

NAGSCREEN ET

SIMULATION DE CLIC

SUR WINZIP 9.0 SR1

SOMMAIRE

I Introduction

II Le nagscreen noir et le nagscreen blanc

III Simuler un clic sur le bouton « use evaluation version »

IV Remerciements

I Introduction

dans ce tut, j'explique:

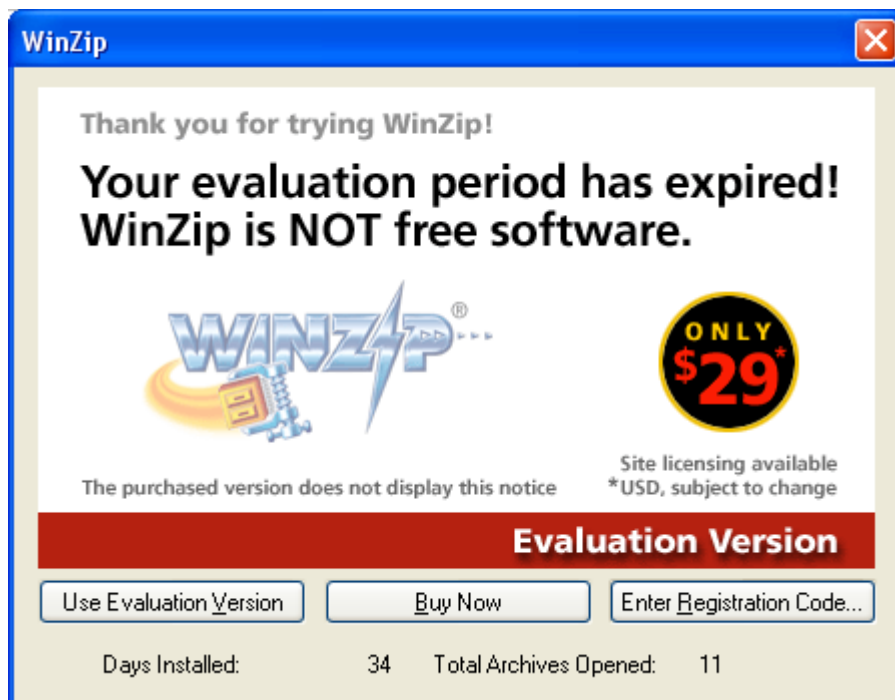
- ce qui déclenche l'apparition du nagscreen noir ou du nagscreen blanc
 - comment simuler un clic de souris sur le bouton « Use Evaluation Version » du nagscreen
- Ce tut concerne la version 9.0 SR1 de winzip. Elle peut être téléchargée ici:

<http://download.oldapps.com/WinZip/winzip90.exe>

<http://download.oldversion.com/winzip90sr1.exe>

remarque: la première partie sur le nagscreen noir ou blanc est indépendante de la deuxième partie sur la simulation de clic.

voilà les deux nagscreen qui peuvent apparaître quand on est pas enregistré:



II Le nagscreen noir et le nagscreen blanc

Notre nagscreen est une dialogbox

on va donc mettre un breakpoint sur la fonction DialogBoxParamA
ollydbg breake et on voit les paramètre de la fonction dans la pile:

```
0012F91C    00400000    |hInst = 00400000
0012F920    00000B6D    |pTemplate = B6D
0012F924    00000000    |hOwner = NULL
0012F928    004028B1    |DlgProc = WINZIP32.004028B1
0012F92C    00000000    \lParam = NULL
```

La fonction DlgProc qui gère les messages commence donc à l'adresse 4028B1
on met un breakpoint sur 4028B1 avec olly --> on arrive sur une fonction switch:

Switch (cases 2..111)

Case 111 (WM_COMMAND) --> Indique que la fenetre a reçu un événement de type interaction
(clique, pression de touche, ..)

Case 10 (WM_CLOSE) --> Indique que l'utilisateur a emis le desir de quitter l'application en cliquant
sur la croix en haut a droite

Case 2 (WM_DESTROY) --> Indique que la fenetre va été detruite

Case 110 (WM_INITDIALOG) --> Indique que la dialogbox va être affichée. Ca laisse la possibilité
d'exécuter quelques actions avant (rajouter une image, du texte, un timer....)

Case 2B (WM_DRAWITEM) --> message envoyé quand l'aspect visuel d'un bouton a changé

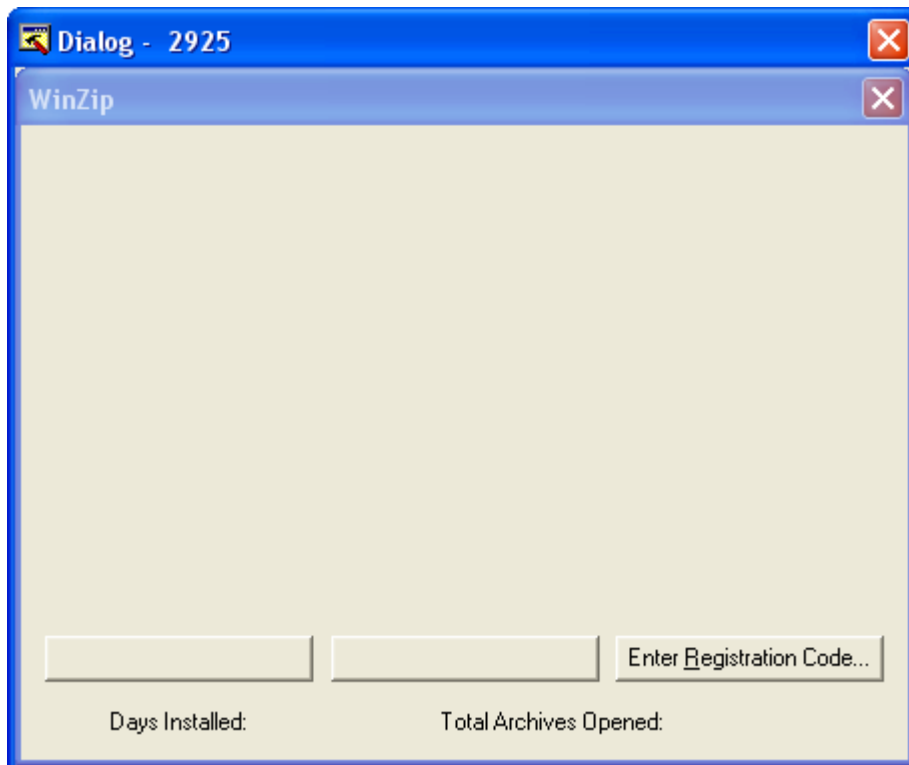
Case 6 (WM_ACTIVATE) --> message envoyé quand une fenetre est activé ou désactivée

en fonction des événements, le prog execute donc certaines portions de code:

--> il execute le code WM_Command quand il y a un clic sur un bouton par exemple.

on va regarder à quoi ressemble la dialogbox du nagscreen avec ressourcehacker

--> on va dans Dialog, et c'est la 2925 qui correspond au nagscreen



Petite surprise --> les deux premiers boutons n'ont pas de nom, et il n'y a pas non plus d'image (on va voir pourquoi dans la partie suivante du tut....)

Autre surprise --> il n'y a pas d'image sur la dialogbox. --> l'image est chargée en fonction, et affiche soit le nagscreen noir, soit le nagscreen blanc

dans ressourcehacker, on a deux images de winzip
on va dans bitmap, et on regarde les images 218 et 219



On va essayer de comprendre quelle partie de code décide de l'affichage du nagscreen noir ou du nagscreen blanc. --> on met un breakpoint sur la ligne Case 110 (WM_INITDIALOG) qui initialise la boite de dialogue, on enleve le breakpoint sur 004028B1 (début de la dialogProc) et F9:

<pre>00402A41 PUSH ESI 00402A42 CALL WINZIP32.004026D9 00402A47 MOV DWORD PTR SS:[ESP], 2B0B 00402A4E PUSH ESI 00402A4F CALL DWORD PTR DS:[&USER32.GetDlgItem] 00402A55 PUSH WINZIP32.00401550 00402A5A MOV EDI, EAX 00402A5C PUSH -4 00402A5E PUSH EDI 00402A5F MOV DWORD PTR DS:[4E3B8C], ESI 00402A65 CALL DWORD PTR DS:[&USER32.SetWindowLongA] 00402A6B MOV DWORD PTR DS:[4E3B20], EAX 00402A70 MOVZX EAX, WORD PTR DS:[4E3B28] 00402A77 PUSH EAX 00402A78 MOVZX EAX, WORD PTR DS:[4E3B34] 00402A7F PUSH EAX 00402A80 PUSH EDI 00402A81 PUSH 1 00402A83 CALL WINZIP32.0048BA57 00402A88 PUSH ESI 00402A89 CALL WINZIP32.0048B4A1 00402A8E PUSH ESI 00402A8F CALL WINZIP32.0045789D 00402A94 XOR EAX, EAX 00402A96 ADD ESP, 18 00402A99 INC EAX 00402A9A JMP WINZIP32.004028B9</pre>	<p>Case 110 (WM_INITDIALOG) of switch 004028EB on rentre dans ce call</p> <div style="border-left: 1px solid black; padding-left: 10px; margin-left: 10px;"><p>hWnd GetDlgItem NewValue = 401550 Index = GWL_WNDPROC hWnd SetWindowLongA</p><p>Arg1 WINZIP32.0045789D</p></div>
--	---

on rentre dans le premier call à l'adresse 402A42 avec F7. Une partie de ce code détermine si on affiche le nagscreen noir ou le nagscreen blanc:

004027BA	CALL WINZIP32.004023A8	decide d'afficher le nagscreen noir ou blanc
004027BF	TEST EAX,EAX	eax = 0 --> on saute --> nagscreen noir
004027C1	POP ECX	si eax = 1 nagscreen blanc et timer
004027C2	JE SHORT WINZIP32.004027F7	si is not taken --> il y a le timer...
004027C4	CALL EBX	
004027C6	PUSH 0	Timerproc = NULL
004027C8	PUSH 32	Timeout = 50. ms
004027CA	PUSH 7B	TimerID = 7B (123.)
004027CC	PUSH DWORD PTR SS:[EBP+8]	hWnd
004027CF	MOV DWORD PTR DS:[4E3B24],EAX	SetTimer
004027D4	CALL DWORD PTR DS:[<USER32.SetTimer>]	
004027DA	MOV BYTE PTR DS:[4E3B74],1	
004027E1	MOV DWORD PTR DS:[4E3B34],0DC	
004027EB	MOV DWORD PTR DS:[4E3B28],0DD	
004027F5	JMP SHORT WINZIP32.0040283B	si nagscreen blanc on saute en 40283B
004027F7	LEA EAX,DWORD PTR SS:[EBP-20]	
004027FA	PUSH EAX	
004027FB	PUSH 0B65	
00402800	PUSH DWORD PTR SS:[EBP+8]	
00402803	CALL ESI	
00402805	PUSH EAX	
00402806	CALL EDI	
00402808	MOV EAX,DWORD PTR SS:[EBP-1C]	si nagscreen noir, on redimensionne la fenetre
0040280B	SUB EAX,DWORD PTR SS:[EBP-C]	avec movewindow pour pas afficher day installed
0040280E	PUSH 0	et total archive opened
00402810	PUSH EAX	Repaint = FALSE
00402811	MOV EAX,DWORD PTR SS:[EBP-8]	Height
00402814	SUB EAX,DWORD PTR SS:[EBP-10]	
00402817	PUSH EAX	
00402818	PUSH DWORD PTR SS:[EBP-C]	Width
0040281B	PUSH DWORD PTR SS:[EBP-10]	Y
0040281E	PUSH DWORD PTR SS:[EBP-8]	X
00402821	CALL DWORD PTR DS:[<USER32.MoveWindow>]	hWnd
00402827	MOV DWORD PTR DS:[4E3B34],0DA	MoveWindow
00402831	MOV DWORD PTR DS:[4E3B28],0DB	
0040283B	CALL WINZIP32.00402266	

on voit que c'est le call 4023A8 qui décide de tout. Au sortir de call

- si eax=0 --> affiche le nagscreen noir
- si eax=1 --> affiche le nagscreen blanc

--> si nagscreen noir --> on redimensionne la fenetre pour pas afficher le bas de la fenetre ou il est indiqué Days Installed et Total Archive Opened

--> si nagscreen noir --> on installe un timer avec SetTimer qui nous empeche de cliquer sur « Use Evaluation Version » avant un certain moment.

On va rentrer dans le call 4023A8 pour savoir ce qui determine l'affichage de l'un ou l'autre nagscreen. On trace et on voit les strings suivantes:

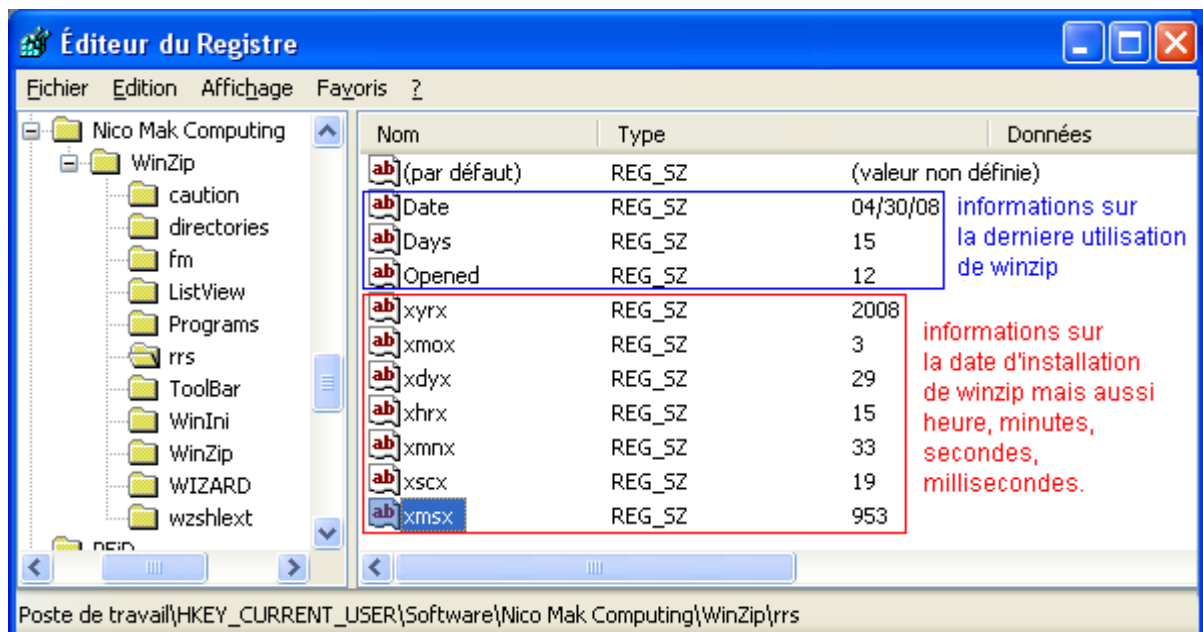
00402486	MOV EDI,WINZIP32.004C4700	; ASCII "winzip32.ini"
0040248B	MOV ESI,WINZIP32.004C472C	; ASCII "rrs"
0040249C	PUSH WINZIP32.004C4754	; ASCII "xyrx"
004024B4	PUSH WINZIP32.004C474C	; ASCII "xmox"
004024CC	PUSH WINZIP32.004C4744	; ASCII "xdyx"
004024DD	MOV EBX,WINZIP32.004C48EC	; ASCII "xhrx"

et on arrive finalement ici:

0040256E	MOV DWORD PTR SS:[EBP-8],15	21 jours de demo dans "evaluation"
00402575	CALL WINZIP32.0041E9D7	call qui determine les "Days Installed"
0040257A	CMP EDX,1792	
00402580	JL SHORT WINZIP32.00402592	si "Day Installed" < 30 --> 21 jours de Demo
00402582	JG SHORT WINZIP32.0040258B	si > 30 on a 10 jours de demo au lieu de 21
00402584	CMP EAX,F8648000	
00402589	JB SHORT WINZIP32.00402592	
0040258B	MOV DWORD PTR SS:[EBP-8],0A	10 jours de demo dans "evaluation"

cette partie de code détermine si on a 21 jours ou 10 jours de démonstration pour calculer ça on calcule depuis quand winzip a été installé sur l'ordinateur Pour faire ça, il compare la date systeme à la date d'installation qui a été enregistré dans la base de registre.

On clique sur le menu démarrer de windows, executer et regedit on va voir la base de registres:
HKEY_CURRENT_USER\Software\Nico Mak Computing\WinZip\rrs



à l'installation, WinZip sauve la date d'installation dans le registre (bloc rouge)

xrx est l'année de l'installation, xmox le mois, xdyx, le jour, xhrx l'heure, xmnx les minutes, xscx les secondes, xmsx les millisecondes.

Le prog compare cette date stockée dans la base de registre à la date actuelle du système.

Si le prog a été installé depuis plus de 30 jours, on a que 10 jours de démonstration au lieu de 21 jours normalement. On continue dans ollydbg et on voit un autre passage intéressant:

004025E2 LEA EAX, DWORD PTR SS:[EBP-2E0]	la premiere date = date actuelle systeme
004025E8 PUSH EAX	
004025E9 LEA EAX, DWORD PTR SS:[EBP-40C]	la deuxieme date = ancienne date du registre
004025EF PUSH EAX	
004025F0 CALL WINZIP32.004A8AE0	teste si date systeme = date base registre
004025F5 ADD ESP, 44	
004025F8 TEST EAX, EAX	
004025FA SETNE AL	
004025FD TEST AL, AL	
004025FF JE SHORT WINZIP32.00402609	on saute si dates identiques
00402601 INC DWORD PTR DS:[4E3B78]	on augmente "Days" de 1 si les dates sont differentes
00402607 JMP SHORT WINZIP32.00402613	
00402609 CMP BYTE PTR SS:[EBP+81], 0	

ça compare la date actuelle du système, à la dernière date d'utilisation de winzip stockée dans la variable « Date » de la base de registre. Si c'est différent, on augmente la variable « Days » de la base de registre de 1.

Remarque: ça signifie qu'on peut pas abuser winzip en modifiant la date système de windows pour lui mettre une date antérieure car dans ce cas, la date est différente, et il augmente « Days » (les jours d'utilisation) de 1.

On trace encore un peu avec ollydbg, et on voit quelque chose d'intéressant à la fin de la procédure:

004026AF MOV EAX, DWORD PTR SS:[EBP-8]	on met "evaluation" (10 ou 21) dans eax
004026B2 CMP DWORD PTR DS:[4E3B78], EAX	compare "Day" a "evaluation"
004026B8 POP EDI	
004026B9 POP ESI	
004026BA POP EBX	
004026BB JBE SHORT WINZIP32.004026CA	si "Day" > "evaluation" --> no jump
004026BD CMP DWORD PTR DS:[4E3B80], EAX	compare "Opened" a "evaluation"
004026C3 JBE SHORT WINZIP32.004026CA	si "Opened" > "evaluation" --> no jump
004026C5 XOR EAX, EAX	si "Day" et "Opened" > "evaluation"
004026C7 INC EAX	eax = 1 et nagscreen blanc
004026C8 JMP SHORT WINZIP32.004026CC	
004026CA XOR EAX, EAX	sinon eax = 0 et nagscreen noir
004026CC MOV ECX, DWORD PTR SS:[EBP-4]	
004026CF XOR ECX, DWORD PTR SS:[EBP+4]	
004026D2 CALL WINZIP32.004A857C	
004026D7 LEAVE	
004026D8 RETN	

Dans [ebp-8] (ce que j'ai nommé « evaluation »), on a soit 10 jours, soit 21 jours.
Le prog compare « Days » (le nombre de jours d'utilisation) à « évaluation » mais aussi
« Opened » (le nombre d'archives ouvertes avec winzip à « evaluation »
si « Days » > « evaluation » et « Opened » > « evaluation » --> on a dépassé le temps de
démonstration et on affiche le nagscreen blanc avec « Your evaluation version has expired »

si c'est pas le cas on garde le nagscreen noir

Pour résumer:

si installé depuis + de 30 jours --> on a que 10 jours pour le tester
--> affiche le nagscreen blanc si > 10 jours d'utilisation et > 10 archives ouvertes
si installé depuis – de 30 jours --> on a 21 jours pour le tester
--> affiche le nagscreen si > 21 jours d'utilisation et > 21 archives ouvertes

Les informations entourées en bleu dans le registre (Date, Days, Opened) sont mises à jour après
chaque utilisation de winzip avec la fonction RegSetValue

Si on veut que le nagscreen blanc n'apparaisse jamais (et avec lui le timer qui nous empêche de
cliquer rapidement sur le bouton «Use Evaluation Version ») on peut noper la ligne suivante:

```
004026C8 JMP SHORT WINZIP32.004026CC
```

De cette manière, on aura toujours 0 dans eax et le nagscreen noir.

III Simuler un clic sur le bouton « use evaluation version »

Fixer le bouton « Use Evaluation Version »

On a remarqué avec RessourceHacker que les deux premiers boutons n'ont pas de nom. Le nom est en fait déterminé au hasard par winzip. Le bouton 1 peut ainsi s'appeler « Use Evaluation Version » ou « Buy Now ». (cf introduction avec les deux captures d'écran)

Pour nous, ça serait mieux que le bouton 1 soit toujours le bouton « Use Evaluation Version » et le bouton 2 « Buy Now » (sinon, on ne sait pas vraiment sur lequel cliquer...)

Dans le premier call de la partie Case 110 (WM_INITDIALOG) de tout à l'heure, on voit un bout de code qui affecte au hasard leurs noms aux deux premiers boutons.

00402720	MOV EBX,DWORD PTR DS:[&KERNEL32]	kernel32.GetTickCount
00402732	CALL EBX	GetTickCount
00402734	TEST AL,1	
00402736	JE SHORT WINZIP32.0040274C	
00402738	MOV WORD PTR DS:[4E3B7C],0B55	identifiant du 1er bouton
00402741	MOV WORD PTR DS:[4E3B2C],0B56	identifiant du 2eme bouton
00402744	JMP SHORT WINZIP32.0040275E	
0040274C	MOV WORD PTR DS:[4E3B7C],0B56	
00402755	MOV WORD PTR DS:[4E3B2C],0B55	
0040275E	TEST AL,2	

La fonction GetTickCount retourne le nombre de millisecondes écoulées depuis que le système a commencé dans EAX. On regarde ensuite si al=1. Si c'est le cas le deuxième bouton aura la valeur « Use Evaluation Version » et le premier bouton aura la valeur « Buy Now »

si al <> 1 on a:

premier bouton = « Use Evaluation Version »

deuxième bouton = « Buy Now »

quand on regarde les boutons avec un éditeur de ressources comme RessourceHacker, on voit ça:

premier bouton

CONTROL "", 2901, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE, 8, 157, 89, 14

deuxième bouton

CONTROL "", 2902, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE, 103, 157, 89, 14

troisième bouton

CONTROL "Enter &Registration Code...", 2922, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE, 198, 157, 89, 14

le premier bouton a comme identifiant 2901. Quand on convertit ce chiffre décimal en hexadécimal, on obtient 0B55. Pour 2902 (l'identifiant du deuxième bouton) on obtient 0B56 pour la conversion hexadécimale --> ce sont bien 0B55 et 0B56 qui sont utilisées pour identifier les boutons dans le code ci dessus.

Pour que le premier bouton soit toujours le bouton « Use Evaluation Version », on copie la ligne suivante:

```
00402736 JE SHORT WINZIP32.0040274C
```

Postmessage

On peut maintenant s'attaquer à la partie la plus intéressante du tutorial: simuler un clic sur le bouton « Use Evaluation Version »

Question: que se passe-t-il quand on clique sur un bouton?

--> ça génère des messages, et particulièrement les messages WM_LBUTTONDOWN (clic appuyé) et WM_LBUTTONUP (clic relâché). Ces messages sont interceptés par notre DialogProc, notamment par la partie Case 111 (WM_COMMAND)

Tout ce qu'on a à faire pour simuler un clic sur un bouton, c'est donc d'envoyer un message au bouton disant WM_LBUTTONDOWN puis WM_LBUTTONUP. Deux fonctions windows permettent d'envoyer ce genre de messages, SendMessage or PostMessage

on veut regarder comment fonctionne Postmessage

cette fonction demande 4 parametres:

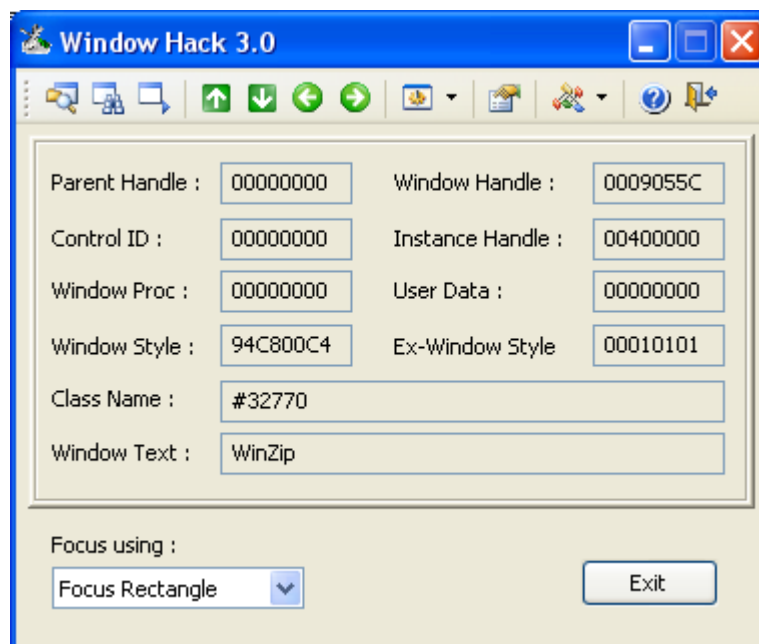
```
BOOL PostMessage (  
    HWND hWnd --> handler du bouton qu'on veut cliquer  
    UINT Msg --> message ID que l'on veut envoyer  
    WPARAM wParam -> optionnel  
    LPARAM lParam -> optionnel  
);
```

Pour simuler un clic, on peut envoyer le message BM_CLICK. Ça envoie au bouton les messages WM_LBUTTONDOWN puis WM_LBUTTONUP. Le message ID qui correspond à BM_CLICK est F5

les handlers

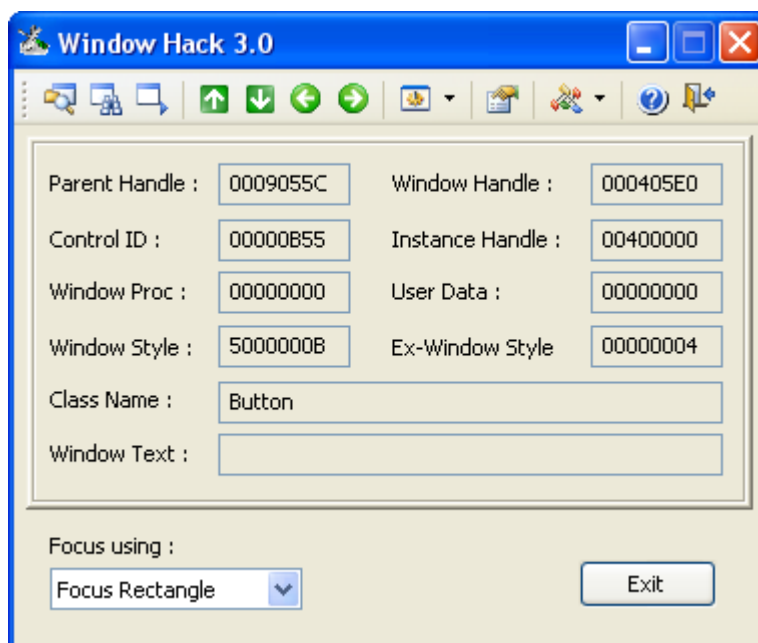
Maintenant le handler du bouton --> d'abord, qu'est ce qu'un handler? --> c'est une valeur que windows donne aux objet (une fenetre, un bouton, un menu) pour y acceder ensuite. Chaque objet de l'application à un handler différent. Pour illustrer ça, on va utiliser WindowHack, un outil vraiment sympa <http://www.geocities.com/asmsoft3264/download/WindowHack30.zip>

On lance winzip --> le nagscreen s'affiche --> on lance windowHack, on clique sur le premier bouton en haut à gauche (Find Window Under Cursor) et on selectionne la fenetre de nagscreen, et on obtient ça: (remarque faire bien attention à selectionner toute la fenetre et pas seulement l'image --> la zone cliquée est materialisée par un cadre vert)



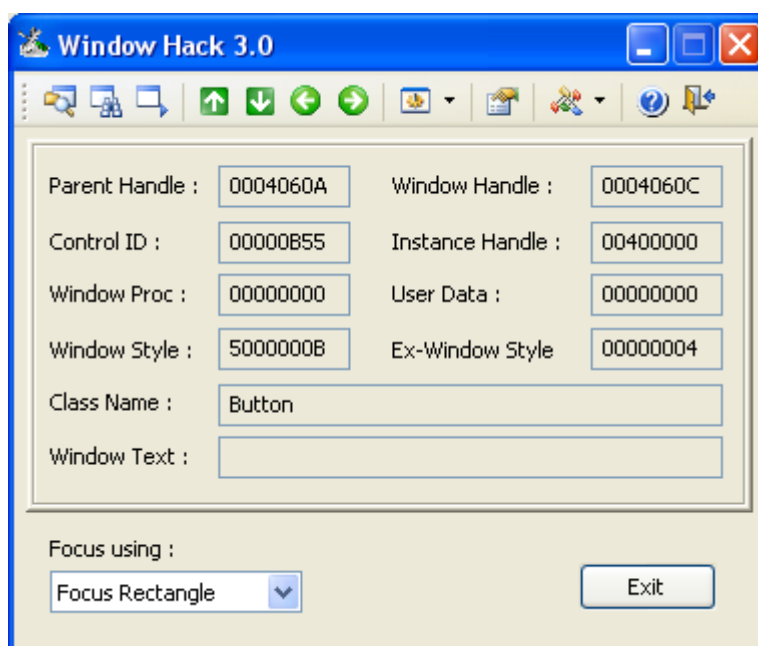
--> le handler de la fenetre est 0009055C

Avec « Find Window Under Cursor » On selectionne maintenant le premier bouton du nagscreen et on obtient ça:



Le handler du bouton est 000405E0
 (on retrouve le handler de la fenêtre dans Parent Handle 0009055C)

Le probleme, c'est que si je relance winzip, et que je regarde à nouveau le handler du bouton, j'obtiens un handler différent:



Le handler du bouton est maintenant 004060C, celui de la fenetre 0004060A
 En fait les handler sont définis au debut de l'execution du prog et ne sont valables que jusqu'à la fin du prog --> conclusion, on ne peut pas fixer definitivement la valeur du handler dans notre code pour simuler un clic, on est obligé d'utiliser des fonctions pour le récupérer.

Au contraire du handler, certaines valeurs restent identiques --> par exemple le control ID du bouton reste à B55 (on avait trouvé cette valeur avec Ressourcehacker mais en décimal), le titre de la fenetre...

Comment récupérer le handler du bouton?
 On va procéder par étapes --> on va d'abord récupérer le handler de la fenetre grace au nom de la fenetre (WinZip). On va ensuite recuperer le handler du bouton grace au handler de la fenetre sur laquelle il se trouve, et du control ID du bouton.

Une fois qu'on a le handler du bouton et le message à envoyer (BM_CLICK), on a plus qu'à appeler la fonction PostMessageA pour simuler le clic sur le bouton

Voilà les instructions assembleur qu'on va utiliser:

```
PUSH 00420600 ; /Title = "WinZip"
PUSH 0 ; |Class = 0
CALL USER32.FindWindowA ; \FindWindowA

PUSH 66 ; /ControlID = 66 (102.)
PUSH EAX ; |hWnd
CALL USER32.GetDlgItem ; \GetDlgItem

PUSH 0 ; /lParam = 0
PUSH 0 ; |wParam = 0
PUSH 0F5 ; |Message = BM_CLICK
PUSH EAX ; |hWnd
CALL USER32.PostMessageA ; \PostMessageA
```

--> la fonction FindWindowA nous permet de recuperer le handler de la fenetre à partir de son titre (la variable 420600 pointe vers «WinZip») --> le handler de la fenetre est mis dans eax

--> la fonction GetDlgItem nous permet de recuperer le handler du bouton à partir du handler de la fenetre (mis dans eax) et du ControlID du bouton --> le handler du bouton est mis dans eax

--> on a l'handler du bouton dans eax, le message OFA (BM_CLICK) donc on peut simuler le clic...

premiere question: quand doit on appeler notre code asm pour simuler le clic?

--> on peut le faire tout à la fin de la partie Case 110 (WM_INITDIALOG) car à ce moment les boutons sont déjà initialisés.

<pre>00402A35 > CALL WINZIP32.0048B589 00402A3A . PUSH 0 00402A3C . JMP WINZIP32.00402B72 00402A41 > PUSH ESI 00402A42 . CALL WINZIP32.004026D9 00402A47 . MOV DWORD PTR SS:[ESP],2B0B 00402A4E . PUSH ESI 00402A4F . CALL DWORD PTR DS:[<&USER32.GetDlgItem>] 00402A55 . PUSH WINZIP32.00401550 00402A5A . MOV EDI,EAX 00402A5C . PUSH -4 00402A5E . PUSH EDI 00402A5F . MOV DWORD PTR DS:[4E3B8C],ESI 00402A65 . CALL DWORD PTR DS:[<&USER32.SetWindowLong>] 00402A6B . MOV DWORD PTR DS:[4E3B20],EAX 00402A70 . MOVZX EAX,WORD PTR DS:[4E3B28] 00402A77 . PUSH EAX 00402A78 . MOVZX EAX,WORD PTR DS:[4E3B34] 00402A7F . PUSH EAX 00402A80 . PUSH EDI 00402A81 . PUSH 1 00402A83 . CALL WINZIP32.0048BA57 00402A88 . PUSH ESI 00402A89 . CALL WINZIP32.0048B4A1 00402A8E . PUSH ESI 00402A8F . CALL WINZIP32.0045789D 00402A94 . XOR EAX,EAX 00402A96 . ADD ESP,18 00402A99 . INC EAX 00402A9A . JMP WINZIP32.00402BF9 00402A9F > MOV ESI,DWORD PTR SS:[EBP+14] 00402AA2 . CALL WINZIP32.00401B0D 00402AA7 > MOVZX EAX,AL 00402AAA > JMP WINZIP32.00402BF9 00402AAF > CMP BX,1</pre>	<pre>Case 10 (WM_CLOSE) of switch 004028EB Case 110 (WM_INITDIALOG) of switch 004028EB hWnd GetDlgItem NewValue = 401550 Index = GWL_WNDPROC hWnd SetWindowLongA Arg1 WINZIP32.0045789D --> fin de wm initdialog Case 2B (WM_DRAWITEM) of switch 004028EB Case 6 (WM_ACTIVATE) of switch 004028EB</pre>
--	--

La partie WM_INITDIALOG fini par un JMP 402BF9 (comme la partie WM_DRAWITEM). On va donc remplacer ce JMP 402BF9 par un JMP notre code. A la fin de notre code, on remettra un JMP 402BF9. Voilà ce qui se passe en 402BF9

```
00402BF9 > POP EDI
00402BFA . POP ESI
00402BFB . POP EBX
00402BFC . POP EBP
00402BFD . RETN 10
```

En 402BF9, ça restaure juste les registres, et ça quitte.

question suivante: ou écrire notre code asm?

Pour écrire notre code asm, on choisi un endroit ou il y a de la place, et on ça gene pas --> par exemple à la fin des instructions du prog en 4C219D

Les trois fonctions qu'on veut utiliser (FindWindowA, GetDlgItem, PostmessageA) sont déjà présentes dans Winzip donc on peut les appeler sans problème.

```
004C4540 .rdata      Import  USER32.FindWindowA
004C44E4 .rdata      Import  USER32.GetDlgItem
004C4564 .rdata      Import  USER32.PostMessageA
004C418C .rdata      Import  KERNEL32.Sleep
```

Il faut aussi une reference vers le titre de la fenetre du nagscreeen (« WinZip ») se terminant par le caractère nul. On recherche ça dans la mémoire, et on trouve cette référence en premier.

004C46F8 57 69 6E 5A 69 70 00 WinZip.

Voilà, tout est OK, reste plus qu'a écrire le code:

```
00402A9A JMP 004C219D ; vers notre routine
```

```
004C219D PUSH 004C46F8 ; /Title = "WinZip"
004C21A2 PUSH 0 ; |Class = 0
004C21A4 CALL [004C4540] ; \FindWindowA
004C21AA PUSH 0B55 ; /ControlID = B55 (2901.)
004C21AF PUSH EAX ; |hWnd
004C21B0 CALL [004C44E4] ; \GetDlgItem
004C21B6 PUSH 0 ; /lParam = 0
004C21B8 PUSH 0 ; |wParam = 0
004C21BA PUSH 0F5 ; |Message = BM_CLICK
004C21BF PUSH EAX ; |hWnd
004C21C0 CALL [004C4564] ; \PostMessageA
004C21C6 PUSH 100 ; /Timeout = 256. ms
004C21CB CALL [004C418C] ; \Sleep
004C21D1 JMP 00402BF9 ; retourne vers le prog
```

A noter qu'on a rajouté la fonction Sleep qui permet de suspendre l'execution du code pendant 256 microsecondes (sinon, le message BM_CLICK arrive trop tot et ne parviens pas à la boucle de message)

On sauvegarde le fichier sous le nom WINZIP32.EXE (un nom différent provoque un message d'erreur). --> c'est fini...

IV Remerciements

Mon tut s'est beaucoup inspiré du tutorial suivant (grands remerciements à l'auteur bb)
<http://www.woodmann.com/fravia/bbnag1.htm>

Remerciements à tous les membres de <http://deezdynasty.xdir.org/> et de Forumcrack et plus particulièrement TiGa et skirby pour leur aide sur la simulation de click, sp0ke et uLysse_31 pour m'avoir redonné envie d'écrire des tuts

mars - le 02 avril 2008