How to easily identify Miracl functions

by bLaCk-eye

THEORY:

This is a small theoretical article describing an idea that could be very useful to crypto cracker out there.

Miracl big number library is one of the most used libraries when it comes to creating protections based on public key algorithms, mostly in crackmes. I have yet to find a commercial software using it as the fee for commercial use is HUGE ~ 1000$. So this little technical paper should come into help of those who want to crack crypto crackmes. All started from one very interesting talk I had with my friend and team colleague, bRain-FaKKer, which described to me a very easy and interesting way to identify the miracle functions that crackme uses. As I didn't found any paper describing it I give full credit for the idea (which is nothing innovative, read down) to bF!.

So what's the idea?

Well, Miracl has a very useful feature: it allows you to find what function caused a crash. Taken from miracl's manual.doc:

"The initial call to **mirsys** also initialises the error tracing system which is integrated with the MIRACL package. Whenever an error is detected the sequence of routine calls down to the routine which generated the error is reported, as well as the error itself. A typical error message might be

```
MIRACL error from routine powltr
     called from isprime
     called from your program
Raising integer to a negative power
```

Such an error report facilitates debugging, and assisted us during the development of these routines. An associated instance variable **TRACER**, initialised to OFF, if set by the user to ON, will cause a trace of the program's progress through the MIRACL routines to be output to the computer screen."

So how do we use this in our favor?

Easy, because miracle must know where the crash happened it uses some constants to define the functions. How do we know that?

Well get miracle package and open from "\Source" almost any file. Here is some code from MrArth3.c:

```
void power(_MIPD_ big x,long n,big z,big w)
{ /* raise big number to int power  w=x^n *
   * (mod z if z and w distinct)          */
    mr_small norm;
```

```
#ifndef MR_GENERIC_MT
    miracl *mr_mip=get_mip();
#endif
    copy(x,mr_mip->w5);
    zero(w);
    if(mr_mip->ERNUM || size(mr_mip->w5)==0) return;
    convert(_MIPP_ 1,w);
    if (n==0L) return;

    MR_IN(17)

    if (n<0L)
    {
        mr_berror(_MIPP_ MR_ERR_NEG_POWER);
        MR_OUT
        return;
    }
     //rest of the code from file
}
```

The code that sets the variable to the function's name is the bolded one. As you see it isn't preceded and followed by any conditional compilation settings. This means that whatever options you choose when building the library, MR_IN(17) will always be in the final library.

How does this get disassembled:

```
mov     dword ptr [esi+eax*4+20h], 11h
```

as 11h = 17 decimal;

You want to identify a miracle function? Easy go inside it, search for instruction like this:

```
mov     dword ptr [esi+eax*4+20h], XXh
```

Note down XX number and search it in your reference. Or you can use the one I supplied with this little paper.

Having all these magic numbers you can identify all the functions from miracl. The advantage over almost any another way to identify the calls is based on the fact that this numbers are very unlikely to be changed over versions in opposition with code. One disadvantage is that you need to check these functions manually. One idea is to make an IDA idc script that searches for the above instruction types and according to the magic constant change the function name. If you manage to do it pls mail it to me.

That's about all I said to say ;-)

bLaCk-eye

APPENDIX:

NUMBER OF FUNCTIONS:08Eh

```
innum                   equ 01h
otnum                   equ 02h
jack                    equ 03h
normalise               equ 04h
multiply                equ 05h
divide                  equ 06h
incr                    equ 07h
decr                    equ 08h
premult                 equ 09h
subdiv                  equ 0Ah
fdsize                  equ 0Bh
egcd                    equ 0Ch
cbase                   equ 0Dh
cinnum                  equ 0Eh
cotnum                  equ 0Fh
nroot                   equ 10h
power                   equ 11h
powmod                  equ 12h
bigdig                  equ 13h
bigrand                 equ 14h
nxprime                 equ 15h
isprime                 equ 16h
mirvar                  equ 17h
mad                     equ 18h
multi_inverse           equ 19h
putdig                  equ 1Ah
add                     equ 1Bh
subtract                equ 1Ch
mirsys                  equ 1Dh
xgcd                    equ 1Eh
fpack                   equ 1Fh
dconv                   equ 20h
mr_shift                equ 21h
mround                  equ 22h
fmul                    equ 23h
fdiv                    equ 24h
fadd                    equ 25h
fsub                    equ 26h
fcomp                   equ 27h
fconv                   equ 28h
frecip                  equ 29h
```

```
fpmul           equ 2Ah
fincr           equ 2Bh
;null entry
ftrunc          equ 2Dh
frand           equ 2Eh
sftbit          equ 2Fh
build           equ 30h
logb2           equ 31h
expint          equ 32h
fpower          equ 33h
froot           equ 34h
fpi             equ 35h
fexp            equ 36h
flog            equ 37h
fpowf           equ 38h
ftan            equ 39h
fatan           equ 3Ah
fsin            equ 3Bh
fasin           equ 3Ch
fcos            equ 3Dh
facos           equ 3Eh
ftanh           equ 3Fh
fatanh          equ 40h
fsinh           equ 41h
fasinh          equ 42h
fcosh           equ 43h
facosh          equ 44h
flop            equ 45h
gprime          equ 46h
powltr          equ 47h
fft_mult        equ 48h
crt_init        equ 49h
crt             equ 4Ah
otstr           equ 4Bh
instr           equ 4Ch
cotstr          equ 4Dh
cinstr          equ 4Eh
powmod2         equ 4Fh
prepare_monty   equ 50h
nres            equ 51h
redc            equ 52h
nres_modmult    equ 53h
nres_powmod     equ 54h
nres_moddiv     equ 55h
nres_powltr     equ 56h
divisible       equ 57h
```

```
remain                  equ 58h
fmodulo                 equ 59h
nres_modadd             equ 5Ah
nres_modsub             equ 5Bh
nres_negate             equ 5Ch
ecurve_init             equ 5Dh
ecurve_add              equ 5Eh
ecurve_mult             equ 5Fh
epoint_init             equ 60h
epoint_set              equ 61h
epoint_get              equ 62h
nres_powmod2            equ 63h
nres_sqroot             equ 64h
sqroot                  equ 65h
nres_premult            equ 66h
ecurve_mult2            equ 67h
ecurve_sub              equ 68h
trial_division          equ 69h
nxsafeprime             equ 6Ah
nres_lucas              equ 6Bh
lucas                   equ 6Ch
brick_init              equ 6Dh
pow_brick               equ 6Eh
set_user_function       equ 6Fh
nres_powmodn            equ 70h
powmodn                 equ 71h
ecurve_multn            equ 72h
ebrick_init             equ 73h
mul_brick               equ 74h
epoint_norm             equ 75h
nres_multi_inverse      equ 76h
;null entry
nres_dotprod            equ 78h
epoint_negate           equ 79h
ecurve_multi_add        equ 7Ah
ecurve2_init            equ 7Bh
epoint2_init            equ 7Ch
epoint2_set             equ 7Dh
epoint2_norm            equ 7Eh
epoint2_get             equ 7Fh
epoint2_comp            equ 80h
ecurve2_add             equ 81h
epoint2_negate          equ 82h
ecurve2_sub             equ 83h
ecurve2_multi_add       equ 84h
ecurve2_mult            equ 85h
```

```
ecurve2_multn           equ 86h
ecurve2_mult2           equ 87h
ebrick2_init            equ 88h
mul2_brick              equ 89h
prepare_basis           equ 8Ah
strong_bigrand          equ 8Bh
bytes_to_big            equ 8Ch
big_to_bytes            equ 8Dh
set_io_buffer_size      equ 8Eh
```