



Manual Unpacking Safecast

Anonymous contribution submitted to ARTeam

Version 1.0 - June 2006

Abstract.....	2
1. lalalala'ing to the OEP.....	3
2. Rebuilding the horribly crippled IAT	5
3. Stolen Commands	16
4. FF15 Annoyance	20
5. Long Jumps Problem	23
6. Overall Code into one piece.....	30
7. History.....	35

Keywords

MUP, safecast

Disclaimer of ARTeam: We received this tutorial from an anonymous author (an author who doesn't want to see his name published) who contacted us in order to see this paper published. Despite it's not our usual style to publish tutorials from members outsiders and less to publish anonymous tutorials, we did it. The content is really interesting and technically well done, so we decided to give it out on our tutorials pages.

But please consider that we want to deliver the same quality content and information you are used to from ARTeam tutorials. Any tutorial submitted to us has to pass the inspection and approval of the entire team before publication.

We applied our usual tutorials formatting to the text we received, and verified correctness of information on the technical side. Any other thing is responsibility of the author, even if the work is not original, do not complain us.

- Internal Editor: Shub-Nigurrath

Disclaimer of the Author: This document may be used for potentially illegal and/or destructive purposes. This is not the intention of this document. This document was created for informational purposes, to educate people. If you are reading this document with the intent of using it for illegal purposes, DELETE IT NOW! If your countries laws prohibit you from reading this document, DELETE NOW! If this document in any ways suggests breaking your countries law, DELETE IT NOW!

The author of this document and the people hosting it DO NOT want this document to be used for illegal purposes, and accept no responsibility for any actions that are inspired, suggested, or arise from reading this document.

To sum it all up: don't do anything illegal with this. I do not accept any responsibility for anything you have done or will do.



Abstract

Ok guys, I'm bored. So, I got a game, and it had a trial, and I was very sad because I liked the game and it expired after 60 minutes and it wasn't ActiveMark so I couldn't crack it so I cried like and emo for an entire week over it before I went out and tried it myself.

The product is this tutorial.

Target: Zuma Deluxe Popcap (Get it off of Boontygames)

First of all, I would like to say that this tutorial is in no way original. I have read other safedisc MUP tutorials by magico and Peex. After reading them, I combined their methods with mine, and used them. If you read their tutorials before mine, you will definitely notice the similarities. So, I would like to send out a big thank you to magico and Peex. Their knowledge has helped me greatly.

Also, let me say another thing. I want this tutorial to be readable by anyone, noobs, leet, gangsta, Al-Quaeda, or Strogg. I will do my best to explain every single thing. I am a noob myself, so my point of view might be a little unprofessional, and my code embarrassing, but bear with me.

Ok, that's about it. If I remember anything else, I will add it here.

Ok, before we dive into Safecast, I would just like to note that Safecast is the little brother of safedisc. The two protections are almost completely identical, just that Safedisc supports SDAPI (interesting stuff) and Safedisc can only be found in CD games.

I poked around the Macrovision website, only to find marketing bullshit on it, no concrete things that would help us in reversing this protection. Oh well, all the more fun for us =).

The tools I used are:

- Ollydbg (Thanks Oleh!) + Ollydump(gigapede?).
- Imprec (Thanks MackT!)
- LordPE (Thanks y0da!)
- MASM (Thanks... A lot of people ;p)



1. lalalala'ing to the OEP

Ok, lets open up the exe in Olly. The standard Safedisc OEP should be staring at us.

0058C09E	> 55	PUSH EBP
0058C09F	8BEC	MOV EBP,ESP
0058C0A1	60	PUSHAD
0058C0A2	BB 9EC05800	MOV EBX,OFFSET Zuma.<ModuleEntryPoint>

Hmm, lets look around. Hmm, if you scroll down, you will see a longjump to .text. I wonder what that is:

0058C159	-E9 EA67F5FF	JMP Zuma.004E2948
0058C15E	CC	INT3
0058C15F	CC	INT3

Put a Breakpoint on the jump by selecting it and pressing F2. Now press F9 to run the exe.

Aww, it found our debugger. That makes me very sad. Ok, so what API could be used to detect our debugger? Hmm lets think... Ahh yes IsDebuggerPresent!

Restart Zuma, press ALT+F1 to bring up the OllyDBG command line, and type in BP IsDebuggerPresent. This will set a breakpoint on the IsDebuggerPresent API. Run the game with F9, and it will kindly break here:

77E72740	> 64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
77E72746	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]
77E72749	0FB640 02	MOVZX EAX,BYTE PTR DS:[EAX+2]
77E7274D	C3	RETN

If there is a debugger present, IsDebuggerPresent returns 1. Otherwise, it returns 0. Trace with F8 over the ret, and you will see that EAX is 0. Heh, it didn't find the debugger... yet
Keep on tracing with F8, until you are here:

667045AA	83C4 04	ADD ESP,4
667045AD	66:85C0	TEST AX,AX
667045B0	0F85 02010000	JNZ 667046B8
667045B6	F6C3 04	TEST BL,4
667045B9	76 12	JBE SHORT 667045CD
667045BB	56	PUSH ESI
667045BC	E8 BF8C0000	CALL 6670D280
667045C1	83C4 04	ADD ESP,4
667045C4	66:85C0	TEST AX,AX
667045C7	0F85 EB000000	JNZ 667046B8
667045CD	F6C3 08	TEST BL,8
667045D0	76 12	JBE SHORT 667045E4
667045D2	56	PUSH ESI
667045D3	E8 D88B0000	CALL 6670D1B0
667045D8	83C4 04	ADD ESP,4
667045DB	66:85C0	TEST AX,AX



This type of routine goes on and on, almost forever :X. So much antidebug. It seems that it calls each antidebug, then if a debugger is detected, AX will return to be non-zero. Then, the app will jump to 667046B8. Keep on tracing with F8, until we get to 667046B8 (We will eventually as a check trips us up). Trace over the ret, and then trace over the next ret. You should end up here:

66702E98	59	POP ECX	
66702E99	3D 00200000	CMP EAX,2000	
66702E9E	59	POP ECX	
66702E9F	74 23	JE SHORT 66702EC4	
66702EA1	3D 00400000	CMP EAX,4000	
66702EA6	^74 D9	JE SHORT 66702E81	
66702EA8	33C9	XOR ECX,ECX	
66702EAA	3D 00000100	CMP EAX,10000	; UNICODE "=:::=:\"
66702EAF	0F94C1	SETC CL	

Well, this routine checks EAX for 2000, 4000, and 10000. While experimenting with the Safedisc Antidebug, I found that Treeayoot's (or however you spell it) anti-anti debug plugin works fine on Safedisc with System hook enabled. However, it also corrupts my Olly, and I had to redownload. Well, with his plugin, EAX here is 10000. We glance up at our EAX and we should see that it is 4000. So, from that we can deduce that if the debugger is detected, EAX will return 4000, if not it will return 10000. You might wonder what happens if it is 2000? Well, just go to the webpage of Yates, and you will see in his little doc about the antidebug of safedisc that it is 2000 when you do not have Administrator privileges. I hope thats not a problem for anyone reading this

So, to bypass the antidebug we will change EAX to 00100000. Now continue tracing with F8

lalalalalalalalalala

After a couple of intense seconds of flexing the muscles in our index finger, we will run across the nag, that politely tells us that we can buy the game. Of course all of us will after reading this tutorial, riiiiiiiiiiight?

Tell the nag to go away with a simple click to the "play for free" button. Keep on tracing

lalalalalalalalalala

Ohoho. After some tracing, we will run across our friendly jump here:

0058C159	-E9 EA67F5FF	JMP Zuma.004E2948
0058C15E	CC	INT3
0058C15F	CC	INT3

Press F8, and fix your eyes on the OEP. Know now that the easiest part is done :)



004E2948	. 6A 60	PUSH 60
004E294A	. 68 A88B5100	PUSH Zuma.00518BA8
004E294F	. E8 64790000	CALL Zuma.004EA2B8
004E2954	. BF 94000000	MOV EDI,94
004E2959	. 8BC7	MOV EAX,EDI
004E295B	. E8 D0F7FFFF	CALL Zuma.004E2130
004E2960	. 8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
004E2963	. 8BF4	MOV ESI,ESP
004E2965	. 893E	MOV DWORD PTR DS:[ESI],EDI
004E2967	. 56	PUSH ESI
004E2968	. FF15 F0E04F00	CALL DWORD PTR DS:[4FE0F0]
004E296E	. 8B4E 10	MOV ECX,DWORD PTR DS:[ESI+10]

Here is the fast way to get to the OEP:

- BP IsDebuggerPresent
- BP On JMP to OEP
- F9 to run
- CTRL + F9 until EAX = 4000. Change EAX to 10000 when you see this.
- F9 until you hit JMP to OEP. Also, say hello to the friendly nag that pops up each bloody time we run this game.
- F8 to get to OEP.

2. Rebuilding the horribly crippled IAT

Now that we are at the OEP, lets try dumping, eh? Use OllyDump to dump (remember to always uncheck rebuild Imports!). label the dump as "OEP_dump.exe". Fire up Imprec, select our process (Zuma.exe), and let it load. It will inform us in the text window of Imprec that Imagebase is 00400000. Thanks, we'll need that!.

Now look at the offset of the OEP in Olly: 004E2948

However, Imprec only accepts raw offsets (offsets without imagebase).

So, $004E2948 - 00400000 = 000E2948$. So, type into the OEP box of Imprec 000E2948, and press on the Get Imports! Button.

Well, look at all those invalid thunks. Sheesh, hurts my eyes. Let remedy that, shall we?

Double click on the first invalid think to look at the offset the first invalid import is pointing too. You should see: 00E1809F

Well, at the OEP, assemble a JMP 00E1809F, then press F8 to trace over the jump you just assembled. You should be here:



00E1809F	68 9412EABF	PUSH BFEA1294
00E180A4	9C	PUSHFD
00E180A5	60	PUSHAD
00E180A6	54	PUSH ESP
00E180A7	68 DF80E100	PUSH 0E180DF
00E180AC	E8 2B739665	CALL ~df394b.6677F3DC
00E180B1	83C4 08	ADD ESP,8
00E180B4	6A 00	PUSH 0
00E180B6	58	POP EAX
00E180B7	61	POPAD
00E180B8	9D	POPFD
00E180B9	C3	RETN

Hmm, this guy is calling the Safedisc DLL. Ok, let trace with F7 into it. You should get to here:

6677F3DC	55	PUSH EBP
6677F3DD	8BEC	MOV EBP,ESP
6677F3DF	83EC 34	SUB ESP,34
6677F3E2	53	PUSH EBX
6677F3E3	56	PUSH ESI

Ok, somewhere in this routine, we will be getting to the correct import. Keep your eyes sharp! Start tracing with F8, until you reach here:

6677F6CD	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677F6D0	61	POPAD
6677F6D1	9D	POPFD
6677F6D2	C3	RETN ; <- Stop tracing here

Well, when we are over the RETN, look at the top of the stack:

0012FFC0 77DD17D8 ADVAPI32.RegCloseKey

Hohoho, the correct import. So, looks like this ret is an ideal place to grab the correct import. Also, I would like to direct your attention to this ret here:

6677F548	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677F54B	61	POPAD
6677F54C	9D	POPFD
6677F54D	C3	RETN

Its another place where Safedisc sometimes rets to the correct import. We should hook here too =).

Ok guys, now its thinking time. We have around 100 invalid imports... waaaaaay too much to fix manually for a lazy person. Since we are lazy people (riiiiiiiiiiiiiiiiiiiiiight??) we will have to somehow automate the process of correcting the imports. Thats where coding comes in. We are going to code a proggy that will fix all the imports for us, and then inject it into the process (preferably the PE Header, at 00400500). At first, I coded an unwrapper, but for this tutorial I would like us to use injected code, since it is much easier to debug, and see what goes wrong.

The program will have to do the following things:



- Search IAT for imports that are redirected
- Call the redirected Imports
- Overwrite the Old, redirected import with the new one.

I would also like to say that to easily resolve later parts of this protection, we will need to keep the original IAT intact. So, the program will also have to make a copy of the IAT. I placed this copy of the IAT at stxt371, because that is the loader section, and it is no longer being used by anything. So, without anything else to add, I present to you the code we will be injecting to fix the IAT:



```
Title imports
.386
.model small, stdcall
Option casemap :none
.code

; NOTE: For .rdata size use the IAT size kindly provided by Imprec
_IB          equ 00400000h      ; ImageBase
_rdata       equ 000FE000h      ; Section infos
_rdataSize   equ 00000398h      ; "
_stxt371     equ 0018C000h      ; "
_stxt371Size equ 00000398h      ; same a .rdata size since we are using
                                   ; it as temp. iat
_lowerrange  equ 00E00000h      ; lower range of all redirected calls
_higherrange equ 00F00000h      ; high range of all redirected calls

start:
pushad
pushfd
mov eax, (_IB+_rdata)           ; beginning of IAT
mov ebx, (_IB+_stxt371)         ; beginning of STXT 371, which is
                                   ; used as temporary IAT

_iloop:
mov ecx, dword ptr [eax]        ; retrieve address of call
cmp ecx, _lowerrange            ; is it below the lower range
                                   ; of the redirected calls?
jl _invalidcall                 ; if so, jump to the call is valid
cmp ecx, _higherrange           ; is it above the zone of redirected calls?
ja _invalidcall                 ; if so, jump to call is valid

_iresolve:
call ecx                        ; will call ecx so that it retrieves
                                   ; the correct value in edx
mov dword ptr [ebx], edx        ; put correct value into temporary IAT
add eax, 4                      ; move pointer to next call in IAT
add ebx, 4                      ; move pointer to next space in temporary IAT
cmp eax, (_IB+_rdata+_rdataSize) ; are we at the end of IAT=1000
                                   ; (zone of all redirected calls)
je _iend                       ; if so, let us end
jmp _iloop                      ; if not, continue program

_invalidcall:
mov dword ptr [ebx], ecx        ; call is not redirect, so just copy
                                   ; it into temporary IAT
add eax, 4                      ; move pointer to next call in IAT
add ebx, 4                      ; move pointer to next space in temporary IAT
cmp eax, (_IB+_rdata+_rdataSize) ; are we at the end of IAT?
je _iend                       ; if so, let us end
jmp _iloop                      ; if not, continue program

_iend:
int 3                          ; int 3 will cause the debugger to break,
                                   ; and we will know that we are done.

popfd
popad

end start
```




If you are wondering what `higherrange` and `lowerrange` are, they are the range of offset where the safedisc redirection code is stored in memory. You get them by just approximating.

Take this code above, paste it to someplace such as `C:\injecting.asm`, and then assemble it with MASM. Now, open another window of Olly, and in this second Olly window open `C:\injecting.exe`.

Now, go to the Olly with the game process opened. Restart the process, and get to the OEP. At the OEP assemble a `JMP 00400500`.

Now, press `CTRL+G`, and goto `00400500`. At `00400500`, select a bunch of code (its all the same, I know, but do it). Once you have around ~50 lines selected, go to the OllyDBG window with `C:\injecting.exe` open. Select all the code there, and

Right Click => Binary => Binary Copy.

Now go to the window with the game process open, and press

Right Click => Binary => Binary Paste.

You will now see your code be pasted into the exe. Excellent.

Now, press `F8`, and the EIP will change to `00400500`. There is one last thing that we need to do before we start the code to fix up our exe.

Remember how I talked about the two places that safedisc rets from to go to the correct API? Well, we will now 'hook' them (for lack of a better word) so that they return the correct import in EDX.

First, let us go to the offset of one of the rets. Now, press `CTRL+B`, and type in `00 00 00 00 00 00 00 00 00 00`, then enter. This will makes Olly do a binary search for a code cave, into which we will inject our code. Olly will wind up here:



6677FD60	0000	ADD BYTE PTR DS:[EAX],AL
6677FD62	0000	ADD BYTE PTR DS:[EAX],AL
6677FD64	0000	ADD BYTE PTR DS:[EAX],AL
6677FD66	0000	ADD BYTE PTR DS:[EAX],AL
6677FD68	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6A	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6C	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6E	0000	ADD BYTE PTR DS:[EAX],AL
6677FD70	0000	ADD BYTE PTR DS:[EAX],AL
6677FD72	0000	ADD BYTE PTR DS:[EAX],AL
6677FD74	0000	ADD BYTE PTR DS:[EAX],AL
6677FD76	0000	ADD BYTE PTR DS:[EAX],AL
6677FD78	0000	ADD BYTE PTR DS:[EAX],AL
6677FD60	0000	ADD BYTE PTR DS:[EAX],AL
6677FD62	0000	ADD BYTE PTR DS:[EAX],AL
6677FD64	0000	ADD BYTE PTR DS:[EAX],AL
6677FD66	0000	ADD BYTE PTR DS:[EAX],AL
6677FD68	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6A	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6C	0000	ADD BYTE PTR DS:[EAX],AL
6677FD6E	0000	ADD BYTE PTR DS:[EAX],AL
6677FD70	0000	ADD BYTE PTR DS:[EAX],AL
6677FD72	0000	ADD BYTE PTR DS:[EAX],AL
6677FD74	0000	ADD BYTE PTR DS:[EAX],AL
6677FD76	0000	ADD BYTE PTR DS:[EAX],AL
6677FD78	0000	ADD BYTE PTR DS:[EAX],AL

Go back to one of the ret places, and do a binary copy of the following code:

6677F548	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677F54B	61	POPAD
6677F54C	9D	POPFD
6677F54D	C3	RETN

Now, go back to our code cave (just press CTRL+B, ENTER), select a couple of lines of the code there, and to a binary paste so it should look like this:

6677FD60	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677FD63	61	POPAD
6677FD64	9D	POPFD
6677FD65	C3	RETN

Now, click on the line 6677FD65, and assemble a POP EDX, RETN, so now our code cave should look like this:

6677FD60	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677FD63	61	POPAD
6677FD64	9D	POPFD
6677FD65	5A	POP EDX
6677FD66	C3	RETN

Now, go back to both ret places, and assemble a JMP to our code cave (JMP 6677FD60). The ret places should look like this:

1st ret place:



6677F542	FF15 4C207866	CALL DWORD PTR DS:[<&KERNEL32.SetEvent>]
6677F548	E9 13080000	JMP ~df394b.6677FD60
6677F54D	C3	RETN

2nd ret place:

6677F6C7	FF15 4C207866	CALL DWORD PTR DS:[<&KERNEL32.SetEvent>]
6677F6CD	E9 8E060000	JMP ~df394b.6677FD60
6677F6D2	C3	RETN

Final thing to do: Press ALT+M to go to the memory map, and set full access on sections .text and .rdata.

Ok, that's it, time to execute. Lets the injected code run, and it will fix all the imports and put them into the temp IAT.

Well, you just use binary copy to copy the temp IAT to the real IAT (I recommend that you do that in the dump window of olly, the one with all the hex data.), and dump the exe (remember to set correct EP!(the OEP)) to IAT_Dump.exe. Use Imprec to rebuild the imports. Here is my listing of all the imports, so you can compare to it ;)



Target: C:\Program Files\BoontyGames\Zuma Deluxe PopCap\Zuma.exe				
OEP: 000E2948	IAT RVA: 000FE000	IAT Size: 00000398		
FThunk: 000FE000	NbFunc: 00000006			
1	000FE000	advapi32.dll	01C9	RegCloseKey
1	000FE004	advapi32.dll	01EC	RegQueryValueExA
1	000FE008	advapi32.dll	01CD	RegCreateKeyExA
1	000FE00C	advapi32.dll	01E2	RegOpenKeyExA
1	000FE010	advapi32.dll	01D2	RegDeleteValueA
1	000FE014	advapi32.dll	01F9	RegSetValueExA
FThunk: 000FE01C	NbFunc: 0000000D			
1	000FE01C	gdi32.dll	003A	CreateFontA
1	000FE020	gdi32.dll	01BD	GetTextMetricsA
1	000FE024	gdi32.dll	024F	TextOutA
1	000FE028	gdi32.dll	023D	SetTextColor
1	000FE02C	gdi32.dll	0090	DeleteObject
1	000FE030	gdi32.dll	003B	CreateFontIndirectA
1	000FE034	gdi32.dll	0196	GetObjectA
1	000FE038	gdi32.dll	020F	SelectObject
1	000FE03C	gdi32.dll	016C	GetDeviceCaps
1	000FE040	gdi32.dll	0051	CreateSolidBrush
1	000FE044	gdi32.dll	01B5	GetTextExtentPoint32A
1	000FE048	gdi32.dll	0217	SetBkMode
1	000FE04C	gdi32.dll	01C8	IntersectClipRect
FThunk: 000FE054	NbFunc: 00000072			
1	000FE054	kernel32.dll	0046	CreateDirectoryA
1	000FE058	kernel32.dll	02F1	SetCurrentDirectoryA
1	000FE05C	kernel32.dll	01CB	GetTickCount
1	000FE060	kernel32.dll	004A	CreateEventA
1	000FE064	kernel32.dll	0137	GetCurrentThread
1	000FE068	kernel32.dll	0327	SetThreadPriority
1	000FE06C	kernel32.dll	0030	CloseHandle
1	000FE070	kernel32.dll	02B6	ResetEvent
1	000FE074	kernel32.dll	0374	WaitForSingleObject
1	000FE078	kernel32.dll	0090	EnterCriticalSection
1	000FE07C	kernel32.dll	023A	LeaveCriticalSection
1	000FE080	kernel32.dll	025D	MulDiv
1	000FE084	kernel32.dll	0191	GetProcAddress
1	000FE088	kernel32.dll	0275	OutputDebugStringA
1	000FE08C	kernel32.dll	007D	DeleteFileA
1	000FE090	kernel32.dll	025E	MultiByteToWideChar
1	000FE094	kernel32.dll	01F6	GlobalUnlock
1	000FE098	kernel32.dll	01EF	GlobalLock
1	000FE09C	kernel32.dll	01E4	GlobalAlloc
1	000FE0A0	kernel32.dll	01EB	GlobalFree
1	000FE0A4	kernel32.dll	01DF	GetWindowsDirectoryA
1	000FE0A8	kernel32.dll	00C6	FindClose
1	000FE0AC	kernel32.dll	00D3	FindNextFileA
1	000FE0B0	kernel32.dll	00CA	FindFirstFileA
1	000FE0B4	kernel32.dll	016D	GetModuleFileNameA
1	000FE0B8	kernel32.dll	016F	GetModuleHandleA
1	000FE0BC	kernel32.dll	0162	GetLastError
1	000FE0C0	kernel32.dll	005B	CreateMutexA
1	000FE0C4	kernel32.dll	0103	GetCommandLineA
1	000FE0C8	kernel32.dll	00EA	FreeLibrary
1	000FE0CC	kernel32.dll	023B	LoadLibraryA
1	000FE0D0	kernel32.dll	032C	SetUnhandledExceptionFilter
1	000FE0D4	kernel32.dll	0135	GetCurrentProcess
1	000FE0D8	kernel32.dll	036C	VirtualQuery
1	000FE0DC	kernel32.dll	028B	QueryPerformanceCounter
1	000FE0E0	kernel32.dll	028C	QueryPerformanceFrequency



1	000FE0E4	kernel32.dll	01C7	GetThreadPriority
1	000FE0E8	kernel32.dll	020F	InitializeCriticalSection
1	000FE0EC	kernel32.dll	007B	DeleteCriticalSection
1	000FE0F0	kernel32.dll	01D5	GetVersionExA
1	000FE0F4	kernel32.dll	0221	IsBadWritePtr
1	000FE0F8	kernel32.dll	02FA	SetErrorMode
1	000FE0FC	kernel32.dll	0217	InterlockedIncrement
1	000FE100	kernel32.dll	02FB	SetEvent
1	000FE104	kernel32.dll	0157	GetFileTime
1	000FE108	kernel32.dll	004E	CreateFileA
1	000FE10C	kernel32.dll	0378	WideCharToMultiByte
1	000FE110	kernel32.dll	0165	GetLocaleInfoA
1	000FE114	kernel32.dll	02BE	RtlUnwind
1	000FE118	kernel32.dll	0290	RaiseException
1	000FE11C	kernel32.dll	00B0	ExitProcess
1	000FE120	kernel32.dll	00B1	ExitThread
1	000FE124	kernel32.dll	0348	TlsSetValue
1	000FE128	kernel32.dll	0347	TlsGetValue
1	000FE12C	kernel32.dll	02B9	ResumeThread
1	000FE130	kernel32.dll	006A	CreateThread
1	000FE134	kernel32.dll	0340	TerminateProcess
1	000FE138	kernel32.dll	01A6	GetStartupInfoA
1	000FE13C	kernel32.dll	0202	HeapFree
1	000FE140	kernel32.dll	021E	IsBadReadPtr
1	000FE144	kernel32.dll	01B7	GetSystemTimeAsFileTime
1	000FE148	kernel32.dll	0145	GetDriveTypeA
1	000FE14C	kernel32.dll	0133	GetCurrentDirectoryA
1	000FE150	kernel32.dll	015B	GetFullPathNameA
1	000FE154	kernel32.dll	0206	HeapReAlloc
1	000FE158	kernel32.dll	01FC	HeapAlloc
1	000FE15C	kernel32.dll	022D	LCMapStringA
1	000FE160	kernel32.dll	022E	LCMapStringW
1	000FE164	kernel32.dll	00F7	GetCPInfo
1	000FE168	kernel32.dll	01CC	GetTimeFormatA
1	000FE16C	kernel32.dll	0139	GetDateFormatA
1	000FE170	kernel32.dll	0036	CompareStringA
1	000FE174	kernel32.dll	0037	CompareStringW
1	000FE178	kernel32.dll	01A9	GetStringTypeA
1	000FE17C	kernel32.dll	01AC	GetStringTypeW
1	000FE180	kernel32.dll	0346	TlsFree
1	000FE184	kernel32.dll	02B8	RestoreLastError
1	000FE188	kernel32.dll	0345	TlsAlloc
1	000FE18C	kernel32.dll	0136	GetCurrentProcessId
1	000FE190	kernel32.dll	029D	ReadFile
1	000FE194	kernel32.dll	0385	WriteFile
1	000FE198	kernel32.dll	0208	HeapSize
1	000FE19C	kernel32.dll	0351	UnhandledExceptionFilter
1	000FE1A0	kernel32.dll	01A8	GetStdHandle
1	000FE1A4	kernel32.dll	00E8	FreeEnvironmentStringsA
1	000FE1A8	kernel32.dll	0147	GetEnvironmentStrings
1	000FE1AC	kernel32.dll	00E9	FreeEnvironmentStringsW
1	000FE1B0	kernel32.dll	0149	GetEnvironmentStringsW
1	000FE1B4	kernel32.dll	024E	LockResource
1	000FE1B8	kernel32.dll	0158	GetFileType
1	000FE1BC	kernel32.dll	0200	HeapDestroy
1	000FE1C0	kernel32.dll	01FE	HeapCreate
1	000FE1C4	kernel32.dll	0367	VirtualFree
1	000FE1C8	kernel32.dll	0364	VirtualAlloc
1	000FE1CC	kernel32.dll	00E0	FlushFileBuffers
1	000FE1D0	kernel32.dll	0300	SetFilePointer



1	000FE1D4	kernel32.dll	01CE	GetTimeZoneInformation
1	000FE1D8	kernel32.dll	036A	VirtualProtect
1	000FE1DC	kernel32.dll	01B2	GetSystemInfo
1	000FE1E0	kernel32.dll	01CF	GetUserDefaultLCID
1	000FE1E4	kernel32.dll	00A6	EnumSystemLocalesA
1	000FE1E8	kernel32.dll	022A	IsValidLocale
1	000FE1EC	kernel32.dll	0228	IsValidCodePage
1	000FE1F0	kernel32.dll	021B	IsBadCodePtr
1	000FE1F4	kernel32.dll	00F0	GetACP
1	000FE1F8	kernel32.dll	0184	GetOEMCP
1	000FE1FC	kernel32.dll	031B	SetStdHandle
1	000FE200	kernel32.dll	02F8	SetEnvironmentVariableA
1	000FE204	kernel32.dll	0166	GetLocaleInfoW
1	000FE208	kernel32.dll	02F7	SetEndOfFile
1	000FE20C	kernel32.dll	0138	GetCurrentThreadId
1	000FE210	kernel32.dll	0213	InterlockedDecrement
1	000FE214	kernel32.dll	0338	Sleep
1	000FE218	kernel32.dll	02AC	RemoveDirectoryA
FThunk: 000FE220 NbFunc: 00000002				
1	000FE220	oleaut32.dll	0006	SysFreeString
1	000FE224	oleaut32.dll	0002	SysAllocString
FThunk: 000FE22C NbFunc: 00000001				
1	000FE22C	shell32.dll	0167	ShellExecuteA
FThunk: 000FE234 NbFunc: 0000003B				
1	000FE234	user32.dll	009A	DestroyWindow
1	000FE238	user32.dll	0232	ScreenToClient
1	000FE23C	user32.dll	010C	GetCursorPos
1	000FE240	user32.dll	01B8	LoadCursorA
1	000FE244	user32.dll	024E	SetCursor
1	000FE248	user32.dll	01DD	MessageBoxA
1	000FE24C	user32.dll	0043	CloseClipboard
1	000FE250	user32.dll	024B	SetClipboardData
1	000FE254	user32.dll	01F4	OpenClipboard
1	000FE258	user32.dll	00C7	EndDialog
1	000FE25C	user32.dll	01EA	MoveWindow
1	000FE260	user32.dll	023C	SendMessageA
1	000FE264	user32.dll	0112	GetDlgItem
1	000FE268	user32.dll	009C	DialogBoxIndirectParamA
1	000FE26C	user32.dll	022A	ReleaseCapture
1	000FE270	user32.dll	0245	SetCapture
1	000FE274	user32.dll	02D6	WindowFromPoint
1	000FE278	user32.dll	00A2	DispatchMessageA
1	000FE27C	user32.dll	02AB	TranslateMessage
1	000FE280	user32.dll	01FE	PeekMessageA
1	000FE284	user32.dll	0257	SetFocus
1	000FE288	user32.dll	0102	GetClipboardData
1	000FE28C	user32.dll	0174	GetWindowPlacement
1	000FE290	user32.dll	027B	SetTimer
1	000FE294	user32.dll	0061	CreateWindowExA
1	000FE298	user32.dll	0002	AdjustWindowRect
1	000FE29C	user32.dll	0281	SetWindowLongA
1	000FE2A0	user32.dll	0258	SetForegroundWindow
1	000FE2A4	user32.dll	008F	DefWindowProcA
1	000FE2A8	user32.dll	00C9	EndPaint
1	000FE2AC	user32.dll	000E	BeginPaint
1	000FE2B0	user32.dll	016F	GetWindowLongA
1	000FE2B4	user32.dll	015E	GetSystemMetrics
1	000FE2B8	user32.dll	0050	CreateCursor
1	000FE2BC	user32.dll	0217	RegisterClassA
1	000FE2C0	user32.dll	01BC	LoadIconA



1	000FE2C4	user32.dll	0200	PostMessageA
1	000FE2C8	user32.dll	021B	RegisterClipboardFormatA
1	000FE2CC	user32.dll	0096	DestroyCursor
1	000FE2D0	user32.dll	0175	GetWindowRect
1	000FE2D4	user32.dll	013B	GetMessageA
1	000FE2D8	user32.dll	01A1	IsDialogMessage
1	000FE2DC	user32.dll	00BD	DrawTextA
1	000FE2E0	user32.dll	00E3	FillRect
1	000FE2E4	user32.dll	0117	GetFocus
1	000FE2E8	user32.dll	0178	GetWindowTextA
1	000FE2EC	user32.dll	015B	GetSysColor
1	000FE2F0	user32.dll	0287	SetWindowTextA
1	000FE2F4	user32.dll	015C	GetSysColorBrush
1	000FE2F8	user32.dll	010F	GetDesktopWindow
1	000FE2FC	user32.dll	022B	ReleaseDC
1	000FE300	user32.dll	0100	GetClientRect
1	000FE304	user32.dll	0041	ClientToScreen
1	000FE308	user32.dll	029A	SystemParametersInfoA
1	000FE30C	user32.dll	0293	ShowWindow
1	000FE310	user32.dll	01F3	OffsetRect
1	000FE314	user32.dll	00BE	DrawTextExA
1	000FE318	user32.dll	0045	CloseWindow
1	000FE31C	user32.dll	010D	GetDC
FThunk: 000FE324 NbFunc: 0000000A				
1	000FE324	winmm.dll	00A6	timeGetTime
1	000FE328	winmm.dll	00A3	timeEndPeriod
1	000FE32C	winmm.dll	00A2	timeBeginPeriod
1	000FE330	winmm.dll	0073	mixerClose
1	000FE334	winmm.dll	0074	mixerGetControlDetailsA
1	000FE338	winmm.dll	0079	mixerGetLineControlsA
1	000FE33C	winmm.dll	007B	mixerGetLineInfoA
1	000FE340	winmm.dll	0076	mixerGetDevCapsA
1	000FE344	winmm.dll	007F	mixerOpen
1	000FE348	winmm.dll	0080	mixerSetControlDetails
FThunk: 000FE350 NbFunc: 0000000E				
1	000FE350	wsock32.dll	0097	__WSAFDIsSet
1	000FE354	wsock32.dll	0012	select
1	000FE358	wsock32.dll	0074	WSACleanup
1	000FE35C	wsock32.dll	0003	closesocket
1	000FE360	wsock32.dll	0010	recv
1	000FE364	wsock32.dll	0013	send
1	000FE368	wsock32.dll	006F	WSAGetLastError
1	000FE36C	wsock32.dll	000A	inet_addr
1	000FE370	wsock32.dll	000C	ioctlsocket
1	000FE374	wsock32.dll	0034	gethostbyname
1	000FE378	wsock32.dll	0009	htons
1	000FE37C	wsock32.dll	0004	connect
1	000FE380	wsock32.dll	0017	socket
1	000FE384	wsock32.dll	0073	WSAStartup
FThunk: 000FE38C NbFunc: 00000002				
1	000FE38C	ole32.dll	0012	CoCreateInstance
1	000FE390	ole32.dll	003B	CoInitialize

Ok... phew. Well, open up IAT_dump.exe... run it... cross your fingers...

...and watch it crash...



3. Stolen Commands

Did you really think Safedisc would be that easy? Nope.

Ok, so, lets look in Olly where this beauty crashed:

Access Violation when executing [667A75D4]

O RLY!!?!?!?!?

Look up in EAX, you will see that we have 667A75D4 in it. This means that a JMP EAX probably caused the crash when it was trying to go to 667A75D4. Now, look at the stack:

0012F670	0127EA36	
0012F674	004E2B8A	RETURN to IAT_dump.004E2B8A from IAT_dump.004EA2F3
0012F678	004768C3	RETURN to IAT_dump.004768C3 from IAT_dump.00401DA9
0012F67C	77E7AD86	kernel32.GetModuleHandleA

Hmm, this looks interesting:

0012F678	004768C3	RETURN to IAT_dump.004768C3 from IAT_dump.00401DA9
----------	----------	--

Let go to 00401DA9, shall we?

00401DA9	/ \$ 51	PUSH ECX
00401DAA	. 50	PUSH EAX
00401DAB	. E8 63FFFFFF	CALL Zuma.00401D13

Hmm, notice all the calls going to this location:

Local calls from 0043C100, 0043D35B, 00440DC2, 0044C807, 0044FD3E, 004501C0, 00455881, 0045C6E9, 0045C73A, 0045C7AF, 0045C87A, 0045C955, 0045C99B, 0045C9EC, 0045CAAD, 0045CB6F, 0045CC30, 0045CC86, 0045CD93, 0045D432, 00464DBF, 00466C9C, 00466CD8, ...

That sure is a lot, eh?

Now, select 00401DAB and press enter to go into the call. Ohh mates, lookit wut I foun' :P

00401D13	\$ B8 8BF8FFFF	MOV EAX, -775
00401D18	. 59	POP ECX
00401D19	. 8D0408	LEA EAX, DWORD PTR DS:[EAX+ECX]
00401D1C	. 8B00	MOV EAX, DWORD PTR DS:[EAX]
00401D1E	. FFE0	JMP EAX

Heh, there is our wily little JMP EAX that is causing the crash . Lets see what it is doing. Open up a new Olly, and open up a new Zuma.exe process (leave the one with the correct imports running!). Get to the OEP in the new Zuma, and then assemble a JMP to 0043C100 (one of the offsets containing a CALL 00401DA9). You will end up here:

0043C100	E8 A45CFCFF	CALL Zuma.00401DA9
0043C105	72 14	JB SHORT Zuma.0043C11B
0043C107	68 E17A843F	PUSH 3F847AE1



Trace with F7 until you wind up here: (you will trace through a lot of interesting stuff)

66728140	83EC 30	SUB ESP,30
66728143	56	PUSH ESI
66728144	8BF1	MOV ESI,ECX
66728146	66:837E 7C 00	CMP WORD PTR DS:[ESI+7C],0
6672814B	74 0A	JE SHORT ~df394b.66728157
6672814D	8D46 60	LEA EAX,DWORD PTR DS:[ESI+60]
66728150	50	PUSH EAX
66728151	FF15 64207866	CALL DWORD PTR DS:[<&KERNEL32.EnterCritic>
66728157	8D4C24 04	LEA ECX,DWORD PTR SS:[ESP+4]
6672815B	E8 00FDFFFF	CALL ~df394b.66727E60
66728160	8B4C24 38	MOV ECX,DWORD PTR SS:[ESP+38]
66728164	51	PUSH ECX
66728165	8D4C24 08	LEA ECX,DWORD PTR SS:[ESP+8]
66728169	E8 82FEFFFF	CALL ~df394b.66727FF0
6672816E	66:8B16	MOV DX,WORD PTR DS:[ESI]
66728171	8D4C24 04	LEA ECX,DWORD PTR SS:[ESP+4]

Look at this call:

667281AA	E8 11000000	CALL ~df394b.667281C0
----------	-------------	-----------------------

It will lead us here:

667281C0	55	PUSH EBP
667281C1	8BEC	MOV EBP,ESP
667281C3	83EC 08	SUB ESP,8
667281C6	53	PUSH EBX
667281C7	56	PUSH ESI
667281C8	57	PUSH EDI
667281C9	894D F8	MOV DWORD PTR SS:[EBP-8],ECX
667281CC	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]
667281CF	83C0 20	ADD EAX,20
667281D2	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
667281D5	FF65 FC	JMP DWORD PTR SS:[EBP-4]
667281D8	5F	POP EDI
667281D9	5E	POP ESI
667281DA	5B	POP EBX
667281DB	8BE5	MOV ESP,EBP
667281DD	5D	POP EBP
667281DE	C3	RETN

This is a line of code we will take A LOT of interest in:

667281D5	FF65 FC	JMP DWORD PTR SS:[EBP-4]
----------	---------	--------------------------

Well, you can continue tracing to the JMP, and over it. You will see that it jumps to some code in the stack. This code restores the registers, and then jumps back to the exe. Once we are back in the exe, do you notice something?? Yes, the call changed to a CMP.

BEFORE:



0043C100	E8 A45CFCFF	CALL Zuma.00401DA9
0043C105	72 14	JB SHORT Zuma.0043C11B
0043C107	68 E17A843F	PUSH 3F847AE1

AFTER:

0043C100	3D 88130000	CMP EAX,1388
0043C105	72 14	JB SHORT Zuma.0043C11B
0043C107	68 E17A843F	PUSH 3F847AE1

So, this protection takes random commands from the game, and replaces them with CALLS. Once the CALL is executed, it emulates the command, and replaces the CALL with the original command, in this case a CMP. So, to defeat this protection, we must first identify it. Well, thats easy. All calls going to Zuma.00401DA9 are obviously part of this protection, so we just need to search for CALL Zuma.00401DA9. Then, we need to execute the call, and hook the program before it continues executing. The JMP DWORD PTR SS:[EBP-4] is a convenient place to 'hook'. Thats about all we need to code something to inject, right?

Ok guys, here is the code:



```
Title stolen_commands
.386
.model small, stdcall
Option casemap :none
.code

_IB          equ 00400000h    ; ImageBase
_text        equ 00001000h    ; Section infos
_textSize    equ 000FC90Ah    ; "
_ICROffset   equ 00401DA9h    ; offset with imagebase of where the
                                ; call reroutes point to.

_ICR:
mov eax, (_IB+_text)          ; beginning of .text (for scanning)

_ICRLoop:
cmp byte ptr[eax], 0E8h        ; is it an E8 (E8 call byte)?
je _ICRCheck                  ; well, check if its rerouted

_ICROk:
inc eax                        ; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize) ; is it the end of .text?
je _ICREnd                    ; if so, end
jmp _ICRLoop                  ; if not, continue

_ICRCheck:
mov ebx, dword ptr[eax+01]     ; gets the value behind the ICR opcode
add ebx, eax                   ; adds the value to the offset
add ebx, 5                     ; and add to ebx 5 so it contains the
                                ; offset of where the call is pointing to
cmp ebx, _ICROffset            ; is it a stolen command pointer?
jne _ICROk                     ; if it isn't, just jump to all is ok

; alright, remember to patch to a jmp _ICRJumpHere
mov esi, eax                   ; saves eax just in case it screws up
jmp eax                        ; jump to the internal call reroute
                                ; to resolve it

_ICRJumpHere:
mov esp, dword ptr[eax+1]      ; this is where we store registers
popad                          ; pops regs
popfd
pop esp
cmp eax, esi                   ; is eax still valid??
jne _ICRFixEAX                 ; if not, fix eax
jmp _ICROk                     ; continue, we have just resolved a
                                ; stolen command :)

_ICRFixEAX:
mov eax, esi                   ; fix eax
jmp _ICROk                     ; continue

_ICREnd:
int 3                          ; int 3 will cause the debugger to break,
                                ; and we will know that we are done.
```

All we need to do now to execute this code is inject it and replace the

667281D5	FF65 FC	JMP DWORD PTR SS:[EBP-4]
----------	---------	--------------------------



With a JMP to the "mov esp, dword ptr[eax+1]" in the code we will be injecting. Now, let the injected code run. If it gives you ANY type of error of access violation while running this code, just change EAX to 00401000. That will solve it.

Now, after this code and the code for fixing the imports is executed, just copy the imports from the temp IAT. to the real IAT. restore the original bytes at the OEP (you do this by selecting the modified bytes, and doing ALT+BACKSPACE, remember?), and then dump(call it ops_dump.exe), (remember to set correct EP!), and rebuild with Imprec.

On a side note: In newer versions of safedisc, the commands are not restored if you execute the call only once. Execute it 10 times, and the commands will be restored, so I recommend for newer versions of safedisc that you just make the above injected code loop 10 times, and then you're fine.

Well, let load the new dump in olly, execute it, and watch .. it crashes!

4. FF15 Annoyance

The crash occurs at this offset:

004722D9	. 8986 D4000000 MOV DWORD PTR DS:[ESI+D4],EAX
----------	---

Now, this problem took me a loooooooooong time to backtrace (an hour or so of banging my head on the wall). In the end, I found out it was caused by this line of code:

00471F7B	. FF15 B4E24F00 CALL DWORD PTR DS:[<user32.GetSystemMetrics>]
----------	---

To backtrace the problem, I did a 'trace over' from the beginning of this routine. I logged the trace from the original exe and our ops_dump.exe and compared in Ultra-Edit to see where the registers went wrong. Well, it went wrong in the line above. After we restored the imports, this was resolved as GetSystemMetrics. However, lets to a little experiment. Open up the original exe in Olly, and go to the

OEP. Now, at the OEP assemble a JMP 00471F7B, so that we go to this call when it is still redirected. Step into it with F7 until you get to the safedisc DLL(you know, the one with the 6677xxxx offset) Now, press CTRL+F9 to run until ret, and then look at the top of the stack:

0012FFBC	77D48EB0 USER32.RegisterWindowMessageA
----------	--

Ahah. So, it seems we have a problem here :(Our IAT fixer resolved it as user32.GetSystemMetrics, while it really should be USER32.RegisterWindowMessageA.

At this point, I went to bed that day and pondered over the problem. While I was in school, in my boring old geometry class, I started thinking of the problem again. Reversing is never far from my mind.

At the end, I thought of this:



There must be something influencing this calculation. But what could it be? What is the difference in between a call from our injected code and a call from .text? How can the protection differentiate the two? And then, I remembered:

In the stolen opcodes protection, how does it know where to overwrite the old opcodes with the news ones? Well, by looking at the return address of the call.

A call is essentially a:

```
PUSH EIP+5
JMP OFFSET_TO_WHERE_CALL_IS_SUPPOSED_TO_GO
```

So, the difference in between a call from our injected code and a call from .text is in the return address. Excellent, we have identified the problem, but now how to solve it?

Remember how I said in the beginning of the tutorial that we should save the IAT to a temporary area in STXT371 because we will need the original IAT to defeat later protections? Well, this is what I was talking about.

Ok, time to code something. We will need some code that will search .text for FF15 (the bytecode for a CALL DWORD PTR), check if it is calling a redirected call, and if it is we will jump to the call, and then wait for it to hit our hook. Then, once it hits our hook, we will retrieve the correct import from the top of the stack. Then, we will check if the CALL DWORD PTR is already pointing to the correct import a vast majority of them are). If not, we will search for the matching import in the IAT, and then fix the CALL DWORD PTR to point towards it.

Heh, I guess that is pretty hard to understand, so I will just translate it into something all of you can read, ASM:

```
Title ff_15
.386
.model small, stdcall
Option casemap :none
.code
_IB          equ 00400000h      ; ImageBase
_text        equ 00001000h      ; Section infos
_textSize    equ 000FC90Ah      ; "
_rdata       equ 000FE000h      ; Section infos
_rdataSize   equ 00000398h      ; "
_stxt371     equ 0018C000h      ; "
_stxt371Size equ 00000398h      ; same a .rdata size since we are
                                   ; using it as temp. iat
_lowerrange  equ 00E00000h      ; lower range of all redirected calls
_higherrange equ 00F00000h      ; high range of all redirected calls

start:

_FF15Fix:
mov  eax, (_IB+_text)           ; beginning of .text

_FF15loop:
cmp  word ptr[eax],15FFh        ; is it a call?
je   _FF15Check

_FF15Ok:
```



```
inc eax ; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize) ; is it the end of .text?
je _FF15End ; if so, end
jmp _FF15loop ; if not, continue

_FF15Check:
cmp dword ptr[eax+2], (_IB+_rdata) ; is it in the zone of all redirected calls?
jl _FF15Ok ; if it is below, exit checking
; is it in the zone of all redirected calls?
cmp dword ptr[eax+2], (_IB+_rdata+_rdataSize)
ja _FF15Ok ; if it is above exit checking
mov ebx, dword ptr[eax+2] ; puts the offset of the import in
; .rdata into ebx

cmp dword ptr[ebx], _lowerrange ; is it a redirected call?
jl _FF15Ok ; if it is less than the offset
; of a normally redirected call, then
; exit checking

cmp dword ptr[ebx], _higherrange ; is it a redirected call?
ja _FF15Ok ; if it is above the offset of a
; normally redirected call, exit checking

_FF15Resolve:
add eax, 6 ; prepares to push the fake return point
; of call
; remember to nop the "twin bitch" jmps
; in the calc routine
; also put a jump _FF15JumpHere in the
; pop edx routine
push eax ; pushes this "hidden parameter"
sub eax, 6 ; restores eax to the way it was before
jmp dword ptr[ebx] ; jumps to call to resolve it

_FF15JumpHere:
pop ebx ; trash the fake return value
mov ebx, (_IB+_stxt371)

_FF15TextFixSearch:
cmp dword ptr[ebx], edx ; is ebx equal to the recovered import??
je _FF15TextFix ; if so, fix import
add ebx, 4 ; if not, continue search
jmp _FF15TextFixSearch

_FF15TextFix:
sub ebx, (_IB+_stxt371)
add ebx, (_IB+_rdata)
mov dword ptr[eax+2], ebx ; fix the import :)
jmp _FF15Ok ; and move on to the next import. yay,
; we have a resolved call :D

_FF15End:
int 3 ; int 3 will cause the debugger to
; break, and we will know that we are done.

end start
```

Ok now, assemble this code in MASM and inject it after the code for fixing the IAT and the stolen commands. Also, before you can execute this code, there is still one little thing you need to do. Remember the code we injected into the safedisc dll?



6677FD60	8B65 0C	MOV ESP,DWORD PTR SS:[EBP+C]
6677FD63	61	POPAD
6677FD64	9D	POPFD
6677FD65	5A	POP EDX
6677FD66	C3	RETN

Right before you execute the code to fix all the CALL DWORD PTR's, change the RET in the above code to a JMP that goes to the 'pop ebx' of the above code that is to be injected. Now you are ready to run it. Execute it, then restore the bytes at the OEP, copy the IAT over, dump, rebuild with imprec blablabla you know the drill. I saved the exe as ff15_dump.exe

Now take the new executable, and run it, and watch it... go to the black screen... you start hoping... and then... *crash*

5. Long Jumps Problem

Do not despair. This is the last protection.

The error is access violation when executing 02B60982 blablabla...

Well, lets look at the stack to get some idea of what is happening. You will see a RETURN to 00455CB3. Lets go to 00455CB3.

00455CAE	. E8 5DB4FEFF	CALL ff15_dum.00441110
00455CB3	. 8B06	MOV EAX,DWORD PTR DS:[ESI]

Press enter to go into the call:

00441110	\$ 55	PUSH EBP
00441111	. 8BEC	MOV EBP,ESP
00441113	. 83EC 20	SUB ESP,20
00441116	. 56	PUSH ESI
00441117	. 57	PUSH EDI
00441118	. 8BF1	MOV ESI,ECX
0044111A	. 8B4E 04	MOV ECX,DWORD PTR DS:[ESI+4]
0044111D	. 8D45 E0	LEA EAX,DWORD PTR SS:[EBP-20]
00441120	. 50	PUSH EAX
00441121	. 51	PUSH ECX
00441122	.-E9 24931400	JMP ff15_dum.0058A44B
00441127	29	DB 29
00441128	. 8B45 E4	MOV EAX,DWORD PTR SS:[EBP-1C]
0044112B	. 8B4D E8	MOV ECX,DWORD PTR SS:[EBP-18]
0044112E	. 8B55 E0	MOV EDX,DWORD PTR SS:[EBP-20]
00441131	. 8B3D 04E34F00	MOV EDI,DWORD PTR DS:[<user32.ClientToS>

You see that longjump at 00441122? It's jumping into STXT 774, and right after it is a junk byte. Very suspicious, considering this is abnormal code and is jumping into a Safedisc section.

Lets open up the original exe, get to the OEP, and then jump to 00441122. Press F7 to execute the jump. You will end up here:



0058A44B	53	PUSH EBX
0058A44C	E8 27F0FFFF	CALL Zuma.00589478

Trace with F7 into the call...

00589478	870424	XCHG DWORD PTR SS:[ESP],EAX
0058947B	9C	PUSHFD
0058947C	05 10FBFFFF	ADD EAX,-4F0
00589481	8B18	MOV EBX,DWORD PTR DS:[EAX]
00589483	^EB 91	JMP SHORT Zuma.00589416

Weeeeeee obfuscation! Not that it's a problem. Keep on tracing with F7, over the JMP.

00589416	6BDB 01	IMUL EBX,EBX,1
00589419	0358 04	ADD EBX,DWORD PTR DS:[EAX+4]
0058941C	9D	POPFD
0058941D	58	POP EAX
0058941E	871C24	XCHG DWORD PTR SS:[ESP],EBX
00589421	C3	RETN

Hmm, the ret looks interesting, lets trace over it, shall we??

02B60982	68 28114400	PUSH 441128
02B60987	68 9D12EABF	PUSH BFEA129D
02B6098C	9C	PUSHFD
02B6098D	60	PUSHAD
02B6098E	54	PUSH ESP
02B6098F	68 C209B602	PUSH 2B609C2
02B60994	E8 43EAC163	CALL ~df394b.6677F3DC
02B60999	83C4 08	ADD ESP,8
02B6099C	6A 00	PUSH 0
02B6099E	58	POP EAX
02B6099F	61	POPAD
02B609A0	9D	POPFD
02B609A1	C3	RETN

Heh, it's going to the Safedisc DLL. F7 to trace into the safedisc DLL, and you will be greeted with an all too familiar sight:

6677F3DC	55	PUSH EBP
6677F3DD	8BEC	MOV EBP,ESP
6677F3DF	83EC 34	SUB ESP,34
6677F3E2	53	PUSH EBX
6677F3E3	56	PUSH ESI

Well, looks like this is some type of weird import redirection we are facing. Press CTRL+F9 to execute until return, and then look at the top of the stack:

0012FFBC	77D445A7	USER32.GetClientRect
----------	----------	----------------------

Ahh, well there we go, we retrieved this import. Now, out of curiosities sake, lets keep on tracing. If you trace over the RET you will end up at the import:



```
77D445A7 > 8B4C24 04      MOV ECX,DWORD PTR SS:[ESP+4]
77D445AB    E8 5CF4FFFF      CALL USER32.77D43A0C
77D445B0    85C0              TEST EAX,EAX
77D445B2    74 0D             JE SHORT USER32.77D445C1
77D445B4    FF7424 08         PUSH DWORD PTR SS:[ESP+8]
77D445B8    50               PUSH EAX
77D445B9    E8 9CFFFFFF      CALL USER32.77D4455A
77D445BE    33C0             XOR EAX,EAX
77D445C0    40              INC EAX
77D445C1    C2 0800          RETN 8
```

Now, trace with F8 until you trace over the RET. You will end up here:

```
00441121    . 51             PUSH ECX
00441122    .-E9 24931400     JMP Zuma.0058A44B
00441127    29              DB 29
00441128    . 8B45 E4         MOV EAX,DWORD PTR SS:[EBP-1C]    <- You end up here
0044112B    . 8B4D E8         MOV ECX,DWORD PTR SS:[EBP-18]
```

From here on, its not too hard to understand what is happening here. Safedisc replaced some CALL DWORD PTR's with LONGJUMPs, thats all. You might wonder how they can do that if a CALL DWORD PTR is 6 bytes long, and a LONGJUMP is 5 bytes long. Well, there is a junk byte back there for a reason, and thats it. That junk byte is simple ignored when the program is executed.

So, now, as always, it is time to code something to inject to defeat this protection.

Here is what the code should do:

It should scan .text for all the longjumps, and see which are jumping to STXT 774. If they are jumping to STXT 774, it will jump to the longjump and wait for it to reach the hook we place. Once it reaches the hook, we just replace the longjump and the junk byte with a CALL DWORD PTR pointing to the correct import. Very simple.

Here is the code:

```
Title longjumps
.386
.model small, stdcall
Option casemap :none
.code

_IB          equ 00400000h          ; ImageBase
_text        equ 00001000h          ; Section infos
_textSize    equ 000FC90Ah          ; "
_rdata       equ 000FE000h          ; Section infos
_rdataSize   equ 00000398h          ; "
_stxt371     equ 0018C000h          ; "
_stxt371Size equ 00000398h          ; same a .rdata size since we are using
                                         ; it as temp. iat
_lowerrange  equ 00E00000h          ; lower range of all redirected calls
_higherrange equ 00F00000h          ; high range of all redirected calls
_stxt774     equ 00189000h          ; "
_stxt774Size equ 00002059h          ; "

start:

_LongJumps:
```



```
mov eax, (_IB+_text) ; beginning of .text

_LJLoop:
cmp byte ptr[eax], 0E9h ; is this byte the beginning byte
; of a longjump??
je _LJCheck1 ; if so, jump to checks

_LJOk:
inc eax ; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize) ; is it the end of .text?
je _LJEnd ; if so, end
jmp _LJLoop ; if not, continue

_LJCheck1:
mov ebx, dword ptr[eax+01] ; gets the value behind the LJ opcode
add ebx, eax ; adds the value to the offset
add ebx, 5 ; and add to ebx 5 so it contains the
; offset of where the longjump is
; pointing to
cmp ebx, (_IB+_stxt774) ; is it in STXT 774??
jl _LJOk ; if its less than the range, its all good
;)
cmp ebx, (_IB+_stxt774+_stxt774Size) ; is it in STXT 774??
ja _LJOk ; if its above the range, its all good ;)
jmp ebx ; alright, jump to this bad guy to
; resolve it ;)
; ok, remember to change the end of the
; pop edx routine to jmp _LJJumpHere

_LJJumpHere:
; return here from pop edx routine
pop ecx ; makes this place ez to identify
mov ecx, (_IB+_stxt371) ; beginning of STXT 371, which is used
; as temporary IAT

_LJFixSearch:
cmp dword ptr[ecx], edx ; ehh, is this import what we are
; looking for??
je _LJTextFix ; if so, lets fix up the LJ ;)
add ecx, 4 ; move the pointer up
jmp _LJFixSearch ; and lets continue the loop

_LJTextFix:
sub ecx, (_IB+_stxt371) ; subtracts the offset of the temp, iat
add ecx, (_IB+_rdata) ; and adds .rdata so that this offset points
; to .rdata
mov word ptr[eax], 15FFh ; adds the customary FF15, which are the
; opcodes for the FF15 call
mov dword ptr[eax+02], ecx ; fix the value behind the FF15 ;)
jmp _LJOk ; oo, ahh, we have a resolved call :DDD

_LJEnd:
int 3 ; int 3 will cause the debugger to break,
; and we will know that we are done.

end start
```

You know the drill :) Assemble and inject this code with the other code. Also, remember the pop edx routine we inserted into the code cave in the safedisc dll? Well, change it so that it jumps to the pop ecx in the above injected code right after it pops into edx the correct import.



Thats it, let the injected code run, dump (I won't describe again how to dump. If you don't understand it by now... just look up).

Here is a list of all the patches made to the exe so far:



Patches Address Comment	Size	State	Old	New
0043C100	5.	Active	CALL Zuma.00401DA9	CMP EAX,1388
0043C73B	6.	Removed	???	MOV DWORD PTR DS:[EAX],FF000000
0043D35B	5.	Active	CALL Zuma.00401DA9	SUB EAX,186A0
00440DC2	5.	Active	CALL Zuma.00401DA9	MOV EAX,88760091
00441122	6.	Active	JMP Zuma.0058A44B	CALL DWORD PTR DS:[4FE300]
0044BD0D	16.	Active	CALL DWORD PTR DS:[4FE024]	CALL DWORD PTR DS:[4FE034]
0044C79E	6.	Active	CALL DWORD PTR DS:[4FE234]	CALL DWORD PTR DS:[4FE304]
0044C807	5.	Active	CALL Zuma.00401DA9	MOV ECX,19
0044FD3E	5.	Active	CALL Zuma.00401DA9	MOV ECX,19
0045015A	6.	Active	CALL DWORD PTR DS:[4FE2AC]	CALL DWORD PTR DS:[4FE304]
004501C0	5.	Active	CALL Zuma.00401DA9	MOV ECX,19
004509A0	6.	Active	CALL DWORD PTR DS:[4FE028]	CALL DWORD PTR DS:[4FE03C]
004509FA	6.	Active	CALL DWORD PTR DS:[4FE038]	CALL DWORD PTR DS:[4FE01C]
00450A2D	6.	Active	CALL DWORD PTR DS:[4FE2E8]	CALL DWORD PTR DS:[4FE2FC]
00450A65	6.	Active	CALL DWORD PTR DS:[4FE2FC]	CALL DWORD PTR DS:[4FE31C]
00450AAC	6.	Active	CALL DWORD PTR DS:[4FE238]	CALL DWORD PTR DS:[4FE314]
00450AC4	6.	Active	JMP Zuma.00589A14	CALL DWORD PTR DS:[4FE2FC]
00450B95	6.	Active	CALL DWORD PTR DS:[4FE030]	CALL DWORD PTR DS:[4FE028]
00450BD1	6.	Active	CALL DWORD PTR DS:[4FE04C]	CALL DWORD PTR DS:[4FE024]
00455867	6.	Active	CALL DWORD PTR DS:[4FE268]	CALL DWORD PTR DS:[4FE304]
00455881	5.	Active	CALL Zuma.00401DA9	MOV ECX,19
0045C664	6.	Active	CALL DWORD PTR DS:[4FE29C]	CALL DWORD PTR DS:[4FE300]
0045C6E9	5.	Active	CALL Zuma.00401DA9	MOV EAX,2
0045C73A	5.	Active	CALL Zuma.00401DA9	MOV EAX,2
0045C7AF	5.	Active	CALL Zuma.00401DA9	MOV EAX,2
0045C87A	5.	Active	CALL Zuma.00401DA9	MOV EAX,6
0045C955	5.	Active	CALL Zuma.00401DA9	MOV EAX,1
0045C99B	5.	Active	CALL Zuma.00401DA9	MOV EAX,4
0045C9EC	5.	Active	CALL Zuma.00401DA9	MOV EAX,5
0045CAAD	5.	Active	CALL Zuma.00401DA9	MOV EAX,3
0045CB6F	5.	Active	CALL Zuma.00401DA9	MOV EAX,1
0045CC30	5.	Active	CALL Zuma.00401DA9	MOV EAX,3
0045CC86	5.	Active	CALL Zuma.00401DA9	MOV EAX,3
0045CD93	5.	Active	CALL Zuma.00401DA9	MOV EAX,6
0045D432	5.	Active	CALL Zuma.00401DA9	MOV EAX,7
00464DBF	5.	Active	CALL Zuma.00401DA9	MOV EAX,10624DD3
0046500F	6.	Active	CALL DWORD PTR DS:[4FE31C]	CALL DWORD PTR DS:[4FE244]
00465067	6.	Active	CALL DWORD PTR DS:[4FE2A4]	CALL DWORD PTR DS:[4FE244]
004661AD	6.	Active	JMP Zuma.0058AE1B	CALL DWORD PTR DS:[4FE248]
004661CB	6.	Active	CALL DWORD PTR DS:[4FE234]	CALL DWORD PTR DS:[4FE254]
0046629D	6.	Active	CALL DWORD PTR DS:[4FE2B4]	CALL DWORD PTR DS:[4FE250]
00466AD4	6.	Active	CALL DWORD PTR DS:[4FE250]	CALL DWORD PTR DS:[4FE31C]
00466B07	6.	Active	CALL DWORD PTR DS:[4FE2F0]	CALL DWORD PTR DS:[4FE300]
00466B2A	6.	Active	JMP Zuma.0058A0EC	CALL DWORD PTR DS:[4FE25C]
00466BE9	6.	Active	CALL DWORD PTR DS:[4FE020]	CALL DWORD PTR DS:[4FE044]
00466C8E	6.	Active	CALL DWORD PTR DS:[4FE2DC]	CALL DWORD PTR DS:[4FE258]
00466C9C	5.	Active	CALL Zuma.00401DA9	MOV EAX,1
00466CD8	5.	Active	CALL Zuma.00401DA9	MOV ECX,0A
00466DB0	6.	Active	CALL DWORD PTR DS:[4FE2D8]	CALL DWORD PTR DS:[4FE268]
00468246	6.	Active	CALL Zuma.00401DA9	CMP ECX,1F4
004682F9	6.	Active	CALL DWORD PTR DS:[4FE2E8]	CALL DWORD PTR DS:[4FE274]
004684EB	6.	Active	CALL DWORD PTR DS:[4FE2B0]	CALL DWORD PTR DS:[4FE270]
0046850D	5.	Active	CALL DWORD PTR DS:[4FE314]	CALL DWORD PTR DS:[4FE270]
004687E8	6.	Active	CALL Zuma.00401DA9	CMP EAX,0FA
0046889B	6.	Active	CALL DWORD PTR DS:[4FE260]	CALL DWORD PTR DS:[4FE30C]
0046B18D	6.	Active	CALL DWORD PTR DS:[4FE010]	CALL DWORD PTR DS:[4FE00C]
0046B25F	6.	Active	CALL DWORD PTR DS:[4FE004]	CALL DWORD PTR DS:[4FE000]
0046BCBF	6.	Active	CALL DWORD PTR DS:[4FE00C]	CALL DWORD PTR DS:[4FE010]
0046BF02	6.	Active	CALL DWORD PTR DS:[4FE004]	CALL DWORD PTR DS:[4FE00C]
0046BF2E	6.	Active	CALL DWORD PTR DS:[4FE00C]	CALL DWORD PTR DS:[4FE004]
0046BFBA	5.	Active	CALL Zuma.00401DA9	MOV EDI,10
0046CDD3	5.	Active	CALL Zuma.00401DA9	MOV EAX,10624DD3
0046CDE0	12.	Active	CALL Zuma.00401DA9	MOV ECX,3C
0046E905	6.	Active	CALL DWORD PTR DS:[4FE2D8]	CALL DWORD PTR DS:[4FE248]
0046FAFD	6.	Active	CALL DWORD PTR DS:[4FE2BC]	CALL DWORD PTR DS:[4FE234]
0046FB32	6.	Active	CALL DWORD PTR DS:[4FE308]	CALL DWORD PTR DS:[4FE298]
0046FF5C	6.	Active	CALL DWORD PTR DS:[4FE234]	CALL DWORD PTR DS:[4FE30C]
0047062D	6.	Active	CALL DWORD PTR DS:[4FE2CC]	CALL DWORD PTR DS:[4FE2B0]
004706E9	6.	Active	CALL DWORD PTR DS:[4FE250]	CALL DWORD PTR DS:[4FE2AC]
00470707	6.	Active	CALL DWORD PTR DS:[4FE27C]	CALL DWORD PTR DS:[4FE2A8]
004709B0	6.	Active	CALL DWORD PTR DS:[4FE2E0]	CALL DWORD PTR DS:[4FE28C]
00471F7B	6.	Active	CALL DWORD PTR DS:[4FE2B4]	CALL DWORD PTR DS:[4FE2C8]
00471FCD	5.	Active	CALL Zuma.00401DA9	CMP EAX,0B7



Manual Unpacking Safecast

0047243E	6.	Active	CALL DWORD PTR DS:[4FE31C]	CALL DWORD PTR DS:[4FE294]
004725F6	6.	Active	CALL DWORD PTR DS:[4FE250]	CALL DWORD PTR DS:[4FE248]
004748D2	6.	Active	CALL DWORD PTR DS:[4FE28C]	CALL DWORD PTR DS:[4FE248]
00474AAD	6.	Active	CALL DWORD PTR DS:[4FE298]	CALL DWORD PTR DS:[4FE248]
00475DB5	6.	Active	JMP Zuma.0058ABF4	CALL DWORD PTR DS:[4FE248]
00475EDD	13.	Active	CALL DWORD PTR DS:[4FE28C]	CALL DWORD PTR DS:[4FE29C]
004768BE	5.	Active	CALL Zuma.00401DA9	MOV EAX,Zuma.005054DC
00476B10	5.	Active	CALL Zuma.00401DA9	MOV ECX,6
00476B98	5.	Active	CALL Zuma.00401DA9	CMP EAX,100
004814C4	5.	Active	CALL Zuma.00401DA9	CALL Zuma.004F4EA6
004814D9	5.	Active	CALL Zuma.00401DA9	CALL Zuma.004F4EA0
004814ED	5.	Active	CALL Zuma.00401DA9	CALL Zuma.004F4EA0
00481501	5.	Active	CALL Zuma.00401DA9	CALL Zuma.004F4EA0
00481691	6.	Active	CALL DWORD PTR DS:[4FE2D4]	CALL DWORD PTR DS:[4FE2D0]
004816DC	6.	Active	CALL DWORD PTR DS:[4FE2E4]	CALL DWORD PTR DS:[4FE25C]
00481733	16.	Active	CALL DWORD PTR DS:[4FE244]	CALL DWORD PTR DS:[4FE27C]
004817AF	6.	Active	JMP Zuma.0058A653	CALL DWORD PTR DS:[4FE284]
004817DD	6.	Active	CALL DWORD PTR DS:[4FE2AC]	CALL DWORD PTR DS:[4FE300]
004817E8	6.	Active	CALL DWORD PTR DS:[4FE284]	CALL DWORD PTR DS:[4FE2AC]
00481807	6.	Active	CALL DWORD PTR DS:[4FE044]	CALL DWORD PTR DS:[4FE048]
00481838	6.	Active	CALL DWORD PTR DS:[4FE048]	CALL DWORD PTR DS:[4FE024]
0048185D	6.	Active	CALL DWORD PTR DS:[4FE040]	CALL DWORD PTR DS:[4FE024]
00481872	6.	Active	CALL DWORD PTR DS:[4FE048]	CALL DWORD PTR DS:[4FE040]
004818B7	6.	Active	CALL DWORD PTR DS:[4FE040]	CALL DWORD PTR DS:[4FE028]
004818EA	6.	Active	CALL DWORD PTR DS:[4FE30C]	CALL DWORD PTR DS:[4FE2DC]
004818FF	6.	Active	CALL DWORD PTR DS:[4FE2C4]	CALL DWORD PTR DS:[4FE2A8]
00481913	6.	Active	JMP Zuma.00589041	CALL DWORD PTR DS:[4FE2A4]
00482891	6.	Active	JMP Zuma.0058AE55	CALL DWORD PTR DS:[4FE390]
004828DF	6.	Active	CALL DWORD PTR DS:[4FE300]	CALL DWORD PTR DS:[4FE298]
0048990D	6.	Active	CALL DWORD PTR DS:[4FE010]	CALL DWORD PTR DS:[4FE00C]
00489A3D	6.	Active	CALL DWORD PTR DS:[4FE250]	CALL DWORD PTR DS:[4FE2EC]
00489A89	6.	Active	JMP Zuma.00589B00	CALL DWORD PTR DS:[4FE2BC]
00489AAD	6.	Active	CALL DWORD PTR DS:[4FE2D0]	CALL DWORD PTR DS:[4FE298]
00489AF0	6.	Active	JMP Zuma.0058981C	CALL DWORD PTR DS:[4FE29C]
00489C57	6.	Active	JMP Zuma.0058AB9D	CALL DWORD PTR DS:[4FE30C]
00489CE7	6.	Active	CALL DWORD PTR DS:[4FE308]	CALL DWORD PTR DS:[4FE234]
00490679	6.	Active	CALL DWORD PTR DS:[4FE29C]	CALL DWORD PTR DS:[4FE2A4]
00493A98	6.	Active	JMP Zuma.005891BF	CALL DWORD PTR DS:[4FE29C]
00493B40	6.	Active	CALL DWORD PTR DS:[4FE304]	CALL DWORD PTR DS:[4FE30C]
00493DA0	6.	Active	CALL DWORD PTR DS:[4FE274]	CALL DWORD PTR DS:[4FE234]
0049426C	6.	Active	CALL DWORD PTR DS:[4FE25C]	CALL DWORD PTR DS:[4FE248]
0049456A	6.	Active	CALL DWORD PTR DS:[4FE258]	CALL DWORD PTR DS:[4FE248]
0049700B	6.	Active	CALL DWORD PTR DS:[4FE2AC]	CALL DWORD PTR DS:[4FE2A4]
00497074	6.	Active	CALL DWORD PTR DS:[4FE294]	CALL DWORD PTR DS:[4FE2C0]
0049709A	6.	Active	CALL DWORD PTR DS:[4FE23C]	CALL DWORD PTR DS:[4FE2BC]
004970BE	6.	Active	CALL DWORD PTR DS:[4FE2A8]	CALL DWORD PTR DS:[4FE298]
004971A0	6.	Active	CALL DWORD PTR DS:[4FE2DC]	CALL DWORD PTR DS:[4FE30C]
00497288	6.	Active	CALL DWORD PTR DS:[4FE27C]	CALL DWORD PTR DS:[4FE2A4]
004972F4	6.	Active	CALL DWORD PTR DS:[4FE270]	CALL DWORD PTR DS:[4FE2C0]
0049731A	6.	Active	CALL DWORD PTR DS:[4FE304]	CALL DWORD PTR DS:[4FE2BC]
0049733E	6.	Active	CALL DWORD PTR DS:[4FE284]	CALL DWORD PTR DS:[4FE298]
004973A6	6.	Active	CALL DWORD PTR DS:[4FE2D0]	CALL DWORD PTR DS:[4FE31C]
004973C4	6.	Active	CALL DWORD PTR DS:[4FE23C]	CALL DWORD PTR DS:[4FE2FC]
00497406	6.	Active	JMP Zuma.00589DB6	CALL DWORD PTR DS:[4FE260]
00497441	6.	Active	CALL DWORD PTR DS:[4FE278]	CALL DWORD PTR DS:[4FE31C]
00497464	6.	Active	JMP Zuma.0058974B	CALL DWORD PTR DS:[4FE2FC]
004974B6	6.	Active	CALL DWORD PTR DS:[4FE2C8]	CALL DWORD PTR DS:[4FE284]
00498181	6.	Active	CALL DWORD PTR DS:[4FE2F0]	CALL DWORD PTR DS:[4FE234]
004982B9	6.	Active	CALL DWORD PTR DS:[4FE2B8]	CALL DWORD PTR DS:[4FE2A4]
004982DC	6.	Active	CALL DWORD PTR DS:[4FE258]	CALL DWORD PTR DS:[4FE2A4]
004983AB	6.	Active	CALL DWORD PTR DS:[4FE270]	CALL DWORD PTR DS:[4FE244]
00498466	6.	Active	CALL DWORD PTR DS:[4FE314]	CALL DWORD PTR DS:[4FE2C0]
004984B1	6.	Active	JMP Zuma.0058AF96	CALL DWORD PTR DS:[4FE298]
00498628	6.	Active	CALL DWORD PTR DS:[4FE25C]	CALL DWORD PTR DS:[4FE260]
00498678	6.	Active	CALL DWORD PTR DS:[4FE270]	CALL DWORD PTR DS:[4FE30C]
0049869D	6.	Active	CALL DWORD PTR DS:[4FE274]	CALL DWORD PTR DS:[4FE27C]
00498A6C	6.	Active	CALL DWORD PTR DS:[4FE048]	CALL DWORD PTR DS:[4FE01C]
0049B5C3	6.	Active	JMP Zuma.0058ABD3	CALL DWORD PTR DS:[4FE248]
0049B9CF	6.	Active	CALL DWORD PTR DS:[4FE260]	CALL DWORD PTR DS:[4FE248]
0049BE53	6.	Active	CALL DWORD PTR DS:[4FE2EC]	CALL DWORD PTR DS:[4FE248]
004E2948	7.	Removed	PUSH 60	JMP Zuma.00400500
667281D5	5.	Active	JMP DWORD PTR SS:[EBP-4]	JMP Zuma.0040062A
6677F548	5.	Active	MOV ESP,DWORD PTR SS:[EBP+C]	JMP ~df394b.6677FD60
6677F6CD	5.	Active	MOV ESP,DWORD PTR SS:[EBP+C]	JMP ~df394b.6677FD60
6677FD60	12.	Active	ADD BYTE PTR DS:[EAX],AL	MOV ESP,DWORD PTR SS:[EBP+C]



It should help you track down any problems

Now, run the game. It should work. If it doesn't, you should be able to find the problem using the list of patches above, and the reconstructed IAT I posted at the beginning of this tut.

Thank you for reading. It's 11 pm here, and I have a big math test tomorrows, so I gotta go! Cy'all later.

6. Overall Code into one piece

Before I forget, here is the code I ended up using in the end, properly formatted and all that.

```
Title imports
.386 .model small, stdcall
Option casemap :none
.code

;this stuff underneath is exe data. modify it from exe to exe
_IB          equ 00400000h      ; ImageBase
_text        equ 00001000h      ; Section infos
_textSize    equ 000FC90Ah      ; "
_rdata       equ 000FE000h      ; "
_rdataSize   equ 00000398h      ; "
_stxt774     equ 00189000h      ; "
_stxt774Size equ 00002059h      ; "
_stxt371     equ 0018C000h      ; "
_stxt371Size equ 00000398h      ; same a .rdata size since we are using it
as temp. iat
_lowerrange  equ 00E00000h      ; lower range of all redirected calls
_higherrange equ 00F00000h      ; high range of all redirected calls
_ICROffset   equ 00401DA9h      ; offset with imagebase of where the
                                ; internal call reroutes point to.

start:
pushad
pushfd
mov eax, (_IB+_rdata)           ; beginning of IAT
mov ebx, (_IB+_stxt371)         ; beginning of STXT 371, which is
                                ; used as temporary IAT

_iloop:
mov ecx, dword ptr[ebx]         ; retrieve adress of call
cmp ecx, _lowerrange            ; is it below the lower range of the
                                ; redirected calls?
jl _invalidcall                 ; if so, jump to the call is valid
cmp ecx, _higherrange          ; is it above the zone of redirected
                                ; calls?
ja _invalidcall                 ; if so, jumo to call is valid

_iresolve:
call ecx                        ; will call ecx so that it retrieves the
                                ; correct value in edx
mov dword ptr [ebx], edx        ; put correct value into temporary IAT
add eax, 4                      ; move pointer to next call in IAT
add ebx, 4                      ; move pointer to next space in temporary
                                ; IAT
cmp eax, (_IB+_rdata+_rdataSize) ; are we at the end of IAT=1000(zone of
```



```
; all redirected calls)
je _iend
; if so, let us end
jmp _iloop
; if not, continue program
_ivalidcall:
mov dword ptr [ebx], ecx
; call is not redirect, so just copy it
; into temporary IAT
add eax, 4
; move pointer to next call in IAT
add ebx, 4
; move pointer to next space in temporary
IAT
cmp eax, (_IB+_rdata+_rdataSize)
; are we at the end of IAT=1000(zone of
; all redirected calls)
je _iend
; if so, let us end
jmp _iloop
; if not, continue program
_iend:
int 3
; int 3 will cause the debugger to break,
; and we will know that we are done.
jmp _FF15Fix
; jump to next fix

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_FF15Fix:
mov eax, (_IB+_text)
; beginning of .text
_FF15loop:
cmp word ptr [eax], 15FFh
; is it a call?
je _FF15Check

_FF15Ok:
inc eax
; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize)
; is it the end of .text?
je _FF15End
; if so, end
jmp _FF15loop
; if not, continue

_FF15Check:
cmp dword ptr [eax+2], (_IB+_rdata)
; is it in the zone of all redirected
calls?
jl _FF15Ok
; if it is below, exit checking
cmp dword ptr [eax+2], (_IB+_rdata+_rdataSize)
; is it in the zone of all
redirected calls?
ja _FF15Ok
; if it is above exit checking
mov ebx, dword ptr [eax+2]
; puts the offset of the import in .rdata
; into ebx
cmp dword ptr [ebx], _lowerrange
; is it a redirected call?
jl _FF15Ok
; if it is less than the offset of a
; normally redirected call, then exit
checking
cmp dword ptr [ebx], _higherrange
; is it a redirected call?
ja _FF15Ok
; if it is above the offset of a normally
; redirected call, exit checking

_FF15Resolve:
add eax, 6
; prepares to push the fake return point
; of call
; remember to nop the "twin bitch" jmps in
; the calc routine
; also put a jump _FF15JumpHere in the pop
; edx routine
push eax
; pushes this "hidden parameter"
```



```
sub eax, 6 ; restores eax to the way it was before
jmp dword ptr[ebx] ; jumps to call to resolve it

_FF15JumpHere:
pop ebx ; trash the fake return value
mov ebx, (_IB+_stxt371) ; put in ebx pointer to temp IAT
; beginning.

_FF15TextFixSearch:
cmp dword ptr[ebx], edx ; is ebx equal to the recovered import??
je _FF15TextFix ; if so, fix import
add ebx, 4 ; if not, continue search
jmp _FF15TextFixSearch

_FF15TextFix:
sub ebx, (_IB+_stxt371)
add ebx, (_IB+_rdata)
mov dword ptr[eax+2], ebx ; fix the import :)
jmp _FF15Ok ; and move on to the next import. yay, we
; have a resolved call :D

_FF15End:

int 3 ; int 3 will cause the debugger to break,
; and we will know that we are done.
jmp _LongJumps ; goto next code section :D

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_LongJumps:

mov eax, (_IB+_text) ; beginning of .text

_LJLoop:
cmp byte ptr[eax], 0E9h ; is this byte the beginning byte of a
; longjump??
je _LJCheck1 ; if so, jump to checks

_LJOk:
inc eax ; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize) ; is it the end of .text?
je _LJEnd ; if so, end
jmp _LJLoop ; if not, continue

_LJCheck1:
mov ebx, dword ptr[eax+01] ; gets the value behind the LJ opcode
add ebx, eax ; adds the value to the offset
add ebx, 5 ; and add to ebx 5 so it contains the
; offset of where the longjump is
; pointing to
cmp ebx, (_IB+_stxt774) ; is it in STXT 774??
jl _LJOk ; if its less than the range, its all
; good ;)
cmp ebx, (_IB+_stxt774+_stxt774Size) ; is it in STXT 774??
ja _LJOk ; if its above the range, its all good ;)
jmp ebx ; alright, jump to this bad guy to
; resolve it ;)
; ok, remember to change the end of the
; pop edx routine to jmp _LJJumpHere
```




```
_LJJumpHere:
; return here from pop edx routine
pop ecx
; makes this place ez to identify
mov ecx, (_IB+_stxt371)
; beginning of STXT 371, which is used as
; temporary IAT

_LJFixSearch:

cmp dword ptr[ecx], edx
; ehh, is this import what we are looking
; for??
je _LJTextFix
; if so, lets fix up the LJ ;)
add ecx, 4
; move the pointer up
jmp _LJFixSearch
; and lets continue the loop

_LJTextFix:
sub ecx, (_IB+_stxt371)
; subtracts the offset of the temp, iat
add ecx, (_IB+_rdata)
; and adds .rdata so that this offset
; points to .rdata
mov word ptr[eax], 15FFh
; adds the customary FF15, which are the
; opcodes for the FF15 call
mov dword ptr[eax+02], ecx
; fix the value behind the FF15 ;)
jmp _LJOk
; oo, ahh, we have a resolved call :DDD

_LJEnd:
int 3
; int 3 will cause the debugger to break,
; and we will know that we are done.
jmp _ICR
; jmp to start resolving internal call
; reroutes ;)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_ICR:
mov eax, (_IB+_text)
; beginning of .text (for scanning)

_ICRLoop:
cmp byte ptr[eax], 0E8h
; is it an E8 (E8 call byte)?
je _ICRCheck
; well, check if its rerouted

_ICROk:

inc eax
; increase pointer to next place in .text
cmp eax, (_IB+_text+_textSize)
; is it the end of .text?
je _ICREnd
; if so, end
jmp _ICRLoop
; if not, continue

_ICRCheck:
mov ebx, dword ptr[eax+01]
; gets the value behind the ICR opcode
add ebx, eax
; adds the value to the offset
add ebx, 5
; and add to ebx 5 so it contains the
; offset of where the call is pointing to
cmp ebx, _ICROffset
; is it an internal call reroute??
jne _ICROk
; if it isn't, just jump to all is ok
; alright, remember to patch 6675F575 to a
; jmp _ICRJumpHere
mov esi, eax
; saves eax just in case it fucks up
jmp eax
; jump to the internal call reroute to
; resolve it
```



```
_ICRJumpHere:

mov esp, dword ptr[eax+1]          ; this is where we store registers
popad                             ; pops regs
popfd
pop esp
cmp eax, esi                      ; is eax still valid??
jne _ICRFixEAX                   ; if not, fix eax
jmp _ICROk                       ; continue, we have just resolved an
                                ; internal call reroute :)

_ICRFixEAX:

mov eax, esi                      ; fix eax
jmp _ICROk                       ; continue

_ICREnd:
int 3                            ; int 3 will cause the debugger to break,
                                ; and we will know that we are done.

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

_CopyIAT:

mov ebx, (_IB+_stxt371)           ; in ebx we put offset of temp IAT
mov ecx, (_IB+_rdata)            ; in ecx we put offset of .rdata

_CopyIATLoop:

mov eax, dword ptr [ebx]         ; copy from .temp iat to eax
mov dword ptr [ecx], eax         ; copy from eax to .rdata
add ebx, 4                       ; move pointer up by a dword
add ecx, 4                       ; move pointer up by a dword
cmp ecx, (_IB+_rdata+_rdataSize) ; are we at the end of the IAT??
jle _CopyIATLoop                ; if not, keep on going
int 3                           ; if so break here
popfd                            ; restore registers
popad

                                ; "

end start

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
```



All the code provided with this tutorial is free for public use, just make a greetz to the authors and the ARTeam if you find it useful to use. Don't use these concepts for making illegal operation, all the info here reported are only meant for studying and to help having a better knowledge of application code security techniques.

7. History

- Version 1.0 – First public release!



<http://cracking.accessroot.com>