

Loading & Patching ActiveMark v6.x – PART II

December 2006

1. Introduction

In Part I, I introduced some of the basics in dumping an ActiveMark [AM] v6.x application. I covered some of the new features and obstacles that lead to the new Level 3 AM protection layer. So what else is new, one might ask? In short, plenty. ActiveMark claims to search the so-called underground message boards for information [Read: cracks] that, in their opinion, can pose a threat, undermine or jeopardize their protection system. I am quite convinced the [ARTEAM] forum is routinely searched for such information. While we don't provide cracks or illegally distribute software, we do provide information that can be abused. Why else would the new AM release have addressed most of the shortcomings of their previous releases in a very profound way. AM has done a full threat assessment of their protection system and has made a significant investment in this new release and it shows. They have increased their level of obfuscation and literal string encryption for the key functions that were addressed in previous Tutorials. Some functions have been rewritten or cleverly disguised to prevent their detection. Every major function is surrounded by a Call to GetTickCount to thwart debugging or more significantly, tracing a running process's code. They have introduced active memory scanning and check summing of specific blocks of code to prevent patching or modifications. All of the new features come at a price. In this cat and mouse game of protection systems, you get an extremely bloated executable that consumes vast amounts of resources, more crap being written to your registry and hard drive and the load times have increased. So this is progress. Welcome to the new release of AM. In this, Part II, I will discuss [3] key areas of functionality that can be exploited in this new protection and introduce a new tool, AMLOADV6, which can aid you in running, analyzing and patching your application.

Loading & Patching ActiveMark v6.x - Part II

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. Loading & Patching ActiveMark v6.x Applications, CondZero.....	3
1.1. Abstract.....	3
1.2. Target	3
1.3. Loading the target	3
1.4. Setup	3
1.4.1 Checking out the target.....	3
1.4.2 Starting Point.....	4
1.5. [3] Key areas of Interest	5
1.5.1 Expiration	5
1.5.2 Browser.....	8
1.5.3 Timer / Timeout	8
1.5.4 WriteFile (data files and Registry key updates) for trial usage.....	10
1.5.5 The AMLOADV6 (Loader / custom loader creation Tool)	11
1.6. References	11
1.7. Conclusions	11
1.8. Greetings.....	12

Loading & Patching ActiveMark v6.x - Part II

1. Loading & Patching ActiveMark v6.x Applications, CondZero.

1.1. Abstract

This Tutorial will highlight [3] key areas of functionality that can be exploited. I will introduce to you a method in loading and patching an ActiveMark v6.,x application so that you can further analyze the protection system and your application. Included is a new tool “AMLOADV6” which can aid you in loading and running an application and show you some important concepts in loaders. I will reference the same application as used in Part I. This way the principles you learn are cohesive and logical in their progression.

1.2. Target

The target is a game called Buku kakuro - 31.3 Mb - 6.1.342 AM release. You can get it at the link below:

http://d.trymedia.com/dd/merscom/60m_d/merscom/BukuKakuroSetup.exe

1.3. Loading the target

1.4. Setup

Tools used: OllyDbg v1.10, OllyAdvanced v1.26 Beta 10 for winnt, AMLOADV6 (New). Read Part I for the recommended options and settings for the tools above. You can either use the original executable or a dumped version, your choice. Make sure that your BaseOfCode section references the Level 3 beginning memory block. This will greatly assist you in referencing text literals, and analyzing code. I will concentrate on Level 3 as this is the protection system's main code layer and area of interest.

1.4.1 Checking out the target

We first open our target in Olly. I briefly discussed the [4] layers of code in Part I. See the [memory map] below:

Loading & Patching ActiveMark v6.x - Part II

00400000	00001000	_kakuro		PE header	
00401000	00149000	_kakuro	.text	code	Code
0054A000	00023000	_kakuro	.rdata		
0056D000	00015000	_kakuro	.data	data	
00582000	00002000	_kakuro	.rsrc		
00584000	0000A000	_kakuro	.reloc		
0058E000	000B5000	_kakuro	.text		Level 3
00643000	0003C000	_kakuro	.data		
0067F000	00010000	_kakuro	.bss		
0068F000	00003000	_kakuro	.idata		
00692000	00003000	_kakuro	.rsrc		
00695000	000DC000	_kakuro	.text		Level 2
00771000	00001000	_kakuro	.idata		
00772000	00002000	_kakuro	.rsrc	resources	
00774000	00015000	_kakuro	.text	SPX	Level 1
00789000	00006000	_kakuro	.bss		
0078F000	00009000	_kakuro	.data		
00798000	00001000	_kakuro	.idata	imports	

Figure 1.

1.4.2 Starting Point

We [you the reader and I] will be getting in the habit of using, setting and resetting many Hardware Breakpoints in this Tutorial. Why? Because using Software BP's has the same effect as directly modifying code and will produce the following:

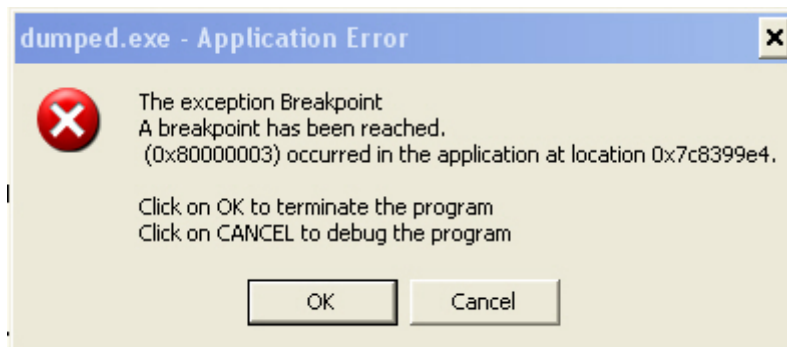


Figure 2.

We will start at the very 1st line of code in Layer 3. Set a HWBP at address 005C658D (shown below) as it will provide a suitable starting point.

005C658D	EB 02	JMP SHORT dumped.005C6591
005C658F	EB FF	JMP SHORT dumped.005C6590
005C6591	8925 B0CE6800	MOV DWORD PTR DS:[68CEB0],ESP
005C6597	60	PUSHAD
005C6598	8925 ACD76800	MOV DWORD PTR DS:[68D7AC],ESP
005C659E	EB 02	JMP SHORT dumped.005C65A2

Figure 3.

Loading & Patching ActiveMark v6.x - Part II

I am not going to go into the details on how-to use Olly's functionality that is referenced in this Tutorial. There are Basic Tutorials for that. So without further ado, let get into the heart of the matter.

1.5. [3] Key areas of Interest

The following sections are sequenced in the order in which AM processes the functions.

1.5.1 Expiration

The venerable Expiration condition lives on in the new release. Gone is "LoadStatePool" although SetKey exists, but it is more cleverly integrated in the AM infrastructure and not so clearly understood. There have been many ideas promoted on how-to best find the TEST AL,AL condition. I will show you a good way. Using Olly's context Search for >> All referenced text strings. Search for text the following Ascii Value shown below:

00602E8A	PUSH dumped.00644B14	ASCII "Implemented Categories"
----------	----------------------	--------------------------------

Figure 4.

This value can appear more than once. If so, set a HWBP on execute for every occurrence in the code. Run (F9) the target and we break at the line below:

00602E8A	68 144B6400	PUSH dumped.00644B14	ASCII "Implemented Categories"
00602E8F	50	PUSH EAX	
00602E90	E8 4B9DFBFF	CALL dumped.005BCBE0	
00602E95	85C0	TEST EAX,EAX	

Figure 5.

Now search the Stack window scrolling down many, many lines until you see something similar to the following:

0012F5D8	005BCE76	RETURN to dumped.005BCE76 from dumped.0062AEA7
0012F5DC	00FB6364	
0012F5E0	0012F5EC	
0012F5E4	00EC0220	
0012F5E8	00EC0224	ASCII "tHd"
0012F5EC	40000060	
0012F5F0	00F7EC50	
0012F5F4	00000001	
0012F5F8	00F4CB18	
0012F5FC	00F41A60	ASCII "YPBMPB"
0012F600	00000073	

Figure 6.

Follow in disassembler, the RETURN to address 005BCE76 above until you come to the condition below:

Loading & Patching ActiveMark v6.x - Part II

005BCE7A	84C0	TEST AL,AL
005BCE7C	EB 02	JMP SHORT dumped.005BCE80
005BCE7E	FF1474	CALL DWORD PTR SS:[ESP+ESI*2]
005BCE81	91	XCHG EAX,ECX

Figure 7.

This is the expiration condition. Set a HWBP on execute for address 005BCE7A above. You can remove the HWBP on “Implemented Categories” and also the 1st line of code in Level 3. Restart your application and run (F9) to the address HWBP above. Note that register EAX has a value of ‘1’.

Normally, we would change this code to XOR EAX,EAX or XOR AL,AL but AM has something different in mind. Go ahead and modify the code and run the target. You will get one of [4] possible errors. The error noted in Figure 2 above, an Access Violation message, An Application Error message (seen below), or Termination of Process with invalid exit code. Or possibly a combination.

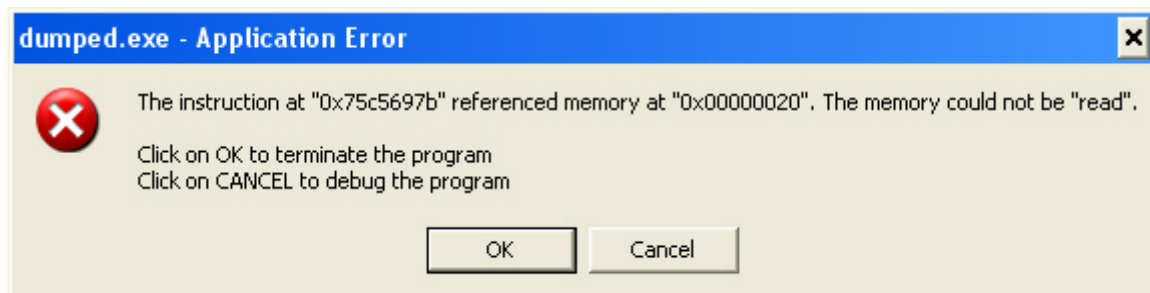


Figure 8.

In order to determine how this error message was generated, we can follow in the dump window, the line of code at address 005BCE7A and set a HWBP on access for the 1st byte. Restart the target. Some important concepts to ponder. If you dump at Level 2, you will break on this address a few times as the decompressor / decryptor processes the code. Irregardless of what Level you dump at (Excluding the possibility of dumping and fixing the original code), AM performs algorithmic checksum regions of code BEFORE and AFTER its execution. The pre execution phase for our byte of interest begins below in a loop. The algorithm processes each byte of code in a specific region or section and tallies the result.

0062B2D8	43	INC EBX
0062B2D9	C1C8 0D	ROR EAX,0D
0062B2DC	E2 F8	LOOPD SHORT dumped.0062B2D6
0062B2DE	01D3	ADD EBX,EDX
0062B2E0	EB 02	JMP SHORT dumped.0062B2E4

Figure 9.

Follow the JMP at address 0062B2E0 above until you come to the following condition:

Loading & Patching ActiveMark v6.x - Part II

0062B1FC	3D 44C99CEF	CMP EAX,EF9CC944
0062B201	↘ EB 02	JMP SHORT dumped.0062B205
0062B203	FF1474	CALL DWORD PTR SS:[ESP+ESI*2]

Figure 10.

Register EAX contains a hash total or checksum. If we modified any of the code in the section (i.e. patching a dump), the value in EAX would be different than the above leading to a “Bad” JMP. Okay, you say, change the code above to reflect our changes and whatever new value is in EAX. Again, any code that is modified is verified so it becomes a matter of the nth degree of finding and/or disabling this self check mechanism. Is it possible? Perhaps, but I like many of you have another life besides RE. Let’s move on to the AFTER execution, active memory scanning function for our byte of code. We hit the line at address 005FC93F below:

005FC93F	F7D1	NOT ECX
005FC941	D3CA	ROR EDX,CL
005FC943	↘ EB 02	JMP SHORT dumped.005FC947
005FC945	4F	DEC EDI
005FC946	↘ 74 31	JE SHORT dumped.005FC979
005FC948	D0EB	SHR BL,1
005FC94A	0381 CB33F7D1	ADD EAX,DWORD PTR DS:[ECX+D1F733CB]
005FC950	↘ EB 02	JMP SHORT dumped.005FC954

Figure 11.

Follow the logic thru, until you come to the highlighted address below:

005FC971	43	INC EBX
005FC972	C1C8 05	ROR EAX,5
005FC975	85C9	TEST ECX,ECX
005FC977	↘ 0F85 3B010000	JNZ dumped.005FCAB8
005FC97D	01D3	ADD EBX,EDX
005FC97F	↘ EB 01	JMP SHORT dumped.005FC982

Figure 12.

Set and Perform this HWBP on execution 2 times, once for the original code, and once for making a change to the code and note the value in register EAX for each. The final tally in register EAX becomes important as it is further processed with the result, which I do not show here [but you are free to explore], being that a data value created and referenced [later on] is either correct or invalid resulting in an Access Violation.

A mechanism needs to be in place to apply the changes to code [just-in-time], whether it’s in a dumped target or not, and then back them out. Or simply modify the Registers at time of execution. If you modify the contents in Register EAX to zero, when this condition is reached, the full evaluation time is restored. This patch need only be performed [2] times (generally) before the Browser screen would appear.

Loading & Patching ActiveMark v6.x - Part II

1.5.2 Browser

The familiar AM Browser screen is there to greet and remind us just how much time we have left. There is a new TEST AL,AL condition to explore for this function. To find, we need to determine how many times the Expiration condition HWBP is executed before the Browser Screen is displayed. Once determining that, for the 2nd or last time expiration hits, do a context Search for >> All commands the following “CALL DWORD PTR DS:[EAX+38]”. In Olly’s [Found Commands] window, set a BP on every command found. This is one of the few times we can utilize Software BP’s.

Run (F9) the target. The very first BP is shown below:

005D0CE6	FF50 38	CALL DWORD PTR DS:[EAX+38]	dumped.005EA7AA
005D0CE9	84C0	TEST AL,AL	
005D0CEB	EB 03	JMP SHORT dumped.005D0CF0	
005D0CED	32D2	XOR DL,DL	

Figure 13.

Sometimes you need to follow a jmp instruction, but the TEST AL,AL condition should soon follow our command as seen on address 005D0CE9 above. Step into this instruction and note the value in register EAX. Specifically, the value of AL, which is == ‘01’. You will also see the beginnings of a URL in register ECX. In the past we would XOR AL,AL this instruction and move on. Set a HWBP on execution for address 005D0CE6. Modify the contents in register EAX to zero.

Run (F9) the application. Note, the browser screen should not appear.

This condition normally occurs (2) times. Once, prior to executing the actual target, and once again on termination. It is this condition which represents your most difficult choice. If you use [just-in-time] memory patching, then your left with waiting until the application terminates to once again apply the patch or the final browser screen displays.

1.5.3 Timer / Timeout

No more looking for the dialog functions which push these (2) values and simply putting a RETN at the top. These functions are obfuscated at best and may be integrated into a new function at worst, which has limited capabilities for patching. You will also need to wait for AM to decrypt their respective literal strings before finding their use. The same can be said for the Browser function above. To get around this situation, we revert back to finding the SetTimer routine and where it is referenced in AM’s Level 3 code.

At this point, we have at least (2) HWBP’s on execution set. One for Expiration and one for the Browser. I talked about presenting the functions in the order in which they’re processed. To get to this condition, we hit the HWBP for the initial Browser screen. We then normally, hit the HWBP for the expiration condition. At this point, do a context Search for >> All commands the following “CALL DWORD PTR DS:[EAX+10]”. In Olly’s [Found Commands] window, set a BP on every command found. This is one of the few times we can utilize Software BP’s.

Run (F9) the target. The very first BP is shown below:

Loading & Patching ActiveMark v6.x - Part II

005F1BCF	FF 50 10	CALL DWORD PTR DS:[EAX+10]
005F1BD2	5F	POP EDI
005F1BD3	EB 02	JMP SHORT dumped.005F1BD7
005F1BD5	F7 05 5EEB0331 FFA05BEB	TEST DWORD PTR DS:[3103EB5E],EB5BA0FF

Figure 14.

The CALL is followed by a POP EDI instruction. The Stack Window should show something similar to the following:


0012F58C	0000003C	 This value
0012F590	0060EAD2	dumped.0060EAD2
0012F594	00F9A740	
0012F598	00000000	
0012F59C	00F9A788	
0012F5A0	0068CF50	dumped.0068CF50
0012F5A4	00EC03C8	

Figure 15.

Stepping into the call leads to the Classic SetTimer function shown below from Olly's Stack Window:

0012F0EC	006160FE	CALL to SetTimer from dumped.006160F9
0012F0F0	005B02E2	hWnd = 005B02E2 ('dumped',class='Static')
0012F0F4	027BB7A0	TimerID = 27BB7A0 (41662368.)
0012F0F8	0000EA60	Timeout = 60000. ms
0012F0FC	0062B4DB	Timerproc = dumped.0062B4DB
0012F100	0012F59C	
0012F104	0068CF50	dumped.0068CF50
0012F108	0068CF50	dumped.0068CF50

Figure 16.

Note the 60000 millisecond time interval above, 60 seconds converted.

Set a HWBP on execution for address 005F1BCF. To bypass this function, we need to NOP this CALL, [3] bytes. The end result would look like as follows:

005F1BCF	90	NOP
005F1BD0	90	NOP
005F1BD1	90	NOP
005F1BD2	5F	POP EDI
005F1BD3	EB 02	JMP SHORT dumped.005F1BD7

Figure 17.

The patch to this condition or instruction would be performed [just-in-time] to memory and then reversed immediately after execution.

Loading & Patching ActiveMark v6.x - Part II

1.5.4 WriteFile (data files and Registry key updates) for trial usage

For those that are interested in the mechanics of data files and registry keys that get updated each and every 60 seconds of trial usage, there is but one CALL instruction that performs everything for this purpose.

Olly'S [Executable modules] window, context choose View Names for module kernel32. Find the Name WriteFile and follow in disassembler this API. Set a HWBP on the beginning address. Simply run (F9) the target in Olly until this HWBP is hit. Scroll down the Stack Window in Olly and you should see something similar to the following:

0012F5F8	005D9FE6	RETURN to dumped.005D9FE6 from dumped.0060089A
0012F5FC	00FB6364	
0012F600	00000000	
0012F604	00EC0220	
0012F608	00EC0224	ASCII "tHd"
0012F60C	40000060	
0012F610	00F9A8D8	
0012F614	00000001	
0012F618	00F4C6D4	
0012F61C	00F44188	ASCII "YPBPBMI"
0012F620	00000073	
0012F624	0000007F	
0012F628	00643168	dumped.00643168

Figure 18.

Note the similarity to the Expiration condition's stack window in Figure 6. Context, follow in disassembler the RETURN at address 005D9FE6 above. Preceding the return address is the CALL below:

005D9FE1	E8 B4680200	CALL dumped.0060089A
005D9FE6	✓ EB 02	JMP SHORT dumped.005D9FEA

Figure 19.

It is this call, when NOP'ed that prevents any writing and updating of the registry keys and data files. You can make the changes below and try it yourself making sure to back out the change after execution each time.

005D9FE1	90	NOP
005D9FE2	90	NOP
005D9FE3	90	NOP
005D9FE4	90	NOP
005D9FE5	90	NOP

Figure 20.

This CALL is performed many times. You can further examine this CALL for AM processing.

Loading & Patching ActiveMark v6.x - Part II

1.5.5 The AMLOADV6 (Loader / custom loader creation Tool)

This tool was designed to Load, Run, Test and Generate [Create] custom loaders for applications using the new AM v6.x protection system. As outlined in this Tutorial, the tool focuses on the [3] key areas of functionality. The tool comes with a companion program, amloadv6.bin file, which is a loader template file, and must reside in the same folder. The readme.txt file pretty much explains all. To recap, AMLOADV6 allows for [3] input text boxes that are clearly labelled. You input the 8 digit addresses for each function in the appropriate text box. These are the addresses that contain the TEST AL,AL condition for Expiration and Browser, and the CALL address for Timer.

- Expiration Address Loader Action: Move zeroes to register EAX
- Browser Address Loader Action: Move zeroes to register EAX
- Timer Address Loader Action: NOP the 3 byte CALL instruction, execute the NOP's, then replace NOP's with original instruction.
-

All memory patching is performed [just-in-time] using Hardware Breakpoints. This means that **ALL** patches are temporary. You can choose to input none, or as many function addresses as you wish to **Test** on each run. Note: the addresses are not saved so you must input on each execution. AMLOADV6 terminates when the target application terminates with one exception. If running winxp or greater, you can choose the detach checkbox. AMLOADV6 will detach from the process **AFTER** processing the TIMER function and not until. No address for the timer function, **NO** detach. This is by design. Note: You **must** remain attached to the process if you want to suppress the final Browser screen. When you are satisfied with the Test results, you can save via the **Generate** button. This will allow you to save a custom loader with a unique name of your choice in the target folder. It is this new loader file which you will then execute instead of the original executable.

Note: The AMLOADV6 tool may only work with Winxp or greater, due to the anti-debugging measures employed. This is also true for any custom loaders created by the tool. Make sure you have Administrative privileges when using.

1.6. References

The target is a game called Buku kakuro - 31.3 Mb - 6.1.342 AM release. You can get it at the link below:

http://d.trymedia.com/dd/merscom/60m_d/merscom/BukuKakuroSetup.exe

Want to learn more about loaders? Use the following link:

[Cracking With Loaders Theory General Approach And A Framework V12](#)

1.7. Conclusions

There are many new features in ActiveMark v6.x to learn and control. This Tutorial is meant as a learning tool to demonstrate the [3] key areas of functionality [vulnerability] in AM v6.x applications and how to exploit them as relates to the front end approach. If, after reading this Tutorial, better methods in analyzing, dumping and patching, AM v6.x applications can be realized, then the author has accomplished his goal. This is still a fairly new release with many opportunities for learning and analyzing its protection scheme.

Loading & Patching ActiveMark v6.x - Part II

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.

1.8. 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.



<http://forums.accessroot.com/>