



Manually Unpacking HASP SL

potassium of ARTeam

Version 1.1 - 14th February 2006

1. Abstract	2
2. Acquiring a target application.....	2
3. Get down to the dirty business	2
4. Conclusions.....	8
5. Greetings	8

Keywords

Manually unpacking, MUP, HASP SL, Minitab 14.20



1. Abstract

According to Aladdin, the provider of HASP products, HASP SL is a secure way to distribute your software, empowering your software with features like 'try before you buy' and 'trial only'. For a complete list of features, check out: <http://www.aladdin.com/HASPSL/features.asp>. In this tutorial it will be demonstrated how all those activation and debugging features simply aren't more than paper walls. ^__^

2. Acquiring a target application

At www.minitab.com you can get a statistical software that is protected with the protection in question. For tools you'll need the usual stuff; Ollydbg with the usual hide debugger plug-ins (Olly Invisible by Teerayoot + Hide Debugger by Asterix) and ImportRec.

3. Get down to the dirty business

When running the target application and you will see this screen:



Figure 1.1 The annoying activation-trial screen



Ok, from here you attack this target with different approaches; try to activate it or unpack it and get rid of the protection for good.

So, as the title says, we will unpack it ;) Launch OllyDbg and start here:

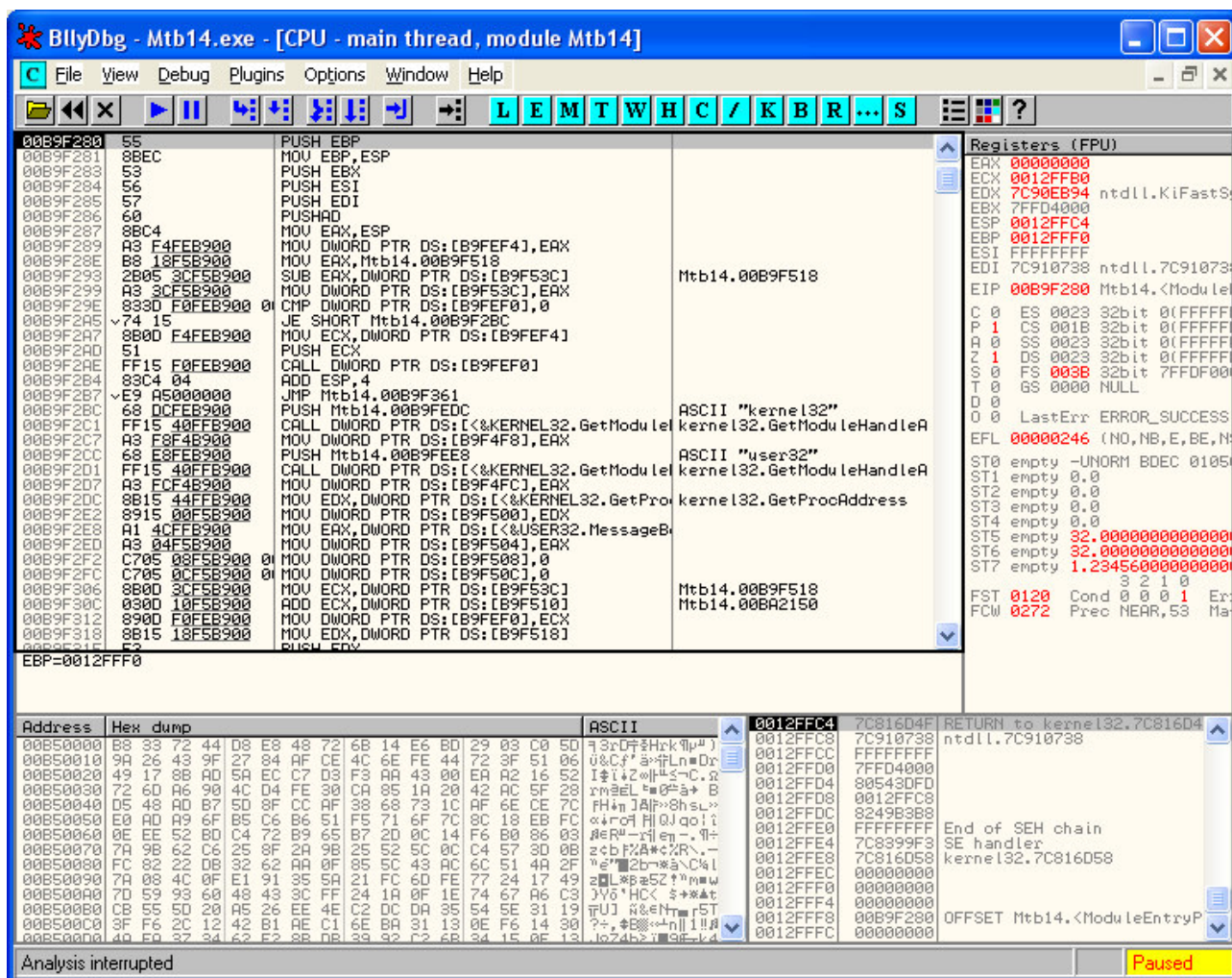


Figure 1.2 Startingpoint

Now, press F9 and wait for the application to load. Please be patient, sometimes the debugger is detected even though you have all the available hidereaders active. So, if you are detected, just keep trying and you'll get through. When the application runs successfully you will be presented with the screen you saw in Figure 1.1. Tick the radio button 'I want to try XXX and so on' but don't press the next button yet!



Go back to Olly and bring up the Memory section (alt+m) :

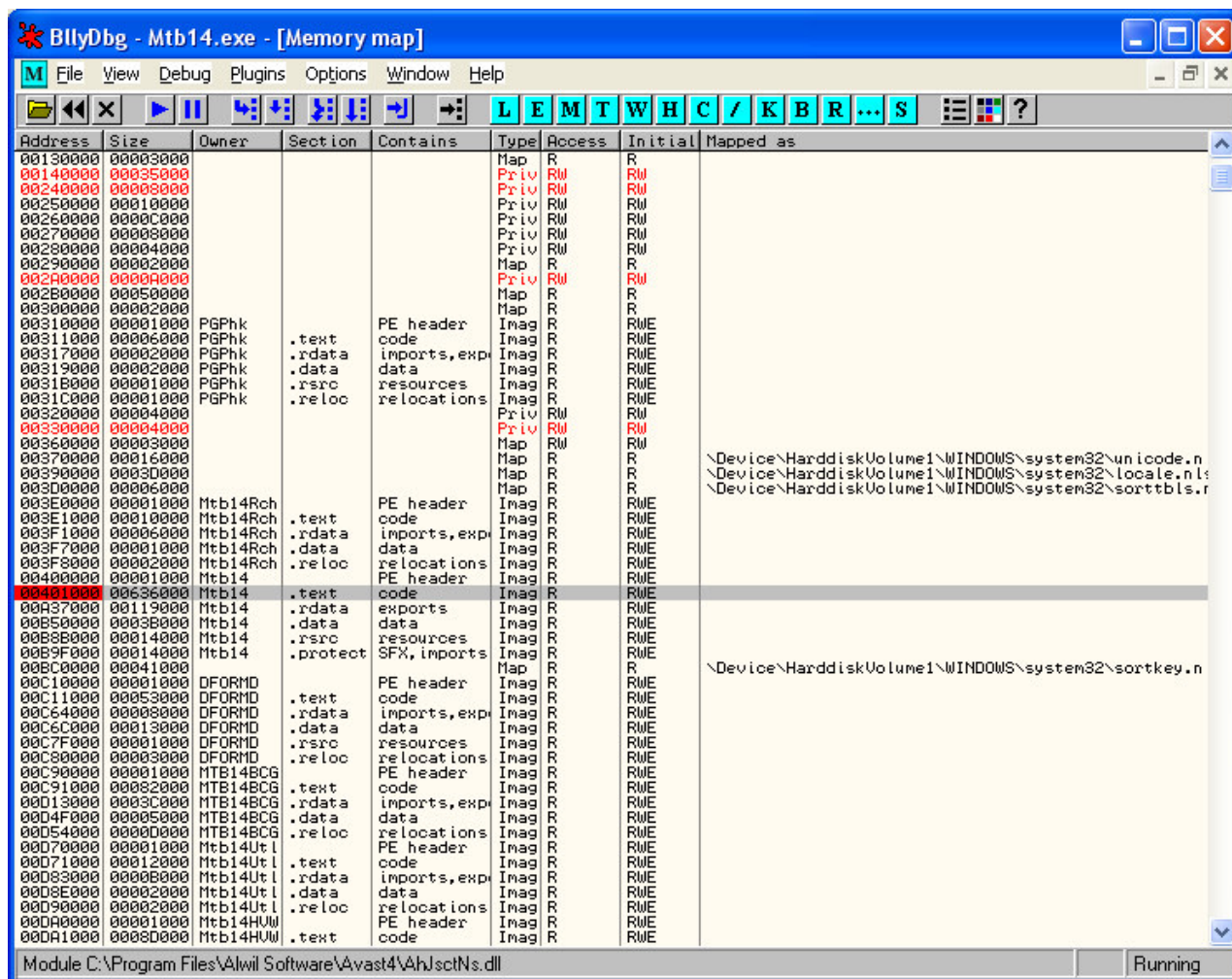


Figure 1.3 Setting memory breakpoint

Here you mark the .text section of Mtb14 and press F2 to set bp on that memory section. The OEP resides in this section, as you will see later. Now get back to the application and press the 'next' button. BAM! You brake on the OEP.. Man it's magic.



Dump as usual (ollydump + no rebuild)

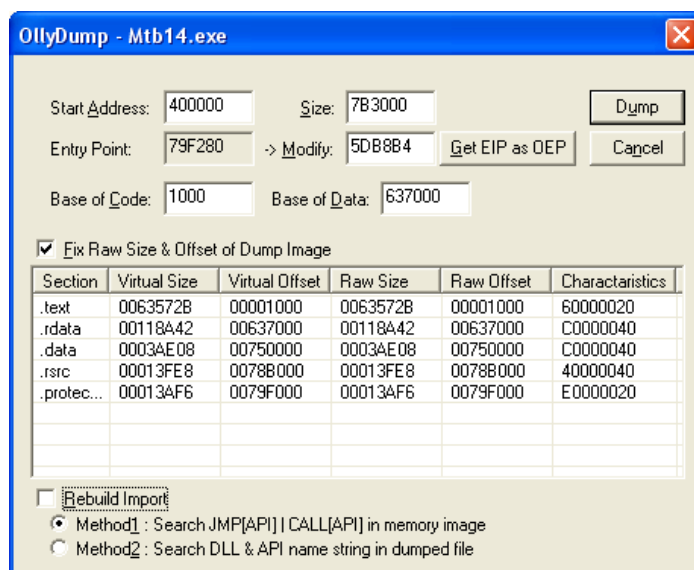


Figure 1.4 Dumping the app

It is time to launch good ol' Imprec. Keep Olly running and app idle.

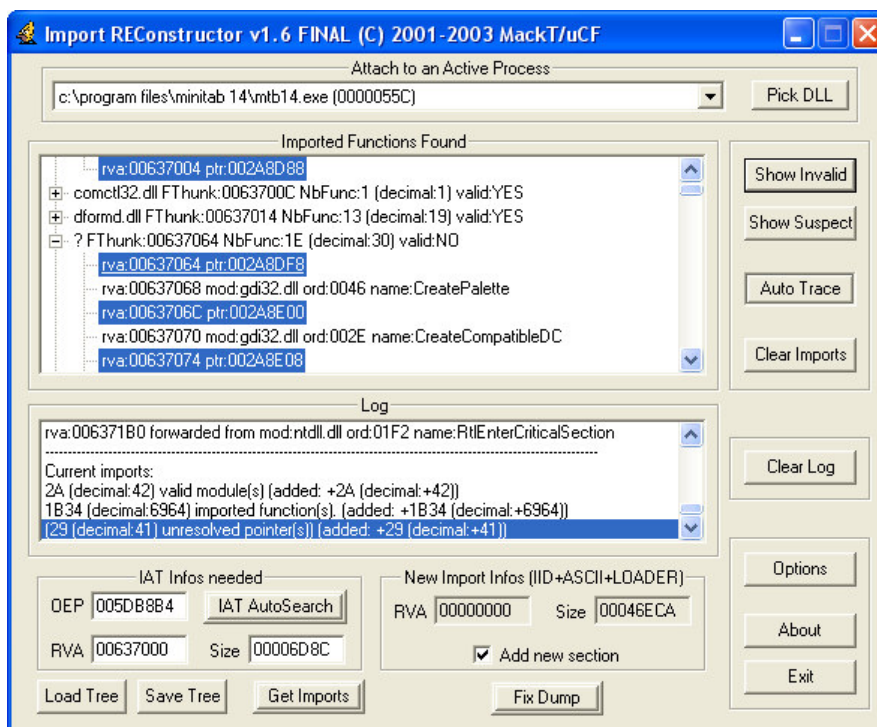


Figure 1.5 Rebuilding with Imprec

Here comes a critical moment while rebuilding, before you start tracing those invalid thunks you MUST proceed running of the main app by pressing F9, or else... Imprec will hang, for sure. So, let the target run. And now right click one of the invalid thunks and choose 'trace level 3' (Trap Flag) Why trace Level 3? And Why not trace level 1? Well, if you were to run a level 1 trace you would end up with all those invalid thunks being RtlEnterCriticalSection. And that's NOT what we want, right?



This is due to the fact that the API calls are redirected within the protector. If you would follow the pointer address of the all the invalid thunks above you will see that all of them points to the same place, a call to this place:

```
00BA3717 55          PUSH EBP      ← Address might be different from time to time
00BA3718 8BEC        MOV EBP,ESP
00BA371A 51          PUSH ECX
00BA371B 53          PUSH EBX
00BA371C 56          PUSH ESI
00BA371D 57          PUSH EDI
00BA371E 60          PUSHAD
00BA371F 8B4424 34    MOV EAX,DWORD PTR SS:[ESP+34]
00BA3723 8945 FC      MOV DWORD PTR SS:[EBP-4],EAX
00BA3726 8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
00BA3729 50          PUSH EAX
00BA372A E8 E9FFFFFF  CALL Mtbl4org.00BA3618 ← This call is of interest :)
00BA372F 83C4 04      ADD ESP,4        ← After call EAX contains the API
00BA3732 894424 34    MOV DWORD PTR SS:[ESP+34],EAX
00BA3736 61          POPAD
00BA3737 5F          POP EDI
00BA3738 5E          POP ESI
00BA3739 5B          POP EBX
00BA373A 8BE5        MOV ESP,EBP
00BA373C 5D          POP EBP
00BA373D C3          RETN
```

It brings you here:

```
00BA3618 55          PUSH EBP
00BA3619 8BEC        MOV EBP,ESP
00BA361B 51          PUSH ECX
00BA361C 833D D43FBA00 00 CMP DWORD PTR DS:[BA3FD4],0
00BA3623 74 7B       JE SHORT Mtbl4org.00BA36A0
00BA3625 68 C052BA00 PUSH Mtbl4org.00BA52C0
00BA362A FF15 9452BA00 CALL DWORD PTR DS:[BA5294] ; ntdll.RtlEnterCriticalSection :)
00BA3630 A1 7052BA00 MOV EAX,DWORD PTR DS:[BA5270]
00BA3635 83C0 01     ADD EAX,1
00BA3638 A3 7052BA00 MOV DWORD PTR DS:[BA5270],EAX
00BA363D 833D 7052BA00 01 CMP DWORD PTR DS:[BA5270],1
00BA3644 75 40       JNZ SHORT Mtbl4org.00BA3686
00BA3646 FF15 A452BA00 CALL DWORD PTR DS:[BA52A4] ; kernel32.GetTickCount
00BA364C 8945 FC      MOV DWORD PTR SS:[EBP-4],EAX
00BA364F 8B4D FC      MOV ECX,DWORD PTR SS:[EBP-4]
00BA3652 2B0D 7452BA00 SUB ECX,DWORD PTR DS:[BA5274]
00BA3658 3B0D D43FBA00 CMP ECX,DWORD PTR DS:[BA3FD4]
00BA365E 77 0B       JA SHORT Mtbl4org.00BA366B
00BA3660 8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
00BA3663 3B15 7452BA00 CMP EDX,DWORD PTR DS:[BA5274]
00BA3669 73 1B       JNB SHORT Mtbl4org.00BA3686
00BA366B 6A 00       PUSH 0
00BA366D 68 22050000 PUSH 522
00BA3672 6A 00       PUSH 0
00BA3674 A1 BC3FBA00 MOV EAX,DWORD PTR DS:[BA3FBC]
00BA3679 FF10        CALL DWORD PTR DS:[EAX]
00BA367B FF15 A452BA00 CALL DWORD PTR DS:[BA52A4] ; kernel32.GetTickCount
00BA3681 A3 7452BA00 MOV DWORD PTR DS:[BA5274],EAX
00BA3686 8B0D 7052BA00 MOV ECX,DWORD PTR DS:[BA5270]
00BA368C 83E9 01     SUB ECX,1
00BA368F 890D 7052BA00 MOV DWORD PTR DS:[BA5270],ECX
00BA3695 68 C052BA00 PUSH Mtbl4org.00BA52C0
00BA369A FF15 F852BA00 CALL DWORD PTR DS:[BA52F8] ; ntdll.RtlLeaveCriticalSection
00BA36A0 8B15 6C52BA00 MOV EDX,DWORD PTR DS:[BA526C] ← Here the calculation of API address begins

00BA36A6 83C2 05     ADD EDX,5
00BA36A9 8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
00BA36AC 2BC2       SUB EAX,EDX
00BA36AE 8945 08     MOV DWORD PTR SS:[EBP+8],EAX
00BA36B1 8B4D 08     MOV ECX,DWORD PTR SS:[EBP+8]
00BA36B4 C1E9 03     SHR ECX,3
00BA36B7 8B15 6852BA00 MOV EDX,DWORD PTR DS:[BA5268]
00BA36BD 8B048A     MOV EAX,DWORD PTR DS:[EDX+ECX*4] ← Done :) API is in EAX now
00BA36C0 8BE5        MOV ESP,EBP
00BA36C2 5D          POP EBP
00BA36C3 C3          RETN
```



Okay, clearly the calls are redirected and calculated at runtime. So instead of replacing these calls by hand you can use the level 3 trace function in Imprec. With the following result:

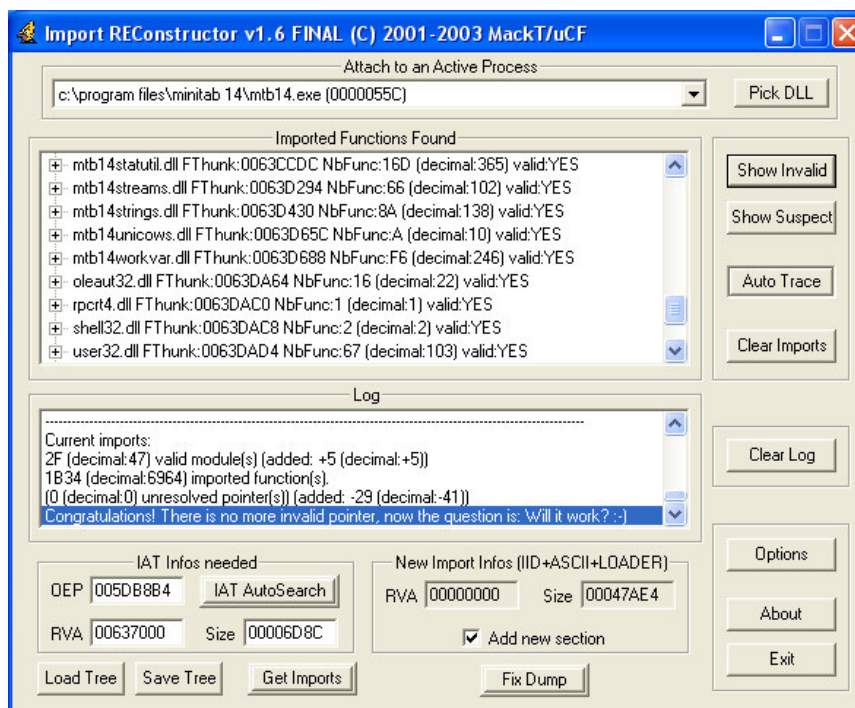


Figure 1.6 Correct looking thunks



Yes, the question is, will it work?

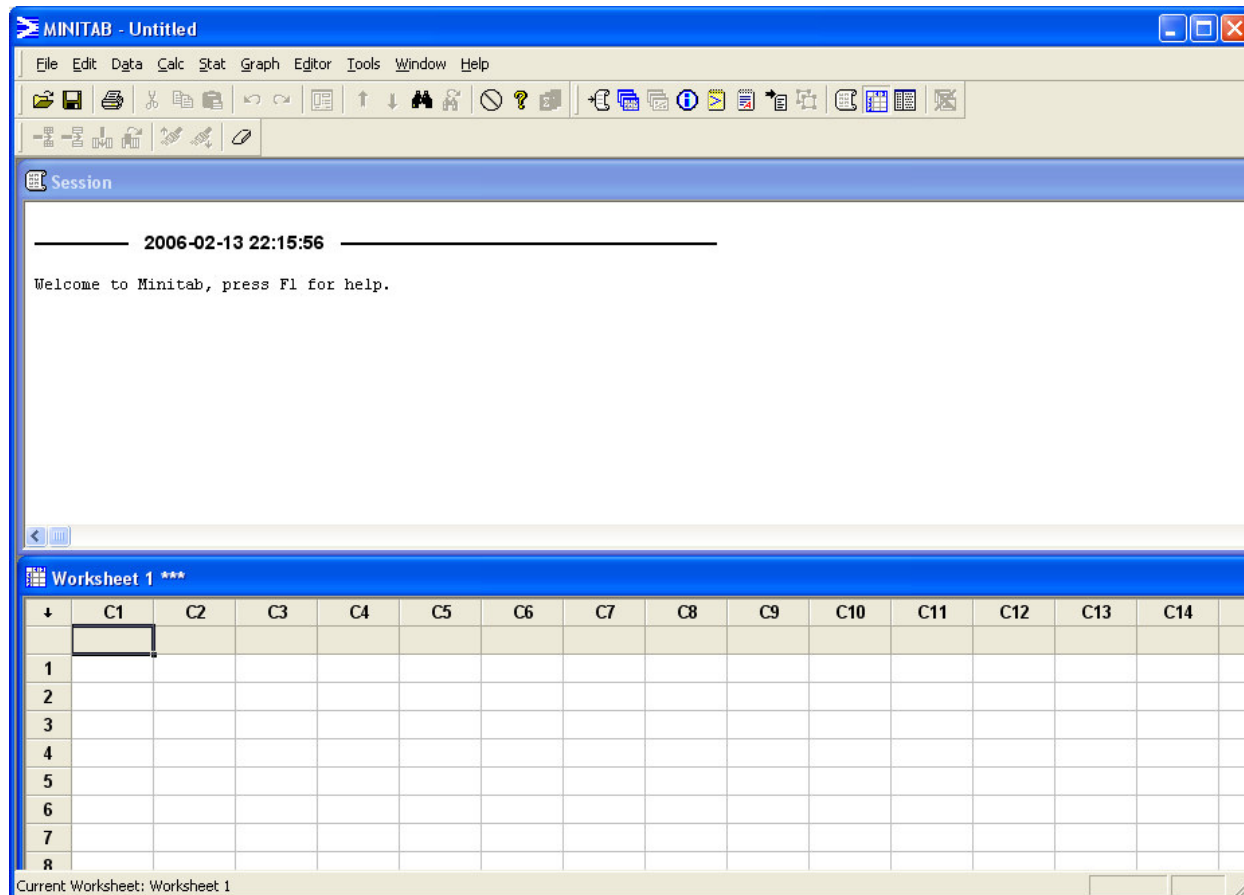


Figure 1.7 Yay! It runs..

Horaay .. It runs! And the ugly and annoying trial screen is very gone, indeed.

4. Conclusions

Although I've stumbled across one application that uses HASP SL, it seems like HASP SL is no harder to dodge than any other average protector.

5. Greetings

Special thanks goes to all ARTeam members (you are one bunch of hilarious ppl)

Greetings fly out to:

- * ALL that in some way contributes to the reverse engineering scene.
- * Those ppl who write tutorials to share the fun and the knowledge.

potassium of ARTeam

<http://cracking.accessroot.com>