CTF Code

Writeups

Криптография

28 сентября 2021 г.

Оглавление

Easy

1 Based task

Теги: baseN, googling

> <условие задачи>

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


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

    return flag

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

    print(flag)


if __name__ == '__main__':
    main()
```

На выходе получаем флаг `oren_ctf_KevinDavidMitnick!`.

2 Hashes among us

Теги: hash, hash crack

<условие задачи>

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

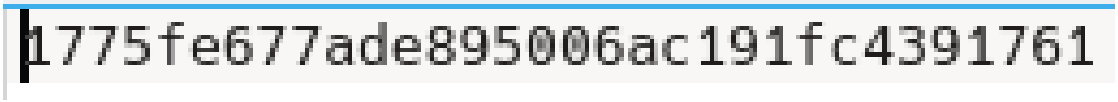

Рис. 4: Слишком много хексов

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

Теги: RSA, Hastad Attack

> <условие задачи>

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:

$$C_1 = M^3 \pmod{n_1}$$
$$C_2 = M^3 \pmod{n_2}$$
$$C_3 = M^3 \pmod{n_3}$$

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


def 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, modulus2),
                                        (ciphertext3, modulus3)])
    flag = int(root(C, 3))
    flag = hex(flag)[2:]

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


if __name__ == '__main__':
    main()
```

После выполнения скрипта на выходе получаем расшифрованный текст с флагом:

```
CIH, also known as Chernobyl or Spacefiller, is a Microsoft Windows 9x
computer virus which first emerged in 1998. Its payload is overwriting
critical information on infected system drives, and in some cases
destroying the system BIOS. Flag: oren_ctf_CIH!
```

2 Please don't share

Теги: Shamir's Secret Sharing, AES

<условие задачи>

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: Выглядит странно

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


def main():
    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[
    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))


if __name__ == '__main__':
    main()
```

После выполнения скрипта на выходе получаем расшифрованный флаг:
`Decrypted flag is: oren_ctf_ILOVEYOU_or_LoveLetter!`

Hard

1 Do you want to play some gamel?

Теги: Elgamal

> <условие задачи>

При подключении к серверу мы видим сообщение

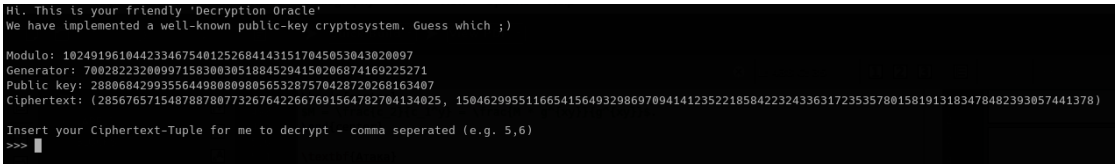

Рис. 1: Опять какая-то асимметрия

Судя по тому, что шифротекст разделен на два числа - перед нами криптосистема Эль-Гамаля.

Криптосистема Эль-Гамаля

Идея данной системы основана на сложности нахождения целого неотрицательного числа x, удовлетворяющего уравнению $g^x = a$ в циклической группе[1] (вообще говоря, такая операция называется дискретным логарифмированием).

Генерация ключа

Алгоритм генеарции ключа работает по следующему принципу:

- Выбираем случайное большое целое число q и строим циклическую группу G_q с образующим элементом g.

[1] Тут нужно некоторое уточнение. Циклическая группа $(G, *)$, если не вдаваться в дебри высшей математики, - множество для элементов которого мы определили по каким правилам можно их складывать (строго говоря, то сложение которое проходят в школе и это - немного разные вещи. В данном случае под сложением подразумевается операция, которая однозначно ставит в соответствие двум элементам e_1 и e_2 группы G некоторый элемент e_3, также принадлежащий группе G). Эта группа особенна тем, что результат операции может быть меньше операндов. По поводу цикличности все совсем просто - группа образуется некоторым элементом g, который называется образующим, а все остальные ее элементы это степени образующего. То есть $G = \{g^1, g^2, ...\}$. Группа становится циклической, если в $g^n = g$. Тогда число n называется порядком группы.

- Выбираем случайный целочисленный положительный x, т. ч. $1 \leq x \geq q-1$ и $(x, q) = 1$
- Считаем $h = g^x$
- Публичный ключ состоит из трех частей: (q, g, h)

Шифрование
Пусть мы хотим передать некоторое сообщение M. Тогда:

- Выбирается случайный целочисленный положительный y, $1 \leq y \geq q-1$ и $(y, q) = 1$
- Вычисляется секрет $s = h^y = (g^x)^y = g^{xy}$
- Вычисляется $c_1 = g^y$
- Вычисляется $c_2 = M * s$

Шифротекстом будет пара (c_1, c_2)
Дешифровка
Мы хотим расшифровать сообщение (c_1, c_2) и у нас есть приватный ключ x. Если провести некоторые математические выкладки:

$$\begin{cases} c_1 = g^y \\ c_2 = M * s \end{cases}$$

$$\Rightarrow$$

$$\begin{cases} c_1 = g^y \\ c_2 = M * h^y \end{cases}$$

$$\Rightarrow$$

$$\begin{cases} c_1 = g^y \\ c_2 = M * g^{xy} \end{cases}$$

То есть для получения оригинального сообщения M необходимо произвести следующие операции:

$$M = \frac{c_2}{c_1^y} = \frac{M * g^{xy}}{g^{xy}}.$$

Атака
Очевидно, что тот же самый шифротекст система отказывается расшифровывать. Но можно немного схитрить - умножить и поделить дробь на два. От этого, как известно, результат не поменяется:

$$c_2' = c_2 * 2$$
$$M * 2 = \frac{c_2'}{c_1^y} = \frac{c_2 * 2}{c_1^y} = \frac{M * g^{xy} * 2}{g^{xy}}.$$

После чего останется только поделить сообщение на два и получить флаг!

Реализация

После всего вышеперечисленного написать простенький скрипт на питоне не составляет труда. Для упрощения самому себе жизни я буду использовать библиотеку `pwntools` (вообще говоря, она задумывалась для работы с бинарщиной, но также через нее весьма удобно работать с сокетами):

Листинг 4: Взлом криптосистемы Эль-Гамаля

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

from pwn import *
import re


def get_cipher(msg):
    str_tp = re.search(r'\d*, \d*', msg).group(0)

    return tuple(map(int, str_tp.split(', ')))


def mul_tuple(tuple):
    new_tuple = (tuple[0], tuple[1] * 2)

    return   ', '.join(map(str, new_tuple))


def get_decrypt_flag(enc_flag):
    hexflag2 = ''.join('{:02x}'.format(ord(ch)) for ch in enc_flag)
    numflag2 = int(hexflag2, 16)
    numflag = numflag2 // 2
    hexflag = hex(numflag)[2:]

    return ''.join([chr(int(''.join(ch), 16)) \
                for ch in zip(hexflag[0::2], hexflag[1::2])])


def main():
    r = remote('localhost', 1488)

    msg = r.recvuntil('>>> ').decode('utf-8')
```

```
    tuple = get_cipher(msg)

    msg_for_flag = mul_tuple(tuple)
    r.sendline(msg_for_flag)

    enc_flag = r.recvline().decode('utf-8')
    flag = get_decrypt_flag(enc_flag)

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

    r.close()


if __name__ == '__main__':
    main()
```

На выходе скрипта получаем флага `oren_ctf_WannaCry!`

2 Curve task

Теги: Diffie-Hellman, Elliptic curve

<условие задачи>

В задаче применяется алгоритм Диффи-Хеллмана, но немного усложненная версия на элептических кривых. Традиционная аббревиатура: ECDH (Elliptic Curve Diffie-Hellman). Подробнее про криптографию на элептических кривых можно почитать вот тут. Нам дается клиент и сетевой дамп его общения с сервером.
Алгоритм клиента выглядит следующим образом:

- Подключение к серверу и получение параметра кривой: a, b, p (уравнение кривой выглядит как $y^2 = x^3 + a * x + b(\pmod p)$).
- Получение начальной точки G
- Подтверждение начальной точки (или отказ от нее и предложение своей)
- Вычисление секретного ключа a как случайного числа
- Отправка $G * a$
- Получение от сервера $G * d$ (где d - секретный ключ сервера)
- Вычисление общего ключа $s = a * (dG) = adG$

Сессия выглядит следующим образом:

```
Establishing secure connection via ECDH...
32593,1864507527,560147175803221327
2309607047,53492280
Accept? [y/n]
y
80842207261242668,482348047571127292
411081575920557992,302474624156986014
0RIGdbNapocC9vboA6JibpPFofBO111RQA10aSzSg5zxEjA8K2QFJpBDZTWOWQ==
jxJ263lpDOs1iu8PsFHFrYjNxc7LwKeXuSwlx2/Sjw==
cDxuXhw=
ugI=
gdOXQjD8ruH6VRtj0topdqS3SAUKy+1nee2XFCpZA/s=
uVf4eg==
uAVKR1ir
b6dQ3YQ0RWHPNE0Z7eRKNDlLCMoF59iJM/cwxaMyADGfxYjHqrQenQ==
njdw
SrmN
```

Рис. 2: Где-то здесь точно есть флаг

Как можно заметить, модуль p не очень большой: 560147175803221327 это примерно 59 бит. Поэтому в данном случае заходит решение "в лоб" через вычисление дискретного логарифма (см. предыдущую задачу). Можно попробовать самому реализовать какой-то из вариантов, но проще всего использовать sage:

```
sage: p = 560147175803221327
sage: E = EllipticCurve(GF(p),[32593,1864507527])
sage: g = E(2309607047,53492280)
sage: v1 = E(411081575920557992,302474624156986014)
sage: v2 = E(80842207261242668,482348047571127292)
sage: g.discrete_log(v1) * v2
(263085750118593959 : 99484538495976508 : 1)
sage: g.discrete_log(v2) * v1
(263085750118593959 : 99484538495976508 : 1)
sage:
```

Рис. 3: Где-то здесь точно есть флаг

После чего не составляет труда, используя данную библиотеку `cryptor` реализовать процесс дешифровки общения и получить флаг:

Листинг 5: Секреты перестают быть таковыми

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

import cryptor


def decrypt(cipher, crypted):
    return cipher.decrypt(crypted)


def main():
    server_msgs = [
        "0RIGdbNapocC9vboA6JibpPFofBO111RQA10aSzSg5zxEjA8K2QFJpBDZTWOWQ==",
        "jxJ263lpDOs1iu8PsFHFrYjNxc7LwKeXuSwlx2/Sjw==",
        "ugI=",
        "uVf4eg==",
        "b6dQ3YQ0RWHPNE0Z7eRKNDlLCMoF59iJM/cwxaMyADGfxYjHqrQenQ==",
        "SrmN"
    ]
    client_msgs = [
        "cDxuXhw=",
        "gdOXQjD8ruH6VRtj0topdqS3SAUKy+1nee2XFCpZA/s=",
        "uAVKR1ir",
        "njdw"
    ]

    priv_x = 263085750118593959
    priv_y = 99484538495976508

    client_cipher = cryptor.cryptor(priv_x, priv_y, "client")
    server_cipher = cryptor.cryptor(priv_x, priv_y, "server")

    print(decrypt(client_cipher, server_msgs[0]))
    print(decrypt(client_cipher, server_msgs[1]))
    for i in range(4):
        print(decrypt(server_cipher, client_msgs[i]))
        print(decrypt(client_cipher, server_msgs[i + 2]))


if __name__ == "__main__":
    main()
```

После чего можно увидеть о чем же общались люди:

```
Successfully established the secure connection
Waiting for operator to jump in
Hello
Hi
Can you give me the flag please?
Sure
Great!
Here is your flag: oren_ctf_TuringBombe!
Ty!
Bye
```