**Escuela Colombiana De Ingeniería**

**Julio Garavito**

**Seguridad y privacidad TI**

**Daniel Esteban Vela Lopez**

**Andrés Felipe Montes**

**Laura Valentina Rodríguez Ortegón**

**Laboratorio No.13**

**2024-2**

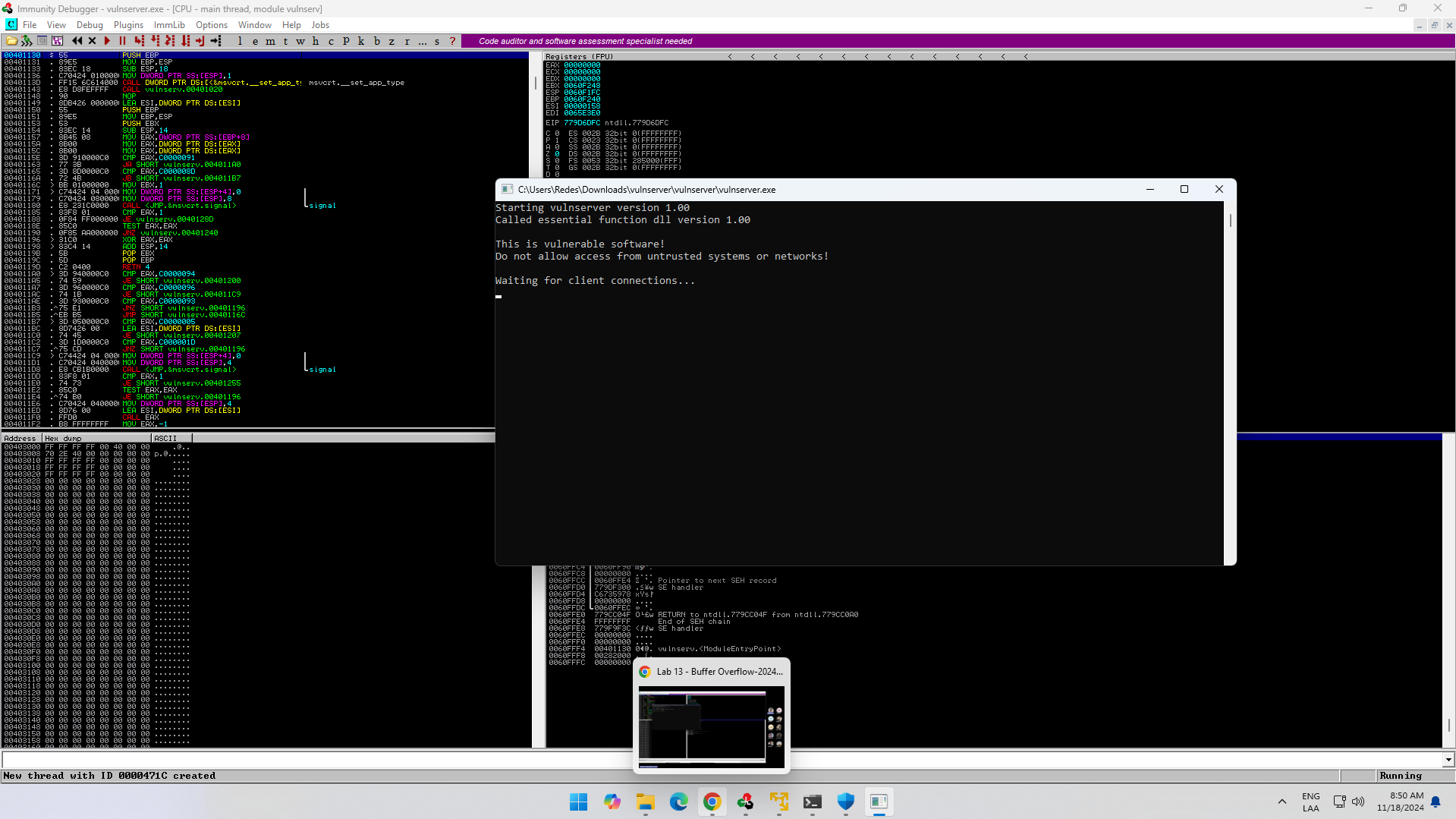
Para la realización de este lab primero descargamos la herramienta immunity debugger y el ejecutable de la aplicación que vamos a testear.

### **Preparativos:**

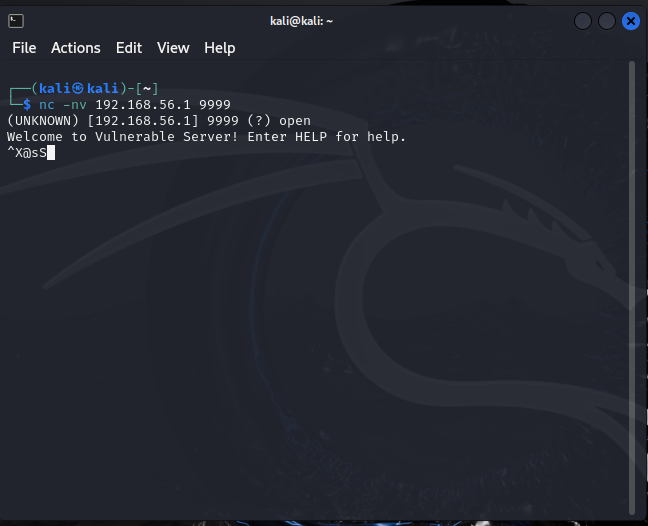
* **Herramientas utilizadas:**
  + **Immunity Debugger**: Para monitorear y analizar la aplicación objetivo en busca de fallos.
  + **Kali Linux**: Como la máquina atacante.
  + **Mona.py**: Un script de ayuda para análisis de vulnerabilidades en Immunity Debugger.
* Configuración:
  + Verificar que ambas máquinas (atacante y víctima) estén en la misma red.
  + Iniciar la aplicación objetivo dentro de Immunity Debugger para observar su comportamiento y analizar su código ensamblador.

Después de tener ambos instalados corremos la aplicación dentro del immunity debugger y podemos observar que este nos ofrece ciertas herramientas que nos serán útiles en nuestro análisis como:

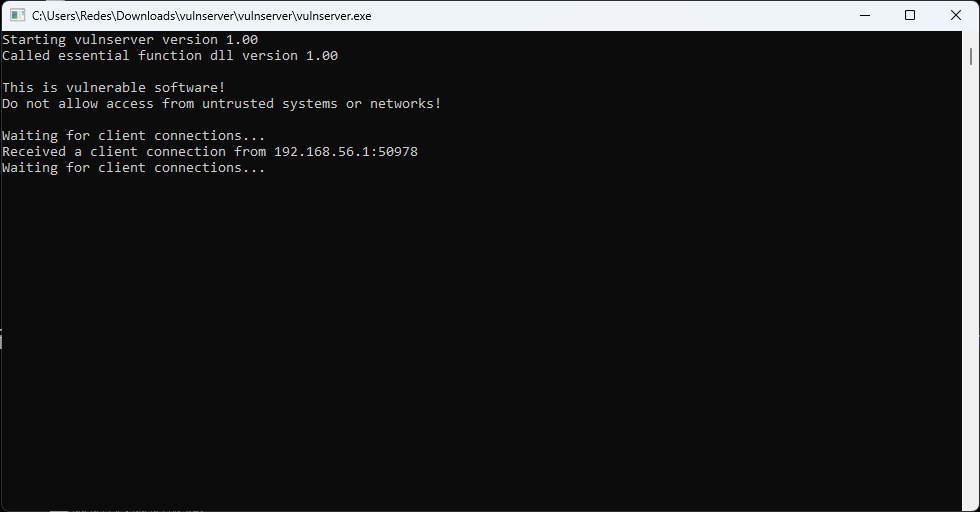
* Código ensamblador de la aplicación
* Análisis de vulnerabilidades
* Creacion de exploits



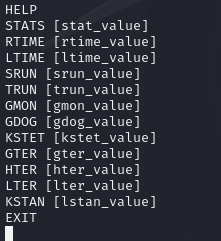
Después de iniciada la aplicación debemos verificar que la máquina atacante “kali” y la app están en la misma red por lo que miramos ambas ip para verificar eso. Acontinuacion nos conectamos a la app mediante el siguiente comando por medio de kali.



Como podemos ver se realizo una conexión exitosa mediante el puerto 9000

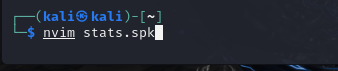


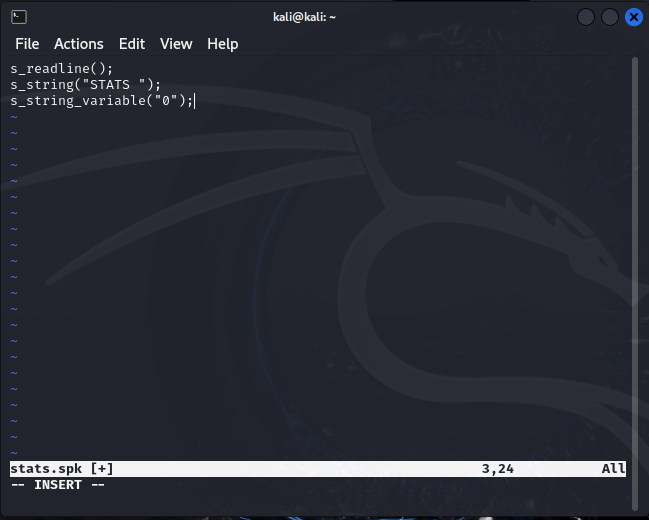
Con el comando HELP podemos ver los comandos básicos que ofrece la app, esto es de mucha ayuda ya que debemos encontrar una vulnerabilidad en alguno de estos.

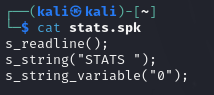


Spiking

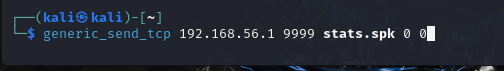
Comenzamos creando un script con el fin de buscar una vulnerabilidad en el comando STATS el cual consiste en meter cadenas de “a” consecutivamente hasta que llegue a un límite y el programa se cuelgue.



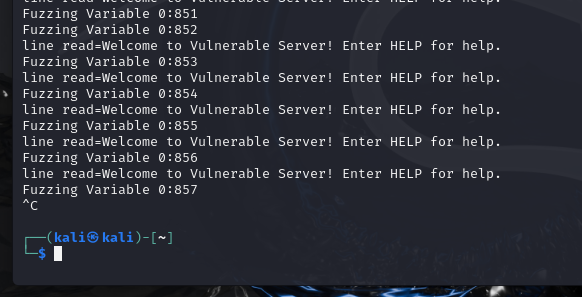




stats

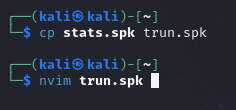


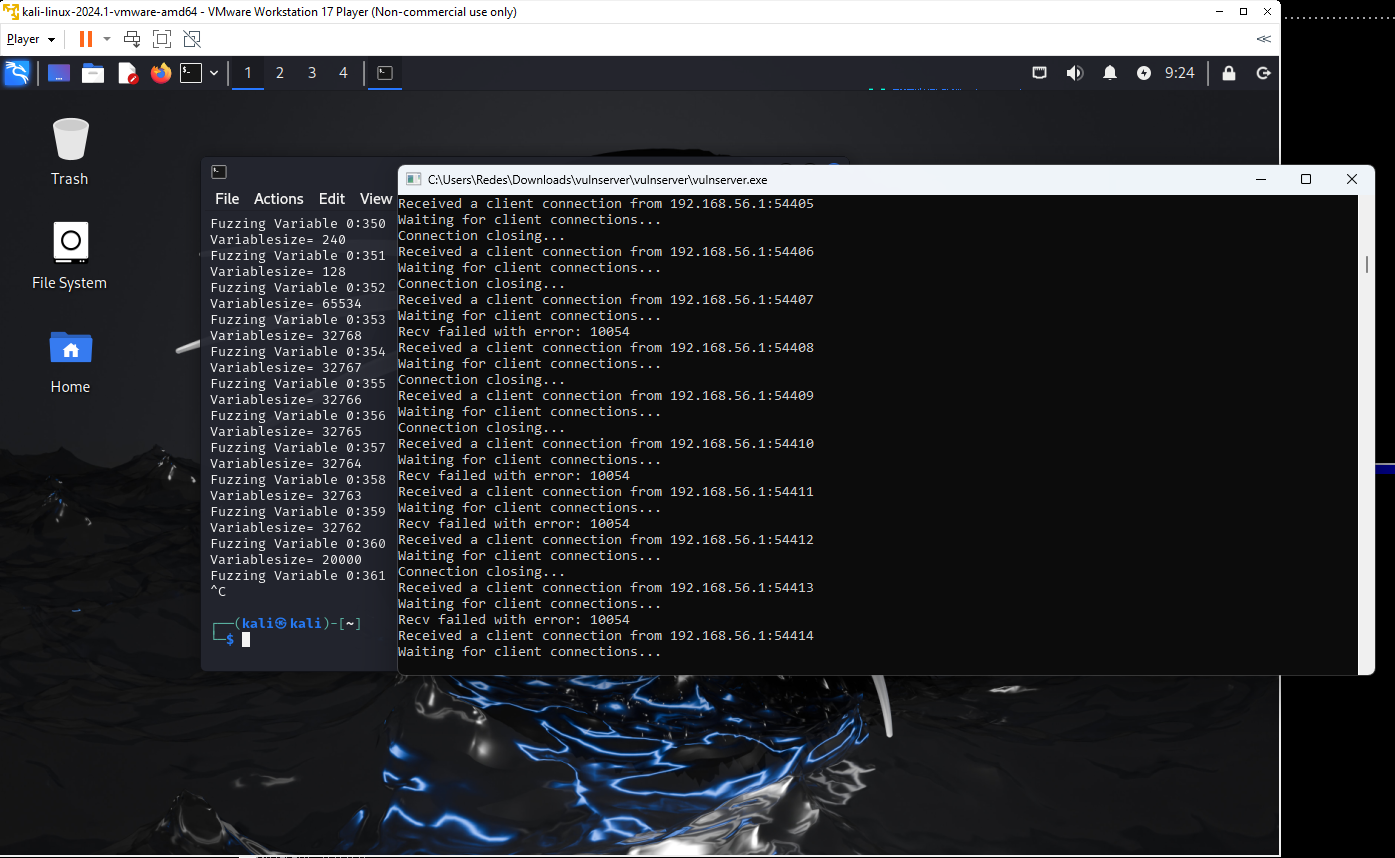
Recibe 800 ceros y aun asi no se rompe por lo tanto este comando no es vulnerable, entonces seguiremos probando con los siguientes hasta encontrar un overflow.



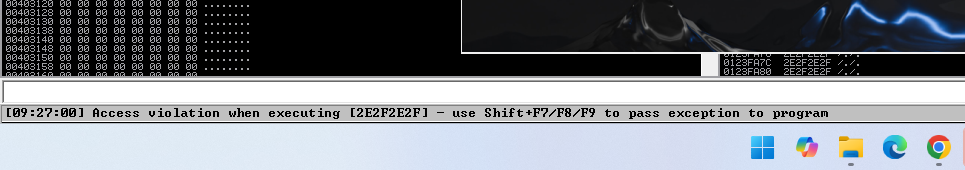
Trun

Para el siguiente comando seguiremos la misma lógica del anterior y insertamos cadenas consecutivas de “a” cada vez más largas para intentar colgar al programa.

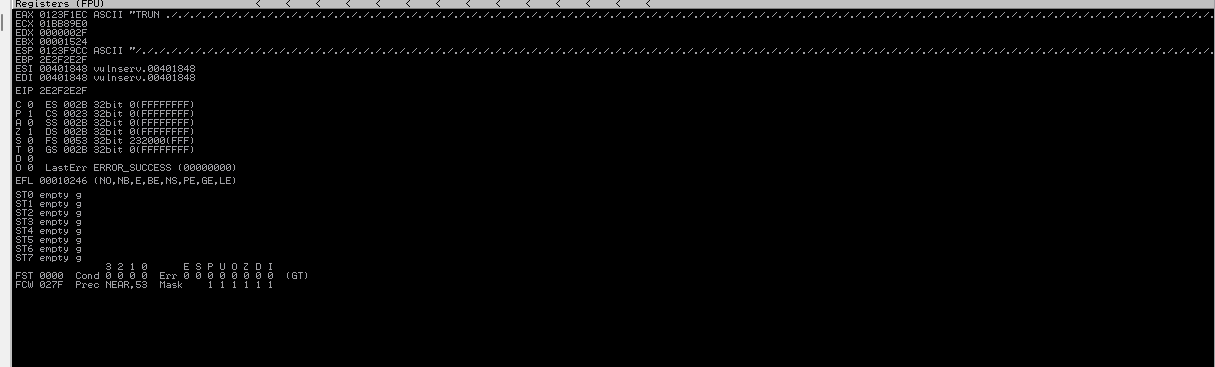




Se rompe el programa después de insertar cierta cantidad de “a” en este caso vemos que se violó la seguridad de la aplicación.



Se logra alcanzar un overflow



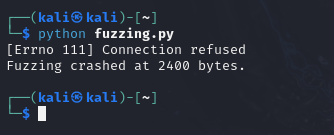
Fuzzing

Técnica de pruebas de software utilizada para detectar errores, vulnerabilidades de seguridad y fallos en aplicaciones. Consiste en proporcionar a un programa entradas aleatorias, inesperadas o inválidas para observar cómo responde. Si el programa no maneja adecuadamente estas entradas, puede generar errores, bloquearse o volverse vulnerable a ataques.

Ya que sabemos que el comando “trun” es vulnerable queremos saber en qué momento exacto se rompe este, por lo tanto el siguiente programa de python insertará cadenas de “a” \* 100

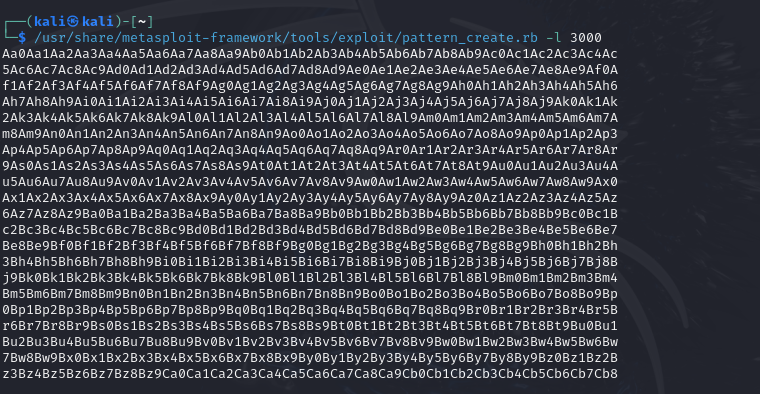


2400 bytes es el número aproximado en el que se rompe el programa



Ahora que sabemos el número aproximado de bytes que se necesitan para causar un overflow en la app, debemos saber el byte exacto en el que se rompe el programa.

Este comando genera 3000 caracteres de manera aleatoria

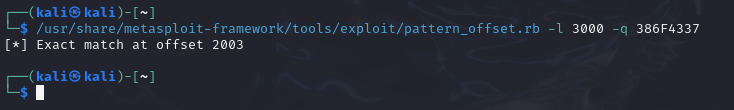




Después de ejecutar el archivo python con un buffer de 3000 conocidos podemos ver la linea exacta donde el programa explota

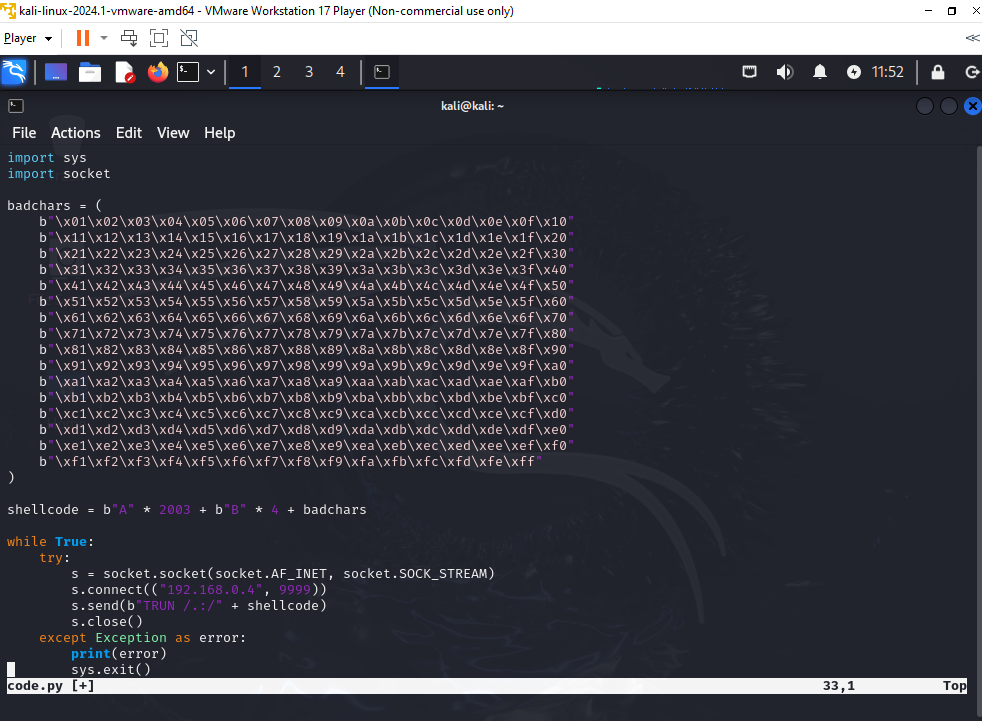


Con el siguiente comando y el código de la línea podemos identificar el byte exacto donde se cuelga la app y esto pasa justo en el byte 2003, esto significa que después de insertar 2003 a el siguiente caracter alcanza un overflow.

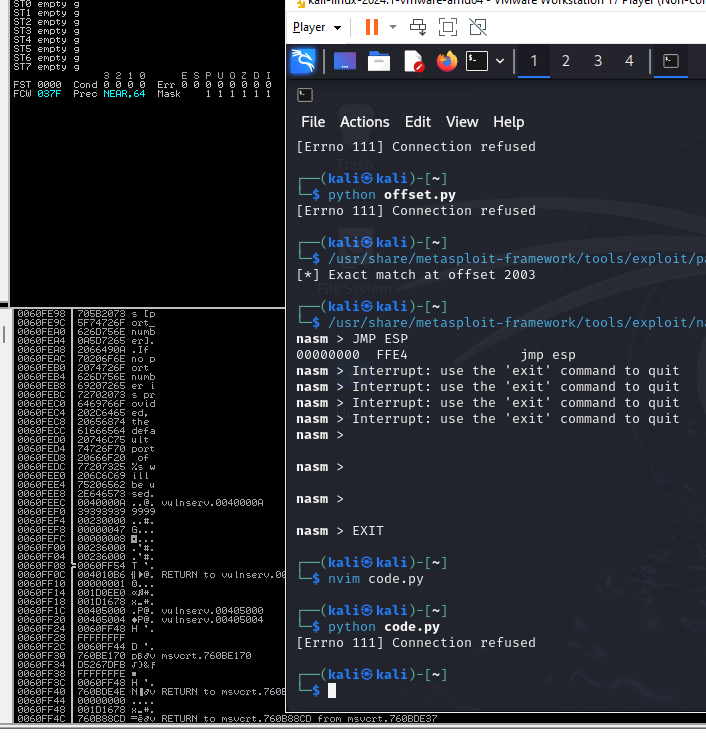


Continuando con la exploración sobre esta app ahora vamos a intentar ver que cadenas de texto admite el programa.

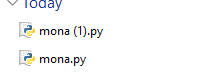
Con este código podemos ver si el programa no acepta caracteres especiales o los toma como nulos.



Después de ejecutar el código podemos ver que aparentemente acepta todos los caracteres hasta los especiales.



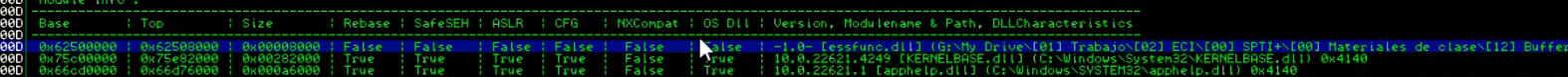
Ahora vamos a descargar mona.py para poder buscar entre los modulos



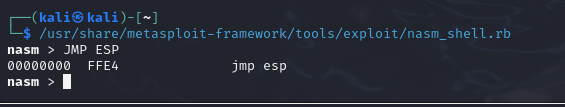
Y lo llevamos hasta la carpeta /PyComands



Después de usar el comando mona encontramos un modulo sin seguridad.



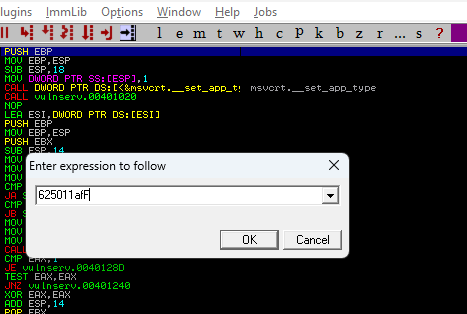
Con este comando podemos traducir ensamblador y tomar el FFE4 que es lo que nos importa para buscarlo en mona.py.





Tomamos la primera que encontramos que es 625011AF pero como mona entrega los valores al revés crearemos el nuevo archivo exploit con el valor cambiado.

Pero antes de eso vamos a definir un break point.



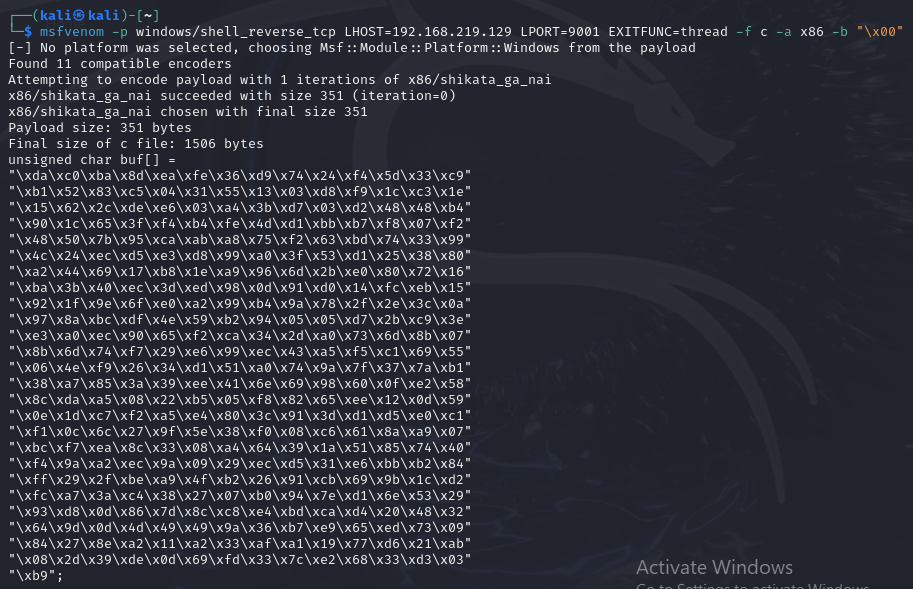


Con f2 podemos generar el breakpoint y debemos seleccionar la línea FFE4 y cuando volvamos a ejecutar el programa debería parar cuando llegue a ese espacio de memoria.

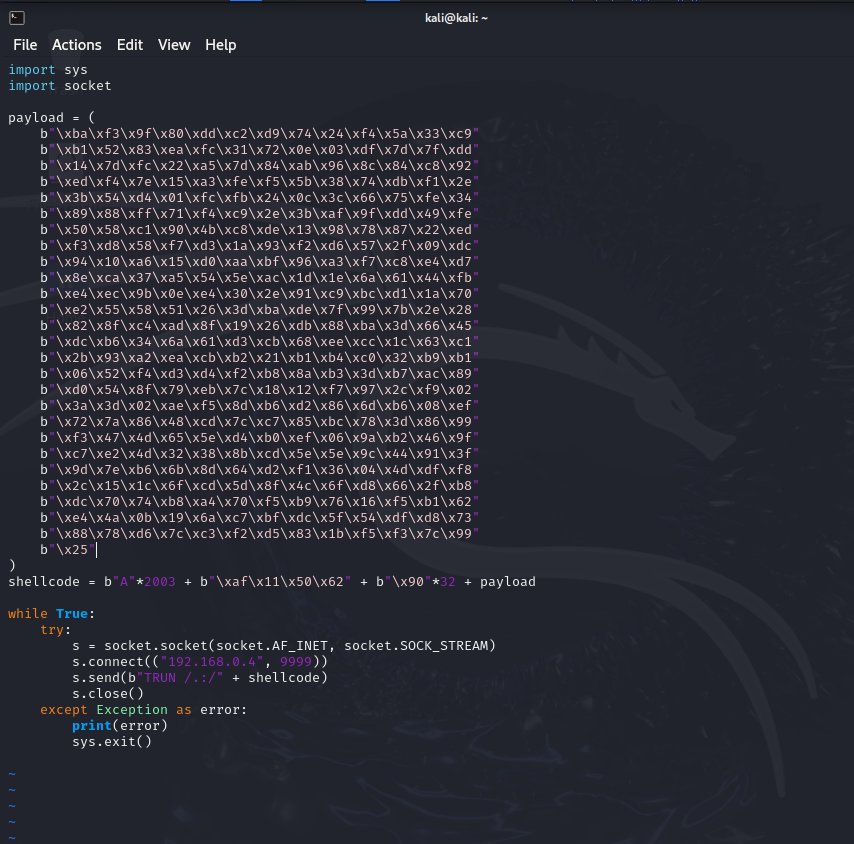


Ya que conocemos la vulnerabilidad de este sistema, estamos listos para generar un exploit que nos permita entrar a la máquina atacada sin permiso alguno.

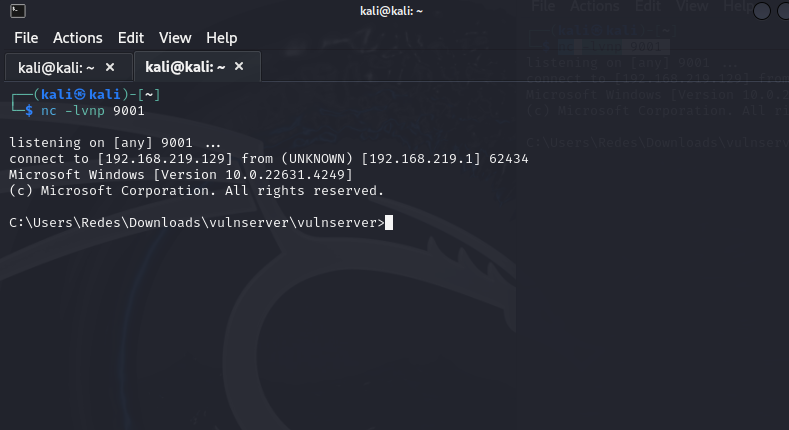
Ahora con este comando vamos a generar el código malicioso que luego vamos a utilizar en el exploit.

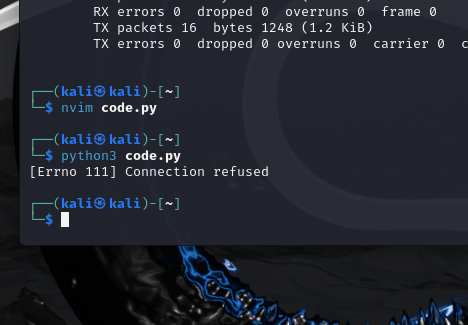


Y modificamos nuestro exploit para agregar el código malicioso, en resumen como sabemos que al introducir 2003 “a” el programa se cuelga, justo después de colgar introducimos el código malicioso.



Para probar que si nos podemos conectar usaremos el puerto 9001 y lo pondremos a escuchar. Mientras en otro cdm ejecutamos el exploit y como podemos observar logramos conectarnos de manera exitosa a la máquina atacada que en este caso es un computador de la escuela con ip 192.168.219.129.





Una vez conectados a la máquina podemos ejecutar programas y demás herramientas que nos pueden servir para robar información, por último para probar su función ejecutamos calc.exe y vemos como en efecto podemos iniciar la calculadora desde “kali” la máquina atacante.

