Orthodox / Team ICU

Unpacking PC Guard (Properly)

For this tutorial we will unpack the target itself. The target can be found at:

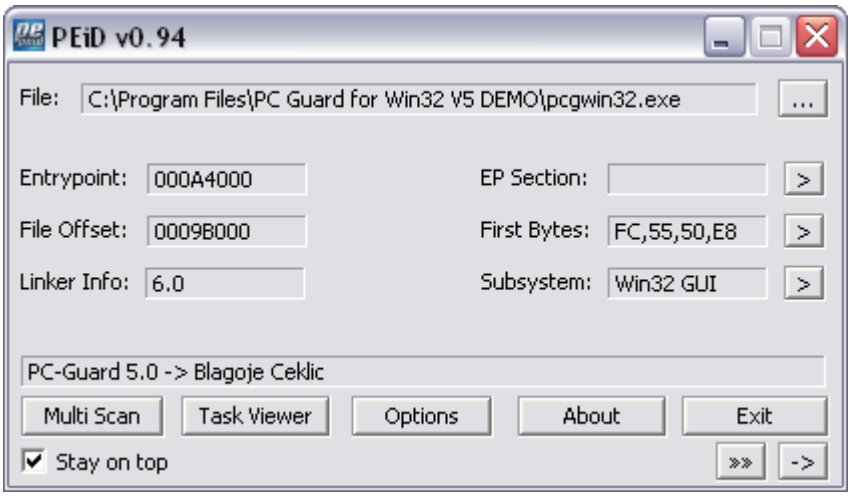
http://www.sofpro.com/

Tools:
-Olly
-Lord PE
-ImportREC

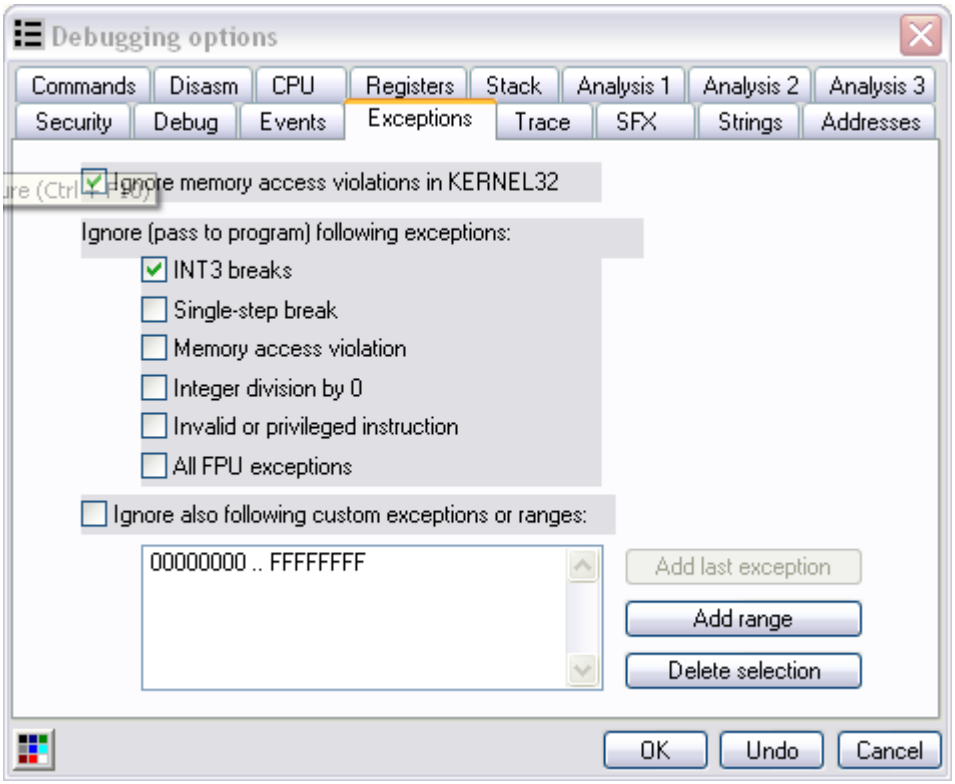
I have read a couple of tutorials regarding PC Guard, and I followed them and found OEP and fixed the imports easily, but when I tried to use my dump it was unusable, the application always crash when I wanted to use some functions. So I traced my dump and found out where application was crashing. It crashed on a call to virtual memory. I tried different options to dump that section and add it to my dump. I also fixed it to a correct offset, but application always crash. So traced original to that call and find out that PC Guard decrypts code on the fly and just before the RET command there is another call that encrypts it again. So to properly dump it we have to trace original and after it decrypts its code we need to binary copy and paste it to our dump and NOP decrypts and encrypts calls.

Enough talking lets start:

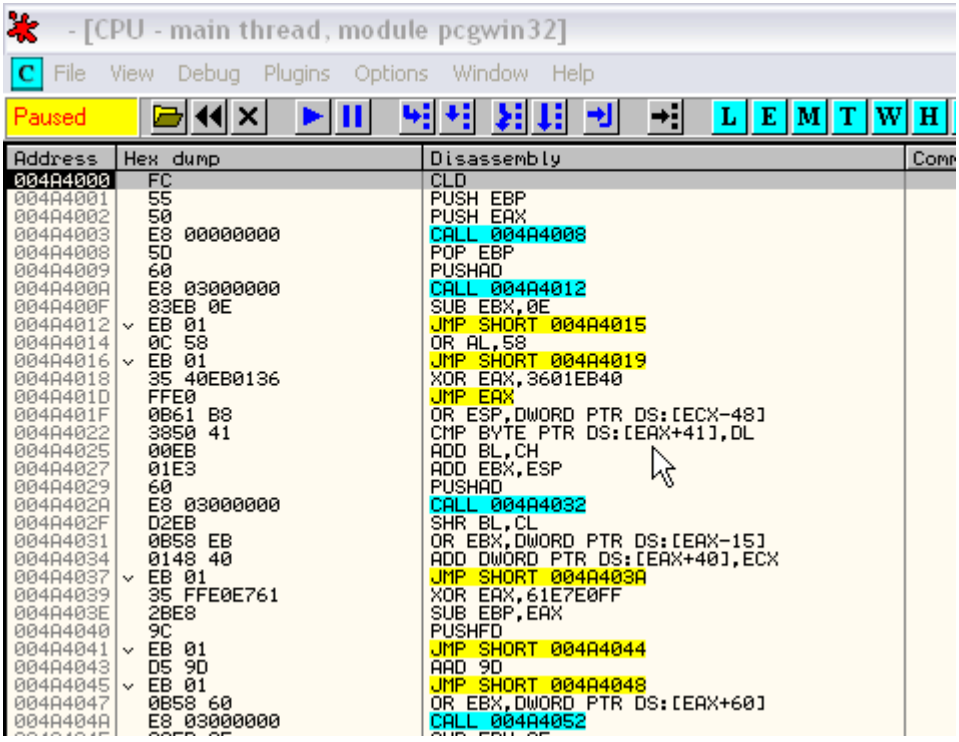
-If we scan application with PE ID we get:



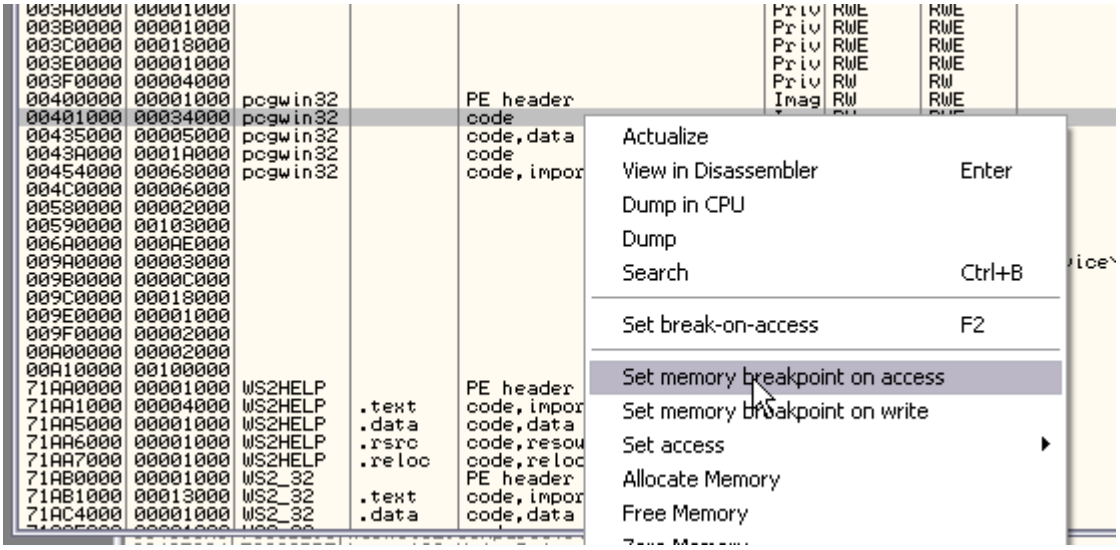
-Set Olly options: Load application in Olly



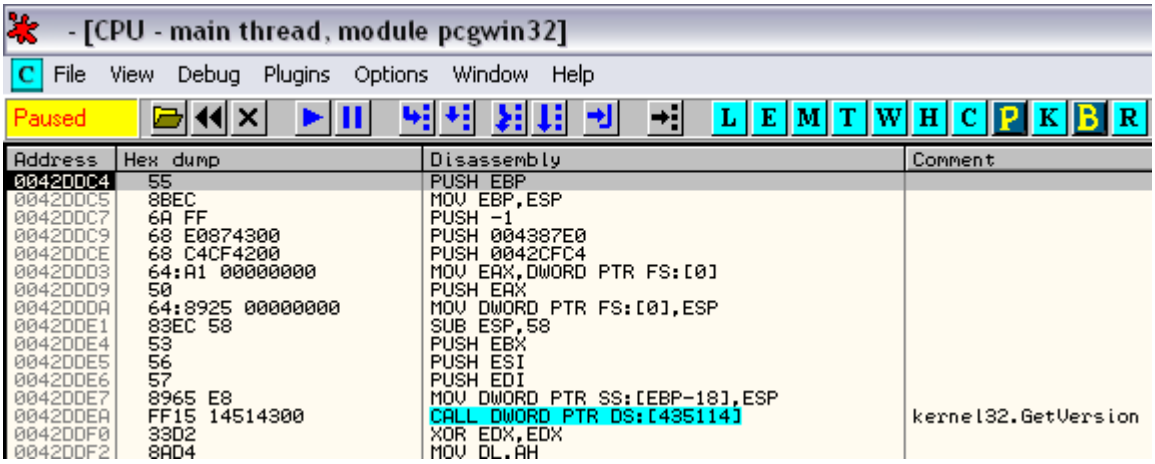
-Load application in Olly and we are on EP of PC Guard;



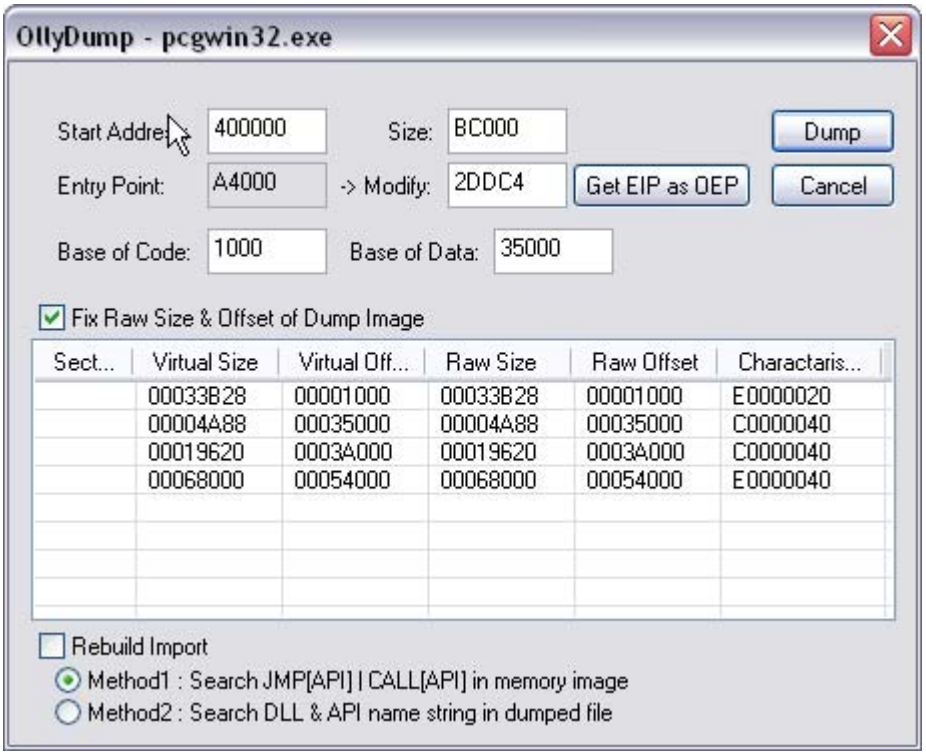
-We will use exceptions to reach OEP. So press Shift+F9 and count exceptions. We pressed 13 times Shift+F9 and application starts. So now count 12 exceptions and go in memory map and on code section set memory break point on access:



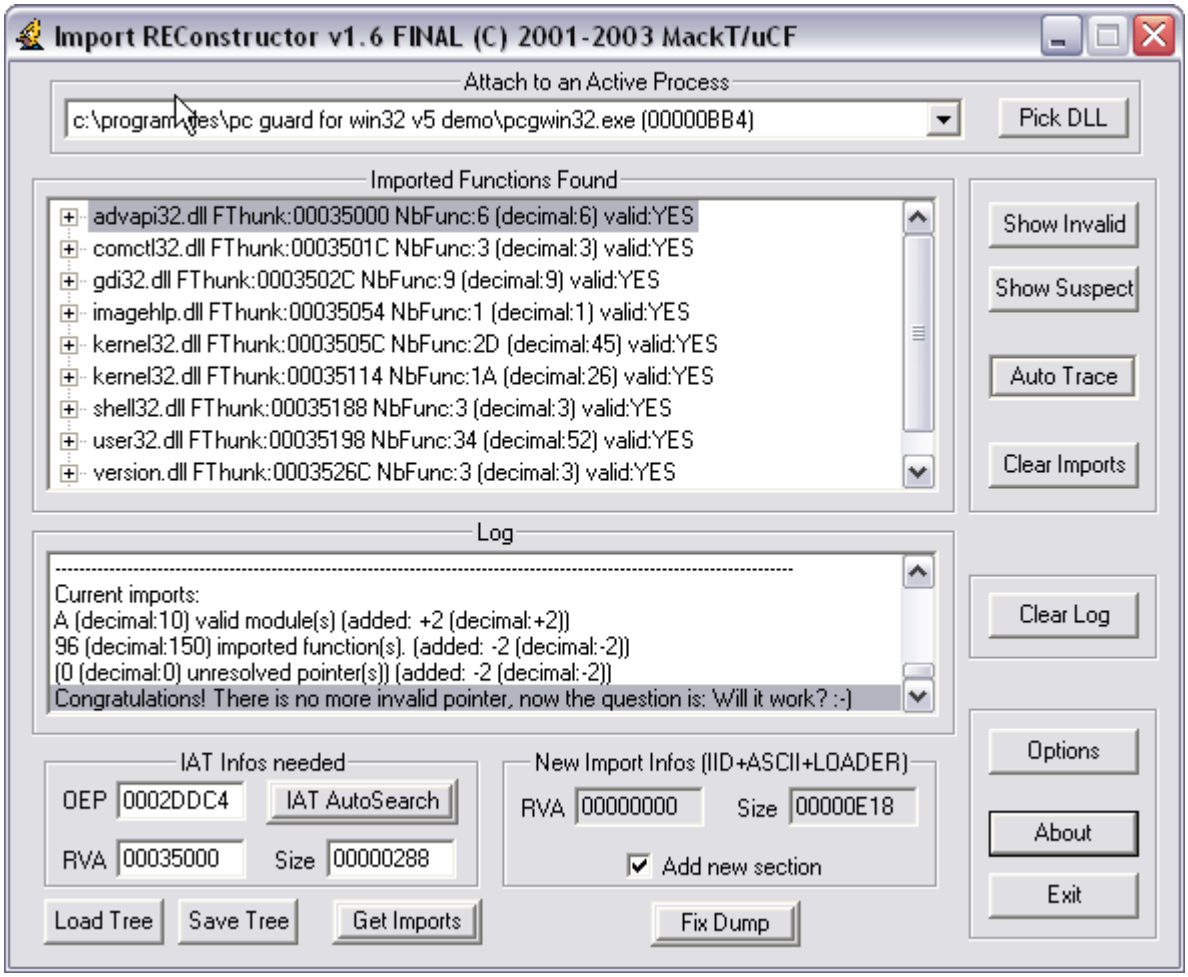
And we are on OEP:



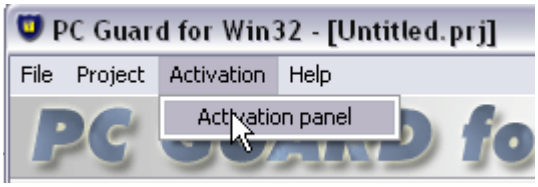
Dump it with Olly dump plug in:



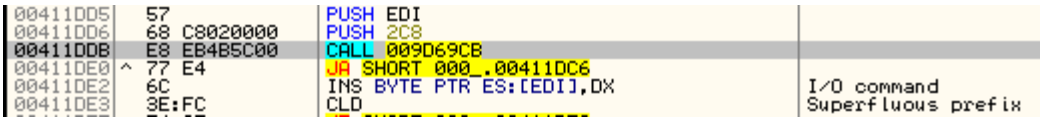
And fix the imports:



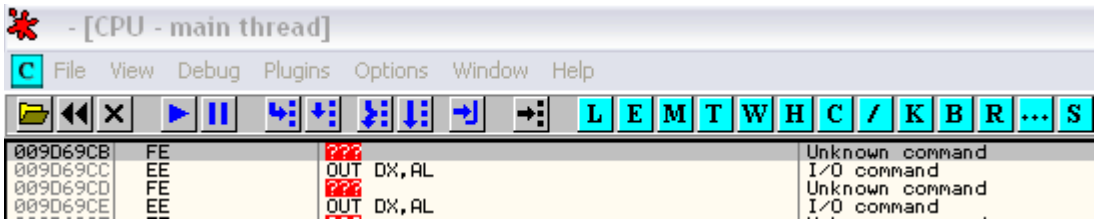
There are 2 invalid which I cut it out. If we try to launch our dump we will see that applications starts, but if we try to use it:



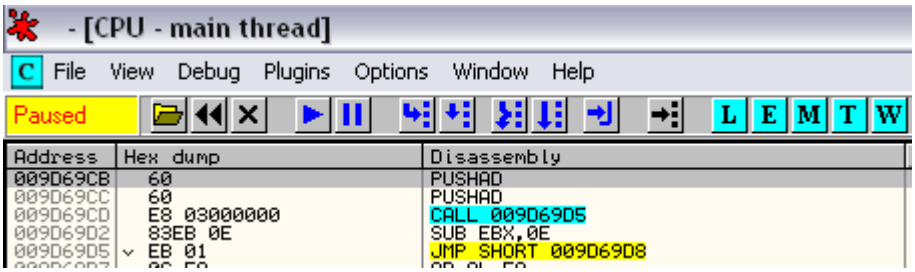
We will see that it's not good and application crashes. After tracing my dump I found the call where it crashes;



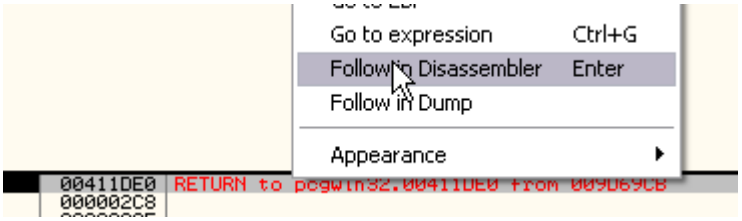
If we follow the call in dump we get;



While in original exe we have;



It points to virtual memory. If we put hardware breakpoint on 009D69CB in original and run application and try again Activation panel we will stop on our breakpoint. Go on stack window and follow in disassmbler:



And we are here;

So put another hardware breakpoint on 00411DE0. Also now ignore all exceptions in Olly options and run application, and we stop at second hardware breakpoint:

```
00411DD5   57                PUSH EDI
00411DD6   68 C8020000       PUSH 2C8
00411DDB   E8 EB4B5C00       CALL 009D69CB
00411DE0   8B35 10524300     MOV ESI,DWORD PTR DS:[435210]   USER32.SetDlgItemTextA
00411DE6   68 EC054500       PUSH 004505EC                   ASCII "Feature 01"
00411DEB   68 5E040000       PUSH 45E
00411DF0   FF35 C8E64400     PUSH DWORD PTR DS:[44E6C8]
00411DF6   FFD6              CALL ESI
00411DF8   68 02064500       PUSH 00450602                   ASCII "Feature 02"
```

We can see that code has changed (It gets decrypted) and we can read it. If we scroll down we will see another call to virtual memory:

```
0041209A   50                PUSH EAX
0041209B   68 6F040000       PUSH 46F
004120A0   FF35 C8E64400     PUSH DWORD PTR DS:[44E6C8]
004120A6   FFD6              CALL ESI
004120A8   68 C8020000       PUSH 2C8
004120AD   E8 BB4A5C00       CALL 009D6B6D
004120B2   5F                POP EDI
004120B3   5E                POP ESI
004120B4   5B                POP EBX
004120B5   C3                RET
```

I tried it and it encrypts the code again. To properly fix our dump we need to binary copy from original when the code gets decrypted (When we are on our second hardware breakpoint). We have to fix all call which means original application has to execute that code in order to get decrypted so we can copy it to our dump.

So when we are on our hardware breakpoint in virtual memory we have to right click on stack and follow in disassembler and put second hardware break point for all these calls:

0040D868 E8 00935C00 CALL 003F6B6D
0040F7D6 E8 9273FEFF CALL 003F6B6D
00411DDB E8 EB4B5C00 CALL 003F69CB
00411BCA E8 FC4D5C00 CALL 003F69CB
0040D916 E8 B090FEFF CALL 003F69CB
0040DCDD E8 E98CFEFF CALL 003F69CB
0040E0B0 E8 1689FEFF CALL 003F69CB
0040E920 E8 A680FEFF CALL 003F69CB
0040EA8B E8 3B7FFEFF CALL 003F69CB
0040D80C E8 BA91FEFF CALL 003F69CB
0040F647 E8 7F73FEFF CALL 003F69CB
0040FD1D E8 A96CFEFF CALL 003F69CB
0040FB8C E8 3A6EFEFF CALL 003F69CB
0040FAE6 E8 E06EFEFF CALL 003F69CB
0040FA42 E8 846FFEFF CALL 003F69CB
0040F36A E8 5C76FEFF CALL 003F69CB
00410000 E8 C669FEFF CALL 003F69CB
0040EFE3 E8 E379FEFF CALL 003F69CB

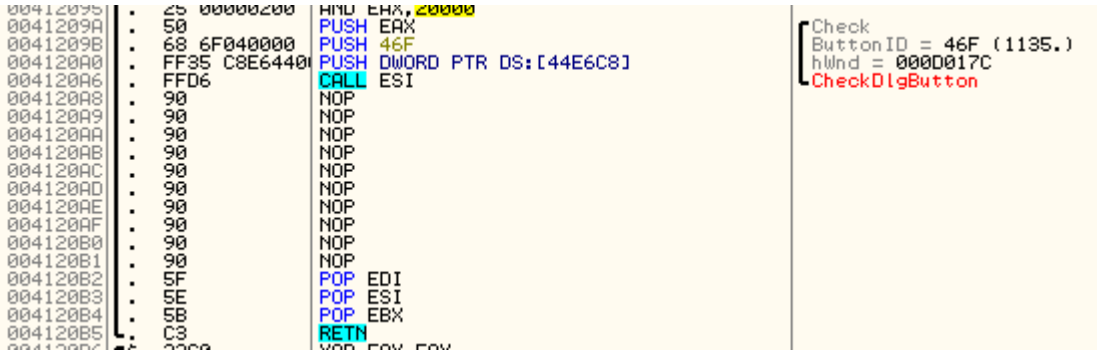
There are also 2 calls that never gets executed. I tried different stuff in original application and couldn't get them executed;

00411BCA E8 FC4D5C00 CALL 003F69CB
004120AD E8 BB4A5C00 CALL 003F69CB

After we copied all code and paste it to our dump bellow decryption code we still need to NOP all decryptions and encryptions call in our dump and also every PUSH command before all decryptions and encryptions call, so we don't have a stack problem in our dump.

Decryption PUSH and CALL that are NOP-ed :

Encryption PUSH and CALL that are NOP-ed :



When you save all changes and run application we can see that runs nicely.

Orthodox / Team ICU 2007

-Greets fly out to all my team mates, TsRh, ARTeam, {RES}, REA (Reverse Engineering Association) and SnD

-Special greets to: ThunderPwr, Ricardo Narvaja, LaFarge, ap0x, deroko, Vrane, lena151 and Blah