

RESULTS OF BAD SOFTWARE PROTECTION-IMPLEMENTATION

FOREWORDS

My name is Danny. Mostly of the time I am a malware analyst. I spend a lot of time to do research of every kind of malware, viruses and reversing in general.

This is my first tutorial which has as title “Results of bad protection-implementation”.

In this tutorial we will investigate a commercial application which is protected by several commercial protectors. The goal is not to harm this commercial application in any way, instead of that I will point the developers in the right direction if they read this tutorial. And I’m sure they will ;).

Also this is not only a learning-process for the developers of this application, but also for other Windows developers!

In this paper we will see that it doesn’t matter how many protections an application got, when it is implemented in the wrong way each protection is useless. This is the case of “**Driver Genius v12.0.0.12.11**” which is our target in this paper.

The developer spend probably a lot of money to protect his application. The first protector where we deal with is **Asprotect**. Well, I said “protector” but I think this one is merely used to embed a file into the main *.exe. However, the second protector is the feared **VMProtect**. Yes I said “feared” because VMProtect is a really strong protector, with really strong features. But like I told before it does not matter how strong a protection is, if it is implemented in the wrong way...

Happily there is just one good protection by the Developers of Driver Genius. Due this one and only good protection I am able to write this tutorial. I am talking about connecting to the license-server of Driver Genius when you want to download your out-to-dated drivers. This makes it impossible to download drivers from their server when you do not have a valid license. If this protection was not present I had never wrote this tutorial to keep Driver Genius alive ;)

References:

Asprotect

<http://www.aspack.com/asprotect.html>

VMProtect

<http://vmpsoft.com>

Driver Genius

<http://www.driver-soft.com>

1 TABLE OF CONTENTS

Forewords	1
Disclaimer/License	3
Verification	3
2 ANALYZING	4
3 UNPACKING AND FIXING DUMP	9
4 BIG MISTAKES	14
5 CONCLUSION	21
6 GREETINGS & CONTACT	22

DISCLAIMER/LICENSE

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

We have potentially illegal stuff inside. All the commercial programs used within our tutorials have been used only for the purpose of demonstrating the theories and methods described. These documents are released under the license of not using the information inside them to attack systems or programs for piracy. If you do it will be against our rules. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched by other fellows, and cracked versions were available since a lot of time. ARTeam or the authors of the papers shouldn't be considered responsible for damages to the companies holding rights on those programs. The scope of this document 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. We are not at all encouraging people to release cracked applications; damages if there will be any have to be claimed to persons badly using information, not under our license.

This disclaimer applies to all ARTeam releases and tutorials!

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>

2 ANALYZING

First of all we have to know how the application works. This means we are going to analyze the behavior of the application, the code of it etc. So load the application in Olly and run it. Play a bit with the features and return back to Olly and reload it. The first thing we see is that the application is compressed or protected as there is no “real” code.

The first 4 commands is typically Asprotect. Easy right?

00401000	68 01E08D00	PUSH DriverGe.008DE001
00401005	E8 01000000	CALL DriverGe.0040100B
0040100A	C3	RET
0040100B	C3	RET

Figure 1

Just unpack it in the way you unpack Asprotect usual. But what the hell! After getting rid of Asprotect we did not land at OEP! Instead, we did landed at a VMProtect stub. How do I know it is VMProtect? Well, if you are familiar with packers/protectors, you will quickly recognize the stubs of it.

So just unpack the VMProtect stub as usual. If you do not know how to do this, just use the ESP-trick, tracing and BP on CODE-section and pressing F9 a couple of times. The explanation of the ESP-trick can be found at:

<http://blog.chackraview.net/2012/03/24/manually-unpacking-dorkbot/>

1. From EP, you should press a few times F7 to land at 008DE001.
2. When you do the ESP-trick at this point, you will break somewhere; from here press **8** times F9 and you should land at VA 0068ABA6.
3. When you trace with F7 a while, you will see a RET 3C opcode, which is at VA 00688B6B. But a BP on it.
4. Now you should press F9 until you breaks for the **14'th** time at 00688B6B.
5. Set a memory breakpoint on access on .CODE section.
6. Press a couple of times F9 and....

You will land at OEP, which is **00412678** in this case. Place a HWBP on execution on it.

004125E0	-FF25 60124000	JMP DWORD PTR DS:[401260]	msvbvm60.__vbaVarInt
004125E6	-FF25 FC104000	JMP DWORD PTR DS:[4010FC]	msvbvm60.__vbaVarTstLt
004125EC	-FF25 80114000	JMP DWORD PTR DS:[401180]	msvbvm60.__vbaLbound
004125F2	-FF25 E0104000	JMP DWORD PTR DS:[4010E0]	msvbvm60.rtcFormatDateTime
004125F8	-FF25 34124000	JMP DWORD PTR DS:[401234]	msvbvm60.rtcPackDate
004125FE	-FF25 20124000	JMP DWORD PTR DS:[401220]	msvbvm60.__vbaDateVar
00412604	-FF25 EC114000	JMP DWORD PTR DS:[4011EC]	msvbvm60.__vbaI2Str
0041260A	-FF25 F8104000	JMP DWORD PTR DS:[4010F8]	msvbvm60.__vbaFpR8
00412610	-FF25 04134000	JMP DWORD PTR DS:[401304]	msvbvm60.rtcBstrFromFormatVar
00412616	-FF25 C8124000	JMP DWORD PTR DS:[4012C8]	msvbvm60.__vbaUVarCopy
0041261C	-FF25 60114000	JMP DWORD PTR DS:[401160]	msvbvm60.rtcIsNumeric
00412622	-FF25 B0124000	JMP DWORD PTR DS:[4012B0]	msvbvm60.__vbaUVarAdd
00412628	-FF25 18114000	JMP DWORD PTR DS:[401118]	msvbvm60.__vbaUVarZero
0041262E	-FF25 CC114000	JMP DWORD PTR DS:[4011CC]	msvbvm60.__vbaUI114
00412634	-FF25 C8114000	JMP DWORD PTR DS:[4011C8]	msvbvm60.__vbaStr2Vec
0041263A	-FF25 18134000	JMP DWORD PTR DS:[401318]	msvbvm60.rtcVarFromError
00412640	-FF25 60104000	JMP DWORD PTR DS:[401060]	msvbvm60.rtcGetObject
00412646	-FF25 6C124000	JMP DWORD PTR DS:[40126C]	msvbvm60.rtcHexBstrFromUVar
0041264C	-FF25 48124000	JMP DWORD PTR DS:[401248]	msvbvm60.rtcFileLength
00412652	-FF25 FC114000	JMP DWORD PTR DS:[4011FC]	msvbvm60.rtcCreateObject2
00412658	-FF25 BC104000	JMP DWORD PTR DS:[4010BC]	msvbvm60.__vbaFileCloseAll
0041265E	-FF25 C4114000	JMP DWORD PTR DS:[4011C4]	msvbvm60.EVENT_SINK_QueryInterface
00412664	-FF25 38114000	JMP DWORD PTR DS:[401138]	msvbvm60.EVENT_SINK_AddRef
0041266A	-FF25 A4114000	JMP DWORD PTR DS:[4011A4]	msvbvm60.EVENT_SINK_Release
00412670	-FF25 9C124000	JMP DWORD PTR DS:[40129C]	msvbvm60.ThunRTMain
00412676	0000	ADD BYTE PTR DS:[EAX],AL	
00412678	68 582C4100	PUSH DriverGe.00412C58	ASCII "UB5*6&*"
0041267D	E8 EFFFFFFF	CALL DriverGe.00412678	JMP to msvbvm60.ThunRTMain
00412682	0000	ADD BYTE PTR DS:[EAX],AL	
00412684	0000	ADD BYTE PTR DS:[EAX],AL	
00412686	0000	ADD BYTE PTR DS:[EAX],AL	
00412688	3000	XOR BYTE PTR DS:[EAX],AL	
0041268A	0000	ADD BYTE PTR DS:[EAX],AL	
00412C58=DriverGe.00412C58 (ASCII "UB5*6&*")			

Figure 2

When you take a look at the IAT right above OEP, you will notice that almost all imports are valid, except one.

00412202	-FF25 3C124000	JMP DWORD PTR DS:[40123C]	msvbvm60._CIlog
00412208	-FF25 08114000	JMP DWORD PTR DS:[401108]	msvbvm60._CIsin
0041220E	-FF25 B4114000	JMP DWORD PTR DS:[4011B4]	msvbvm60._CIsqrt
00412214	-FF25 20134000	JMP DWORD PTR DS:[401320]	msvbvm60._Citan
0041221A	-FF25 0C134000	JMP DWORD PTR DS:[40130C]	msvbvm60._allmul
00412220	-FF25 6C114000	JMP DWORD PTR DS:[40116C]	
00412226	-FF25 F4124000	JMP DWORD PTR DS:[4012F4]	msvbvm60.__vbaAryCopy
0041222C	-FF25 78114000	JMP DWORD PTR DS:[401178]	msvbvm60.__vbaUVarLateMemSt
00412232	-FF25 04114000	JMP DWORD PTR DS:[401104]	msvbvm60.rtcFilter
00412238	-FF25 24124000	JMP DWORD PTR DS:[401224]	msvbvm60.rtcGetTimer
0041223E	-FF25 04114000	JMP DWORD PTR DS:[401104]	msvbvm60.rtcSplit
00412244	-FF25 A8104000	JMP DWORD PTR DS:[4010A8]	msvbvm60.__vbaAryUVar
0041224A	-FF25 28114000	JMP DWORD PTR DS:[401128]	msvbvm60.__vbaNewtEachCollObj
00412250	-FF25 B8104000	JMP DWORD PTR DS:[4010B8]	msvbvm60.__vbaForEachCollObj
00412256	-FF25 58114000	JMP DWORD PTR DS:[401158]	msvbvm60.__vbaAryConstruct2
0041225C	-FF25 64124000	JMP DWORD PTR DS:[401264]	msvbvm60.rtcEndOfFile
00412262	-FF25 F0124000	JMP DWORD PTR DS:[4012F0]	msvbvm60.rtcRightCharBstr

Figure 3

So put a BP on the invalid import and run the application. Once you break, take the jump, it leads you to outside the *.exe file memory. Well, let's trace this little code. Trace till the 5'th line. At the 5'th line there is something interesting. It will move a pointer to EAX. This pointer points to an ASCII string which is "kernel32". We all know "kernel32.dll" is part of the library files of Microsoft.

The screenshot shows the DriverGenius debugger window titled "KernelMode - DriverGenius.exe - [*G.P.U* - main thread]". The interface includes a menu bar (File, View, Debug, Plugins, Options, Window, Help) and a toolbar with various navigation and execution icons. Below the toolbar is a list of assembly instructions with their addresses and hex values. Comments on the right side of the instructions provide context, such as "DriverGe.00427B44", "ASCII '180J'", "msvbvm60.dllFunctionCall", and "Unknown command". The instructions are as follows:

Address	Hex	Instruction	Comment
002D5174	55	PUSH EBP	
002D5175	8BEC	MOV EBP,ESP	
002D5177	53	PUSH EBX	
002D5178	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]	
002D517B	8B08	MOV EAX,DWORD PTR DS:[EBX]	DriverGe.00427B44
002D517D	8B00	MOV EAX,DWORD PTR DS:[EAX]	
002D517F	3B05 C8672E00	CMP EAX,DWORD PTR DS:[2E67C8]	
002D5185	75 1C	JNZ SHORT 002D51A3	
002D5187	8B43 04	MOV EAX,DWORD PTR DS:[EBX+4]	
002D518A	8A00	MOV AL,BYTE PTR DS:[EAX]	
002D518C	E8 9F84FFFF	CALL 002CD630	
002D5191	8BD8	MOV EBX,EAX	
002D5193	85DB	TEST EBX,EBX	
002D5195	75 15	JNZ SHORT 002D51AC	
002D5197	68 BC512D00	PUSH 2D51BC	ASCII "180J"
002D519C	E8 BF13FEFF	CALL 002B6560	
002D51A1	EB 09	JMP SHORT 002D51AC	
002D51A3	53	PUSH EBX	
002D51A4	FF15 206C2F00	CALL DWORD PTR DS:[2F6C20]	msvbvm60.dllFunctionCall
002D51AA	8BD8	MOV EBX,EAX	
002D51AC	8BC3	MOV EAX,EBX	
002D51AE	5B	POP EBX	
002D51AF	5D	POP EBP	
002D51B0	C2 0400	RETN 4	
002D51B3	00FF	ADD BH,BH	
002D51B5	FFFF	Unknown command	
002D51B7	FF05 00000031	INC DWORD PTR DS:[31000000]	
002D51BD	3B30	CMP BYTE PTR DS:[EAX],DH	
002D51BF	0D 0A000000	OR EAX,0A	

At the bottom, the register state is shown: DS:[00428864]=00427B44 (DriverGe.00427B44), ASCII "kernel32" and EAX=00412220 (DriverGe.00412220).

Figure 4

The next line is moving the first 4 bytes of the ASCII string to EAX. This is because the next line is comparing EAX to FFFFFFFF.

In this case (EAX != FFFFFFFF) so we will jump, but what will happen when we do **not** jump? Try by yourself and you will see the following result.

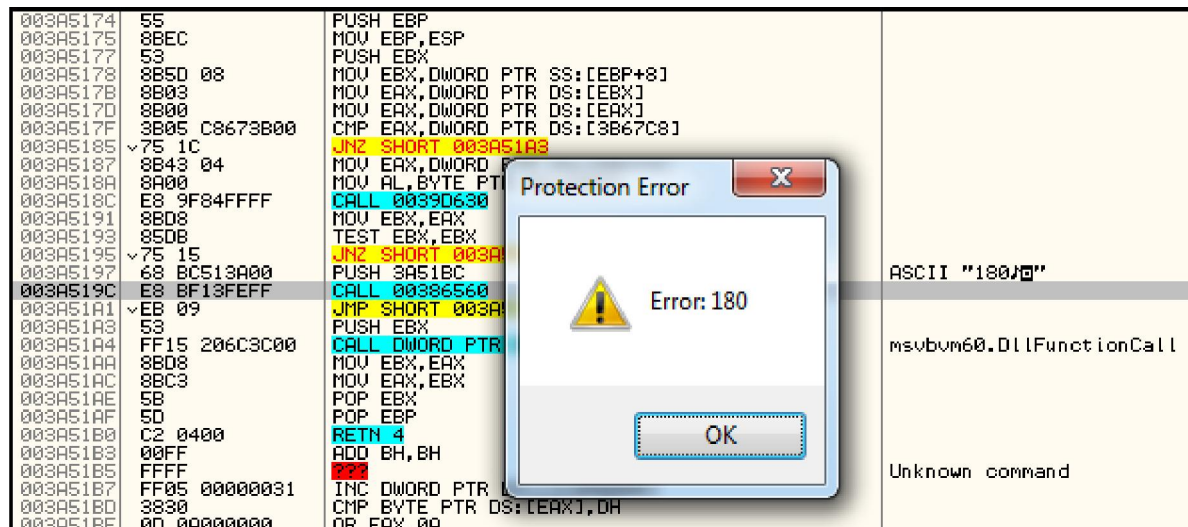


Figure 5

I guess this error is caused by Asprotect. When you restart the application in Olly, follow the dump of 0xFFFFFFFF when you reached the comparing from above, scroll up a little bit in the dump-window and you will see a confirm that we are dealing with Asprotect code.

Address	Hex dump	ASCII
003BF9BC	45 6E 63 72 79 70 74 69 6F 6E 20 63 6F 6E 73 74	Encryption const
003BF9CC	61 6E 74 73 20 66 6F 72 20 64 69 66 66 65 72 65	ants for differe
003BF9DC	6E 74 20 73 68 61 72 65 64 20 60 6F 64 65 73 20	nt shared modes
003BF9EC	61 72 65 20 6E 6F 74 20 65 71 75 61 6C 20 20 20	are not equal -
003BF9FC	65 6E 63 72 79 70 74 69 6F 6E 20 73 65 63 74 69	encryption secti
003BFA0C	6F 6E 20 73 29 20 6D 69 67 68 74 20 6E 6F 74 20	on(s) might not
003BFA1C	62 65 20 70 72 6F 70 65 72 6C 79 20 64 65 63 72	be properly decr
003BFA2C	79 70 74 65 64 2E 00 0A 50 6C 65 61 73 65 2C 20	ypted...Please,
003BFA3C	63 6F 6E 74 61 63 74 20 41 53 50 72 6F 74 65 63	contact AS Protec
003BFA4C	74 20 73 75 70 70 6F 72 74 21 00 00 5C FA 3B 00	t support!.. \ ; .

Figure 6

However, let's continue our research! Normally the jump is taken so let's take it.

When you land at the line of "PUSH EBX" follow EBX in dump, it should be clear what the next call does. Indeed, it is calling the function pushed by EBX.

Address	Value	ASCII	Comment
00428864	00427B44	DtB.	ASCII "kernel32"
00428868	00428854	TtB.	ASCII "CreateMutexA"

Figure 7

So in this case it will call the function "CreateMutexA" from "kernel32.dll". After this call, the address of the relevant API is in EAX, and will be executed.

So there are several ways to get a clean and working dump. For instance, we could just dump the file, fix the imports with a tool, cut the unresolved API and fix this manually. So I did this but the fixed dump crashed. After done a little research I saw there is many more code which refers to a **second** high memory address. But don't worry, we could dump this high memory address and add it to the dump. Then our dump will run. So let's do that!

3 UNPACKING AND FIXING DUMP

Restart the application in Olly and run it; we break at oep. Let's dump the file. After dumping, fix the imports and cut the unresolved thunk. Now we have to fix the invalid import manually, which means we have to add the missing import. I am using LordPE-> PE Editor to add the missing import.

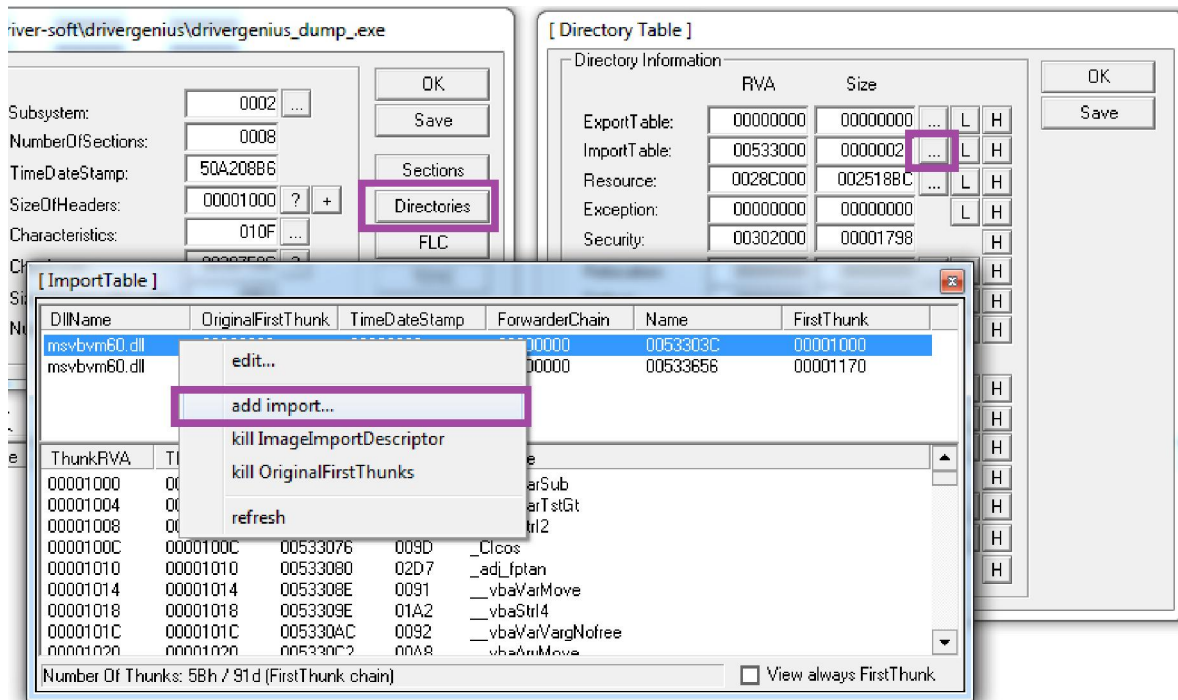


Figure 8

So we know we have to add "DllFunctionCall" exported by "msvbvm60.dll". After adding this import, write down the ThunkRVA of it, in my case it is "0053401F". This is the RVA of the new import. Now load the dumped file in Olly and write down the imagebase, which is "00400000" in my case. So the VA of the added import is **0093401F**. Let's fix this!

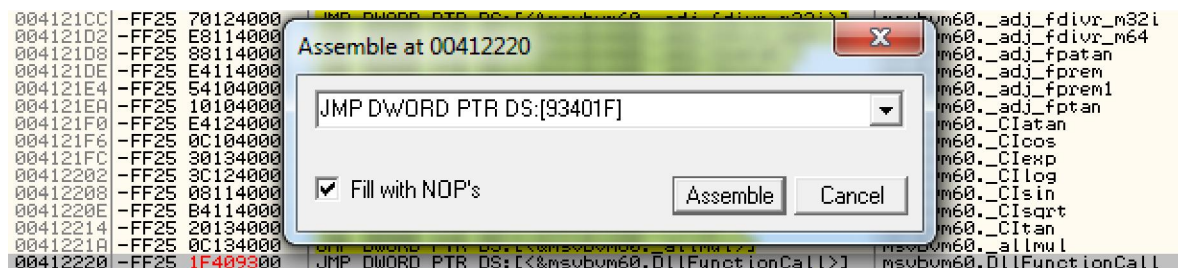


Figure 9

After fixing the imports the dump is still not running. After doing some research I found the place which causes this crash.

004CE514	80AC24 1F67C2FF	LEA EBP,DWORD PTR SS:[ESP+FFC2671F]
004CE518	F7DD	NEG EBP
004CE51D	8D6C24 34	LEA EBP,DWORD PTR SS:[ESP+34]
004CE521	66:85FF	TEST DI,DI
004CE524	0FA3C5	BT EBP,EAX
004CE527	81EC 8C000000	SUB ESP,8C
004CE52D	60	PUSHAD
004CE52E	8D6424 20	LEA ESP,DWORD PTR SS:[ESP+20]
004CE532	✓E9 0B350600	JMP driverge.00531A42
004CE537	5D	POP EBP
004CE538	8D6C24 30	LEA EBP,DWORD PTR SS:[ESP+30]
004CE53C	51	PUSH ECX
004CE53D	81EC 8C000000	SUB ESP,8C
004CE543	68 C518A856	PUSH 56A818C5
004CE548	8D6424 04	LEA ESP,DWORD PTR SS:[ESP+4]
004CE54C	✓E9 F1340600	JMP driverge.00531A42
004CE551	9C	PUSHFD
004CE552	✓E9 201D0600	JMP driverge.00530277
004CE557	9C	PUSHFD
004CE558	✓E9 B8EB1000	JMP driverge.005D0115
004CE55D	8B00	MOV EAX,DWORD PTR DS:[EAX]
004CE55F	68 43497F01	PUSH 17F4943
004CE564	8945 00	MOV DWORD PTR SS:[EBP],EAX
004CE567	C60424 2C	MOV BYTE PTR SS:[ESP],2C
004CE56B	8D6424 24	LEA ESP,DWORD PTR SS:[ESP+24]
004CE56F	✓E9 F1340600	JMP driverge.00531A65
004CE574	9C	PUSHFD
004CE575	897C24 08	MOV DWORD PTR SS:[ESP+8],EDI
004CE579	66:0FB6FB	MOVZX DI,BL
DS:[002DED44]=???		
EAX=002DED44		

Figure 10

As you can see it points to a high memory address which doesn't exist. This means when you dump the main exe, we have to dump this high memory section too, and add this section to the main exe!

So reload the original protected file, run till OEP and put a BP at **004CE55D**. In my case, the high memory address is 002FED44 and the value of it is **1CD5AF15**. Surprisingly enough when you restart the file, the high memory address is different than before and also two bytes of the value has changed. Repeating reloading the original file causes the same results, every time 2 bytes could be changed. So we can create a pattern to search for the high memory section to dump when we are at OEP: **15AF??1C**

So restart the original file and break at OEP. Go to the memory-map and search for binary string and type the pattern.

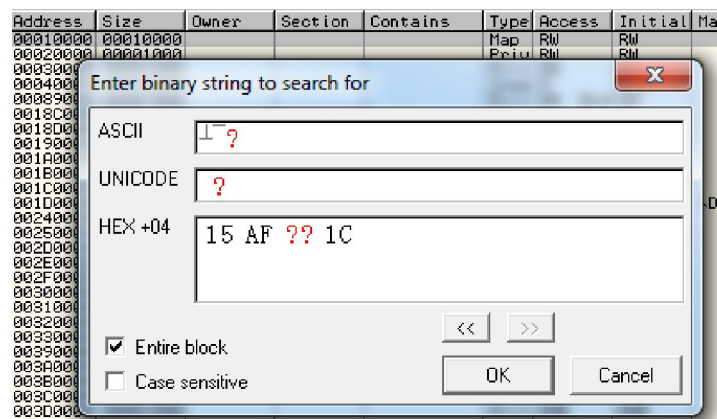


Figure 11

If everything is correct then it should find the pattern bytes and opens the dump-window. Make sure it is the right one we need, because in my case the first result found is **not** the correct one. Scroll to top and save the high memory section. In my case this section begins at VA 009C0000.

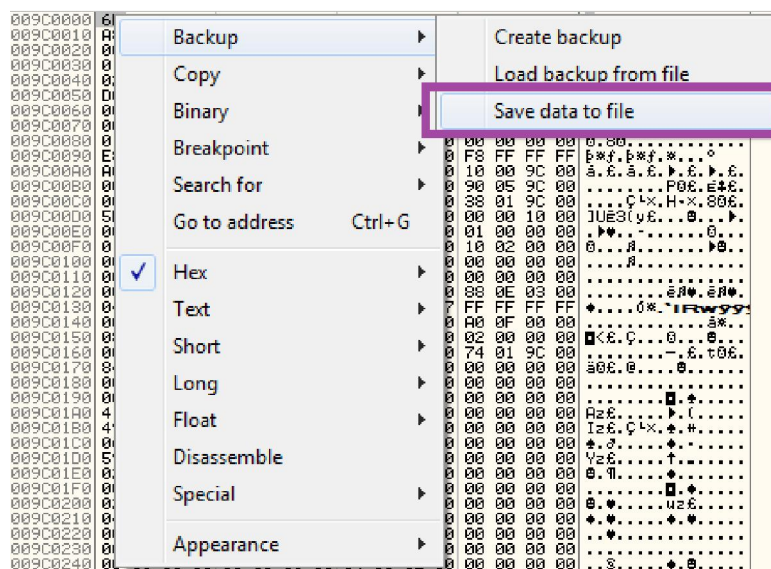


Figure 12

Now it's time to dump the main exe as usual. After adding the invalid import like we did before, it is time to add the dumped high memory section.

Open LordPE -> PE Editor and go to "Sections"

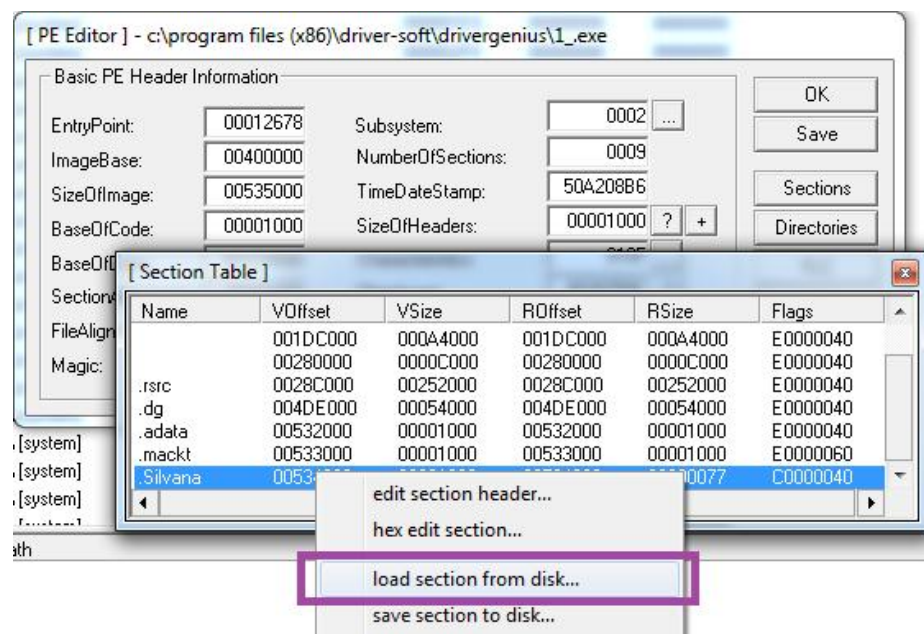


Figure 13

When you choose the dumped section, you need to change its VirtualAddress. In my case I have to change it to: $009C0000 - 00400000 = 5C0000$. So now rebuild the PE with LordPE and the dump is fixed!

But what the hell!? When you run the fixed dumped file it will crash again! We did not make any mistakes, did we?

No, we did not. Well, in the next chapter we will investigate in this problem.



Figure 14

4 BIG MISTAKES

I assume you can find the place of the crash by yourself. Right, the crash is caused in a CALL at VA **005D12A1**. Put a BP on it and run the application.

005D1249	· 7D 23	JGE SHORT driverge.005D126E	
005D124B	· 6A 10	PUSH 10	
005D124D	· 68 84994200	PUSH driverge.00429984	
005D1252	· 8B95 FCFFFFFF	MOV EDX, DWORD PTR SS:[EBP-104]	
005D1258	· 52	PUSH EDX	
005D1259	· 8B85 F8FFFFFF	MOV EAX, DWORD PTR SS:[EBP-108]	
005D125F	· 50	PUSH EAX	
005D1260	· FF15 94104000	CALL DWORD PTR DS:[<&msvbm60.__vbaHresu	msvbm60.__vbaHresultCheckObj
005D1266	· 8985 DCFDFFFF	MOV DWORD PTR SS:[EBP-224], EAX	
005D126C	· EB 0A	JMP SHORT driverge.005D1278	
005D126E	> C785 DCFDFFFF	MOV DWORD PTR SS:[EBP-224], 0	
005D1278	> 8D4D 8C	LEA ECX, DWORD PTR SS:[EBP-74]	msvbm60.__vbaFreeObj
005D127B	· FF15 34134000	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeO	
005D1281	> C745 FC 6C0000	MOV DWORD PTR SS:[EBP-4], 6C	
005D1288	· 8D8D D0FFFFFF	LEA ECX, DWORD PTR SS:[EBP-130]	
005D128E	· 51	PUSH ECX	
005D128F	· 68 3C884200	PUSH driverge.0042889C	
005D1294	· FF15 90104000	CALL DWORD PTR DS:[<&msvbm60.__vbaRecDe	msvbm60.__vbaRecDestruct
005D129A	· 8D95 D0FFFFFF	LEA EDX, DWORD PTR SS:[EBP-130]	
005D12A0	· 52	PUSH EDX	
005D12A1	· E8 CAD0FFFF	CALL driverge.005CE370	
005D12A6	· 66:83BD EAFEF	CMP WORD PTR SS:[EBP-116], 0FFFF	
005D12AE	· 0F85 F1020000	JNZ driverge.005D15A5	
005D12B4	· C745 FC 6D0000	MOV DWORD PTR SS:[EBP-4], 6D	
005D12B8	· BA 2CF84300	MOV EDX, driverge.0043F82C	UNICODE "IsFirstRun"
005D12C0	· 8D4D 9C	LEA ECX, DWORD PTR SS:[EBP-64]	
005D12C3	· FF15 74124000	CALL DWORD PTR DS:[<&msvbm60.__vbaStrCo	msvbm60.__vbaStrCopy
005D12C9	· 68 24864200	PUSH driverge.00428624	UNICODE "SOFTWARE\Driver-Soft\
005D12CE	· A1 08515D00	MOV EAX, DWORD PTR DS:[5D5108]	
005D12D3	· 50	PUSH EAX	
005D12D4	· FF15 70104000	CALL DWORD PTR DS:[<&msvbm60.__vbaStrCa	msvbm60.__vbaStrCat
005D12DA	· 8B00	MOV EDX, EAX	
005D12DC	· 8D4D A0	LEA ECX, DWORD PTR SS:[EBP-60]	
005D12DF	· FF15 E8124000	CALL DWORD PTR DS:[<&msvbm60.__vbaStrMo	msvbm60.__vbaStrMove
005D12E5	· C785 04FFFFFF	MOV DWORD PTR SS:[EBP-FC], 80000002	
005D12EF	· 8D4D 9C	LEA ECX, DWORD PTR SS:[EBP-64]	
005D12F2	· 51	PUSH ECX	
005D12F3	· 8D55 A0	LEA EDX, DWORD PTR SS:[EBP-60]	
005D12F6	· 52	PUSH EDX	
005D12F7	· 8D85 04FFFFFF	LEA EAX, DWORD PTR SS:[EBP-FC]	
005D12FD	· 50	PUSH EAX	
005D12FE	· E8 1D1AF6FF	CALL driverge.00532D20	
005D1303	· 8945 D0	MOV DWORD PTR SS:[EBP-30], EAX	
005D1306	· 8D4D 9C	LEA ECX, DWORD PTR SS:[EBP-64]	
005D1309	· 51	PUSH ECX	
005D130A	· 8D55 A0	LEA EDX, DWORD PTR SS:[EBP-60]	
005D130D	· 52	PUSH EDX	
005D130E	· 6A 02	PUSH 2	
005D1310	· FF15 80124000	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeS	msvbm60.__vbaFreeStrList
005D1316	· 83C4 0C	ADD ESP, 0C	
005D1319	· C745 FC 6E0000	MOV DWORD PTR SS:[EBP-4], 6E	
005D1320	· 8D85 7CFFFFFF	LEA EAX, DWORD PTR SS:[EBP-84]	

Figure 15

When you enter this call you will see this procedure is heavily obfuscated by VMProtect. Now we cannot read which operations are executed in this function! Really '**GENIUS**' right? ... Right...? Well, let's take a look closer.

Right after this call there is a comparison. So I guess in the call before the comparison, a value will be assigned to [EBP-16], and **after** the call there will be checked if [EBP-16] is equal to **0xFFFF**. And when this is true, it will not jump, which means it is the first run in this case.

So the huge mistake here is that the comparison is **outside** the obfuscated CALL, so we can easily decide whenever it is the first run or not. Really GENIUS right! So let me call this CALL the “magic CALL”

This CALL causes a crash so we can NOP this CALL or change the first byte of it to C3 (= RET). Let’s do the second option and you will see the dump runs fine now!

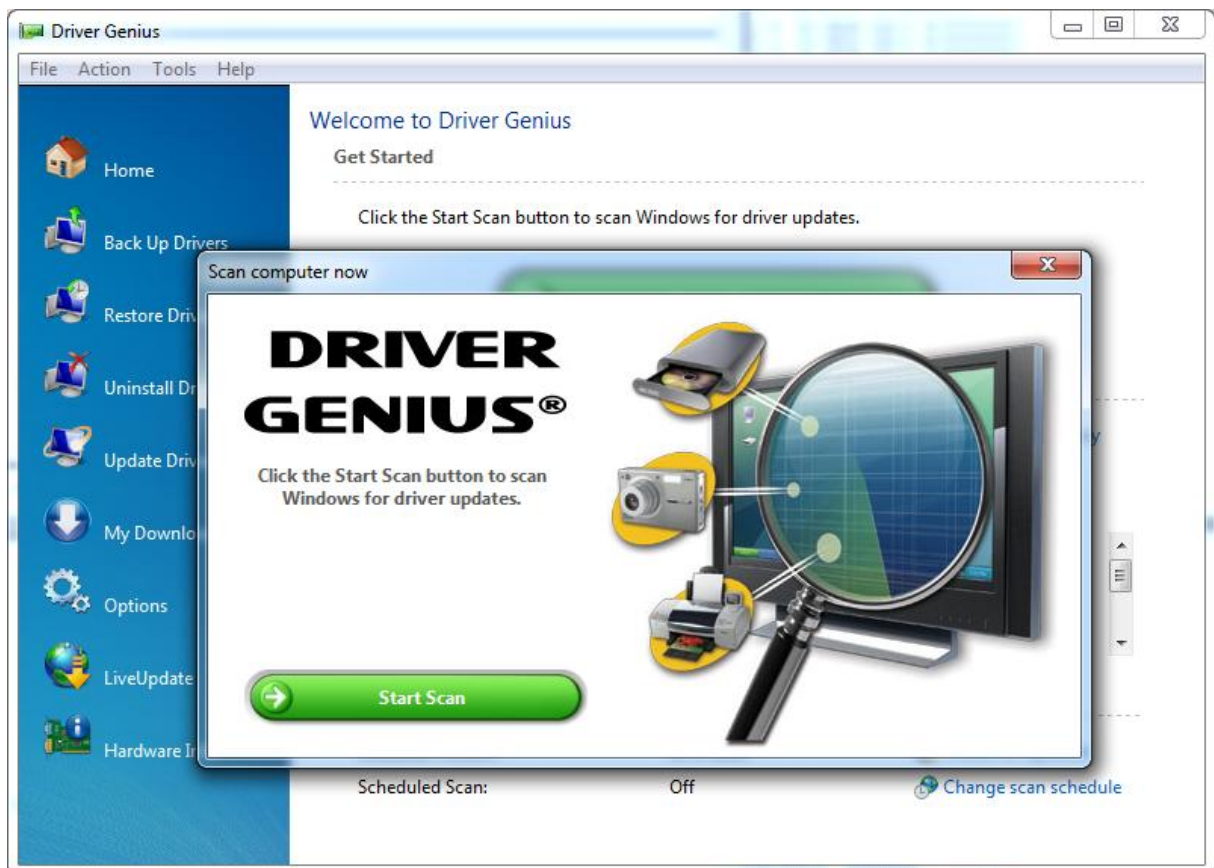


Figure 16

But maybe the magic CALL is used more times by other code. So put a BP on it and test some options of the application. In example, I want to see the about-window so I press “Help->About”. Bengg, we break at the magic CALL!

005CE36E	CC	INT3
005CE36F	CC	INT3
005CE370	C3	RETN
005CE371	8BEC	MOV EBP,ESP
005CE373	83EC 18	SUB ESP,18
005CE376	68 96214100	PUSH <JMP.&msvbvm60.__vbaExceptionHandler>
005CE37B	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
005CE381	50	PUSH EAX
005CE382	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
005CE389	B8 68000000	MOV EAX,68
005CE38E	E8 FD3DE4FF	CALL <JMP.&msvbvm60.__vbaChkstk>
005CE393	53	PUSH EBX
005CE394	56	PUSH ESI
005CE395	57	PUSH EDI

Figure 17

Now just return and you will face another comparison.

CMP WORD PTR SS:[EBP-0x6A], 0x0FFFF

So you will see this is not true so we will jump. But what if we do not jump? Change the Z-flag, run the application and see the result.

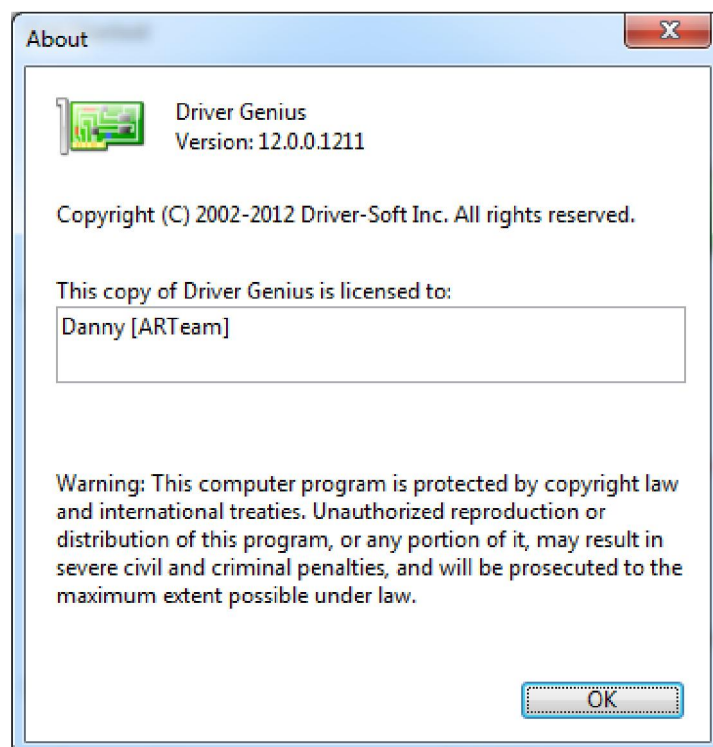


Figure 18

So there trial application becomes **Registered** after not jumping! If you test some features of the application you will see that the magic CALL determines if you are running a **trial** version or a **registered** one. So again a really really big mistake of the developers!



Figure 19

However, right now there is a new version released so we will take a look. Maybe they improved their security...?

After downloading and installing **Driver Genius v12.0.0.1314** open it in Olly.

In the first place everything looks pretty the same as the previous version. But when you want to unpack it like before, you will notice a little difference. Namely the **stolen OEP bytes**.

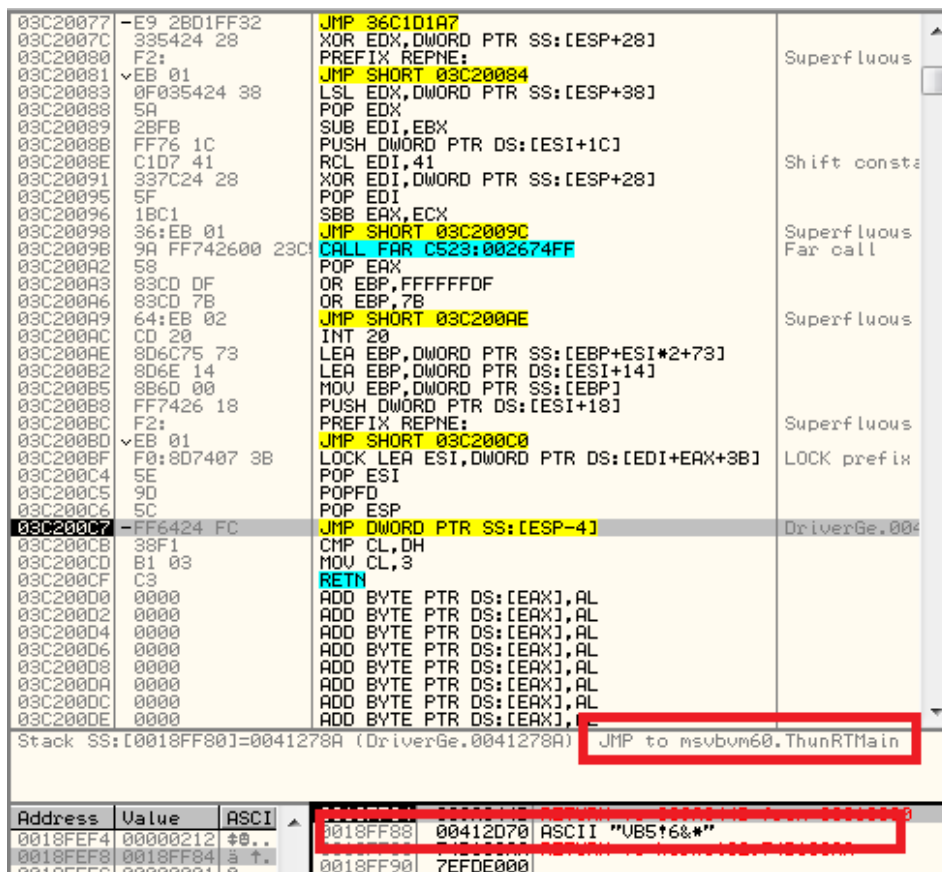


Figure 20

As you can see we are still in the high memory section, the first OEP bytes (**VB5!6&***) are already pushed onto the stack and now we will jump to ThunRTMain immediately! So when you want to unpack this new version you have to rebuild OEP.

So after unpacking and fixing dump, do a search for strings and you will see some plain internet urls, which is of course not a big mistake but it is unsecure for sure.

```

UNICODE "*.*)"
UNICODE "Version.tmp"
UNICODE "http://www.driver-soft.com/liveupdate/Version.ini"
UNICODE "Verify"
UNICODE "http://www.driver-soft.com/VerifyKey.asp?LicenseType="
UNICODE "DG12DS"
UNICODE "&LicenseCode="
UNICODE "&HardwareID="
UNICODE "Verifydata.tmp"
UNICODE ",L"

```

Figure 21

If you do a little research you will find the magic CALL like in the previous version. But the main difference in this version the magic CALL is **not obfuscated!!!** This is really a big fail because the magic CALL determines the license; trial or registered. The magic CALL is at VA **005D04F0**. You can read the purpose of this magic CALL clearly. We will leave that part because this is not our goal.

So, now we will take a look at downloading the drivers. Go back to the application GUI and push "Start Scan -> Fix Now".

Via the magic CALL you can manipulate the jumps so that you are able to download the drivers.

Put a BP on **InternetCrackUrlA**, and press "Download All"

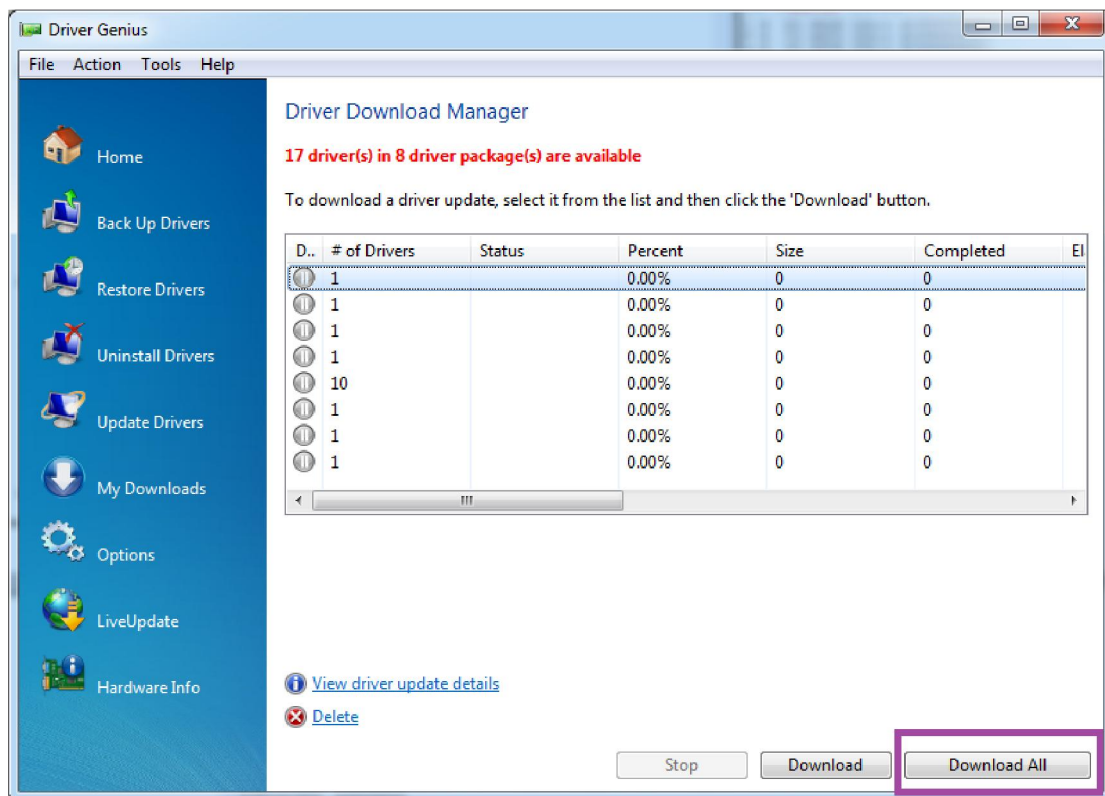


Figure 22

When you break check the register. So this is clear right? This URL will check if the server is online. If it cannot download/access "test.txt" then the server is probably down.

Registers (FPU)		
EAX	07B17DA8	ASCII "http://www.Driver-Soft.com/Download.asp?ServerID=1&FileName=test.txt"
ECX	00000003	
EDX	00000000	
EBX	07BB34D8	
ESP	0018C7B0	
EBP	0018C7C0	
ESI	00000044	
EDI	07B17DA8	ASCII "http://www.Driver-Soft.com/Download.asp?ServerID=1&FileName=test.txt"
EIP	079C8690	CLMUL"1.079C8690

Figure 23

In the same way but different API's you can catch the URL it is connecting to.

In example, when you put a BP on **InternetConnectA**, it will connect to...

079C8849	. 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
079C884C	. 50	PUSH EAX
079C884D	. FF15 3C5EA107	CALL DWORD PTR DS:[7A15E3C]
079C8853	. 5D	POP EBP
079C8854	. C2 2000	RETN 20
079C8857	. 90	NOP

DS:[07A15E3C]=74B6567E (wininet.InternetConnectA)

0FB2FED0	759B6FF0	KERNELBA.759B6FF0
0FB2FED4	A1314850	
0FB2FED8	00CC0004	
0FB2FEDC	07BBB508	ASCII "www.Driver-Soft.com"
0FB2FEE0	00000050	

Figure 24

When you continue you will see it breaks many more times on these API's, and it tries to download the drivers. But finally the developers programmed a secure protection **at server-side** (not client-side!), because when you does not have a valid license you simply cannot download the drivers, you cannot even access them via an internet browser.



5 CONCLUSION

We saw a lot of mistakes by the developer. He (probably) purchased pretty expensive protectors as Asprotect & VMProtect, but definitely used it in a wrong way.

I guess the developer used Asprotect to embed a file (the high memory address I guess), and used VMProtect to protect his application against reversing. Well, he failed in all ways.

VMProtect has a lot of strong features such as “Anti debugging” & “File check integrity” and “Code obfuscation”. The first two are not present and the last is very bad implemented.

Besides that, the IAT is not touched which means it is extremely easy to unpack the application. Also, in the newer version, the OEP bytes were stolen but this was not a big deal.

There was just one and the same CALL which determines the application is trial or registered. And the “real” decision was outside this CALL, which means it can be patched easily since it is not obfuscated!

The last but not least mistake is the plain internet URLs. This is not really secure don't you think?

The one and only good protection is from server-side. Without a valid license we are unable to download the drivers. Well done.

6 GREETINGS & CONTACT

Of course the thanks and greetings goes to **ARTeam**. In special to **Shub & Nacho**.

You can always contact me though the ARTeam forum:

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

Or via email:

danny_arteam@outlook.com



Danny / ARTeam

Reveal the unknown, hide the known