

# Ghiribizzo's Cracking Tutorial

## Function Hijacking : HexWorkshop 2.52

---

### Function Hijacking

This tutorial combines several styles and techniques of cracking and covers a range of depths. This tutorial shows how functions within the program can be hijacked for our own devious little purposes, in this case, a nice serial generator.

---

---

### 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](mailto:Ghiribizzo@geocities.com)

My Backup Email: [Ghiribizzo@hotmail.com](mailto:Ghiribizzo@hotmail.com)

---

This document is Copyright © 1997 by Ghiribizzo. This document may be distributed non-commercially, provided that is it not modified in any way. This publication may not be sold or packaged, in whole or in part, as a service, or with a product for sale in any form without the prior written permission of the author. This document is presented with no warranties or guarantees of any kind including fitness for any particular purpose. If you use the information contained herein, you do so at your own risk.

# Cracking HexWorkshop 2.52

## What is it?

HexWorkshop is a Windows 95 hexeditor. It is liked by many, but I haven't really used it a lot, mainly because I prefer 'Hackers View' (or Hiew as it's commonly known). This version is an old version, and I've not bothered to check the newer ones, but the cracks here probably will work (assuming the serial number system doesn't change).

## Quick lame crack No. 1

Use my serial number 'syGhirib'.

## Quick lame crack No. 2

Easy, use the serial number JN11mARQ. The whole world has seen this serial number. That's why it has been hardwired into the program to prevent lamers from using it. If you've read my previous tutorial (ghiric4.pdf) then you'll know exactly how to get around this. Damage the 2 hardwired strings inside the exe and register as normal.

## Quick lazy crack

Search the exe file for null terminated strings of length 8. You'll find the above serial number and can continue from there. You should write your own utility to do this and it can find lots of serials from programs. Also there is a ready made utility (search for strings.zip). But you'll need to write a filtering program to get strings of the correct length. Usually searching for strings of size 4, 8, 16, 32 bytes work well and also searching for strings containing the '-' character. I don't think anyone has mentioned this technique before, but try it. If you do find the string and use it in conjunction with the string damaging technique you can crack very quickly indeed.

## Dead Listing

Normally, I use Hiew to crack by raw listing, but this time I'll use the ever popular W32dasm. I've never actually used W32dasm before, but I have been impressed by the results I got from this project. Firstly, deadlist and save the results to a file. Searching around for strings related to the registration procedure we get the following snippets:

```
Name: DialogID_0064, # of Controls=009, Caption:"About Hex Workshop - Unregistered"
001 - ControlID:0412, Control Class:"" Control Text:"Hex Workshop Version 0.00"
002 - ControlID:0436, Control Class:"" Control Text:"16/32 Bit Version"

Name: DialogID_0075, # of Controls=003, Caption:"Registration Unsuccessful"
001 - ControlID:FFFF, Control Class:"" Control Text:"You have entered an invalid registration"
```

Examining the references to these and searching around a little we come across the following pieces of code:

```
Possible StringData Ref from Data Obj ->"JN11mARQ"

:004317AB 680C534800          | push 0048530C
:004317B0 8D45DC             | lea eax, dword ptr [ebp-24]
```

*Again the attempted 'protection' becomes a liability and points us to part of the protection routine.*

```

:004317B3 50          push eax
:004317B4 E867CE0000 call 0043E620
...
:004317C7 50          push eax
:004317C8 E893970000 call 0043AF60      Moving on...
:004317CD 83C404      add esp, 00000004
:004317D0 8945EC      mov dword ptr [ebp-14], eax
:004317D3 E907000000 jmp 004317DF

```

\* Referenced by a CALL at Addresses:

```

|:0041EDDE , :004317C8
|
:0043AF60 83EC14      sub esp, 00000014
:0043AF63 B9FFFFFF      mov ecx, FFFFFFFF
...
:0043AF77 83F908      cmp ecx, 00000008
:0043AF7A 7408          je 0043AF84      And again...
:0043AF7C 33C0          xor eax, eax
:0043AF7E 5F          pop edi
:0043AF7F 5E          pop esi
:0043AF80 83C414      add esp, 00000014
:0043AF83 C3          ret

```

Referenced by a (U)nconditional or (C)onditional Jump at Address:

```

|:0043AF7A(C)
|
:0043AF84 6A0A          push 0000000A
:0043AF86 8D4602      lea eax, dword ptr [esi+02]
:0043AF89 6A00          push 00000000
:0043AF8B 50          push eax
:0043AF8C E85F250000 call 0043D4F0
:0043AF91 8D4C2414     lea ecx, dword ptr [esp + 14]
:0043AF95 83C40C      add esp, 0000000C
:0043AF98 8BFE          mov edi, esi
:0043AF9A 51          push ecx
:0043AF9B 50          push eax
:0043AF9C E83FFFFFFF call 0043AEE0
:0043AFA1 83C408      add esp, 00000008
:0043AFA4 B9FFFFFF      mov ecx, FFFFFFFF
:0043AFA9 2BC0          sub eax, eax
:0043AFAB F2          repnz
:0043AFAC AE          scasb
:0043AFAD F7D1          not ecx
:0043AFAF 2BF9          sub edi, ecx
:0043AFB1 8BC1          mov eax, ecx
:0043AFB3 C1E902      shr ecx, 00000002
:0043AFB6 8BF7          mov esi, edi
:0043AFB8 8D7C240C     lea edi, dword ptr [esp + 0C]
:0043AFBC F3          repz
:0043AFBD A5          movsd
:0043AFBE 8BC8          mov ecx, eax
:0043AFC0 83E103      and ecx, 00000003
:0043AFC3 F3          repz
:0043AFC4 A4          movsb
:0043AFC5 8D742408     lea esi, dword ptr [esp + 08]
:0043AFC9 8D4C240C     lea ecx, dword ptr [esp + 0C]
:0043AFCD 51          push ecx
:0043AFCE E84D300000 call 0043E020
:0043AFD3 83C404      add esp, 00000004

```

\* Possible Reference to Menu: MenuID\_0002

\* Possible Reference to String Resource ID=00002: "Hex Workshop"

```

|
:0043AFD6 B902000000 mov ecx, 00000002
:0043AFDB 8BF8          mov edi, eax
:0043AFDD 2BC0          sub eax, eax
:0043AFDF F3          repz
:0043AFE0 A6          cmpsb
:0043AFE1 7405          je 0043AFE8
:0043AFE3 1BC0          sbb eax, eax
:0043AFE5 83D8FF      sbb eax, FFFFFFFF

```

I'll not bother commenting on most of the above code. I've included it only so that you'll be able to see how I got to the relevant code. The last few lines are the interesting ones (for lazy crackers that is). Let's look at them again:

|                      |                   |   |
|----------------------|-------------------|---|
| :0043AFD6 B902000000 | mov ecx, 00000002 | >sets 2 bytes for the repz cmpsb instruction. |
| :0043AFDB 8BF8       | mov edi, eax      | >sets our serial in ds:edi                    |
| :0043AFDD 2BC0       | sub eax, eax      | >zeros out eax                                |
| :0043AFDF F3         | repz              | >compare                                      |
| :0043AFE0 A6         | cmpsb             |   |
| :0043AFE1 7405       | je 0043AFE8       | >jump if equal                                |

There are some 'obvious' cracks we could try: invert the jump, repz cmpsb -> repz movsb and others if you take a look at surrounding code. I haven't tried any of these but they probably work. Actually, I did try the repz cmpsb to repz movsb. This does work and is strange in that you've tricked it into producing a .reg file which the program uses to check validity when in fact the serial number is not valid. This is very peculiar as you could generate as many .reg files as you like, with none of them having a valid serial yet all of them working. Clearly this is a poor protection scheme.

In any case, I don't like this method, I want to find some valid serials. And it's pretty obvious how. The above compare code looks at ds:edi and es:esi. Where es:esi contains the 2 bytes it is going to check in our serial number which is pointed at by ds:edi. So what do we do? Simple, run the program under a debugger and then place a breakpoint on 'sub eax, eax'. When you break simply dump es:esi. You'll find that es:esi points at code just 4 bytes ahead of your serial number so you'll see something like this:

```

S Y . . a b G h i r i b
^          ^
|          |
|          | ds:edi points here at your serial
|          |
|          | es:esi point here at the correct 2 'checksum' bytes

```

Where 'abGhirib' is the serial number you entered and 'SY' is what the first 2 bytes should have been. I haven't actually investigated the protection scheme fully, but I assume that it is some sort of checksum functions which calculates the 2 checksum bytes (the first 2 bytes in the serial number) from the last 6 bytes of the serial number. So all you need to do is type in the serial number you want and then dump the memory and enter the correct serial next time.

**Note for newbies:** you may have difficulty in setting the breakpoint. I did this by setting a breakpoint on execution of 'CreateDialogIndirectParamA'. This is the function which displays the 'sorry wrong serial' message. When it breaks you can scan around for the relevant code sections or do a search for the opcodes (latter is easier and quicker). If you have difficulty try varying the search string length.

## Let's look a bit deeper

Try typing a few different serial numbers and if like me you're quite lazy and use letters only, you'll see that the checksum routine is seriously flawed. Most, if not all, 'letter-only' serial numbers have a checksum of 'SY'. Only when I introduce numbers do I get a serial of anything else. What does this mean? It means we don't even need to debug! We just choose any letter-only serial of our choice so long as it starts 'SY'. So if I'm correct any serial of the form 'SYxxxxxx' where x is any letter, will be valid. This means that next time, the authors will need to put in a routine to disallow any serial which is letter only, or which starts 'SY'. I don't know how the serials are assigned but they probably contain numbers if we take the JN11mARQ serial number as being typical. If so, then they won't need to change the serial numbering system. Crackers often say that it should be changed once it has been cracked, but the costs of the change usually make this prohibitive.

## What now?

When I began this project, I intended to develop a brute force serial number collector (working by trying serials in sequence and testing them using the protection algorithm and keeping only the valid ones - incidentally, I haven't seen a generator of this sort used by crackers before).

When I discovered that the serial just checked for the 2 first bytes and stored them quite simply, I was going to make a key generator which 'corrected' the first 2 bytes, but now having seen that we can already generate so many serials from the flaw, it seems quite pointless in making a generator.

Still, it's worth having a look at the protection scheme and it would be easy to patch the exe to correct the serial 'on-the-fly' rather than build a generator. Alternatively, you can patch it to display your 'corrected' serial and thus convert the patched program into a serial generator. This is a quicker and uglier way to make a serial generator.

So, have a go at some of my suggestions. Happy cracking.

## Post Script

I decided to write a patch to convert HexWorkshop version 2.52 into a valid serial finder. This patch can be found in 'ghiric5a.zip'. Instructions for using the patch will be supplied in the zip file. I will briefly describe how it works here.

I decided to modify the exe itself rather than rebuild the protection scheme engine. This is because it is much easier this way. The serial finder works by hijacking the 'invalid serial' dialogue box. The cosmetic side of the dialogue box change was done simply using a hex editor (Hiew). The meat of the crack (such as it is) is simply at our favourite point in the program:

|           |            |                               |                               |
|-----------|------------|-------------------------------|-------------------------------|
| :0043AFC5 | 8D742408   | lea esi, dword ptr [esp + 08] |                               |
| :0043AFC9 | 8D4C240C   | lea ecx, dword ptr [esp + 0C] |                               |
| :0043AFCD | 51         | push ecx                      |                               |
| :0043AFCE | E84D300000 | call 0043E020                 |                               |
| :0043AFD3 | 83C404     | add esp, 00000004             |                               |
| :0043AFD6 | B902000000 | mov ecx, 00000002             | Our code will overwrite here. |
| :0043AFDB | 8BF8       | mov edi, eax                  |                               |
| :0043AFDD | 2BC0       | sub eax, eax                  |                               |
| :0043AFDF | F3         | repz                          |                               |
| :0043AFE0 | A6         | cmps                          |                               |
| :0043AFE1 | 7405       | je 0043AFE8                   |                               |
| :0043AFE3 | 1BC0       | sbb eax, eax                  |                               |
| :0043AFE5 | 83D8FF     | sbb eax, FFFFFFFF             |                               |

This is the modified code as shown by Hiew:

|            |              |      |                |                                  |
|------------|--------------|------|----------------|----------------------------------|
| .0003AFD6: | 66A120F26000 | mov  | ax,[00060F220] | Copy 2 checksum bytes into AX    |
| .0003AFDC: | A21C074A00   | mov  | [0004A071C],al | Copy first checksum byte from AL |
| .0003AFE1: | 88251E074A00 | mov  | [0004A071E],ah | Copy second byte from AH         |
| .0003AFE7: | 90           | nop  |                | Maintain the offsets             |
| .0003AFE8: | 85C0         | test | eax,eax        | Continue as if invalid...        |
| .0003AFEA: | B800000000   | mov  | eax,00000000   |                                  |
| .0003AFEF: | 7505         | jne  | .00003AFF6     |                                  |
| .0003AFF1: | B801000000   | mov  | eax,00000001   |                                  |

I'm sure you can figure this out. It simply copies the correct 'checksum' into the location where the dialogue text is stored. When the dialogue box is called, the text will already have been patched. Note that this patch alters the program so that it will *never* register, even if you enter a valid serial number. This patch is merely to convert the program into a serial generator. Once you've got your serial you can re-install. Alternatively, you can rename the program and keep it - if you think you'll need more serials in future.

There are several files in 'ghiric5a.zip' 3 including 3 different ways to crack HexWorkshop. Enjoy.