NO MORE annoying anti SOFT-ICE tricks
(how to improve SOFT-ICE)

by The Owl (Winice wizard eccelso)

HCU

[Picture]Courtesy of fravia's page of reverse engineering

NO MORE annoying anti SOFT-ICE tricks or

how to improve SOFT-ICE!
intro
today's best EXE protectors contain code to prevent debugging with SOFT-ICE, which is the best debugger among the ones i came across so far. following protectors are known to me to defeat SOFT-ICE:

EEXE (Encrypt Exe found in FZC.EXE)
HACKSTOP (found in WWPACK.EXE)
PROTECT! (found in various files)
GUARDIAN ANGEL (found in some versions of HWINFO.EXE) EXELITE (a Polish exe compressor) the one written by PREDATOR 666 (found in DCA.EXE v1.4) the one used by Martin Mal¡k (found in HWINFO.EXE v3.05 and up) DS-CRP by Dark Stalker (hi!)

SECURE v0.19 by the authors of WWPACK ALEC v1.5 (the very best protector :-) by Random and a few others i don't remember right now...

the problems

SOFT-ICE can be detected/halted/crashed in many ways, here's a short list of them (some of them only makes single stepping harder but not impossible). i also included a short note on the effects for each version of SOFT-ICE:

:-(this trick works,
:-) it causes no problems

1. by the use of the INT3 interface provided by SOFT-ICE one can

   check for the presence of SOFT-ICE and then have it execute various

   commands, e.g. HBOOT (see HackStop). the loader part of SOFT-ICE

   also leaves some traces in the low DOS memory, so one can find out

   the entrance values of the INT3 interface even if they were changed

   (see HMVS - a heuristic macro virus scanner - by J. Valky and

   L. Vrtik that uses SECURE v0.19). the Windows versions can also be

   detected by the following code (thanks to Dark Stalker and ACP :-):

   ```
   mov ebp,"BCHK"
   mov ax,4
   int 3
   cmp ax,4
   jne winice_installed
   ```

it seems that BoundsChecker talks to SOFT-ICE via an interface very similar to the one between LDR.EXE and SOFT-ICE (although it's completely undocumented, as far as i know...)

DOS: :-(
W31: :-((but the SECURE method is :-)
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

2. by checking for various devices SOFT-ICE installs ("CVDEBUG", "NU-MEGA", "SOFTICE1") and searching the part of the code SOFT-ICE leaves in the low DOS memory for some patterns (e.g. "CVDEBUG", "NU-MEGA", "SEGMAP", "S-ICE" (the latter coming from the filename), and any instruction sequence that's left there) one can detect the presence of SOFT-ICE. the Window$ versions provide a VXD entry point that can be get by:

```
mov ax,01684h
mov bx,0202h ; VXD ID for Winice, check out Ralf Brown's INTLIST
xor di,di
mov es,di
int 2fh
mov ax,es
add di,ax
cmp di,0
jne winice_is_installed
```

the Window$ versions can also be detected by calling the "debugger install check" function of the Window$ debug kernel (int 42/ax=0041), for more details see Ralf Brown's INTLIST.
see HWINFO.EXE and DS-CRP.COM for an extensive application of these checks...

DOS: :-(
W31: :-) but the VXD entry point check is :-(
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

3. by the use of the undocumented ICEBP/INT01 instruction (opcode 0xF1):

   SOFT-ICE gives a short beep before the execution of this instruction
   (it will be reflected back to the V86 mode handler, thus at least
   the intended handler will get it) which can be VERY annoying (and
   make the execution VERY slow), see EEXE for an application of this.
   DOS: :-(
   W31: :-)
   W95: not tested yet (probably same as W31)
   WNT: not tested yet (probably same as W31)

4. by using the TRAP flag, one can use the single stepping feature to

   call a protection routine (e.g. a decryptor). the problem is, that
   during single stepping SOFT-ICE clears the TRAP flag for the V86 task
   and will neither execute nor step into the INT01 handler of the
   V86 task. many schemes use this trick.
   DOS: :-(
   W31: :-(
   W95: not tested yet (probably same as W31)
   WNT: not tested yet (probably same as W31)

5. by the use of the debug (DRx) and control (CRx) registers:

   accessing these registers in V86 mode leads to a General Protection
   Fault (INT0D), which SOFT-ICE doesn't handle correctly (it's normally
   used for emulating instructions that access the interrupt flag, the
   debug registers, the control registers, etc.). the protected mode
   handler emulates instructions that access these registers by
   executing them, however it doesn't make note about this for itself,
   i.e. whenever a debug fault is triggered SOFT-ICE will think that
   it must pop up and won't reflect back this exception to the V86 mode
   handler (that's waiting for it in vain). for a working example see
   GUARDIAN ANGEL. furthermore, accessing DR4/5 and CR1 will halt
   SOFT-ICE with a General Protection Violation error message, which is

of course quite disturbing if it's used many times in the program...

and best of all by accessing CR4 SOFT-ICE simply crashes since

there's no emulation code for this kind of instructions (there's a

jump table that tells SOFT-ICE which routine to use to handle these

instructions and the table ends with CR3...). this method was first

used by Random in his ALEC.EXE v1.4 :-)

another sad fact is that the emulation in SOFT-ICE is not complete:

it ALWAYS uses eax no matter what the original instruction was...

e.g. mov dr0,ebx will load dr0 from eax! mov ecx,dr0 will load eax from dr0!

i guess it's needless to say how easy it is to detect this behavior...

a sidenote for protection writers: other memory managers that run DOS in V86 mode may or may not handle these instructions correctly, so the use of this trick is highly discouraged.

DOS: :-(

W31: :-(
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

6. by the use of INT08, which is the timer interrupt in real mode DOS, and the Double Fault Exception in protected mode. the protected mode handler checks whether it was entered from V86 mode or not, and in the first case it reflects back this interrupt to the V86 mode handler. however, one can't single step into it.

DOS: :-(

W31: :-)
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

7. by the use of the INT07 interrupt (this is the Coprocessor Not Available Exception):

instead of reflecting back this interrupt to the V86 mode handler, SOFT-ICE tries to skip the offending coprocessor instruction (it checks for some opcodes). it seems the Nu-Mega folks never thought it would be called directly... for a real life application get some programs written (and protected) by Predator 666.

DOS: :-(

W31: :-)
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

8. by the use of the Invalid Opcode Exception (INT06):

SOFT-ICE tries to emulate some instructions (e.g. LOADALL), and then reflects back this exception to the V86 mode handler. however, certain opcodes aren't recognized and will give you an error message (i.e. execution will be interrupted, if the protection scheme uses this trick).

DOS: :-(

W31: :-(
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

9. by using direct INTxx calls that are triggered by hardware interrupts (e.g. INT08-INT0F, INT70-INT77, if the vectors in the PIC are not reprogrammed), one will not be able to single step into the interrupt handler. in fact, SOFT-ICE will even execute the next instruction and just then stop (if the next instruction is also an INTxx call of this type then it will be stepped over as well, and so on). so far, i know of no protection scheme that uses this trick, but i guess i've just given out a good idea :-).

DOS: :-(

W31: :-)
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

A. by reloading IDTR one can change the base and size of the interrupt table in real mode as well. however, SOFT-ICE will not emulate this instruction (it causes a General Protection Fault in V86 mode) thus a protection using LIDT won't run. the only problem is that memory managers don't like it very much, so probably we won't see it in a real life protection scheme, but one never knows :-).

DOS: :-(

W31: :-(
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

B. sometimes one has to call an interrupt directly (GENINT xx), e.g. to dump a memory range to disk by using one's memory resident dumper (you know what i mean :-) and it's very annoying that SOFT-ICE doesn't stop after the interrupt call but executes the program being debugged (thus one has to set a breakpoint for a moment at the current CS:IP which will result in an unwanted 0xCC byte in the dump, if all debug registers are already used).

DOS: :-(

W31: :-(
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

C. in plain DOS INT3 and INT1 are handled by the same routine (which is

a simple IRET). however, SOFT-ICE changes the INT3 handler of the V86 task to another IRET which can be detected by comparing its offset to the one of the INT1 handler. see HWINFO.EXE for an application

DOS: :-(

W31: :-)
W95: not tested yet (probably same as W31)
WNT: not tested yet (probably same as W31)

the solutions

in the following part you'll be presented with some ideas about the solution for the problems described above (the file offsets refer to the DOS version v2.80, but the ideas should work for other versions, as well. the easiest way to apply the patches is using Hacker's View which can be found in HIEW531.ZIP).

one general problem is that SOFT-ICE (at least the DOS version) doesn't reprogram the hardware interrupt vectors, and this makes life (and the interrupt handlers) a bit more complex. the IDT that SOFT-ICE uses has entries that point to the following type of code:

```
push xx
jmp handler_xx
```

where xx goes from 0x00 to 0xFF. in v2.80 this code begins at offset 0x4534. the Win31 version has a very similar code beginning at offset 0x14167 in v1.52:

```
push d,[offset_xx]
jmp handler_xx
```

if you want to understand the patches that follow right below, you should study the interrupt handlers (and you should also have a good understanding of protected mode). however, some problems cannot be solved without understanding the internal flags of SOFT-ICE, and this requires a complete disassembly of it, which is a quite hard task i can tell.

anyway, sooner or later it will be done, and then we'll have the ultimate debugging/cracking tool in our hands 'cos we'll be able to put in some missing functions, e.g. emulation of FlatRealMode, tracing INT1, PIC reprogramming, prefetch queue emulation, dumping a memory range to disk, etc. until then, enjoy the poor man's patches...

1. some exe protections mentioned earlier are based upon the INT3 interface of SOFT-ICE (see Ralf Brown's Interrupt List for details). this interface is activated when the protected mode INT3 handler of SOFT-ICE encounters the magic values in SI and DI. that is, when you try to trace through an INT3 call, SOFT-ICE will regain control, check for the magic values, and in case they are not found, it will reflect back this interrupt to the V86 mode INT3 handler (which it was supposed to do anyway). if it finds the magic values, then it'll execute the command given in AX (and DS:DX). all of these checks happen invisibly to the hacker, so there seems to be no solution to defeat this kind of protection (well, there's a slow way if you step through every instruction and before the "guilty" INT3 call you change one or two registers).

however, there's a simple solution: change the magic values SOFT-ICE is looking for and this will defeat those protectors based upon the INT3 interface. however, it's easier said than done because both SOFT-ICE (and WIN-ICE) itself and (W)LDR.EXE use this interface for some kind of intra/inter process communication. so every reference to the magic values will have to be changed!

to keep the story short here's what i've come up with: browsing a few minutes in Hacker's View (another important tool ;-) i found the places where those changes had to be done. in order to avoid changes where those magic values occur by chance, i wrote an MSUB script to change whole instructions (they represent enough context). the amount of necessary changes would have forced me to use some search&replace utility, anyway. MSUB.EXE can be found in MSUB13.ZIP (use an ftp search engine to find it).

the scripts
SICE-VAL.MS: you should specify the old and new magic values in it (note that numbers are decimal!)
SICE2NEW.MS: it will replace the old magic values with the new ones. there are almost 2^32 possible values, only that value is forbidden for SI that equals to the version of SOFT-ICE (for v2.80 it is \128\2, i.e. 0x0280). before SOFT-ICE installs itself, it checks for its presence by using an undocumented function of its INT3 interface by setting AH to 0 or 1. on return SI will be loaded with the version number or left unmodified if it's not installed yet, that's why the version number must differ from the preloaded value of SI.

example usage

MSUB.EXE SICE2NEW.MS S-ICE.EXE LDR.EXE

if you're fed up with the shareware delay built into MSUB.EXE, here's how you can get rid of it (thanks to a friend of mine ;-)

patch MSUB.EXE v1.3 (113,152 bytes long) as follows:
offset old new
00002307: ?? EB
00002308: ?? 03
0000C0DF: 77 EB

the protection that SECURE uses is quite clever 'cos it checks the
first 16 kBytes for the instruction sequence of
mov si,SI_MAGIC
mov di,DI_MAGIC

they're encoded as 0BEh,x,y,0BFh,u,v where x,y,u and v are the magic values. when it finds 0BEh and 0BFh separated by two bytes from each other then it calls INT3 with the supposed magic values (the INT3 handler is a simple IRET, so no problem should arise if SOFT-ICE is not installed). and guess what happens when SECURE finds the traces of the loader part of SOFT-ICE in that low memory area...

the only solution that would defeat this kind of protection is to change the instructions that load SI and DI before the INT3 calls (unfortunately you can't avoid those calls since SOFT-ICE has to check for its presence and do other things). for obvious reasons (i guess one of the previous versions of this file gave the SECURE authors the idea...) i won't give you tips on how to do it, i hope you're smart enough to do it yourselves.

the only limit is that you have 6 bytes of space to load both SI and DI (i checked that whenever SOFT-ICE uses the INT3 interface it loads SI and DI by two consecutive instructions, i.e. you can use the MSUB scripts to search for and replace them). another method could be to change the HBOOT command to something else (by MSUB of course), however other dangerous commands could be still issued...

the BoundsChecker interface can be modified in the same way we did it with the rest of the INT3 interface: simply modify the value that is expected in ebp (but BoundsChecker must be changed as well!). look for the string "KHCB" to find the appropriate place. the following file offsets are valid for WINICE for Win 3.1 v1.52:

offset old new
000130C7: 'K' ?

000130C8: ‘H’ ?

000130C9: ‘C’ ?

000130CA: ‘B’ ?

for WINICE for Win95 v3.0:
offset old new
00016FDD: ‘K’ ?

00016FDE: ‘H’ ?

00016FDF: ‘C’ ?

00016FE0: ‘B’ ?

and for WINICE for Win95 v3.01:
offset old new
000243F0: ‘K’ ?

000243F1: ‘H’ ?

000243F2: ‘C’ ?

000243F3: ‘B’ ?

2. the device names can be changed to anything you want, look at the beginning of S-ICE.EXE. the real problem is when the protection checks for instruction sequences. my only advice is that step through the protection and see which part of the code is checked for, then try to modify it in S-ICE.EXE (or disassemble, modify and then recompile the whole executable, but that’s not gonna happen for a long time, i guess... :-).

to defeat the int41 check, you have to change both the return value of the real and protected mode handlers in WINICE and the value that is checked for in KRNL386.EXE, VMM32.VXD, EMM386.EXE, IFSHLP.SYS and NET.EXE. all these changes are necessary otherwise some WINICE commands (HWND, TASK) won’t work ‘cos apparently they rely on some Window$ functions which are available only in the debug kernel (and during startup these programs/drivers do this int41 check which will fail if you don’t change the values they’re expecting, as well).

there’re five places in WINICE that have to be patched (one is a cmp, the other four are mov’s) and one in each one of the rest. sorry, that i don’t provide you with detailed offsets, but there’re too many versions/combinations of both WINICE and Win31/Win95... note, that after these changes other programs that want to use debug kernel functions will fail :-)

and now let’s talk about the VXD entry point check. the ID is stored at file offset 0x7821E in WINICE for Win95 v3.01 (to find it in other versions as well just look for “SICE “, and the ID will be a few bytes before this text).

so to change the ID just overwrite it there. however, it’s not enough since some companion programs also test for WINICE by trying to get the VXD entry point, i.e. they have to be modified as well. they’re DLOG.EXE and WLDR.EXE, and search for int2F (opcode: 0xCD 0x2F) to get to the right place... :-)

anyway, for the versions that come with WINICE for Win95 v3.0, the file offsets of the ID are 0x625/6 and 0x68B9/A respectively.

3. what we have to do is simply skip the unnecessary parts in the handler (the beeps) and simulate the instruction as it would have been a direct INT 01 call (opcode: 0xCD 0x01). this way one will not only get rid of the beeps but be able to trace into the handler as well (and we’ll have some space to put some extra code in when we’ll need it :-).

offset old new
00001DD5: 50 EB
00001DD6: 51 60

4. [this part is being worked on]

5. there are two possible solutions: we either disable the DRx emulation feature of SOFT-ICE (this is quite easy to do) or correct it (this is really hard to do). SOFT-ICE emulates each instruction by executing a function whose offset is looked up in a table. each function ends in the same way: IP of the V86 task is incremented by the appropriate amount of bytes. so to disable emulation we'll change the pointers in that table to the common end of the functions, this way these instructions will essentially be handled as NOPs. i don't know whether it's worth to do it or not, since this modification can be detected by simply loading one of the debug registers and then checking whether it's really been modified or not. a more elegant solution would be to reserve a few bytes in the data segment of SOFT-ICE for storing the values of these registers and emulate the instructions (or at least their loading). anyone out there willing to do that?

anyway, here's how to do the patch: the table itself is at file offset 0x1F1DD and it has 12 words in it pointing to the emulation code of the following instructions:

mov eax,dr0 and mov dr0,eax
mov eax,dr1 and mov dr1,eax
mov eax,dr2 and mov dr2,eax
mov eax,dr3 and mov dr3,eax
mov eax,dr4 and mov dr4,eax
mov eax,dr5 and mov dr5,eax
mov eax,dr6 and mov dr6,eax
mov eax,dr7 and mov dr7,eax
mov eax,cr0 and mov cr0,eax
mov eax,cr1 and mov cr1,eax
mov eax,cr2 and mov cr2,eax
mov eax,cr3 and mov cr3,eax

note that there's no offset for emulating CR4...
the offset of the common exit point (i.e. the new value you may want to set these pointers to) is 0x175A (this is NOT a file offset...).
these patches still don't cure the problem with CR4. if you decided to get rid of the emulation entirely (including CR4) then you can do it much easier:

offset old new
00002E7E: 03 0A
00002E7F: 00 01

6. i'll give you a general solution in section 8 since INT08 is just a a subset of the hardware interrupts which are discussed there.

7. this problem can be solved quite easily: we replace the original protected mode handler with the one that serves all not_IRQ_related interrupts (and whose only task is to reflect them back to the V86 mode handler).

offset old new
0000455A: B6 14
0000455B: DF D6

8. what we have to do is to skip the call that prints the error message and simply reflect this interrupt back to the V86 mode handler. note that your buggy programs won't cause SOFT-ICE to pop up after this patch :-) (however, you'll be able to break in and see what went wrong).

offset old new
00002447: A9 C3

9. now we'll make use of the extra space we made in the original INT01 handler. we have to check whether the interrupt was triggered by a hardware IRQ or not. this can be done by the following routine:

```
 push eax          ; save the registers that will be modified

 pushf             ; the order of PUSHs is important!

 mov al,0Bh        ; we'll read in the in-service registers

 out 20h,al        ; master PIC

 out 0A0h,al       ; slave PIC

; there might be needed a short delay here
; however, on my machine it isn't :-)
 in al,20h         ; read INTs being serviced by master PIC

 mov ah,al         ; save for later test

 in al,0A0h        ; read INTs being serviced by slave PIC

 test ax,0FFFFh    ; was it a hardware int?

 jnz original      ;

 popf              ; restore flags

 pop eax           ; the general handler doesn't expect it

jmp offset 1B70h  ; this has to be the general handler (this offset ; is valid in v2.80) original:

 mov ax,8          ; the first two instructions of the original handler

 popf              ; were push eax, mov ax,8, thus we won't pop eax

jmp offset 1A77h  ; this has to be the original handler's offset ; plus 5 bytes (the length of push eax, mov ax,8)
```

and at the beginning of the original handler (offset 0x1A72) we put: jmp offset 1DD7h ; and since it's only 3 bytes long, ; we get 2 spare bytes :-)

the binary patches follow:
offset old new
00001A72: 66 E9
00001A73: 50 62

00001A74: B8 03

00001DD5: 50 EB

00001DD6: 51 60
00001DD7: B9 66
00001DD8: 03 50
00001DD9: 00 9C
00001DDA: B0 B0
00001DDB: 03 0B
00001DDC: E6 E6
00001DDD: 61 20
00001DDE: 51 E6
00001DDF: 33 A0
00001DE0: C9 E4
00001DE1: E2 20
00001DE2: FE 8A
00001DE3: E2 E0
00001DE4: FE E4
00001DE5: E2 A0
00001DE6: FE A9
00001DE7: E2 FF
00001DE8: FE FF
00001DE9: E2 B8
00001DEA: FE 08
00001DEB: E2 00
00001DEC: FE 75
00001DED: E2 06
00001DEE: FE 9D
00001DEF: E2 66
00001DF0: FE 58
00001DF1: E2 E9
00001DF2: FE 7C
00001DF3: E2 FD
00001DF4: FE 9D
00001DF5: E2 E9
00001DF6: FE 7F
00001DF7: E2 FC

A. to solve this problem we should do a complete disassembly of SOFT-ICE since we have to keep track of the base and size of the V86 mode interrupt table, and this requires too many changes to be worth to do it with simple byte patches.

B. a somewhat lame (but it's more than nothing) solution is the following: we chain in a P RET command after GENINT by patching the jump at the end of the GENINT handler to the beginning of P RET. after executing GENINT we'll land at the IRET of our handler and a further P or T will take us back to the original instruction we were at. i said it was somewhat lame... but it works :-)

offset   old new
000118BF: BC 00
000118C0: E0 C7

C. before executing anything call up SOFT-ICE and change the offset of the V86 mode INT3 handler to the one of the INT1 handler.

D. !!! SURPRISE !!!

while you're in WINICE for Win95 v3.0 or 3.01 type in the following command: "ver ice" and see what you get... however, note that due to a bug (or feature?) you can use only once this command during a session, you have to restart WINICE to be able to get the message again.

and while we're at undocumented features, try out "ver ?" as well. in the next release, i'll try to write a detailed description of the new commands (not that if it were that hard to find out...)

unsolved mysteries :-)

1. even the Nu-Mega docs tell about a problem when SOFT-ICE for DOS is loaded from the command line: if HIMEM.SYS is installed the machine simply reboots. i tracked down the problem and found out that the processor resets itself because of a triple fault. it happens so:

after preparing the IDT and GDT (and loading IDTR and GDTR), paging will be enabled in CR0, and then DS and ES will be loaded a descriptor offset of 8. however, both this descriptor and the IDT seem to be invalid, and this leads to a triple fault. unfortunately, i couldn't find out what goes wrong during the setup of IDT and GDT, perhaps someone else out there will do the dirty job :-)... and maybe Nu-Mega will award you with a free, legal version :-)).

2. on a Cyrix 486DLC-40 system (both with and without a coproc) SOFT-ICE gets a Page Fault and halts if the processor is running at 40 MHz, but works fine at 33 MHz. the Page Fault seems to happen while the code window is being put out (i.e. during the execution of the wc command). this is nonsense! so far, no solution... and yes, it's another dirty job...

P.S. for everyone

perhaps there's someone out there who didn't know it so far... WINICE can be used without Window$ (and you'll be able to debug programs that use some DOS extender, if it can make use of DPMI, but that's the case with the most popular ones, e.g. DOS4GW or PMODE/W)! the trick is that you have to make a 'crippled' version of Window$, i.e. make Window$ start without its GUI. a complete description of this can be found at the following URL:

http://www.fys.ruu.nl/~faber/Windows_No_GUI

after creating this GUI-less Window$ all you have to do is to start WINICE (beware, your normal Window$ shouldn't be on your PATH!) and voila, you'll be in a DOS session (i hope that you could find it out yourself that you had to start COMMAND.COM as your KRNL386.EXE...) with WINICE being able to pop up whenever you need it (of course, for native Window$ programs you'll need a full Window$).

and if we're at Window$ here's another trick: if you don't want to see the logo being shown every time you start Window$, either start WIN.COM with a command line paramater of ':' i.e.

WIN.COM :

or (as the above method doesn't work with our GUIless version since WINICE doesn't seem to pass any parameters or you are too lazy to type in every time 2 more characters :-) look for the string 'LOGO' in WIN.COM and change it to something else (it's enough to change only one bit). note, that the WIN.COM of Window$ for Workgroups doesn't like this patch...

P.S. for protection writers

i know that some of the above modifications can be defeated (i could do it myself) however i won't make your life easier... i hope you understand why :-)

The Owl 1997

SICE-VAL.MS

cut from here ------

```
# OLD_SI: 'FG'
# OLD_DI: 'JM'


# OLD_SI
!:p=\71\70
# OLD_DI
!:q=\77\74


# NEW_SI
!:x=\71\68
# NEW_DI
!:y=\77\74
```

---------- until here

SICE2NEW.MS

cut from here ------

```
!binary
!include sice-val.ms


# mov si,OLD_SI -> mov si,NEW_SI
\190:p
\190:x
# mov di,OLD_DI -> mov di,NEW_DI
\191:q
\191:y


# cmp si,OLD_SI -> cmp si,NEW_SI
\129\254:p
\129\254:x
# cmp di,OLD_DI -> cmp di,NEW_DI
\129\255:q
\129\255:y


# cmp w,[bp+4],OLD_SI -> cmp w,[bp+4],NEW_SI
\129\126\4:p
\129\126\4:x
# cmp w,[bp+0],OLD_DI -> cmp w,[bp+0],NEW_DI
\129\126\0:q
\129\126\0:y
```

mov w,[bp+4],OLD_SI -> mov w,[bp+4],NEW_SI
\199\70\4:p
\199\70\4:x
mov w,[bp+0],OLD_DI -> mov w,[bp+0],NEW_DI
\199\70\0:q
\199\70\0:y

not sure about these ones...
cmp w,[4],OLD_SI -> cmp w,[4],NEW_SI
\129\62\4\0:p
\129\62\4\0:x

mov ax,OLD_SI -> mov ax,NEW_SI
\184:p
\184:x
---------- until here

You are deep inside fravia's page of reverse engineering, choose your way out:

[Picture] project 2
[Picture] homepage[Picture] links red anonymity [Picture] +ORC [Picture] students' essays
[Picture] tools [Picture] antismut [Picture] cocktails [Picture] search_forms [Picture] mail_fravia
[Picture] Is reverse engineering illegal?