CTF Code

Writeups

Криптография

22 сентября 2021 г.

Оглавление

Easy

1 Based task

Достаточно простая задача. Нам дается файл с какими-то иероглифами:

Рис. 1: Китайцы уже близко

Казалось бы, что тут можно придумать, переводчик выдает какую-то дичь. Если заметить название таска, то можно подумать про какую-то кодировку из base'ов. Немного гуглинга и можно наткнуться на весьма интересную штуку под названием base65536. Прогнав через него натыкаемся на какую-то случайную последовательность эмодзи:

Рис. 2: Кто-то слишком эмоционален

Погуглив еще немного можно найти base100, после извлечения из которого получаем старый-добрый base64:

```
b3Jlbl9jdGZfS2V2aW5EYXZpZE1pdG5pY2shCg==
```

Рис. 3: То, что знакомо почти всем

И тут два варианта:

- Прогнать обратно ручками
- Написать питоновский скрипт

Чтобы райтап был полным, рассмотрим второй вариант, потому что первый достаточно очевидный и не требует пояснений. Скрипт для расшифровки выглядит примерно следующим образом:

Листинг 1: Дешифровка флага

```
#! /usr/bin/env python3
# -*- coding: utf-8 -*-

import base65536
import pybase100 as base100
import base64

with open('flag.enc', 'r') as flag_file:
    flag = flag_file.read()

flag = base65536.decode(flag)
flag = base100.decode(flag)
flag = base64.b64decode(flag)
flag = flag.decode('utf-8')

print(flag)
```

На выходе получаем флаг `oren_ctf_KevinDavidMitnick!`.

2 Hashes among us

Судя по виду зашифрованного флага и названию задачи перед нами какой-то хэш.

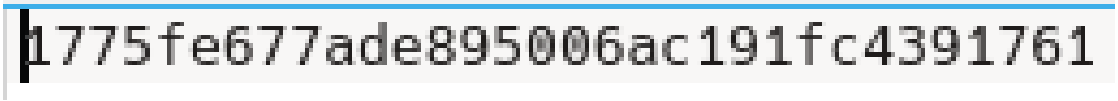

Рис. 4: Слишком много хексов

Если посмотреть длинну, то мы увидим, что длинна флага ровно 32 символа, что позволяет подумать про MD5. Дальше можно либо брутить локально с помощью HashCat/JhonTheRipper или воспользоваться онлайн-сервисами наподобие CrackStation. После чего получаем строку `RobertMorris`, которую нужно обернуть в `oren_ctf_` и `!`. После чего получаем флаг `oren_ctf_RobertMorris!`.

Medium

1 Please be careful with ASR

Судя по ключам и названию задчи речь явно идет про RSA. Что же, можно погуглить про атаки на этот алгоритм и наткнуться на весьма интересную штуку под названием Håstad's broadcast attack. После чего, если попробовать сдампить открытую экспоненту и модуль из наших открытых ключей, то можно увидеть, что во всех трех ключах экспонента маленькая и равна 3. Что уже точно намекает на эту атаку.

Суть атаки

Пользователь отсылает зашифрованное сообщение m нескольким пользователям. в данном случае, трём (по числу файлов): $P1$, $P2$, $P3$. У каждого пользователя есть свой ключ, представляемый парой «модуль-открытая экспонента» (n_i, e_i), причём $M < n1, n2, n3$. Для каждого из трех пользователей зашифровывает сообщение на соответствующем открытом ключе и отсылает результат адресату. Атакующий же реализует перехват сообщений и собирает переданные шифртексты (обозначим их как C_1, C_2, C_3), с целью восстановить исходное сообщение M. Значит, по имеющимся трем шифртекстам нужно восстановить сообщение, которое будет флагом.

Почему точно сможем ее реализовать?

Как известно, шифрование сообщения по схеме RSA происходит следующим образом: $C = M^e \pmod{n}$. В случае с открытой экспонентой, равной 3, получение шифртекстов выглядит так:

$$\begin{aligned} C_1 &= M^3 \pmod{n_1} \\ C_2 &= M^3 \pmod{n_2} \\ C_3 &= M^3 \pmod{n_3} \end{aligned}$$

Зная, что n_1, n_2, n_3 взаимно просты, можем применить к шифртекстам китайскую теорему об остатках. Получим в итоге некоторое C', корень кубический из которого и даст нам искомое сообщение M.

$$C' = M^3 \pmod{n_1 * n_2 * n_3}$$

Вспоминаем, что M меньше каждого из трёх модулей n_i, значит, справедливо равенство:

$$C' = M^3$$

Так мы и найдем наше искомое сообщение M.

Программная реализация атаки Хастада После всего вышесказанного не составляет труда написать простенький скрипт на питоне, который выдает зашифрованное сообщение:

Листинг 2: Атака Хастада

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import gmpy2
gmpy2.get_context().precision = 2048

from binascii import unhexlify
from functools import reduce
from gmpy2 import root
from Crypto.PublicKey import RSA

def chinese_remainder_theorem(items):
    N = 1
    for a, n in items:
        N *= n

    result = 0
    for a, n in items:
        m = N // n
        r, s, d = extended_gcd(n, m)
        if d != 1:
            raise "Input not pairwise co-prime"
        result += a * s * m

    return result % N


def extended_gcd(a, b):
    x, y = 0, 1
    lastx, lasty = 1, 0

    while b:
        a, (q, b) = b, divmod(a, b)
        x, lastx = lastx - q * x, x
        y, lasty = lasty - q * y, y

    return (lastx, lasty, a)
```

```
def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1:
        return 1
    while a > 1:
        q = a // b
        a, b = b, a % b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0:
        x1 += b0

    return x1

def get_cipher(filename):
    with open(filename, 'rb') as cipher:
        value = cipher.read().hex()

    return int(value, 16)

def get_modulus(filename):
    with open(filename) as keyfile:
        keystr = keyfile.read()
        key = RSA.import_key(keystr)

    return key.n

if __name__ == '__main__':

    ciphertext1 = get_cipher("flag.enc.alice")
    ciphertext2 = get_cipher("flag.enc.bob")
    ciphertext3 = get_cipher("flag.enc.eve")

    modulus1 = get_modulus("alice.pub")
    modulus2 = get_modulus("bob.pub")
    modulus3 = get_modulus("eve.pub")

    C = chinese_remainder_theorem([(ciphertext1, modulus1), (ciphertext2, mod
    flag = int(root(C, 3))
    flag = hex(flag)[2:]

    print(unhexlify(flag).decode('utf-8'), end='')
```

После выполнения скрипта на выходе получаем расшифрованный текст с флагом:

```
CIH, also known as Chernobyl or Spacefiller, is a Microsoft Windows 9x computer virus whic
```

2 Please don't share

При подключении к серверу мы видим сообщение

```
Here's a base64-encoded and encrypted flag: KSlIe5gsRMTPIaUlWr5pDpXFz8WrL9vqOHpDRkokWVhsIxzzWqF6Cfo7tt01br8H
You need a secret and literally zero ivent to get it!

Here is three part:
Part 1: 1-1510d2b3cacd8f387ff5b7eb52900b5ff60241072443620f7ae55d0efccc9fbb
Part 2: 2-42a67d81d71fae91aeb79dd30d61cd12f2afaf3c01d7102bf7df8ce068cad82d
Part 3: 3-88c1006a24f65e0b8c45b1b730754519db7640440f45c4af86ce1e8854b4ccf0
```

Рис. 1: Это "жжж"явно неспроста

Исходя из названия задачи и формулировки подсказки можно предположить, что речь идет о Shamir Secret Sharing Scheme (или иногда можно встретить сокращение SSSS). Помимо этого есть опечатка в слове `event`, что тоже дает какую-то подсказку. По подсказке понятно, что схемой разбит не сам флаг, а ключ для какого-то алгоритма шифрования с нулевым вектором инициализации, которым уже зашифрован флаг.

Немного о схеме Шамира

Если очень кратко, то секрет (информация, которой мы хотим поделиться) S разбивается на n частей таким образом, чтобы можно было восстановить исходное сообщение S только в случае, если мы имеем не менее k кусочков из n. Даже наличие $k-1$ куска недостаточно, чтобы восстановить исходную S. Чуть более подробно про это написано вот здесь.

Реализация

Можно писать разделение секрета самому, но это выходит за рамки данного райтапа. Если немного погуглить, то можно найти достаточно большое количество реализаций этого алгоритма на самых разных языках. Для питона существует вот эта реализация, но, к сожалению, она имеет проблемы с третьей версией из-за использования типа `long`, который был удален. Но ничего не мешает написать скрипт на втором. Итак, после получения полного секрета можно заметить, что это строка длинной 16 символов, что, в купе с нулевым инициализирующим вектором, дает возможность предположить, что используется AES-128. Остается лишь понять, какой из двух самых известных вариантов используется - ECB или CBC. Это можно сделать просто попробовав каждый из них. После чего можно написать небольшой питоновский скрипт, чтобы получить флаг:

Листинг 3: Расшифровка флага AES-CBC

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

import sys
from base64 import b64decode
from Crypto.Cipher import AES
from secretsharing import SecretSharer


if len(sys.argv) < 4:
    print("Usage: {} part1 part2 part2".format(sys.argv[0]))
    exit(1)

AES_KEY = SecretSharer.recover_secret([sys.argv[1], sys.argv[2], sys.argv[3]])
IV = b'\x00' * 16
cipher = AES.new(key=AES_KEY, mode=AES.MODE_CBC, iv=IV)

with open('flag.enc', 'r') as flagfile:
    enc_flag = b64decode(flagfile.read())

flag = cipher.decrypt(enc_flag)

print("Decrypted flag is: {}".format(flag))
```

Hard

1 Do you want to play some gamel?

2 Swift task