look first at the program we want to crack. Run it. It gives you a file screen to choose the file to hexedit, then it prompts you to backup your file and then displays the nag screen. Run it again, this time specify a file on the command line. The same thing happens only this time you skip the file screen. Always experiment with your target program before you start your debugger, it will save you time later. Armed with this knowledge we can now continue.

Now you can load your debugger. I will use Turbo Debug, if you're using SoftIce then you should already have it loaded. You can load PSedit for debugging by typing "LDR PSEDIT.EXE" I don't think you need to include the '.EXE' (I'm not sure on this, I haven't used SoftIce for ages!) and you should have 'LDR.EXE' in your path.

You should now be faced with a screen full of numbers. Don't panic. With Turbo Debug, you should have a code window in the top left corner of the screen, a hexdump window below that and on the right hand side you should have the status of the flags, registers and stack. With SoftIce for Dos, you may not have much on your screen at the moment depending on how your configuration file is set. If I remember correctly, typing WC will toggle the code window on and off you can also specify the number of lines the code window will take by adding the number of lines after WC, e.g. WC 20. Other commands are WR for registers, WD for data etc. You can always use help and check your documentation.

Now we will go through debugging using two basic commands:

- Step - This will execute a program instruction. e.g. if at a call, stepping over the call will run it and wait for you to step again over the next instruction - hopefully, you can lose control, more on this later.
- Trace - You will trace though program instuctions. This is identical to step except that for some instructions such as calls, instead of running the entire call, it will run the first instruction in the call and you will 'arrive inside' the call.

Tracing can be done by pressing F7 in TD and I think it is the same in SoftIce. You can also type '[P][ENTER]' in SoftIce. Stepping can be done by pressing F8 in TD and F10 in SoftIce. You can also type '[T][ENTER]' in SoftIce. I would reconfigure SoftIce to use F8 for program step instead of F10. This will mean your fingers will need to move less and you can interchange between the two debuggers with less hassle.

When you execute certain programs you will occasionally see the screen being updated, however, the debugger will switch back to the debugging screen. To see what is going on with the program press '[R][S][ENTER]' (or F4 on my version) with SoftIce and 'alt-F5' with TD. Now we are ready to begin.

From what we have learned with our initial experimentation, it would be easier to load the program with the file on the command line parameter as this will avoid having to debug through the file selection screen. So we load our file by typing 'TD PSedit.exe testfile' or 'LDR PSedit.exe testfile' make sure 'testfile' exists. I will use TD as I am lazy and don't want to load SoftIce.

Step over the instructions one at a time until you reach the first CALL. We hope that the nagscreen drawing is in a call. We may lose control at a call so we should write down the address of each CALL we encounter. However, we notice nothing happens and we know that plenty of other stuff happens before the nagscreen anyway so carry on. A bit later you're caught in a LOOP like this:

```
cs:0037 F2AE        repnz scasb
cs:0039 E343        jcxz   007E          *Possible exit
cs:003B 43          inc    bx
cs:003C 263805      cmp    es:[di],al
cs:003F 75F6        jne    0037          *Possible exit
cs:0041 80CD80      or     ch,80         *Possible exit
```

Rather than spending an eternity going through loops, you can set BREAKPOINTS. The breakpoints we will consider now work by returning you to the debugging screen when the instruction it is placed on is reached. So you can RUN the program and control will return to you automatically. Where do we set the breakpoints? From the above listing there are three possible ways out of the loop. These have been labelled with a '*' symbol. The JUMP instructions can take us to other sections of code. The first one takes us beyond this loop so we **put a breakpoint at the location CS:007E**. The second JUMP on inspection is the actual loop which takes us to the top of the loop. We should not put a breakpoint at CS:0037. Note that we should *not* put a breakpoint on the actual JUMP instruction for either of these JUMPS. I'm sure it is obvious why. The last possible exit is simply at CS:0041. This happens when the JUMP at CS:003F is not made. **Put a breakpoint at CS:0041**. Breakpoint can be placed in TD by moving the cursor over the instruction and pressing F2 or by alt-F2 to specify the location explicitly. With SoftIce use the BPX command.

Once you've placed the breakpoints, you can run the program by pressing ctrl-G in SoftIce or F9 in TD. You should then be returned to the debugging screen. Where have you landed? I've stopped at 0041. Now you can remove the breakpoints if you like, or just leave them there. Let's continue.

Carrying on stepping until you get to CS:013B here there is a CALL. If you step over the CALL you will find that you have 'launched' the program and it runs as usual and prompts you for at backup and displays the nagscreen. You've narrowed the search for the nagscreen down somewhat but now you must reload the program file and set a breakpoint for this location (you did write down the location of all the CALLs didn't you?). Now reload and go to this CALL. When you get there again, this time TRACE through the CALL. You will now end up somewhere else. The CODE SEGMENT will change. Don't worry about this, but remember when you're in a different segment you can't just set a BPX/RUN by specifying the OFFSET only.

If something went wrong and you got an "Out of Memory" error or something similar, don't worry, it just happened to me as well (I've not optimised my new system yet). Don't worry - you can either do what I've done and load SoftIce for 95 (as I can use two screens and write this tutorial at the same time!) or if you don't have it (search for it on the internet!) load debug you can use that just as well.

If don't know what SEGMENT/OFFSET are then read on. The address in the code window is given by xxxx:yyyy where xxxx is the segment and yyyy is the offset. When you run your debugger make sure that you have the exact same programs loaded so that all your SEG/OFFs will be the same so once you've finished one debugging session you can return to it and continue your work.

Remember when we first ran the program to see what happened? It displays a backup request screen. Now that we're in the main program 'procedure' we can reasonably assume that the different parts of the programs will be placed in separate CALLs. What does this mean? Well for a start it means that you can stop writing down all the CALL addresses until you reach the backup request screen as you know that will be run before you reach the nagscreen. As I said, experimenting with the program beforehand instead of diving right in can save you time - but only if you don't go at it like a robot. Remember, think - and it will save you time and energy.

Soon you reach the backup screen on my computer that's 2690:6637 CALL etc. etc. Now a bit further on from that you reach another CALL on my system it's at 2690:6642 CALL etc. etc. BINGO! There's the nagscreen. Now how to get rid of it? First quit the program and reload your debugger. Go back to this location without running the nagscreen (you can put a BP on the main CALL at CS:013B then TRACE and BP on CS:6642. The two most basic ways to bypass a CALL are:

- NOP out the CALL intstruction - i.e. wipe the CALL blah from the program by replacing it with NOP instructions. The NOP instruction does nothing - NO oPeration.
- RET/RETF the CALL - i.e. go to where the first instruction in the call is and change it to a RET for near calls and RETF for far calls. RET/RETF basically say return to the next instruction after the call instruction. How does it know where it came from - simple, a CALL PUSHES its address onto the STACK. This has important ramifications - more on this in later tutorials.

Let's do it the simplest way. Edit the memory location where the call is and replace it with NOPs, you can either assemble the operation there or do a direct hexedit, the NOP instruction has hex value 90h. Using SoftIce, this can be done by pressing 'a' for assemble and then typing 'NOP' 3 times after the last NOP press 'enter' an extra time to return to the code. Now press 'g' to RUN through the code and Hey Presto! It works. The program had no protection against the most basic crack possible.

Now we don't want to load the debugger every single time we want to hexedit do we? So we need to make this patch permanent. We want to patch the instruction but how do we know where the instructions are in the executable file? Well in the code winder after the SEG/OFF information there are hex values followed by the assembly instructions. The hex values are what the assembly code represents i.e. it is the machine code. All we need to do is write down a few of the hex values so that we have a *search string* with which to locate the code we want. There are some pitfalls here however as some of the code is assigned 'on the fly' so with far calls etc. the hex values may be different. With some practise you will know which values to select. Remember you must have a search string to find the instruction but not too wide as to give you many possibilities. PSedit can be loaded with or without command line parameters so maybe there are two such calls to locate - maybe we need to debug in both ways to find the two calls. If this is the case would it be easier to RET the call as it is likely to use the same code for both calls? These are the decisions you must make when you crack on your own. As it turns out there is only the one call so you should only get one search 'hit' if you have made the search string correctly. If you have more than one choice you can either examine the code at that point (HIEW is a hex editor that can display the code and I think that this is the best hex editor you can get although PSedit will complement it nicely as it has some features which HIEW does not have) and determine whether it is the right one. Alternatively you can redefine your search string to narrow the search down to a single hit.

As it happens the actual CALL instruction E8 0A 9F is a perfect search string and you can use PSedit to change this to 90 90 90. When you run the patched version of the program you will find that it works. Congratulations! You've just cracked you're first program!

There’s More?

So you’ve cracked your first program. But you want to let other’s use your crack too and show off that you can crack. What can you do:

- Publish the search string(s) - If it’s narrow enough then there should be only possibility and the patch can be easily made.

    - Advantages: Simple. Takes very few bytes. Requires no programming. Quick.
    - Disadvantages: Morons can’t use it. You can’t show off you programming skill.

- Write a patching program - Write a program (best in assembly for size and speed) to automatically patch your program. This can be easily made so that it can be re-used with minimum alteration.

    - Advantages: Morons can use it. You make a snazzy patching program.
    - Disadvantages: Requires initial time investment to write the program. Susceptible to viruses.

I have seen somewhere a patching program template written in assembly. Basically you can just fill in the offset and patch values then compile and go. I may write one if I have the time. It’s always worth writing your own. If you don’t know how to write simple assembly programs - LEARN! It will be invaluable for cracking. I advise anyone who cracks or writes viruses to be cross-trained in assembly/cracking/virus writing/programming. You can learn a lot from these ‘fields of study’ and knowledge in all of them will aid you in each of the fields.

One note of warning though. When writing your own patching programs, be aware that there are certain lame people who having learned how to use a hex editor (a process which with which they require expert tuition) they think they are kewl,c00l or eL1tE by hacking their names in place of yours. A simple way to guard against this is to put a simple XOR loop encryption scheme into your patching program. OK it’s easy to defeat but the lame morons won’t be able to do it. I hope that you would be able to defeat such a scheme and that is the point as it allows fellow crackers to examine what is being patched so can examine the crack.

Another advantage is that once you have removed the ‘encryption’ you can examine the code and detect any Trojans or viruses (that is if you know how to write them - don’t think that your anti-virus programs will protect you from the new viruses - and they are mostly useless against Trojans anyway). Remember that the internet especially the alt.cracks newsgroup is a breeding ground for new viruses. On the subject of newsgroups, if you’re a cracker you may be rather annoyed how cracks and requests for cracks are spilling over from alt.cracks into alt.crackers. Can we make alt.crackers a forum purely for discussion of cracking, please? Let’s stop this useless crosspost. I ask all not to respond to requests for cracks which have been posted or cross-posted to alt.crackers.

Going back to writing your own patching program. Don’t go over the top when writing it. I’ve seen many with anti-debugging tricks etc. It’s just not necessary. For patches one can simple run the patch, compare the files and then create a new patch from the comparison information. If you do see a bloated patch program I suggest you do this and release a leaner version of the patch as a service to everyone.

What Now?

Well this is the end of the first tutorial. If I have time I'll write more. In the mean time look for the fred.zip archive on my website and try to crack that. You should be able to crack it using the techniques you've learned in this tutorial. Look for other tutorials on the web and keep cracking. Practice makes perfect!

My PGP Key

Here's my PGP public key. You should also be able to get if from the keyserver at pgpkeys.mit.edu port: 11371.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGP for Personal Privacy 5.0

mQGiBDQajkIRBADEEWE2foxCs5dhfGBdgkCF2C9Ayr0CAVH7KR8Ko2odEFTfn/kq
Agm0i5XFxjQfcnf2RKqkTZV9T4IRpdZf9O+ZIAOvaLoFTzJ+uGWnoX1JoToiMKse
BFdLya7PoTwMsYdDAhD5Wx4VenEO9DJQw4kTGcejxo8hXJdf5/y9R8oqUwCg/yrz
ZCzFkcKqM1Gh6hIqozncOJcEAIUYZ7gYA3OIT6Y/zPVk6bIHMsSp1N25PUklASMl
DOeVEQhTnprgcjFpTQ3lGt3cSbpeGHYpTFL5w7KV5RETExw8EAgVp1bt9K9EnARm
W8NpZLNOWq+fA1AnlrozJqRtRFMR//E0+SY6sWaWc+l7zFzZLtg59EheKAs3goHt
XxukBACrqwUtEu2bWo3IrO/cfcL1lVqFTHHO1HhZj7drAa/u/pePBU1/lQO1VB4Q
GGrP+AjByaZuoaaq+5Zsn613dYBMwTYidvavvXE2sMTtTpPzZrsRmuQO2e0VCQFj
/URcMeiq1HbWFgzAae3Z3DIo3B0B/cR3xz2QzwncvrV4LtOrvLQjR2hpcmliaXp6
byA8R2hpcmliaXp6b0Bob3RtYWlsLmNvbT6JAEsEEBECAAsFAjQajkIECwMBAgAK
CRCHrpKzXfMtCKmRAJ9glxCHUFUEm3E+8btKRNoUsgOPowCfbpFB3sPAcoGL3low
5NogFIc61ea5Ag0ENBqPrBAIAOPGH97Q2EJkRU+3jf7O/i3WTCMdtyvFVFV8IW1R
ltshMsGrj4c0yCTz6FqRv6pbOHmLa8jFMOXeXXWrdgdci68wq1I4KrCihklKpzeR
Ac05GD6BEiguL7HGW+ZSXrYmT05OVsXv4LLpx+k3ubIGx6wNHMst0CTUeSgGoNO9
8esRxD2hw2Jl0peV4p8MLTFefV5CYJy77uFBTC1xpzKi/mw6rupVNy6MY0BD+yhF
3O2bKgSm1adi24TOV/N7juMfDDWiUCbdqNDlLQMWLH0Bh9qA08/24dpBxTTmJ2Wo
nzdmNLKGdgAY6KoK/Jfi4bFqtcQR5xddZBAkx8prXnjCbB0AAgIIAKJCwmHzrbtq
neCGmtpEFsi3VTi19xfaQ4L4CUfYp8vxzve7zbTyWvkRLu3iePCKb/8+nPEUI394
NzTNbCI1ymKpkJlyptNsSS+d2Mo0e7Iox2TTrldXH3pWD6MSKJPyEnqUxzZ8Dr/6
kBapm4FoyAYMPRBJOBQHj90WL/JNVNoQ43OU4YAmY7mPMrFQ44FG21BDMY9Yi4oK
mX8UZTSy8GnbtR4dE9O3Bqr11dmsiWWKBntOytyBb7jkI482YFKTSZksavgSEVXT
RKivPpWHGhiqCGGhmXF3vRYLAqI32tYs0W9NsFC8mtlbZ14xneO74cOmxl1QGDaq
+pC1o2QCcKmJAD8DBRg0Go+sh66Ss13zLQgRAq1xAJ0TQYCiYVSu7vCV+a0Xadoe
BeHsfgCg99Uayd7RPLk02phc4Ha2M2BlN1g=
=p+vb
-----END PGP PUBLIC KEY BLOCK-----
```

Formats

This tutorial will first be released in Acrobat PDF format. This is because it will be more difficult to modify the tutorial and so I can ensure that during the first round of distribution everyone will be able to get my PGP key and be able to verify the tools supplied on my web page.

It is best to print out the tutorial and follow it from the hard copy. Alternatively if you have more than one computer in the room you can view the tutorial from one computer and crack from the other. Version 1.0 of the PDF document will not allow you to copy and paste the text. Sorry for the inconvenience. A plain text version will be released at a later date if there is enough demand.