EXPLOITING Y REVERSING

USANDO HERRAMIENTAS

GRATUITAS (PARTE 4)

Hemos visto en los tres desensambladores interactivos, IDA FREE, RADARE y GHIDRA como analizar los dos primeros ejercicios llamados stacks, que son los más sencillos.

Seguiremos en esta parte analizando los ejercicios llamados stacks 3 y 4 pero antes necesitamos introducir un nuevo concepto, el de caracteres inválidos o bad chars.

## Definicion de caracter invalido o bad char

Los badchars o caracteres inválidos o “malos” son caracteres que recibe el programa objetivo, y este los filtra y actúan como delimitadores o, mediante sus algoritmos internos, los elimina o los sustituye por otros valores, haciendo inservible nuestra shellcode.

La búsqueda de badchars es una parte crucial en la metodología de la creación de exploits, ya que, si estos no son localizados y evitados durante la generación del payload, lo convierten en inservible ya que este no se interpretaría correctamente en el sistema objetivo.

Explicando un poco la definición anterior, el tema pasa por los datos que le enviamos a un programa, este los ingresa y los procesa y en ese proceso que realiza hasta llegar al punto de explotación, puede ocurrir que chequee, filtre, sustituya o bloquee ciertos caracteres lo cual impida que los utilicemos en nuestro payload, obviamente esto complica la explotación y habrá que tenerlo en cuenta al momento de crear el exploit.

Hay muchas formas de chequear si en la explotación de un programa, el mismo produce caracteres inválidos, una es obviamente analizando el programa desde donde ingresan nuestros datos hasta el punto de explotación y ver que hace con los mismos. Este método es el más preciso pero a veces es sumamente tedioso si el programa realiza múltiples chequeos, copias y manipulaciones de los datos ingresados.

Otra opción es pasar una cadena que contenga todos los caracteres posibles y ir viendo qué ocurre con ella y con varios intentos, llegar a generar una cadena que pase completa y se copie entera al buffer de destino, eliminando los caracteres inválidos.

Normalmente se evita pasar en un primer intento los caracteres 0xA, 0xD, 0x0 y 0xFF que son los principales caracteres inválidos, si el resto pasa bien y llega a la zona de explotación sin problemas se pueden ir agregando estos cuatro, uno a uno para ver si no hay problemas con ellos.

En el caso de un buffer overflow, podriamos pasar la cadena de caracteres de prueba y ver si se copia al buffer y mirar si llegan todos los que enviamos.

Un código en python 3 para armar un payload para testear los caracteres inválidos seria.

SCRIPT BÁSICO PARA TESTEAR BAD CHARS

**payload = b""**

**excluded = (0x0, 0xa, 0xd, 0xff)**

**for i in range(256):**

**if i not in excluded:**

**payload += bytes([i])**

**print (payload)**

Vemos que en excluded están los caracteres que suponemos inválidos, a los cuales iremos quitando o agregando más, según las pruebas que vayamos realizando.

Aquí realice una modificación al script del stack 1 para probar los caracteres inválidos, los que hallemos serán válidos para los 4 ejercicios ya que todos procesan los datos de la misma forma, ingresando por medio de la funcion gets, la cual los copia al buffer **buf** de destino.

ADAPTANDO EL SCRIPT BASICO PARA PROBAR BAD CHARS EN STACK 1

**import sys**

**from subprocess import Popen, PIPE**

**payload = b""**

**excluded = (0x0, 0xa, 0xd, 0xff)**

**for i in range(256):**

**if i not in excluded:**

**payload += bytes([i])**

**print (payload)**

**p1 = Popen(r"STACK1\_VS\_2017.exe", stdin=PIPE)**

**print ("PID: %s" % hex(p1.pid))**

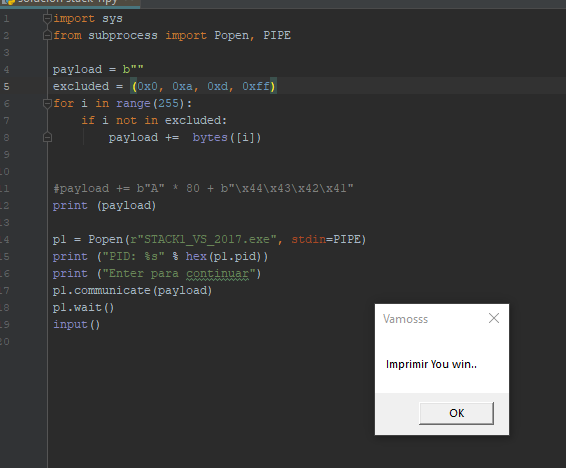
**print ("Enter para continuar")**

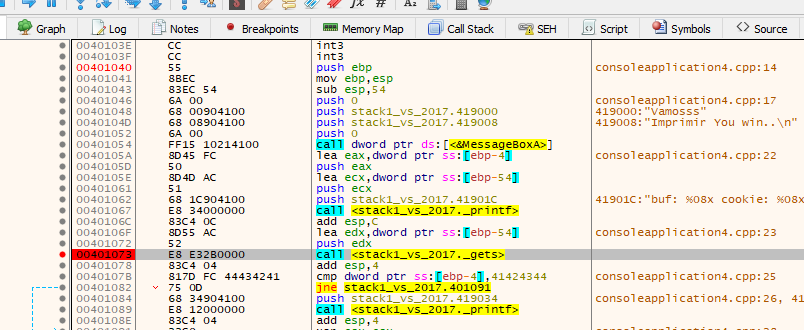
**p1.communicate(payload)**

**p1.wait()**

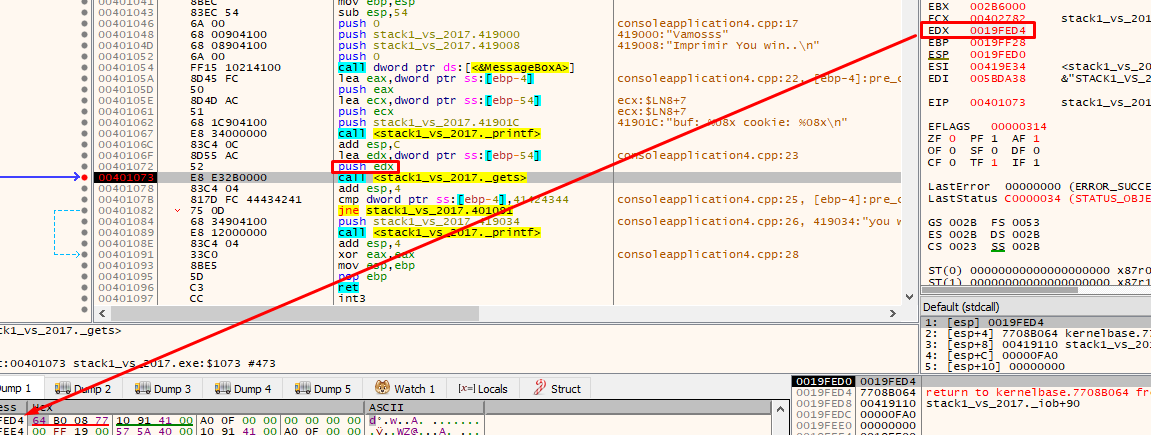
**input()**

Lo ejecutaré a este script y cuando salga el MessageBox lo atacheare con x64dbg y llegaré al gets(), lo ejecutaré y al volver del mismo, veré si copia todos los caracteres a buf qué es el buffer de destino que va a ser desbordado.



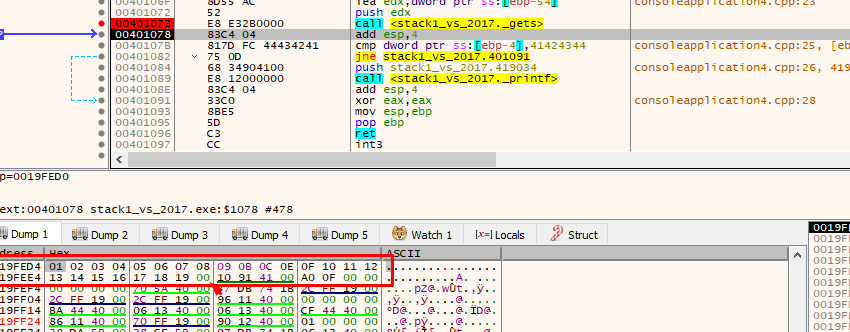


Le pongo un breakpoint en el gets() y acepto el MessageBox, parara allí.



Como veo que a EDX se encuentra la dirección de buf, marco el registro EDX y elijo FOLLOW IN DUMP para ver en el mismo la zona donde copiará los bytes que le envíe.

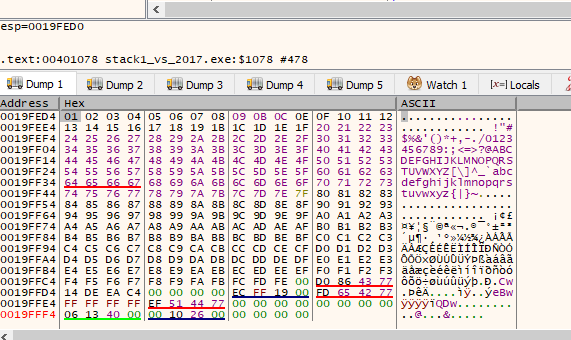
Ahora presiono f8 para que se ejecute el gets()



Veo que mi cadena de caracteres se cortó en el 0x1a, agreguemos el mismo a la lista de caracteres inválidos y volvamos a probar.

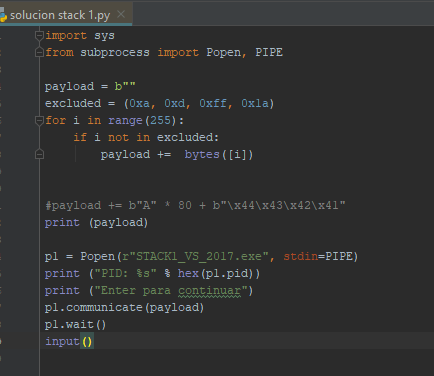
**excluded = (0x0, 0xa, 0xd, 0xff, 0x1a)**

Vuelvo a repetir el proceso, lo lanzo, atacheo el programa y paso el gets() con f8.

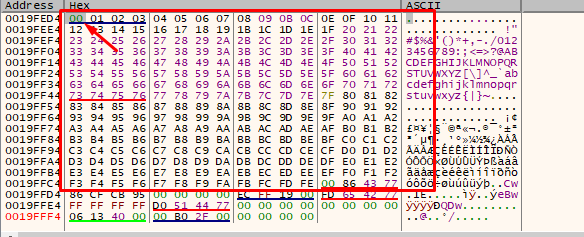


Ya vemos que ingresó desde 0x1 hasta 0xfe, de esta forma ya sabemos que evitando los 5 caracteres que pusimos en la lista de inválidos pasará nuestro payload.

Ahora quitaremos los caracteres inválidos iniciales uno a uno, empecemos con el 0x0.

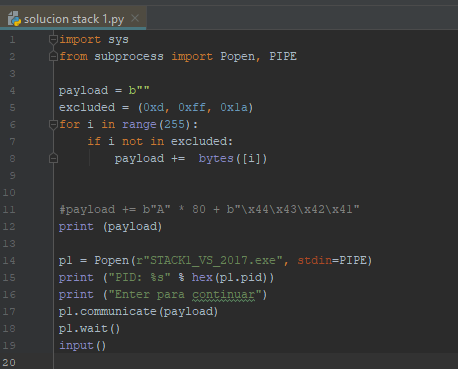


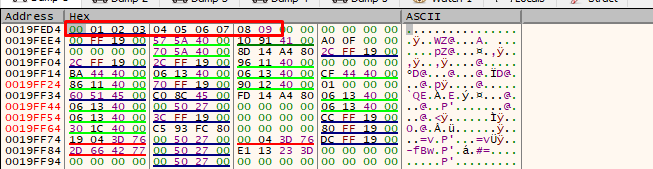
Probamos nuevamente.



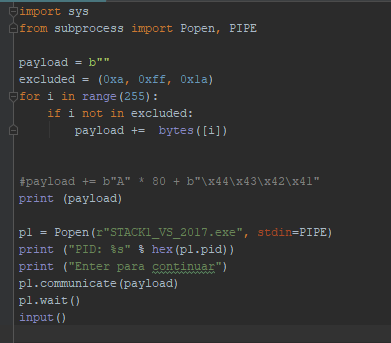
Vemos que el 0x0 no es un carácter inválido ya que pasó y no corto el payload, sigamos con los otros.

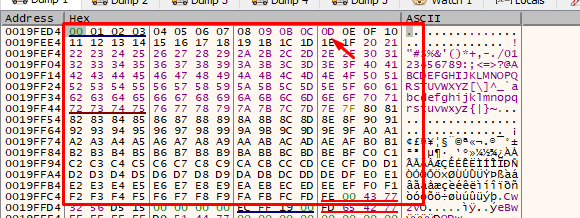
Quitamos el 0xa.



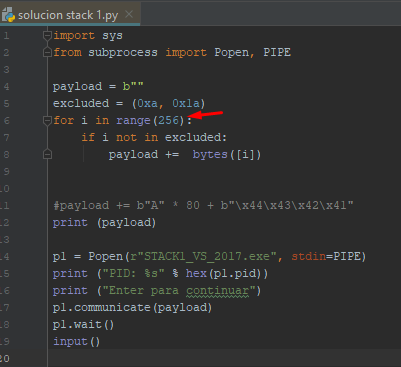


Vemos que el 0xa es un caracter invalido ya que corta el payload y no ingresa completo, lo vuelvo a agregar y quito el 0xd.

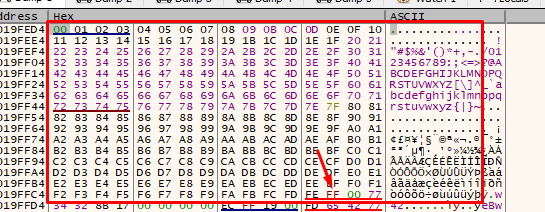




Vemos que el 0xd no es caracter invalido ya que no cortó el payload, solo queda el 0xff, pruebo.



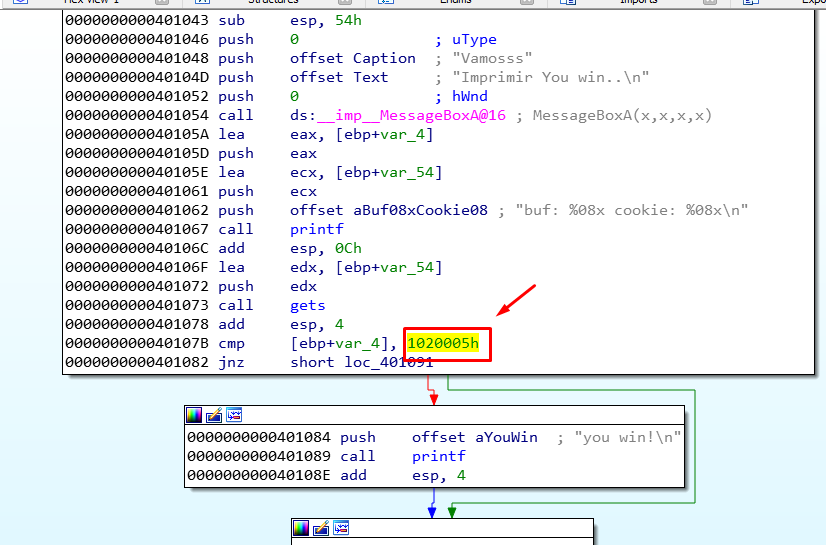
Veo que en el loop, para que ingrese el carácter 0xFF debe loopear hasta range( 256) ya que no incluye el ultimo, asi entra el 255.(0xFF)



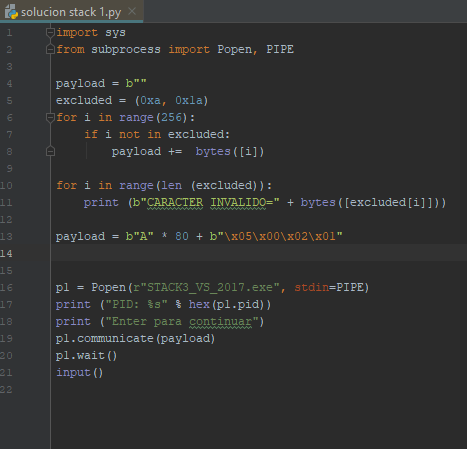
Vemos que ya entro toda la cadena y probamos todos los caracteres, los inválidos para los stacks son el **0x1a** y el **0xa** el resto pueden enviarse sin problemas.

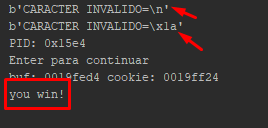
Por eso en el stack 3 que en la cookie tiene un cero, está hecho para que uno investigue si el cero es caracter inválido y como no lo es y puedo enviarlo en mi payload, el script será similar al del stack 1 y 2 solo cambiando el valor para que compare con la cookie.

SOLUCIÓN DEL STACK 3



Por lo tanto hay que pasarle “\x05\x00\x02\x01” para que compare con la cookie, ninguno es caracter invalido asi que no hay problema.

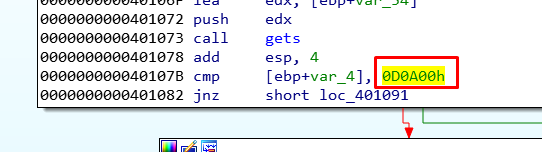




Allí imprimimos los caracteres inválidos y vemos que con el mismo análisis que hicimos para el stack 1 y 2, se puede solucionar el 3 sin problemas.

Es conveniente siempre que tengamos que realizar un exploit, realizar el trabajo de detectar los caracteres inválidos para estar seguros de que nuestro payload no será afectado por los mismos.

El caso del stack 4 es diferente, en la cookie hay un caracter invalido.



Vemos que deberíamos pasarle para que pise la cookie

**payload = b"A" \* 80 + b"\x00\x0a\x0d\x00"**

Vemos que en nuestro payload hay un caracter invalido, por lo cual no podemos pasar el valor que necesitamos para pisar la cookie y que nos lleve a YOU WIN.

En la parte siguiente mostraremos las soluciones.

El que quiere intentarlo sepa que hay que continuar pisando más abajo de la cookie y intentar pisar el return address, para retornar a la dirección donde el programa llama al printf con el mensaje YOU WIN a la fuerza, o buscar alguna forma de lograr imprimir lo que querramos.

Por supuesto es muy posible que el programa se rompa al hacer eso, así que hay dos soluciones válidas:

1) Que imprima YOU WIN y se rompa que es la más sencilla.

2) Que imprima YOU WIN y que podamos evitar que se rompa lo cual implica más dificultad obviamente.

A ver quien se anima a hacerlo de alguna de las dos formas (o ambas je).

Hasta la parte 5

Ricardo Narvaja

15/11/2019