**Web 1**

1. Переходим на данный в таске сайт

2. Переходим по гипер ссылке, вшитой в "20" и находим подсказку, что флаг находится по адресу etc/secret (см. рис 1)

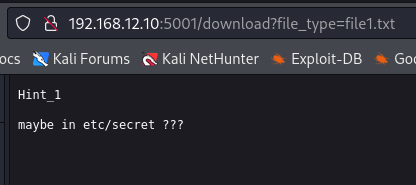
3. Запускаем программу OWAST\_ZAP, вводим полученный url(http://192.168.12.10:5001/download?file\_type=file1.txt) и начинаем атаку

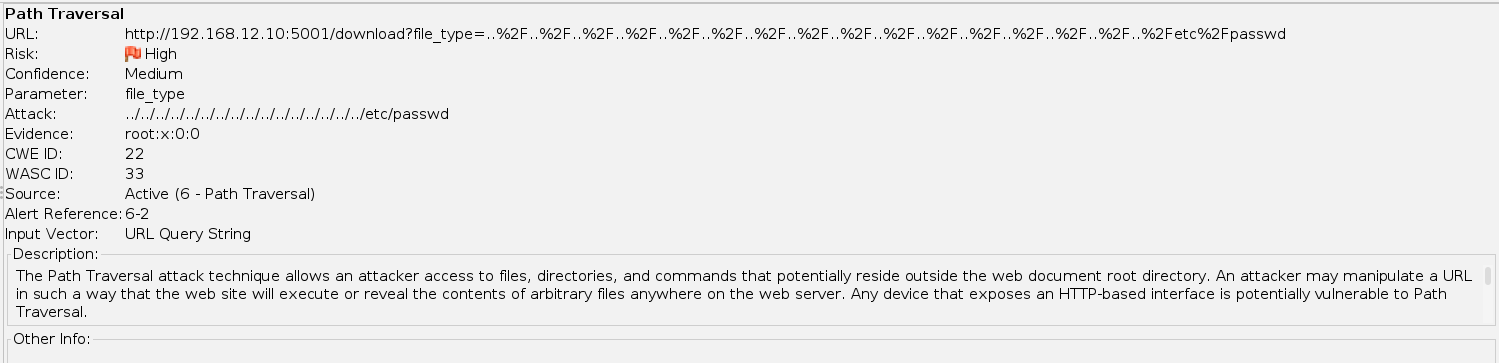
4. Находим уязвимость типа Path Traversal, получаем url: http://192.168.12.10:5001/download?file\_type=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd

и данные что атака производилась на ../../../../../../../../../../../../../../../../etc/passwd (см. Рис 2)

5. Меняем passwd на secret: http://192.168.12.10:5001/download?file\_type=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fsecret,

получаем файл, содержащий флаг (см. рис 3)

Рис 1.

Рис 2.

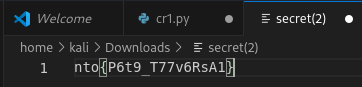
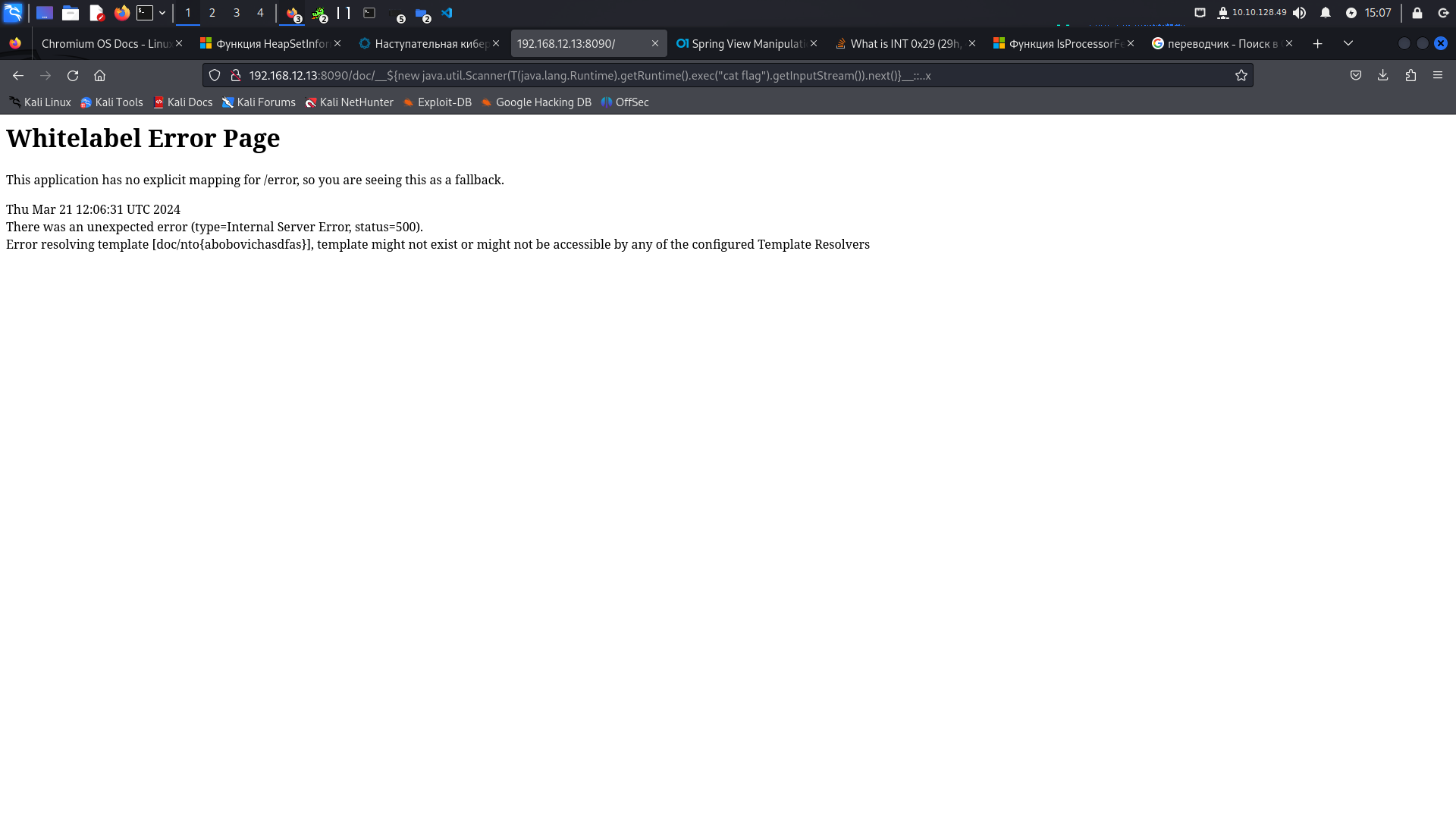


Рис 3.

**web 2**

Декомпилировав legacy.jar, мы обнаружили класс HelloContorller. Мы предположили, что здесь есть уязвимость SSTI. Поискав в интернете, мы обнаружили сайт(https://www.veracode.com/blog/secure-development/spring-view-manipulation-vulnerability), где рассказано, как производить уязвимость SSTI для spring framework. Введя следующий url: “/doc/\_\_${new java.util.Scanner(T(java.lang.Runtime).getRuntime().exec("cat flag").getInputStream()).next()}\_\_::..x”, нам вывелся флаг(рис 1.)

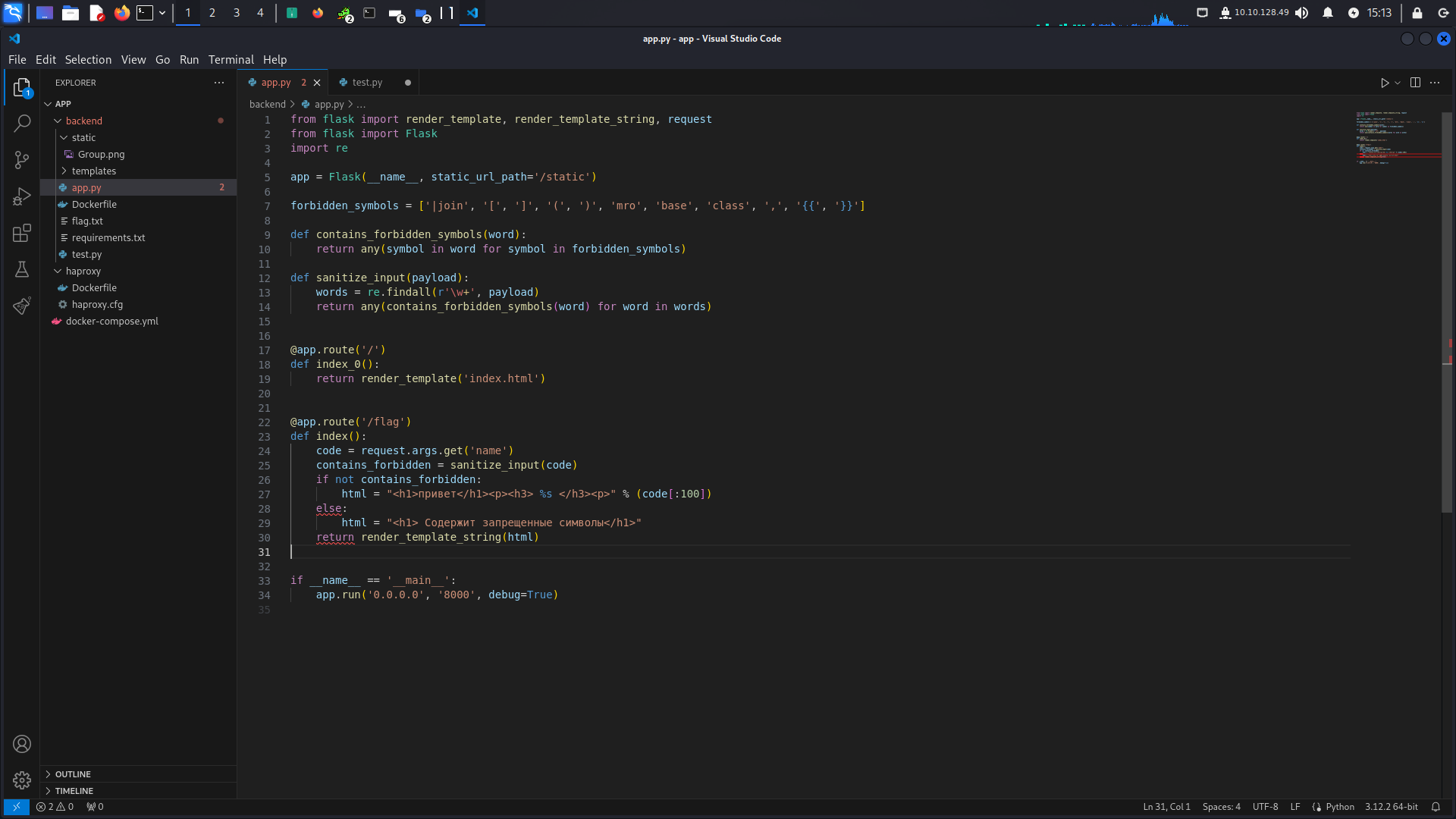
Рис 1. Атака SSTI на spring framework.

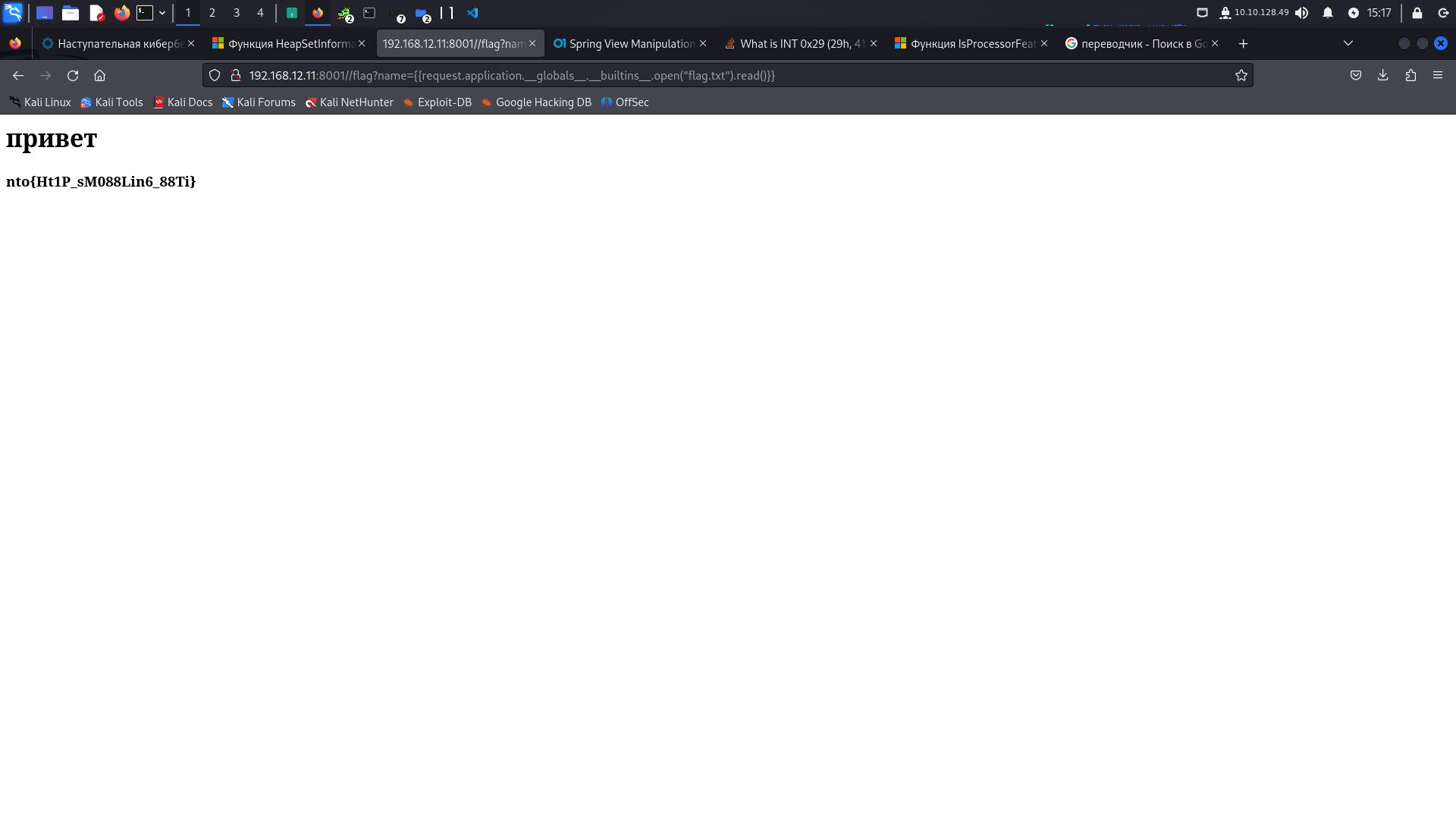
Флаг: nto{abobovichasdfas}

**web 3**

При заходе по url //flag, нам выводится значение, которые мы введём в url. Это SSTI атака на сайт flask. Но в программе фильтруются значения(рис 1). Однако попробовав ввести следующий url, содержащий запрещенные символы://flag?name={{request.application.\_\_globals\_\_.\_\_builtins\_\_.open("flag.txt").read()}}, проверка пропустила его(рис 2).

Через глобальные объекты, которые доступны, обращаемся к функции open и читаем флаг

Рис 1. Исходный код сайта.

Рис 2. Атака SSTI.

Флаг: nto{Ht1P\_sM088Lin6\_88Ti}

**pwn 1**

Если открыть файл в дебаггере ghidra, то можно видеть, что программа считывает 1024 символа, записыает в буффер и выводит его на экран(рис 1). Это типичная атака на printf, где мы можем переписать байты в любом месте, где мы только захотим.

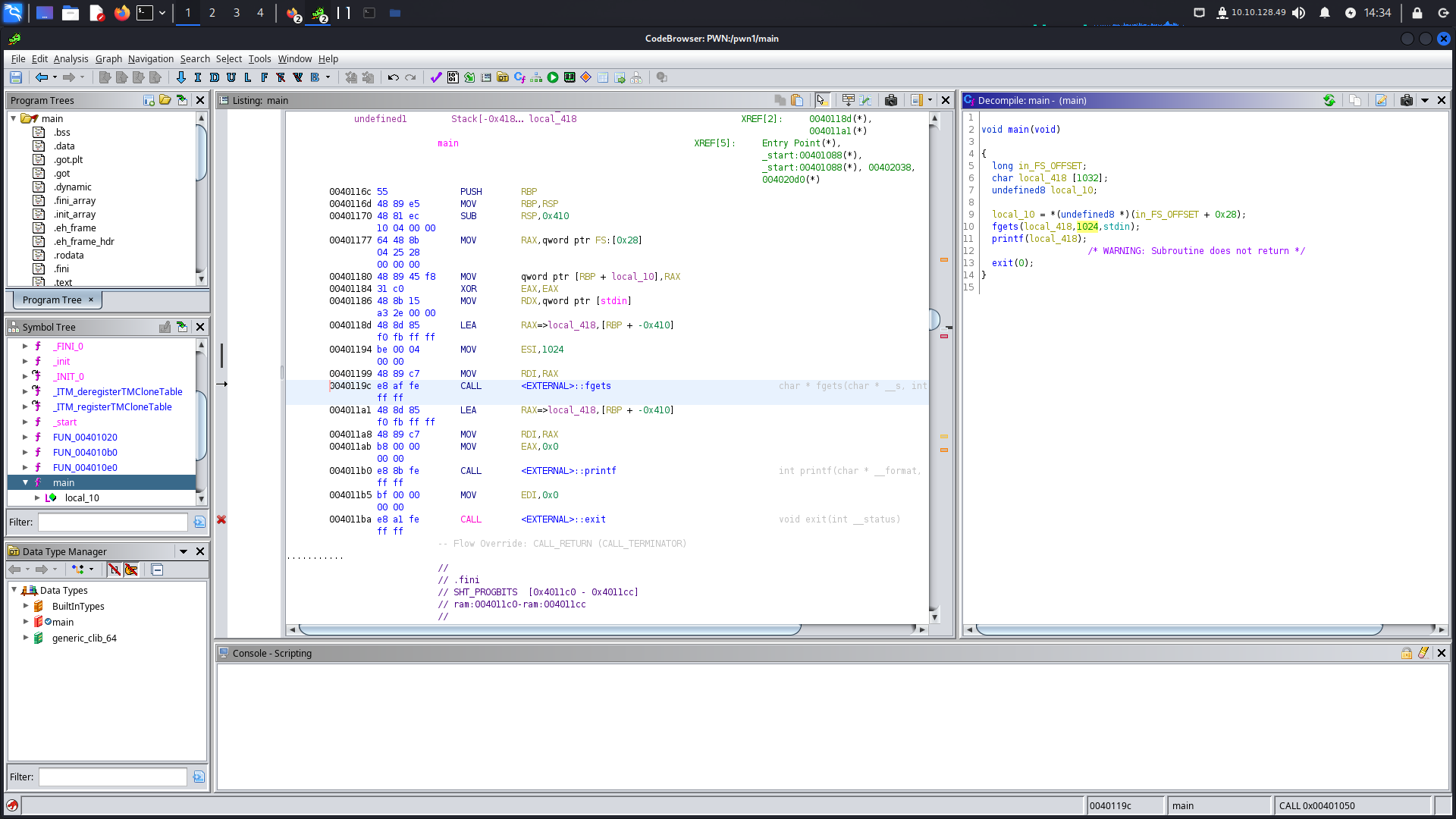
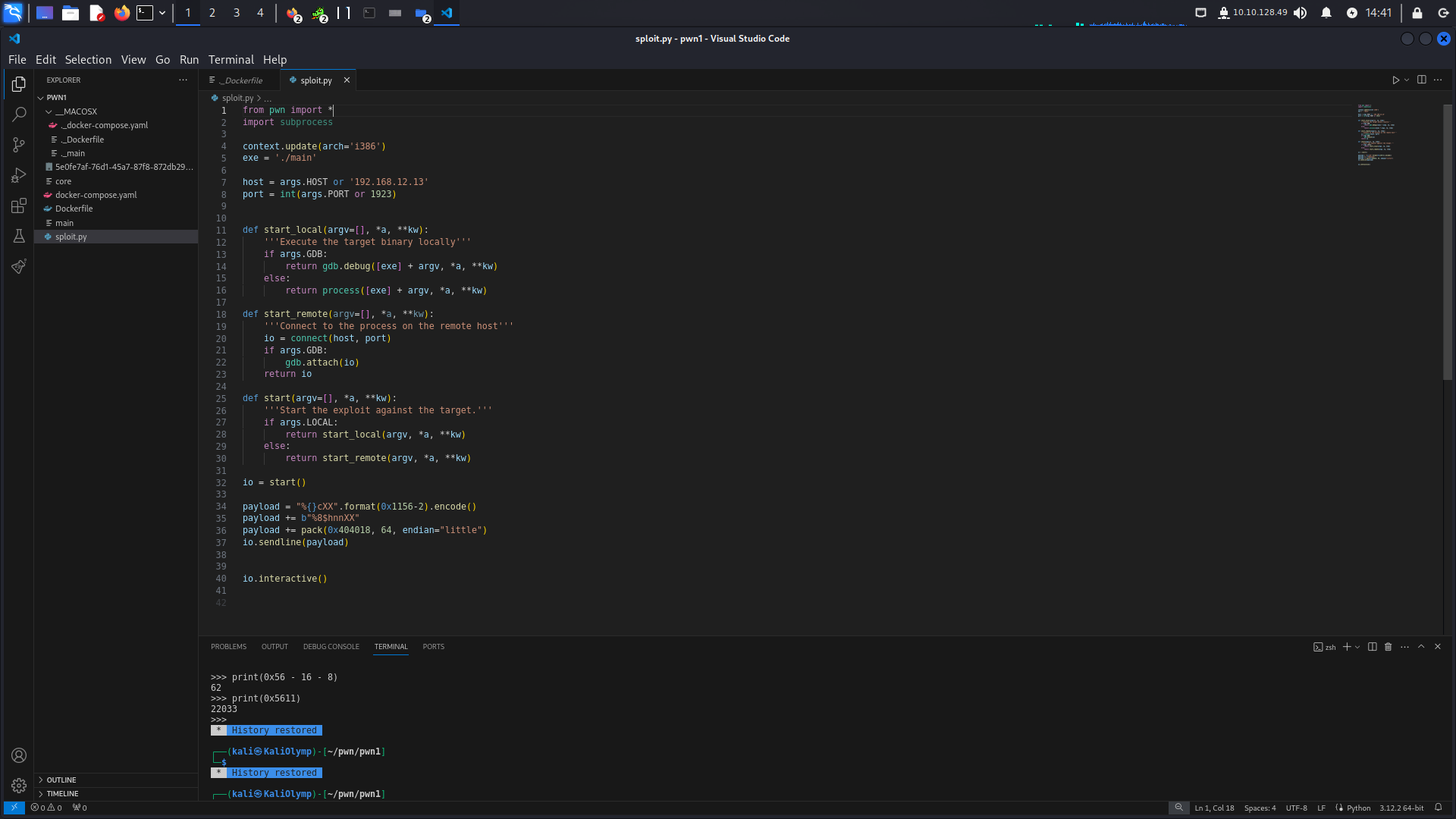


Рис 1. Функция main в дизассемблированном виде.

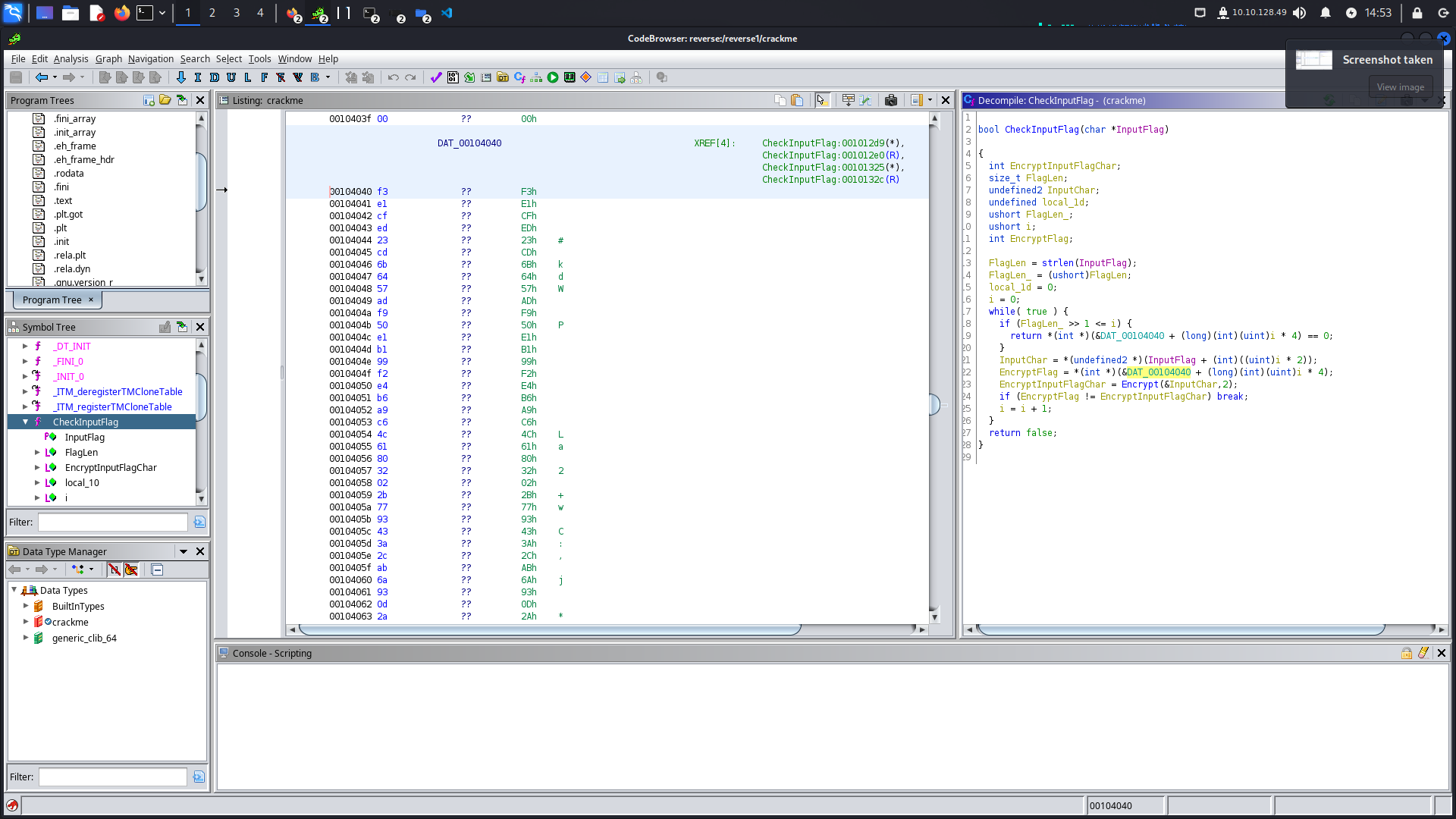
Атака заключается в следующем. Существует спецификатор форматной строки %n, который по указанному адресу запишет, сколько байтов до него вывелось на экран. Заметим, что в программе есть функция win, которая вызывает system(“/bin/sh”), а также заметим, что функция main завершается вызовом библиотечной функции exit(). Переписав адрес exit в сегменте got.plt на адрес функции win, мы получим доступ к компьютеру.

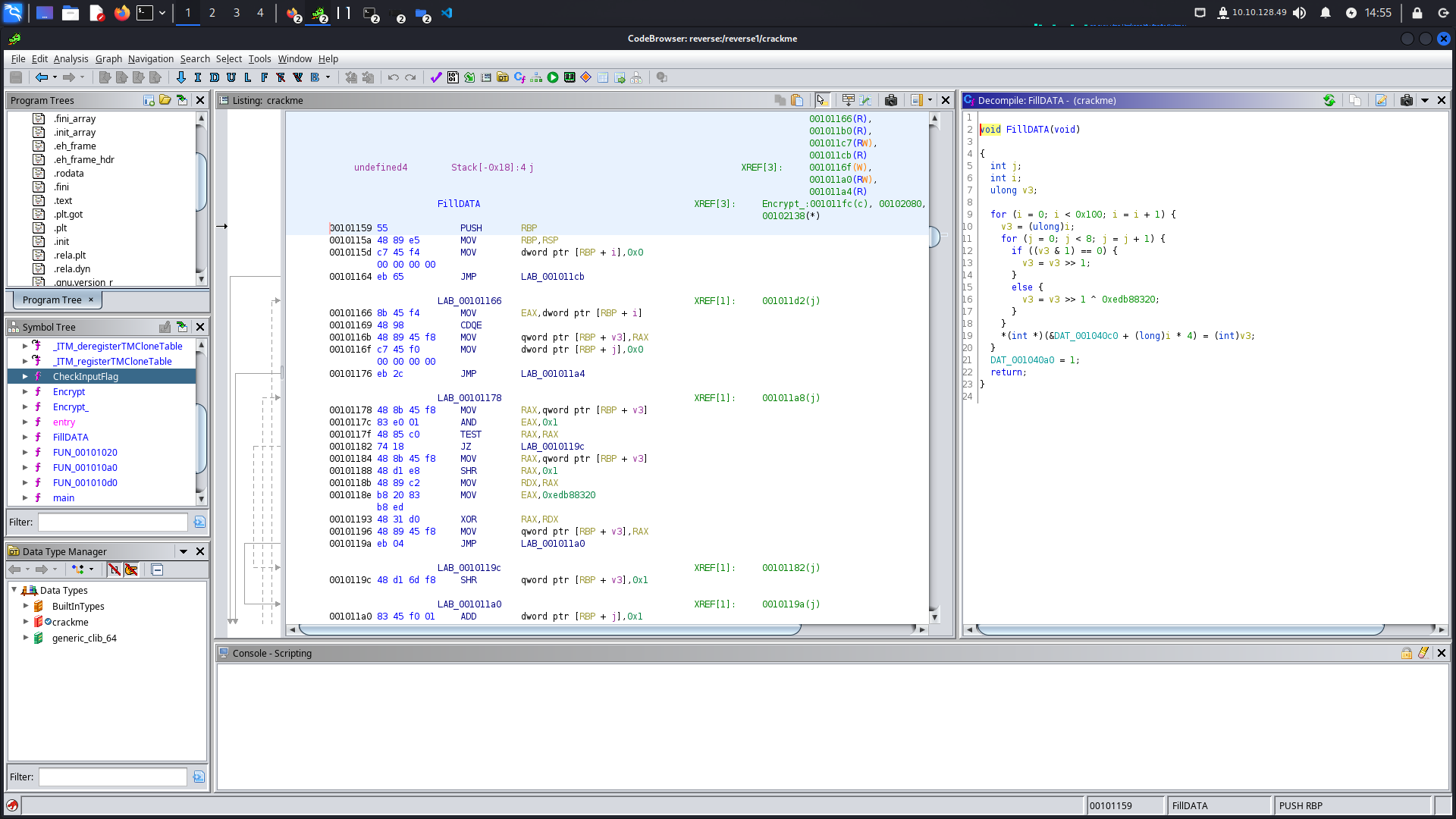
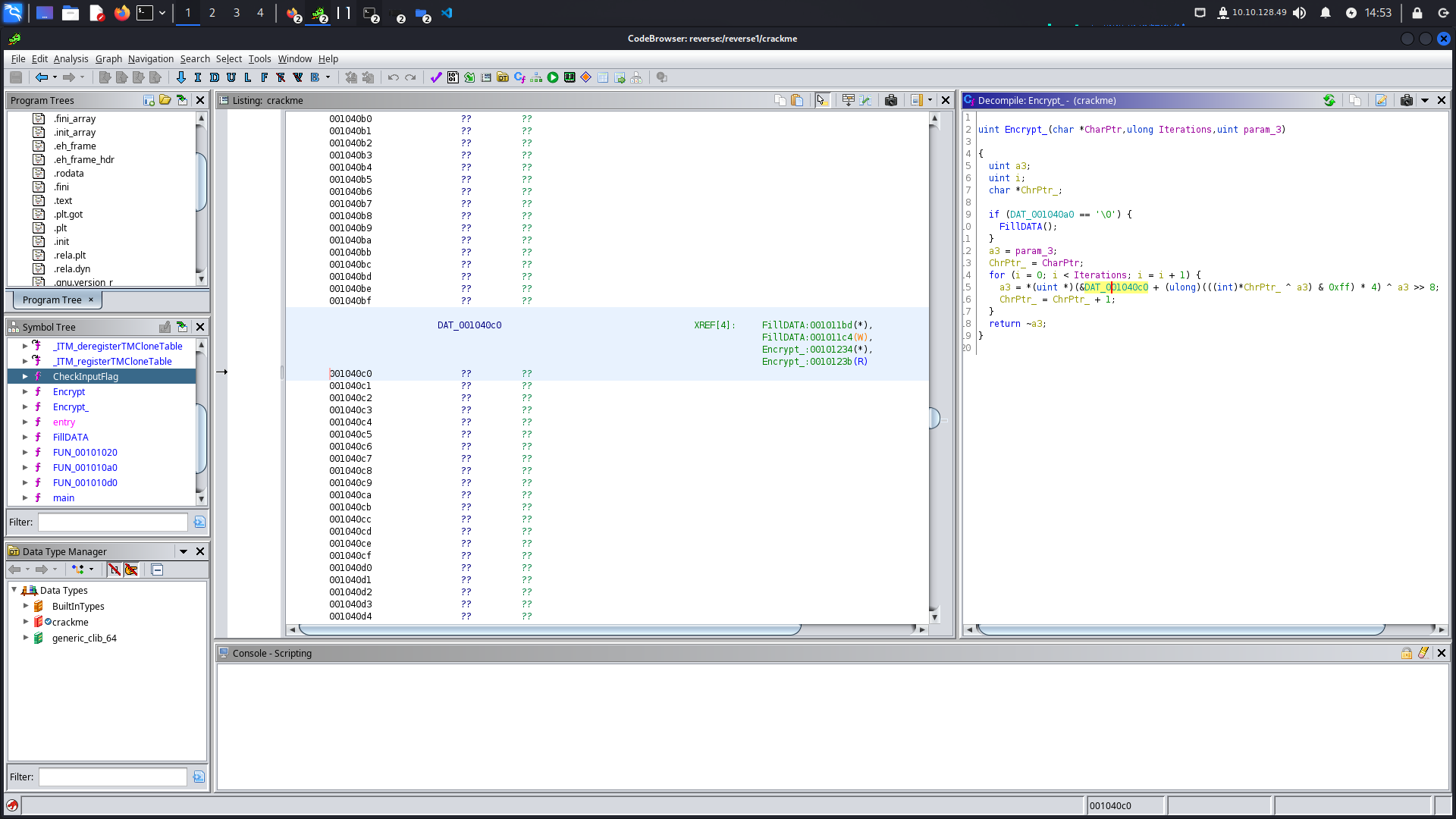
Рис 2. Код для получения доступа к shell

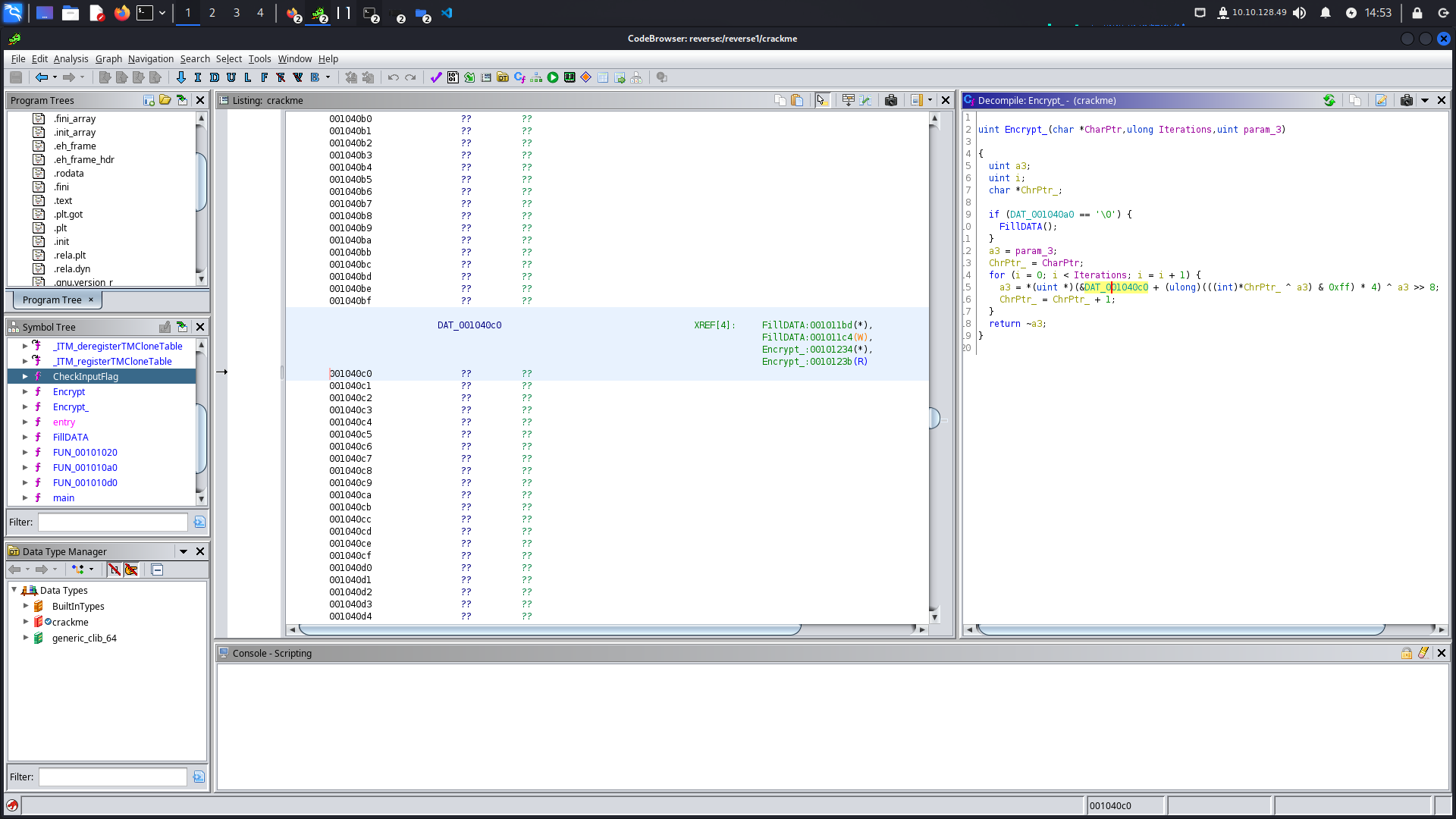
Полученный флаг: nto{easy\_formt\_string}

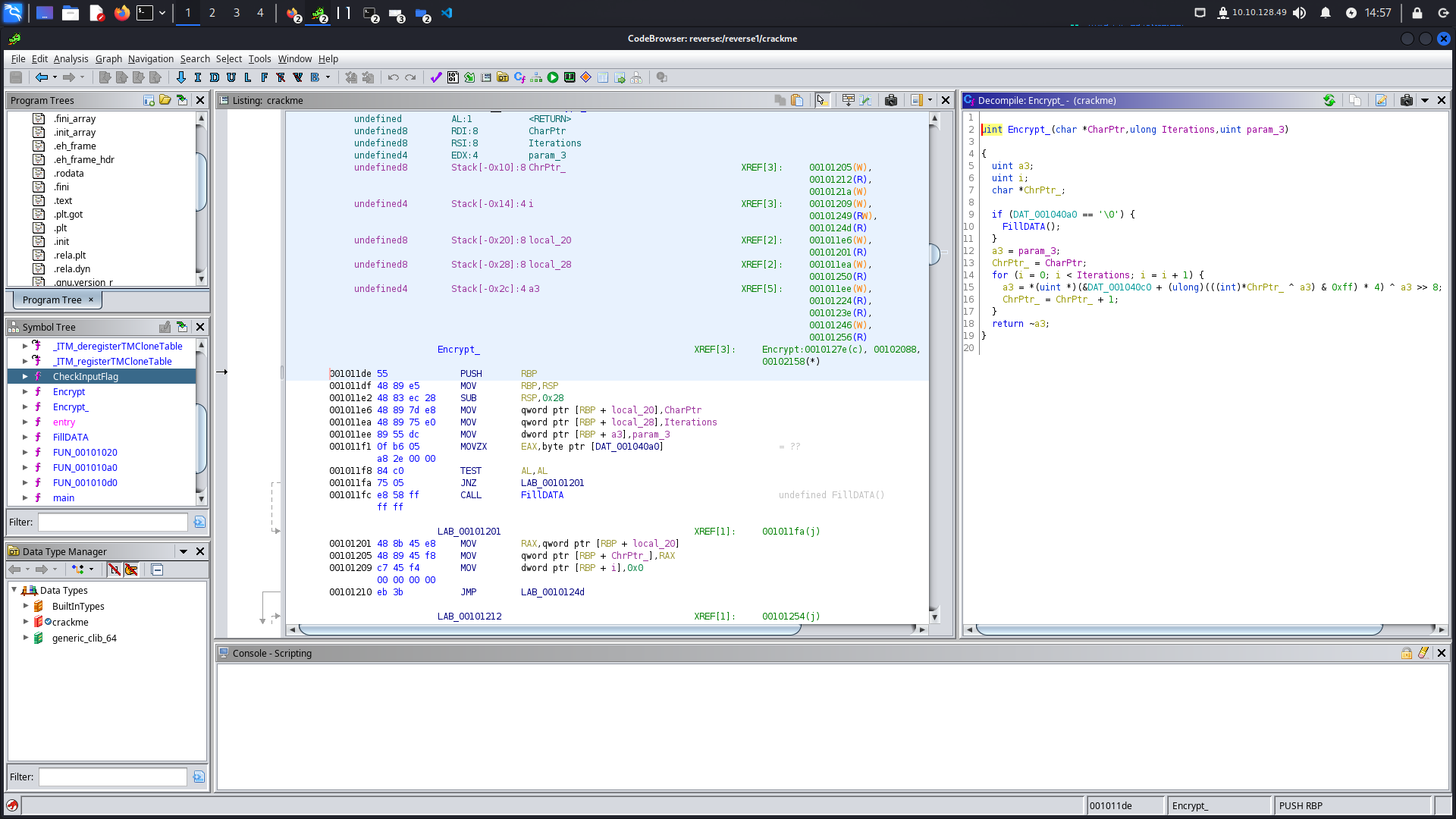
**reverse1**

Проанализировав файл, можно заметить участок с байтами(рис 1.), алгоритм заполнения массива байт(рис 2.1, рис 2.2) и алгоритм шифрования символов флага(рис 3.).

Рис 1. Участок с байтами зашифрованного флага.

Рис 2.1. Алгоритм генерации байтов.

Рис 2.2 Участок с массивом генерируемых байтов.

Рис 3. Алгоритм шифрования символов флага.

Алгоритм проверки флага:

1. Берём два символа флага;

2. Объявляем пременную a1 = 0xFFFFFFFF

2. Делаем операцию xor с байтом флага и с a1;

3. Берём первый байт получившегося числа;

4. Обращаемся к массиву байтов(рис 2.) по индексу получившегося числа и записыаем его в переменную a2;

5. Смешаем биты a1 вправо на 8;

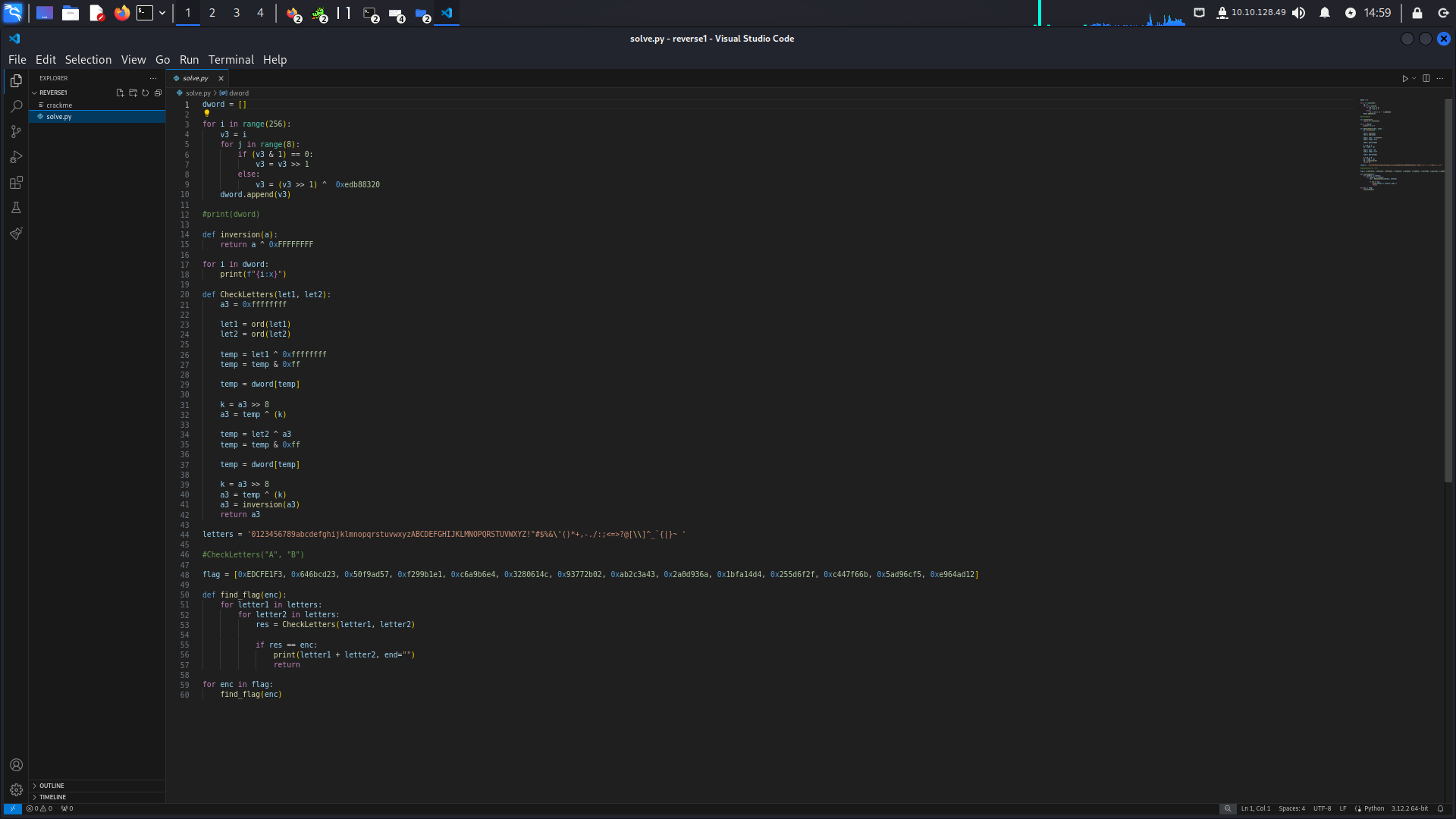
6. Производим операцию a2 xor a3.

7. Записываем результат в a3;

8. Повторяем те же действия, но уже со вторым байтом флага;

Если получившиеся число равно байту из участка с байтами(рис 1.), то мы берём следущие два байта флага и повторяем те же действия. В противном случае программа завершается.

Видно, что это задание можно решить с помощью подбора букв(рис 4.).

Рис 4. Алгоритм дешифровки флага.

Флаг: nto{4n0TH3R\_bRu73F0RC3\_7ASk}

**forensics**

1 и 2) Вредоносное ПО попало на компьютер вследствие неосторожности пользователя Windows.

На компьютере лежал backup файл программы outlook, открыть который не составляет сложности с помощью Outlook или же любой доступной программы читалки в интернете. Пользователь получил письмо с адреса [j.nathan@microsoft.com](mailto:j.nathan@microsoft.com) 3 марта 2024 в 16:40:56. К письму прилагался архив под названием classfied.rar. Архив содержит папку TOP\_SECRET и файл TOP\_SECRET.pdf.

В папке содержится файл TOP\_SECRET.pdf .cmd - скрипт, который скачивает с сайта [http://95.169.192.220:8080/prikol.exe](http://95.169.192.220:8080/prikol.exe), который впоследствие запускался и скачивал файл Rjomba.exe.

“prikol.exe” является дропером

3) В скачанном с почты архиве содержался файл TOP\_SECRET.pdf.cmd который скачивал с сайта prikol.exe, который в свою очередь скачивает и запускал файл Rjomba.exe. Уязвимость заключается в двух факторах: powershell позволяет испольнять код из любой части ОС и у powershell нет встроенной политики безопасности, что позволяет скачивать файлы из незащищённых источников.

4) Программа вызывает функцию IsDebuggerPresent из библиотеки Kernel32.dll, а конкретно из заголовка debugapi.h, которая определяет, отлаживается ли вызывающий процесс отладчиком пользовательского режима. Также присутсвует вызов функции CheckRemoteDebuggerPresent из той же самой библиотеки. По документации, эта функция определяет, находится ли процесс под отладкой