

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

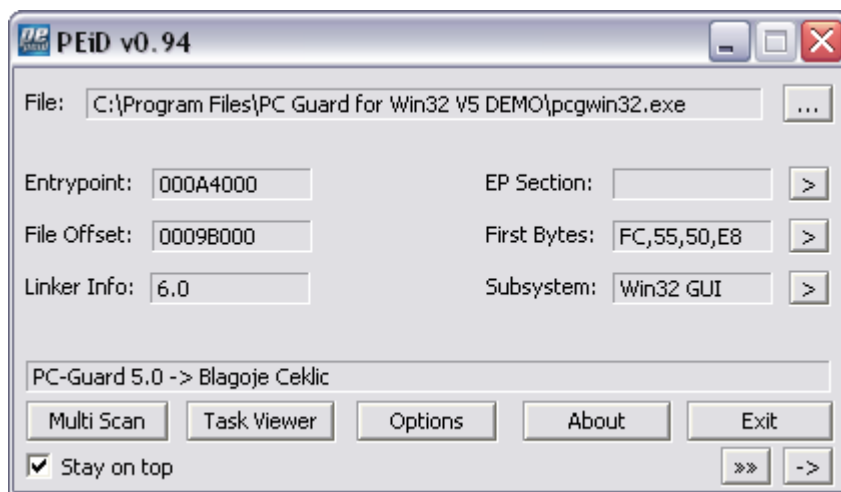
-Lord PE

-ImportREC

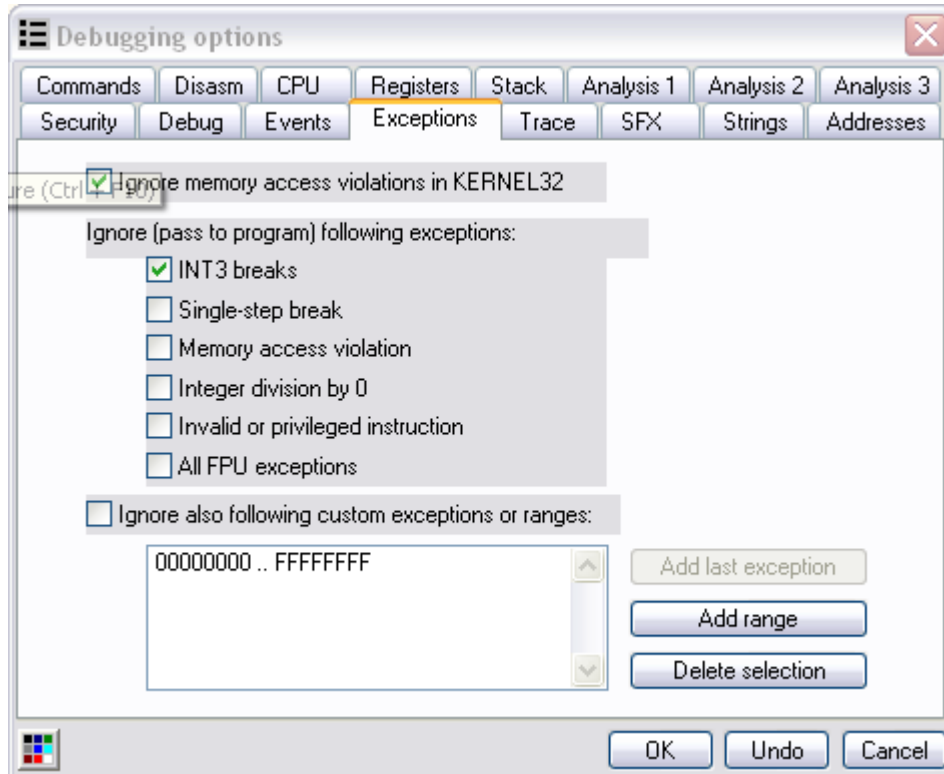
I have read a couple of tutorials regarding PC Guard, and I followed them and found OEP and fixed the imports easily, but when I tried to use my dump it was unusable, the application always crash when I wanted to use some functions. So I traced my dump and found out where application was crashing. It 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:

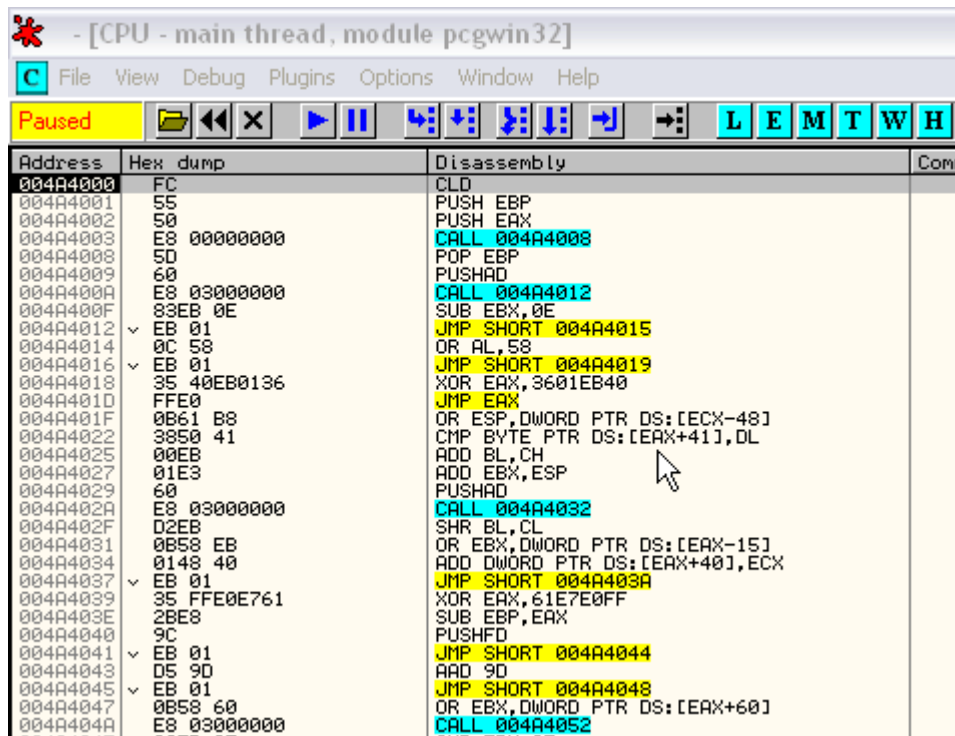
-If we scan application with PE ID we get:



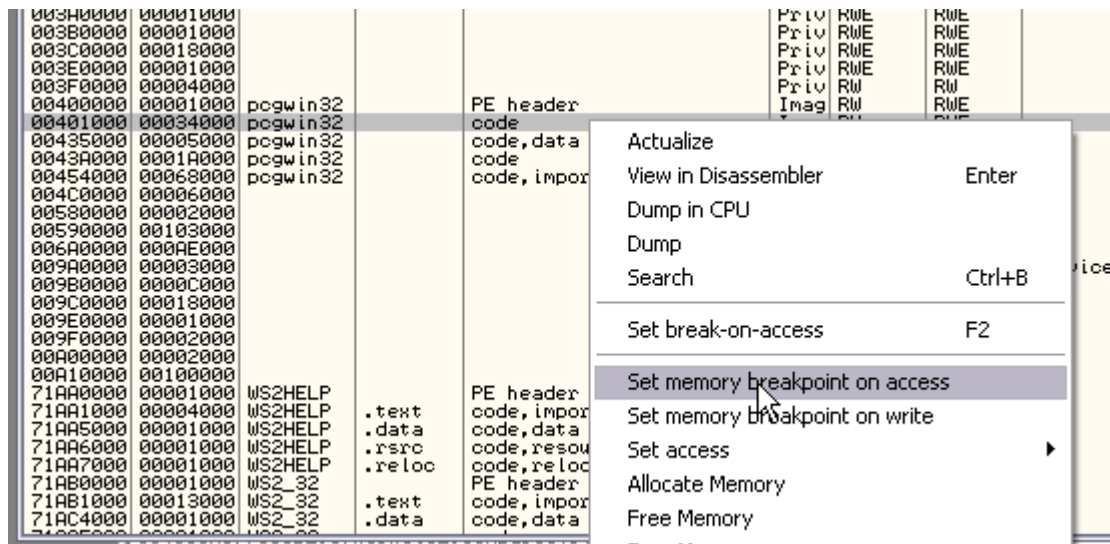
-Set Olly options: Load application in Olly



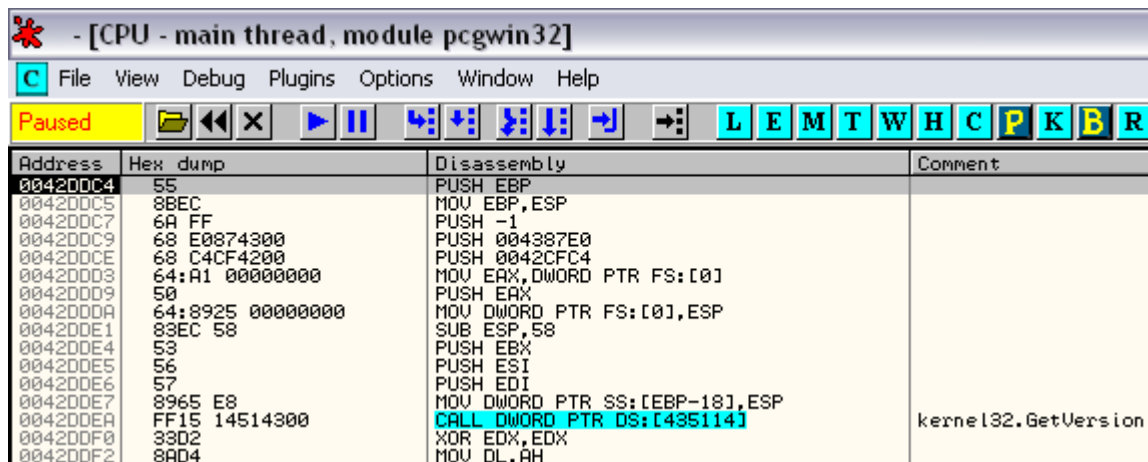
-Load application in Olly and we are on EP of PC Guard;



-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:

OllyDump - pcgwin32.exe

Start Address: 400000 Size: BC000 Dump

Entry Point: A4000 -> Modify: 2DDC4 Get EIP as OEP Cancel

Base of Code: 1000 Base of Data: 35000

☒ Fix Row Size & Offset of Dump Image

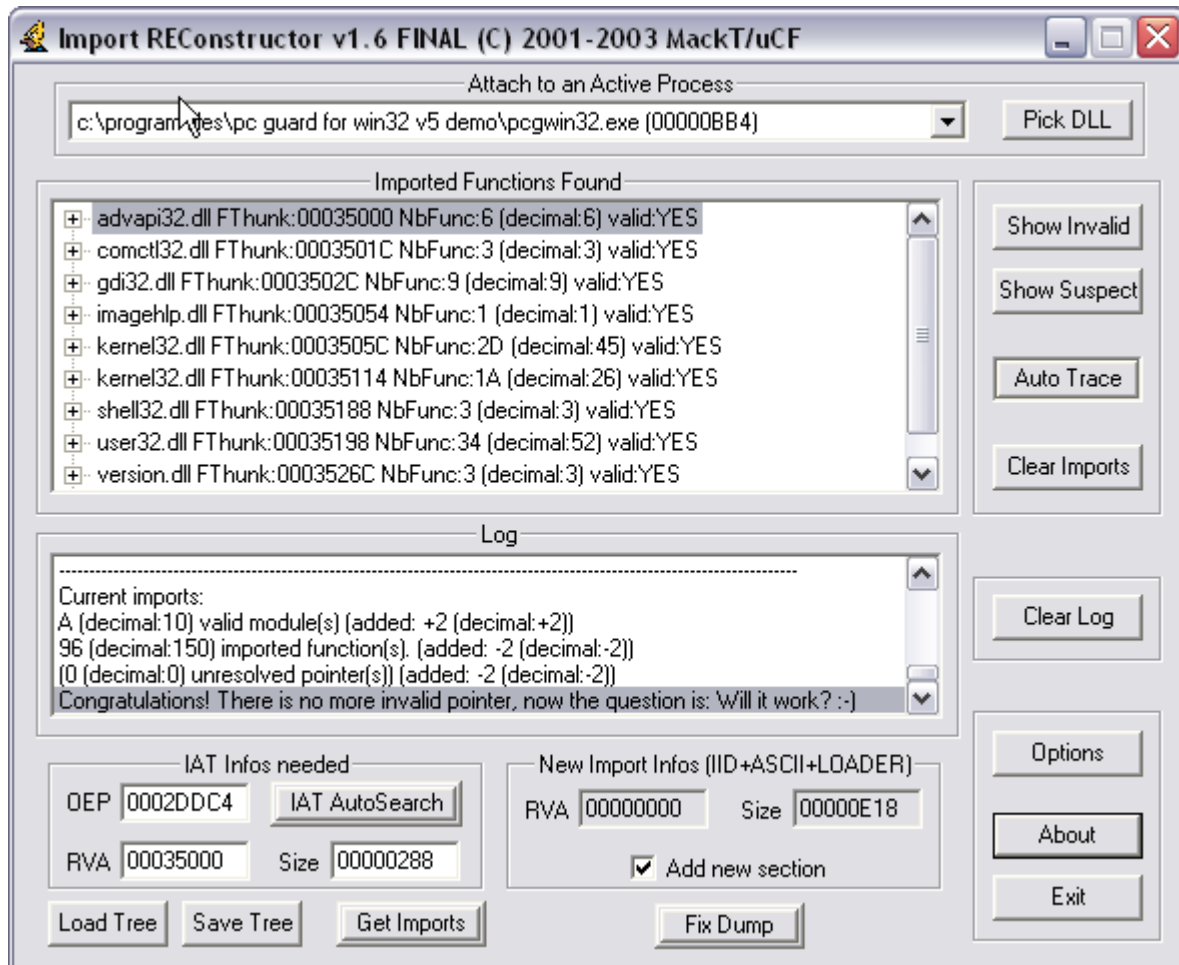
Sect...	Virtual Size	Virtual Off...	Raw Size	Raw Offset	Characteris...
	00033B28	00001000	00033B28	00001000	E0000020
	00004A88	00035000	00004A88	00035000	C0000040
	00019620	0003A000	00019620	0003A000	C0000040
	00068000	00054000	00068000	00054000	E0000040

☐ Rebuild Import

☒ Method1 : Search JMP[API] | CALL[API] in memory image

☐ Method2 : Search DLL & API name string in dumped file

And fix the imports:



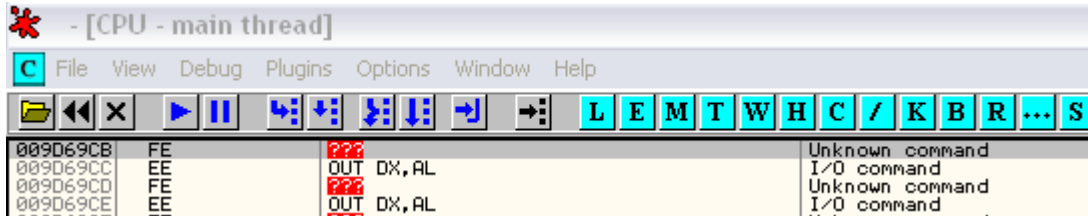
There are 2 invalid which I cut it out. If we try to launch our dump we will see that applications starts, but if we try to use it:



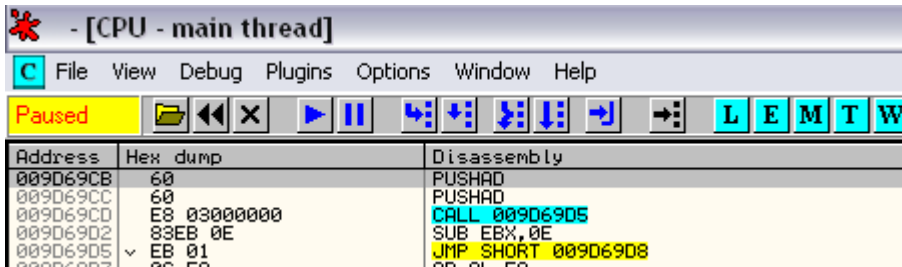
We will see that it's not good and application crashes. After tracing my dump I found the call where it crashes;

004110D5	57	PUSH EDI	
004110D6	68 C8020000	PUSH 2C8	
004110D8	E8 EB4B5C00	CALL 009D069CB	
004110E0	^ 77 E4	JA SHORT 000_004110DC6	
004110E2	6C	INS BYTE PTR ES:[EDI],0X	I/O command
004110E3	3E:FC	CLD	Superfluous prefix

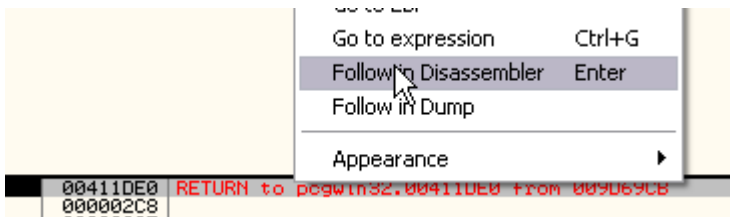
If we follow the call in dump we get;



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 disassembler:



And we are here;

00411DD5	57	PUSH EDI
00411DD6	68 C8020000	PUSH 2C8
00411DD8	E8 EB4B5C00	CALL 009D69CB
00411DE0	77 E4	JR SHORT 00411DC6
00411DE2	6C	INS BYTE PTR ES:[EDI],DX
00411DE3	3E:FC	CLD

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:

00411DD4	35	PUSH ESI	
00411DD5	57	PUSH EDI	
00411DD6	68 C8020000	PUSH 2C8	
00411DD8	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"
00411DFB	74 05	CALL 45F	

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 BB4B5C00	CALL 009D6B6D	
004120B2	5F	POP EDI	
004120B3	5E	POP ESI	
004120B4	5B	POP EBX	
004120B5	C3	RET	
004120B7	3333	CALL EAX	

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 :

00411DD3	53	PUSH EBX	
00411DD4	56	PUSH ESI	
00411DD5	57	PUSH EDI	
00411DD6	90	NOP	
00411DD7	90	NOP	
00411DD8	90	NOP	
00411DD9	90	NOP	
00411DDA	90	NOP	
00411DDB	90	NOP	
00411DDC	90	NOP	
00411DDD	90	NOP	
00411DDE	90	NOP	
00411DDF	90	NOP	
00411DE0	8B35 1052430	MOV ESI,DWORD PTR DS:[<&user32.SetDlgIt	USER32.SetDlgItemTextA
00411DE6	68 EC054500	PUSH pcgwin32.004505EC	Text = "Feature 01"
00411DEB	68 5E040000	PUSH 45E	ControlID = 45E (1118.)
00411DF0	FF35 C8E6440	PUSH DWORD PTR DS:[44E6C8]	hWnd = 0000017C
00411DF6	FFD6	CALL ESI	SetDlgItemTextA
00411DF8	68 02064500	PUSH pcgwin32.00450602	Text = "Feature 02"
00411DFD	68 5F040000	PUSH 45F	ControlID = 45F (1119.)
00411FA2	FF35 C8E6440	PUSH DWORD PTR DS:[44E6C8]	hWnd = 0000017C

Encryption PUSH and CALL that are NOP-ed :

00412095	25 00000200	AND EAX,20000	
0041209A	50	PUSH EAX	
0041209B	68 6F040000	PUSH 46F	Check
004120A0	FF35 C8E6440	PUSH DWORD PTR DS:[44E6C8]	ButtonID = 46F (1135.)
004120A6	FFD6	CALL ESI	hWnd = 0000017C
004120A8	90	NOP	CheckDlgButton
004120A9	90	NOP	
004120AA	90	NOP	
004120AB	90	NOP	
004120AC	90	NOP	
004120AD	90	NOP	
004120AE	90	NOP	
004120AF	90	NOP	
004120B0	90	NOP	
004120B1	90	NOP	
004120B2	5F	POP EDI	
004120B3	5E	POP ESI	
004120B4	5B	POP EBX	
004120B5	C3	RETN	

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

Orthodox / Team ICU 2007

-Greetings 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