

# Unpacking Fire Daemon 1.8 Armadilloed dll

deroko/ARTeam

December 2005

1.	Abstract.....	2
2.	Finding OEP in DLL and fixing eliminated IAT.....	3
3.	Fixing Code Splices .....	17
4.	Fixing not fixed part of IAT .....	21
5.	References.....	22
6.	Greetings.....	22

## Keywords

coding, armadillo, dll, unpacking

## 1. Abstract

Target that we are facing here is armadillo packed dynamic link library with strategic code splicing and iat elimination. In theory packed dlls may have copy-memII and nanomites, but in practice it is very hard to implement, requires separate debug loader for protected app, but it is doable.

I've planed to show here how to code own tools to fix code-splicing and IAT elimination. It is not smart to use ArmInline all the time, you have to learn how to code and how to create your own tools for some targets, and not only how to follow tutorials. Thinking and coding is all we need to solve this target, and of course some tools.

Some coding knowledge is required, all macros/includes/lib/tasm32/tlink32 are supplied with this document so you can compile my sources without a problem and adjust them for VAs on your machine. If you plan to use masm32, I wish you best of luck with changing codes to masm32 syntax. tasm32 rulz

Tools that we are going to use:

- SoftICE
- OllyDbg 1.10 with HideDebugger plug-in
- ImportRec 1.6
- LordPE
- tasm32 5.3 (tasm32/tlink32 are supplied with this document)
- PeID 0.94
- Hiew

Included sources and files:

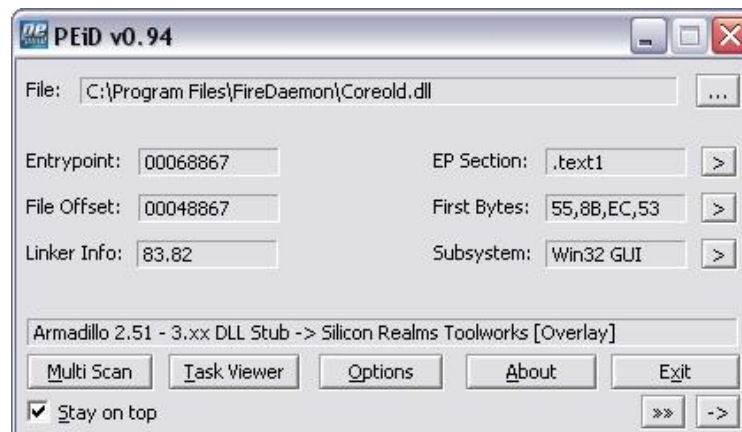
- **eliminate** [eliminate\eliminate.asm] used to rebase eliminated IAT
- **addsec** [addsec\addsec.asm] used to append apis loader to the last section of dll
- **codesplicingfixer** [codesplicingfixer\codesplicingfixer.asm] used to fix codesplices in dumped file
- **eax** [eax\eax.asm] yet another progy to fix some leftovers in final dump that didn't change with eliminate
- **dllbane** [dllbande\dllbande.zip] Dll Break 'n' Enter used to load dlls in softice for debugging
- **dump.exe** [dump.exe\dump.exe] used to get hex binary dump of any file so it can be used inside of your asm source codes, check addsec/import.inc which is binary dump of tree.txt
- **[tasm32]** – includes tasm32/tlink32 and pwrite.exe needed to compile source codes, so don't worry if you don't have tasm32 :D You will be able to compile source codes :D:D
- **[importlib]** – import32.lib needed to link my files
- **[includez]** – all my include files needed to compile my source codes

**Note:** each progy in it's own folder has **make.bat**, when you change virtualaddresses, run **make.bat** and you will get nice and clean compiled file.

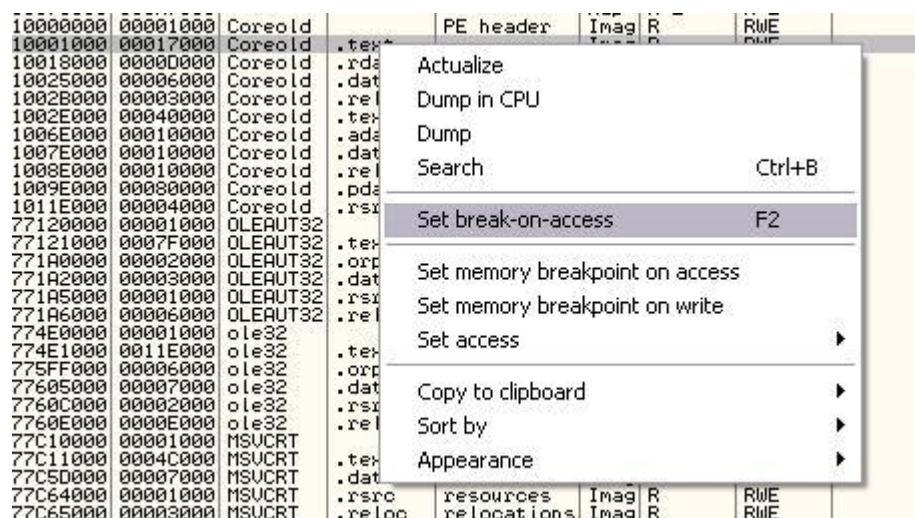
**S verom u Boga, deroko/ARTeam**

## 2. Finding OEP in DLL and fixing eliminated IAT

PeID shows that we are dealing with armadilloed dll:



Finding OEP in armadillo packed dlls is not hard at all, all you have to do is load dll in olly dbg, set memory breakpoint on Access on code section:



And run code:

003AE018	0FBF00	MOVX EAX, BYTE PTR DS:[EAX]	
003AE01B	3305 D8553C00	XOR EAX, DWORD PTR DS:[3C55D8]	
003AE021	25 FF000000	AND EAX, 0FF	
003AE026	8B0D D8553C00	MOV ECX, DWORD PTR DS:[3C55D8]	
003AE02C	C1E9 08	SHR ECX, 8	
003AE02F	8B8485 B4ACFFFF	MOV EAX, DWORD PTR SS:[EBP+EAX*4+FFFFACB]	
003AE036	33C1	XOR EAX, ECX	
003AE038	A3 D8553C00	MOV DWORD PTR DS:[3C55D8], EAX	
003AE03D	EB B8	JMP SHORT 003ADFF7	
003AE03F	6A 01	PUSH 1	
003AE041	58	POP EAX	
003AE042	85C0	TEST EAX, EAX	
003AE044	74 09	JE SHORT 003AE04F	

## Unpacking Fire Daemon 1.8 Armadilloed dll

Now set BPX as shown on picture and run code till you break on it, after you break at BPX, clear it and set new Memory break point on access at .text section and run code (same as on 1<sup>st</sup> picture):

100158E1	832D D06D0210 01	SUB DWORD PTR DS:[10026DDC],4	MSUCR71.free
100158F3	3905 DC6D0210	CMP DWORD PTR DS:[10026DDC],EAX	
100158F9	73 DE	JNB SHORT Coreold.100158D9	
100158FB	50	PUSH EAX	
100158FC	FF15 E4619D00	CALL DWORD PTR DS:[9D61E4]	
10015902	8325 E06D0210 01	AND DWORD PTR DS:[10026DE0],0	
10015909	59	POP ECX	
1001590A	33C0	XOR EAX,EAX	
1001590C	40	INC EAX	
1001590D	C2 0C00	RETN 0C	
10015910	6A 0C	PUSH 0C	
10015912	68 E8C80110	PUSH Coreold.1001C8E8	
10015917	E8 20010000	CALL Coreold.10015A3C	
1001591C	33C0	XOR EAX,EAX	
1001591E	40	INC EAX	
1001591F	8945 E4	MOV DWORD PTR SS:[EBP-1C],EAX	
10015922	33FF	XOR EDI,EDI	
10015924	897D FC	MOV DWORD PTR SS:[EBP-4],EDI	
10015927	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+C]	
1001592A	3BF7	CMP ESI,EDI	
1001592C	75 0C	JNZ SHORT Coreold.1001593A	
1001592E	393D D06D0210	CMP DWORD PTR DS:[10026DD0],EDI	
10015934	0F84 AC000000	JE Coreold.100159E6	
1001593A	3BF0	CMP ESI,EAX	
1001593C	74 05	JE SHORT Coreold.10015943	
1001593E	83FE 02	CMP ESI,2	
10015941	75 31	JNZ SHORT Coreold.10015974	
10015943	A1 D86D0210	MOV EAX,DWORD PTR DS:[10026DD8]	
10015948	3BC7	CMP EAX,EDI	
1001594A	74 0C	JE SHORT Coreold.10015958	

10015910 is OEP of this DLL, so now you know where to set hardware breakpoint to get always on OEP of this dll.

Note: whenever you plan to unpack dll, and you are not sure about oep, locate suspicious entrypoint and set hardware breakpoint on execution, so you might examine stack for handle of dll, reason, reserved arguments of DllEntry callback. Just note, has nothing to do with this tutorial :D

Now I will show you IAT elimination and code splices so you know what we have here:

IAT Elimination is located at : [100158FC](#) , as you can see call to API is out-of our dll range. Base of dll is [10000000h](#) and IAT is rebased at [9D6000h](#) so we are sure that it is rebased, but it is not only rebased, it is also shuffled to trick importrec when fixing IAT, take a look at next picture:

Address	Value	Comment
009D61E4	7C342151	MSUCR71.free
009D61E8	00396F88	
009D61EC	77DD7753	ADUAP132.OpenProcessToken
009D61F0	77DD6BF0	ADUAP132.RegCloseKey
009D61F4	7C3A6888	MSUCP71.??V?basic_string@DU?char_traits@D@std@@V?allocator@D@2@@@std@@QAEAAU01@PBD@Z
009D61F8	7C341CBE	MSUCR71._initterm
009D61FC	7C3505DC	MSUCR71._stricmp
009D6200	77124BC2	OLEAUT32.SysAllocString
009D6204	7C3A1E3D	MSUCP71.??I?basic_string@DU?char_traits@D@std@@V?allocator@D@2@@@std@@QAE@XZ
009D6208	77E1A841	ADUAP132.LsaRemoveAccountRights
009D620C	7C3A458D	MSUCP71.?find@?basic_string@DU?char_traits@D@std@@V?allocator@D@2@@@std@@QBEIPBDII@Z
009D6210	00371139	
009D6214	0039705C	
009D6218	7C80C6E0	kernel32.lstrlenA
009D621C	7C812851	kernel32.GetVersionExA
009D6220	00397022	
009D6224	77D8583D	USER32.GetUserObjectSecurity
009D6228	7C809794	kernel32.InterlockedDecrement
009D622C	7C356FBF	MSUCR71._stat

Yup, something is wrong, you see all those imports shuffled, they are kinda annoying to fix with `importrec`, at least I didn't succeed to fix them via `importrec` so I took another approach that may be used on any shuffled import table, in any other protector.

Also if you step into first call you will see code splices:

10015A3C	68 A0560110	PUSH Coreold.100156A0	JMP to MSUCR71._except_handler3
10015A41	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
10015A47	-E9 1285B2F1	JMP 01B3DF5E	
10015A4C	66:87F1	XCHG CX,SI	
10015A4F	73 00	JNB SHORT Coreold.10015A51	
10015A51	22:87E1	VPUS PV ST	

JMP at [10015A47h](#) is one out of many code splices, but we have to fix IAT first, code splices are easy to fix, but with imports we must not make any mistake, we need all imports and we will force armadillo to store redirected apis in IAT and latter we can dump it and code loader to load those APIs. Remember [\\_strcmpi](#) ? :D

Also code-splices are located at random addresses so address pointer by jmp will be changed latter during this tutorial :D Armadillo uses [time\(\)](#) to get random memory address for code-splices, this can be fixed at runtime but we are talking about coding our own tools to fix this and we are taking harder way which requires more knowledge(mostly coding knowledge). If you wanna know more how code splices can be defeated at runtime, I suggest you to read [haggar's Armadillo reversing tutorials](#) [1], he did outstanding job with code-splices :D

To fix imports I will use SoftICE, there is to much typing and clicking in olly, and I hate that + I've got familiar with sice conditional breakpoints and macros:

fire up dllbande [src/exe] and you will be in your SoftICE soon: (dllbande is silly name for DLL Break 'n' Enter progty that I've coded for my own needs)



Now, set [bpm 10015910 x dr0](#) to break at your oep, make sure that [yates's anti-bpm](#) is on so we can avoid drX-clearing trough SEH.

Well remember all those addresses that I've told that might change? well guess what, IAT address is changed due to simple fact that I've changed my os to win 2k sp4, IAT is still the same but it has now different address. Now we shall hunt down everything we need to rebuild IAT from SoftICE with nice screenshots :D



```

001B:100158D9 MOU ECX,[10026DDC]
001B:100158DF MOU ECX,[ECX]
001B:100158E1 TEST ECX,ECX
001B:100158E3 JZ ↓100158EC
001B:100158E5 CALL ECX
001B:100158E7 MOU EAX,[10026DE0]
001B:100158EC SUB DWORD PTR [10026DDC],04
001B:100158F3 CMP [10026DDC],EAX
001B:100158F9 JAE ↑100158D9
001B:100158FB PUSH EAX
001B:100158FC CALL [00E661E4]
001B:10015902 AND DWORD PTR [10026DE0],00
001B:10015909 POP ECX
001B:1001590A XOR EAX,EAX
001B:1001590C INC EAX
001B:1001590D RET 000C
001B:10015910 PUSH 0C
001B:10015912 PUSH 1001C8F8
001B:10015917 CALL ↓10015A3C
001B:1001591C XOR EAX,EAX
001B:1001591E INC EAX
001B:1001591F MOU [EBP-1C],EAX
001B:10015922 XOR EDI,EDI
001B:10015924 MOU [EBP-04],EDI
001B:10015927 MOU ESI,[EBP+0C]
001B:1001592A CMP ESI,EDI
001B:1001592C JNZ ↓1001593A
001B:1001592E CMP [10026DD0],EDI
001B:10015934 JZ ↓100159E6
001B:1001593A CMP ESI,EAX
001B:1001593C JZ ↑10015943

```

Now take a closer look at [100158FC](#), call to eliminated IAT, and we see that it points to: [E661E4](#), so all we want right now is to see when we have some writing at that address. You may use olly to break at certain address, but I will use SoftICE to go fast on this subject:

So load dll trough [dllbande](#) again and set [BPMD E661E4](#), and run code, you will break here:

```

001B:00C9AC45 MOU EAX,[EBP+FFFFD618]
001B:00C9AC4B XOR EDX,EDX
001B:00C9AC4D MOU ESI,00002710
001B:00C9AC52 DIV ESI
001B:00C9AC54 IMUL EAX,[EBP+FFFFA028]
001B:00C9AC5B XOR EDX,EDX
001B:00C9AC5D MOU ESI,00002710
001B:00C9AC62 DIV ESI
001B:00C9AC64 MOU ECX,[ECX*4+EBP+FFFFD794]
001B:00C9AC6B ADD ECX,EAX
001B:00C9AC6D MOU EAX,[EBP+FFFFD614]
001B:00C9AC73 MOU EDX,[EBP+FFFFD7E4]
001B:00C9AC79 MOU [EAX*4+EDX],ECX
001B:00C9AC7C JMP ↑100C9AB85 (JUMP ↑)
001B:00C9AC81 MOU EAX,[00CBAF9C]
001B:00C9AC86 MOU EAX,[EAX]
001B:00C9AC88 MOU [EBP+FFFFD61C],EAX
001B:00C9AC8E MOU EAX,[00CBAF9C]
001B:00C9AC93 ADD EAX,04
001B:00C9AC96 MOU [00CBAF9C],EAX
001B:00C9AC9B MOU EAX,[EBP+FFFFD61C]
001B:00C9ACA1 SHL EAX,02
001B:00C9ACA4 MOU [EBP+FFFFD620],EAX
001B:00C9ACA8 MOU EAX,[EBP+FFFFD620]
001B:00C9ACB5 ADD EAX,00010000 ; "=C:=C:\tutorial\armadll"
001B:00C9ACB6 PUSH EAX
001B:00C9ACB8 CALL MSUCRT:??20YAPAXI0Z

```

Nope, not the one we need, so type X, or press F5 to see if we are going to break at something good:

```

001B:00C9CD63 JMP ↓00C9CD9F
001B:00C9CD65 MOU EAX,[EBP+08]
001B:00C9CD68 MOU EAX,[EAX]
001B:00C9CD6A MOU DWORD PTR [EAX],00000003
001B:00C9CD70 CALL [KERNEL32*GetLastError]
001B:00C9CD76 PUSH EAX
001B:00C9CD77 LEA EAX,[EBP+FFFC268]
001B:00C9CD7D PUSH EAX
001B:00C9CD7E PUSH DWORD PTR [EBP+FFFFD384]
001B:00C9CD84 PUSH 00CAD030 ; "File \"%s", function \"%s" (error %d)"
001B:00C9CD89 MOU EAX,[EBP+08]
001B:00C9CD8C PUSH DWORD PTR [EAX+04]
001B:00C9CD8F CALL [MSUCRT*sprintf]
001B:00C9CD95 ADD ESP,14
001B:00C9CD98 XOR EAX,EAX
001B:00C9CD9A JMP ↓00C9E637
001B:00C9CD9F MOU EAX,[EBP+FFFFD910]
001B:00C9CDA5 CMP EAX,[EBP+FFFFD964]
001B:00C9CDAB JAE ↓00C9CDA
001B:00C9CDAD MOU EAX,[EBP+FFFFD910]
001B:00C9CDB3 MOU ECX,[EBP+FFFC68]
001B:00C9CDB9 MOU [EAX],ECX
001B:00C9CDBB MOU EAX,[EBP+FFFFD910]
001B:00C9CDBF ADD EAX,04
001B:00C9CDC4 MOU [EBP+FFFFD910],EAX
001B:00C9CDC6 JMP ↑00C9CA1C
001B:00C9CDCF CALL [KERNEL32*GetTickCount]

```

Hell yeah, if you take closer look at [C9CDB9h](#) you will see import filling process, now here is part where Armadillo decides which API address is stored as is, and which API is redirected to armadillo's virtual security.dll or any other memory bridge :D Scroll a little bit up:

```

001B:00C9CBC8 MOV     EAX,[EBP+FFFFFFC258]
001B:00C9CBCE CMP     DWORD PTR [EAX+08],00
001B:00C9CBD2 JZ      +00C9CC1D
001B:00C9CBD4 PUSH   00000100
001B:00C9CBD9 LEA     EAX,[EBP+FFFFFFC158]
001B:00C9CBDF PUSH   EAX
001B:00C9CBE0 MOV     EAX,[EBP+FFFFFFC258]
001B:00C9CBE6 PUSH   DWORD PTR [EAX]
001B:00C9CBE8 CALL    +00C72105
001B:00C9CBED ADD     ESP,0C
001B:00C9CBF0 LEA     EAX,[EBP+FFFFFFC158]
001B:00C9CBF6 PUSH   EAX
001B:00C9CBF7 LEA     EAX,[EBP+FFFFFFC268]
001B:00C9CBFD PUSH   EAX
001B:00C9CBFE CALL    [MSUCRT!_strcmpi]
001B:00C9CC04 POP     ECX
001B:00C9CC05 POP     ECX
001B:00C9CC06 TEST    EAX,EAX
001B:00C9CC08 JNZ     +00C9CC1B
001B:00C9CC0A MOV     EAX,[EBP+FFFFFFC258]
001B:00C9CC10 MOV     EAX,[EAX+08]
001B:00C9CC13 MOV     [EBP+FFFFFFCA68],EAX
001B:00C9CC19 JMP     +00C9CC1D
001B:00C9CC1B JMP     +00C9CBB9
001B:00C9CC1D MOV     EAX,[EBP+FFFFFD4A8]
001B:00C9CC23 INC     EAX
001B:00C9CC24 MOV     [EBP+FFFFFD4A8],EAX

```

Haha, do you see `_strcmpi` at [C9CBFE](#)? Yeah, do you see `jnz` at [C9CC06](#)? Well we are not going to patch `jnz` to `jmp`. Trick here is to set hardware breakpoint at [C9CC0Ah](#) (eax is zero) and whenever that bpm is triggered to continue execution from [C9CC1Bh](#). Luckily Armadillo will try to obfuscate a few calls from `kernel32.dll`, `user32.dll` and as I know one from `advapi32.dll` (`RegCreateKeyExA`).

So, again, fire up `dllbande` and set hardware breakpoint like this (ofcourse don't delete one that will stop at entrypoint):

**BPM C9CC0A x dr1 do "r eip C9CC1B; x"**

Softice users know what this will do, but for olly users I'll explain this, simply we tell softice whenever(in this particular case) `eip` reaches [C9CC0Ah](#), to change `eip` to [C9CC1Bh](#) and continue execution from there, instead of patching `jnz` to `jmp` :D:D

SoftICE will popup a few times, and finally it will stop at our entrypoint because we didn't delete our hardware breakpoint on execution at : [10015910](#).

When we reach OEP we are interested in moving IAT in some section that will be present in our dump. After a little bit of thinking and looking in hiew, I decided to use `.adata` section:

Number	Name	UirtSize	RVA	PhysSize	Offset	Flag
1	.text	00016216	00001000	00000000	00000000	60000020
2	.rdata	0000C08C	00018000	0000D000	00001000	40000040
3	.data	00005DE4	00025000	00000000	00000000	C0000040
4	.reloc	00002C48	0002B000	00000000	00000000	42000040
5	.text1	00040000	0002E000	0003B000	0000E000	60000020
6	.adata	00010000	0006E000	0000D000	00049000	60000020
7	.data1	00010000	0007E000	00007000	00056000	C0000040
8	.reloc1	00010000	0008E000	00004000	0005D000	42000040
9	.pdata	00080000	0009E000	0007C000	00061000	C0000040
10	.rsrc	00004000	0011E000	00004000	000DD000	40000040

Cursor's out of sections

.adata section has RVA of **6E000** which added to image base gives us **1006E000h** VA. Now we shall code progy to fix calls/jmps and some mov instructions and reabse IAT.

First we need some constants [code is located in eliminate\eliminate.asm]:

```
<++>
e_start      equ      0E66000h
e_end        equ      0E666D4h

e_new        equ      1006e000h

c_start      equ      10001000h
c_si ze      equ      17000h
c_end        equ      c_start + c_si ze

pi d         equ      2e0h
<++>
```

**e\_start** and **e\_end** are constants that represent eliminated IAT start VirtualAddress and eliminated IAT end VirtualAddress.

**e\_new** is VirtualAddress of .adata section

**c\_start** - VirtualAddress of code section

**c\_size** - size of Code section

**c\_end** - VirtualAddress that represents end of code section

**pid** - well no need to explain this :D

Idea is now simple, read whole code section out of DLL using ReadProcessMemory, and rearrange all calls to point to .adata section using this simple formula:

$$\text{OldVA} - \text{e\_start} + \text{e\_new}$$

This formula will be used for each eliminated call, when this process is done we shall simply read old eliminated IAT and store that into .adata section using WriteProcessMemory.

```
<++>
push  pid
push  0
push  PROCESS_ALL_ACCESS
call  OpenProcess                    ; open process, not much to
mov   phandl e, eax                 ; comment here

push  offset oldprot
push  PAGE_EXECUTE_READWRITE
push  c_si ze
push  c_start
push  phandl e
call  VirtualProtectEx              ; change protecti on on whol e code
                                       ; secti on
```



```
push    PAGE_READWRITE
push    MEM_COMMIT
push    c_size+1000h
push    0
call    VirtualAlloc           ; allocate big enough buffer so
mov     code, eax              ; we can copy whole code section
                                   ; and make all modification
                                   ; in our process

push    offset oldprot
push    PAGE_EXECUTE_READWRITE
push    e_end - e_start
push    e_start
push    phandle
call    VirtualProtectEx      ; change protection of eliminated
                                   ; IAT code block, not required

push    PAGE_READWRITE
push    MEM_COMMIT
push    e_end - e_start
push    0
call    VirtualAlloc           ; allocate buffer for eliminated
mov     eliminated, eax        ; IAT because we will copy that
                                   ; .adata section section in this
                                   ; example

push    0
push    e_end - e_start
push    eax
push    e_start
push    phandle
call    ReadProcessMemory      ; read eliminated IAT

push    0
push    c_size
push    code
push    c_start
push    phandle
call    ReadProcessMemory      ; and read whole code section in
                                   ; allocated buffer
```

<++>

This is simple part of code, all we do is allocate big enough buffer to read whole CODE section, and also one buffer for eliminated IAT (we need it so we can copy that to .adata section).

Code is commented, and there is no much to talk about this part, what we are interested here is fixing call/jmps and mov instructions. Hold on, simple algo is coming:

<++>

```
mov     ebx, code
mov     edx, ebx
add     edx, c_size
```

; whole engine to eliminate IAT is here, note that this proggy is more  
; than good to use lde and speed up process without any chance to foobar  
; something during static analyze of this code, sometimes I love vc  
; compiler: D No embedded data in code section, procs are padded with int  
; 3h so we can use LDE without any problem on whole code section

; As you can see ebx is pointer to buffer that contains code section, and  
; edx is used as end\_of\_code check

\_\_cycleredirection:

```
    cmp     ebx, edx    ; if end of code we are done and ready to
    jae     __done     ; copy modified section back to dll for
                        ; dumping
```

```
    cmp     word ptr[ebx], 15FFh ; is it call []?
    je      __checkelimination ; if yes, try to rebase it
    cmp     word ptr[ebx], 25FFh ; is it jmp []?
    je      __checkelimination ; if yes, try to rebase it
    cmp     byte ptr[ebx], 8Bh   ; one more condition in this dll
    jne     __incebxeax        ; mov ebx, dword ptr[iat]
                                ; 6 byte len instruction
                                ; mov eax, [redirected iat]
                                ; should be fixed latter
```

```
    push    ebx           ; at this point I call lde to
    call    lde86         ; be sure that this is not wrong
    cmp     eax, 6         ; instruction, no sib/dis/imm
    je      __checkelimination
```

\_\_incebxe:

```
    inc     ebx
    jmp     __cycleredirection
```

\_\_checkelimination:

```
    mov     eax, dword ptr[ebx+2] ; now we take address used
    cmp     eax, e_start         ; by this instruction in eax
    jb      __notmine           ; and check if that address
    cmp     eax, e_end           ; is in IAT eliminated range
    ja      __notmine           ; if not we continue scan
```

```
    ; if it is in our eliminated iat range, we use simple
    ; formula that I've presented above:
    ; IAT_ELIMINATED_PTR - IAT_ELIMINATION_BASE + NEW_IAT_BASE
```

```
    sub     eax, e_start
    add     eax, e_new
    mov     dword ptr[ebx+2], eax
```

```
__notmine:    inc     ebx
              jmp     __cycleredirection
```

<++>

Done, simple we scan code section for FF25 and FF15 and changing addresses :D

Now run this code(compiled to take your values, run make.bat ) and check how code has changed.

Before:

```

001B:100158D9  MOV     ECX,[10026DDC1]
001B:100158DB  MOV     ECX,[ECX]
001B:100158DD  TEST    ECX,ECX
001B:100158DE  JZ      ↓100158EC
001B:100158E0  CALL    ECX
001B:100158E2  MOV     EAX,[10026DE01]
001B:100158E4  SUB     DWORD PTR [10026DDC1],04
001B:100158E6  CMP     [10026DDC1],EAX
001B:100158E8  JAE     ↑100158D9
001B:100158EA  CALL    [00E661E4]
001B:100158EC  AND     DWORD PTR [10026DE01],00
001B:100158EE  XOR     EAX,EAX
001B:100158F0  INC     EAX
001B:100158F2  RET     000C
001B:10015910  PUSH    00
001B:10015912  PUSH    1001C8E8
001B:10015914  CALL    ↓10015A3C
001B:10015916  XOR     EAX,EAX
001B:10015918  INC     EAX
001B:1001591A  MOV     EAX,EAX

```

After:

```

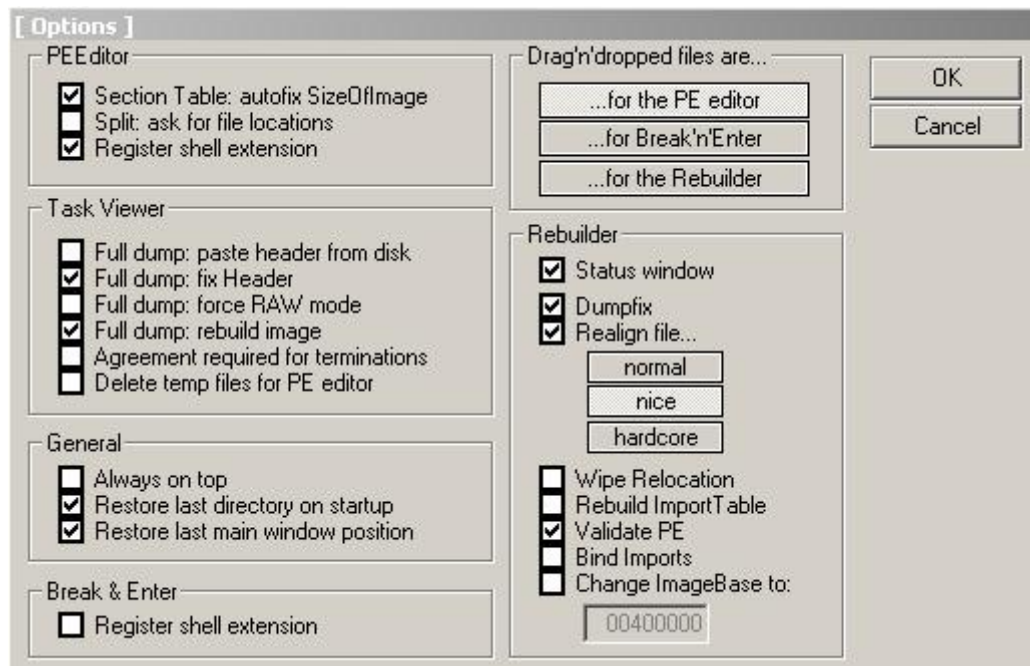
001B:100158D9  8B0DDC6D0210  MOV     ECX,[10026DDC1]
001B:100158DF  8B09          MOV     ECX,[ECX]
001B:100158E1  85C9          TEST    ECX,ECX
001B:100158E3  7407          JZ      ↓100158EC
001B:100158E5  FFD1          CALL    ECX
001B:100158E7  A1E06D0210   MOV     EAX,[10026DE01]
001B:100158EC  832DDC6D021004 SUB     DWORD PTR [10026DDC1],04
001B:100158F3  3905DC6D0210 CMP     [10026DDC1],EAX
001B:100158F9  73DE          JAE     ↑100158D9
001B:100158FC  FF15E4E10610 CALL    [MSUCR71+free1]
001B:10015900  59           AND     DWORD PTR [10026DE01],00
001B:10015909  59           POP     EAX
001B:1001590A  33C0          XOR     EAX,EAX
001B:1001590C  40           INC     EAX
001B:1001590D  C20C00        RET     000C
==> 10015910  EBFE          JMP     ↑10015910 (JUMP ↑)
001B:10015912  6818C8E8     PUSH    1001C8E8
001B:10015917  F820010000   CALL    ↓10015A3C

```

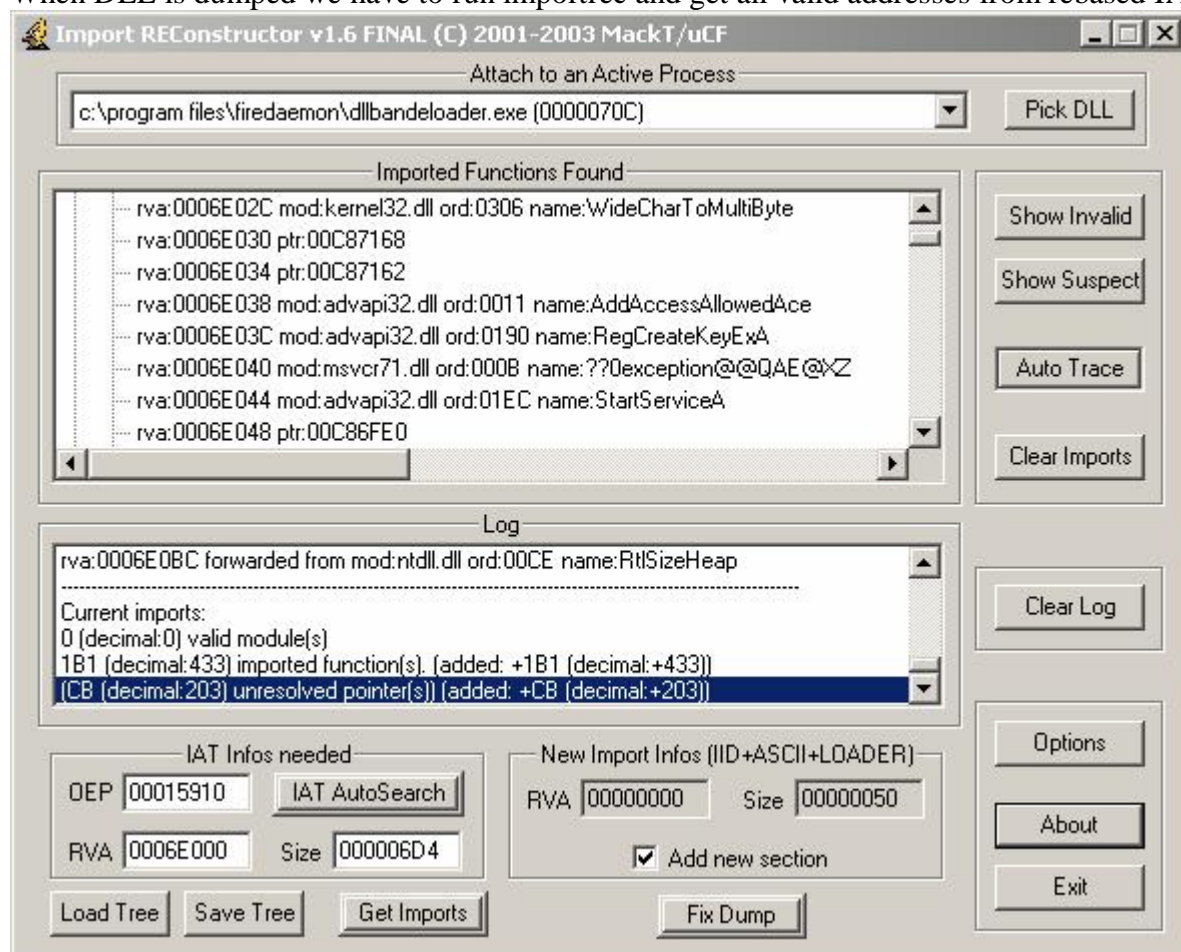
Heh, check hex value of call : 1006E1E4h , it is rebased IAT :D :D we did it :D

Note: there are also some leftovers like mov eax, [] but we are going to fix them latter on, to be honest I'm writing all steps that I've took to solve this armadilloed DLL for the first time. Approach used here is not common, I guess :D

Run LordPE and dump this dll, settings that I've used in my LordPE:



When DLL is dumped we have to run importrec and get all valid addresses from rebased IAT:



Note that I didn't use IAT AutoSearch because it will report invalid OEP, click on [Get Imports](#), and you will get similar output as I did.

If you mark invalid pointers with importrec you will see that some APIs will be marked as invalid, if you cut them you will mess everything because they are needed for final dump to work!!! So we can't use importrec to fix imports but we have all API names, including DLL and RVA where address of API should be. Cut invalid thunks and fix dump just to make imports because on win2k non exe/dll will work which doesn't have import table!!! This doesn't apply for XP. Note, when you fix dll it is not fixed yet, some imports are still missing, I figured this when I've switched to XP and figured that half of imports are missing due to wrong recognition of INVALID ptrs by importrec. [So loader is only way to fix our imports](#). Anyone knows any better way? :D

Save imports that you got with [Save tree](#). (including invalid ptrs, delete them from tree.txt using text editor and also remove importrec comments from tree.txt)

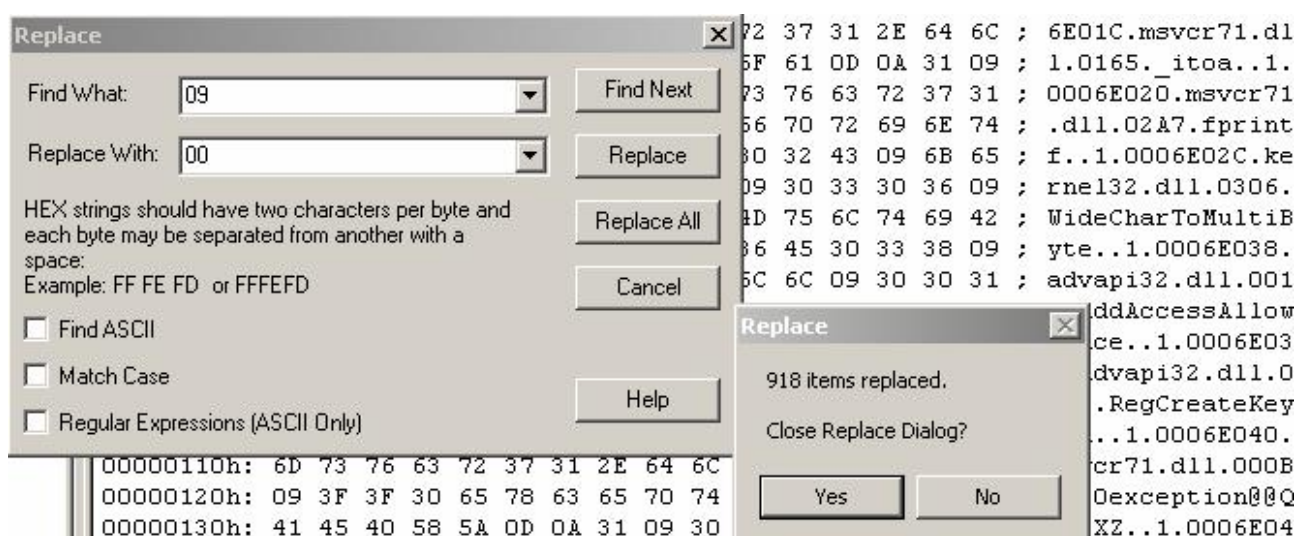
Let's have a look at modified tree.txt to see layout of importrec tree:

1	0006E45C	msvcr71.dll	0144	_iob
1	0006E460	advapi32.dll	0172	OpenServiceA
1	0006E464	advapi32.dll	01BF	RegisterEventSourceA
1	0006E468	user32.dll	00C2	EnumDesktopWindows
	^^^^^^^^	^^^^^^^^^^^^^^^^		^^^^^^^^^^^^^^^^^^^^^^^^^^^^
	Address	RVA	Dynamic Link Library	API Names

Wonderful, ALL WE NEED TO FIX THIS.

We have to modify tree.txt so it becomes binary file, as far as you all know, importrec separates columns using TAB(\t in C) and we have to change all of them to '0' because we are gonna work on some strings here. Open tree.txt in UltraEdit32 and delete invalid ptrs including importrec header.

select Search -> Replace:



Not bad, we changed all tabs to '0', same thing should be applied for CR/LF.

In Find What type : 0D 0A and in Replace With: 00 00, click on Replace All :D



Bingo, we got binary dump file ready to be used with our loader.

Now is time to code code injector that will load our imports, I will focus more on structure of loader(API resolver) because you may find a lot of documentation about PE on net, and also about VX coding, or just check addsec\addsec.asm

So far our modified tree.txt looks like this:

```
00000000: 31 00 30 30-30 36 45 30-30 38 00 6B-65 72 6E 65 1 0006E008 kerne
00000010: 6C 33 32 2E-64 6C 6C 00-30 30 33 35-00 43 72 65 132.dll 0035 Cre
00000020: 61 74 65 45-76 65 6E 74-41 00 00 31-00 30 30 30 ateEventA 1 000
00000030: 36 45 30 31-43 00 6D 73-76 63 72 37-31 2E 64 6C 6E01C msvc71.dl
```

```
1\x000006E008\x00kernel32.dll\x000035CreateEventA\x00\x00
```

Steps that loader will take:

1. increment pointer by 2 to get RVA
2. convert string-RVA to hex-RVA and add imagebase to it
3. load dll via LoadLibraryA and increment counter by 5 to skip ordinal
4. load API and store address to VA calculated in stage 2
5. locate end of API string and set pointer to next line from tree.txt

Here we go again with code:

```
<+>
del ta:      pushad
              call    del ta          ;usual code to get del ta
              pop     ebp              ;in any offset independent
              sub     ebp, offset del ta ;code
                                              ;eg. viruses, shell codes
                                              ;packers, protectors

              mov     eax, dword ptr FS:[30h]
              mov     eax, dword ptr[eax+0ch]
              mov     eax, dword ptr[eax+1ch]
              mov     eax, dword ptr[eax]
              mov     eax, [eax+8]
              mov     [ebp+kernel 32], eax
              ;get k32 base address from PEB, all greetingz fly
              ;to Ratter/29a who discovered this nice technique
              ;that works great on NT based os-es

              gethash <GetProcAddress>    ;gethash is macro that will
              push    hash                  ;generate hash at link-time
              push    eax                    ;for certain api, and may be
              call    getprocaddress        ;used later to get API by
              mov     [ebp+GetProcAddress], eax ;scanning export table of
              gethash <LoadLibraryA>      ;dll for matching API name
              push    hash                  ;Check 29a #4 zine for more
              push    [ebp+kernel 32]      ;info, tutorials by Billy
              call    getprocaddress        ;Bel cabu and Lethal Mind.
```

## Unpacking Fire Daemon 1.8 Armadilloed dll

---

```
mov     [ebp+LoadLibraryA], eax; gethash uses z0mbi e's hash
                                           ; algo, 7 instructions code

lea     esi, [ebp+imports]      ; esi pointer to our binary
                                           ; dump of modified tree.txt
__cycleallimportz:
lea     edi, [ebp+imports]
add     edi, import_end
cmp     esi, edi
jae     __gotooep
add     esi, 2                  ; skip 1+0x00
push    esi
call    htodw                  ; convert string to HEX
add     eax, 10000000h         ; align to imagebase
mov     edi, eax               ; va of thunk in edi
add     esi, 9                 ; rva of api + 0x00
push    esi
call    LoadLibraryA         ; load this dll
mov     ebx, eax              ; dll handle in ebx

__getdllend:
lodsb                   ; increment pointer
test    al, al            ; by searching for 0 at end of
jnz     __getdllend       ; dll name

add     esi, 5            ; ordinal +0

push    esi               ; pointer to API name
push    ebx               ; dll base
call    GetProcAddress    ; get api address
mov     [edi], eax        ; store it in calculated
                                           ; VA

__getapiend:
lodsb                   ; find end of API =)
test    al, al
jnz     __getapiend
inc     esi               ; api is terminated by 2 zeros
cmp     dword ptr[esi], 0
je      __gotooep
jmp     __cycleallimportz

__gotooep:
popad
db      0E9h              ; This should be patched in hi ew
dd      0                 ; as jmp OEP for this dll...
nop

GetProcAddress         dd      ?      ; GetProcAddress is stored here
LoadLibraryA           dd      ?      ; LoadLibraryA is stored here
kernel 32              dd      ?      ; kernel 32 base is stored here

include                api_zloader.inc ; my include file for
                                           ; getkernel base and getprocadd
include                htodw.inc       ; string to hex
<++>
```

DONE, run addsec.asm and you are ready to open hiew and change JMP to OEP with good address. Also take care that since this is entry of DLL this will be called whenever DLL entry is called (process\_attach, trhead\_attach, process\_deattach, thread\_deattach) so it would be smart to check if this is called for PROCESS\_ATTACH, if so load APIs, if not jmp to oep, but I'll let that up to you to code :D

This code will be treated as virus by many AVs due to the fact that entrypoint is in last section, but as you can see there is no any evil code in here =), also we will use dump.exe (progy included) to get binary dump of modified tree.txt so it can be used within our source code, compile and run addsec.asm.

Go to last section and locate code start : "60E8" which for me starts at VA 10123000, and patch `jmp 00000000` to `jmp OEP`

```

fixing_.dll  4FWO EDITMODE  a32  PE  000EC0A1  Hiew 7.21  (c)SEN
000EC099: 833E00      cmp     d,[esi],0
000EC09C: 7402        je      000EC0A0
000EC09E: EBB5        jmps    000EC055
000EC0A0: 61          popad
000EC0A1: E9000000    jmp     000EC0A6
000EC0A6: 90          nop
000EC0A7: 0000        add     [eax],al
000EC0A9: 0000        add     [eax],al
000EC0AB: 0000        add     [eax],al
00
00
00
00
00
00
00
00
00
00
000EC0B6: 648B32      mov     esi,fs:[edx]
000EC0B9: AD          lodsd
000EC0BA: 83F8FF      cmp     eax,-1
000EC0BD: 7404        je      000EC0C3
000EC0BF: 8BF0        mov     esi,eax
000EC0C1: EBF6        jmps    000EC0B9
000EC0C3: 8B7E04      mov     edi,[esi][04]
000EC0C6: 81E70000FFF and     edi,0FFFFFFF ;'
000EC0CC: 66813F4D5A  cmp     w,[edi],05A4D ;'ZM'
1          2          3          4          5          6          7          8          9          10
  
```

Hehe, change entry of DLL in PE header to beginning of your code (for me it is 123000) and load code in Olly, you will see that all imports are being resolved at runtime.

Done with boring part so far, all we need is to fix code splices now.

### 3. Fixing Code Splices

Heh, annoying part starts now. Why? Because I figured that my lde doesn't handle well prefix 67 with some instructions. 67: jecxz was recognized as 4 byte long opcode, due to wrong flags in lookup table, but I've fixed that now.

Dump code splices and finally nail this bitch(follow any jmp that takes to code splices):



```

001B:026BD000 XCHG     ESI,EBX
001B:026BD002 XCHG     EDI,EBX
001B:026BD004 XCHG     EDI,EBX
001B:026BD006 XCHG     SI, BX
001B:026BD009 XCHG     EDI,EDX
001B:026BD00B NOT      ECX
001B:026BD00D MOV     EAX,[ESP+1C]
001B:026BD011 JMP     +100144F5
001B:026BD016 PUSH    EAX
001B:026BD017 LEA     ECX,[ESP+24]
001B:026BD01B PUSH    ECX
001B:026BD01C LEA     ECX,[ESP+1C]
001B:026BD020 NOT      EBX
001B:026BD022 MOV     EBX,EBX
001B:026BD024 BSWAP   EDX
001B:026BD026 PUSH    EDI
001B:026BD027 JLE     +026BD02B
001B:026BD029 JLE     +026BD051
001B:026BD02B XCHG     EDX,EDX
001B:026BD02D POP     EDI
001B:026BD02E JO      +026BD032
001B:026BD030 JO      +026BD059
001B:026BD032 BSWAP   EDX
001B:026BD034 NOT      EBX
001B:026BD036 MOV     BYTE PTR [ESP+60],09
001B:026BD03B JMP     +1001451F
001B:026BD040 XCHG     SI, BX
001B:026BD043 XCHG     DI, SI
001B:026BD046 NOT      EDX
001B:026BD048 XCHG     AX, DX
001B:026BD04A XCHG     AX, DX

```

Locate with LordPE region that holds your code splices and dump it, also don't forget to remember base address of your code splices because we are gonna need it to fix them.

if we take a closer look at code splices code, we might figure that they are easy to fix. Why? Do you see JMP at [26BD001h](#) ? It will take us to place from where jmp in code section jumped to code splice:

Example:

```

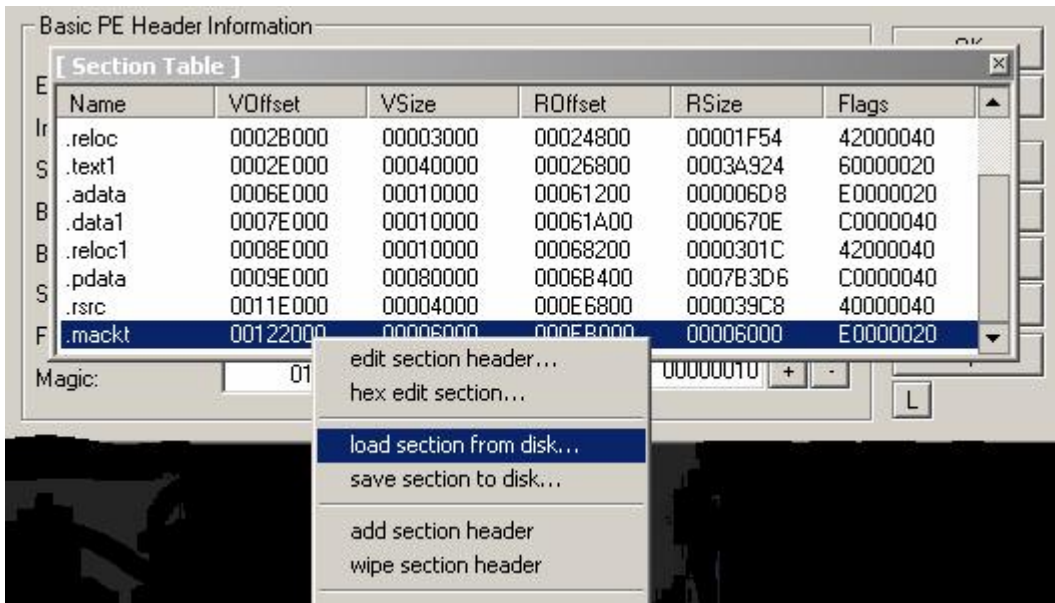
100144F0 JMP 26BD000 -----> XCHG EDI, EBX
100144F5 NOP                  XCHG EDI, EBX
                                MOV  EAX, [ESP+1C]
                                JMP  100144F5

```

JMP in code splice will take us back to instruction after [JMP 26BD000](#), which is at [100144F5h](#), now when you calculate that address use it to locate instruction in CODE section, sub that address with 5 and bingo you are at VA of jmp that will redirect execution to this particular code-splice. Simple? Just change offsets in jmps in code spliced section, and in code section :D

Source code is included [codesplicingfixer/codesplicingfixer.asm]

Oki, when you have dumped spliced section, add that as new section to our partially fixed dump file using LordPE:



And we are ready to fix splices once and for all.

Sorry but I've dumped codesplices in 2nd run so I'll have to use 2 base addresses for code splices:

- 1st - from 1st run : 26B0000h used to fix jmps in CODE section (I've dumped dll at this point)
- 2nd - from 2nd run : 35A0000h used to calculate where splice should jmp

All needed data to fix this are here:

```
<-->
codespl i ceva          equ      26B0000h    ; VA of frist run
codespl i ceva2ndrun    equ      35A0000h    ; VA of second run
codespl i cenewva       equ      10128000h    ; VA of code splices in image
                        ; newly added section
i magebase              equ      10000000h    ; i magebase
<-->
```

Note: if you have dumped code splices at the same time when you have dumped dll then [codespliceva](#) and [codespliceva2ndrun](#) should be equal!!!!!!

Let me introduce you my code splice fixing engine :D :

```
<++>
__di sassembl e:
    push    s_ptr
    call    l dex86                ; returns l en i n eax

    mov     esi , s_ptr            ; first we check for end
    cmp     dword ptr[esi], 0      ; of spliced section, whi ch
    jne     __goon                 ; is padded wi th 0s
    cmp     dword ptr[esi+4], 0
    je      __done
```



```

__goon:
    cmp     byte ptr[esi], 0E9h ;is this jmp 32?
    je      __check_code_splice ;if so check for code splice

    add     s_ptr, eax
    jmp     __disassemble

__check_code_splice:
    mov     ecx, s_ptr           ;s_ptr raw of instruction
    sub     ecx, s_raw           ;PointerToRawData of spliced sec
    sub     ecx, mhandle         ;handle of mapped file
    add     ecx, codespliceva2ndrun ;constant to get VA of splice
    add     ecx, 5               ;len of instruction
    add     ecx, dword ptr[esi+1] ;ecx has virtual address of next
                                ;instruction after jmp to code
                                ;splice

    mov     edx, ecx
    cmp     edx, 10001000h       ;check if jmp is in our range
                                ;"E9 00 00 00 00" and similar
    jb      __check             ;should be skipped
    cmp     edx, 10018000h
    ja      __check             ;check is used for debugging
                                ;of this code

    mov     eax, s_ptr           ;now we repeat same steps but
    sub     eax, s_raw           ;this time we need Virtual Address
    sub     eax, mhandle         ;of splice in dumped file so we
    add     eax, codesplicesnewva ;can calculate relative jmp to
    add     eax, 5               ;opcode after jmp code_splice

    sub     ecx, eax             ;calculate new relative

    mov     eax, s_ptr
    mov     dword ptr[eax+1], ecx ;and fix code splice =)

    push    edx                 ;now locate jmp in code section
    pop     eax                 ;which is at jmp from code splice
    sub     edx, imagebase      ;addr-5
    sub     edx, c_rva          ;edx has VA, we sub VA with image
    add     edx, c_raw          ;base and section RVA, and add to
    add     edx, mhandle        ;it section RAW and mhandle so we
                                ;get position of jmp in code
                                ;section
    sub     edx, 5              ;heh jmpy located (raw)
    sub     eax, 5              ;jumpy located (VA)

    ; now we have to fix jmp in code section which is located in EDX

    mov     ecx, eax            ;calculate address at which jmp
    add     ecx, 5              ;to code splice is pointing
    add     ecx, [edx+1]

```

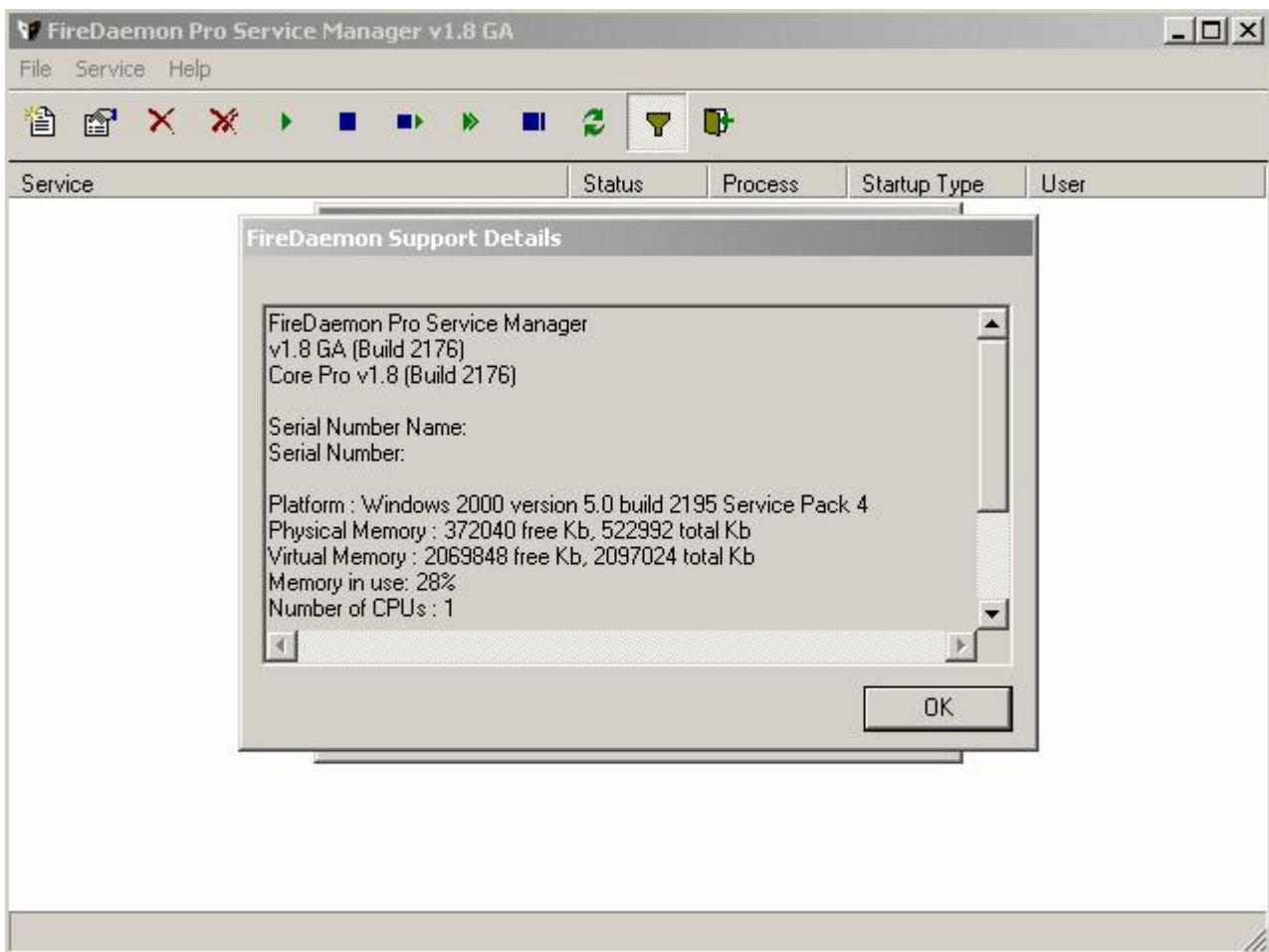
## Unpacking Fire Daemon 1.8 Armadilloed dll

---

```
sub    ecx, codespl i ceva           ; wrong addr in ecx so we have
add    ecx, codespl i cewva         ; to recalculate it: sub ECX with
sub    ecx, eax                     ; base of code splice region and
sub    ecx, 5                       ; add to it VA of code splice
mov    [edx+1], ecx                 ; section in our dump file

__conti nue:
    add    s_ptr, 5
    j mp   __di sassembl e
__done:
<++>
```

that is ALL, run codesplicefixer.asm and now you have fully functional dll unpacked by you without usage of any external tools, copy final.dll to core.dll and run Fire Daemon, bingo it works:



## 4. Fixing not fixed part of IAT

Well this works, nice, now click on Service -> New, damn proggy crashes =( Luckily I've set olly to be my JIT (nah I switched to XP so no more softice in this tutorial), olly shows that this instruction caused crash:

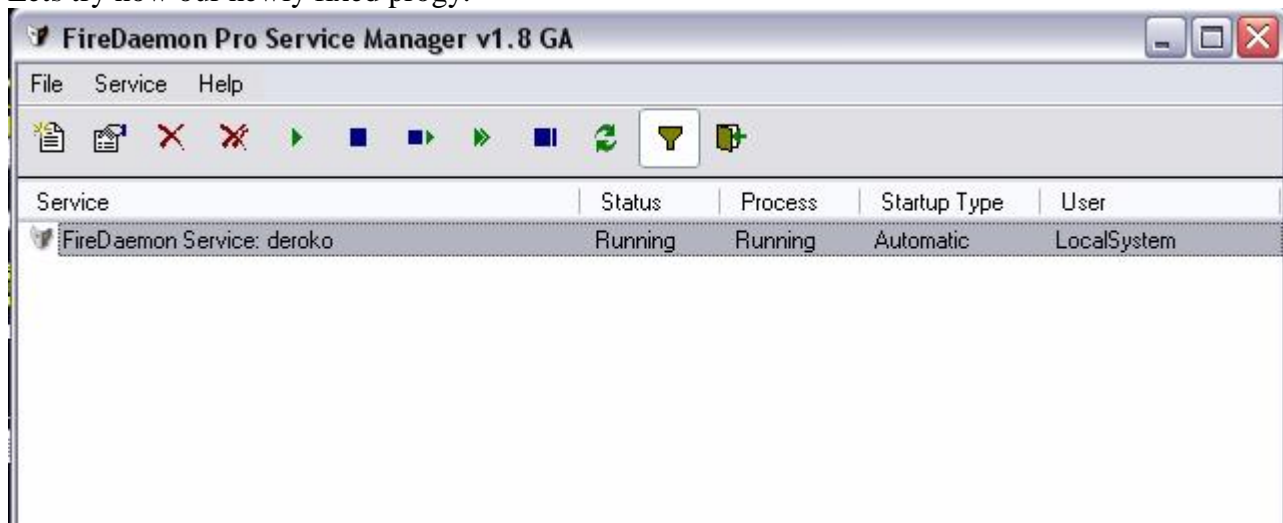
10010913	56	PUSH ESI	
10010914	FFD5	CALL EBP	
10010916	A1 5C64E600	MOV EAX,DWORD PTR DS:[E6645C]	
1001091B	83C0 20	ADD EAX,20	
1001091E	83C4 20	ADD ESP,20	
10010921	3BF0	CMP ESI,EAX	
10010923	74 0A	JE SHORT Core.1001092F	
10010925	56	PUSH ESI	
10010926	FF15 F8F00610	CALL DWORD PTR DS:[F1006F0F81]	MSI0R71 - close

Heh, some leftovers? No problemo we are gona code one more simple tool to fix those mov eax,[] , there is only 6-7 of them so no problemo here.

This is ms vc it has no embedded data in it, and all are nice and clean opcodes so no problemo with running lde trough it and scanning for 0xA1 with addr in eliminated iat section.

Code is included [eax\eax.asm] and there is not much to explain about it:D It will scan dumped file for mov eax, [IAT] and fix them to point to .adata section :D

Lets try now our newly fixed proggy:



Done, we nailed this bitch...

## 5. References

- § <http://www.reversing.be/article.php?story=20050926230916418> – haggar's way to fix code splices, easier way.
- § <http://cracking.accessroot.com> – anything more to say?
- § <http://vx.netlux.org/29a/> - 29a e-zine

## 6. Greetings

I wish to tank all the ARTeam members for sharing their knowledge, and to 29a for great e-zine.



<http://cracking.accessroot.com>