REVERSE CODING

REVERSE CODING

# Released by Cybnet Security Group
# legalz: modify and use at will, if you make any changes, improvements, updates or use the code
# in another project, please send us what you did and give credit
# if you have any questions, post them at forum.hackerthreads.net
# be sure to check out hackerthreads.org for updates and new tutorials/downloads

Index:

1.Introduction
2.Disclaimer
3.Hexadecimal
4.RAM and ROM
5.ASM
6.Needed programs
7.Cracking
8.Conclusion

---Introduction----------

Welcome to my Reverse Coding tutorial! In this paper, you will
learn how to crack and modify your own software. I'll try to get
into as much detail as possible, yet also dumb it down a bit. =)

---Disclaimer------------

All information is purely for educational purposes only! The author
cannot be held responsible for any (ab)use of this information.
USE AT YOUR OWN RISK!!!

---Hexadecimal----------

To begin, I'm going to teach you about hexadecimal, so if you already
know it, then move on. Even if you do already know it, I suggest
sticking around for a refreshment of your memory.=)

Hexadecimal, or hex as it's more commonly known, is a base 16
numbering system. Base 16 meaning that it consists of 16 numbers:
0-9 and A-F. Each of these numbers (A-F=10-16) have a value of 4 bits

and are also called nibbles. In representing a hexadecimal number, one
would write an "0x" before the actual bit set. 0x is simply a tag put
before a hex number to let programmers know that it is in fact, hex.
When writing hex, you will not need to use this prefix.

If you haven't already noticed, the 0x prefix looks similar to that of exponential
notation. Actually this is where 0x has been derived, seeing as how
hex is simply a number that has been raised to a power of 16.
This means 10 in hexadecimal represents the value 16+0, or 16. So check
out this example:

0xB3 (hex)= 2*16(squared)+11*16(to the 1st power)+3*16(to the power of 0)
=2*256+11*16+3=691 (decimal)

Yeah, you could do all of that, or you could be lazy and use an automated
program that does it all for you. Why do you need to know hex? Because
it's used by every piece of software and hardware. How? Memory based address
allocation. Here's an example:

When you clicked on your browsers icon to launch it, the click triggered a "call"
(an asm function that will be discussed more in depth in later chapters.) which
went back to the programs memory with the "click in it's hand." It finds the
address where the code is that makes the program launch and executes it. The
address is written in, you guessed it, hex. An example of an address would be
something like this:

101c5018

5108 would be the actual specific address and 101c would be the sector
of RAM were the address is located. Those are the basics of Hexadecimal
You should probley read this chapter againbecause getting a firm grasp on hex
is essential to cracking and moding programs.

---RAM and ROM--------

In this section we are gonna learn about RAM and ROM. Many people kno about
the hardware part of RAM and ROM and that's gonna be very useful to you......
just not in this tutorial. =) We are about to learn about the "software" side. I use the
term software loosly in that software tends to have a GUI (Graphical User Interface)
and this does not. BUT, there are ways to access and modify the behavior of it that
I will talk about in this chapter, as well as in the next. To start off, I'll answer some
common questions:

What is RAM?

RAM (Random Access Memory) is basically memory and the process of accessing it.
The term "Random Access Memory" was appropriety given to this memory unit because
when executing a command, the CPU doesn't have to scroll through all the memory on
your PC until it finds the right address. It "randomly" whips out the addy from it's back
pocket and serves it up.This process is both quick and efficient. Learning this process
will help you understand the ASM functions in the next chapter.

How does RAM work?

When a command is issued and the memory is pulled from file, it must first go through what is called a "vector". A vector is a "gateway" or a "sector" of RAM where the address of the function is stored with others of it's own kind. An example of a vector would be something like this:

8c0000b4-8c00ffff

This means that all "addressii" (hehe) that are between those values are stored in that sector of RAM. A vector acts as a gateway in that, first, pass through a vector to get to address. Your average program probley has about 30 to 40 main vectors, sectioning off from boot until exit. Knowing the vector of an addy or a function will greatly reduce your headache when you start searching for it.

ROM. ROM is a part of memory that doesn't change. (Although we can change it.=)) Boot ROM for instance, follows the same plan of action it is called upon. ROM also has vectors, just like RAM. ROM is not that important when it comes to cracking to we will leave it alone for now.

Back to RAM. Believe it or not, but addressii (there I go again, I'm such a g33k.) actually follow certain formats or syntax's for certain functions. Take hot keys for example: In the under ground, we call them "Joker commands". By pressing a certain combonation of keys, a program will run, close, be stupid, whatever. The syntax for a Joker command is as follows:

0d-aaaaaf
000zvvvv

Let's examine this format a little closer.

0d= The proclemation of a specifyed format

aaaaa= The address of the function

f= The float or remainder; "Floating point number" ; decimal

000= "NOP" No operation

z= The "Booleon" as we the C++ programmers call it. A booleon is an "IF, THEN" statement. "IF this is true, THEN do this." Value 0= equal; 1= different; 2=less than; 3=greater than.

vvvv= The combonation of hex values (The values of the keys pressed) used to execute the "CALL"

Say the "A" key had a vlaue of fffb and the "B" key has a vlaue of fffd. You would then add both values using a hex calculator and get fff9 as the sum. The output on you calculator would show 1fff8. Add the first value and the last value to find the fourth byte segment. So say we've found the address of the Joker function (usually in the boot ROM sector) commonly called the "Maple address" and we are ready to program in some hex code. Our code may look like this:

0d7ae671
0000fff9

This means that IF the value of fff9 (A and B) is equal (0) to the address (aaaaf) of the function, THEN execute it. See? Easy isn't it? You'll need to know things like this when modding programs

as a use of executing of your arbitrary code in certain parts of your program at a certain time.
Joker commands are also reversable in that if you enter the same code except with a 1,2, or 3,
in the z slot and by changing the button combonations. Reversable meaning terminating the
function or other functions that were started. A good use for this is for firewalls and babysitting
programs. Are you on a college machine and can't download stuff because of that pesky firewall?
Crack it open and program in some Joker commands so you can turn it on and off at will
WITHOUT the administrator's password!

---ASM----------------------

To start off with our small and to the point ASM section, I'll warn you in advance, after reading this,
you'll need to go take a shower cause this is disgusting! Here we go!

To begin, I'm gonna define for you some functions that you'll be seeing alot of, and be using. Here they are:

.:Hex:.	.:ASM:.	.:MEANING:.
75,0f85	jne	jump if not equal
74,0f84	je	jump is equal
eb	jmp	jump directly to
90	nop	no operation
77,0f87	ja	jump if above
0f86	jna	jump if not above
0f83	jae	jump if above or equal to
0f82	jnae	jump if not above or equal
0f82	jb	jump if below
0f83	jnb	jump is not below
of86	jbe	jump if below or equal
0f87	jnbe	jump if not below or equal
0f8f	jg	jump if greater
0f8e	jng	jump if not greater
0f8d	jge	jump if greater or equal
0f8c	jnge	jump if not greater or equal
0f8c	jl	jump if less
0f8d	jnl	jump if not less
0f8e	jle	jump if less or equal
0f8f	jnle	jump if not less or equal

The easy thing about most of the functions in ASM are that they sound like what they mean.
Jump, means of coarse, to Jump from one thing to another. Example:

"jmp 00401744" would mean to jump directly to the address 00401744 once the code
hits the function.

Let's look at "CALL". Call is a function that is used to "call" a certain task, string, address, whatever.
Take a look at this example:

"Call 0040ccc2" this would of coarse call the address 0040ccc2 and use it. Those are the functions
you'll be using.

The reason why I'm not going into loads of detail in this chapter is because when
cracking software, not an extensive amount of knowledge of ASM is needed. If you want
to know more or need help with something, e-mail me at the address provided at the end of
this tutorial. This chapter wasn't so nasty was it? Nah, it was easy =)

---Needed Programs----------------

The programs you will need are as follows:

WDasm 8.9 or Higher
Hiew 6.1
Softice for win9x v3.24
SubmitWolf(demo)v4.01 (http://www.trellian.com/swolf)
Programming Language (C,C++,Pascal,ASM whatever you would like) Prefably C for this tutorial!
And a brain (no seriously)

---Cracking-----------------------------

Ok, here we go! The first thing you need to do is to open up SoftIce and then swolf32.exe which is the name given to our
target program. Go to the help menu and select register. Here's where your brain will come in, start to look
for how the protection is running by entering some random crap into the blank space. Don't press the OK button yet though.
Instead, press CTRL-D to bring up SoftIce. What we are gonna try to do is define a breakpoint, using BPX hmemcpy.
Hit CTRL-D again and it will bring you back to the program. Click OK on the box and SoftIce will again pop up. Now press F12
and it will bring you to the target program code. Scroll down a few lines and find:

```
:004167D9 8D4C2410 lea ecx, dword ptr {esp+10}--;ecx=the random crap you typed in.
:004167DD 8D94290000000 lea edx, dword ptr {esp+00000090}-;edx=name
:004167E4 51 push ecx
:004167E5 52 push edx
:004167E6 E8B5450100 call 0042ADA0----;this is the call which calculates the serial
:004167EB 83C410 add esp, 00000010--;
:004167EE 85C0 test eax, eax----;and return eax=1 if true (booleon =) )
:004167F0 0F8596000000 jne 0041688C----;jump to registered
:004167F6 8D442408 lea eax, dword ptr {esp+08}
:004167FA 8D8C2488000000 lea ecx, dword ptr {esp+00000088}
:00416801 50 push eax
:00416802 51 push ecx
:00416803 E868470100 call 0042AF70----;this call tests our serial
:00416808 83C408 add esp, 00000008---;
:0041680B 85C0 test eax, eax----;for v3.XX one.
:0041680D 7433 je 00416842;jump is equal
```

The call that we want to focas on is at 004167E6. This call tests wether our serial is for the correct version or not.
Let's trace the call 004ADA0:

```
*Referenced by a CALL at address:
:0042ABFC
:0042ADA 83EC30 sub esp, 00000030
:0042ADA3 55 push ebp
:0042ASA4 56 push esi
```

```
:004ADA5 57 push edi
:0042ADA6 8B7C24444 mov edi, dword ptr {esp+44}--;edi=our fake serial
:004ADAA 85FF test edi, edi
:004ADAC 0F4A7010000 je 0042AF59----;die if empty
:004ADB2 8B6C2440 mov ebp, dword ptr {esp+40}--ebp=our name
:0042ADB6 85ED test ebp, ebp
:004ADB8 0F849B010000 je 0042AF59---;die if empty
:004ADBE 8A07 mov al, byte ptr {edi}--;compare 1st byte of serial with 'p', die
:0042ADC0 3C50 cmp al, 50----;
:0042ADC2 0F8587010000 jne 0042AF4F----;if not equal
:0042ADC8 807F0134 cmp byte ptr {edi+01}, 34--:compare byte of serial with '4'
:004ADCC 750C jne 0042ADDA----;
:0042ADCE C70500C8430000000000 mov dword ptr {0043C800}, 00000000
:0042ADD8 EB1C jmp 0042ADF6
```

As we can see by the above, the code tells us that the first value of our serial will
be 'p' and a cycle of a four byte algorythm. I could go on and on about all of the internals
of all this stuff but that would be going beyond the scope of this tutorial. The idea was to show
how to crack this pro, and thats what I'm going to do. Based on the information I've given you, and the
information that you can deduce from reading the code, I've written a small key generator in C.
If you know C, then you'll be able to tell where i got the algorythms to write it. So here it is:

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
long code=555583,count1,count2;
char name[25],cod[5],type='0';
clrscr();
textcolor(14);
printf("This is a simple key-generator written by k33t of CYBNET Security Group");
printf("=================================================");
text color(10);
printf("SubmitWolf(demo)ver4.1 cracked by k33t");
textcolor(14);
printf("%c%c%c",0x10,0x10,0x10");
textcolor(12);
printf("Yup")
prinf("-November 2002");
prinf(""\n\nSelect Edition PRO(0) or Enterprise(1) (0/1)=");
scanf("%c",&type);
if(type=='1')code=557283;
getchar();
prinf("Enter Registration Name=");
scanf("%[^\n]",name);
for(count1=0;count1<=3;count1++
cod[count1]=name[count1];
for(count=1;count1=3;count1++){
for(count2=0;count2<=3;count2++)
cod[count2]=cod[count2]*(code%100);
code=code/100;
}
for(count1=0;name[count1]>0;count1++);
for(count2=0;count2<=3;count2++)
cod[count2]=cod[count2]^(name[count1]+3);
for=(count1-3;count1>=0;count1--){
code=code+(cod[count1]&0xFF);
if(count1>0)
code=code*0x100;
}
if(code<0)code=-code;
for(;code<10000;) code=code*10;
for(;code>999999;) code=code/10;
printf(Your Serial Number=P%c4-%ld",(type=='1')? 'E':'4'code);
return ;
}
```

Ok! So! An overall conclusion of this code is:

1.First two characters of the serial must be either 'PE' or 'P4'.
2.Multiply every first four characters or our name with every byte of our serial before '-'
3.XOR every four byte with every byte of our name.
4.Convert to positive number if<0.
5.Convert to number between 10000 and 1000000.

Forgive me if this code is buggy as I wrote it very quickly in the little spare time I had.