

## Re: Reverse engineering != piracy (was Re: RosAsm disassembler output vs. IDA Pro)

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-01/1356.html>

---

**From:** Gerhard W. Gruber (*sparhawk\_at\_gmx.at*)

**Date:** 01/28/04

Date: Wed, 28 Jan 2004 22:16:14 +0100

On Wed, 28 Jan 2004 03:18:07 GMT wrote "Randall Hyde"  
<randyhyde@earthlink.net> in alt.lang.asm with  
<PRFRb.30425\$zj7.13444@newsread1.news.pas.earthlink.net>

>*Rene continues to believe that an automatic disassembler is a perfectly  
>reasonable thing and that he will succeed, even though the concept is  
>a proven mathematical impossibility (it violates the "halting theorem").*

I know why it is impossible. One point you already mentioned are these, after compilation, hard coded sizeof() statements. Of course all the other equates suffer from the same problem.

A code which will look like this:

```
#define ER_NO_ACCESS 1
#define ER_FILE_NOT_FOUND 2

...
#define MENU_LOAD_FILE 1
#define MENU_SAVE_AS 2

SaveFile()
{
    int rc = 0;

    if(item == MENU_SAVE_AS)
    {
        if((open() == -1)
            {
                rc = ER_FILE_NOT_FOUND;
                goto Quit;
            }
    }
}

...
```

alt.lang.asm: Re: Reverse engineering != piracy (was Re: RosAsm disassembler output vs. IDA Pro)

```
    return rc;
}
```

Will translate to something like this:

```
_SomeLabel:
    mov _val1,0

    cmp _val2,2
    jne _LocalLabel1

    call _somefunction
    cmp eax, -1
    jne _LocalLabel1

    mov _val1, 2
    jmp _LocalLabel1
```

...

```
_LocalLabel1:
    mov eax, _val1
    ret
```

A disassembler can NEVER determine what 2 originally was and the programmer has to decide if the two 2s are the same 2 or different ones, by analyzing the logic behind it. Just from looking at the code, you simply can not determine what the values mean or how to relate to each other, if they relate at all.

I don't think that the "Halting Problem" is really a proof for a disassembler. Of course, with self modifying code this would be a problem, but lets assume that the majority of potentially to be disassembled code will just be compiler generated, which means that you could IN THEORY successfully determine all code and data objects.

Of course compilers generate dynamic code like using jump tables via registers, which means that a moderate successfull disassembler will have to have an idea what a register is currently used for to recognize such things.

Table:

```
ADDR Fkt1
ADDR Fkt2
ADDR Fkt3
```

Value: dw xxx

Like:

```
mov eax, OFFSET Table
shl ecx, 2
mov edx, [eax+ecx]
jmp edx
```

alt.lang.asm: Re: Reverse engineering != piracy (was Re: RosAsm disassembler output vs. IDA Pro)

In this example, I wonder how the disassembler, supposing that he recognices the table successfully, will determine how many entries of the table are really jump addresses and where normal data continues (like Value which is no part of the Table anymore).

If the disassembler doesn't recognize such a usage, then code which is activated ONLY by the table will never be recognized as code and will falsly end up as data in the final disassembly. I guess there are more complicated examples then these simple ones.

I could think that another thing which might confuse diassembler and hide the code (I have to test this with IDA Pro).

```
push offset LowFkt
push offset HighFkt
ret
```

Actually it's quite interesting to think about such problems of a decompiler and how this could be solved. :)

```
--
Gerhard Gruber
Maintainer of
SoftICE for Linux - http://pice.sourceforge.net/
Fast application launcher - http://sourceforge.net/projects/launchmenu
```