



HASP SL – A Deeper Dig

potassium of ARTeam

Version 1.1 – 29th May 2007

1. Introduction.....	2
2. Acquiring The Target Application	2
3. Manually Dodging The Anti-Debug Tricks	2
4. What Version Are We Dealing With?.....	10
5. New HASP tricks in Minitab 15	10
6. Conclusions.....	13
7. References.....	13
8. Greetings.....	13

Keywords

Manually unpacking, MUP, Privilege, HASP SL, Quality Companion 2.1.0.0, Minitab 15

IMPORTANT NOTICE:

This tutorial is provided ‘as is’ and is **ONLY** intended for educational purposes **NOT** to provide the reader with sufficient information to generate a crack, therefore some information has been intentionally left out of this tutorial. If you think that the software used as an example in this tutorial seems good enough to use you should also pay for it.



1. Introduction

In my previous tutorial¹ on HASP SL (6.41.95) I demonstrated how to easily find the OEP, dumping and rebuilding of the target application. The main goal of this tutorial is to do a little more in-depth analysis, how to manually bypass the anti-debug feature and finding out what version you are dealing with etc. Also this tutorial includes a flash-movie of the manual handy work described in this tutorial. Dodging the anti-debug, finding OEP, dumping and rebuilding. All done in about 2.5 minutes! So are you ready? Got your gear?

Beer? Check!
Peanuts? Check!
ImportRec? Check!
OllyDbg? CHECK!

Alrighty then.. Let's go!

2. Acquiring The Target Application

The example target can be downloaded from the minitab site:

http://fs1.minitab.com/Store/QC2_Setup.exe (It's 61 megs though)

3. Manually Dodging The Anti-Debug Tricks

Ok, to make things a little more of a 'sport' I will not use any anti-debug plug-ins but rather dodge them manually using some tricks. Right, if we where to load our executable into Olly and run it, we will get something like this:

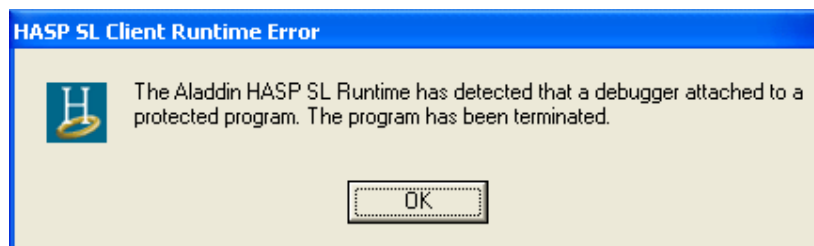
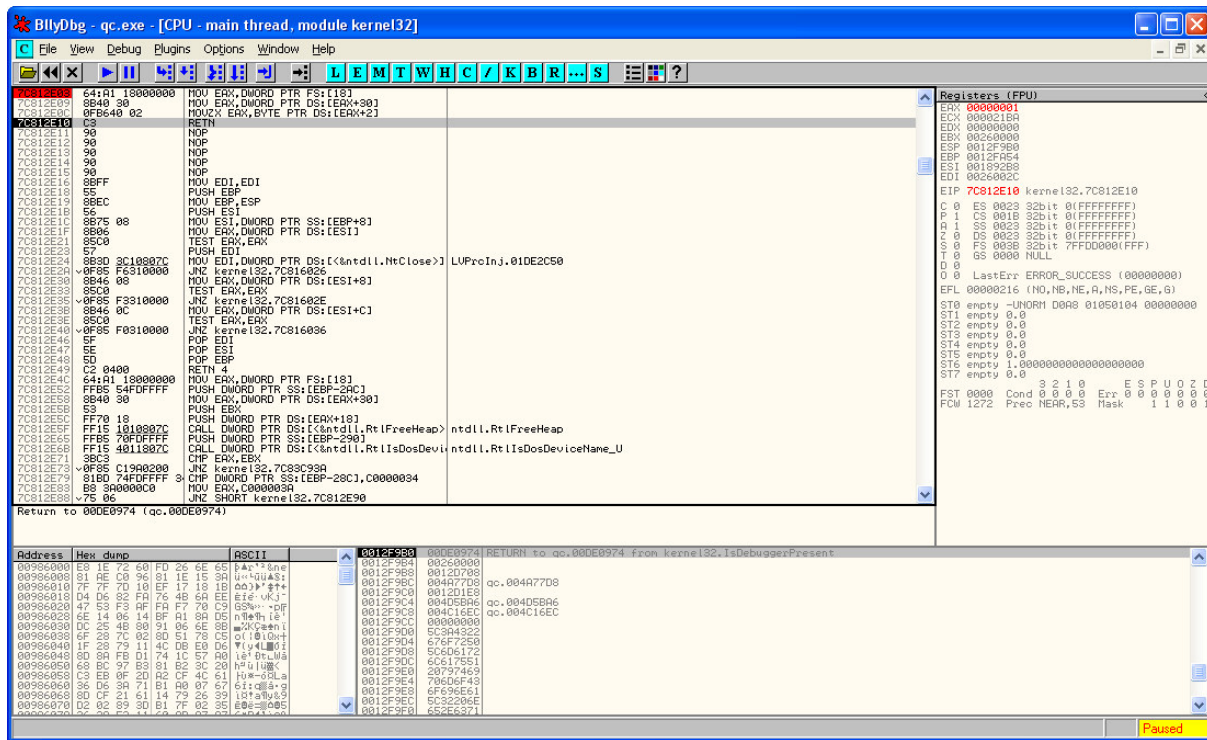


Figure 1.1 Ugly “Badboy, keep your debugger out, please”-screen (for you Lena☺)

Obviously our attempt to look into the code has stirred up some angry feelings of certain protector developers. Setting a breakpoint on IsDebuggerPresent will do the trick here, wont it? Let's try!



Ok, we break all right. And EAX is 1 (debugger present)



Now we have a couple of options, manually change the EAX to 0, which of course is easy, but in this case I know this routine is called.. well.. a lot. So doing that manually would be quite a waste of time. Lets follow that RETN and see where that leads us, shall we?

```
00DE0974 8BD8      MOV EBX, EAX
00DE0976 7D 05     JGE SHORT qc.00DE097D
00DE0978 7C 08     JL SHORT qc.00DE0982
00DE097A C786 90669090 66>MOV DWORD PTR DS:[ESI+90906690], 57D9066
00DE0984 7C 08     JL SHORT qc.00DE098E
00DE0986 C786 90669090 66>MOV DWORD PTR DS:[ESI+90906690], DB859066
00DE0990 7A 06     JPE SHORT qc.00DE0998
00DE0992 7B 03     JPO SHORT qc.00DE0997
00DE0994 C783 90900F85 DF>MOV DWORD PTR DS:[EBX+850F9090], 0DF
00DE0996 74 04     JE SHORT qc.00DE09A4
00DE0998 75 02     JNZ SHORT qc.00DE09A4
00DE09A2 C786 90576A24 87>MOV DWORD PTR DS:[ESI+246A5790], 7059C987
00DE09AC 06        PUSH ES
00DE09AD 71 04     JNO SHORT qc.00DE09B3
```

Ok, this is what we get, nasty, slightly obfuscated code, eeeuuw.. evil. You see that EAX is moved into EBX? Later, if you trace on a little bit, you'll see this:

```
00DE098B 90        NOP
00DE098C 66:90     NOP
00DE098E 85DB     TEST EBX,EBX <--- EBX is 1 now since we are debugging
00DE0990 7A 06     JPE SHORT qc.00DE0998 jump is not taken
00DE0992 7B 03     JPO SHORT qc.00DE0997
```

So, either we change MOV EBX,EAX to MOV EBX,0 or simply force the JPE to jump at all times. But! If we where to do this and let it run now, we would get this:

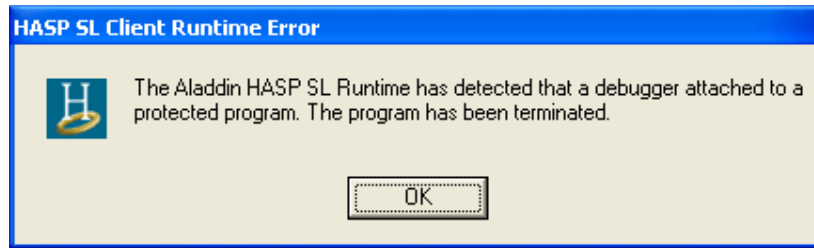


Figure 1.2 Ugly “Badboy, keep your debugger out, please”-screen (yes, again)

Oh no! Not again?! Oh yes! This is why:

```
00DE0A1E FF15 20BEDE00 CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcessId
00DE0A24 50          PUSH EAX <--- PID of our executable
00DE0A25 53          PUSH EBX
00DE0A26 68 00040000 PUSH 400
00DE0A2B FF15 1CBED000 CALL DWORD PTR DS:[<&KERNEL32.OpenProcess>]
00DE0A31 8BF8        MOV EDI,EAX
00DE0A33 85FF        TEST EDI,EDI
00DE0A35 74 45       JE SHORT qc.00DE0A7C
00DE0A37 68 68C4DE00 PUSH qc.00DEC468
00DE0A3C FF15 24BEDE00 CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>]
00DE0A42 85C0        TEST EAX,EAX
00DE0A44 74 2F       JE SHORT qc.00DE0A75
00DE0A46 68 4CC4DE00 PUSH qc.00DEC44C          ; ASCII "ZwQueryInformationProcess"
00DE0A4B 50          PUSH EAX
00DE0A4C FF15 14BEDE00 CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>]
00DE0A52 85C0        TEST EAX,EAX
00DE0A54 74 1F       JE SHORT qc.00DE0A75
00DE0A56 215D F8     AND DWORD PTR SS:[EBP-8],EBX
00DE0A59 215D FC     AND DWORD PTR SS:[EBP-4],EBX
00DE0A5C 8D4D F8     LEA ECX,DWORD PTR SS:[EBP-8]
00DE0A5F 51          PUSH ECX
00DE0A60 6A 04       PUSH 4
00DE0A62 8D4D FC     LEA ECX,DWORD PTR SS:[EBP-4]
00DE0A65 51          PUSH ECX
00DE0A66 6A 07       PUSH 7
00DE0A68 57          PUSH EDI
00DE0A69 FFD0        CALL EAX <--- Calls ZwQueryInformationProcess
00DE0A6B 85C0        TEST EAX,EAX
00DE0A6D 75 06       JNZ SHORT qc.00DE0A75
00DE0A6F 3945 FC     CMP DWORD PTR SS:[EBP-4],EAX
00DE0A72 74 01       JE SHORT qc.00DE0A75
00DE0A74 43          INC EBX
00DE0A75 57          PUSH EDI
00DE0A76 FF15 C4BDDE00 CALL DWORD PTR DS:[<&KERNEL32.CloseHandle>]
00DE0A7C 5F          POP EDI
00DE0A7D 8BC3        MOV EAX,EBX
00DE0A7F 7A 06       JPE SHORT qc.00DE0A87
00DE0A81 7B 03       JPO SHORT qc.00DE0A86
00DE0A83 C783 90907A04 7B>MOV DWORD PTR DS:[EBX+47A9090],1265027B
00DE0A8D 74 03       JE SHORT qc.00DE0A92
00DE0A8F 75 01       JNZ SHORT qc.00DE0A92
```

It opens the process of our dear executable (EAX@Openprocess = exe pid) and then fetches the address of the function “ZwQueryInformationProcess”. It’s address is then stored in EAX and called @ 00DE0A69. If Being Debugged, 0 is returned in EAX register. Ok, where were we? Changing the return values in EAX to 0 for IsDebuggerPresent and changing EAX to 1 for ZwQueryInformationProcess will do the trick then.



Well, to save you some time I'll tell you what happens if you simply alter the value of EAX register directly after returning from respective API. The same routine is run through and through several times, so this is simply something you do NOT want to do. Alternative two is to do a straight up, good old patch :) For instance, remember this piece of code from above:

```
00DE0974  8BD8          MOV EBX,EAX
00DE0976  7D 05         JGE SHORT qc.00DE097D
00DE0978  7C 08         JL  SHORT qc.00DE0982
```

At run-time you'll see that in this case EDX holds the value 0, so changing MOV EBX,EAX to MOV EBX,EDX will just do the trick for us. Then we head on for the ZwQueryInformationProcess thingy. There we had this :

```
00DE0A69  FFD0          CALL EAX
00DE0A6B  85C0          TEST EAX,EAX
00DE0A6D  75 06         JNZ SHORT qc.00DE0A75
```

Changing TEST EAX,EAX to INC EAX would do the trick. Done! Now it should run, or?



Figure 1.3 We don't want you to tamper with thing, okay?

Darn! They saw that one coming :P So now, what do we do!?! Start all over!
Reload the executable in Olly and I'll tell you.

```
7C812E03 > 64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C812E09  8B40 30         MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C  0FB640 02       MOVZX EAX,BYTE PTR DS:[EAX+2]
7C812E10  C3             RETN
7C812E11  90             NOP
7C812E12  90             NOP
7C812E13  90             NOP
7C812E14  90             NOP
7C812E15  90             NOP
```

Remember good ol' kernel32.dll from before? He's got plenty of space, don't you think?
-“Hey Kernel, nice NOP's you got there”. How about we give that kernel some extra temporary ‘upgrades’, huh?

```
7C812E03 > 64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C812E09  8B40 30         MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C  0FB640 02       MOVZX EAX,BYTE PTR DS:[EAX+2]
7C812E10  33C0           XOR EAX,EAX
7C812E12  C3             RETN
7C812E13  90             NOP
```

That's better!



And now it's time to give ntdll.dll (@ZwQueryInformationProcess) a shave:

```
7C90E01B > B8 9A000000    MOV EAX,9A
7C90E020    BA 0003FE7F    MOV EDX,7FFE0300
7C90E025    FF12          CALL DWORD PTR DS:[EDX]
7C90E027    C2 1400       RETN 14
7C90E02A    90           NOP
7C90E02B    90           NOP
7C90E02C    90           NOP
7C90E02D    90           NOP

7C90E01B > B8 9A000000    MOV EAX,9A
7C90E020    BA 0003FE7F    MOV EDX,7FFE0300
7C90E025    FF12          CALL DWORD PTR DS:[EDX]
7C90E027    40           INC EAX
7C90E028    C2 1400       RETN 14
7C90E02B    90           NOP
7C90E02C    90           NOP
```

Of course you would have to set the necessary breakpoints and step into the functions themselves and make the changes and then remove the bp's. Now what? Does it get the job done? Oh yeeeah!



Figure 1.4 A Classic, The trialscreen



Well then, if you were to do the approach presented in my tutorial “Manually Unpacking HASP SL v1.1”, which pretty much was; press alt+m, highlight the .text section of your target executable press F2 (set break-on-access), tick the “I would like to try...” button and then press the “next” button on the trialscreen (fig 1.3) and .. Bang! You’re at OEP.

Sorry to say that doesn’t work with this one. But It’s not very much harder though. Simply put bp (F2 break-on-access) on the .data section instead of the .text section. Olly will now break here:

```
00DDE080 8A1C3E      MOV BL, BYTE PTR DS:[ESI+EDI]
00DDE083 3A1F        CMP BL, BYTE PTR DS:[EDI]
00DDE085 7A 04       JPE SHORT qc.00DDE08B
00DDE087 7B 05       JPO SHORT qc.00DDE08E
00DDE089 C786 90669087 D2>MOV DWORD PTR DS:[ESI+87906690], 710770D2
00DDE093 02C7        ADD AL, BH
00DDE095 8690 66909066 XCHG BYTE PTR DS:[EAX+66909066], DL
00DDE09B 90          NOP
00DDE09C 72 03       JB SHORT qc.00DDE0A1
00DDE09E 73 01       JNB SHORT qc.00DDE0A1
00DDE0A0 C2 7D05     RETN 57D
00DDE0A3 7C 08       JL SHORT qc.00DDE0AD
```

Now, press alt+m again and set breakpoint (F2) on the .text section. When run you will land on the OEP!! → The standard procedure; dump, rebuild and repack or whatever you like.

According to the HASP SL SDK² (page 19 in included pdf SDK) there are functions in the protector for enabling different degrees of both encryption AND obfuscation. And, yes, of course there is a debugger detection feature as well. This is probably why the behavior is different in various applications, for instance Minitab 14.20 and Quality Companion 2.1.0.0. Compare these two sections of code:

From Minitab 14.20 (HASP SL 6.41)

```
00BAA4BC 8BD8      MOV EBX, EAX
00BAA4BE 85DB      TEST EBX, EBX
00BAA4C0 0F85 AC000000 JNZ Mtb14org.00BAA572
00BAA4C6 57        PUSH EDI
00BAA4C7 90        NOP
00BAA4C8 90        NOP
00BAA4C9 6A 24     PUSH 24
00BAA4CB 90        NOP
00BAA4CC 59        POP ECX
00BAA4CD 8BD8 68FFFFFF LEA EDI, DWORD PTR SS:[EBP-98]
00BAA4D3 F3:AB     REP STOS DWORD PTR ES:[EDI]
00BAA4D5 90        NOP
00BAA4D6 8D85 64FFFFFF LEA EAX, DWORD PTR SS:[EBP-9C]
00BAA4DC 50        PUSH EAX
00BAA4DD 90        NOP
00BAA4DE C785 64FFFFFF 94>MOV DWORD PTR SS:[EBP-9C], 94
00BAA4E8 FF15 4CF8BA00 CALL DWORD PTR DS:[<&KERNEL32.GetVersion>]
00BAA4EE 85C0      TEST EAX, EAX
00BAA4F0 74 7F     JE SHORT Mtb14org.00BAA571
00BAA4F2 83BD 74FFFFFF 02 CMP DWORD PTR SS:[EBP-8C], 2
00BAA4F9 75 76     JNZ SHORT Mtb14org.00BAA571
00BAA4FB FF15 48F8BA00 CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcessId>]
00BAA501 50        PUSH EAX
00BAA502 53        PUSH EBX
00BAA503 68 00040000 PUSH 400
00BAA508 90        NOP
00BAA509 90        NOP
00BAA50A 90        NOP
00BAA50B FF15 44F8BA00 CALL DWORD PTR DS:[<&KERNEL32.OpenProcess>]
00BAA511 8BF8      MOV EDI, EAX
00BAA513 90        NOP
00BAA514 90        NOP
00BAA515 90        NOP
```



```
00BAA516 90 NOP
00BAA517 85FF TEST EDI,EDI
00BAA519 74 56 JE SHORT Mtb14org.00BAA571
00BAA51B 68 88FEBA00 PUSH Mtb14org.00BAFE88 ; ASCII "ntdll.dll"
00BAA520 90 NOP
00BAA521 FF15 50F8BA00 CALL DWORD PTR DS:[<&KERNEL32.GetModuleH>
00BAA527 85C0 TEST EAX,EAX
00BAA529 74 3F JE SHORT Mtb14org.00BAA56A
00BAA52B 90 NOP
00BAA52C 90 NOP
00BAA52D 68 6CFEBA00 PUSH Mtb14org.00BAFE6C ; ASCII "ZwQueryInformationProcess"
00BAA532 90 NOP
00BAA533 90 NOP
00BAA534 50 PUSH EAX
00BAA535 90 NOP
00BAA536 90 NOP
00BAA537 90 NOP
00BAA538 FF15 3CF8BA00 CALL DWORD PTR DS:[<&KERNEL32.GetProcAdd>
00BAA53E 85C0 TEST EAX,EAX
00BAA540 74 28 JE SHORT Mtb14org.00BAA56A
00BAA542 215D F8 AND DWORD PTR SS:[EBP-8],EBX
00BAA545 90 NOP
00BAA546 215D FC AND DWORD PTR SS:[EBP-4],EBX
00BAA549 8D4D F8 LEA ECX,DWORD PTR SS:[EBP-8]
00BAA54C 90 NOP
00BAA54D 90 NOP
00BAA54E 90 NOP
00BAA54F 51 PUSH ECX
00BAA550 6A 04 PUSH 4
00BAA552 8D4D FC LEA ECX,DWORD PTR SS:[EBP-4]
00BAA555 51 PUSH ECX
00BAA556 6A 07 PUSH 7
00BAA558 90 NOP
00BAA559 90 NOP
00BAA55A 57 PUSH EDI
00BAA55B FFDD CALL EAX
00BAA55D 85C0 TEST EAX,EAX
00BAA55F 75 09 JNZ SHORT Mtb14org.00BAA56A
00BAA561 90 NOP
00BAA562 3945 FC CMP DWORD PTR SS:[EBP-4],EAX
00BAA565 74 03 JE SHORT Mtb14org.00BAA56A
00BAA567 90 NOP
00BAA568 90 NOP
00BAA569 43 INC EBX
00BAA56A 57 PUSH EDI
00BAA56B FF15 F4F7BA00 CALL DWORD PTR DS:[<&KERNEL32.CloseHandle>
00BAA571 5F POP EDI
00BAA572 8BC3 MOV EAX,EBX
00BAA574 5B POP EBX
00BAA575 90 NOP
00BAA576 C9 LEAVE
00BAA577 C3 RETN
```

..and from Quality Companion 2.1.0.0 (also HASP SL 6.41)

```
00DE0974 8BD8 MOV EBX,EAX
00DE0976 7D 05 JGE SHORT qc.00DE097D
00DE0978 7C 08 JL SHORT qc.00DE0982
00DE097A C786 90669090 66>MOV DWORD PTR DS:[ESI+90906690],57D9066
00DE0984 7C 08 JL SHORT qc.00DE098E
00DE0986 C786 90669090 66>MOV DWORD PTR DS:[ESI+90906690],DB859066
00DE0990 7A 06 JPE SHORT qc.00DE0998
00DE0992 7B 03 JPO SHORT qc.00DE0997
00DE0994 C783 90900F85 DF>MOV DWORD PTR DS:[EBX+850F9090],0DF
00DE099E 74 04 JE SHORT qc.00DE09A4
00DE09A0 75 02 JNZ SHORT qc.00DE09A4
00DE09A2 C786 90576A24 87>MOV DWORD PTR DS:[ESI+246A5790],7059C987
00DE09AC 06 PUSH ES
00DE09AD 71 04 JNO SHORT qc.00DE09B3
00DE09AF 81BE 66908DBD 68>CMP DWORD PTR DS:[ESI+BD8D9066],-98
00DE09B9 F3:AB REP STOS DWORD PTR ES:[EDI]
00DE09BB 70 04 JO SHORT qc.00DE09C1
00DE09BD 71 02 JNO SHORT qc.00DE09C1
```



```
00DE09BF 8DBD 7A047B02 LEA EDI,DWORD PTR SS:[EBP+27B047A]
00DE09C5 83C4 8D ADD ESP,-73
00DE09C8 8564FF FF TEST DWORD PTR DS:[EDI+EDI*8-1],ESP
00DE09CC FF7D ???
00DE09CE 05 7C05C786 ADD EAX,86C7057C
00DE09D3 0087 D2507D06 ADD BYTE PTR DS:[EDI+67D50D2],AL
00DE09D9 7C 03 JL SHORT qc.00DE09DE
00DE09DB C745 FC 669090C7 MOV DWORD PTR SS:[EBP-4],C7909066
00DE09E2 8564FF FF TEST DWORD PTR DS:[EDI+EDI*8-1],ESP
00DE09E6 FF9400 0000FF15 CALL DWORD PTR DS:[EAX+EAX+15FF0000]
00DE09ED ^E0 BD LOOPDNE SHORT qc.00DE09AC
00DE09EF DE00 FIADD WORD PTR DS:[EAX]
00DE09F1 72 03 JB SHORT qc.00DE09F6
00DE09F3 73 01 JNB SHORT qc.00DE09F6
00DE09F5 C2 85C0 RETN 0C085
00DE09F8 7A 06 JPE SHORT qc.00DE0A00
00DE09FA 7B 02 JPO SHORT qc.00DE09FE
00DE09FC 8DBD 9090747A LEA EDI,DWORD PTR SS:[EBP+7A749090]
00DE0A02 83BD 74FFFFFF 02 CMP DWORD PTR SS:[EBP-8C],2
00DE0A09 70 05 JO SHORT qc.00DE0A10
00DE0A0B 71 02 JNO SHORT qc.00DE0A0F
00DE0A0D 65:1290 756A7007 ADC DL,BYTE PTR GS:[EAX+7706A75]
00DE0A14 71 02 JNO SHORT qc.00DE0A18
00DE0A16 C786 90669090 66>MOV DWORD PTR DS:[ESI+90906690],15FF9066
00DE0A20 20BE DE005053 AND BYTE PTR DS:[ESI+535000DE],BH
00DE0A26 68 00040000 PUSH 400
00DE0A2B FF15 1CBEDE00 CALL DWORD PTR DS:[<&KERNEL32.OpenProcess>
00DE0A31 8BF8 MOV EDI,EAX
00DE0A33 85FF TEST EDI,EDI
00DE0A35 74 45 JE SHORT qc.00DE0A7C
00DE0A37 68 68C4DE00 PUSH qc.00DEC468 ; ASCII "ntdll.dll"
00DE0A3C FF15 24BEDE00 CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>
00DE0A42 85C0 TEST EAX,EAX
00DE0A44 74 2F JE SHORT qc.00DE0A75
00DE0A46 68 4CC4DE00 PUSH qc.00DEC44C ; ASCII "ZwQueryInformationProcess"
00DE0A4B 50 PUSH EAX
00DE0A4C FF15 14BEDE00 CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>; kernel32.GetProcAddress
00DE0A52 85C0 TEST EAX,EAX
00DE0A54 74 1F JE SHORT qc.00DE0A75
00DE0A56 215D F8 AND DWORD PTR SS:[EBP-8],EBX
00DE0A59 215D FC AND DWORD PTR SS:[EBP-4],EBX
00DE0A5C 8D4D F8 LEA ECX,DWORD PTR SS:[EBP-8]
00DE0A5F 51 PUSH ECX
00DE0A60 6A 04 PUSH 4
00DE0A62 8D4D FC LEA ECX,DWORD PTR SS:[EBP-4]
00DE0A65 51 PUSH ECX
00DE0A66 6A 07 PUSH 7
00DE0A68 57 PUSH EDI
00DE0A69 FFD0 CALL EAX
00DE0A6B 85C0 TEST EAX,EAX
00DE0A6D 75 06 JNZ SHORT qc.00DE0A75
00DE0A6F 3945 FC CMP DWORD PTR SS:[EBP-4],EAX
00DE0A72 74 01 JE SHORT qc.00DE0A75
00DE0A74 43 INC EBX
00DE0A75 57 PUSH EDI
00DE0A76 FF15 C4BDDE00 CALL DWORD PTR DS:[<&KERNEL32.CloseHandle>; kernel32.CloseHandle
00DE0A7C 5F POP EDI
00DE0A7D 8BC3 MOV EAX,EBX
```

Clearly, as you can see, the obfuscation feature has been turned on in the latter example, making it a little more difficult to foresee the flow of execution.



4. What Version Are We Dealing With?

This is what the “debugger detected”-nag looks like if it’s the older Privilege version you’re dealing with:

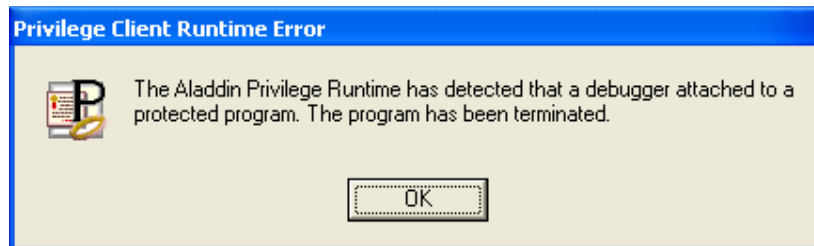


Figure 1.5 Privilege debugger detected

And this is what the HASP SL looks like:

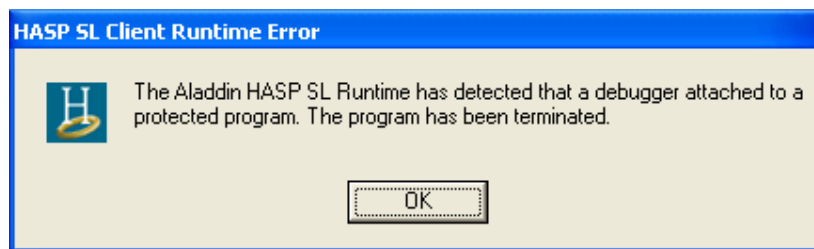


Figure 1.6 HASP SL debugger detected

What version of HASP you are dealing with can be found in `prv_ee.dll`, which can be viewed almost instantaneously after execution, with `alt+e`. In the targets I have studied so far have had the same version of `prv_ee.dll`; 6.41.95.0. The latest version of HASP SL happens to be 6.41 so this has to be a good and easy way to control what version you have on your HASP SL.

5. New HASP tricks in Minitab 15

If you were to unpack the latest version of Minitab 15 you’re in for a little surprise. The dump will not work as it is not possible to open any project files and the anti-debugging features are still present in when running our dumped exe. After some tracing it is realized the dll in question is `MtbMessaging.dll`. I have verified this in three ways;

1. Deletion of register posts for HASP licensing → dumped exe to prompt for missing license.
2. Deletion of `prv_ee.dll` → dumped exe to prompt that the protection framework is missing.
3. Patching of `MtbMessaging.dll` resolves the two above ;) → Dump runs fine without licence and protection dlls.



There is however one small problem with patching the MtbMessaging.dll, it too is protected with HASP SL. The data in the file is encrypted. During run-time memory is decrypted and copied to a memory section that is allocated via VirtualAlloc (in this case 00F90000). This memory section is also used for hooking LoadLibraryExW;

```
7C801AF1 >- E9 0AE57884 JMP 00F90000 <-- Hook redirection
7C801AF6 90 NOP
7C801AF7 90 NOP
7C801AF8 E8 C9090000 CALL kernel32.7C8024C6
7C801AFD 33FF XOR EDI,EDI
7C801AFF 897D D8 MOV DWORD PTR SS:[EBP-28],EDI
7C801B02 897D D4 MOV DWORD PTR SS:[EBP-2C],EDI
7C801B05 897D E0 MOV DWORD PTR SS:[EBP-20],EDI
7C801B08 897D E4 MOV DWORD PTR SS:[EBP-1C],EDI
7C801B0B 8B5D 10 MOV EBX,DWORD PTR SS:[EBP+10]
7C801B0E F6C3 01 TEST BL,1
7C801B11 0F85 E2000000 JNZ kernel32.7C801BF9
7C801B17 F6C3 10 TEST BL,10
7C801B1A 0F85 E5000000 JNZ kernel32.7C801C05
```

Above is a code snippet of LoadLibraryExW function directly after WriteProcessMemory, which is called from MtbMessaging.dll, has written a redirection jump to the allocated memory section at 00F90000 (in this case, probably some other address on your computer). The code at 00F90000 is copied from the decrypted section of MtbMessaging.dll.

```
5CB2A9BF F3:A5 REP MOVSD WORD PTR ES:[EDI],DWORD PTR DS:[ESI]
5CB2A9C1 74 04 JE SHORT MtbMessa.5CB2A9C7
5CB2A9C3 75 04 JNZ SHORT MtbMessa.5CB2A9C9
5CB2A9C5 C786 66908B35 5>MOV DWORD PTR DS:[ESI+358B9066],MtbMessa.5CB31358
5CB2A9CF 8DBB 5C170000 LEA EDI,DWORD PTR DS:[EBX+175C]

Registers: ESI 5CB2E007 MtbMessa.5CB2E007
           EDI 00F90384
```

So, the main goal is to inline patch the encrypted data in MtbMessaging.dll so that the checking routine for protection integrity is bypassed. Here is the call back into MtbMessaging.dll that does that.

```
00F90365 8B46 08 MOV EAX,DWORD PTR DS:[ESI+8]
00F90368 3B45 F0 CMP EAX,DWORD PTR SS:[EBP-10]
00F9036B 74 04 JE SHORT 00F90371
00F9036D 85C0 TEST EAX,EAX
00F9036F 75 30 JNZ SHORT 00F903A1
00F90371 F646 F4 01 TEST BYTE PTR DS:[ESI-C],1
00F90375 74 20 JE SHORT 00F90397
00F90377 8B06 MOV EAX,DWORD PTR DS:[ESI]
00F90379 85C0 TEST EAX,EAX
00F9037B C746 F4 0200000>MOV DWORD PTR DS:[ESI-C],2
00F90382 74 13 JE SHORT 00F90397
00F90384 FF76 FC PUSH DWORD PTR DS:[ESI-4]
00F90387 FFD0 CALL EAX <-- Integrity & Debugger checks
00F90389 85C0 TEST EAX,EAX
00F9038B 75 0A JNZ SHORT 00F90397
00F9038D 2145 0C AND DWORD PTR SS:[EBP+C],EAX
00F90390 C746 F4 0100000>MOV DWORD PTR DS:[ESI-C],1
```

As it is plain to see, exchanging \$74h with \$EBh will always bypass the integrity/debugger check routine. But since MtbMessaging.dll is protected with HASP SL and, thus, encrypted we need to know what and where to make the necessary changes. By searching the .protect section of MtbMessaging.dll for the conditional jump (74 13) and a couple of more bytes you will find a slice of code that is identical to the one above, but located within the MtbMessaging.dll.



```
00F90365 8B46 08 MOV EAX,DWORD PTR DS:[ESI+8]
00F90368 3B45 F0 CMP EAX,DWORD PTR SS:[EBP-10]
00F9036B 74 04 JE SHORT 00F90371
00F9036D 85C0 TEST EAX,EAX
00F9036F 75 30 JNZ SHORT 00F903A1
00F90371 F646 F4 01 TEST BYTE PTR DS:[ESI-C],1
00F90375 74 20 JE SHORT 00F90397
00F90377 8B06 MOV EAX,DWORD PTR DS:[ESI]
00F90379 85C0 TEST EAX,EAX
00F9037B C746 F4 0200000>MOV DWORD PTR DS:[ESI-C],2
00F90382 74 13 JE SHORT 00F90397
00F90384 FF76 FC PUSH DWORD PTR DS:[ESI-4]
00F90387 FFD0 CALL EAX <-- Integrity & Debuggerchecks
00F90389 85C0 TEST EAX,EAX
00F9038B 75 0A JNZ SHORT 00F90397
00F9038D 2145 0C AND DWORD PTR SS:[EBP+C],EAX
00F90390 C746 F4 0100000>MOV DWORD PTR DS:[ESI-C],1
```

Same data in .protect section of MtbMessaging.dll

```
5CB2DFE8 8B46 08 MOV EAX,DWORD PTR DS:[ESI+8]
5CB2DFEB 3B45 F0 CMP EAX,DWORD PTR SS:[EBP-10]
5CB2DFEE 74 04 JE SHORT MtbMessa.5CB2DFF4
5CB2DFF0 85C0 TEST EAX,EAX
5CB2DFF2 75 30 JNZ SHORT MtbMessa.5CB2E024
5CB2DFF4 F646 F4 01 TEST BYTE PTR DS:[ESI-C],1
5CB2DFF8 74 20 JE SHORT MtbMessa.5CB2E01A
5CB2DFFA 8B06 MOV EAX,DWORD PTR DS:[ESI]
5CB2DFFC 85C0 TEST EAX,EAX
5CB2DFFE C746 F4 0200000>MOV DWORD PTR DS:[ESI-C],2
5CB2E005 74 13 JE SHORT MtbMessa.5CB2E01A
5CB2E007 FF76 FC PUSH DWORD PTR DS:[ESI-4]
5CB2E00A FFD0 CALL EAX <-- Integrity & Debuggerchecks
5CB2E00C 85C0 TEST EAX,EAX
5CB2E00E 75 0A JNZ SHORT MtbMessa.5CB2E01A
5CB2E010 2145 0C AND DWORD PTR SS:[EBP+C],EAX
5CB2E013 C746 F4 0100000>MOV DWORD PTR DS:[ESI-C],1
```

To accomplish the patching task we could simply replace the bytes of interest by trial-and-error but it would be much more elegant to know exactly which byte to replace with what. Now, let's check out the main decryption loop.

```
5CB36192 8B55 F8 MOV EDX,DWORD PTR SS:[EBP-8]
5CB36195 83C2 01 ADD EDX,1
5CB36198 8955 F8 MOV DWORD PTR SS:[EBP-8],EDX
5CB3619B 8B45 F8 MOV EAX,DWORD PTR SS:[EBP-8]
5CB3619E 3B05 8463B35C CMP EAX,DWORD PTR DS:[5CB36384]
5CB361A4 73 5F JNB SHORT MtbMessa.5CB36205
5CB361A6 8B4D F8 MOV ECX,DWORD PTR SS:[EBP-8]
5CB361A9 69C9 396D7159 IMUL ECX,ECX,59716D39
5CB361AF 8B15 8863B35C MOV EDX,DWORD PTR DS:[5CB36388]
5CB361B5 03D1 ADD EDX,ECX
5CB361B7 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
5CB361BA 8B08 MOV ECX,DWORD PTR DS:[EAX]
5CB361BC 33CA XOR ECX,EDX <-- XOR!
5CB361BE 8B55 FC MOV EDX,DWORD PTR SS:[EBP-4]
5CB361C1 890A MOV DWORD PTR DS:[EDX],ECX
5CB361C3 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
5CB361C6 8B10 MOV EDX,DWORD PTR DS:[EAX]
5CB361C8 0315 8863B35C ADD EDX,DWORD PTR DS:[5CB36388]
5CB361CE 8B4D F8 MOV ECX,DWORD PTR SS:[EBP-8]
5CB361D1 83E1 1F AND ECX,1F
5CB361D4 D3E2 SHL EDX,CL
5CB361D6 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
5CB361D9 8B00 MOV EAX,DWORD PTR DS:[EAX]
5CB361DB 0305 8863B35C ADD EAX,DWORD PTR DS:[5CB36388]
5CB361E1 8B4D F8 MOV ECX,DWORD PTR SS:[EBP-8]
5CB361E4 83E1 1F AND ECX,1F
5CB361E7 BE 1F000000 MOV ESI,1F
5CB361EC 2BF1 SUB ESI,ECX
```



```
5CB361EE      8BCE      MOV ECX,ESI
5CB361F0      D3E8      SHR EAX,CL
5CB361F2      03D0      ADD EDX,EAX
5CB361F4      8915 8863B35C MOV DWORD PTR DS:[5CB36388],EDX
5CB361FA      8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
5CB361FD      83C2 04      ADD EDX,4
5CB36200      8955 FC      MOV DWORD PTR SS:[EBP-4],EDX
5CB36203      ^ EB 8D      JMP SHORT MtbMessa.5CB36192
```

Setting an on-access breakpoint on right before the \$74h \$13h is written

```
5CB2DFFE      C746 F4 02000000>MOV DWORD PTR DS:[ESI-C],2    <-- hr here
5CB2E005      74 13      JE SHORT MtbMessa.5CB2E01A
```

Now, if you loop through the decryption loop two more times your byte of interest will be next up. Stop at 5CB361BC (XOR ECX,EDX) and examine the registers ECX and EDX, which now holds the following values: ECX: 7ED8D7E5 and EDX: 81CBA3E5. So, what you figure happens if you XOR \$D7h with \$A3h and XOR \$D8h with \$CBh ? Try with calculator. Yes, \$74h and \$13h!! But we want \$74h to be \$EBh so we need to XOR \$EBh with \$A3h which is \$48h. Now simply exchange the \$D8h with \$48h and your dumped exe will run just fine. Even without licenses. :)

6. Conclusions

Even though HASP SL is equipped with several strong features, such as anti-tampering, anti-debugging, product activation, encryption of the main executable and code obfuscation, it is still, in my personal opinion, too weak in certain aspects. Don't get me wrong, some portions of it is very hard to analyze due to the extensive anti-tampering and anti-debugging routines but I feel that there is still much to be done on this protector to call it 'top notch'.

So what's next? Some import protection? More extreme code obfuscation? Tougher anti-debugging features? Only the future will tell.

7. References

- [1] Manually Unpacking HASP SL v1.1 by potassium of ARTeam
- [2] <http://library.aladdin.com/> HASP SL SDK, last visited 2006-10-06

8. Greetings

Greetings goes out to the entire ARTeam family with it's past, present and future members. I would also like to send a big hug to all forum members that does a great job in learning and helping each other. True ARTeam spirit :) If you've got suggestions and/or comments about this tutorial or just want to say hi, stop by at the forum and drop me a pm.

potassium of ARTeam

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