

Inline patching ActiveMark v6.3.x/6.6.x

November 2010

1. Introduction

This document will focus on the changes relative to inline patching the latest releases of ActiveMark v6.6x. It emerges from the latest researches of our teammate Condzero. Because his new findings were easily applicable to the “old” ActiveMark v6.3.x, I decided to code a tool which will help you generating binary strings for inline patching. Such tool was once created by another ARTeamer, Nieylana, covering only AM v6.3. ARTeam’s ActiveMark Inliner v1.0 covers both ActiveMark versions (6.3x and 6.6x) and also two protection “flavors” of ActiveMark v6.6x which are depending on ActiveMark options used during protection process.

To dump an ActiveMark target you can still use Condzero’s ActiveMark v2.2, which still works ok for most of the targets. Some of the targets protected with ActiveMark v6.6x are not using CPUID (machine tied protection) but I strongly recommend to patch those NOP’s at OEP with MOV DWORD PTR DS:[XXXXXXXX],-1 anyway in order to avoid unpleasant surprises later.

As for the tool, your task will be to find appropriate hexadecimal values and enter them to the boxes. When all values are in place, just press GENERATE button and your inline patch binary string is displayed and ready to be copied.

Yes, OK, but how to find those hexadecimal values ? Well, that is the scope of this tutorial, so read further and you’ll find out ... ☺☺☺

Regards, SSLEVIN

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Disclaimers

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

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

Verification

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

Table of Contents

Verification	2
1. Inline patching ActiveMark v6.3x/6.6x.....	3
1.1. Abstract	3
1.2. Targets.....	3
1.3. Tools used.....	4
2. Inline patch making using ActiveMark Inliner v1.0.....	4
2.1. AM v6.3.x (Old Inline template).....	5
2.2. AM 6.6.x (Set_Timer template).....	13
2.3. AM 6.6.x (Create_Thread template).....	17
3. Conclusions.....	21
4. Greetings.....	21

1. Inline patching ActiveMark v6.3x/6.6x

1.1. Abstract

If you have no previous experience in inline patching ActiveMark protected targets, we encourage you to read/watch our previous tutorials about the subject which can be found in the Tutorials section of ARTeam site.

If you do, then you are already familiar with terms like code cave, main jump, API hooks etc.etc. In this part of general considerations, I will only quote the sentence from one of the previous Condzero's tutorials: "The most important thing I can tell you is the Stack is your friend." All values needed for inlining can be found by setting HWBP's at certain API's and then browsing the stack for returning values.

1.2. Targets

Template used for making the tool, ARTeam's ActiveMark Inliner, is based on target "The Clockwork Man 2 Premium Edition" downloadable here:

http://d.trymedia.com/dd/gh_tot/60m_d/csc_gh_tot/TheClockworkMan2PE.exe

Default values in AM v6.6.x (SetTimer Inline template) will produce valid binary string for this target, but this is a huge download (357MB) so I decided to show tool's functionality on smaller targets.

So, target for practicing Old_Inline template will be this one: "Marooned" downloadable here:
http://d.trymedia.com/dd/gamehouse/60m_d/trygames/Marooned.exe (size 47 MB)

Target for practicing Set_Timer Inline template will be this one: "AvenueFlo" downloadable here:
http://d.trymedia.com/dd/playfirst/60m_d/csc_playfirst/AvenueFlo_Setup.exe (size 72MB)

Target for practicing Create_Thread Inline template will be this one: "World Riddles – Seven Wonders" downloadable here:
http://d.trymedia.com/dd/funnybear/60m_d/csc_funnybear/WorldRiddlesSevenWonders.exe
(size 28MB)

With targets defined let's get to some work.

1.3. Tools used

- Protection ID v6.4.0 (PiD Team)
- AM Dumper v2.2 (Condzero/Arteam)
- Olly debugger v1.10 (Oleg Yuschuk)
- Arteam's ActiveMark Inliner v1.0 (SSIEvIN/ARTeam)
- Some brain as usual ...

2. Inline patch making using ActiveMark Inliner v1.0

Before you dump your ActiveMark protected target always scan it with Protection ID. Some of you will probably ask: “Why, we already know its AM, so whats the big deal ?” The answer is simple: AM version. I.E. if your PID shows its AM v6.3.xyz you can dump/fix your file, load it in Olly, fire up Inliner, choose first template (Old_Inline) and start entering values “on the fly” as you find them. In the case of AM v6.6.xyz you first have to write down all the values you find and then to choose appropriate template depending on what protection option is used – SetTimer or CreateThread.



FIGURE 1. Main screen.

Introducing AM Inliner GUI in the FIGURE 1.

2.1. AM 6.3.x (Old Inline template)

Old_Inline

Code Cave: 01D95000 Browser Call: 00D48EFF

Main Jump: 014FA014 Call After SetTimer Pushed: 00BDED9F

Netbios API: 00F3E4DC End Nag Call: 00C02A38

FindFirstFileA API: 00F3E2E0 SetTimer Original Call Bytes: E82295F8

CreateWindowExA API: 00F3E60C SetTimer Original Call 5th Byte: FF

DestroyWindow API: 00F3E608

Expiration Check: 00D7E921

> Cycle thru boxes using TAB key and click GENERATE when all values entered <

Generate

FIGURE 2. Old inline template.

Ok, I hope you have our target for this type of inline (“Marooned”) dumped and that you fixed that DWORD holding value of -1 in order to fool GetTickCount. If you do then your target should look like this when opened in Olly (GetTickCount line marked red is assembled instead of NOPs AMDumper left for you):

00A930BC	64:A1 30000000	MOV EAX, DWORD PTR FS:[30]
00A930C2	A3 202C8600	MOV DWORD PTR DS:[862C20], EAX
00A930C7	A3 ECB88700	MOV DWORD PTR DS:[87B8EC], EAX
00A930CC	A3 28CD8700	MOV DWORD PTR DS:[87CD28], EAX
00A930D1	A3 34678800	MOV DWORD PTR DS:[886734], EAX
00A930D6	C705 34BA8700 FFFFFFFF	MOV DWORD PTR DS:[87BA34], -1
00A930E0	33C0	XOR EAX, EAX
00A930E2	0FA2	CPUID
00A930E4	8915 2CB28800	MOV DWORD PTR DS:[88B22C], EDX
00A930EA	68 C9F98D00	PUSH 8DF9C9
00A930EF	C3	RETN

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

If you have problem fixing this check our previous tutorials. Lets continue. Trace all those lines with F8 and when you get to (in this case) 8DF9C9 scroll down the code until this line below (marked red again) set a HWBP there and press run (F9).

```
008DFB34  64:A3 00000000      MOV  DWORD PTR FS:[0], EAX
008DFB3A  66:61              POPA
008DFB3C  FF25 14F08D00      JMP  NEAR DWORD PTR DS:[8DF014]
008DFB42  33C0              XOR  EAX, EAX
008DFB44  5F                POP  EDI
```

This is where our adventure begins. Instead of that red line we will assemble our jump to code cave later.

CodeCave: a place where we will place our inline patch. For many reasons last .bss section is the best choice (see previous tutorials). When choosing VA for the code cave start address I tend to use virtual addresses ending with zero/zeroes for easier orientation, but Inliner calculates all addresses necessary so you can choose literally any virtual address. You only have to pay attention there is at least 306 (132h) bytes of space (zeroes), in this case of course, because length of inline patch differs in other two templates. In my inline patch made for this tutorial I took **00ABD000** to be my code cave start address.

Main Jump: it's a DWORD pointer used for jumping from so called Layer 2 to Layer 3 of ActiveMark. In this case you see that value above – its **008DF014**.

Sidenote: in some rare targets where Layer 2 does not exist (targets with three .text sections instead of usual four – read Condzero's tutorials about targets with and without delayed imports) you won't find **JMP NEAR DWORD PTR DS:[0XXX014]** so you'll have to use that PUSH/RETN combo from the dumped target entry point to jump to code cave and then again use PUSH/RETN to jump to layer three.

Primer is the target called Hidato Adventure:

```
00905064      MOV  EAX, DWORD PTR FS:[30]
0090506A      MOV  DWORD PTR DS:[897D24], EAX
0090506F      MOV  DWORD PTR DS:[8A149C], EAX
00905074      MOV  DWORD PTR DS:[8B1B53], EAX
00905079      MOV  DWORD PTR DS:[8C307B], EAX
0090507E      MOV  DWORD PTR DS:[8ED5C4], -1
00905088      XOR  EAX, EAX
0090508A      CPUID
0090508C      MOV  DWORD PTR DS:[8B83CA], EDX
00905092      PUSH  92FD00      prior to making inline patch this was PUSH  6A7E40
00905097      RETN            (PUSH/RETN combo == unconditional JMP – jump)
```

(When this is executed we are in our code cave where we have this).....

```
0092FD00      PUSHAD      CodeCave start address
0092FD01      PUSHFD
0092FD02      MOV  EAX, DWORD PTR DS:[8F8424]
0092FD07      MOV  DWORD PTR DS:[92FCF0], EAX
0092FD0C      MOV  DWORD PTR DS:[8F8424], 92FD1E
0092FD16      POPFD
0092FD17      POPAD
```

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

0092FD18	PUSH 6A7E40	PUSH/RETN is redirecting code execution back to layer 3
0092FD1D	RETN	until next API is hooked ...
0092FD1E	PUSHAD	
0092FD1F	PUSHFD	
.....		
.....		

(Assembled commands **JMP NEAR DWORD PTR DS:[0XXX014]** and **PUSH 0XXXXXXXXX/RETN** occupy the same number of bytes so there will not be further inline patch reparations after switching from JMP to PUSH/RETN.

API Hooks:

For Old Inline Template you will need those Windows API to hook

1. NETAPI32.Netbios – for our target – Marooned – this API is at VA **00890454**. No alternative API's needed.
2. kernel32.FindFirstFileA - for our target – Marooned – this API is at VA **00890278**. No alternative API's needed.
3. USER32.CreateWindowExA - for our target – Marooned – this API is at VA **008905D0**. No alternative API's needed.
4. USER32.DestroyWindow - for our target – Marooned – this API is at VA **0089060C**. For this target you don't need alternative API's to catch and prevent endnag call to execute and generate that annoying “Buy before your trial period expires” window, but for some other targets you also might try to use kernel32.GetTickCount or kernel32.ReleaseMutex as alternative. If none of these works you have to find alternative for USER32.DestroyWindow on your own.

The only API other than those mentioned above you need is USER32.SetTimer in order to find where it is pushed at stack, so write down its VA also (**00890778** for this target). But, lets go on ...

To find all these you can use either “Search for all intermodular calls” after hitting HWBP at 008DFB3C or you can press M (memory map) and follow this:

Address=00890000
Size=00004000 (16384.)
Owner=dumped1 00400000
Section=.idata
Type=Imag 01001004
Access=RW
Initial access=RWE

in dump window (right-click > Dump in DSM) and enable Long address type of view, then browse through API's. Choice is yours.

Expiration check: when you hit your HWBP at

008DFB3C - FF25 14F08D00 JMP NEAR DWORD PTR DS:[8DF014]

press Ctrl+G (Go to), enter CreateThread in the Olly's box and press OK.

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Set HWBP at first bytes of CreateThread (7C8106D7 MOV EDI, EDI) and press Run (F9). When HWBP hit remove it. Top of your stack window should look like this:

0012F84C	5B89229C	C4LL to CreateThread from NETAPI32.5B892296
0012F850	00000000	pSecurity = NULL
0012F854	00000000	StackSize = 0
0012F858	5B891851	ThreadFunction = NETAPI32.5B891851
0012F85C	00000000	pThreadParm = NULL
0012F860	00000000	CreationFlags = 0
0012F864	0012F89C	pThreadId = 0012F89C

.....

Scroll way down the stack window until you find this:

0012FD68	007CE07E	RETURN to dumped1.007CE07E	follow this RETURN in disassembler
0012FD6C	01A526E0		
0012FD70	00000014		
0012FD74	83BF215F		
0012FD78	008413F4	RETURN to dumped1.008413F4	
0012FD7C	0086E41C	dumped1.0086E41C	
0012FD80	00000001		
0012FD84	0086E41C	dumped1.0086E41C	
0012FD88	01A526E0		
0012FD8C	0012FDB4	Pointer to next S3H record	
0012FD90	0080F382	SE handler	
0012FD94	00000001		

.....

This is the first RETURN that has no “Return to XXXXXXXX from XXXXXXXX” just “Return to”. So follow it in disassembler and you’ll end up here:

007CE07E	EB 03	JMP SHORT 007CE083
----------	-------	--------------------

Set HWBP there and run (F9). When it breaks remove HWBP and trace until you hit first TEST AL,AL at

007CE083	84 C0	TEST AL, AL
----------	-------	-------------

Check content of EAX and you will see its 0012F801. If you remember my previous tuts this is “visual aid” that you are on the right track. Set HWBP there and enter **007CE083** in the Inliner as a value for Expiration check. Hit F9 three times, untill you see “ERROR_ENVVAR_NOT_FOUND ” as LastErr message in Olly’s Registers window on the right.

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Browser call: press Ctrl+G again and enter CreateWindowExA in the box. Set HWBP at first bytes of this API and run (F9). When hit, remove HWBP and look at your stack window again. It will look similar to this:

```
0012F034      0073C46D   C4LL to CreateWindowExA from dumped1.0073C467
0012F038      00040100   ExtStyle = WS_EX_WINDOWEDGE|WS_EX_APPWINDOW
0012F03C      0000C29D   Class = C29D
0012F040      0012F0C8   WindowName = ""
0012F044      06CF0000   Style =
                WS_OVERLAPPED|WS_MINIMIZEBOX|WS_MAXIMIZEBOX|WS_CLIPSIBLINGS|WS_CLIPCHILDREN|WS_SYSM
                ENU|WS_THICKFRAME|WS_CAPTION
0012F048      00000000   X = 0
0012F04C      00000000   Y = 0
0012F050      00000320   Width = 320 (800.)
0012F054      0000028F   Height = 28F (655.)
0012F058      00000000   hParent = NULL
0012F05C      00000000   hMenu = NULL
0012F060      00400000   hInst = 00400000
0012F064      01B86738   lParam = 01B86738
```

Scroll again way down the stack window until you find this:

```
0012F54C      0012F628   Pointer to next S3H record
0012F550      00818542   SE handler
0012F554      00000000
0012F558      0012F5EC
0012F55C      0078B45A   RETURN to dumped1.0078B45A from dumped1.007AFFCD
0012F560      01B86738   ASCII
"http://amlocalhost.trymedia.com/offline/0086e73c/open.html?currenttime=2010/11/18@18:04:58&pool
time=2010/11/18@18:04:56&license_ts=0&contentid=44364b5669821e520dba6c2fcb48a7c5&offering=6
0m_d&affiliate=trygames&file=dumped1.exe&contentid=..."
0012F564      00000001
0012F568      0079EB24   RETURN to dumped1.0079EB24
0012F56C      01B86738   ASCII
"http://amlocalhost.trymedia.com/offline/0086e73c/open.html?currenttime=2010/11/18@18:04:58&pool
time=2010/11/18@18:04:56&license_ts=0&contentid=44364b5669821e520dba6c2fcb48a7c5&offering=6
0m_d&affiliate=trygames&file=dumped1.exe&contentid=..."
0012F570      00000001
0012F574      00000000
0012F578      00000000
0012F57C      0086E418   dumped1.0086E418
```

Follow the line marked red in disassemble. It is again a call from “nowhere” (no “from”, just “to”). You’ll end up at this line in disassembler:

```
0079EB24  E9 DE000000      JMP  0079EC07
```

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Check the line above it:

```
0079EB21 . FF50 04          CALL NEAR DWORD PTR DS:[EAX+4]
```

Rings a bell ? ...he,he ...sure it does ... Exactly ... that's the call to Browser screen as used in ActiveMark v6.3.x. So you can enter **0079EB21** as value for Browser call in the Inliner tool, set a HWBP at **0079EB21**, restart application and run till this last HWBP is hit. Elegant, eehh ?

Call after USER32.SetTimer is pushed at stack: time to use the address of SetTimer we wrote down before. If you remember it was **00890778** for this target. So, in disassemble window press Ctrl+B(Search for binary string) and enter **FF 35 78 07 89 00** in the last box (HEX). Note for newbies: **FF 35** are mnemonics for **PUSH DWORD PTR DS:[some address]** command and **78 07 89 00** is SetTimer's VA in reverse order. Press OK and you'll end up here:

```
007E9199    FF35 78078900    PUSH  DWORD PTR DS:[890778]    ; USER32.SetTimer
007E919F    E8 2BB3FDFF    CALL  007C44CF    ; dumped1.007C44CF
007E91A4    85C0          TEST  EAX, EAX
007E91A6    0F84 EFFFFFFF    JE    007E909B    ; dumped1.007E909B
```

Set a HWBP at the line marked red and run (F9).

(Note: if you can't clearly see the call bellow SetTimer pushed address either use Analyse this! (Olly plugin) or follow E8h (mnemonic for call) in dump window and set HWBP at it there.)

When Browser window appears, press "Start Free Trial" button. Application will break at our breakpoint (**007E919F**). Now copy the whole line to clipboard and paste to notepad.

```
007E919F    E8 2BB3FDFF    CALL  007C44CF    ; dumped1.007C44CF
```

Enter **007E919F** as value for Call After SetTimerPushed in the Inliner.

Endnag call: scroll way down the stack window again until you find this

```
0012F760    007D5FDD    RETURN to dumped1.007D5FDD from dumped1.00739BBF
0012F764    01B9BCB0
0012F768    0012F868
0012F76C    0082109C    ASCII "License"
0012F770    00821EBC    ASCII "entry"
0012F774    00000000
0012F778    00000006
0012F77C    4CE56282
```

Follow line marked red in disassembler. You should end up here:

```
007D5FDD    83C4 38      ADD  ESP, 38
```

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Because you probably won't see clearly the call above this line follow again E8h in dump and set a HWBP at it (till now all your HWBP slots – 4 of them, are probably used, so delete HWBP at Expiration check if you entered its VA in the Inliner). Restart the application and run (F9). You'll notice that this last HWBP set is hit immediately after first one you set at

```
008DFB3C  FF25 14F08D00          JMP  NEAR DWORD PTR DS:[8DF014]
```

So, your next stop is this:

```
007D5FD8  E8 E23BF6FF          CALL 00739BBF ; dumped1.00739BBF
```

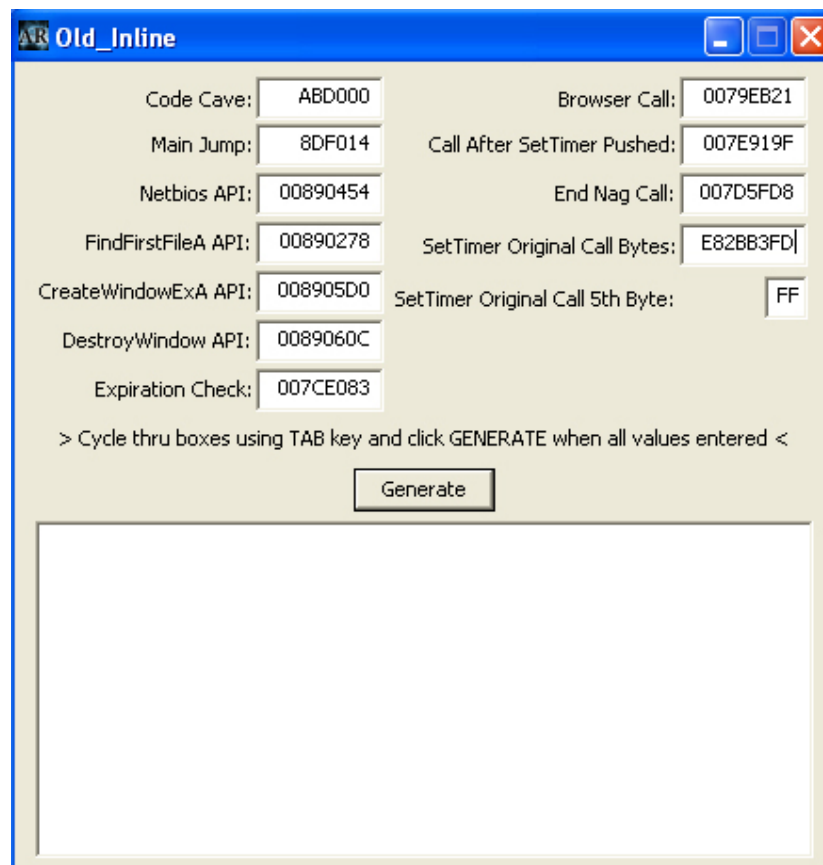
VA **007D5FD8** is your End Nag Call value. Enter it in the appropriate box of Inliner and you are done.

SetTimer Original Call bytes: back to our Call after SetTimer pushed HWBP.

```
007E919F  E8 2BB3FDFF          CALL 007C44CF ; dumped1.007C44CF
```

Enter first four bytes **E82BB3FD** to SetTimer Original Call bytes box of the Inliner. That's it.

SetTimer Original Call 5th byte: for this target ("Marooned") this value remains the same FFh. From target to target it can be either FF or 00. If you did everything right, your finished Old Inline template should look like in the picture below.



The screenshot shows the 'Old_Inline' application window with the following fields and values:

Field	Value
Code Cave:	ABD000
Main Jump:	8DF014
Netbios API:	00890454
FindFirstFileA API:	00890278
CreateWindowExA API:	008905D0
DestroyWindow API:	0089060C
Expiration Check:	007CE083
Browser Call:	0079EB21
Call After SetTimer Pushed:	007E919F
End Nag Call:	007D5FD8
SetTimer Original Call Bytes:	E82BB3FD
SetTimer Original Call 5th Byte:	FF

> Cycle thru boxes using TAB key and click GENERATE when all values entered <

Generate

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Press Generate button and your inline patch binary string is displayed in that big box below like so:

```
60 9C A1 54 04 89 00 A3 F0 CF AB 00 C7 05 54 04 89 00 1E D0 AB 00 9D 61 FF 25 14 F0 8D 00 60 9C A1 F0 CF AB 00
A3 54 04 89 00 C6 05 83 E0 7C 00 32 A1 78 02 89 00 A3 F0 CF AB 00 C7 05 78 02 89 00 4D D0 AB 00 9D 61 FF 25 54
04 89 00 60 9C A1 F0 CF AB 00 A3 78 02 89 00 C6 05 83 E0 7C 00 84 66 C7 05 21 EB 79 00 90 90 C6 05 23 EB 79 00 90
A1 D0 05 89 00 A3 F0 CF AB 00 C7 05 D0 05 89 00 8C D0 AB 00 9D 61 FF 25 78 02 89 00 60 9C A1 F0 CF AB 00 A3 D0
05 89 00 66 C7 05 21 EB 79 00 FF 50 C6 05 23 EB 79 00 04 C7 05 9F 91 7E 00 58 58 58 59 C6 05 A3 91 7E 00 59 A1 78
02 89 00 A3 F0 CF AB 00 C7 05 78 02 89 00 D5 D0 AB 00 9D 61 FF 25 D0 05 89 00 60 9C A1 F0 CF AB 00 A3 78 02 89
00 C7 05 9F 91 7E 00 E8 2B B3 FD C6 05 A3 91 7E 00 FF A1 0C 06 89 00 A3 F0 CF AB 00 C7 05 0C 06 89 00 0E D1 AB
00 9D 61 FF 25 78 02 89 00 60 9C A1 F0 CF AB 00 A3 0C 06 89 00 C7 05 D8 5F 7D 00 B8 01 00 00 C6 05 DC 5F 7D 00
00 9D 61 FF 25 0C 06 89 00
```

I hope you know the rest of the procedure. If not, then ☺☺☺ ...

Reload dumped/fixed game in Olly and go to **008DFB3C**.

Replace

```
008DFB3C    JMP     NEAR DWORD PTR DS:[8DF014]
```

with

```
008DFB3C    JMP     00ABD000
008DFB41    NOP
```

Then copy that string above and binary paste it at 00ABD000. Right-click > Copy to executable > Save file under another name. Test your patched file. Renaming to original usually is not necessary except for some Flash based games.



2.2. AM 6.6.x (Set Timer template)

Code Cave:	01D95000	Browser Call:	00D48EFF
Main Jump:	014FA014	Call After SetTimer Pushed:	00BDED9F
Netbios API:	00F3E4DC	End Nag Call:	00C02A38
FindFirstFileA API:	00F3E2E0	Browser Original Call Bytes:	E86041FC
CreateWindowExA API:	00F3E60C	SetTimer Original Call Bytes:	E82295F8
DestroyWindow API:	00F3E608	Browser Original Call 5th Byte:	FF
Expiration Check:	00D7E921	SetTimer Original Call 5th Byte:	FF

> Cycle thru boxes using TAB key and click GENERATE when all values entered <

Generate

FIGURE 3. SetTimer inline template.

Ok, I hope you have our target for this type of inline (“AvenueFlo”) dumped/fixed the same way as explained for the previous one. In order to spare some time repeating the same things again and again, I decided to give explanations for this template only when there are some procedural differences between this and Old Inline template. If there is no difference only the value needed for the Inliner will be displayed.

CodeCave: its again in last .bss section of the executable and I choosed **00FD8000**. Choice is totally up to you.

Main Jump: is found in the same manner as above and value for this target is **00C9E014**.

API Hooks: are the same ones as above and values for “Avenue Flo” are these

1. NETAPI32.Netbios VA: **00C6F47C**
2. kernel32.FindFirstFileA VA: **00C6F284**
3. USER32.CreateWindowExA VA: **00C6F598**
4. USER32.DestroyWindow VA: **00C6F594**

API also needed is again USER32.SetTimer and for this target its VA is **00C6F6DC**.

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Expiration check: follow the same procedure as in Old Inline template. In this case your “call from nowhere” return is located here in the stack window when CreateThread HWBP is hit.

0012FD58	008A6062	RETURN to dumped_.008A6062	follow this in disassembler
0012FD5C	01792808		
0012FD60	00000014		

Follow in disassembler leads here:

```
008A6062  7E 01          JLE  SHORT 008A6065          ; dumped_.008A6065
```

Trace (F8) and your value for expiration check is this TEST AL,AL

```
008A6083  84 C0          TEST  AL, AL
```

Enter **008A6083** in the Inliner for Expiration check. Repeat same steps as above and notice that EAX value is different > **A7EFA501** but AL is still **01** like in previous version.

Browser call: procedure for finding the right value is the same as in Old Inline template, but the difference is that this time we have the call “from somewhere”. Look for the return marked red way down the stack window:

0012F4D4	0012F5AC	Pointer to next S3H record	
0012F4D8	00B58E5B	SE handler	
0012F4DC	00000000		
0012F4E0	0089DF27	RETURN to dumped_.0089DF27 from dumped_.00A321E5	follow this in disass.
0012F4E4	0199A6B8	ASCII	
"http://amlocalhost.trymedia.com/offline/00c633c8/open.html?currenttime=2010/11/18@21:59:27&pooltime=2010/11/18@21:59:26&license_ts=0&contentid=8d8ebe9d81c8a66c455fe2a2d98f2ee9&offering=60m_d&affiliate=csc_playfirst&file=dumped_.exe&track"...			
0012F4E8	0012F401		

Following the Return you’ll land at RETN 10 and the call above is our browser call:

0089DF20	50	PUSH EAX	
0089DF21	52	PUSH EDX	
0089DF22	E8 BE421900	CALL 00A321E5	BROWSER CALL
0089DF27	C2 1000	RETN 10	RETURN lands here
0089DF2A	EB 02	JMP SHORT 0089DF2E	; dumped_.0089DF2E

So, enter **0089DF22** for Browser call value, **E8BE4219** for first four Browser Original Call bytes, and **00** for Browser Original Call 5th byte. Repeat all steps as in Old Inline template.

Call after USER32.SetTimer is pushed at stack: procedure is the same. We search for binary string **FF 35 DC F6 C6 00** (SetTimer, see above). We land here:

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

```
00AFF3ED      PUSH  DWORD PTR DS:[C6F6DC]      ; USER32.SetTimer
00AFF3F3      JMP   SHORT 00AFF3F9                  ; dumped_.00AFF3F9
00AFF3F5      DB    29                          ; CHAR ')'
00AFF3F6      DB    FF
00AFF3F7      DB    05
00AFF3F8      IN     AL, DX
00AFF3F9      JMP   SHORT 00AFF3FD                  ; dumped_.00AFF3FD
00AFF3FB      DB    81
00AFF3FC      DB    EB
00AFF3FD      E8 60C9E7FF  CALL  0097BD62              Call we are looking for
00AFF402      PUSH  EBX
```

So, enter **00AFF3FD** for SetTimer call value, **E860C9E7** for first four SetTimer Original Call bytes, and **FF** for SetTimer Original Call 5th byte.

SIDENOTE: in these two calls (Browser and SetTimer) lies the main difference between the two inline methods – Old Inline and SetTimer Inline. In old inline we were able to just NOP these two calls and restore the code back later. In SetTimer Inline, as well as in CreateThread Inline, we cant NOP the calls, because in order to continue we have to “repair” the stack by POPing the values from it. Luckily, the Inliner takes care of this, and you can check this when testing finished inline patch by seting HWBP’s at these calls then tracing to see what has been POPed from stack and why. 😊😊😊😊😊

Endnag call: look again for the same pattern (“License entry”) in the stack window and follow the return in disassembler. You land here:

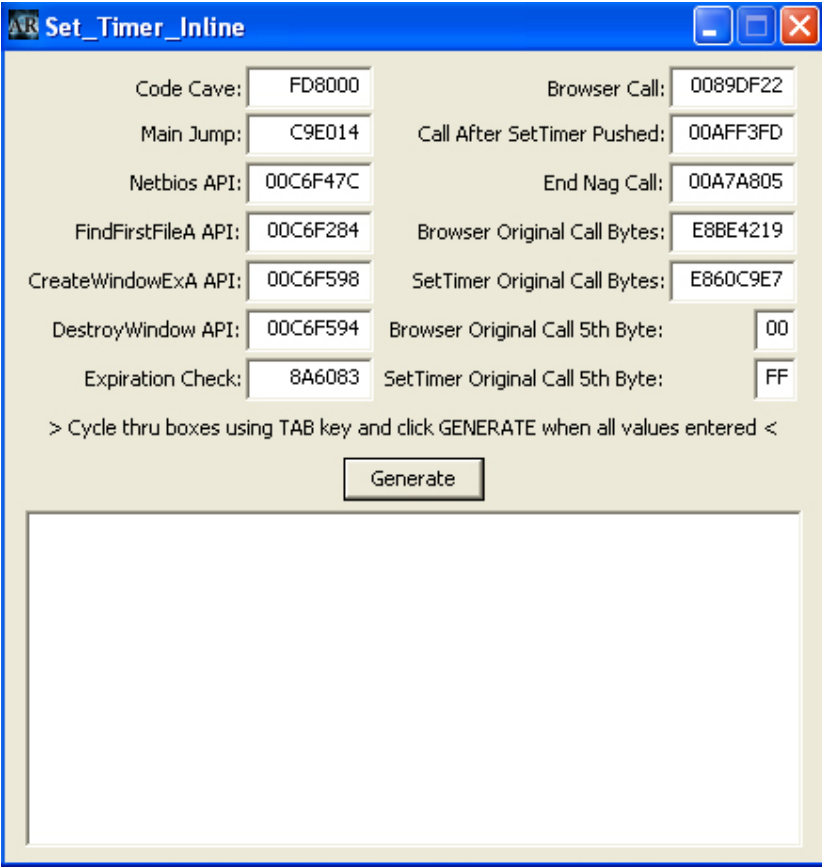
```
00A7A805      E8 E616D8FF      CALL  007FBEF0      End Nag Call
00A7A80A      EB 03            JMP   SHORT 00A7A80F  Landing here
```

So, enter 00A7A805 as value for End Nag Call and you are done with entering values.



INLINE PATCHING ACTIVEMARK V6.3X/6.6X

If you did everything right your SetTimer inline template window should look like in the picture below:



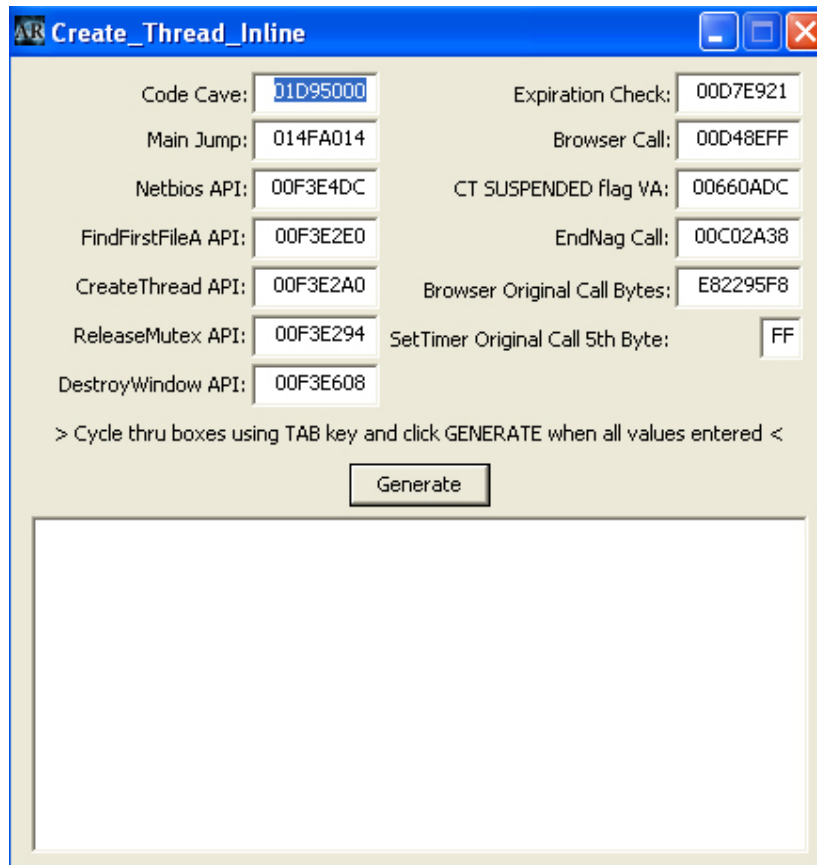
The screenshot shows a window titled "Set_Timer_Inline" with a blue title bar. Inside, there are two columns of text boxes for inputting addresses. The left column includes: Code Cave (FD8000), Main Jump (C9E014), Netbios API (00C6F47C), FindFirstFileA API (00C6F284), CreateWindowExA API (00C6F598), DestroyWindow API (00C6F594), and Expiration Check (8A6083). The right column includes: Browser Call (0089DF22), Call After SetTimer Pushed (00AFF3FD), End Nag Call (00A7A805), Browser Original Call Bytes (E8BE4219), SetTimer Original Call Bytes (E860C9E7), Browser Original Call 5th Byte (00), and SetTimer Original Call 5th Byte (FF). Below these fields is a line of text: "> Cycle thru boxes using TAB key and click GENERATE when all values entered <". At the bottom center is a "Generate" button. Below the button is a large, empty rectangular box for the output.

Press Generate button and your inline patch binary string is displayed in that big box below like so:

```
60 9C A1 7C F4 C6 00 A3 F0 7F FD 00 C7 05 7C F4 C6 00 1E 80 FD 00 9D 61 FF 25 14 E0 C9 00 60 9C A1 F0 7F FD 00 A3
7C F4 C6 00 C6 05 83 60 8A 00 32 A1 84 F2 C6 00 A3 F0 7F FD 00 C7 05 84 F2 C6 00 4D 80 FD 00 9D 61 FF 25 7C F4 C6
00 60 9C A1 F0 7F FD 00 A3 84 F2 C6 00 C6 05 83 60 8A 00 84 C7 05 22 DF 89 00 58 58 58 58 C6 05 26 DF 89 00 40 A1
98 F5 C6 00 A3 F0 7F FD 00 C7 05 98 F5 C6 00 8D 80 FD 00 9D 61 FF 25 84 F2 C6 00 60 9C A1 F0 7F FD 00 A3 98 F5 C6
00 C7 05 22 DF 89 00 E8 BE 42 19 C6 05 26 DF 89 00 00 C7 05 FD F3 AF 00 58 58 58 59 C6 05 01 F4 AF 00 59 A1 84 F2
C6 00 A3 F0 7F FD 00 C7 05 84 F2 C6 00 D7 80 FD 00 9D 61 FF 25 98 F5 C6 00 60 9C A1 F0 7F FD 00 A3 84 F2 C6 00 C7
05 FD F3 AF 00 E8 60 C9 E7 C6 05 01 F4 AF 00 FF A1 94 F5 C6 00 A3 F0 7F FD 00 C7 05 94 F5 C6 00 10 81 FD 00 9D 61
FF 25 84 F2 C6 00 60 9C A1 F0 7F FD 00 A3 94 F5 C6 00 C7 05 05 A8 A7 00 B8 01 00 00 C6 05 09 A8 A7 00 00 9D 61 FF
25 94 F5 C6 00
```

I think you already guess the rest of the job to be done ...

2.3. AM 6.6.x (CreateThread template)



The screenshot shows a window titled "Create_Thread_Inline" with a blue title bar. Inside, there are two columns of text boxes for entering hexadecimal addresses. The left column contains: Code Cave (01D95000), Main Jump (014FA014), Netbios API (00F3E4DC), FindFirstFileA API (00F3E2E0), CreateThread API (00F3E2A0), ReleaseMutex API (00F3E294), and DestroyWindow API (00F3E608). The right column contains: Expiration Check (00D7E921), Browser Call (00D48EFF), CT SUSPENDED flag VA (00660ADC), EndNag Call (00C02A38), Browser Original Call Bytes (E82295F8), and SetTimer Original Call 5th Byte (FF). Below these fields is a line of text: "> Cycle thru boxes using TAB key and click GENERATE when all values entered <". At the bottom center is a "Generate" button. Below the button is a large empty rectangular box.

FIGURE 4. CreateThread inline template.

Ok, in this last target “World Riddles – Seven Wonders” different protection option of ActiveMark is used. Namely, you WILL find the call after USER32.SetTimer is pushed at stack. You can even set HWBP at that call but unfortunately it will never trigger out (code is there, compiled, but never gets executed).

And this is the point where great Arteam’s ActiveMark Master, Condzero, enters the scene. He found that other protection option used.

I will try to reproduce his explanation:

In this “flavor” of AM protection immediately after you press “Start Free Trial” button a thread is created via kernel32.CreateMutexA as a thread synchronization measure. This Call to CreateThread is creating a sleeper thread, that when it wakes up after the time duration, goes back to the program entry to recheck the time left (The ThreadFunction is the function that calls the kernel32.Sleep API with the value 60000ms or 60 secs).

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Fun fact that protection developers overlooked is that ActiveMark creates this thread as active but doesn't care about its state after creation. It's initial creation flag is 0 (zero) and easiest way to bypass this was to catch its code before creation and set its initial flag to 4 (which equals to CREATE_SUSPENDED). This way, it still exists but has no influence (does not check the program entry for the time left) due to its suspended state.

Since finding other values for this target is identical as in previous templates, I will only exhaustively enumerate other values and explain how to find this VA for changing thread creation flag in details.

CodeCave: I took **00A9E000**.

Main Jump: **008BA014**.

API hooks:

- | | | |
|--------------------------|-----------------|------------------------------|
| • Netbios VA | 00863544 | |
| • FindFirstFileA VA | 00863374 | |
| • CreateThread VA | 00863334 | one extra API to use as HOOK |
| • ReleaseMutex VA | 00863328 | |
| • DestroyWindow VA | 00863684 | |

Expiration check: its virtual address is **00690451**

Browser call: its virtual address is **0061BB9C**

End Nag Call: its virtual address is **0067375C**

Browser Original Call bytes: are **E8C7CD02**

Browser Original Call 5th byte: is **00**

CT(CreateThread) SUSPENDED flag VA: ok, lets finish the story about finding values (virtual addresses) for inlining. For this one do the following:

Run the target until Browser Call HWBP hit. Then press run once again so the browser screen appears. Back to Olly, press Ctrl+G (Go to), enter CreateThread in the box and press OK. Set HWBP at first bytes of CreateThread. Press "Start Free Trial" button in browser screen. CreateThread HWBP is hit many times, but we are looking for this one in stack window:

0012F47C	006D42D7	C4LL to CreateThread from dumped1.006D42D2
0012F480	00000000	pSecurity = NULL
0012F484	00000000	StackSize = 0
0012F488	006E86E2	ThreadFunction = dumped1.006E86E2
0012F48C	014190C0	pThreadParm = 014190C0
0012F490	00000000	CreationFlags = 0
0012F494	014190D0	pThreadId = 014190D0

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

See the ThreadFunction and Creation Flags marked red. Now we need to find where from was all this pushed on the stack. So, select ThreadFunction in stack window and follow it in disassembler. You will land here:

```

006E86E2  56                PUSH  ESI
006E86E3  8B7424 08         MOV  ESI, DWORD PTR SS:[ESP+8]
006E86E7  8B46 04          MOV  EAX, DWORD PTR DS:[ESI+4]
006E86EA  69C0 E8030000     IMUL EAX, EAX, 3E8
006E86F0  50              PUSH  EAX
006E86F1  FF35 3C328600     PUSH DWORD PTR DS:[86323C] ; kernel32.Sleep
006E86F7  E8 26C2FFFF      CALL 006E4922 ; dumped1.006E4922

```

Notice how value of EAX (3Chex, 60dec) is multiplied with 3E8h == 1000dec. CPU “counts” time in milliseconds so 60 times 1000 equals 60000ms or 60 seconds. That interval is delivered to kernel32.Sleep function called below (marked red).

But this is just the proof we are on the right track, not the place to do any patching. Mind you we have to find the place in code where our ThreadFunction (ThreadFunction=dumped1.006E86E2) with all other thread parameters

Now, press Ctrl+B (Search for binary string) and enter **68** (mnemonic for PUSH) **E2 86 6E 00** (function bytes in reverse order).

We land here:

```

006D42C3  68 E2866E00      PUSH  6E86E2
006D42C8  6A 00            PUSH  0
006D42CA  6A 00            PUSH  0
006D42CC  FF35 34338600     PUSH DWORD PTR DS:[863334] ;kernel32.CreateThread
006D42D2  E8 4B060100      CALL 006E4922 ; dumped1.006E4922

```

Ok, cool, but where are other arguments passed to the CreateThread function ?

Analyse the code, and you will notice the arrow coming “from below” (see the picture)

Address	Hex dump	Disassembly
006D42B9	? 858F FEFFFFE9	TEST DWORD PTR DS:[EDI+E9FFFFFF], EAX
006D42BF	?- 79 FE	JNS SHORT 006D42BF
006D42C1	? FFFF	???
006D42C3	68 E2866E00	PUSH 6E86E2
006D42C8	6A 00	PUSH 0
006D42CA	6A 00	PUSH 0
006D42CC	FF35 34338600	PUSH DWORD PTR DS:[863334]
006D42D2	E8 4B060100	CALL 006E4922

006E86E2=dumped1.006E86E2
Jump from 006D4545

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

In Olly pane window you see that this jump comes from **006D545**. Go to that jump using right click
> Go to JMP 006D4545. Then in the same manner go to other jumps until you see this:

006D4520	8D46 10	LEA	EAX, DWORD PTR DS:[ESI+10]	
006D4523	50	PUSH	EAX	
006D4524	6A 00	PUSH	0	
006D4526	EB 05	JMP	SHORT 006D452D	; dumped1.006D452D
006D4528	EB	DB	EB	
006D4529	07	POP	ES	
006D452A	EB 04	JMP	SHORT 006D4530	; dumped1.006D4530

Marked in red are first two CreateThread arguments we couldn't see at first glance because of the code obfuscation. First one, PUSH EAX is pThreadID argument, and second one PUSH 0 represents CreationFlags argument. To achieve our goal – this thread to be created as suspended we have to assemble PUSH 4 at 006D4524. We are changing it from PUSH 0 and since second byte is changed our VA for CT(CreateThread) SUSPENDED flag is **006D4525**. Enter it in the appropriate box of the Inliner tool and you are done.

If you did everything right your Create_Thread Inline template should look like this:

Code Cave: A9E000 Expiration Check: 690451

Main Jump: 8BA014 Browser Call: 61BB9C

Netbios API: 863544 CT SUSPENDED flag VA: 6D4525

FindFirstFileA API: 863374 EndNag Call: 57375C

CreateThread API: 863334 Browser Original Call Bytes: E8C7CD02

ReleaseMutex API: 863328 SetTimer Original Call 5th Byte: 00

DestroyWindow API: 863684

> Cycle thru boxes using TAB key and click GENERATE when all values entered <

Generate

INLINE PATCHING ACTIVEMARK V6.3X/6.6X

Press Generate and your inline patch binary string for “World Riddles – Seven Wonders” is created

```
60 9C A1 44 35 86 00 A3 F0 DF A9 00 C7 05 44 35 86 00 1E E0 A9 00 9D 61 FF 25 14 A0 8B 00 60 9C
A1 F0 DF A9 00 A3 44 35 86 00 C6 05 51 04 69 00 32 A1 74 33 86 00 A3 F0 DF A9 00 C7 05 74 33 86
00 4D E0 A9 00 9D 61 FF 25 44 35 86 00 60 9C A1 F0 DF A9 00 A3 74 33 86 00 C6 05 51 04 69 00 84
C7 05 9C BB 61 00 58 58 58 58 C6 05 A0 BB 61 00 40 C6 05 25 45 6D 00 04 A1 34 33 86 00 A3 F0 DF
A9 00 C7 05 34 33 86 00 94 E0 A9 00 9D 61 FF 25 74 33 86 00 60 9C A1 F0 DF A9 00 A3 34 33 86 00
C7 05 9C BB 61 00 E8 C7 CD 02 C6 05 A0 BB 61 00 00 A1 28 33 86 00 A3 F0 DF A9 00 C7 05 28 33 86
00 CD E0 A9 00 9D 61 FF 25 34 33 86 00 60 9C A1 F0 DF A9 00 A3 28 33 86 00 C6 05 25 45 6D 00 00
A1 84 36 86 00 A3 F0 DF A9 00 C7 05 84 36 86 00 FC E0 A9 00 9D 61 FF 25 28 33 86 00 60 9C A1 F0
DF A9 00 A3 84 36 86 00 C7 05 5C 37 67 00 B8 01 00 00 C6 05 60 37 67 00 00 9D 61 FF 25 84 36 86
00
```

You can test it as described before ...

3. Conclusions

Through the history of this “cat and mouse” game (ActiveMark reversing) there were many different ways of inline patching it, but the basis remained the same ... Push registers and flags, hook certain API, assemble patch, pop flags and registers, run, ... push registers and flags, hook another API, restore original code, pop flags and registers, run ... etc ... etc...

What really has been improved is the way of finding values necessary for building inline patch. All credits for this go to my teammate, Condzero, who is a real AM wizard.

I am really satisfied that I made my modest contribution to the history of ActiveMark reversing by coding this tool, Arteam’s ActiveMark Inliner and presented its practical use in this tutorial. I also think it would be unfair, if I did not mention here my infinitely patient programming teacher, my other teammate, Ghandi, who introduced me to the world of C++ programming.

Cheers and thanks to both of ya, fellows !!!

4. Greetings

I wish to thank all the ARTeam members and of course you who, have read this tutorial and perhaps, can contribute something worthwhile to the RCE community.

