



# Special Issue for SecuRom 7.30.0014 Complete Owning

Version 1.0

Last Rev.: September 2007

## Forewords

### Into this Tutorial

1. Some Insights into SecuROM 7.30.0014 by AnonymouS
2. Complete Cracking SecuRom 7.xx by Human
3. SecuROM for the masses by deroko
4. Well deep Inside SecuRom by AnonymouS

After the publication of our first essay on SecuROM I received a lot of interest and replies on this argument, the first essay was contributed by AnonymouS and was just a bit, an insight into SecuROM, a lot of things were still missing, like fixing other anti-debugging and anti-dump methods, fixing redirections and VM. I then organized with deroko, Human and AnonymouS again to write a more complete walkthrough on SecuROM, this time covering all the required issues to successfully own it.

The result is this new Special Issue collecting the contributions of these authors.

This protector is on the scene since time and all the games protected with it have already been cracked then it's time to make these things clearly public so as anyone can understand them.

I am also distributing the tools and scripts used in the tutorials composing this special issue.

But, stop a moment, thinking how many other teams are giving to you so much for free! Not many..

Of course I want to thanks authors here for their time and their tools. SecuROM was a protection which unprotection was kept secret for a long time, now no more ☺

*Have phun,  
Shub*

---

Editor: Shub-Nigurrath



## Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

**All the commercial programs used within this document have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible for damages to the companies holding rights on those programs. The scope of this tutorial as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art and generally the comprehension of what happens under the hood. We are not releasing any cracked application. We are what we know..**

## Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: <http://releases.accessroot.com>

## Table of Contents

Some Insights into SecuROM 7.30.0014 by Anonymous .....	3
1. Forewords.....	3
2. Settings and Target .....	3
2.1. Target .....	3
2.2. Tool Used.....	3
3. Defeating the mysterious debug-detection .....	4
4. Reaching OEP .....	5
5. Defeating the anti-dumping trick. ....	7
6. Conclusion .....	9
7. Final words and greetings.....	9
Complete Cracking SecuRom 7.xx by Human .....	10
1. Foreword and needed tools .....	10
2. First step: start of journey .....	10
3. Second step: prepare things .....	10
4. Third Step: Load the game into Olly and rebase .....	11
5. Fourth Step: daemons tools and OEP .....	13
6. Fifth Step: Fixing Anti-dumps .....	17
7. Sixth Step: fixing the CRCChecks .....	18
8. Seventh Step: taking care of antidumps.....	20
9. Conclusions .....	34
SecuROM for the masses by deroko .....	35
1. Forewords.....	35
2. Tools and Target .....	35
3. Few words about SecuROM .....	36
4. Dumping SecuROM.....	37
5. Anti-Dump fixing .....	48
6. Conclusion .....	55
7. References .....	55
8. Greetings.....	55
Well deep Inside SecuRom by Anonymous .....	56
1. The funny side of things.....	56
2. Code morphing.....	56
3. Basic API redirection.....	57
4. Code splicing .....	58
5. Advanced API redirections .....	59
6. Virtual Machine .....	60

# Some Insights into SecuROM 7.30.0014 by Anonymous

---

## 1. Forewords

It's been a while since I did any reversing. It hasn't been much reversing since the release of the X-Prot v2 unpacker. First of all because I'm lazy, but real life also had a lot going on. Anyway, I have been following the SecuRom thread (<http://forums.accessroot.com/index.php?showtopic=4361&st=0>) on ARTeam's forum for a while, decided to look deeper into SecuRom 7.

Initially I wanted to code an unpacker and started coding unpackers for SecuRom 7.10 through 7.12. As I began coding on an unpacker for SecuRom V7.18 I found out that the task was quite demanding so I abandoned 7.18 and moved on to 7.30.0014.

This small tutorial/essay is not about completely reversing SecuRom 7.30.0014. It's just a help for people on how to reach OEP and on the way defeating the anti-debugger trick that apparently stops a lot of people. I will also show I bypass the anti-dump trick used by SecuRom.

The tutorial/essay is not very explaining as I do think that people reading this will be somehow more than just a newbie reverser. SecuRom is a tough protection and good reversing skills are needed in order to fully reverse this protection.

## 2. Settings and Target

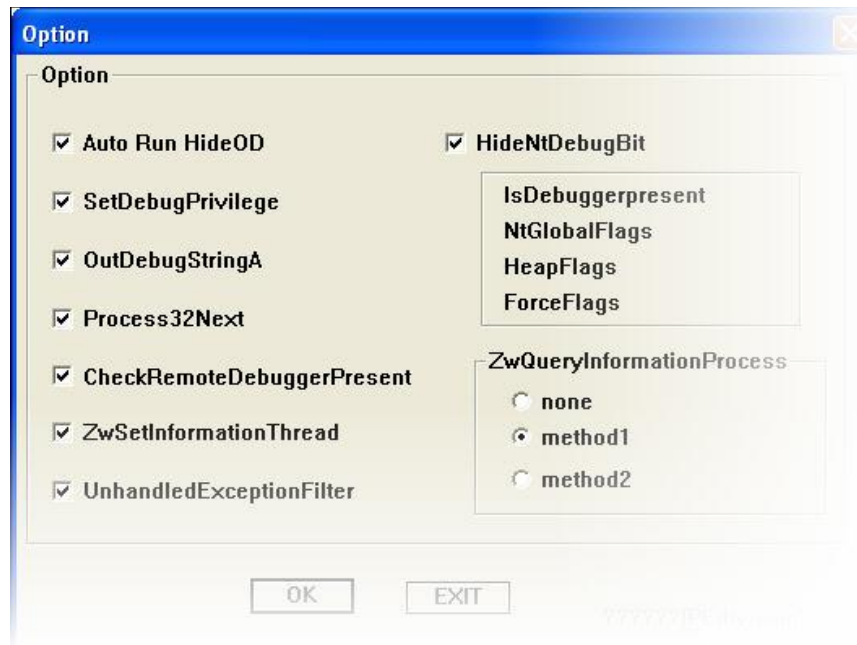
### 2.1. Target

Resident Evil 4 from Capcom

### 2.2. Tool Used

OllyDbg V1.10

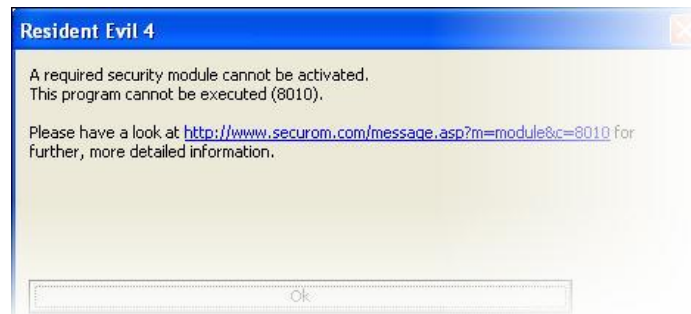
OllyDbg plugin (HideOD) with the following settings:



TaskMngr by drizz (<http://drizz.t35.com/main.php>)

### 3. Defeating the mysterious debug-detection

Okay, let's go to work... Run Olly, select executable and let Olly loose. We hit to exceptions before get this error message:



In the earlier versions I used to get this whenever the ZwQueryObject trick was launched, but when I patch this I still get the error message, so I tend to think it's the mysterious debugger detection people are talking about on the ARTeam thread.

How does it detect us?? We're using HideOD and ZwQueryObject is not the reason, so this must be new debugger detection. Let's start tracing... I will spare you for the agonizing of tracing through huge amount for checksums with SecuRom and let you straight to the answer.



Notice the CMP BYTE PTR DS:[EAX+4],0 ??? It's really a part of a much large procedure... Anyway, modify this by setting TRUE back to FALSE (1 to 0).



To be honest, I'm not quite sure what exactly happen, but take a look at this clean code, stripped from obfuscation, trap-flag protection and checksums:

```

0577F556      FFD5          CALL EBP                <-- call RtlAcquirePebLock
056D89E3      8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]  <-- [EBP+8] == 30h
056D89E6      64:8B00       MOV EAX,DWORD PTR FS:[EAX]    <-- Get PEB address
0577FAA2      8B40 0C       MOV EAX,DWORD PTR DS:[EAX+C]  <-- PPEB_LDR_DATA
0577FDC9      8078 04 00    CMP BYTE PTR DS:[EAX+4],0     <-- TRUE if debugged
0577FDD3      75 12        JNZ SHORT game.0577FDE7      <-- jump bad boy

```

From Microsoft's library I came across this:

```

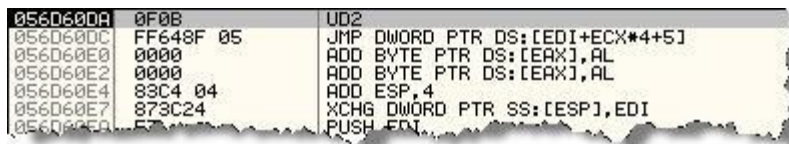
typedef struct _PEB_LDR_DATA {
    BYTE Reserved1[8];
    PVOID Reserved2[3];
    LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA;
*PPEB_LDR_DATA;

```

From what I can see the flag at [EAX+4] is somehow switched on when debugged. I tried coding a little test myself but I always came up with a TRUE result even if I was not running a debugger !?!

## 4. Reaching OEP

Fire up Olly and run through all exceptions, including fixing the anti-debug trick, until you reach:



Then set a conditional BPX on VirtualProtect and run it.



Once Olly breaks clear BPX and suspend all threads except the main one.

Threads						
Ident	Entry	Data block	Last error	Status	Priority	User time
00000C2C	05BCA100	7FFDF000	ERROR_PROC_NOT_FOUND	Active	32 + 0	3.1562
00000F28	7C810659	7FFDE000	ERROR_SUCCESS	Suspended	32 + 0	0.0000

Now set a breakpoint on ReleaseMutex and let Olly run until it breaks again.

CTRL+F9 will lead you to the RETN in ReleaseMutex. Trace back into user-code and you will end up here:

056B1E0B	FF35 50E5B005	PUSH DWORD PTR DS:[5BDE550]
056B1EE1	E8 1BD8FEFF	CALL game.0569FA01
056B1EE6	8325 50E5B005 0	AND DWORD PTR DS:[5BDE550],0
056B1EED	B0 01	MOV AL,1
056B1EEF	C3	RETN
056B1EF0	55	PUSH EBP
056B1EF1	8D6C24 A0	LEA EBP,DWORD PTR SS:[ESP-60]
056B1EF5	81EC 34060000	SUB ESP,634

Now search for the pattern: **C9 87 3C 24**

Enter binary string to search for

ASCII:

UNICODE:

HEX +04:

☒ Entire block

☐ Case sensitive

OK Cancel

Be aware there are more offsets that match this pattern but the first one found should be the "good" one. If this is not the case, you better start tracing ;) Anyway should look something like this:

05682B40	C9	LEAVE
05682B4E	873C24	XCHG DWORD PTR SS:[ESP],EDI
05682B51	57	PUSH EDI
05682B52	877C24 04	XCHG DWORD PTR SS:[ESP+4],EDI
05682B56	C1E0 00	SHL EAX,0
05682B59	C74424 04 C2040	MOV DWORD PTR SS:[ESP+4],4C2
05682B61	EB FA	JMP SHORT game.05682B5D
05682B63	CC	INT3
05682B64	CC	INT3

Place a Hardware breakpoint on the instruction after the LEAVE opcode and run Olly. When Olly breaks go to Memory Map and place a breakpoint code section using F2 and run Olly.

00400000	00001000	game		PE header	Imag 01001002	R	RWE
00401000	005A0000	game	.text	code	Imag 01001002	R	RWE
009A0000	00050000	game	.rdata	code	Imag 01001002	R	RWE
00A03000	04C6D000	game	.data	code,data	Imag 01001002	R	RWE
05670000	00003000	game	.idata		Imag 01001002	R	RWE
05673000	00003000	game	.rsrc	resources	Imag 01001002	R	RWE
05676000	0054B000	game	nunc		Imag 01001002	R	RWE
05BC1000	0000D000	game	bibendum	SFX	Imag 01001002	R	RWE
05BCE000	00234000	game	est		Imag 01001002	R	RWE
05E02000	000F6000	game	.securom	imports	Imag 01001002	R	RWE

Next time Olly breaks we're at OEP.

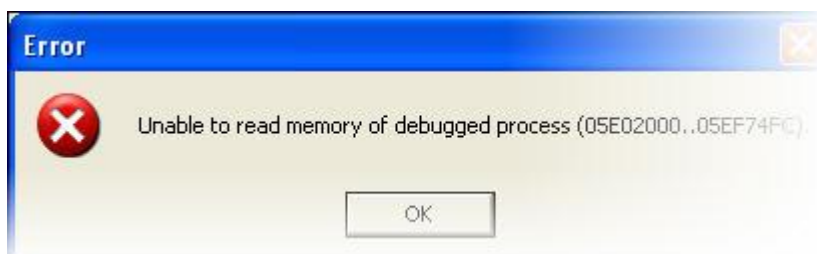


00954C2C	6A 60	PUSH 60	
00954C2E	68 B8549E00	PUSH game.0095488	
00954C30	E8 103C0000	CALL game.0095848	
00954C32	BF 34000000	MOV EDI,94	
00954C34	8BC7	MOV EAX,EDI	
00954C36	E8 0CFFFFFF	CALL game.00954850	
00954C38	965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00954C3A	8BF4	MOV ESI,ESP	
00954C3C	93E	MOV DWORD PTR DS:[ESI],EDI	
00954C3E	56	PUSH ESI	
00954C40	FF15 0C0A6705	CALL DWORD PTR DS:[5670A0C]	kernel32.GetVersionExA
00954C42	8B4E 10	MOV ECX,DWORD PTR DS:[ESI+10]	
00954C44	89D0 74D86605	MOV DWORD PTR DS:[566D874],ECX	
00954C46	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]	
00954C48	A3 80D86605	MOV DWORD PTR DS:[566D880],EAX	
00954C4A	8B56 08	MOV EDX,DWORD PTR DS:[ESI+8]	
00954C4C	8915 84D86605	MOV DWORD PTR DS:[566D884],EDX	
00954C4E	8B76 0C	MOV ESI,DWORD PTR DS:[ESI+C]	
00954C50	81E6 FF7F0000	AND ESI,7FFF	
00954C52	8935 78D86605	MOV DWORD PTR DS:[566D878],ESI	
00954C54	83F9 02	CMPEB ECX,2	
00954C56	74 0C	JE SHORT game.00954C8C	
00954C58	81CE 00800000	OR ESI,8000	
00954C5A	8935 78D86605	MOV DWORD PTR DS:[566D878],ESI	
00954C5C	C1E0 08	SHL EAX,8	
00954C5E	03C2	ADD EAX,EDX	
00954C60	A3 7CD86605	MOV DWORD PTR DS:[566D87C],EAX	
00954C62	33F6	XOR ESI,ESI	
00954C64	56	PUSH ESI	
00954C66	8B3D 80A6705	MOV EDI,DWORD PTR DS:[5670A80]	kernel32.GetModuleHandleA
00954C68	FFD7	CALL EDI	
00954C6A	6618138 4D50	CMPEB PTR DS:[EAX],5A4D	

Now we can dump.... Or can we ???

## 5. Defeating the anti-dumping trick.

As one can see, reaching the OEP of SecuRom 7.30.0014 is fairly easy. However, the authors did another attempt to slow us down. When we fire up our PE-dumper we get this message:



Hmm... What happens here ?? Our dumper won't dump ?! Don't worry... This is easily defeated ... Luckily for you I did all the tracing through SecuRom's checksum hell. Let's rewind time a little bit ;)

As I encountered this I decided to find out where exactly made this anti-dump trick possible. I let the executable run until the first exception and tried to dump. Here I also got the error message... So, this means that the anti-dump trick is setup before the first exception. Now I simple started the slow process of tracing through tons of checksums. First I encountered this:

0023F95C	0023F97C	ASCII "Wrapper - setting guard page at 05E0600h"
0023F960	0567BF57	game.0567BF57
0023F964	0023F97C	ASCII "Wrapper - setting guard page at 05E0600h"
0023F968	0023F97C	ASCII "Wrapper - setting guard page at 05E0600h"
0023F96C	056D8D50	ASCII "Wrapper - setting guard page at 708xh"

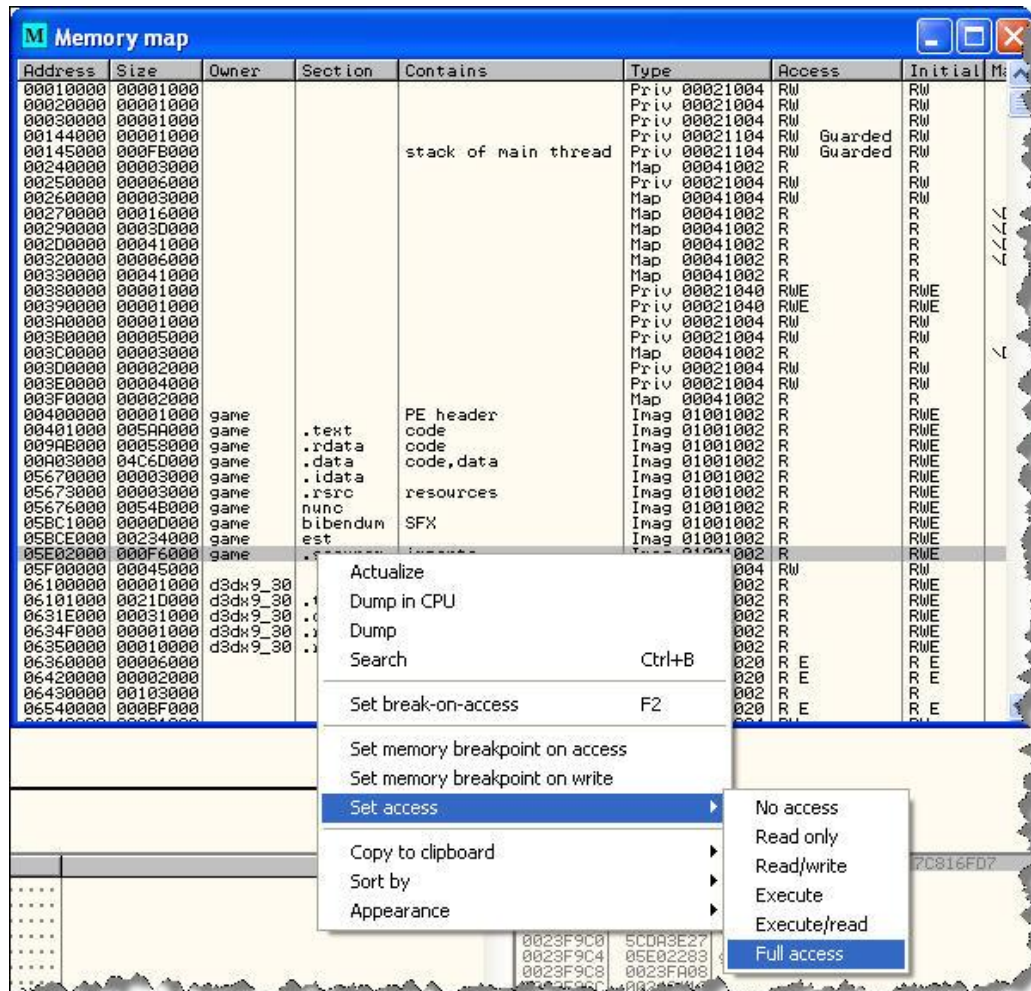
This let me to the thought that SecuRom for some strange reason GUARD PAGE the .securom section. So I set at breakpoint on VirtualProtect and after several hits I ended here:

0023F960	0567C150	CALL to VirtualProtect from game.0567C14A
0023F964	05E06000	Address = game.05E06000
0023F968	00001000	Size = 1000 (4096.)
0023F96C	00000001	NewProtect = PAGE_NOACCESS
0023F970	0023F9FC	OldProtect = 0023F9FC

I then retraced my way back to the user code (CTRL-F9) and ended here:

0567C13F	9D	POPF	
0567C140	870C24	XCHG DWORD PTR SS:[ESP],ECX	
0567C143	8D6424 FC	LEA ESP,DWORD PTR SS:[ESP-4]	
0567C147	891C24	MOV DWORD PTR SS:[ESP],EBX	
0567C14A	FF15 CB8CE005	CALL DWORD PTR DS:[<&KERNEL32.VirtualProtect	kernel32.VirtualProtect
0567C150	83EC 04	SUB ESP,4	
0567C153	90	NOP	

Now we know what to do when reach the OEP, so let's fast forward to the OEP and open up the Memory Map and set the access back to FULL ACCESS on the .securom section:



Now we can finally dump the executable without any problems.





## 6. Conclusion

As one can see it is fairly easy to reach the OEP of SecuRom 7.30.0014. It's also possible to dump the executable once we set back the page access to FULL. As for the mysterious debugger detection, we found it and is able to fix the problem. However, reaching OEP and being able to dump the executable is not enough to defeating SecuRom. The dump is filled with VM code, code-splicings etc. etc. I only showed the way to the promised land, it's now up to you to work your way through it yourself ;)

## 7. Final words and greetings

As stated in the forewords this is not a complete tutorial on how to reverse SecuRom, but merely a quick step-by-step on how to reach the OEP of version 7.30.0014. I must admit that I haven't looked into the VM and code-splicing of this version. I initially kind of promised that I would code another unpacker but I doubt that I will ever code any again. Basically I haven't got the time anymore and the credits you get from your many hours of work are minimal. However, if my good friend, Nacho DJ, talk me into it I might do some more tutorials ;)

Last but not least I would like to send out my greetings to:

- My wife, Kristine and my son Frederik
- All ARTeam member (especially Nacho DJ)
- Drizz
- All I forgot

Sincerely,  
AnonymouS / ARTeam

# Complete Cracking SecuRom 7.xx by Human

---

## 1. Foreword and needed tools

Hello and Welcome to my tut about cracking securom 7.xx (something around 7.30)

What we need..

Target:

- Resident evil 4 or Biohazard 4 ISO
- Resident evil 4 1.1 patch
- Resident evil 4 maxi image(included)

Tools:

- Winhex
- Cff Explorer
- Ollydbg 1.10
- Ollydump plugin
- My oepfind:) newest ofcourse.

And my scripts:

- Securom 7.x Cpuid Fixer (included into this distribution)
- Securom 7.x CRC Check Fixer (included into this distribution)
- Securom 7.x Jump Bridge & Crypted Code Fixer (included into this distribution)

Note: this tutorial is more like unpacking tutorial and not a deep analyze why I do things, I spent a lot of time analyzing this, so if you want understand better.

Do it alone, check how secuRom uses those. With this tut it would be easier for you.

## 2. First step: start of journey

Lets start with installing game.

## 3. Second step: prepare things

Install patch(if you have biohazard you still can install patch after some steps)

First insert Reg file with path to your biohazard 4

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\CAPCOM\resident evil 4]
"PATH"="e:\\Games\\biohazard 4\\"
```

Next lets load patch into ollydbg and patch its complain about no game to update :P

Address	Hex dump	Disassembly	Comment	Registers (FPU)
00407E0C	896424 4C	MOV DWORD PTR SS:[ESP+4C],ESP		ERX 00000000
00407E10	68 30E54200	PUSH up_d_pal1.0042E530	ASCII "Installation destination is	ECX 0012FFB0
00407E15	✓ E8 04C9FFFF	JMP SHORT up_d_pal1.00407E6F		EDX 7C90E894 ntdll.KiFastSystemCallRet
00407E17	85C0	TEST EAX,EAX		EBX 7FFDF000
00407E1C	74 24	JZ SHORT up_d_pal1.00407E44		ESP 0012FFC4
00407E1E	53	PUSH EBX		ESI FFFFFFFF
00407E20	83EC 1C	SUB ESP,1C		EDI 7C910738 ntdll.7C910738
00407E24	8BCC	MOV ECX,ESP		EIP 00423548 up_d_pal1.<ModuleEntryPoint>
00407E25	896424 30	MOV DWORD PTR SS:[ESP+30],ESP		C 0 ES 0023 32bit 0(FFFFFFFF)
00407E2A	68 4FE34200	PUSH up_d_pal1.0042E34F		P 1 CS 001B 32bit 0(FFFFFFFF)
00407E2F	E8 7C86FFFF	CALL up_d_pal1.004024B0		A 0 SS 0023 32bit 0(FFFFFFFF)
00407E34	83EC 1C	SUB ESP,1C		C 1 DS 0023 32bit 0(FFFFFFFF)
00407E37	8BCC	MOV ECX,ESP		C 0 FS 003B 32bit 7FFDE000(FFF)
00407E39	896424 4C	MOV DWORD PTR SS:[ESP+4C],ESP		T 0 GS 0000 NULL
00407E3D	68 10E54200	PUSH up_d_pal1.0042E510		D 0
00407E42	✓ E8 2B	JMP SHORT up_d_pal1.00407E6F	ASCII "already attached even versic	O 0 LastErr ERROR_MOD_NOT_FOUND (0000007E)
00407E44	E8 97C7FFFF	CALL up_d_pal1.004045E0		EFL 00010246 (NO,NG,E,BE,NS,PE,GE,LE)
00407E49	85C0	TEST EAX,EAX		ST0 empty UNORM D0A8 01050104 00000000
00407E4B	✓ 75 5E	JNZ SHORT up_d_pal1.00407E6B		ST1 empty 0.0
00407E4D	57	PUSH EDI	ntdll.7C910738	ST2 empty 0.0
00407E4E	83EC 1C	SUB ESP,1C		ST3 empty 0.0
00407E51	8BCC	MOV ECX,ESP		ST4 empty 0.0
00407E53	896424 30	MOV DWORD PTR SS:[ESP+30],ESP		ST5 empty 0.0
00407E57	68 4AE54200	PUSH up_d_pal1.0042E5A4	ASCII "Error"	ST6 empty 1.00000000000000000000
00407E5C	E8 4F86FFFF	CALL up_d_pal1.004024B0		ST7 empty 1.00000000000000000000
00407E61	83EC 1C	SUB ESP,1C		FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (E0)
00407E64	8BCC	MOV ECX,ESP		FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
00407E6A	68 04E44200	PUSH up_d_pal1.0042E4D4	ASCII "No game requiring an update	
00407E6F	E8 3C86FFFF	CALL up_d_pal1.004024B0		
00407E74	53	PUSH EBX		
00407E75	E8 66540000	CALL up_d_pal1.004002E0		
00407E7A	83C4 40	ADD ESP,40		
00407E7D	808C24 94000000	LEA ECX,DWORD PTR SS:[ESP+94]		
00407E84	✓ E8 4796FFFF	CALL up_d_pal1.00401600		
00407E89	808C24 78	LEA ECX,DWORD PTR SS:[ESP+78]		
00407E8D	E8 3E96FFFF	CALL up_d_pal1.00401600		
00407E92	804C24 14	LEA ECX,DWORD PTR SS:[ESP+14]		

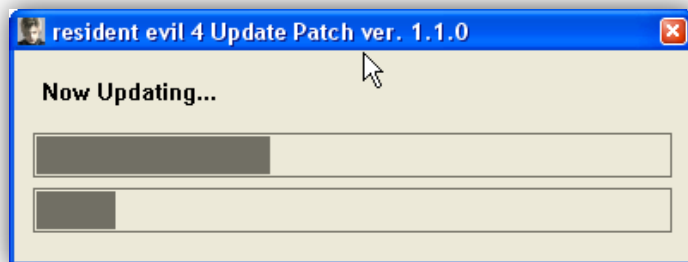
Jump is NOT taken  
00407EAB=up\_d\_pal1.00407EAB

Address	Hex dump	ASCII	Address	Value	Comment
00434000	00 00 00 00 FR 44 42 00 40 D3 42 00 60 D3 42 00	....DB.0EB.TEB.	0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00434010	70 D3 42 00 00 D3 42 00 00 D3 42 00 00 D3 42 00	pEB.CEB.FEB.AEB.	0012FFC8	7C910738	ntdll.7C910738
00434020	C0 D3 42 00 E0 D3 42 00 00 D4 42 00 00 D4 42 00	EB.0EB..0F..0F.	0012FFCC	FFFFFFFF	
00434030	20 D4 42 00 30 D4 42 00 40 D4 42 00 60 D4 42 00	0F.0F.0F.0F.0F.	0012FFD0	7FFDF000	
00434040	80 D4 42 00 A0 D4 42 00 C0 D4 42 00 E0 D4 42 00	0F.0F.0F.0F.0F.	0012FFD4	8054A6E0	
00434050	F0 D4 42 00 10 D5 42 00 30 D5 42 00 50 D5 42 00	0F.F0B.0FB.F0B.	0012FFD8	0012FFC8	
00434060	70 D5 42 00 80 D5 42 00 00 00 00 00 00 00 00 00	0F.0F.0F.0F.0F.	0012FFDC	890ED3A0	
00434070	00 D4 42 00 E1 92 42 00 0E A1 42 00 00 00 00 00	24B.8(B.4B.1B.	0012FFE0	FFFFFFFF	End of SEH chain
00434080	00 00 00 00 00 00 00 00 AF A1 42 00 00 00 00 00	....IB.....	0012FFE4	7C839AA0	SE handler
00434090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....TB.....	0012FFE8	7C816FE0	kernel32.7C816FE0
004340A0	01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00	0.....0.....	0012FFEC	00000000	
004340B0	00 00 00 00 32 00 00 00 70 01 00 00 1C 00 00 00	....2...p0.....	0012FFF0	00000000	
004340C0	20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0012FFF4	00000000	
004340D0	00 00 00 00 00 00 00 00 00 00 00 00 0F 00 00 00	.....*	0012FFF8	00423548	
004340E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....*	0012FFFC	00000000	ASCII "j'h'h'B"
004340F0	00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00	.....*			
00434100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....*			
00434110	00 00 00 00 0F 00 00 00 00 00 00 00 0F 00 00 00	.....*			
00434120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....*			
00434130	0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....*			
00434140	00 00 00 00 00 00 00 00 00 00 00 00 0F 00 00 00	.....*			
00434150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....*			

Command:

Program entry point:  Paused

You see 00407E4B change it to JMP SHORT up\_d\_pal1.00407EAB and patch will start patching us to 1.1 and securom protected (well it takes a while)



## 4. Third Step: Load the game into Olly and rebase

Lets load game.exe into olly. Press alt+M to see memory.

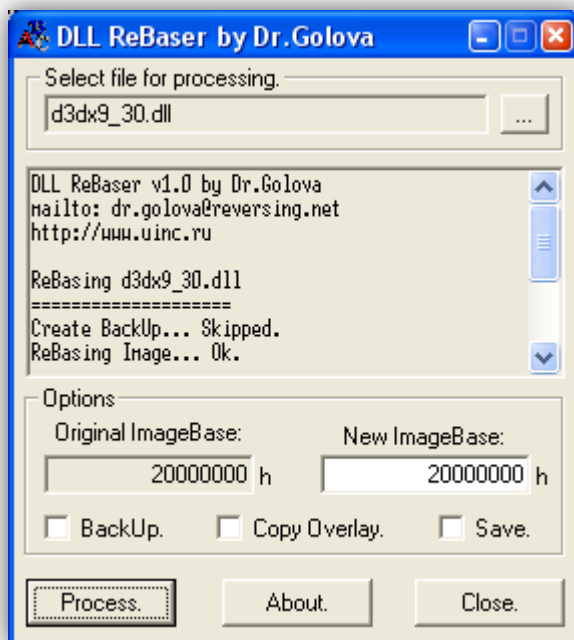
As you can see d3dx9\_30.dll blocks our dump to have linear regions :(

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00400000	00001000	game		PE header	Image	R	RWE	
00401000	0005F000	game	.text	code	Image	R	RWE	
007C0000	00001000	game	.pdata	code	Image	R	RWE	
00821000	04DC7000	game	.data	code,data	Image	R	RWE	
007E8000	00003000	game	.idata	code	Image	R	RWE	
007EE000	00003000	game	.rsrc	code,resource	Image	R	RWE	
007EE000	00050000	game	.ars	code	Image	R	RWE	
00D49000	00008000	game	.est	code	Image	R	RWE	
00D51000	002DC000	game	.artem	code	Image	R	RWE	
00E02000	00006000	game	.celare	code	Image	R	RWE	
00E03000	000F0000	game	.securom	code,import	Image	R	RWE	
00E13000	00007000				Priv	RM	RM	
00E33000	00001000	d3dx9_30		PE header	Image	R	RWE	
00E331000	00210000	d3dx9_30	.text	code,import	Image	R	RWE	
00E4E000	00031000	d3dx9_30	.data	code,data	Image	R	RWE	
00E57F000	00001000	d3dx9_30	.rsrc	code,resource	Image	R	RWE	
00E580000	00010000	d3dx9_30	.reloc	code,reloc	Image	R	RWE	
00E590000	00004000				Map	R E	R E	
00E5E0000	00002000				Map	R E	R E	
00E6E0000	00103000				Map	R	R	
00E770000	0005C000				Map	R E	R E	
00E7F0000	00001000				Priv	RM	RM	
00E8F0000	00002000				Map	R	R	
00E800000	00003000				Priv	RM	RM	
4FD050000	00001000	d3d9		PE header	Image	R	RWE	
4FD051000	00130000	d3d9	.text	code,import	Image	R	RWE	
4FEDC0000	0000B000	d3d9	.data	code,data	Image	R	RWE	
4FEE70000	00001000	d3d9	.rsrc	code,resource	Image	R	RWE	
4FEE80000	0000E000	d3d9	.reloc	code,reloc	Image	R	RWE	
5D5200000	00001000	COMCTL32		PE header	Image	R	RWE	
5D5210000	00071000	COMCTL32	.text	code,import	Image	R	RWE	
5D5920000	00003000	COMCTL32	.data	code,data	Image	R	RWE	
5D5950000	00020000	COMCTL32	.rsrc	code,resource	Image	R	RWE	
5D5B50000	00005000	COMCTL32	.reloc	code,reloc	Image	R	RWE	
6D3500000	00001000	DINPUT8		PE header	Image	R	RWE	
6D3510000	00023000	DINPUT8	.text	code,import	Image	R	RWE	
6D3740000	0000A000	DINPUT8	.data	code,data	Image	R	RWE	
6D37E0000	00009000	DINPUT8	.rsrc	code,resource	Image	R	RWE	
6D3870000	00002000	DINPUT8	.reloc	code,reloc	Image	R	RWE	
6DED00000	00001000	d3d8thk		PE header	Image	R	RWE	
6DED10000	00002000	d3d8thk	.text	code,import	Image	R	RWE	
6DED30000	00001000	d3d8thk	.data	code,data	Image	R	RWE	
6DED40000	00001000	d3d8thk	.rsrc	code,resource	Image	R	RWE	
6DED50000	00001000	d3d8thk	.reloc	code,reloc	Image	R	RWE	
71A400000	00001000	WS2HELP		PE header	Image	R	RWE	
71A410000	00004000	WS2HELP	.text	code,import	Image	R	RWE	
71A450000	00001000	WS2HELP	.data	code,data	Image	R	RWE	
71A460000	00001000	WS2HELP	.rsrc	code,resource	Image	R	RWE	
71A470000	00001000	WS2HELP	.reloc	code,reloc	Image	R	RWE	
71A500000	00001000	WS2_32		PE header	Image	R	RWE	
71A510000	00013000	WS2_32	.text	code,import	Image	R	RWE	
71A640000	00001000	WS2_32	.data	code,data	Image	R	RWE	
71A650000	00001000	WS2_32	.rsrc	code,resource	Image	R	RWE	
71A660000	00001000	WS2_32	.reloc	code,reloc	Image	R	RWE	
71A700000	00001000	WSOCK32		PE header	Image	R	RWE	
71A710000	00003000	WSOCK32	.text	code,import	Image	R	RWE	
71A740000	00001000	WSOCK32	.data	code,data	Image	R	RWE	
71A750000	00004000	WSOCK32	.rsrc	code,resource	Image	R	RWE	
71A790000	00001000	WSOCK32	.reloc	code,reloc	Image	R	RWE	
73E400000	00001000	DSOUND		PE header	Image	R	RWE	
73E410000	00050000	DSOUND	.text	code,import	Image	R	RWE	
73EF60000	00002000	DSOUND	.data	code,data	Image	R	RWE	
73EF80000	00001000	DSOUND	.rsrc	code,resource	Image	R	RWE	
73EF90000	00003000	DSOUND	.reloc	code,reloc	Image	R	RWE	
763600000	00001000	IMM32		PE header	Image	R	RWE	
763610000	00015000	IMM32	.text	code,import	Image	R	RWE	
763760000	00001000	IMM32	.data	code,data	Image	R	RWE	
763770000	0000E000	IMM32	.rsrc	code,resource	Image	R	RWE	

Command: Module D:\WINXP\WinSxS\x86\_Microsoft.Windows.Common-Controls\_6595b64144ccf1df\_6.0.2600.2982\_x-ww\_ac3f9c03\comctl32.dll

Paused

So we have to use rebaser from Dr.Golova to fix it to our needs (dont worry All Works after it) rebase it to 20000000h



Lets reload game.exe and voila All is fine All range after exe till 20000000h is free:)

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00400000	00001000	game		PE header	Image	R	RWE	
00401000	0000F000	game	.text	code	Image	R	RWE	
009C0000	00001000	game	.idata	code	Image	R	RWE	
00A21000	04DC7000	game	.data	code,data	Image	R	RWE	
007E8000	00003000	game	.idata	code	Image	R	RWE	
007E8000	00003000	game	.rsrc	code,resour	Image	R	RWE	
007EE000	00005000	game	.ars	code	Image	R	RWE	
00D49000	00008000	game	.est	code	Image	R	RWE	
00D51000	002DC000	game	.arten	code	Image	R	RWE	
00E02000	00006000	game	.celare	code	Image	R	RWE	
00E03000	00007000	game	.securum	code,import	Image	R	RWE	
00E13000	00007000			Priv	RW			
00E33000	00004000			Map	R E		R E	
00E3F000	00002000			Map	R E		R E	
00E40000	00103000			Map	R		R	
00E51000	00005000			Map	R E		R E	
00E81000	00001000			Priv	RW		RW	
00E89000	00002000			Map	R		R	
00E9A000	00003000			Priv	RW		RW	
20000000	00001000	d3dx9_30		PE header	Image	R	RWE	
20001000	0021D000	d3dx9_30	.text	code,import	Image	R	RWE	
20021E000	000031000	d3dx9_30	.data	code,data	Image	R	RWE	
20024F000	00001000	d3dx9_30	.rsrc	code,resour	Image	R	RWE	
200250000	00010000	d3dx9_30	.reloc	code,reloca	Image	R	RWE	
4FD50000	00001000	d3d9		PE header	Image	R	RWE	
4FD51000	0018B000	d3d9	.text	code,import	Image	R	RWE	
4FEDC000	0000B000	d3d9	.data	code,data	Image	R	RWE	
4FEED000	00001000	d3d9	.rsrc	code,resour	Image	R	RWE	
4FEED000	0000E000	d3d9	.reloc	code,reloca	Image	R	RWE	
5D520000	00001000	COMCTL32		PE header	Image	R	RWE	
5D521000	00071000	COMCTL32	.text	code,import	Image	R	RWE	
5D532000	00003000	COMCTL32	.data	code,data	Image	R	RWE	
5D535000	00020000	COMCTL32	.rsrc	code,resour	Image	R	RWE	
5D5B5000	00005000	COMCTL32	.reloc	code,reloca	Image	R	RWE	
6D350000	00001000	DINPUT8		PE header	Image	R	RWE	
6D351000	00023000	DINPUT8	.text	code,import	Image	R	RWE	
6D374000	00004000	DINPUT8	.data	code,data	Image	R	RWE	
6D37E000	00009000	DINPUT8	.rsrc	code,resour	Image	R	RWE	
6D387000	00002000	DINPUT8	.reloc	code,reloca	Image	R	RWE	
6DED0000	00001000	d3d8thk		PE header	Image	R	RWE	
6DED1000	00002000	d3d8thk	.text	code,import	Image	R	RWE	
6DED3000	00001000	d3d8thk	.data	code,data	Image	R	RWE	
6DED4000	00001000	d3d8thk	.rsrc	code,resour	Image	R	RWE	
6DED5000	00001000	d3d8thk	.reloc	code,reloca	Image	R	RWE	
71A40000	00001000	WS2HELP		PE header	Image	R	RWE	
71A41000	00004000	WS2HELP	.text	code,import	Image	R	RWE	
71A45000	00001000	WS2HELP	.data	code,data	Image	R	RWE	
71A46000	00001000	WS2HELP	.rsrc	code,resour	Image	R	RWE	
71A47000	00001000	WS2HELP	.reloc	code,reloca	Image	R	RWE	
71A50000	00001000	WS2_32		PE header	Image	R	RWE	
71A51000	00013000	WS2_32	.text	code,import	Image	R	RWE	
71A64000	00001000	WS2_32	.data	code,data	Image	R	RWE	
71A65000	00001000	WS2_32	.rsrc	code,resour	Image	R	RWE	
71A66000	00001000	WS2_32	.reloc	code,reloca	Image	R	RWE	
71A70000	00001000	WSOCK32		PE header	Image	R	RWE	
71A71000	00003000	WSOCK32	.text	code,import	Image	R	RWE	
71A74000	00001000	WSOCK32	.data	code,data	Image	R	RWE	
71A75000	00004000	WSOCK32	.rsrc	code,resour	Image	R	RWE	
71A79000	00001000	WSOCK32	.reloc	code,reloca	Image	R	RWE	
73E90000	00001000	DSOUND		PE header	Image	R	RWE	
73E91000	00055000	DSOUND	.text	code,import	Image	R	RWE	
73EF5000	00002000	DSOUND	.data	code,data	Image	R	RWE	
73EF5000	00001000	DSOUND	.rsrc	code,resour	Image	R	RWE	
73EF9000	00003000	DSOUND	.reloc	code,reloca	Image	R	RWE	
76360000	00001000	IMM32		PE header	Image	R	RWE	
76361000	00015000	IMM32	.text	code,import	Image	R	RWE	
76376000	00001000	IMM32	.data	code,data	Image	R	RWE	
76377000	00005000	IMM32	.rsrc	code,resour	Image	R	RWE	

## 5. Fourth Step: daemons tools and OEP

1. Lets load maxi image into daemon tools 4.10
2. Run yasu to cloak virtual drives
3. Set break point on CreateProcessInternalA and run, after break you will see on stack param to use with oepfind, so lets use it.
4. For me command to run In Total commander is like this:

```
oep game.exe /Sonydac /05f0612d /05f0612d /220792E1 /1
```

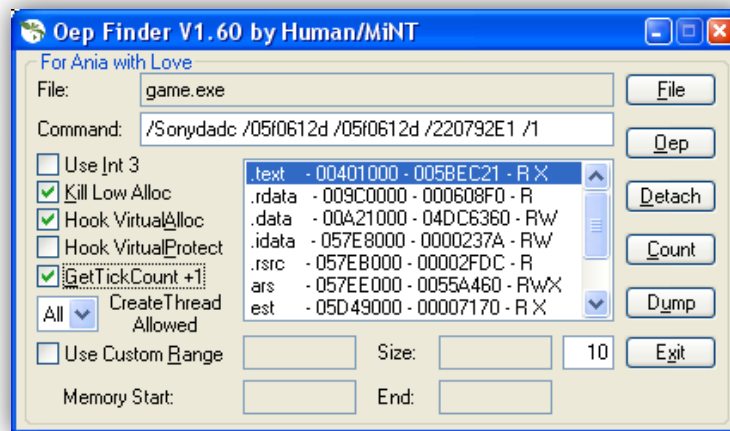
last param is time in ms, antidebug that spawns another instance of exe and kills parent if difference is too high from current GetTickCount.

But due we patch GetTickCount to count slower in rate of +1, we use /1 instead of original time

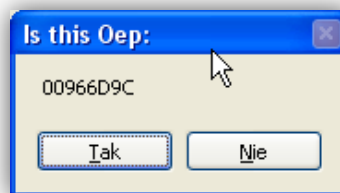
With oepfind we use these settings:

- kill low Alloc so it will not Alloc Any memory under imagebase
- hook virtualalloc to Alloc memory linear not random
- GetTickCount +1 to disable spawn of another proper process





Now Press detach to look for OEP. After about 10 seconds we have:



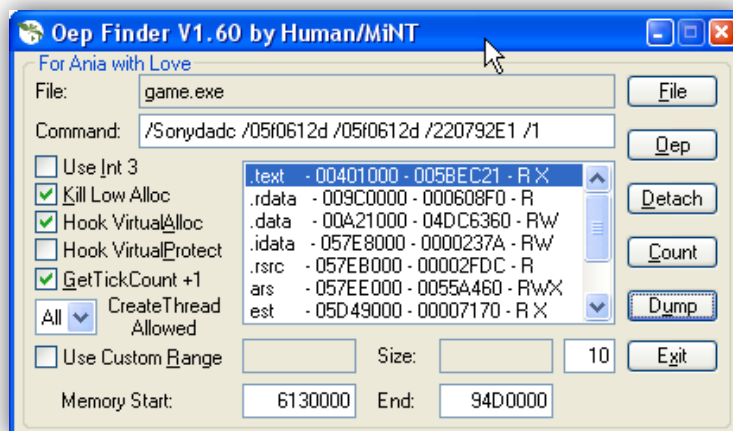
Fine so Press yes or Any key In your language. Run Olly and attach to our game.exe. Remember PID due we will need it later, mine is here **75C**

Don't run just in menu select view/threads and double click that one suspended.



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00113000	00001000			stack of na	Priv	RM	RM	
00114000	0011C000				Priv	RM	Gu	
00230000	00003000				Map	R	R	
00400000	00001000	game		PE header	Imag	R	RME	
00401000	005BF000	game	.text	code	Imag	R	RME	
009C0000	00061000	game	.idata	code, data	Imag	R	RME	
00A21000	040C7000	game	.data	code	Imag	R	RME	
057E8000	00003000	game	.idata	code	Imag	R	RME	
057EB000	00003000	game	.rsrc	code, resour	Imag	R	RME	
057EE000	00050000	game	ars	est	Imag	R	RME	
05D49000	00003000	game	est	SFX	Imag	R	RME	
05D51000	002DC000	game	artem		Imag	R	RME	
0602D000	00006000	game	celare		Imag	R	RME	
06033000	000F3000	game	.securom	imports	Imag	R	RME	
06130000	00043000				Priv	RM	RM	
06330000	00006000				Priv	RM	RM	
06340000	00003000				Priv	RM	RM	
06350000	00016000				Map	R	R	
06370000	00020000				Map	R	R	
06380000	00041000				Map	R	R	
06400000	00006000				Map	R	R	
06410000	00041000				Map	R	R	
06460000	00004000				Map	R E	R E	
06520000	00002000				Map	R E	R E	
06530000	00103000				Map	R	R	
06540000	00001000				Priv	RM	RM	
06650000	0000C000				Map	R E	R E	
06950000	00001000				Priv	RM	RM	
06960000	00005000				Priv	RM	RM	
06970000	00003000				Map	R	R	
06980000	00001000				Priv	RM	RM	
06A00000	00002000				Priv	RM	RM	
06A10000	00001000				Priv	RM	RM	
06A20000	00002000				Map	R	R	
06A30000	00010000				Priv	RM	RM	
06A40000	00002000				Map	R	R	
06A50000	00001000				Priv	RM	RM	
06A90000	0000C000				Priv	RM	RM	
06AA0000	00001000				Priv	RM	RM	
06AB0000	00001000				Priv	RM	RM	
06AC0000	00002000				Map	R	R	
06AD0000	0000B000				Priv	RME	RM	
06EC0000	00010000				Priv	RM	RM	
06ED0000	00001000				Priv	RM	RM	
06EE0000	00001000				Priv	RM	RM	
06EF0000	00001000				Priv	RM	RM	
06C00000	00001000				Priv	RM	RM	
06C10000	00001000				Priv	RM	RM	
06C20000	00001000				Priv	RM	RM	
06C30000	00001000				Priv	RM	RM	
06C40000	00001000				Priv	RM	RM	
06C50000	00001000				Priv	RM	RM	
06C60000	00001000				Priv	RM	RM	
06C70000	00001000				Priv	RM	RM	
06C80000	00001000				Priv	RM	RM	
06C90000	00001000				Priv	RM	RM	
06CA0000	00001000				Priv	RM	RM	
06CB0000	00001000				Priv	RM	RM	
06CC0000	00001000				Priv	RM	RM	
06CD0000	00001000				Priv	RM	RM	
06CE0000	00001000				Priv	RM	RM	
06CF0000	00001000				Priv	RM	RM	
06D00000	00001000				Priv	RM	RM	
06D10000	00001000				Priv	RM	RM	
06D20000	00001000				Priv	RM	RM	
06D30000	00001000				Priv	RM	RM	
06D40000	00001000				Priv	RM	RM	
06D50000	00001000				Priv	RME	RME	

And ends at 94CF000+1000 so its 94D0000 (last region that doesn't show Any file name, here last before d3dx9\_30.dll).  
Let put those into oepfind to Fields that are now not grayed out.



Press dump and in your dir you'll find a file named DUMP\_06130000-094D0000 of 54mb, with all the dumped range. Now you can close oepfind.

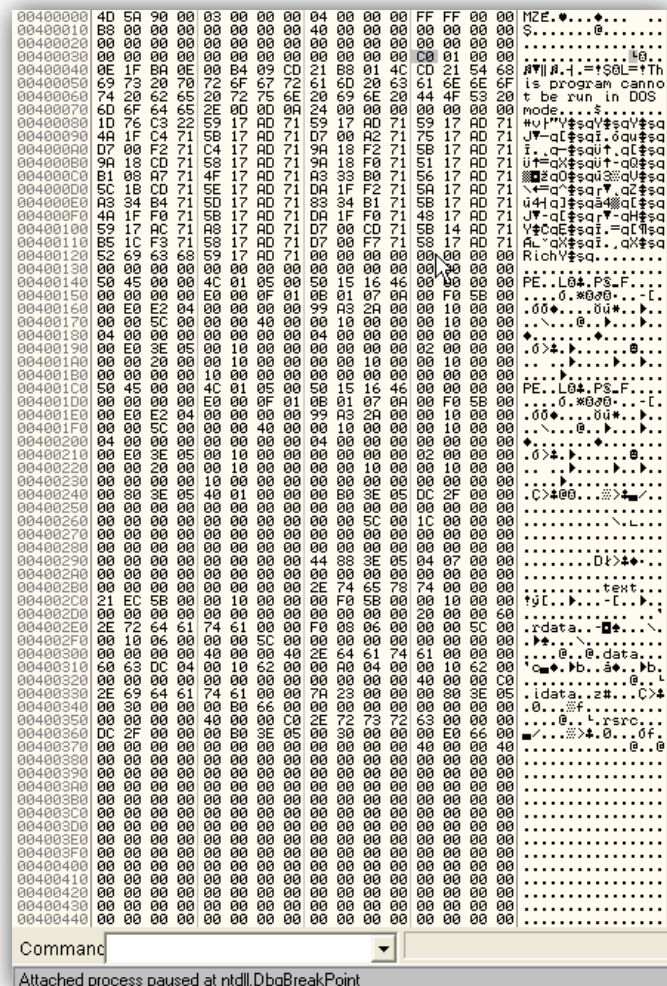
## 6. Fifth Step: Fixing Anti-dumps

Now we are ready to dump exe, but before that we must do one thing. Another securom protection is left. Securom uses also EP as antidump. So when ollydump change now EP to OEP we will be screwed.

Securom adds return params from many Apis to address. So when for example original PID is 200 and dumped program's PID is 100, we have 200-100+address gets wrong data and end with a page fault. So it always should be 0, when both are same.

Ok, then back to our EP, what to do? Simply move the header somewhere else. Select all from 400140 till 400300 do binary copy and binary paste it to 4001C0

Then we change 40 to C0 at 40003C to point to a new place.



Now we can dump our exe with unticked rebuild imports.

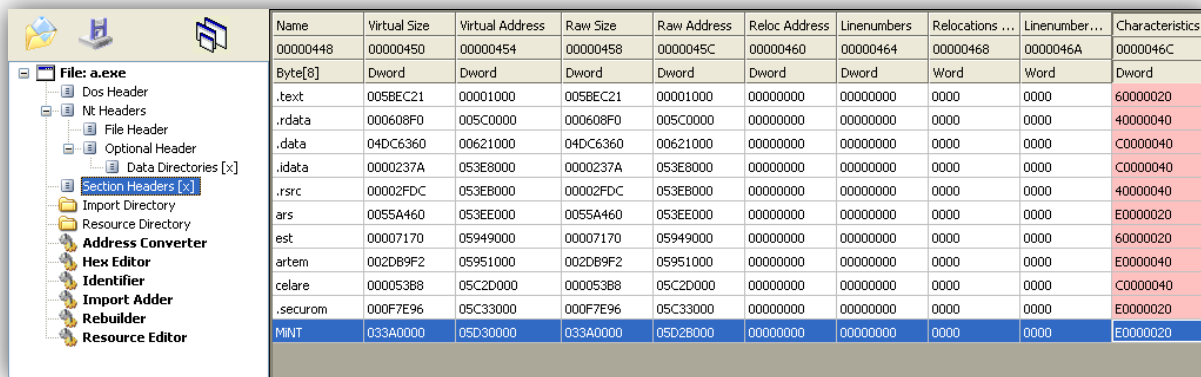
As you can see exe is 97Mb, biohazard exe without securom is Just 6Mb, that's how protections hurt customers, not my fault then.

Next step will be to fix imports and add our regions.

So let fire up CFF Explorer (I patched mine due I was pissed with asking should I load more than fucked 20MB, what a dumb question today, when minimum memory is 2GB).

Click section headers and right click to do "Add Section (file data)" to add our dumped regions.

Fix virtual address with start address-imagebase 6130000-400000=5D30000 and change section rights to E0000020



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
00000448	00000450	00000454	00000458	0000045C	00000460	00000464	00000468	0000046A	0000046C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	005BEC21	00001000	005BEC21	00001000	00000000	00000000	0000	0000	60000020
.rdata	000608F0	005C0000	000608F0	005C0000	00000000	00000000	0000	0000	40000040
.data	04DC6360	00621000	04DC6360	00621000	00000000	00000000	0000	0000	C0000040
.idata	0000237A	053E8000	0000237A	053E8000	00000000	00000000	0000	0000	C0000040
.rsrc	00002FDC	053EB000	00002FDC	053EB000	00000000	00000000	0000	0000	40000040
.ars	0055A460	053EE000	0055A460	053EE000	00000000	00000000	0000	0000	E0000020
.est	00007170	05949000	00007170	05949000	00000000	00000000	0000	0000	60000020
.artem	002DB9F2	05951000	002DB9F2	05951000	00000000	00000000	0000	0000	E0000040
.celare	000053B8	05C2D000	000053B8	05C2D000	00000000	00000000	0000	0000	C0000040
.securom	000F7E96	05C33000	000F7E96	05C33000	00000000	00000000	0000	0000	E0000020
MINT	033A0000	05D30000	033A0000	05D2B000	00000000	00000000	0000	0000	E0000020

Do right click, rebuild image size, header and save exe.

Now fix imports, well we don't need imprec all we need is into the .idata section. Run Winhex with game.exe and press alt+G now go to 66b000 can you see those addresses?  
So copy those till API names.

Close it and now run Winhex on dump.exe, press alt+G and go to 53E8000, now Ctrl+B to paste our IAT, that now as you can see is not RVA, only 7Cxxxxx. Save the exe and rebuild from the beginning. After the rebuild we will again paste IAT due from Olly we will dump exe again with API addresses changed from RVA to 7Cxxxxx.

## 7. Sixth Step: fixing the CRCChecks

Till now we dumped the exe, fixed the IAT and the PE header, and added missing file regions. We can start with fixing rest.

We start with fixing CRCchecks, for this we will use my Securom 7.x CRC Check Fixer.txt script (included in goodies folder of this distribution).

CRCchecks comes in 2 flavours: memory and register. Memory CRC updates always some address [esp+xx] and later uses it, other type updates one of registers eax,ebx,ecx etc.

So what this script does?

It searches for patterns of CRCcheck and sets an HW breakpoint, after calculation loop it replaces everything with nops and paste there just calculated value that will be then hardcoded.



Address	Hex dump	Disassembly	Comment
057EE002	89EC 18	SUB ESP,18	
057EE005	C74424 14 99813AD6	MOV DWORD PTR SS:[ESP+14],D63A8199	
057EE00D	C74424 10 2E080000	MOV DWORD PTR SS:[ESP+10],82E	
057EE015	894424 0C	MOV DWORD PTR SS:[ESP+C],EAX	
057EE019	B8 00E07E05	MOV EAX,a.057EE000	
057EE01E	C14C24 14 18	ROR DWORD PTR SS:[ESP+14],18	
057EE023	C1FA 00	SAR EDI,0	Shift constant out of range 1..31
057EE026	897C24 08	MOV DWORD PTR SS:[ESP+8],EDI	ntdll.7C910738
057EE02A	8B38	MOV EDI,DWORD PTR DS:[EAX]	
057EE02C	90	NOP	
057EE02D	017C24 14	ADD DWORD PTR SS:[ESP+14],EDI	ntdll.7C910738
057EE031	83C0 04	ADD EAX,4	
057EE034	664FF4C24 10	DEC WORD PTR SS:[ESP+10]	
057EE039	75 EF	JNZ SHORT a.057EE02D	
057EE03B	90	NOP	
057EE03C	804C24 14 01	OR BYTE PTR SS:[ESP+14],1	
057EE041	8B4424 18	MOV EAX,DWORD PTR SS:[ESP+18]	
057EE045	8B7C24 14	MOV EDI,DWORD PTR SS:[ESP+14]	
057EE049	894424 14	MOV DWORD PTR SS:[ESP+14],EAX	
057EE04D	0FA4E9 00	SHLD ECX,EBP,0	Shift constant out of range 1..31
057EE051	50	PUSH EAX	
057EE052	16	PUSH SS	
057EE053	17	POP SS	
057EE054	87D2	XCHG EDX,EDX	Modification of segment register
057EE056	9C	PUSHFD	ntdll.KiFastSystemCallRet
057EE057	8B8424 00	MOV EAX,DWORD PTR SS:[ESP]	ntdll.7C910738
057EE05A	0FA4D0 00	SHLD EDX,EBX,0	Shift constant out of range 1..31
057EE05E	80E4 01	AND AH,1	
057EE061	0FA4D0 00	SHRD EBP,EBX,0	Shift constant out of range 1..31
057EE065	74 05	JE SHORT a.057EE06C	
057EE067	B9 BE070000	MOV EAX,7BE	
057EE06C	33C0	XOR EAX,EAX	
057EE06E	EB 00	JMP SHORT a.057EE070	
057EE070	74 02	JE SHORT a.057EE074	
057EE072	0FA590 588B4424	SHLD DWORD PTR SS:[EBP+24448B58],EBX,CL	
057EE079	0C 09	OR AL,9	
057EE07B	7C 24	JL SHORT a.057EE081	
057EE07D	18EB	SBB BL,CH	
057EE07F	008B 7C240883	ADD BYTE PTR DS:[EBX+8308247C],CL	

ESP=002FF44

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF44	7C910738	ntdll.7C910738
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF48	FFFFFFFF	
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF4C	7FFD9000	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF50	00780018	a.00780018
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF54	00000008	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF58	00000000	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF5C	00000000	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF60	00000000	
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF64	804FEDD8	
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF68	00000005	
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF6C	00010002	UNICODE "D:=D:\WINXP\system32"
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF70	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF74	7C900000	ntdll.7C900000
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF78	00000000	
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF7C	00000000	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF80	00000000	
00A21100	00 00 00 00 DA A7 96 00 00 00 00 00 00 00 00 00 00	.....r.....	0022FF84	00000000	
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF88	00966D9C	a.<ModuleEntryPoint>
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF8C	00000000	
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF90	88ECAD80	
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF94	80502442	
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF98	00000000	

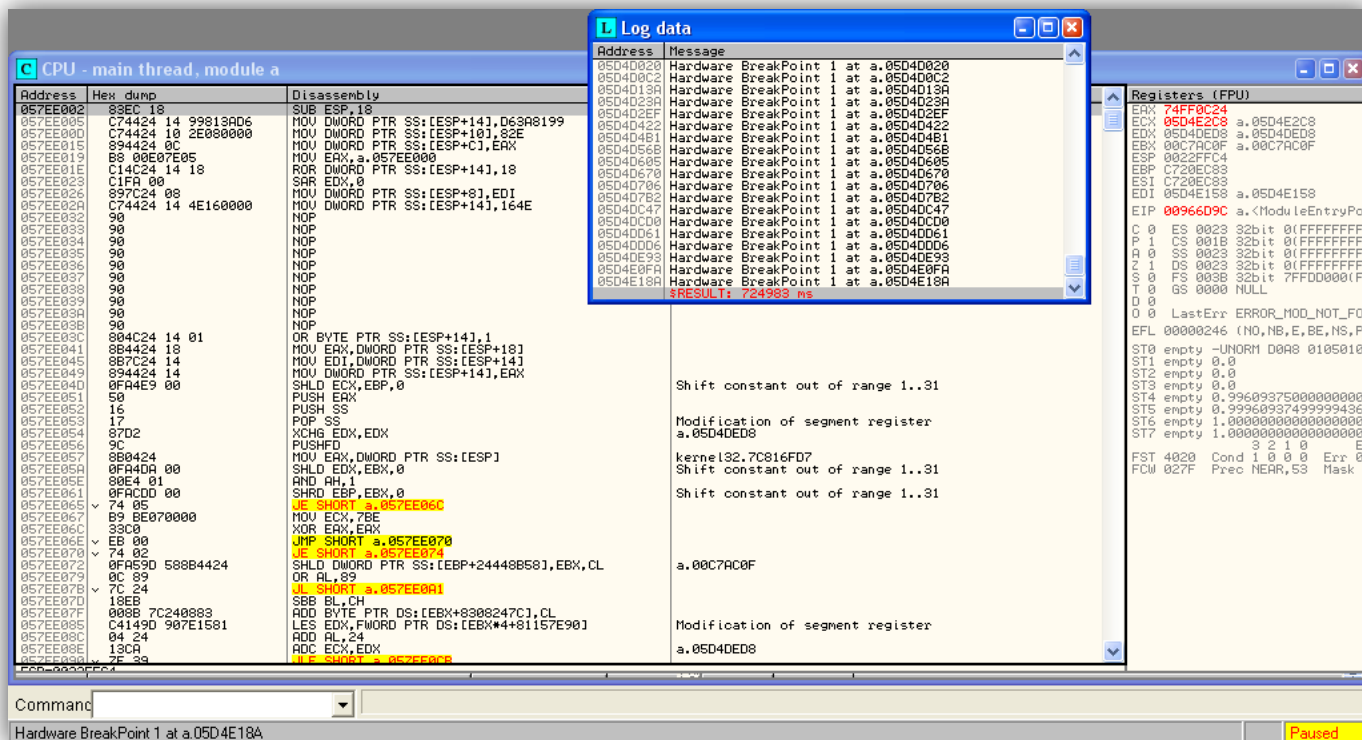
Command: Module D:\WINXP\WinSxS\x86\_Microsoft.Windows.Common-Controls\_6595b64144ccf1df\_6.0.2600.2982\_x-ww\_ac3f9c03\comctl32.dll

Paused

Like you can see 057EE02D ADD DWORD PTR SS:[ESP+14],EDI. This is the place of our fix. (I noticed that, and really don't know why, odbgscrip runs faster when I have some some movie in background, open but stopped).

It takes a lot of time even on my Intel C2D E6700 (with a movie stopped it runs 4x faster :P). Anyway the homework this script does is really huge, it has to fix more than 100.000 locations. The works is done into a temporary memory buffer: CRCchecks calc CRCs using their own native loops, after this the temp buffer is copied back into sections substituting the CrCCheck just executed. Minimizing Olly window also speeds up because Windows doesn't redraw all.

A result of the script is shown here:



Like you can see in above figure, ESP+14 updates always to 164E, and log shows script took 724983ms so 12 minutes :P

Don't alter script, it will not be faster, I already optimized it to max. Even direct write of opcodes as bytes is faster than assembling new instruction. CRCcheck pattern is in 2 sections, so both those are in script.

## 8. Seventh Step: taking care of antidumps

Now its time to take care of antidump APIs. In this version securom uses the following:

- GetCurrentProcessId: every process gets PID but to make it work, dump needs to return PID as original process
- GetVersion: every windows have own version, so to make it work on vista but dumped on xp we need to return xp version
- CPUID: every CPU have own ID returned in eax & edx, so to make it work on other CPUs it has to return the ID stored in the dump, ours then.
- ResetEvent: every securom exe creates event or events, dump doesn't have them so we need to return 1
- GetComputerNameA: every pc has a name, so other pc must match what we store into the dump
- GetUserNameA: every user logged has a name, other user logged and it crashes, so same applies here
- RtlGetLastWin32Error: here again we need to return 1, in case there are errors we tell there aren't any
- GetSystemInfo: every pc has own 20 bytes System info table returned, again it must match our

Knowing all this we can start patching securom. For this we need a place, I choose 9BFF00. And that's how it will look:

```

009BFF00  B8 88030000    MOV EAX,388
009BFF05  034424 04      ADD EAX,DWORD PTR SS:[ESP+4] ; ntdll.7C910738
009BFF09  C2 0400        RETN 4
009BFF0C  B8 0501280A    MOV EAX,0A280105
009BFF11  2B4424 04      SUB EAX,DWORD PTR SS:[ESP+4] ; ntdll.7C910738
009BFF15  C2 0400        RETN 4
009BFF18  B8 D6060000    MOV EAX,6D6
009BFF1D  334424 04      XOR EAX,DWORD PTR SS:[ESP+4] ; ntdll.7C910738
009BFF21  C2 0400        RETN 4
009BFF24  A1 88BAD705    MOV EAX,DWORD PTR DS:[5D7BA88]
009BFF29  C2 0400        RETN 4
009BFF2C  8B4424 04      MOV EAX,DWORD PTR SS:[ESP+4] ; ntdll.7C910738
009BFF30  8B0D 98BAD705  MOV ECX,DWORD PTR DS:[5D7BA98]
009BFF36  03C1          ADD EAX,ECX
009BFF38  35 A416827C    XOR EAX,7C8216A4
009BFF3D  C2 0400        RETN 4

```

```

009BFF40      A1 F4BAD705      MOV EAX,DWORD PTR DS:[5D7BAF4]
009BFF45      8B0D 04BBD705      MOV ECX,DWORD PTR DS:[5D7BB04]
009BFF4B      03C1                ADD EAX,ECX
009BFF4D      0305 ECBAD705      ADD EAX,DWORD PTR DS:[5D7BAEC]
009BFF53      C2 0400            RETN 4
009BFF56      B8 E0F7CB85        MOV EAX,85CBF7E0
009BFF5B      C2 0400            RETN 4
009BFF5E      E8 00000000        CALL b.009BFF63
009BFF63      5E                POP ESI                      ; kernel32.7C816FD7
009BFF64      83C6 19            ADD ESI,19
009BFF67      57                PUSH EDI                    ; ntdll.7C910738
009BFF68      8B7C24 08          MOV EDI,DWORD PTR SS:[ESP+8]
009BFF6C      B9 24000000        MOV ECX,24
009BFF71      F3:A4              REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
009BFF73      BE 10000000        MOV ESI,10
009BFF78      5F                POP EDI                      ; kernel32.7C816FD7
009BFF79      C2 0400            RETN 4

```

Here you have binary copy of the patch you can paste in binary format:

```

B8 74 05 00 00 03 44 24 04 C2 04 00 B8 05 01 28 0A 2B 44 24 04 C2 04 00 B8 D6 06 00 00 33 44 24
04 C2 04 00 A1 88 BA D7 05 C2 04 00 8B 44 24 04 8B 0D 98 BA D7 05 03 C1 35 A4 16 82 7C C2 04 00
A1 F4 BA D7 05 8B 0D 04 BB D7 05 03 C1 03 05 EC BA D7 05 C2 04 00 B8 CF F7 4B 87 C2 04 00 E8 00
00 00 00 5E 83 C6 19 57 8B 7C 24 08 B9 24 00 00 00 F3 A4 BE 10 00 00 00 5F C2 04 00 00 00 00 00
00 10 00 00 00 01 00 FF FF FE 7F 03 00 00 00 02 00 00 00 4A 02 00 00 00 01 00 06 00 06 0F

```

Now I will explain why it looks like it. First of all go to address 0x5A61A7B

Address	Hex dump	Disassembly	Comment	Registers (FPU)
05A61A7B	50	PUSH EAX		EAX: 00000000
05A61A7B	FF15 3446EA05	CALL DWORD PTR DS:[5EA4634]	b.05A66760	ECX: 0022FFB0
05A61A81	35 5067A605	XOR EAX,5A66760		EDX: 7C90B94 ntdll.KiFastSystemCallRet
05A61A86	3305 3446EA05	XOR EAX,DWORD PTR DS:[5EA4634]	b.05A66760	EBX: 7FFD8000
05A61A8C	E9 B2000000	JMP b.05A61B43		ESP: 0022FFC4
05A61A92	50	PUSH EAX		EBP: 0022FFB0
05A61A92	FF15 3846EA05	CALL DWORD PTR DS:[5EA4638]	b.05A66770	ESI: FFFFFFFF
05A61A98	35 7067A605	XOR EAX,5A66770		EDI: 7C910738 ntdll.7C910738
05A61A9D	3305 3846EA05	XOR EAX,DWORD PTR DS:[5EA4638]	b.05A66770	EIP: 00966D9C b.<ModuleEntryPoint>
05A61AA3	E9 9B000000	JMP b.05A61B43		C 0 ES 0023 32bit 0(FFFFFFFF)
05A61AA8	50	PUSH EAX		P 1 CS 001B 32bit 0(FFFFFFFF)
05A61AA9	FF15 3C46EA05	CALL DWORD PTR DS:[5EA463C]	b.05A66780	A 0 SS 0023 32bit 0(FFFFFFFF)
05A61AAE	35 8067A605	XOR EAX,5A66780		Z 1 DS 0023 32bit 0(FFFFFFFF)
05A61AB4	3305 3C46EA05	XOR EAX,DWORD PTR DS:[5EA463C]	b.05A66780	S 0 FS 003B 32bit 7FFDF000(FFF)
05A61AB8	E9 84000000	JMP b.05A61B43		T 0 GS 0000 NULL
05A61ABF	50	PUSH EAX		O 0
05A61AC0	FF15 4046EA05	CALL DWORD PTR DS:[5EA4640]	b.05A66800	LastErr ERROR_MOD_NOT_FOUND (0000007E)
05A61AC6	35 9067A605	XOR EAX,5A66800		EFL 00010246 (NO,NB,E,BE,HS,FE,GE,LE)
05A61ACB	3305 4046EA05	XOR EAX,DWORD PTR DS:[5EA4640]	b.05A66800	ST0 empty -UNORM D108 01050104 00000000
05A61AD1	EB 70	JMP SHORT b.05A61B43		ST1 empty 0.0
05A61AD3	83D0 50B0705 00	CMP DWORD PTR DS:[5D7BA50],0		ST2 empty 0.0
05A61AD8	50	PUSH EAX		ST3 empty 0.0
05A61AD8	74 08	JE SHORT b.05A61B25		ST4 empty 0.39503375000000000000
05A61ADD	FF15 3446EA05	CALL DWORD PTR DS:[5EA4634]	b.05A66760	ST5 empty 0.99503374999999999999
05A61AE3	EB 06	JMP SHORT b.05A61B25		ST6 empty 1.00000000000000000000
05A61AE5	FF15 4446EA05	CALL DWORD PTR DS:[5EA4644]	b.05A66850	ST7 empty 1.00000000000000000000
05A61AEB	35 5068A605	XOR EAX,5A66850		FST 4020 Cond 1 0 0 0 Err 0 1 0 0 0 0 (EO)
05A61AF0	3305 4446EA05	XOR EAX,DWORD PTR DS:[5EA4644]	b.05A66850	FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
05A61AF6	EB 4B	JMP SHORT b.05A61B43		
05A61AF8	50	PUSH EAX		
05A61AF9	FF15 4846EA05	CALL DWORD PTR DS:[5EA4648]	b.05A66900	
05A61AFF	35 0069A605	XOR EAX,5A66900		
05A61B04	3305 4846EA05	XOR EAX,DWORD PTR DS:[5EA4648]	b.05A66900	
05A61B09	EB 37	JMP SHORT b.05A61B43		
05A61B0C	83D0 50B0705 00	CMP DWORD PTR DS:[5D7BA50],0		
05A61B13	50	PUSH EAX		
05A61B14	74 08	JE SHORT b.05A61B1B		
05A61B16	FF15 3846EA05	CALL DWORD PTR DS:[5EA4638]	b.05A66770	
05A61B1C	EB 06	JMP SHORT b.05A61B24		

DS:[05EA4634]=05A66760 (b.05A66760)

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FFD8000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	8054A6ED	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	8927D020	End of SEH chain
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	SE handler
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C839A08	kernel32.7C816FE0
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	00000000	
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	00966D9C	b.<ModuleEntryPoint>
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFC	00000000	
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command: Module D:\WINXP\WinSxS\86\_Microsoft.Windows.Common-Controls\_6595b64144ccf1df\_6.0.2600.2982\_x-ww\_ac3f9c03\comctl32.dll

Paused

Lets see what's inside CALL DWORD PTR DS:[5EA4634]?

```
05A66760    FF15 7CBAD705    CALL DWORD PTR DS:[5D7BA7C]    ; kernel32.GetCurrentProcessId
05A66766    034424 04      ADD EAX,DWORD PTR SS:[ESP+4]    ; ntdll.7C910738
05A6676A    C2 0400      RETN 4
```

This code wants our PID, and later adds value pushed on stack to it. That's why we should have here:

```
MOV EAX,75C    ; is my PID of securom process that I dumped
ADD EAX,DWORD PTR SS:[ESP+4]
RETN 4
```

Now get back out of it. Do you see this?

```
05A61A7A    50      PUSH EAX
05A61A7B    FF15 3446EA05    CALL DWORD PTR DS:[5EA4634]    ; b.05A66760
05A61A81    35 6067A605      XOR EAX,5A66760
05A61A86    3305 3446EA05    XOR EAX,DWORD PTR DS:[5EA4634]    ; b.05A66760
```

Tricky bastard, it does a XOR on EAX, with the address of this small code part and then does again a XOR with the address of small code part that is held in [5EA4634] so we can't change address for our patch. But hey just change 2<sup>nd</sup> XOR to same as 1<sup>st</sup> one, 2 chained XORs with same value will give us 0 so EAX will not change. Same trick we will do on other below, and under [5EA4634] we will now put 9BFF00 for our patch.

One done few more to go.

What have we in 05A61A92 CALL DWORD PTR DS:[5EA4638] ?

Oh its:

```
05A66770    FF15 80BAD705    CALL DWORD PTR DS:[5D7BA80]    ; kernel32.GetVersion
05A66776    2B4424 04      SUB EAX,DWORD PTR SS:[ESP+4]    ; ntdll.7C910738
05A6677A    C2 0400      RETN 4
```

So replacing it with:

```
009BFF0C    B8 0501280A      MOV EAX,0A280105    ; my xp version
009BFF11    2B4424 04      SUB EAX,DWORD PTR SS:[ESP+4]    ; ntdll.7C910738
009BFF15    C2 0400      RETN 4
```

We will defeat that part too.

Next

```
05A61AA9    FF15 3C46EA05    CALL DWORD PTR DS:[5EA463C]    ; b.05A66780
05A66780    8D6424 FC      LEA ESP,DWORD PTR SS:[ESP-4]
05A66784    892C24      MOV DWORD PTR SS:[ESP],EBP
05A66787    8BEC      MOV EBP,ESP
05A66789    8D6424 FC      LEA ESP,DWORD PTR SS:[ESP-4]
05A6678D    890C24      MOV DWORD PTR SS:[ESP],ECX
05A66790    8D6424 FC      LEA ESP,DWORD PTR SS:[ESP-4]
05A66794    891C24      MOV DWORD PTR SS:[ESP],EBX
05A66797    8B0D 2046EA05    MOV ECX,DWORD PTR DS:[5EA4620]
05A6679D    85C9      TEST ECX,ECX
05A6679F    7E 16      JLE SHORT b.05A667B7
05A667A1    A1 B4BAD705      MOV EAX,DWORD PTR DS:[5D7BAB4]
05A667A6    3345 08      XOR EAX,DWORD PTR SS:[EBP+8]    ; b.<ModuleEntryPoint>
05A667A9    2B0D 2446EA05    SUB ECX,DWORD PTR DS:[5EA4624]
05A667AF    890D 2046EA05    MOV DWORD PTR DS:[5EA4620],ECX
05A667B5    EB 41      JMP SHORT b.05A667F8
05A667B7    D13D 2446EA05    SAR DWORD PTR DS:[5EA4624],1
05A667BD    53      PUSH EBX
05A667BE    51      PUSH ECX
05A667BF    52      PUSH EDX    ; ntdll.KiFastSystemCallRet
05A667C0    B8 01000000      MOV EAX,1
05A667C5    0FA2      CPUID
05A667C7    5A      POP EDX    ; kernel32.7C816FD7
05A667C8    59      POP ECX    ; kernel32.7C816FD7
05A667C9    5B      POP EBX    ; kernel32.7C816FD7
05A667CA    83E0 DF      AND EAX,FFFFFFDF
```

```

05A667CD    A3 B4BAD705    MOV  DWORD PTR DS:[5D7BAB4],EAX
05A667D2    3345 08        XOR  EAX,DWORD PTR SS:[EBP+8]
05A667D5    8945 FC        MOV  DWORD PTR SS:[EBP-4],EAX
05A667D8    833D 2446EA05 00    CMP  DWORD PTR DS:[5EA4624],0
05A667DF    8B45 FC        MOV  EAX,DWORD PTR SS:[EBP-4]
05A667E2    C705 2046EA05 00000100    MOV  DWORD PTR DS:[5EA4620],10000
05A667EC    75 0A        JNZ  SHORT b.05A667F8
05A667EE    C705 2446EA05 01000000    MOV  DWORD PTR DS:[5EA4624],1
05A667F8    5B        POP  EBX
05A667F9    C9        LEAVE
05A667FA    C2 0400    RETN 4

```

Oh it's the CPUID!!!

And as you can see it does on it:

```

05A667CA    83E0 DF        AND  EAX,FFFFFFDF
05A667D2    3345 08        XOR  EAX,DWORD PTR SS:[EBP+8]

```

So proper patch for my CPU, a little optimized, is:

```

009BFF18    B8 D6060000    MOV  EAX,6D6
009BFF1D    334424 04      XOR  EAX,DWORD PTR SS:[ESP+4]
009BFF21    C2 0400    RETN 4

```

Why not esp+8? Why I don't push and pop ebx?

Next

```

05A61AC0    FF15 4046EA05    CALL DWORD PTR DS:[5EA4640]
05A66800    A1 90BAD705    MOV  EAX,DWORD PTR DS:[5D7BA90]
05A66805    3B05 2846EA05    CMP  EAX,DWORD PTR DS:[5EA4628]
05A6680B    7C 2C        JL  SHORT b.05A66839
05A6680D    FF35 88BAD705    PUSH DWORD PTR DS:[5D7BA88]
05A66813    FF15 84BAD705    CALL DWORD PTR DS:[5D7BA84]
05A66819    85C0        TEST EAX,EAX
05A6681B    75 22        JNZ  SHORT b.05A6683F
05A6681D    2105 90BAD705    AND  DWORD PTR DS:[5D7BA90],EAX
05A66823    813D 2846EA05 00001000    CMP  DWORD PTR DS:[5EA4628],100000
05A6682D    7D 06        JGE  SHORT b.05A66835
05A6682F    D125 2846EA05    SHL  DWORD PTR DS:[5EA4628],1
05A66835    33C0        XOR  EAX,EAX
05A66837    EB 0B        JMP  SHORT b.05A66844
05A66839    FF05 90BAD705    INC  DWORD PTR DS:[5D7BA90]
05A6683F    A1 88BAD705    MOV  EAX,DWORD PTR DS:[5D7BA88]
05A66844    C2 0400    RETN 4

```

And all we need of this is:

```

009BFF24    A1 88BAD705    MOV  EAX,DWORD PTR DS:[5D7BA88]
009BFF29    C2 0400    RETN 4

```

Next

```

05A61AE5    FF15 4446EA05    CALL DWORD PTR DS:[5EA4644]
05A66850    8D6424 FC        LEA  ESP,DWORD PTR SS:[ESP-4]
05A66854    EB 00        JMP  SHORT b.05A66856
05A66856    893424        MOV  DWORD PTR SS:[ESI],ESI
05A66859    A1 94BAD705    MOV  EAX,DWORD PTR DS:[5D7BA94]
05A6685E    3B05 2C46EA05    CMP  EAX,DWORD PTR DS:[5EA462C]
05A66864    7C 70        JL  SHORT b.05A668D6
05A66866    BE D4BAD705    MOV  ESI,b.05D7BAD4
05A6686B    56        PUSH ESI
05A6686C    FF15 B8BAD705    CALL DWORD PTR DS:[5D7BAB8]
05A66872    68 98BAD705    PUSH b.05D7BA98
05A66877    68 C4BAD705    PUSH b.05D7BAC4
05A6687C    C705 98BAD705 10000000    MOV  DWORD PTR DS:[5D7BA98],10
05A66886    FF15 C0BAD705    CALL DWORD PTR DS:[5D7BAC0]
05A6688C    85C0        TEST EAX,EAX
05A6688E    75 24        JNZ  SHORT b.05A668B4
05A66890    813D 2C46EA05 00100000    CMP  DWORD PTR DS:[5EA462C],1000
05A6689A    7D 06        JGE  SHORT b.05A668A2

```



```

05A6689C    D125 2C46EA05    SHL  DWORD PTR DS:[5EA462C],1
05A668A2    8325 94BAD705 00    AND  DWORD PTR DS:[5D7BA94],0
05A668A9    56                PUSH ESI
05A668AA    FF15 BCBAD705    CALL DWORD PTR DS:[5D7BABC]                ; ntdll.RtlLeaveCriticalSection
05A668B0    33C0             XOR  EAX,EAX
05A668B2    EB 3A           JMP  SHORT b.05A668EE
05A668B4    56                PUSH ESI
05A668B5    FF15 BCBAD705    CALL DWORD PTR DS:[5D7BABC]                ; ntdll.RtlLeaveCriticalSection
05A668BB    8325 94BAD705 00    AND  DWORD PTR DS:[5D7BA94],0
05A668C2    813D 2C46EA05 00100000    CMP  DWORD PTR DS:[5EA462C],1000
05A668CC    7D 0E           JGE  SHORT b.05A668DC
05A668CE    D125 2C46EA05    SHL  DWORD PTR DS:[5EA462C],1
05A668D4    EB 06           JMP  SHORT b.05A668DC
05A668D6    FF05 94BAD705    INC  DWORD PTR DS:[5D7BA94]
05A668DC    8B4424 08        MOV  EAX,DWORD PTR SS:[ESP+8]
05A668E0    8B0D 98BAD705    MOV  ECX,DWORD PTR DS:[5D7BA98]
05A668E6    03C1             ADD  EAX,ECX
05A668E8    3305 C0BAD705    XOR  EAX,DWORD PTR DS:[5D7BAC0]                ; kernel32.GetComputerNameA
05A668EE    5E                POP  ESI                ; kernel32.7C816FD7
05A668EF    C2 0400          RETN 4

```

And proper patch of this code will be? Well do you know already?

```

009BFF2C    8B4424 04        MOV  EAX,DWORD PTR SS:[ESP+4]                ; ntdll.7C910738
009BFF30    8B0D 98BAD705    MOV  ECX,DWORD PTR DS:[5D7BA98]
009BFF36    03C1             ADD  EAX,ECX
009BFF38    35 A416827C      XOR  EAX,7C8216A4
009BFF3D    C2 0400          RETN 4

```

Again no esp+8 no need to push pop esi. Well now you wonder why XOR EAX, 7C8216A4

Look closer and you will see it XOR with address of GetComputerNameA, so for me address of this api is 7C8216A4

Next

```

05A61AF9    FF15 4846EA05    CALL DWORD PTR DS:[5EA4648]                ; b.05A66900

```

Here as you will follow you will see nothing is conditional or memory dependent:

```

08AF0000    E8 00000000      CALL b.08AF0005
08AF0005    58                POP  EAX                ; kernel32.7C816FD7
08AF0006    05 AA4EA605      ADD  EAX,b.05A64EAA
08AF000B    2D AF4EA605      SUB  EAX,b.05A64EAF
08AF0010    C3                RETN

```

So we leave it as it is

Next

```

05A61B1E    FF15 4C46EA05    CALL DWORD PTR DS:[5EA464C]                ; b.05A66910
05A66910    8D6424 FC        LEA  ESP,DWORD PTR SS:[ESP-4]
05A66914    892C24           MOV  DWORD PTR SS:[ESP],EBP
05A66917    8BEC            MOV  EBP,ESP
05A66919    8D6424 FC        LEA  ESP,DWORD PTR SS:[ESP-4]
05A6691D    890C24           MOV  DWORD PTR SS:[ESP],ECX
05A66920    8D6424 FC        LEA  ESP,DWORD PTR SS:[ESP-4]
05A66924    90                NOP
05A66925    893424           MOV  DWORD PTR SS:[ESP],ESI
05A66928    A1 68BAD705      MOV  EAX,DWORD PTR DS:[5D7BA68]
05A6692D    3B05 3046EA05    CMP  EAX,DWORD PTR DS:[5EA4630]
05A66933    7C 6B           JL   SHORT b.05A669A0
05A66935    BE F0BBD705      MOV  ESI,b.05D7BBF0
05A6693A    56                PUSH ESI
05A6693B    FF15 08BCD705    CALL DWORD PTR DS:[5D7BC08]                ; ntdll.RtlEnterCriticalSection
05A66941    8D45 FC        LEA  EAX,DWORD PTR SS:[EBP-4]
05A66944    50                PUSH EAX
05A66945    68 ECBAD705      PUSH b.05D7BAEC                ; ASCII "Human"
05A6694A    C745 FC 01010000 MOV  DWORD PTR SS:[EBP-4],101
05A66951    FF15 6CBAD705    CALL DWORD PTR DS:[5D7BA6C]                ; ADVAPI32.GetUserNameA
05A66957    85C0            TEST EAX,EAX
05A66959    75 23           JNZ  SHORT b.05A6697E
05A6695B    2105 68BAD705    AND  DWORD PTR DS:[5D7BA68],EAX
05A66961    813D 3046EA05 00100000    CMP  DWORD PTR DS:[5EA4630],1000

```

```

05A6696B 7D 06 JGE SHORT b.05A66973
05A6696D D125 3046EA05 SHL DWORD PTR DS:[5EA4630],1
05A66973 56 PUSH ESI
05A66974 FF15 0CBCD705 CALL DWORD PTR DS:[5D7BC0C] ; ntdll.RtlLeaveCriticalSection
05A6697A 33C0 XOR EAX,EAX
05A6697C EB 3B JMP SHORT b.05A669B9
05A6697E 56 PUSH ESI
05A6697F FF15 0CBCD705 CALL DWORD PTR DS:[5D7BC0C] ; ntdll.RtlLeaveCriticalSection
05A66985 8325 68BAD705 00 AND DWORD PTR DS:[5D7BA68],0
05A6698C 813D 3046EA05 00100000 CMP DWORD PTR DS:[5EA4630],1000
05A66996 7D 0E JGE SHORT b.05A669A6
05A66998 D125 3046EA05 SHL DWORD PTR DS:[5EA4630],1
05A6699E EB 06 JMP SHORT b.05A669A6
05A669A0 FF05 68BAD705 INC DWORD PTR DS:[5D7BA68]
05A669A6 A1 F4BAD705 MOV EAX,DWORD PTR DS:[5D7BAF4]
05A669AB 8B0D 04BBD705 MOV ECX,DWORD PTR DS:[5D7BB04]
05A669B1 03C1 ADD EAX,ECX
05A669B3 0305 ECBAD705 ADD EAX,DWORD PTR DS:[5D7BAEC]
05A669B9 5E POP ESI ; kernel32.7C816FD7
05A669BA C9 LEAVE
05A669BB C2 0400 RETN 4

```

So proper patch is:

```

009BFF40 A1 F4BAD705 MOV EAX,DWORD PTR DS:[5D7BAF4]
009BFF45 8B0D 04BBD705 MOV ECX,DWORD PTR DS:[5D7BB04]
009BFF4B 03C1 ADD EAX,ECX
009BFF4D 0305 ECBAD705 ADD EAX,DWORD PTR DS:[5D7BAEC]
009BFF53 C2 0400 RETN 4

```

And finally last one is

```

05A61B32 FF15 5046EA05 CALL DWORD PTR DS:[5EA4650] ; b.05A669C0

05A669C0 8B15 74BAD705 MOV EDX,DWORD PTR DS:[5D7BA74] ; ADVAPI32.77DC0000
05A669C6 66:813A 4D5A CMP WORD PTR DS:[EDX],5A4D
05A669CB 75 0C JNZ SHORT b.05A669D9
05A669CD A1 78BAD705 MOV EAX,DWORD PTR DS:[5D7BA78]
05A669D2 66:8138 4D5A CMP WORD PTR DS:[EAX],5A4D
05A669D7 74 04 JE SHORT b.05A669DD
05A669D9 33C0 XOR EAX,EAX
05A669DB EB 38 JMP SHORT b.05A66A15
05A669DD 8B48 3C MOV ECX,DWORD PTR DS:[EAX+3C]
05A669E0 03C8 ADD ECX,EAX
05A669E2 8B42 3C MOV EAX,DWORD PTR DS:[EDX+3C]
05A669E5 03D0 ADD EDX,EAX
05A669E7 8B42 58 MOV EAX,DWORD PTR DS:[EDX+58]
05A669EA 0B42 28 OR EAX,DWORD PTR DS:[EDX+28]
05A669ED 56 PUSH ESI
05A669EE 0B42 08 OR EAX,DWORD PTR DS:[EDX+8]
05A669F1 8B71 58 MOV ESI,DWORD PTR DS:[ECX+58]
05A669F4 0B71 28 OR ESI,DWORD PTR DS:[ECX+28]
05A669F7 0B71 08 OR ESI,DWORD PTR DS:[ECX+8]
05A669FA 03C6 ADD EAX,ESI
05A669FC 0FB772 42 MOVZX ESI,WORD PTR DS:[EDX+42]
05A66A00 0FB752 40 MOVZX EDX,WORD PTR DS:[EDX+40]
05A66A04 03C6 ADD EAX,ESI
05A66A06 03C2 ADD EAX,EDX ; ntdll.KiFastSystemCallRet
05A66A08 0FB751 42 MOVZX EDX,WORD PTR DS:[ECX+42]
05A66A0C 0FB749 40 MOVZX ECX,WORD PTR DS:[ECX+40]
05A66A10 03C2 ADD EAX,EDX ; ntdll.KiFastSystemCallRet
05A66A12 03C1 ADD EAX,ECX
05A66A14 5E POP ESI ; kernel32.7C816FD7
05A66A15 C2 0400 RETN 4

```

What it does? Well calculates CRC of advapi32.dll, with other dll version, language all will be wrong. So, the proper patch will be just to return the value of EAXx, just before returning. For me it is:

```

009BFF56 B8 E0F7CB85 MOV EAX,85CBF7E0
009BFF5B C2 0400 RETN 4

```

Now final move is update address table of antidumps to our patches so at 5EA4634 we will binary paste:

```
00 FF 9B 00 0C FF 9B 00 18 FF 9B 00 24 FF 9B 00 2C FF 9B 00 00 69 A6 05 40 FF 9B 00 56 FF 9B 00
```

It's better to not touch addresses and opcodes due securom also uses those as encryption, when it calculates some value in EAX or other register it likes to do ROL with value of some memory.

For example from

```
05A61B32      FF15 5046EA05          CALL DWORD PTR DS:[5EA4650]
```

it can do:

```
ROL EAX, [05A61B34]
```

so it does ROL EAX, 50 when we change address in that call then encryption is screwed.

It's like a mine field or small CRCs, you must really watch out what you change even with CRCchecks fixed.

Next step is:

```
05ADC7D5      E8 18440000          CALL a.05AE0BF2
05ADC7DA      8B48 14             MOV ECX,DWORD PTR DS:[EAX+14]
05ADC7DD      69C9 FD430300       IMUL ECX,ECX,343FD
05ADC7E3      81C1 C39E2600       ADD ECX,269EC3
05ADC7E9      8948 14             MOV DWORD PTR DS:[EAX+14],ECX
```

And inside 5AE0BF2 we have:

```
05AE0BF2      53                 PUSH EBX
05AE0BF3      56                 PUSH ESI
05AE0BF4      FF15 D8990306       CALL DWORD PTR DS:[60399D8]          ; ntdll.RtlGetLastWin32Error
05AE0BFA      FF35 7473F405       PUSH DWORD PTR DS:[5F47374]
05AE0C00      8BD8              MOV EBX,EAX
05AE0C02      FF15 2CC3D705       CALL DWORD PTR DS:[5D7C32C]          ; kernel32.TlsGetValue
05AE0C08      8BF0              MOV ESI,EAX
05AE0C0A      85F6              TEST ESI,ESI
05AE0C0C      75 49             JNZ SHORT b.05AE0C57
05AE0C0E      68 8C000000        PUSH 8C
05AE0C13      6A 01             PUSH 1
05AE0C15      E8 E7D0FFFF        CALL b.05ADDD01
05AE0C1A      8BF0              MOV ESI,EAX
05AE0C1C      85F6              TEST ESI,ESI
05AE0C1E      59                 POP ECX          ; kernel32.7C816FD7
05AE0C1F      59                 POP ECX          ; kernel32.7C816FD7
05AE0C20      74 2D             JE SHORT b.05AE0C4F
05AE0C22      56                 PUSH ESI
05AE0C23      FF35 7473F405       PUSH DWORD PTR DS:[5F47374]
05AE0C29      FF15 30C3D705       CALL DWORD PTR DS:[5D7C330]          ; kernel32.TlsSetValue
05AE0C2F      85C0              TEST EAX,EAX
05AE0C31      74 1C             JE SHORT b.05AE0C4F
05AE0C33      C746 54 2877F405    MOV DWORD PTR DS:[ESI+54],b.05F47728
05AE0C3A      C746 14 01000000    MOV DWORD PTR DS:[ESI+14],1
05AE0C41      FF15 BC9A0306       CALL DWORD PTR DS:[6039ABC]          ; kernel32.GetCurrentThreadId
05AE0C47      834E 04 FF         OR DWORD PTR DS:[ESI+4],FFFFFFFF
05AE0C4B      8906              MOV DWORD PTR DS:[ESI],EAX
05AE0C4D      EB 08             JMP SHORT b.05AE0C57
05AE0C4F      6A 10             PUSH 10
05AE0C51      E8 0DCAFFFF        CALL b.05ADD663
05AE0C56      59                 POP ECX          ; kernel32.7C816FD7
05AE0C57      53                 PUSH EBX
05AE0C58      FF15 209B0306       CALL DWORD PTR DS:[6039B20]          ; ntdll.RtlSetLastWin32Error
05AE0C5E      8BC6              MOV EAX,ESI
05AE0C60      5E                 POP ESI          ; kernel32.7C816FD7
05AE0C61      5B                 POP EBX          ; kernel32.7C816FD7
05AE0C62      C3                 RETN
```

So as I said we must return 1, assemble just there a MOV EAX,1 and RET and all is fine.

You can now ask why not change call to 05AE0BF2 into `MOV EAX, 1` it's also 5 bytes? Well, because I bet that E8 from call is used as ROL in some place. I have already seen that in older securom when I tried to fix E8 call that leads to some winapi to point to my IAT table jump.

Well its not over yet with `RtlGetLastWin32Error`! Why?

Lets look at:

```
05ADC7DA    8B48 14          MOV ECX,DWORD PTR DS:[EAX+14]
05ADC7E9    8948 14          MOV DWORD PTR DS:[EAX+14],ECX
```

When we return 1 and add 14 then we have 15 (lol I'm so good at math :P, but I still don't know how much is 2\*2 :P) and when those two execute then we get page fault :( So only possible way is to NOP those two and all is fine.

Next step is:

```
05BB8F15    E8 2CB0C7FF      CALL a.05833F46
05BB8F1A    83FE 04          CMP ESI,4
05BB8F1D    7C 05           JL SHORT a.05BB8F24
```

And 5833F46 after many instructions leads to:

```
05833FA6    68 EC55FD05      PUSH b.05FD55EC          ; ASCII "GetSystemInfo"
05833FAB    68 1454FD05      PUSH b.05FD5414          ; ASCII "KERNEL32.dll"
05833FB0    FF15 A09A0306    CALL DWORD PTR DS:[6039AA0] ; kernel32.GetModuleHandleA
05833FB6    50              PUSH EAX
05833FB7    FF15 5C9A0306    CALL DWORD PTR DS:[6039A5C] ; b.05813282
05833FBD    A3 E066D505      MOV DWORD PTR DS:[5D566E0],EAX
05833FC2    EB 05           JMP SHORT b.05833FC9
```

With `GetProcAddress` and call so only possible solution is to copy those 20 bytes, paste them at 9BFF7A, change that call to point into our patch and it looks like it:

```
009BFF5E    E8 00000000      CALL b.009BFF63
009BFF63    5E              POP ESI                  ; kernel32.7C816FD7
009BFF64    83C6 19          ADD ESI,19
009BFF67    57              PUSH EDI                 ; ntdll.7C910738
009BFF68    8B7C24 08        MOV EDI,DWORD PTR SS:[ESP+8]
009BFF6C    B9 24000000      MOV ECX,24
009BFF71    F3:A4           REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
009BFF73    BE 10000000      MOV ESI,10
009BFF78    5F              POP EDI                  ; kernel32.7C816FD7
009BFF79    C2 0400         RETN 4
```

And finally again at

```
05BBD15E    FF15 A49A0306    CALL DWORD PTR DS:[6039AA4]
```

This one can be replaced with

```
MOV EAX, 75C          ;my PID
```

And we are almost home. When you now run exe it will work, but only on your machine, but goal is to make it run on any other.

Now after we patched all antidump apis run this script: Securom 7.x Jump Bridge & Crypted Code Fixer.txt (included into goodies folder).

Do you see it? And where it goes?

Address	Hex dump	Disassembly	Comment
0041B0A9	FF25 1CDC0606	JMP DWORD PTR DS:[606DC1C]	b.060E7E60
0041B0B6	EF	OUT DX, EAX	I/O command
0041B0F7	C8 050600	ENTER 605,0	
0041B0FB	0000	ADD BYTE PTR DS:[EAX],AL	
0041B0FD	0000	ADD BYTE PTR DS:[EAX],AL	
0041B0FF	0000	ADD BYTE PTR DS:[EAX],AL	
0041B101	0000	ADD BYTE PTR DS:[EAX],AL	
0041B103	0000	ADD BYTE PTR DS:[EAX],AL	
0041B105	0033	ADD BYTE PTR DS:[EBX],DH	
0041B107	C9	LEAVE	
0041B108	05 06000000	ADD EAX,6	
0041B10D	0000	ADD BYTE PTR DS:[EAX],AL	
0041B10F	0000	ADD BYTE PTR DS:[EAX],AL	
0041B111	0000	ADD BYTE PTR DS:[EAX],AL	
0041B113	0000	ADD BYTE PTR DS:[EAX],AL	
0041B115	0077 C9	ADD BYTE PTR DS:[EDI-37],DH	
0041B118	05 06000000	ADD EAX,6	
0041B11D	0000	ADD BYTE PTR DS:[EAX],AL	
0041B11F	0000	ADD BYTE PTR DS:[EAX],AL	
0041B121	0000	ADD BYTE PTR DS:[EAX],AL	
0041B123	0000	ADD BYTE PTR DS:[EAX],AL	
0041B125	00E9	ADD CL,CH	
0041B127	FD	STD	
0041B128	96	XCHG EAX,ESI	
0041B129	FE	INC	
0041B12A	FFEB	JMP FAR EBX	Unknown command
0041B12C	AS CCCCCCCC	TEST EAX,CCCCCCCC	Illegal use of register
0041B131	CC	INT3	
0041B132	CC	INT3	
0041B133	CC	INT3	
0041B134	CC	INT3	
0041B135	CC	INT3	
0041B136	CC	INT3	
0041B137	CC	INT3	
0041B138	CC	INT3	
0041B139	CC	INT3	
0041B13A	CC	INT3	
0041B13B	CC	INT3	
0041B13C	CC	INT3	

DS:[606DC1C]=060E7E60 (b.060E7E60)

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FFDF000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	8054A6ED	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	39142020	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	End of SEH chain
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C839A88	SE handler
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C816FE0	kernel32.7C816FE0
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	00966D9C	b.<ModuleEntryPoint>
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFC	00000000	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 DA A7 96 00 00 00 00 00 00 00 00 00 00	.....r&P.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command:

Program entry point:

Paused

What here script does it will set HWBP on write into that DWORD when securom will update it with new address.  
That's how it looks now:

Address	Hex dump	Disassembly	Comment
060E7E60	68 6A7E0E06	PUSH b.060E7E6A	
060E7E65	E8 F60B72FF	CALL b.06015A60	
060E7E6A	6A DA	PUSH -26	
060E7E6C	0C 06	OR AL, 6	
060E7E6E	90	NOP	
060E7E6F	00	INS	I/O command
060E7E70	0000	ADD DWORD PTR ES:[EDI], 0x	
060E7E72	7A 5D	JPE SHORT b.060E7E01	
060E7E74	0000	ADD BYTE PTR DS:[EAX], AL	
060E7E76	00 02000000	CHP EAX, 2	
060E7E78	0000	ADD BYTE PTR DS:[EAX], AL	
060E7E7D	0000	ADD BYTE PTR DS:[EAX], AL	
060E7E7F	0068 9A	ADD BYTE PTR DS:[EAX-66], CH	
060E7E82	7E 0E	JLE SHORT b.060E7E45	
060E7E84	06	PUSH ES	
060E7E85	68 3C164000	PUSH b.0040163C	
060E7E8A	68 B8368305	PUSH b.05833688	
060E7E8F	9C	PUSHFD	
060E7E90	16C24 04 58DC0100	SUB DWORD PTR SS:[ESP+4], 1DC58	
060E7E98	9D	POPF	
060E7E99	C3	RETN	
060E7E9A	42	INC EDX	ntdll.KiFastSystemCallRet
060E7E9B	E1 0D	LOOPDE SHORT b.060E7E9A	
060E7E9D	06	PUSH ES	
060E7E9E	CA 4F00	RETF 4F	Far return
060E7EA1	00C1	ADD CL, AL	
060E7EA3	40	INC EAX	
060E7EA4	0000	ADD BYTE PTR DS:[EAX], AL	
060E7EA6	E9 1C000000	JMP b.060E7EC7	
060E7EAB	0000	ADD BYTE PTR DS:[EAX], AL	
060E7EAD	0000	ADD BYTE PTR DS:[EAX], AL	
060E7EAF	0068 9A	ADD BYTE PTR DS:[EAX-46], CH	
060E7EB2	7E 0E	JLE SHORT b.060E7EC2	
060E7EB4	06	PUSH ES	
060E7EB5	E8 A60B72FF	CALL b.05815A60	
060E7EBA	D8E0	FISUB EAX	Illegal use of register
060E7EBC	0D 06321800	OR EAX, 183206	
060E7EC1	000B	ADD BYTE PTR DS:[EBX], CL	
060E7EC3	3B00	CMPI PTR DS:[EAX], AL	

060E7E6A=b.060E7E6A

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C910738	RETURN to kernel32.7C816F07
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FFDF000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	8054A6ED	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	89142020	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	End of SEH chain
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C899A88	SE handler
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C816FE0	kernel32.7C816FE0
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	00966D9C	b.<ModuleEntryPoint>
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFC	00000000	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 DA A7 96 00 00 00 00 00 00 00 00 00 00	.....r2f.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command:

Program entry point:

Paused



Well after write it looks like it:

Address	Hex dump	Disassembly	Comment
0041B0F0	FF25 1C0C0606	JMP DWORD PTR DS:[006DC1C]	b.005EEDF0
0041B0F6	EF	OUT DX, EAX	I/O command
0041B0F7	C8 050600	ENTER 605,0	
0041B0FB	0000	ADD BYTE PTR DS:[EAX],AL	
0041B0FD	0000	ADD BYTE PTR DS:[EAX],AL	
0041B0FF	0000	ADD BYTE PTR DS:[EAX],AL	
0041B101	0000	ADD BYTE PTR DS:[EAX],AL	
0041B103	0000	ADD BYTE PTR DS:[EAX],AL	
0041B105	0033	ADD BYTE PTR DS:[EBX],DH	
0041B107	C9	LEAVE	
0041B109	05 06000000	ADD EAX,6	
0041B10D	0000	ADD BYTE PTR DS:[EAX],AL	
0041B10F	0000	ADD BYTE PTR DS:[EAX],AL	
0041B111	0000	ADD BYTE PTR DS:[EAX],AL	
0041B113	0000	ADD BYTE PTR DS:[EAX],AL	
0041B115	0077 C9	ADD BYTE PTR DS:[EDI-37],DH	
0041B118	05 06000000	ADD EAX,6	
0041B11D	0000	ADD BYTE PTR DS:[EAX],AL	
0041B11F	0000	ADD BYTE PTR DS:[EAX],AL	
0041B121	0000	ADD BYTE PTR DS:[EAX],AL	
0041B123	0000	ADD BYTE PTR DS:[EAX],AL	
0041B125	00E9	ADD CL,CH	
0041B127	FD	STD	
0041B128	96	XCHG EAX,ESI	
0041B129	FE	INC EDI	
0041B12A	FFEB	JMP FAR EBX	Unknown command Illegal use of register
0041B12C	A9 CCCCCCCC	TEST EAX,CCCCCCCC	
0041B131	CC	INT3	
0041B132	CC	INT3	
0041B133	CC	INT3	
0041B134	CC	INT3	
0041B135	CC	INT3	
0041B136	CC	INT3	
0041B137	CC	INT3	
0041B138	CC	INT3	
0041B139	CC	INT3	
0041B13A	CC	INT3	
0041B13B	CC	INT3	
0041B13C	CC	INT3	

DS:[006DC1C]=005EEDF0 (b.005EEDF0)

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF94	634C957E	
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF98	00000246	
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FF9C	7C910738	ntdll.7C910738
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFA0	FFFFFFFF	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFA4	0022FFF0	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFA8	0022FFBC	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFAC	7FFDF000	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFB0	7C90EB94	ntdll.KiFastSystemCallRet
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFB4	0022FFB0	
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFB8	00000000	
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFBC	060E7E6A	RETURN to b.060E7E6A from b.05815A60
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC0	060E7E6A	RETURN to b.060E7E6A from b.05815A60
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FFDF000	
00A21100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	0054A6ED	
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	89142020	
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	End of SEH chain
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C839A00	SE handler
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C816FE0	kernel32.7C816FE0

Command:

Hardware BreakPoint 1 at b.071A00AA - EIP points to next instruction

Paused

Well at this point we could finish and go to next jump bridge, but as you can see I have in script now HWBP on execute. It's not needed here, just to make loop universal for crypted code, instead of code splicing or virtualized code by securom it can lead to crypted code. That we must execute on our machine to decrypt it, due inside there is crypted CPUID check and will not uncrypt on other CPU just crash. So let analyze another jump bridge:

```
004DDD50 - FF25 54E40606      JMP DWORD PTR DS:[606E454] ; b.06073A10
```

That leads us to:

Address	Hex dump	Disassembly	Comment
06073A10	E8 262082FF	CALL b.05895A3B	
06073A15	0033	ADD BYTE PTR DS:[EBX],0H	
06073A17	C041 E1 6A	ROL BYTE PTR DS:[ECX-1F],6A	Shift constant out of range 1..31
06073A1B	8250 2A 65	ADC BYTE PTR DS:[EAX+2A],65	
06073A1F	06	PUSH ES	
06073A20	96	XCHG EAX,ESI	
06073A21	27	DAA	
06073A22	D3DF	RCR EDI,CL	
06073A24	8E5C69 3D	MOV DS,WORD PTR DS:[ECX+EBP*2+3D]	Modification of segment register
06073A28	D858 8A	FCMP DWORD PTR DS:[EAX-76]	
06073A2B	00C3	ADD BL,AL	
06073A2D	27	DAA	
06073A2E	E0 87	LODPD SHORT b.060739B7	
06073A30	C7	Unknown command	
06073A31	2865 F0	SUB BYTE PTR SS:[EBP-10],AH	
06073A34	43	INC EBX	
06073A35	E4 67	IN AL,67	I/O command
06073A37	06	PUSH ES	
06073A38	C568 A3	LDS EBP,FWORD PTR DS:[EAX-5D]	Modification of segment register
06073A3B	8018 2A	SBB BYTE PTR DS:[EAX],2A	
06073A3E	65 06	PUSH ES	Superfluous prefix
06073A40	C5C42 8B	LDS EBX,FWORD PTR DS:[EDX+EAX*2-75]	Modification of segment register
06073A44	C8 1C5A09	ENTER 5A1C,9	
06073A48	C3	RETN	
06073A49	A9 8C7EF828	TEST EAX,28F87E8C	
06073A4E	65 1F9	STC	Superfluous prefix
06073A50	CC	INT3	
06073A51	27	DAA	
06073A52	E1 9B	LODPD SHORT b.060739E9	
06073A54	06	Unknown command	
06073A55	A3 9F8D0EAB	MOV DWORD PTR DS:[AB0E8D9F],EAX	
06073A5A	8C06	MOV WORD PTR DS:[ESI],ES	
06073A5C	B1 2F	MOV CL,2F	
06073A5E	2C 73	SUB AL,73	
06073A60	CC	INT3	
06073A61	1BBE EDC09364	SBB EDI,DWORD PTR DS:[ESI+6493C0ED]	
06073A67	06	PUSH ES	
06073A68	C528	LDS EBP,FWORD PTR DS:[EAX]	Modification of segment register
06073A6A	E8 0BA2186A	CALL 701FDC7A	
05895A3B=b.05895A3B			

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FDE0000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	00000640	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	8928A1A8	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	End of SEH chain
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C839AA8	SE handler
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C816FE0	kernel32.7C816FE0
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	00966D9C	b.<ModuleEntryPoint>
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFF	00000000	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command:

Module D:\WINXP\WinSxS\x86\_Microsoft.Windows.Common-Controls\_6595b64144ccf1df\_6.0.2600.2982\_x-ww\_ac3f9c03\comctl32.dll

Paused

You see that call? Its call to decrypted code at and after address of that call, so after execute and HWBP on execute on 4DDD50 we get:

Address	Hex dump	Disassembly	Comment
06073A10	8A8E 00020000	MOV CL, BYTE PTR DS:[ESI+200]	
06073A16	33C9	XOR EBX, EBX	
06073A18	84C9	TEST CL, CL	
06073A1A	0F84 95020000	JE b.06073A65	
06073A20	53	PUSH EBX	
06073A21	0FB6D9	MOVZX EBX, CL	
06073A24	4B	DEC EBX	
06073A25	74 0C	JE SHORT b.06073A33	
06073A27	3B1D 70EF0606	CMPL EBX, DWORD PTR DS:[606EF70]	
06073A2D	0F85 81020000	JNE b.06073A64	
06073A33	F686 C0020000 40	TEST BYTE PTR DS:[ESI+2CC], 40	
06073A3A	C686 D0020000 00	MOV BYTE PTR DS:[ESI+2DD], 0	
06073A41	74 27	JE SHORT b.06073A64	
06073A43	8D00 343F0F06	LEA ECX, DWORD PTR DS:[60F3F34]	
06073A49	91E9 763D0000	SUB ECX, 3D78	
06073A4F	FF09	DEC DWORD PTR DS:[ECX]	
06073A51	0F84 9D1E8BFA	JE b.009253F4	
06073A57	8BC8	MOV ECX, EBX	
06073A59	33E9 00	XOR ECX, 0	
06073A5C	74 07	JE SHORT b.06073A65	
06073A5E	49	DEC ECX	
06073A5F	75 09	JNZ SHORT b.06073A6A	
06073A61	3306	XOR EBX, EBX	
06073A63	EB 05	JMP SHORT b.06073A6A	
06073A65	BB 01000000	MOV EBX, 1	
06073A6A	8D00 67300F06	LEA ECX, DWORD PTR DS:[60F3067]	
06073A70	81E9 A42E0000	SUB ECX, 2E84	
06073A76	FF09	DEC DWORD PTR DS:[ECX]	
06073A78	0F84 E7D79CFA	JE b.00941265	
06073A7E	8BC8	MOV ECX, EBX	
06073A80	33E9 00	SUB ECX, 0	
06073A83	74 2B	JE SHORT b.06073A80	
06073A85	49	DEC ECX	
06073A86	75 46	JNZ SHORT b.06073A8E	
06073A88	53	PUSH EBX	
06073A89	90	NOOP	
06073A8A	8D1D EA9C0F06	LEA EBX, DWORD PTR DS:[60F9CEA]	
06073A90	81EB 209B0000	SUB EBX, 9B20	
06073A96	FF0B	DEC DWORD PTR DS:[EBX]	

Registers (FPU)

EAX 00000000  
ECX 0022FF00  
EDX 7C90EB94 ntdll.KiFastSystemCallRet  
EBX 7FFDE000  
ESP 0022FFC4  
EBP 0022FFFF  
ESI FFFFFFFF  
EDI 7C910738 ntdll.7C910738  
EIP 06073A10 b.06073A10

C 0 ES 0023 32bit 0(FFFFFFFF)  
P 1 CS 001B 32bit 0(FFFFFFFF)  
A 0 SS 0023 32bit 0(FFFFFFFF)  
Z 1 DS 0023 32bit 0(FFFFFFFF)  
S 0 FS 0006 32bit 7FDD0000(FFF)  
T 0 GS 0000 NULL  
D 0  
O 0 LastErr ERROR\_MOD\_NOT\_FOUND (0000007E)  
EFL 00000246 (NO, NB, E, BE, NS, PE, GE, LE)  
ST0 empty -UNORM D0A8 01050104 00000000  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.99609375000000000000  
ST5 empty 0.999609374999999943601  
ST6 empty 1.00000000000000000000  
ST7 empty 1.00000000000000000000

FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ)  
FCW 027F Prec NEAR, S3 Mask 1 1 1 1 1 1

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816F07	RETURN to kernel32.7C816F07
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFCC	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	7FFDE000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	00000640	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	8928A1A8	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	FFFFFFFF	End of SEH chain
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	7C839AA8	SE handler
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C816FE0	kernel32.7C816FE0
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	00966D9C	b.<ModuleEntryPoint>
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFC	00000000	
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command: Hardware BreakPoint 1 at b.06073A10

Paused

Nice uncrpyted code that will stay same after we again dump and run on any other machine, due no more checks there. Run whole scrip to achive that. Again 54890ms (only asm can make it faster)

And now last part to fix is CPUID checks that can be in spliced code.

Run Securom 7.x Cpuid Fixer.txt to fix all those (included into the goodies folder of this tutorials original package):

Address	Hex dump	Disassembly	Comment
06A32544	B9 01000000	MOV ECX, 1	
06A3254F	C1E1 03	SHL ECX, 3	
06A32552	D1E9	SHR ECX, 1	
06A325B4	01D9	ADD ECX, EBX	
06A325B6	8101 04000000	ADD DWORD PTR DS:[ECX], 4	
06A325BC	C1E2 02	SHL EDX, 2	
06A325BF	01C2	ADD EDX, EAX	
06A325C1	52	PUSH EDX	
06A325C2	B9 72000000	MOV ECX, 72	
06A325C7	81F1 7E000000	XOR ECX, 7E	
06A325CD	01D9	AND ECX, EBX	
06A325CF	8B01	MOV EDI, DWORD PTR DS:[ECX]	
06A325D1	8B1424	MOV EDI, DWORD PTR SS:[ESP]	
06A325D4	8B12	MOV EDI, DWORD PTR DS:[EDX]	
06A325D6	891424	MOV DWORD PTR SS:[ESP], EDX	
06A325D9	50	PUSH EAX	
06A325DA	B8 01000000	MOV EAX, 1	
06A325DF	53	PUSH EBX	
06A325E0	0FA2	CPUID	
06A325E2	5B 0FFFFFFF	MOV EAX, FFFFFFFD	
06A325E7	5B	POP EBX	
06A325E8	88C1	MOV CL, AL	
06A325EA	D34C24 04	ROR DWORD PTR SS:[ESP+4], CL	
06A325EE	5B	POP EAX	
06A325EF	010424	ADD DWORD PTR SS:[ESP], EAX	
06A325F2	58	POP EAX	
06A325F3	FFEB	JMP EAX	
06A325F5	BC 03A373D4	MOV ESP, D473A303	
06A325FA	5D	POP ESP	
06A325FB	05 84	AND 84	
06A325FD	5C	POP ESP	
06A325FE	5C	POP ESP	
06A325FF	4B	DEC ECX	
06A32600	00 0021001D	OP EAX, 1D002100	
06A32605	010F	ADD DWORD PTR DS:[EDI], ECX	
06A32607	0081 C3240000	ADD BYTE PTR DS:[ECX+24C3], AL	
06A3260D	00C7	ADD BH, AL	
06A3260F	0300	ADD EDI, DWORD PTR DS:[EAX]	
06A32611	0000	ADD BYTE PTR DS:[EAX], AL	

Registers (FPU)	
EAX	000000D6
ECX	0000E38D
EDX	8FEB8FFF
EBX	00020800
ESP	0022FFC4
EBP	0022FFD0
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	0096D09C b.<ModuleEntryPoint>
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 0018 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_MOD_NOT_FOUND (0000007E)
EFL	00000246 (NO, NB, E, BE, NS, PE, GE, LE)
ST0	empty -UNORM D0A8 01050104 00000000
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.99609375000000000000
ST5	empty 0.99609374999999993601
ST6	empty 1.00000000000000000000
ST7	empty 1.00000000000000000000
FST 4020	Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ)
FCW 027F	Prec NEAR, S3 Mask 1 1 1 1 1 1

Address	Hex dump	ASCII	Address	Value	Comment
00A21000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC4	7C816FD7	RETURN to kernel32.7C816FD7
00A21010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFC8	7C910738	ntdll.7C910738
00A21020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD0	FFFFFFFF	
00A21030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD4	7FFDC000	
00A21040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFD8	8054A6ED	
00A21050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFDC	0022FFC8	
00A21060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE0	892E6638	
00A21070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE4	FFFFFFFF	End of SEH chain
00A21080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFE8	7C839A88	SE handler
00A21090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFEC	7C816FE0	kernel32.7C816FE0
00A210A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF0	00000000	
00A210B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF4	00000000	
00A210C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFF8	0096D09C	b.<ModuleEntryPoint>
00A210D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0022FFFC	00000000	
00A210E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A210F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			
00A21150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....			

Command: [ ] Paused

What script does is to search for CPUID and checks if there is an instruction like "and eax,FFFFFFFD", just after the CPUID also there can be EB jump to "and eax,FFFFFFFD", so we must also handle it. Simplest way to fix it is to replace "and eax,FFFFFFFD" with a "mov eax,6D6" that is my CPUID after and'ing. Script gets your CPUID automatically so you don't need to modify it.

## 9. Conclusions

Finally we are at end and can now dump and again fix IAT with Winhex. It should run now on any pc we like. Of course you can dump just once at end, but I did this few times due its better for me to write this tut and get pictures of code so I could paste them here. I hope you enjoyed it and learned something.

Best regards to: all scene competition, all people that make scene alive, ARTeam, exetools, SND and unpack.cn.

Human/MiNT

# SecuROM for the masses by deroko

---

## 1. Forewords

SecuROM is a famous protection used by many games nowadays. Funny thing is that it's cracking doesn't take too much time, only takes time when you are doing it for the first time, after that it goes flowless. There aren't tutorials about SecuROM in the public as far as I'm aware. First tutorial which deals with SecuROM 7.xx was submitted to ARTeam by Anonymou\$ author, so I think we should write more about it, just for fun...

## 2. Tools and Target

Target that we will be using is Command & Conquer : Tiberium Wars v1.0, so you will need to get that DVD to follow this tut. Well any SecuROM will work as approach is kinda generic.

Tools:

- SoftICE
- Olly only for kewl screenshots...
- IDA
- Asm/C compiler

Also make sure that you have original DVD as game developers deserve money for their work...

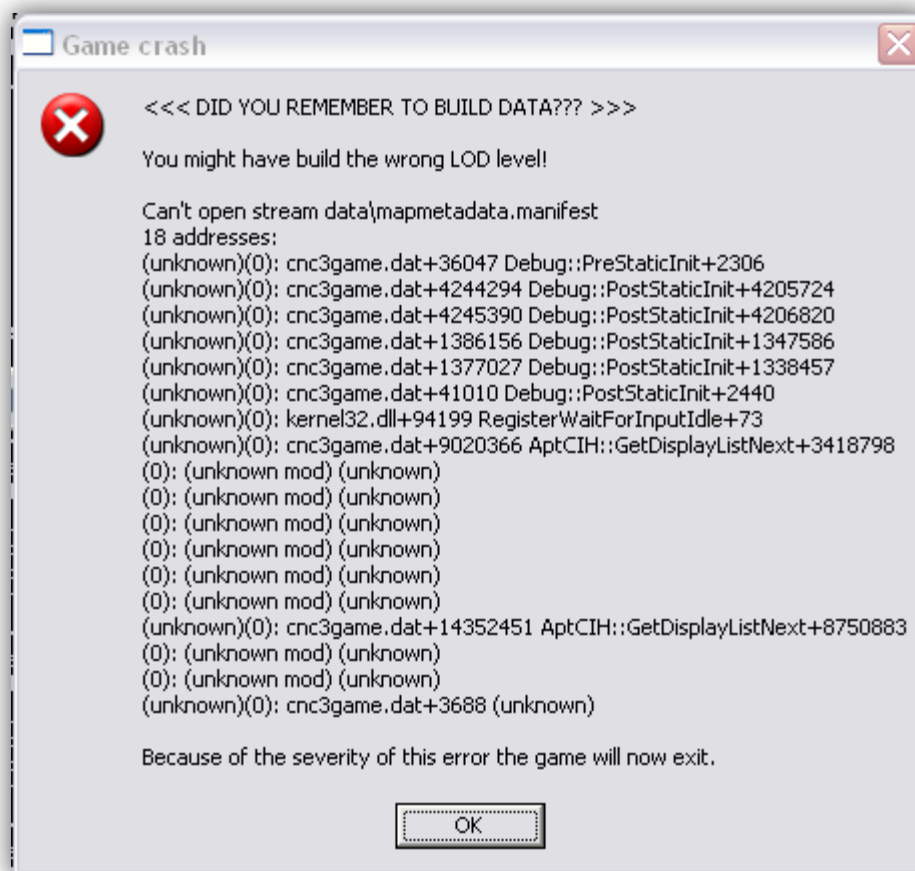
To run this game you will have to copy cnc3game.dat from RetailExe\1.0 to folder where is located cnc3.exe and simply type:

**cnc3game.dat -win -config CNC3\_english\_1.0.SkuDef**

-win is for windowed mode ☺

Of course, you don't have to do this while you are unpacking protection, it is only required when you are testing your dump, otherwise you will get this message if you run dump without that command line:





Similar output you will get when your dumped file is started without proper command line.

### 3. Few words about SecuROM

SecuROM protection consists of one .exe attached to original .exe. If you trace a little bit through SecuROM layer you will see 2<sup>nd</sup> exe being appended, even standard MSVC initialization routine are held in this upper layer. If you want you may dump at this layer and analyze securom protection, but that is not very important for us atm.

SecuROM uses a lot of buffers to make itself anti-dump which are always allocated on different base addresses which leads us to simple conclusion that some kind of random generator is used to allocate those buffers. Also SecuROM uses several anti-dump tricks such as GetCurrentProcessId, OpenEventA/CloseHandle, ResetEvent, TlsSet/GetValue, entrypoint address, imagesize from PE header etc... which we can patch of course.

It allocates a lot of buffers to execute its code and when dumped those buffers take ~40mb. Luckily when compressed they take around 1MB or less, as all redirections are allocated at 64K boundary but only 1KB is committed. This is result of calling VirtualAlloc a lot of times (try making infinite loop with VirtualAlloc and watch what happens). Thus 64K-1K is padded with zeros in your dump and giving really really good compression ratio. Just for comparison, I have dump of 70MB compressed back to 7MB, well, almost like virgin file ☺ Well I'll also discuss about making this dump smaller : virgin + ~20mb.

Before we even think to fire up SoftICE with SecuROM protection we have to hide it properly, well, my softice is hidden it this way:

- NtCreateFile
- Int1/int3/int41 patched
- NtQuerySystemInformation
- UnhandledExceptionFilter

One more way remains to detect SoftICE and that's to query it's service and check if it is active, for this I use hook of OpenServiceA to eliminate NTICE service opening. This hook code and other stuff used to log SecuROM execution may be found in hookdll.c. Also SoftICE activity can be detected with EnumServicesStatusExA which is used by ActiveMARK but that's another story.

## 4. Dumping SecuROM

There are a few ways to dump SecuROM at the OEP:

- Use method described by anonymous author [see first chapter of this document]
- Find vm\_exit, hook it, and wait when it jumps back to 1<sup>st</sup> section
- Hook commonly used APIs at MSVC oep and watch when those are called from 1<sup>st</sup> section
- Just dump .exe, load it in IDA, and apply MSVC signatures, then search for OEP
- Use PAGE\_GUARD to find when code section jmp to oep

Well there are several ways as you may see, but it is upto you to find method which fits your needs the best.

OEP in this target is located at: 0x40A1B3

```
.text:0040A1B3      call     ___security_init_cookie
.text:0040A1B8      jmp      ___tmainCRTStartup

...

.text:00409EF2  ___tmainCRTStartup:
.text:00409EF2      push    58h
.text:00409EF4      push    offset unk_B95678
.text:00409EF9      call    sub_40A250
.text:00409EFE      xor     ebx, ebx
.text:00409F00      mov     [ebp-1Ch], ebx
.text:00409F03      mov     [ebp-4], ebx
.text:00409F06      lea     eax, [ebp-68h]
.text:00409F09      push    eax
.text:00409F0A      call    ds:GetStartupInfoA
.text:00409F10      mov     dword ptr [ebp-4], 0FFFFFFFFh
.text:00409F17      mov     dword ptr [ebp-4], 1
.text:00409F1E      mov     eax, large fs:18h
.text:00409F24      mov     esi, [eax+4]
.text:00409F27      mov     edi, offset unk_C5DD54
```

When dumping SecuROM you have to know that it's PE header in memory is actually PE header of a virgin file (except AddressOfEntryPoint is foobared and used as anti-dump on several places).

As we know this fact we may write dumper for SecuROM and dump virgin file to the disk. You may see sromd.asm for detailed code (nothing special just read PE header from memory and dump image to disk + add extra section for SecuROM sections).

SecuROM uses jmp to execute some stolen procedures which are mixed with SecuROM code. To find such procedure you won't have to search much, first call in OEP leads us to SecuROM code:

0040A250	-FF25 ECB03C01	JMP DWORD PTR DS:[13CB0EC]
0040A256	CC	INT3
0040A257	CC	INT3
0040A258	CC	INT3
0040A259	CC	INT3
0040A25A	CC	INT3
0040A25B	CC	INT3
0040A25C	CC	INT3

This jmp only first time will take you to SecuROM virtual buffers:

SF450000	✓EB 1D	JMP SHORT SF45001F
SF450002	6E	OUTS DX, BYTE PTR ES:[EDI]
SF450003	6F	OUTS DX, DWORD PTR ES:[EDI]
SF450004	626F 64	BOUND EBP, QWORD PTR DS:[EDI+64]
SF450007	✓79 20	JNS SHORT SF450029
SF450009	6D	INS DWORD PTR ES:[EDI], DX
SF45000A	6F	OUTS DX, DWORD PTR ES:[EDI]
SF45000B	✓76 65	JBE SHORT SF450072
SF45000D	2C 20	SUB AL, 20
SF45000F	6E	OUTS DX, BYTE PTR ES:[EDI]
SF450010	6F	OUTS DX, DWORD PTR ES:[EDI]
SF450011	626F 64	BOUND EBP, QWORD PTR DS:[EDI+64]
SF450014	✓79 20	JNS SHORT SF450036
SF450016	67:65:74 73	JE SHORT SF45008D
SF45001A	2068 75	AND BYTE PTR DS:[EAX+75], CH
SF45001D	✓72 74	JB SHORT SF450093
SF45001F	60	PUSHAD
SF450020	9C	PUSHFD
SF450021	E8 00000000	CALL SF450026
SF450026	E8 02000000	CALL SF45002D
SF45002B	0101	ADD DWORD PTR DS:[ECX], EAX
SF45002D	5A	POP EDX
SF45002E	F0:FE0A	LOCK DEC BYTE PTR DS:[EDX]
SF450031	✓79 30	JNS SHORT SF450063
SF450033	803A 00	CMP BYTE PTR DS:[EDX], 0
SF450036	53	REPSTB REP.

If you keep tracing trough it you will eventually end up here:

025B2F88	81C3 24000000	ADD EBX, 24
025B2F8E	C703 00000000	MOV DWORD PTR DS:[EBX], 0
025B2F94	9D	POPF
025B2F95	61	POPAD
025B2F96	68 00000000	PUSH 0
025B2F9B	68 00000000	PUSH 0
025B2FA0	68 00000000	PUSH 0
025B2FA5	68 00000000	PUSH 0
025B2FAA	68 00000000	PUSH 0
025B2FAF	68 00000000	PUSH 0
025B2FB4	68 00000000	PUSH 0
025B2FB9	68 00000000	PUSH 0
025B2FBE	68 00000000	PUSH 0
025B2FC3	68 00000000	PUSH 0
025B2FC8	68 00000000	PUSH 0
025B2FCD	68 00000000	PUSH 0
025B2FD2	8D424 30	LEA ESP, DWORD PTR SS:[ESP+30]
025B2FD6	C2 0400	RETN 4

Now look, it will take you to stolen and mixed procedure here:

0044F250	68 A9A24000	PUSH cnc3.0040A2A9
0044F255	64:FF35 00000001	PUSH DWORD PTR FS:[0]
0044F25C	8B4424 10	MOV EAX,DWORD PTR SS:[ESP+10]
0044F260	896C24 10	MOV DWORD PTR SS:[ESP+10],EBP
0044F264	802D C3FD4301	LEA EBP,DWORD PTR DS:[143FDC3]
0044F26A	81ED DC490000	SUB EBP,49DC
0044F270	FF4D 00	DEC DWORD PTR SS:[EBP]
0044F273	-0F84 1CDEFE00	JE cnc3.0143D095
0044F279	8D6C24 10	LEA EBP,DWORD PTR SS:[ESP+10]
0044F27D	2BE0	SUB ESP,EAX
0044F27F	53	PUSH EBX
0044F280	56	PUSH ESI
0044F281	57	PUSH EDI
0044F282	FF0D ECB34301	DEC DWORD PTR DS:[143B8EC]
0044F288	0F84 97FD0100	JE cnc3.0046F025
0044F28E	A1 9005BA00	MOV EAX,DWORD PTR DS:[BA0590]
0044F293	3145 FC	XOR DWORD PTR SS:[EBP-4],EAX
0044F296	33C5	XOR EAX,EBP
0044F298	50	PUSH EAX
0044F299	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
0044F29C	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
0044F29F	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
0044F2A2	C745 FC FFFFFFFF	MOV DWORD PTR SS:[EBP-4],-2
0044F2A9	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
0044F2AC	8D45 F0	LEA EAX,DWORD PTR SS:[EBP-10]
0044F2AF	64:A3 00000000	MOV DWORD PTR FS:[0],EAX
0044F2B5	C3	RET
0044F2B6	CC	INT3
0044F2B7	CC	INT3

You may see how this procedure is mixed with addresses from .securom section, and some jccs are leading to that section. If we take one step back and look again at the jmp from which we have started to trace, we may see that SecuROM wrote to jmp dword ptr[] correct address:

0040A250	FF25 ECB03C01	JMP DWORD PTR DS:[13CB0EC]	cnc3.0044F250
0040A256	CC	INT3	
0040A257	CC	INT3	
0040A258	CC	INT3	
0040A259	CC	INT3	

Note address in 2<sup>nd</sup> column, it is same as address of stolen procedure. There are at least 2 reasons why this is done in such way:

1. Numerous execution of same procedure would slow down game execution
2. Make code anti-dump as you may not simply dump it without fixing those jmps

Those jmp dword ptr[] can be easily found by using byte search, thus, you will either have to write your own tool, or use olly scripts to do the job for you. Before we even dump file, we have to fix those jmps, and for that I use srom\_logger.exe and vmtrace.dll.

Srom\_logger.exe is very simple search engine which searches for jmp dword ptr[] and checks if they are leading us to .securom section. If such jmp is found, loader is injected into remote process which will be responsible for loading vmtrace.dll and calling it's export vmtrace!tracer.

Let's analyze those codes a little bit so you may know how and why I'm doing what:

```

loader:
    call    __delta
__delta:
    pop     ebp
    sub     ebp, offset __delta

    x_push  ecx, <vmtrace.dll~>
    call    [ebp+loadlibrarya], esp
    x_pop

    x_push  ecx, <tracer~>
    call    [ebp+getprocaddress], eax, esp
    x_pop

    push    0deadc0deh
    =       $-4
    call    eax
    mov     [ebp+redirection], eax
    call    [ebp+exitthread], 0

loadlibrarya    dd    ?
getprocaddress  dd    ?
exitthread      dd    ?
redirection     dd    ?
size_loader     =     $-loader

```

tracer export from vmtrace.dll is very simple and it's job is to set HWBP on write at address from jmp dword ptr[]:

```

tracer
    proc
    arg    trace_address
    pusha
    mov     eax, trace_address
    mov     global_trace_address, eax

    xor     eax, eax
    push    offset setdr0
    push    dword ptr fs:[eax]
    mov     dword ptr fs:[eax], esp
    mov     eax, [eax]
    pop     dword ptr fs:[eax]
    add     esp, 4

    call    hook_kiuser
    mov     save_esp, esp
    mov     eax, trace_address
    jmp     [eax]

__baby:
    call    unhook_kiuser
    mov     eax, destination

    mov     [esp.Pushad_eax], eax
    popa
    leave
    retn    4

```

Also you may see in vmtrace.asm code responsible for hooking KiUserExceptionDispatcher, which will perform stealth tracing from context of our target.

Now simply execute that address and we are going to nonintrusive tracer:

```

kiuser_hook:      mov     ecx, [esp+4]
                  mov     ebx, [esp]
                  pusha
                  cmp     dword ptr[ebx], EXCEPTION_SINGLE_STEP
                  je      __checkdrx

                  xor     eax, eax
                  mov     [ecx.context_dr6], eax
                  mov     [ecx.context_dr0], eax
                  mov     [ecx.context_dr1], eax
                  mov     destination, eax
                  mov     [ecx.context_eip], offset __baby
                  mov     eax, save_esp
                  mov     [ecx.context_esp], eax
                  jmp     __allgood

__checkdrx:       test    [ecx.context_dr6], 4000h           ;single stepping...
                  jnz     __goback
                  test    [ecx.context_dr6], 1
                  jnz     __write
                  test    [ecx.context_dr6], 2
                  jz      __goback                           ;dr1 hit...
                  xor     eax, eax
                  mov     [ecx.context_dr6], eax
                  mov     [ecx.context_dr0], eax
                  mov     [ecx.context_dr1], eax
                  mov     eax, save_esp
                  mov     [ecx.context_esp], eax
                  mov     eax, offset __baby
                  xchg     [ecx.context_eip], eax
                  mov     destination, eax
                  jmp     __allgood

                  ;dr0 hit...
__write:          mov     esi, ecx
                  call     ReadProcessMemory, -1, global_trace_address, o
destination, 4, 0
                  call     ReadProcessMemory, -1, destination, o old_data, 4, 0
                  test    eax, eax
                  jz      __goback

                  mov     eax, destination
                  mov     [esi.context_dr1], eax
                  or      [esi.context_dr7], 4

__allgood:        popa
                  call     NtContinue, ecx, 1
                  nop
                  nop

__goback:         popa
                  jmp     old_kiuser

```



You would probably wonder why I am using ReadProcessMemory to read address from my own process. Well trick is simple, as I used HWBP on r/w each read from r3 would cause HWBP to generate exception, even when you are reading from exception handler, now as code is rewritten to use HWBP on write this is not needed anymore. Also you may see that I'm setting HWBP on execution on final destination. This is done to avoid wrong identification of destination when SecuROM writes 2 times to this address. Whenever write there occurs we take that address and HWBP on execution on that address. When dr1 is hit (execution HWBP) we know we have good pointer, and we log it.

Also it will occur that procedures are in virtual memory so we dump that too to the disk which will allow us to fix them easily. Also srom\_logger.asm will produce log file which will look something like this:

```
redirection at 0x00401023 to 0x013D34C3
redirection at 0x00401098 to 0x013F11F8
redirection at 0x004010C5 to 0x013F66A5
redirection at 0x00401103 to 0x00434D33
redirection at 0x00401141 to 0x0044D2F1
redirection at 0x0040116F to 0x013EA3CF
redirection at 0x004011AD to 0x013F2BBD
```

There is 2957 those jmps so we need to automate process of fixing. Also regions with procedures are dumped to the disk for later fixing (again in another tool named sfixer.asm). Note that after you run srom\_logger.asm you will have fixed jmp dword ptr[] in your target in memory so this is the point when you dump it to the disk with fixed jmps.

Here is an example of one stolen procedure stored somewhere in virtual memory:

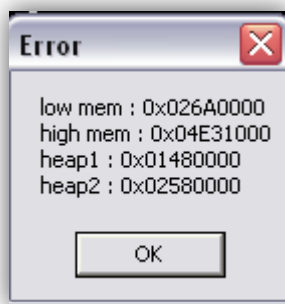
```
seg000:03A5000B      push     esi
seg000:03A5000C      mov      esi, [esp+8]
seg000:03A50010      jmp      short loc_3A5002A
seg000:03A50012
seg000:03A50012 loc_3A50012:
seg000:03A50012      mov      ecx, esi
seg000:03A50014      push     offset loc_3A50027
seg000:03A50019      push     7FE20Fh
seg000:03A5001E      retn
seg000:03A5001F      dd      0DF8A7B4Ah
seg000:03A50023      dd      0E5B9FAh
seg000:03A50027
seg000:03A50027 loc_3A50027:
seg000:03A50027      add      esi, 10h
seg000:03A5002A
seg000:03A5002A loc_3A5002A:
seg000:03A5002A      cmp      esi, [esp+0Ch]
seg000:03A5002E      jnz      short loc_3A50012
seg000:03A50030      pop      esi
seg000:03A50031      retn
```

You can't put it back to its original place as SecuROM uses that space in code section for other anti-dump tricks.

We are almost close to dumping this target ☺

Next thing we should know is where are all virtual buffers that we have to dump and append. For this purpose I use hookdll.c which can be used with my Ultimate Hooking Engine [1]. Also it is recommended to use hook of rdtsc to avoid randomness in memory allocation.

Oki, inject hookdll.dll into target by typing: hook cnc3game.dat, and after a few seconds you will be greeted with MessageBoxA similar to this one on the picture:



Write down those memory addresses, and press ok. Your target will be in the infinite loop in hook of GetTickCount. Do not attach Olly yet!!! Run srom\_logger.exe and you will get output similar to this one :

```

C:\WINDOWS\system32\CMD.exe - srom_logger.exe
redirection at 0x0040DC89 to 0x0040BBA1
locating 0x0040E025
redirection at 0x0040E025 to 0x04B50005
locating 0x0040E226
redirection at 0x0040E226 to 0x0041CD46
locating 0x0040E2C0
redirection at 0x0040E2C0 to 0x00492590
locating 0x0040E2E8
redirection at 0x0040E2E8 to 0x00449C48
locating 0x0040E523
redirection at 0x0040E523 to 0x00467C03
locating 0x0040E572
redirection at 0x0040E572 to 0x00499682
locating 0x0040E643
redirection at 0x0040E643 to 0x013E4E43
locating 0x0040E664
redirection at 0x0040E664 to 0x013E80B4
locating 0x0040E67F
redirection at 0x0040E67F to 0x0045862F
locating 0x0040E77E
redirection at 0x0040E77E to 0x0048B5FE
locating 0x0040E7AF
redirection at 0x0040E7AF to 0x013E25AF
locating 0x0040E84E

```

Oki doki, we have produced 0XXXXXXXXX.dmp files and splices.bin + we have fixed jmp dword ptr[] in protected program. Now you may attach olly, or simply ctrl+d if you are using SoftICE and you will be here:

021514EC	-EB FE	JMP SHORT hookdll.021514EC
021514EE	FF15 80A11502	CALL DWORD PTR DS:[Detoured_GetTickCount]
021514F4	8BE5	MOV ESP,EBP
021514F6	5D	POP EBP
021514F7	C3	RETN
021514F8	55	PUSH EBP
021514F9	8BEC	MOV EBP,ESP
021514FB	53	PUSH EBX
021514FC	8B5D 0C	MOV EBX,DWORD PTR SS:[EBP+C]

What I have done here is to check for return address in GetTickCount, and if it is called from certain location I put target into infinite loop, this giving me possibility to always break at OEP fast. You should nop out this jmp \$ but do NOT run target yet, as you will need to get also TlsValue with index 0xF which is used as anti-dump at one point. Simply assemble this code:

021514EC	6A 0F	PUSH 0F
021514EE	E8 6D826B7A	CALL kernel32.TlsGetValue
021514F3	90	NOP
021514F4	90	NOP
021514F5	90	NOP
021514F6	5D	POP EBP

The reason why I'm doing this at this moment is to save you some time, you could easily figure this thing later on, and then you will have to dump all over again ☺ Not letting you know at this point about this trick would be mean ☺ **Also you have to write down PID of this process!!!!**

Now, you may see that in PE header there is valid virgin header, so you may dump it like that. I use my dumper for SecuROM which dumps that pe header from memory and image as it was before packing, and appends to it other memory occupied by SecuROM.

After running sromd.asm you will have dumped\_securom.exe which will look like this:

```

dumped_securom> JPRO ----- PE.00400000 Hiew 7.26 <c>SEN
.00400000: 4D 5A 90 00-03 00 00 00-04 00 00 00-FF FF 00 00  MZ  ♥
.00400010: B8 00 00 00-00 00 00 00-40 00 00 00-00 00 00 00  7
.00400020: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00

Number Name      VirtSize      RVA      PhysSize  Offset      Flag
-----
1 .text      00636C23      00001000      00637000      00001000      60000020
2 .rdata     001672E9      00638000      00168000      00638000      40000040
3 .data      000BDDC0      007A0000      00069000      007A0000      C0000040
4 rts.ver    00000200      0085E000      00001000      00809000      40000040
5 .tls       00000009      0085F000      00001000      0080A000      C0000040
6 .rsrc      00019CA4      00860000      0001A000      0080B000      40000040
7 .bla       00801000      0087A000      00801000      00825000      E0000020

Cursor's out of sections

.00400150: B3 A1 00 00-00 10 00 00-00 80 63 00-00 00 40 00  i  ▶  ⚡  c  e
.00400160: 00 10 00 00-00 10 00 00-04 00 00 00-00 00 00 00  ▶  ▶  ⚡  c  e
1 2 3 4 5 6 7 8 9 10

```

So far so good. You wrote down addresses displayed in MessageBoxA? If not go all over again.

Now take a closer look at spliced files dumped by srom\_logger.exe:

Splices start:

0x037F0000	dmp	4,096
0x03800000	dmp	4,096
0x03810000	dmp	4,096
0x03820000	dmp	4,096

Splices end:

0x04DF0000	dmp	4,096
0x04E00000	dmp	4,096
0x04E10000	dmp	4,096
0x04E20000	dmp	4,096

By looking at splices we may see that those are allocated on 0x10000 boundary, but there is one small gap between all those splices located here:

0x045A0000	dmp	4,096
0x045B0000	dmp	4,096
0x045C0000	dmp	4,096
0x045D0000	dmp	4,096
0x04960000	dmp	4,096
0x04970000	dmp	4,096
0x04980000	dmp	4,096
0x04990000	dmp	4,096

Look closer and you will see that we are missing memory region between 45D0000 and 4960000. So we will have to dump that region too.

So far we have 2 regions which have to be dumped:

1480000 – 37F0000 and 45D0000 – 4960000, dump them, and use **CFF explorer by Daniel Pistelli** to add those regions to dumped\_securom.exe. For me it looks like this:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenum
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00637000	00001000	00637000	00001000	00000000	00000000
.rdata	00168000	00638000	00168000	00638000	00000000	00000000
.data	000BE000	007A0000	00069000	007A0000	00000000	00000000
.rsrc	00001000	0085E000	00001000	00809000	00000000	00000000
.tls	00001000	0085F000	00001000	0080A000	00000000	00000000
.rsrc	0001A000	00860000	0001A000	0080B000	00000000	00000000
.bld	00806000	0087A000	00801000	00825000	00000000	00000000
.arteam	03150000	01080000	02370000	01026000	00000000	00000000
.arteam2	00390000	041D0000	00390000	03396000	00000000	00000000

One more thing left to go over, and that's to add splices, with sfixer.exe . Before you run sfixer.exe you should save splices.bin to, for example, splices\_save.bin as sfixer.exe will modify this file, and if something goes wrong you won't have this file, also you should know that sfixer.exe assumes that you are fixing file "final.exe" so copy updated dumped\_securom.exe to "final.exe". Note also that this step is NOT required,

you could just dump memory from heap1 to highmem from MessageBoxA but your dump would be + ~20mb. In this way, with splices fixing we are reducing dump size + we are making nicer dump.

After running sfixer.asm we may check some of our virtual memory redirections from log\_redirection.exe (produced by srom\_logger.asm):

redirection at 0x00410BC1 to 0x03B20001 <--- from log\_redirections.txt

00410BC1	-FF25 94A43C01	JMP DWORD PTR DS:[13CA494]	final.04960168
00410BC7	9C	PUSHFD	
00410BC8	A1 748A3C01	MOV EAX,DWORD PTR DS:[13C8A74]	
00410BCD	F7D0	NOT EAX	
00410BCF	3305 7FC14300	XOR EAX,DWORD PTR DS:[43C17F]	
00410BD5	8B00	MOV EAX,DWORD PTR DS:[EAX]	

And fixed splice:

04960168	✓EB 29	JMP SHORT final.04960193
0496016A	8B06	MOV EAX,DWORD PTR DS:[ESI]
0496016C	0FB000	MOVSX EAX,BYTE PTR DS:[EAX]
0496016F	50	PUSH EAX
04960170	68 8C019604	PUSH final.0496018C
04960175	FF35 2886A300	PUSH DWORD PTR DS:[A38628]
0496017B	C3	RETN
0496017C	290A	SUB DWORD PTR DS:[EDX],ECX
0496017E	04 6F	ADD AL,6F
04960180	E8 D23CEA9D	CALL A2803E57
04960185	F0:1278 BD	LOCK ADC BH,BYTE PTR DS:[EAX-43]
04960189	37	AAA
0496018A	✓E0 20	LOOPNE SHORT final.049601AC
0496018C	85C0	TEST EAX,EAX
0496018E	59	POP ECX
0496018F	✓74 09	JE SHORT final.0496019A
04960191	FF06	INC DWORD PTR DS:[ESI]
04960193	8B06	MOV EAX,DWORD PTR DS:[ESI]
04960195	8038 00	CMP BYTE PTR DS:[EAX],0
04960198	^75 D0	JNZ SHORT final.0496016A
0496019A	C3	RETN

I'll also show you two splices which weren't fixed by my tool correctly, and which I have fixed by hand (from protected game in memory):

039B0032	57	PUSH EDI
039B0033	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-8]
039B0036	50	PUSH EAX
039B0037	8D8E 48020000	LEA ECX,DWORD PTR DS:[ESI+248]
039B003D	68 4C009B03	PUSH 39B004C
039B0042	68 4C009B03	PUSH 39B004C
039B0047	C3	RETN
039B0048	0029	ADD BYTE PTR DS:[ECX],CH

From my dump:

049643ED	57	PUSH EDI
049643EE	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-8]
049643F1	50	PUSH EAX
049643F2	8D8E 48020000	LEA ECX,DWORD PTR DS:[ESI+248]
049643F8	68 07449604	PUSH final.04964407
049643FD	68 4C009B03	PUSH final.039B004C
04964402	C3	RETN

And when fixed by hand it should look like:

049643ED	57	PUSH EDI
049643EE	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-8]
049643F1	50	PUSH EAX
049643F2	8D8E 48020000	LEA ECX,DWORD PTR DS:[ESI+248]
049643F8	68 07449604	PUSH final.04964407
049643FD	68 07449604	PUSH final.04964407
04964402	C3	RETN

And second splice:

03F301CE	9C	PUSHFD
03F301CF	90	NOP
03F301D0	68 79DC3B58	PUSH 583BDC79
03F301D5	68 164F9B76	PUSH 769B4F16
03F301DA	54	PUSH ESP
03F301DB	68 A9563C01	PUSH cracked.013C56A9
03F301E0	C1F8 00	SAR EAX,0
03F301E3	E8 487BD5FC	CALL cracked.00C87D30
03F301E8	57	PUSH EDI

And when fixed:

049672D1	9C	PUSHFD
049672D2	90	NOP
049672D3	68 79DC3B58	PUSH 583BDC79
049672D8	68 164F9B76	PUSH 769B4F16
049672DD	54	PUSH ESP
049672DE	68 A9563C01	PUSH final.013C56A9
049672E3	C1F8 00	SAR EAX,0
049672E6	E8 450A32FC	CALL final.00C87D30
049672EB	57	PUSH EDI

As you may see in this picture, splice is fixed, but before fixing this call was causing crash as whole splice was rebased to the new address.

Theoretically speaking sfixer.asm can be rewritten to follow execution flow and rebuild those procedures in better way. We may see 3 patterns used as call:

```

push    <ret_splice_address>
push    <proc_address>
ret

push    <ret_splice_address>
jmp     __proc_address

push    <ret_splice_address>
push    dword ptr[API_pointer_from_IAT]
ret

```

And those are only patterns which are fixed by sfixer.asm, of course, better engine could be written, but, this is good enough ☺

Voila, dumping is done now. All we have to do is to get rid of anti-dump tricks present in the SecuROM, and we will have dumped and fixed game ☺



## 5. Anti-Dump fixing

This is part where your head might start hurting a little bit, so my advice is to grab pen and paper and write your observation. Well at least that's how I do with all protectors.

First anti-dump that you will see is when you enter into SecuROM vm interpreter. I don't want to trouble you much so here is address to be hit first:

```
.bla:00CBEC00 sub_CBEC00      proc near
.bla:00CBEC00      jmp      ds:dword_125D9EC
.bla:00CBEC00 sub_CBEC00      endp
```

Depending on your dump address to which this jmp is leading could be different, but for me it is at 3320000:

```
seg000:03320345      mov     eax, large fs:18h
seg000:0332034B      inc     ebp
seg000:0332034C      db      3Eh
seg000:0332034C      mov     eax, [eax+30h]
seg000:03320350      db      3Eh
seg000:03320350      mov     eax, [eax+8]
seg000:03320354      db      3Eh
seg000:03320354      mov     edx, [eax+3Ch]
seg000:03320358      dec     ebp
seg000:0332035A      db      3Eh
seg000:0332035A      mov     edx, [edx+eax+50h]
```

Here it takes OptionalHeader.SizeOfImage if you dump your target it will have different image size from the one that SecuROM expects to be there. So you will have to fix it, it is, 87A000, well simply dump target from memory with LordPE and you will see correct values.

Ok you need to patch it... simply patch mov edx, [edx+eax+50h] with mod edx, 87A000, and first anti-dump is defeated. Next anti dump is CPUID trick ☺ which makes dump only CPU specific eg. It won't work on other CPUs if you don't fix it:

```
seg000:03320305      mov     eax, 1
seg000:0332030A      push    ebx
seg000:0332030B      add     ebp, eax
seg000:0332030D      cpuid
seg000:0332030F      and     eax, 0FFFFFFDFh
```

This is relatively simple to fix, what you will do is to search trough dumped vm regions for certain byte patern: mov eax, 1. When you find it, now use lde to check if cpuid is present in next 5 instructions, if you find jmp \_\_ follow it. When you find cpuid you know what to fix. Use again lde from cpuid offset to find and eax, 0FFFFFFDFh and patch it with mov eax, value which is returned on your CPU.

This search'n'replace is easy to write, so you may exercise a little bit ☺

There are 2 more anti-dumps in VM interpreter. GetCurrentProcessId and OpenEventA/CloseHandle. Trick here is to patch GetCurrentProcessId call to return PID of your dump, and OpenEventA/CloseHandle to return 1 for CloseHandle:

```

seg000:028400F1      mov     edx, [edx]
seg000:028400F3      xor     edx, 0CC5C4764h
seg000:028400F9      call    edx <-- GetCurrentProcessId
seg000:028400FB      and     ebp, edx
seg000:028400FD      mov     edx, eax
seg000:028400FF      sub     edx, ecx
seg000:02840101      btc     ebp, eax
seg000:02840104      pop     ecx

```

So it should look something like this when patched:

028400F1	8B12	MOV EDX,DWORD PTR DS:[EDX]
028400F3	B8 9C0C0000	MOV EAX,0C9C
028400F8	90	NOP
028400F9	90	NOP
028400FA	90	NOP

OpenEventA/CloseHandle:

```

seg000:028600DB      mov     edx, [edx]
seg000:028600DD      xor     edx, 0BA6258DBh
seg000:028600E3      call    sub_286010A
seg000:028600E8      a01a3ee67056205 db '01A3EE67056205C94339FAF75B158E1CD',0
seg000:0286010A      sub_286010A      proc near
seg000:0286010A
seg000:0286010A      btc     ebp, eax
seg000:0286010D      push    0
seg000:02860112      mov     ebp, 8A9FAD20h
seg000:02860117      push    2
seg000:0286011C      call    edx <--- OpenEvent
seg000:0286011E      inc     ebp
seg000:02860120      push    eax
seg000:02860121      xadd    ebp, ebp
seg000:02860124      mov     edx, [ebx+28h]
seg000:02860127      add     edx, 14h
seg000:0286012D      xchg    ebp, ebp
seg000:0286012F      mov     edx, [edx]
seg000:02860131      xor     edx, 0AC86A5A7h
seg000:02860137      xor     ebp, 0C9C0E01Dh
seg000:0286013D      call    edx <--- CloseHandle

```

You should patch this part in a smart way, first you have to patch push 2/call edx with add esp, 8 to eliminate 2<sup>nd</sup> and 3<sup>rd</sup> arguments passed to OpenEvent, now you will have to patch push eax (event handle with nop) and xor edx, 0AC86A5A7h with mov eax,1 and nop call edx (CloseHandle).

Good, good, VM handlers are now anti-dump patched ☺

Let's proceed to another anti-dump in code of SecuROM:

00F9CD4D	55	PUSH EBP
00F9CD4E	8BEC	MOV EBP,ESP
00F9CD50	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
00F9CD53	8BC8	MOV ECX,EAX
00F9CD55	C1E9 04	SHR ECX,4
00F9CD58	83E1 07	AND ECX,7
00F9CD5B	83F9 07	CMP ECX,7
00F9CD5E	0F87 A8000000	JAE final.00F9CE0C
00F9CD64	FF248D 11CEF900	JMP DWORD PTR DS:[ECX*4+F9CE11]
00F9CD6B	50	PUSH EAX
00F9CD6C	FF15 B8FD2E01	CALL DWORD PTR DS:[12EFD88]
00F9CD72	35 0067FA00	XOR EAX,0FA6700
00F9CD77	3305 B8FD2E01	XOR EAX,DWORD PTR DS:[12EFD88]
00F9CD7D	E9 8A000000	JMP final.00F9CE0C
00F9CD82	50	PUSH EAX
00F9CD83	FF15 BCFD2E01	CALL DWORD PTR DS:[12EFD8C]
00F9CD89	35 1067FA00	XOR EAX,0FA6710
00F9CD8E	3305 BCFD2E01	XOR EAX,DWORD PTR DS:[12EFD8C]
00F9CD94	EB 76	JMP SHORT final.00F9CE0C
00F9CD96	50	PUSH EAX
00F9CD97	FF15 C0FD2E01	CALL DWORD PTR DS:[12EFD80]
00F9CD9D	35 2067FA00	XOR EAX,0FA6720
00F9CDA2	3305 C0FD2E01	XOR EAX,DWORD PTR DS:[12EFD80]
00F9CDA8	EB 62	JMP SHORT final.00F9CE0C
00F9CDA9	50	PUSH EAX
00F9CDAE	FF15 C4FD2E01	CALL DWORD PTR DS:[12EFD84]
00F9CDB1	35 5067FA00	XOR EAX,0FA6750
00F9CDB6	3305 C4FD2E01	XOR EAX,DWORD PTR DS:[12EFD84]
00F9CDBC	EB 4E	JMP SHORT final.00F9CE0C
00F9CDBE	50	PUSH EAX
00F9CDBF	FF15 C8FD2E01	CALL DWORD PTR DS:[12EFD88]
00F9CDC5	35 7067FA00	XOR EAX,0FA6770
00F9CDDA	3305 C8FD2E01	XOR EAX,DWORD PTR DS:[12EFD88]
00F9CDD8	EB 3A	JMP SHORT final.00F9CE0C
00F9CDD2	50	PUSH EAX
00F9CDD3	FF15 CCFD2E01	CALL DWORD PTR DS:[12EFD8C]
00F9CDD9	35 D067FA00	XOR EAX,0FA67D0
00F9CDDC	3305 CCFD2E01	XOR EAX,DWORD PTR DS:[12EFD8C]
00F9CDE4	EB 26	JMP SHORT final.00F9CE0C
00F9CDE6	50	PUSH EAX
00F9CDE7	FF15 D0FD2E01	CALL DWORD PTR DS:[12EFD80]
00F9CDED	35 E067FA00	XOR EAX,0FA67E0
00F9CDF2	3305 D0FD2E01	XOR EAX,DWORD PTR DS:[12EFD80]
00F9CDF8	EB 12	JMP SHORT final.00F9CE0C
00F9CDFA	50	PUSH EAX
00F9CDFB	FF15 D4FD2E01	CALL DWORD PTR DS:[12EFD84]
00F9CE01	35 5068FA00	XOR EAX,0FA6850
00F9CE06	3305 D4FD2E01	XOR EAX,DWORD PTR DS:[12EFD84]
00F9CE0C	3345 0C	XOR EAX,DWORD PTR SS:[EBP+C]
00F9CE0F	5D	POP EBP
00F9CE10	C3	RET

Hmmm what is what here? Lets start by going to each one of them and seeing what is going on:

1<sup>st</sup>:

00FA6700	FF15 185A1F01	CALL DWORD PTR DS:[11F5A18]
00FA6706	034424 04	ADD EAX,DWORD PTR SS:[ESP+4]
00FA670A	C2 0400	RET 4
00FA670D	CC	INT3
00FA670E	CC	INT3
00FA670F	CC	INT3

This is GetCurrentProcessId check, which you will have to patch with **mov eax, dump\_pid**

2<sup>nd</sup>:

00FA6710	FF15 1C5A1F01	CALL DWORD PTR DS:[11F5A1C]	kernel32.GetVersion
00FA6716	2B4424 04	SUB EAX,DWORD PTR SS:[ESP+4]	
00FA671A	C2 0400	RET 4	
00FA671D	CC	INT3	
00FA671E	CC	INT3	
00FA671F	CC	INT3	

Check for OS version via GetVersion, so you patch it with **mov eax, version\_of\_your\_os**

3<sup>rd</sup> check:

00FA6720	8D6424 FC	LEA ESP,DWORD PTR SS:[ESP-4]
00FA6724	890C24	MOV DWORD PTR SS:[ESP],ECX
00FA6727	8D6424 FC	LEA ESP,DWORD PTR SS:[ESP-4]
00FA672B	891C24	MOV DWORD PTR SS:[ESP],EBX
00FA672E	B8 01000000	MOV EAX,1
00FA6733	0FA2	CPUID
00FA6735	83E0 DF	AND EAX,FFFFFFDF
00FA6738	334424 0C	XOR EAX,DWORD PTR SS:[ESP+C]
00FA673C	894424 04	MOV DWORD PTR SS:[ESP+4],EAX
00FA6740	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]
00FA6744	5B	POP EBX
00FA6745	59	POP ECX
00FA6746	C2 0400	RETN 4
00FA6749	CC	INT3
00FA674A	CC	INT3
00FA674B	CC	INT3

Well pretty much self explanatory, patch it with value which cpuid returns on your machine

4<sup>th</sup> check:

00FA6750	FF35 245A1F01	PUSH DWORD PTR DS:[11F5A24]	
00FA6756	FF15 205A1F01	CALL DWORD PTR DS:[11F5A20]	kernel32.ResetEvent
00FA675C	F7D8	NEG EAX	
00FA675E	1BC0	SBB EAX,EAX	
00FA6760	2305 245A1F01	AND EAX,DWORD PTR DS:[11F5A24]	
00FA6766	C2 0400	RETN 4	
00FA6769	CC	INT3	
00FA676A	CC	INT3	
00FA676B	CC	INT3	

Patch this as **mov eax, 1**, and **nop** out call to **ResetEvent**

5<sup>th</sup> check:

00FA6770	8D6424 FC	LEA ESP,DWORD PTR SS:[ESP-4]	
00FA6774	893424	MOV DWORD PTR SS:[ESP],ESI	
00FA6777	BE 5C5A1F01	MOV ESI,cnc3.011F5A5C	
00FA677C	56	PUSH ESI	
00FA677D	FF15 745A1F01	CALL DWORD PTR DS:[11F5A74]	ntdll.RtlEnterCriticalSection
00FA6783	A1 B0FD2E01	MOV EAX,DWORD PTR DS:[12EFD80]	
00FA6788	83E0 7F	AND EAX,7F	
00FA678B	3C 42	CMP AL,42	
00FA678D	75 1E	JNZ SHORT cnc3.00FA67AD	
00FA678F	68 2C5A1F01	PUSH cnc3.011F5A2C	
00FA6794	68 4C5A1F01	PUSH cnc3.011F5A4C	ASCII "39BF05KBJE4A"
00FA6799	C705 2C5A1F01	MOV DWORD PTR DS:[11F5A2C],10	
00FA67A3	FF15 485A1F01	CALL DWORD PTR DS:[11F5A48]	kernel32.GetComputerNameA
00FA67A9	85C0	TEST EAX,EAX	
00FA67AB	74 1F	JE SHORT cnc3.00FA67CC	
00FA67AD	FF05 B0FD2E01	INC DWORD PTR DS:[12EFD80]	
00FA67B3	56	PUSH ESI	
00FA67B4	FF15 785A1F01	CALL DWORD PTR DS:[11F5A78]	ntdll.RtlLeaveCriticalSection
00FA67BA	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
00FA67BE	8B0D 2C5A1F01	MOV ECX,DWORD PTR DS:[11F5A2C]	
00FA67C4	03C1	ADD EAX,ECX	
00FA67C6	3305 485A1F01	XOR EAX,DWORD PTR DS:[11F5A48]	kernel32.GetComputerNameA
00FA67CC	5E	POP ESI	
00FA67CD	C2 0400	RETN 4	
00FA67D0	FF15 005A1F01	CALL DWORD PTR DS:[11F5A00]	
00FA67D6	334424 04	XOR EAX,DWORD PTR SS:[ESP+4]	
00FA67DA	C2 0400	RETN 4	
00FA67DB	CC	INT3	

Hey, there is my computer name, nice ☺ So SecuROM here checks for length of your computer name. Check line 0xFA678F and 2 lines after it is set to 0x10, which is maximum size of buffer passed to GetComputerNameA. Then at line 0xFA67BE it takes length of computer name returned by call to GetComputerName, so your patch here would be to nop out EnterCriticalSection and LeaveCriticalSection, also nop out call to GetComputerNameA and nop passing arguments to it, and assemble there **mov [12F5A2C], len\_of\_your\_computer\_name** , and nop je at 0xFA67AB. Easy...

6<sup>th</sup> check:

```

00FA67D0 FF15 005A1F01 CALL DWORD PTR DS:[11F5A00]
00FA67D6 334424 04 XOR EAX,DWORD PTR SS:[ESP+4]
00FA67DA C2 0400 RETN 4
00FA67DD CC INT3
00FA67DE CC INT3
00FA67DF CC INT3

```

And buffer:

```

04650000 E8 00000000 CALL 04650005
04650005 58 POP EAX
04650006 05 5E4CFA00 ADD EAX,0FA4C5E
04650008 2D 634CFA00 SUB EAX,0FA4C63
04650010 C3 RETN
04650011 0000 ADD BYTE PTR DS:[EAX],AL

```

This buffer will simply return it's address, so above check you should patch with:

**mov eax, 4650000**

7<sup>th</sup> check:

```

00FA67E0 8D6424 FC LEA ESP,DWORD PTR SS:[ESP-4]
00FA67E4 892C24 MOV DWORD PTR SS:[ESP],EBP
00FA67E7 8BEC MOV EBP,ESP
00FA67E9 8D6424 FC LEA ESP,DWORD PTR SS:[ESP-4]
00FA67ED 890C24 MOV DWORD PTR SS:[ESP],ECX
00FA67F0 8D6424 FC LEA ESP,DWORD PTR SS:[ESP-4]
00FA67F4 893424 MOV DWORD PTR SS:[ESP],ESI
00FA67F7 BE 805B1F01 MOV ESI,cnc3.011F5B80
00FA67FC 56 PUSH ESI
00FA67FD FF15 985B1F01 CALL DWORD PTR DS:[11F5B98] ntdll.RtlEnterCriticalSection
00FA6803 A1 B4FD2E01 MOV EAX,DWORD PTR DS:[12EFD84]
00FA6808 83E0 7F AND EAX,7F
00FA680B 3C 00 CMP AL,00
00FA680D 75 1A JNZ SHORT cnc3.00FA6829
00FA680F 8D45 FC LEA EAX,DWORD PTR SS:[EBP-4]
00FA6812 50 PUSH EAX
00FA6813 68 7C5A1F01 PUSH cnc3.011F5A7C ASCII "deroko"
00FA6818 C745 FC 01010001 MOV DWORD PTR SS:[EBP-4],101
00FA681F FF15 085A1F01 CALL DWORD PTR DS:[11F5A08] ADVAPI32.GetUserNameA
00FA6825 85C0 TEST EAX,EAX
00FA6827 74 20 JE SHORT cnc3.00FA6849
00FA6829 FF05 B4FD2E01 INC DWORD PTR DS:[12EFD84]
00FA682F 56 PUSH ESI
00FA6830 FF15 9C5B1F01 CALL DWORD PTR DS:[11F5B9C] ntdll.RtlLeaveCriticalSection
00FA6836 A1 945A1F01 MOV EAX,DWORD PTR DS:[11F5A94]
00FA683B 8B0D 845A1F01 MOV ECX,DWORD PTR DS:[11F5A84]
00FA6841 03C1 ADD EAX,ECX
00FA6843 0305 7C5A1F01 ADD EAX,DWORD PTR DS:[11F5A7C]
00FA6849 5E POP ESI
00FA684A C9 LEAVE
00FA684B C2 0400 RETN 4
00FA684E CC INT3

```

That's my user name over there ☺ Weeeeeee ☺ Well at line 0xFA6843 it takes 4 chars from name and adds them to EAX, this is very simple to patch so you can do it on your own. Not much stuff going on.

In simple words, if you run dump as different user, it will crash at some point.

And last and 8<sup>th</sup> check:

00FA6850	8B15 105A1F01	MOV EDX,DWORD PTR DS:[11F5A10]	ADVAPI32.77DD0000
00FA6856	66:813A 4D5A	CMP WORD PTR DS:[EDX],5A4D	
00FA685B	75 0C	JNZ SHORT cnc3.00FA6869	
00FA685D	A1 145A1F01	MOV EAX,DWORD PTR DS:[11F5A14]	
00FA6862	66:8138 4D5A	CMP WORD PTR DS:[EAX],5A4D	
00FA6867	74 04	JE SHORT cnc3.00FA686D	
00FA6869	33C0	XOR EAX,EAX	
00FA686B	EB 38	JMP SHORT cnc3.00FA68A5	
00FA686D	8B48 3C	MOV ECX,DWORD PTR DS:[EAX+3C]	
00FA6870	03C8	ADD ECX,EAX	
00FA6872	8B42 3C	MOV EAX,DWORD PTR DS:[EDX+3C]	
00FA6875	03D0	ADD EDX,EAX	
00FA6877	8B42 58	MOV EAX,DWORD PTR DS:[EDX+58]	
00FA687A	0B42 28	OR EAX,DWORD PTR DS:[EDX+28]	
00FA687D	56	PUSH ESI	
00FA687E	0B42 08	OR EAX,DWORD PTR DS:[EDX+8]	
00FA6881	8B71 58	MOV ESI,DWORD PTR DS:[ECX+58]	
00FA6884	0B71 28	OR ESI,DWORD PTR DS:[ECX+28]	
00FA6887	0B71 08	OR ESI,DWORD PTR DS:[ECX+8]	
00FA688A	03C6	ADD EAX,ESI	
00FA688C	0FB772 42	MOVZX ESI,WORD PTR DS:[EDX+42]	
00FA6890	0FB752 40	MOVZX EDX,WORD PTR DS:[EDX+40]	
00FA6894	03C6	ADD EAX,ESI	
00FA6896	03C2	ADD EAX,EDX	
00FA6898	0FB751 42	MOVZX EDX,WORD PTR DS:[ECX+42]	
00FA689C	0FB749 40	MOVZX ECX,WORD PTR DS:[ECX+40]	
00FA68A0	03C2	ADD EAX,EDX	
00FA68A2	03C1	ADD EAX,ECX	
00FA68A4	5E	POP ESI	
00FA68A5	C2 0400	RETN 4	

Very simple anti-dump check, it loads address of advapi32.dll in edx, and address of kernel32.dll to eax, now it plays a little bit with PE header. Of course on different windows version kernel32.dll and advapi32.dll probably will have different values there. It could happen that during MS updates those two, or at least one of them is updated, and in such PE will be screwed, and wrong value will be returned. One simple way to bypass this anti-dump is to run this procedure and get value returned in eax then simply patch this routine with **mov eax, that\_value/retn 4**

Only 2 more anti-dumps left to go over, only 2 more ☺

Next anti-dump is related to EntryPoint stored in PE header in memory and is located here:

013D886C	F8	CLC	
013D886D	8B3D 40883C01	MOV EDI,DWORD PTR DS:[13C8840]	
013D8873	133D 50014000	ADC EDI,DWORD PTR DS:[400150]	
013D8879	873C24	XCHG DWORD PTR SS:[ESP],EDI	
013D887C	814424 04 1A0000	ADD DWORD PTR SS:[ESP+4],1A	
013D8884	C3	RETN	
013D8885	FF0D 37C34301	DEC DWORD PTR DS:[143C337]	
013D888B	0F84 18F50BFF	JE cnc3.00497009	

adc edi, [400150] is actually adding OptionalHeader.AddressOfEntryPoint to edi. Your dump will have different EntryPoint, but in this real target, SecuROM expects it to be: **4d5b98h**. I have located at least 1 more check, but no need to look for all of them, it is more than enough to add extra code to the dump which will overwrite OptionalHeader.AddressOfEntryPoint with this value, and also will update OptionalHeader.SizeOfImage with value which SecuROM wants in its VM interpreter, so we will be injection this code into SecuROM:



```

loader:
    pusha
    call    __delta
__delta:
    pop     ebp
    sub     ebp, offset __delta

    call    getkernelbase
    xchg    eax, ebx
    gethash <VirtualProtect>
    call    getprocaddress, ebx, hash

    push    esp
    mov     ecx, esp
    call    eax, 400000h, 1000h, PAGE_READWRITE, ecx
    add     esp, 4

    mov     esi, 400000h
    add     esi, [esi+3ch]
    mov     [esi.pe_addressofentrypoint], 4d5b98h
    mov     [esi.pe_sizeofimage], 87a000h

    mov     eax, [ebp+old_entry_point]
    mov     [esp.Pushad_eax], eax
    popa
    jmp     eax

```

And now, do you remember that TLS stuff I mentioned earlier? Well here it goes:

00F19F59	53	PUSH EBX	
00F19F5A	56	PUSH ESI	
00F19F5B	FF15 409C3901	CALL DWORD PTR DS:[<&KERNEL32.GetLastError	ntdll.RtlGetLastWin32Error
00F19F61	FF35 98982A01	PUSH DWORD PTR DS:[12A9898]	
00F19F67	8B08	MOV EBX,EAX	
00F19F69	FF15 54621B01	CALL DWORD PTR DS:[11B6254]	kernel32.TlsGetValue
00F19F6F	8BF0	MOV ESI,EAX	
00F19F71	85F6	TEST ESI,ESI	
00F19F73	75 49	JNZ SHORT cnc3.00F19FB6	
00F19F75	68 8C000000	PUSH 8C	
00F19F7A	6A 01	PUSH 1	
00F19F7C	E8 D2FFFFFF	CALL cnc3.00F17F53	
00F19F81	8BF0	MOV ESI,EAX	
00F19F83	85F6	TEST ESI,ESI	
00F19F85	59	POP ECX	
00F19F86	59	POP ECX	
00F19F87	74 2D	JE SHORT cnc3.00F19FB6	
00F19F89	56	PUSH ESI	
00F19F8A	FF35 98982A01	PUSH DWORD PTR DS:[12A9898]	
00F19F90	FF15 58621B01	CALL DWORD PTR DS:[11B6258]	kernel32.TlsSetValue
00F19F96	85C0	TEST EAX,EAX	
00F19F98	74 1C	JE SHORT cnc3.00F19FB6	
00F19F9A	C746 54 E89A2A00	MOV DWORD PTR DS:[ESI+54],cnc3.012A9AE8	
00F19FA1	C746 14 01000000	MOV DWORD PTR DS:[ESI+14],1	
00F19FA8	FF15 789A3901	CALL DWORD PTR DS:[<&KERNEL32.GetCurrentThreadId	kernel32.GetCurrentThreadId
00F19FAE	834E 04 FF	OR DWORD PTR DS:[ESI+4],FFFFFFFF	
00F19FB2	8906	MOV DWORD PTR DS:[ESI],EAX	
00F19FB4	EB 08	JMP SHORT cnc3.00F19FB6	
00F19FB6	6A 10	PUSH 10	
00F19FB8	E8 BABBFFFF	CALL cnc3.00F15B77	
00F19FBD	59	POP ECX	
00F19FBE	53	PUSH EBX	
00F19FBF	FF15 7C9C3901	CALL DWORD PTR DS:[<&KERNEL32.SetLastError	ntdll.RtlSetLastWin32Error
00F19FC5	8BC6	MOV EAX,ESI	
00F19FC7	5E	POP ESI	
00F19FC8	5B	POP EBX	
00F19FC9	C3	RETN	
00F19FCA	60 10	PUSH 10	

It calls TlsGetValue(0x0F), so you will have to patch it with value you got like this:

00F19F59	B8 901E5802	MOV EAX,final.02581E90
00F19F5E	C3	RETN
00F19F5F	3901	CMP DWORD PTR DS:[ECX],EAX
00F19F61	FF35 98982A01	PUSH DWORD PTR DS:[12A9898]
00F19F67	8BD8	MOV EBX,EAX
00F19F69	FF15 54621B01	CALL DWORD PTR DS:[11B6254]
00F19F6F	8BF0	MOV ESI,EAX
00F19F71	85F6	TEST ESI,ESI
00F19F73	75 49	JNZ SHORT final.00F19FBE
00F19F75	68 8C000000	PUSH 8C
00F19F7A	6A 01	PUSH 1
00F19F7B	EB 00000000	CALL EBX

And so it has been done, now run your dumped/patched exe, and you will have your game up and running.

That's all folks...

## 6. Conclusion

Well as you may see it is relatively simple to fix SecuROM, still, going for the virgin file would be nice, but it requires more free time as you will have to reverse SecuROM VM to fix it properly.

## 7. References

[1] Ultimate Hooking Engine, deroko of ARTeam, <http://deroko.phearless.org>

## 8. Greetings



I wish to thank to all my mates in ARTeam for sharing their knowledge, to 29a for one of the best e-zines, unpack.cn crew (fly, shoooo, heXer, softworm, okododo), AnonymouS tut contributor, he know who he is, and of course, you for reading this document.

С вером у Бога, deroko of ARTeam

# Well deep Inside SecuRom by Anonymous

## 1. The funny side of things

The authors of SecuRom cracks me up:

DUM	nunc		Imag R	RWE
DUM	bibendum		Imag R	RWE
DUM	est		Imag R	RWE
DUM	.securom	imports	Imag R	RWE
			Imag R	RWE

"Time to drink .securom"

And in the redirector proc:

"nobody move, nobody gets hurt"  
 "Masses Against the Classes"  
 "yates is still ere.something kinda Ooooh"

Are there no honor among thieves Mr. Yates ??

## 2. Code morphing

This PUSH 10 and JMP look a bit strange:

00961F32	\$ 6A 10	PUSH 10
00961F34	^E9 0D89EBFF	JMP TEST_DUM.0081A846
00961F39	> E8 0A69FFFF	CALL TEST_DUM.00958848
00961F3E	. A1 B0DC6605	MOV EAX,DWORD PTR DS:[566DCB0]
00961F43	85C0	TEST EAX,EAX

Follow jump and you will end up here:

0081A846	> 56	PUSH ESI	0023FE90	00000010	
0081A847	. 9C	PUSHFD	0023FE94	00958FA4	RETURN to TEST_DUM.00958FA4 from TEST_DUM.
0081A848	. F9	STC	0023FE98	0566D8F0	TEST_DUM.0566D8F0
0081A849	. 8B35 28C1E305	MOV ESI,DWORD PTR DS:[5E3C128]	0023FE9C	00000F00	
0081A84F	. 1335 92D04900	ADC ESI,DWORD PTR DS:[49D092]			
0081A855	. 9D	POPF			
0081A856	. 873424	XCHG DWORD PTR SS:[ESP],ESI	0023FE8C	009EA518	TEST_DUM.009EA518
0081A859	> E9 DB761400	JMP TEST_DUM.00961F39	0023FE90	00000010	
0081A85F	. 8B35 28C1E305	MOV ESI,DWORD PTR DS:[5E3C128]	0023FE94	00958FA4	RETURN to TEST_DUM.00958FA4 from TEST_DUM.
			0023FE98	0566D8F0	TEST_DUM.0566D8F0
			0023FE9C	00000F00	

This is the stack at 081A846h (beginning)

This the stack at at 081A859h (end)

As one can all this does is that it pushes a value onto stack. Anyway, this is not important to us. The dump works fine without patching this ;)

### 3. Basic API redirection

Here is where we first meet a redirection of a call to API:

00961F5F	• 68 ECA49E00	PUSH TEST_DUM.009EA4EC	ASCII "InitializeCriticalSectionAndSpinCount"
00961F64	• 50	PUSH EAX	
00961F65	• FF15 380A6705	CALL DWORD PTR DS:[5670A38]	TEST_DUM.05693644
00961F68	• A3 B0DC6605	MOV DWORD PTR DS:[566DCB0],EAX	
00961F70	• 85C0	TEST EAX,EAX	

Let's trace into 05693644h

05693644	55	PUSH EBP	kernel32.GetProcAddress
05693645	8BEC	MOV EBP,ESP	
05693647	53	PUSH EBX	
05693648	56	PUSH ESI	
05693649	57	PUSH EDI	
0569364A	60	PUSHAD	
0569364B	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
0569364E	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
05693651	FF15 E813B005	CALL DWORD PTR DS:[5BD13E8]	
05693657	8BF0	MOV ESI,EAX	
05693659	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+C]	
0569365C	C1E8 10	SHR EAX,10	
0569365F	66:85C0	TEST AX,AX	
05693662	✓74 12	JE SHORT game.05693676	
05693664	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
05693667	56	PUSH ESI	
05693668	E8 E0BC0000	CALL game.0569F34D	
0569366D	85C0	TEST EAX,EAX	
0569366F	59	POP ECX	
05693670	59	POP ECX	
05693671	8945 0C	MOV DWORD PTR SS:[EBP+C],EAX	
05693674	✓75 03	JNZ SHORT game.05693679	
05693676	8975 0C	MOV DWORD PTR SS:[EBP+C],ESI	
05693679	61	POPAD	
0569367A	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+C]	
0569367D	5F	POP EDI	
0569367E	5E	POP ESI	
0569367F	5B	POP EBX	
05693680	5D	POP EBP	
05693681	C2 0800	RETN 8	
05693684	55	PUSH EBP	
05693685	8BEC	MOV EBP,ESP	

As one can see, this is actually a call to GetProcAddress.

If we load our dump into LordPE and take a look at the Directory Table (ImportTable):

[ ImportTable ]					
DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk
KERNEL32.dll	05A0910B	00000000	00000000	05A07058	05A08AC3
USER32.dll	05A09423	00000000	00000000	05A07DF4	05A08DDB
ole32.dll	05A09573	00000000	00000000	05A08347	05A08F2B
d3dx9_30.dll	05A09597	00000000	00000000	05A083DF	05A08F4F
DSOUND.dll	05A095A7	00000000	00000000	05A0843F	05A08F5F
Thunk RVA	Thunk Offset	Thunk Value	Hint	ApiName	
05A08B7B	009C6B7B	05693644	6805	× @	
05A08B7F	009C6B7F	056936CE	0000	hÜ6ë èæaj N0ë RS	
05A08B83	009C6B83	7C80ABDE	-	Memory Address: 7C80ABDEh	
05A08B87	009C6B87	7C802367	-	Memory Address: 7C802367h	
05A08B8B	009C6B8B	7C85C4AB	-	Memory Address: 7C85C4ABh	
05A08B8F	009C6B8F	7C834EB1	-	Memory Address: 7C834EB1h	
05A08B93	009C6B93	7C8361EE	-	Memory Address: 7C8361EEh	
05A08B97	009C6B97	7C8138FC	-	Memory Address: 7C8138FCh	
05A08B9B	009C6B9B	7C86D8FF	-	Memory Address: 7C86D8FFh	
Number Of Thunks: C5h / 197d (FirstThunk chain)					<input checked="" type="checkbox"/> View always FirstThunk

Running through the ImportTable (kernel32.dll) we noticed the following API's gets redirected:

GetProcAddress (5670A38)  
LoadLibraryA (5670A3C)  
ExitProcess (5670AD8)  
TerminateProcess (5670AE0)

We need to fix these API's. One can do this either by hand or by using ImpRec or ReVirgin. I prefer nothing it automatically coding a unpacker !!!

## 4. Code splicing

The code splicing in SecuRom usually looks like this:

0095A859	-FF25 BCB6E305	JMP DWORD PTR DS:[5E3B6BC]	TEST_DUM.05EB4000
0095A85F	3E:6A FF	PUSH -1	Superfluous prefix
0095A862	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
0095A868	68 90DF9900	PUSH TEST_DUM.0099DF90	
0095A86D	50	PUSH EAX	
0095A86E	. 64:8925 000000	MOV DWORD PTR FS:[0],ESP	
0095A875	. 83EC 54	SUB ESP,54	
0095A878	. 53	PUSH EBX	
0095A879	. 55	PUSH EBP	
0095A87A	. 8B6C24 6C	MOV EBP,DWORD PTR SS:[ESP+6C]	
0095A87E	. 56	PUSH ESI	
0095A87F	. 57	PUSH EDI	
0095A880	. 55	PUSH EBP	
0095A881	. 8BF9	MOV EDI,ECX	
0095A883	. 68 9EA89500	PUSH TEST_DUM.0095A89E	
0095A888	. 68 508C8F00	PUSH TEST_DUM.008F8C50	
0095A88D	. C3	RETN	Entry address
0095A88E	. 9C	DB 9C	
0095A88F	. D4	DB D4	

Once we trace into 05EB4000h we will enter the wonderful world of SecuRom redirection. I will spare you the heartbreak for going through all the calculation loops, but what it basically does is redirection you to the code splicing code. Anyway, here is the beginning:

05EB4000	68 1A40E805	PUSH game.05EB401A
05EB4005	68 F9314000	PUSH game.004031F9
05EB400A	68 98E66F05	PUSH game.056FE698
05EB400F	9C	PUSHFD
05EB4010	816C24 04 D84D00	SUB DWORD PTR SS:[ESP+4],4DD8
05EB4018	9D	POPFD
05EB4019	C3	RETN
05EB401A	F2	REPNE SCASB

It will eventually end up here:

05E6EEE9	83EC 48	SUB ESP,48
05E6EEEC	53	PUSH EBX
05E6EEED	BB 80040000	MOV EBX,480
05E6EEF2	53	PUSH EBX
05E6EEF3	68 0AEFE605	PUSH game.05E6EF0A
05E6EEF8	68 9A5A9500	PUSH game.00955A9A
05E6EEFD	C3	RETN
05E6EEFF	24	AND AL,79

00955A9A	FF35 94DC6605	PUSH DWORD PTR DS:[566DC94]
00955AA0	FF7424 08	PUSH DWORD PTR SS:[ESP+8]
00955AA4	E8 C5FFFFFF	CALL game.00955A6E
00955AA9	59	POP ECX
00955AAB	59	POP ECX
00955AAB	C3	RETN
00955AAC	CC	INT3

05E6EF0A	85C0	TEST EAX,EAX
05E6EF0C	59	POP ECX
05E6EF0D	75 0B	JNZ SHORT game.05E6EF1A
05E6EF0F	0B05 0815E405	OR EAX,DWORD PTR DS:[5E41508]
05E6EF15	E9 BC020000	JMP game.05E6F1D6
05E6EF1A	A3 20E16605	MOV DWORD PTR DS:[566E120],EAX
05E6EF1F	C705 08E06605 20	MOV DWORD PTR DS:[566E0D8],20
05E6EF29	8D88 80040000	LEA ECX,DWORD PTR DS:[EAX+480]
05E6EF2F	EB 1E	JMP SHORT game.05E6EF4F
05E6EF31	8308 FF	OR DWORD PTR DS:[EAX],FFFFFFFF
05E6EF34	8360 08 00	AND DWORD PTR DS:[EAX+8],0
05E6EF38	C640 04 00	MOV BYTE PTR DS:[EAX+4],0
05E6EF3C	C640 05 0A	MOV BYTE PTR DS:[EAX+5],0A
05E6EF40	8B0D 20E16605	MOV ECX,DWORD PTR DS:[566E120]
05E6EF46	83C0 24	ADD EAX,24
05E6EF49	81C1 80040000	ADD ECX,480
05E6EF4F	3BC1	CMP EAX,ECX
05E6EF51	72 DE	JB SHORT game.05E6EF31
05E6EF53	55	PUSH EBP
05E6EF54	56	PUSH ESI
05E6EF55	57	PUSH EDI
05E6EF56	8D4424 14	LEA EAX,DWORD PTR SS:[ESP+14]
05E6EF5A	50	PUSH EAX
05E6EF5B	68 D65E4200	PUSH game.00425ED6
05E6EF60	68 A7D545B7	PUSH B745D5A7
05E6EF65	9C	PUSHFD
05E6EF66	817424 04 674D20	XOR DWORD PTR SS:[ESP+4],B22A4D67
05E6EF6E	9D	POPFD
05E6EF6F	58	POP EAX
05E6EF70	FFD0	CALL EAX
05E6EF72	55:837C	MOV WORD PTR DS:[ESP+2],0

As one might see this is part of the code splicing. However we can fix this by forcing the jump at 095A859h to jump to 05E6EEE9h. Now here is something interesting... Look above at line 05E6EF70h... What is this ?? Let's trace... A new chapter begins...

## 5. Advanced API redirections

Like with the code splicing the advanced API redirection is triggered by a run through the calculation loops. Most often around 10 times. No big deal... After a little tracing we end up here:

068E25D3	68 00000000	PUSH 0
068E25D8	68 00000000	PUSH 0
068E25DD	68 00000000	PUSH 0
068E25E2	68 00000000	PUSH 0
068E25E7	68 00000000	PUSH 0
068E25EC	68 00000000	PUSH 0
068E25F1	68 00000000	PUSH 0
068E25F6	68 00000000	PUSH 0
068E25FB	68 00000000	PUSH 0
068E2600	68 00000000	PUSH 0
068E2605	68 00000000	PUSH 0
068E260A	8D6424 30	LEA ESP,DWORD PTR SS:[ESP+30]
068E260E	C3	RETN

Return to 7C801EEE (kernel32.GetStartupInfoA)

This is a call to GetStartupInfoA !!

As one can see it's not advanced at all. I only chose to call it this because of the calculation/obfuscation loops.

Often calls to API looks very similar to calls to another SecuRom trick.... The Virtual Machine (VM)... Let's take a look at this feature...



## 6. Virtual Machine

05E4D82C	FFD0	CALL EAX
05E4D82E	3B05 3811E405	CMP EAX,DWORD PTR DS:[5E41138]
05E4D834	✓0F85 62010000	JNZ game.05E4D99C
05E4D83A	50	PUSH EAX
05E4D83B	9C	PUSHFD
05E4D83C	C1EA 00	SHR EDI,0
05E4D83F	B8 90136805	MOV EAX,game.05681390
05E4D844	90	NOP
05E4D845	54	PUSH ESP
05E4D846	68 B6C5E305	PUSH game.05E3C5B6
05E4D848	FFD0	CALL EAX
05E4D84D	8BFF	MOV EDI,EDI
05E4D84F	9D	POPFD
05E4D850	870424	XCHG DWORD PTR SS:[ESP],EAX
05E4D853	33C0	XOR EAX,EAX

0594A3A8	55	PUSH EBP	
0594A3A9	8BEC	MOV EBP,ESP	
0594A3AB	83EC 40	SUB ESP,40	
0594A3AE	53	PUSH EBX	
0594A3AF	56	PUSH ESI	
0594A3B0	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
0594A3B3	8B46 18	MOV EAX,DWORD PTR DS:[ESI+18]	
0594A3B6	83F8 58	CMP EAX,58	
0594A3B9	57	PUSH EDI	
0594A3BA	✓0F8F 2D060000	JG game.0594A9ED	
0594A3C0	✓0F84 ED050000	JE game.0594A9B3	
0594A3C6	83F8 26	CMP EAX,26	
0594A3C9	✓0F8F 0B030000	JG game.0594A6DA	
0594A3CF	✓0F84 35020000	JE game.0594A60A	
0594A3D5	83F8 15	CMP EAX,15	
0594A3D8	✓0F8F 95010000	JG game.0594A573	
0594A3DE	✓0F84 76010000	JE game.0594A55A	
0594A3E4	83F8 03	CMP EAX,3	
0594A3E7	✓0F84 5B010000	JE game.0594A548	
0594A3ED	83F8 0A	CMP EAX,0A	
0594A3F0	✓0F84 14010000	JE game.0594A50A	
0594A3F6	83F8 10	CMP EAX,10	
0594A3F9	✓0F84 DF000000	JE game.0594A4DE	
0594A3FF	83F8 14	CMP EAX,14	
0594A402	✓0F85 FC070000	JNZ game.0594AC04	
0594A408	833D 7C4EC005	CMP DWORD PTR DS:[5C04E7C],64	
0594A40F	✓0F82 A2000000	JB game.0594A4B7	
0594A415	6A 1C	PUSH 1C	
0594A417	8D45 E4	LEA EAX,DWORD PTR SS:[EBP-1C]	
0594A41A	50	PUSH EAX	
0594A41B	56	PUSH ESI	
0594A41C	FF15 784EC005	CALL DWORD PTR DS:[5C04E78]	
0594A422	33FF	XOR EDI,EDI	
0594A424	83F8 1C	CMP EAX,1C	
0594A427	✓0F85 84000000	JNZ game.0594A4B1	
0594A42D	8B4E 20	MOV ECX,DWORD PTR DS:[ESI+20]	
0594A430	83F9 06	CMP ECX,6	
0594A433	A1 804EC005	MOV EAX,DWORD PTR DS:[5C04E80]	

kernel32.VirtualQuery

... ☺