Финальный доклад НТО 2023

Команда: Evgen3301 (№26)

1. Часть CTF

1.1. Crypto-1

В данном задании флаг хэшируется с помощью DihedralGroup. Обратить такой алгоритм достаточно сложно, однако можно использовать тот факт, что каждый байт хэшируется отдельно, независимо от других, что позволяет подобрать каждый байт по отдельности. Для этого необходимо воссоздать сам алгоритм хэширования и просто перебирать все 256 возможных значений каждого байта до тех пор, пока результат на выходе хэш-функции не будет равен тому, что указан в hashed.txt.

Код, решающий задание, приведён ниже:

```
from sage.all import *

dihedral = DihedralGroup(1337)
dihedral_gen = dihedral.gens()[0]
dihedral_list = dihedral.list()
padder = 31337

enc = [277, 92, 775, 480, 160, 92, 31, 586, 277, 801, 355, 489, 801, 31, 62,
926, 725, 489, 160, 92,     31, 586, 277, 801, 355, 489, 1281, 62, 801, 489,
1175, 277, 453, 489, 453, 348, 725, 31, 348,     864, 864, 348, 453, 489, 737,
288, 453, 489, 889, 804, 96, 489, 801, 721, 775, 926, 1281, 631]


def unmap(index):
    return dihedral_list[index]

unmapped = [unmap(i) for i in enc]

def decrypt_byte(byte):
    for t in range(256):
        bits = bin(t * padder)[2:][::-1]
        answer = dihedral(())
        aggregator = dihedral(dihedral_gen)
        for bit in bits:
            if bit == "1":
                answer *= aggregator
            aggregator *= aggregator
        if answer == byte:
            return t

for b in unmapped:
    print(chr(decrypt_byte(b)), end="")
```

Данный код просто итерирует массив с результатами, полученными на выходе из функции при хэшировании флага, затем пробует хэшировать все возможные значения одного байта до тех пор, пока результат не будет соответствовать результату, полученному при пропускании через алгоритм конкретного байта флага. Затем выводит ответ в stdout:
nto{5tr4ng3_gr0up_5tr4ng3_l0g_and_depressed_kid_zxc_ghoul}

1.2. Crypto-2

В этом задании дан веб-сервер, у которого можно запросить бит под конкретным индексом во флаге, и в зависимости от того, является этот бит нулём или единицей, веб-сервер вернёт число, сгенерированное по определённому (Разному в каждом случае) алгоритму.

Для генерации чисел сервер использует определённое число **n**, которое является произведением двух простых чисел. Это число он не скрывает, его можно узнать, зайдя в корень сайта.

Если бит под запрашиваемым индексом единица, то он генерирует возвращаемое число по следующему алгоритму: число 7 возводится в рандомную (И очень большую) степень по модулю n. Если бит является нулём, то возвращается рандомное число в диапозоне от **n / 2** до **n.**

В обоих случаях получаются огромные числа, которые достаточно сложно анализировать, однако можно использовать тот факт, что если бит является нулём, то эти числа могут быть только в диапазоне от **n / 2** до **n**. Следовательно, если полученное число меньше **n / 2,** то бит под запрашиваемым индексом однозначно является единицей.

Программа, решающая Crypto-2, представлена ниже:

```
from Crypto.Util.number import *

import requests

addr = "http://10.10.26.10:1177"

session = requests.session()

page = session.request("GET", addr).text
n = int(page.splitlines()[0].split('=')[1].strip().replace("</p>", ""))

def get_bit(index):
    for _ in range(100):
        guess = int(session.request("GET", addr+f'/guess_bit?bit={index}').json()["guess"])
        if guess < n // 2:
            return 1
    return 0


bits = ""
byte_idx = 0
try:
    while 1:
        for bit in range(8):
            bits += str(get_bit(bit + byte_idx*8))
        byte_idx += 1
except:
    pass

print(long_to_bytes(int(bits, 2)))
```

Эта программа сначала запрашивает **n**, а потом запрашивает каждый бит по 100 раз. Если во время хотя бы одной итерации возвращенное значение менее **n / 2**, то значение вычисляемого на данный момент бита ставится на единицу и программа переходит к вычислению следующего. В ином случае значение ставится на ноль. **try**/**except** нужен для того, чтобы остановиться тогда, когда все нужные значения будут вычислены, т.к. при этом индекс вычисляемого бита станет больше длины флага и сервер вернёт ошибку (В возвращемом JSON не будет поля «guess», поэтому строчка, пытающеяся получить его, вызовет исключение). На сервере флаг перевоздится в число с помощью **bytes_to_long**, в конце программы перед принтингом эта операция реверсится. При запуске программа выдаёт нам флаг: nto{0h_n0_t1m1ng}

1.3. Reverse-1

В этом таске представлен exe файл, собранный для DOS. Если запустить его при помощи Dosbox, то он начнёт выводить флаг, но после каждого выведенного символа программа будет засыпать, причём с каждым разом на всё большее и большее время, поэтому просто так получить флаг не представляется возможным. Проанализировав ассемблерный код программы, декомпилированный с помощью radare2, находим две такие инструкции:

```
0000:006c      b486          mov ah, 0x86                ; 134
0000:006e      cd15          int 0x15
```

Посмотрев индекс системных вызовов DOS, понимаем, что это вызов, помещающий программу в сон на определённое время. Избавившись от этих инструкций, можно избавиться от задержки между выводом символов флага. Для этого достаточно переписать их на инструкции NOP с помощью HEX редактора или Питона. Эти две инструкции занимают четыре байта, так что мы заменяем их на четыре байта 0x90 (Соответствующих инструкции NOP), чтобы программа не засыпала. Результат сохраняем в файл и запускаем при помощи Dosbox, после чего мгновенно получаем флаг: nto{h3ll0_n3w_5ch000l_fr0m_old!!}

1.4. Web-2

Здесь одно приложение посылает запросы другому через HTTP. В первом приложении представлена простая система регистрации и аутентификации пользователей. Второе приложение получает на вход имя пользователя и флаг через хедер **Cookie.** Если флаг правильный, то оно возвращает «Hello, <Имя пользователя>», а если нет, то «I don’t trust you». Когда ты залогинен в первом приложении, то, при запрашивании главной страницы, оно формирует запрос, куда вставляет имя залогиненного пользователя и флаг, отсылает его второму приложению и рендерит результат на странице, которую посылает в ответ клиенту.

Здесь можно воспользоваться тем, что первое приложение не экранирует символы в имени пользователя при регистрации, а значит туда можно вставить символы, имелющие специальное значение в протоколе HTTP, тем самым изменив ответ второго приложения непредусмотренным образом. К примеру, можно испортить HTTP запрос, чтобы второе приложение вернуло ошибку. Для этого можно добавить в конец имени пользователя символы CR-LF-CR (\r\n\r). HTTP сервер будет ожидать, что после этого будет идти ещё один LF, однако его не будет, поэтому сервер вернёт ошибку. В запросе, который формирует первый сервис, сразу после этого будет идти флаг, поэтому второй сервис вернёт его в открытом виде в сообщении об ошибке.

При помощи Burp Suite изменяем запрос на регистрацию, добавляем в конец имени пользователя три указанных выше символа. Первый сервис возвращает нам валидный Cookie. После этого он перенаправляет нас в корень сайта, где на странице рендерится полученный от второго сервиса ответ с ошибкой, содержащий флаг:

Bad Request Bare CR or LF found in header line
" ;flag=NTO{request_smuggling_917a34072663f9c8beea3b45e8f129c5}" (generated by waitress)

2. Часть с исследованием зараженной Ubuntu

Способ решения

Для удобства работы мы смонтировали файловую систему вирутальной машины из файла vmdk при помощи `guestmount` и исследовали её

Как злоумышленник попал на машину?

Злоумышленник проник на машину через заражённый JAR-файл, замаскированный под лаунчер для Minecraft. Разархивировав архив можно найти директорию Malware, в которой будет файл ReverseShell.class. Декомпилировав его через JD-GUI, можно получить исходный код. При его прочтении подтверждается, что это реверс-шелл. Таким образом злоумышленник получил возможность исполнять команды на уже заражённой машине.

Как повысил свои права?

При помощи скрипта `linpeas.sh`, который делает это полностью автоматически. Этот самый скрипт можно найти в директории `/home/sergey/Downloads`

Как злоумышленник узнал пароль от password.kdbx ?

При помощи кейлоггера logkeys, записав нажатые на клавиатуре клавиши в то время, пока пользователь вводил пароль от БД

Куда logkeys пишет логи?

В файл `/var/log/logkeys.log`

Пароль от чего лежит в passwords.kdbx ?

Предположительно, от удалённого рабочего стола Windows (Судя по названию)