CTF Code

Writeups

Reverse Engineering

5 октября 2021 г.

Оглавление

Easy

1 Check the license!

Теги: Java, License key

> <условие задачи>

Нам дается программа на Java, которая хочет какую-то лицензию. Самое время ее разреверсить и посмотреть, что же там за лицензия нам нужна. Так как это Java, то можно восстановить исходный код с точностью до имен переменных с помощью любого декомпилятора. В райтапе будет использоваться JD-GUI. После открытия файла видим, что он совсем небольшой и состоит всего из трех классов:

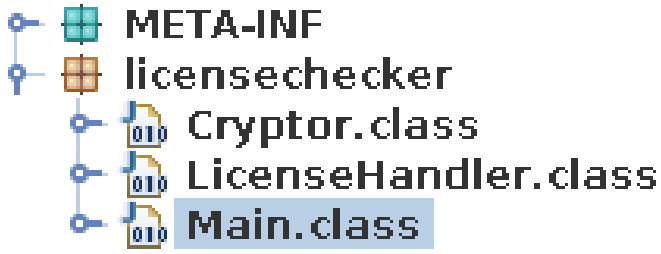


Рис. 1: Java изнутри

После рассмотрения `Main`'а понимаем, что это просто драйвер и ничего связанного с лицензией или ее обработкой не делает. С классом `LicenseHandler` ситуация интереснее, но тоже ничего нужного нам - ни расшифровки, ни каких-либо проверок. Просто чтение из класса и обращение к классу `Cryptor`, который, судя по всему, нам и нужен. Декомпилируем и смотрим:

```
package licensechecker;

public class Cryptor {
  private final byte[] HASH_PATTERN = new byte[] { 9, 67, 23, 83, 16, 70, 28 };

  private final String FLAG = "oren_ctf_z3r0d4y!";

  public String decrypt(byte[] encrypted) {
    StringBuilder msg = new StringBuilder();
    msg.append("oren_ctf_z3r0d4y!".substring(0, 9));
    for (int i = 0; i < 7; i++) {
      char ch = (char)("oren_ctf_z3r0d4y!".charAt(i + 9) ^ this.HASH_PATTERN[i]);
      msg.append(ch);
    }
    msg.append("oren_ctf_z3r0d4y!".charAt(16));
    return msg.toString();
  }

  public boolean hash(byte[] encryptedLicense) {
    if (encryptedLicense.length != 17)
      return false;
    int offset = encryptedLicense.length;
    int last = encryptedLicense.length + 1;
    if (encryptedLicense.length % 2 != 0) {
      offset++;
      last -= 2;
    }
    offset /= 2;
    for (int i = offset; i < last; i++) {
      if (encryptedLicense[i] != this.HASH_PATTERN[i - offset])
        return false;
    }
    return true;
  }
}
```

Рис. 2: Когда создал свою крипту

С первого взгляда флаг лежит прямо перед нами. Но это как-то слишком просто даже для easy-задачи. Посмотрим чуть ниже. Дейсвительно, сначала происходит какая-то проверка хэша. Если посмотреть внимательнее - никаких хешей нет. Сначала проверяем, что длина лицензии 17 символов, потом просто массив байтиков, с 9 по 15 элементы, сверяется с константой `HASH_PATTERN`. После чего в функции `decrypt` собирается флаг - обертка остается без изменений, а вот 7 символов ксорятся с `HASH_PATTERN`. После чего совсем не сложно написать простенький скрипт для ксора или (что еще проще) написать скрипт, который "сгенерирует"лицензию и скормить ее программе:

Листинг 1: Генератор лицензии

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

def main():
    xored = ['\x00', '\x00', '\x00', '\x00', '\x00', '\x00', '\x00',
    '\x00', '\x00', '\x09', '\x15', '\x17', '\x0c', '\x10', '\x13',
    '\x1c', '\x00']

    with open('license.bin', 'wb') as licensefile:
```

```
        for xb in xored:
            licensefile.write(bytes(xb, 'utf-8'))


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

И получаем флаг:

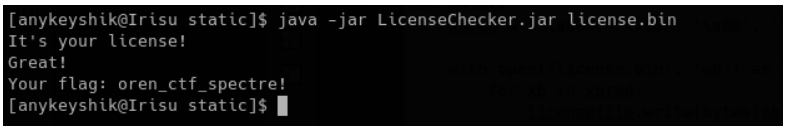


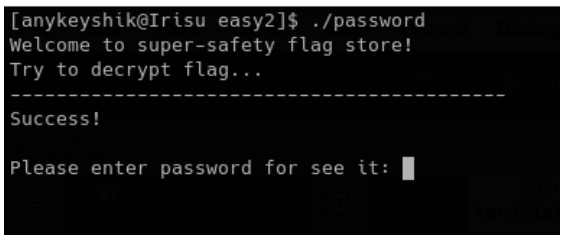
Рис. 3: Привет от Intel'a

2 Guess the password

Теги: C, ELF32, strip, dynamic, several ways of solve

<условие задачи>

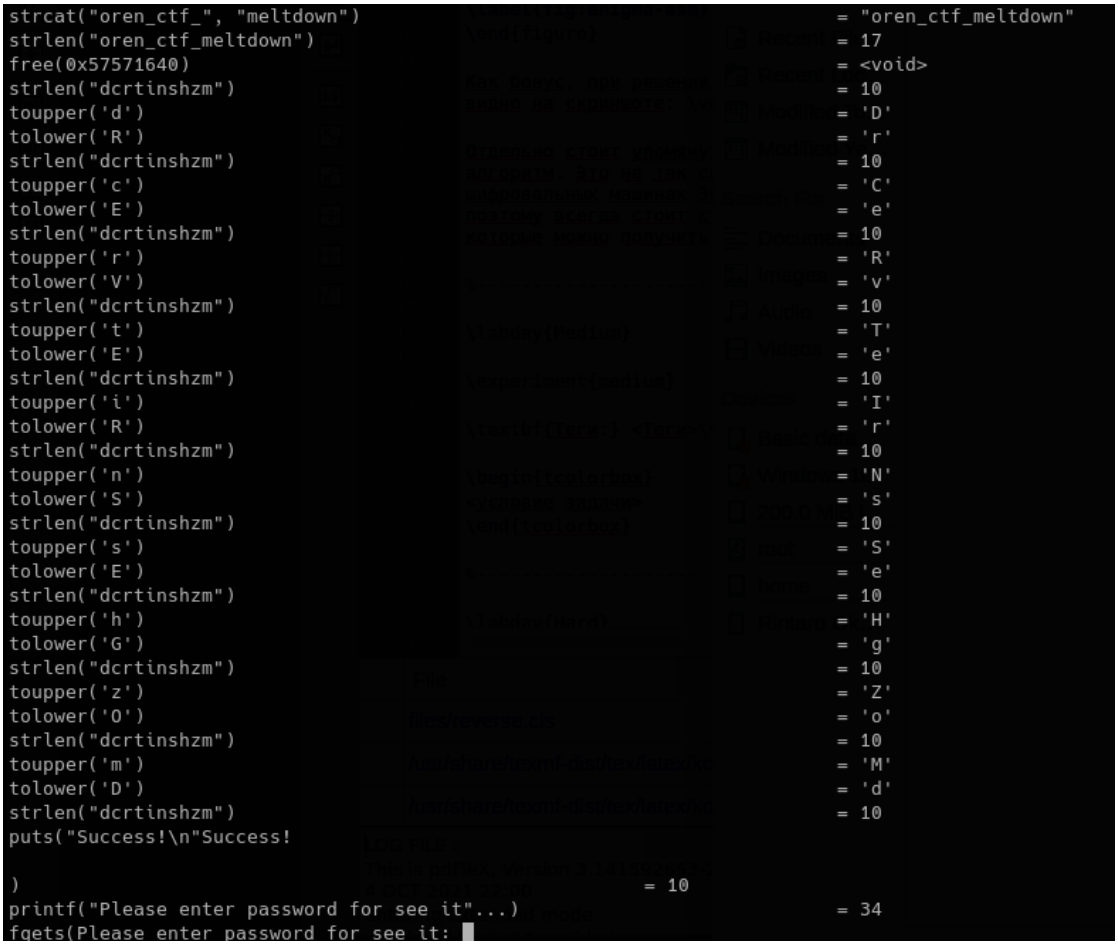
Программа расшифровывает флаг и ждет от нас пароля, чтобы отдать его нам.



Посомтрим, что же в этот момент происходит внутри:

```
lea     eax, (aWelcomeToSuper - 4000h)[ebx] ; "Welcome to super-safety flag store!"
push    eax             ; s
call    _puts
add     esp, 10h
sub     esp, 0Ch
lea     eax, (aTryToDecryptFl - 4000h)[ebx] ; "Try to decrypt flag..."
push    eax             ; s
call    _puts
add     esp, 10h
sub     esp, 0Ch
lea     eax, (asc_22C0 - 4000h)[ebx] ; "----------------------------------------"...
push    eax             ; s
call    _puts
add     esp, 10h
sub     esp, 0Ch
push    [ebp+ptr]
call    sub_18AC
add     esp, 10h
sub     esp, 8
push    [ebp+ptr]       ; int
push    [ebp+var_14]    ; s
call    sub_1AEA
add     esp, 10h
mov     eax, (off_40B0 - 4000h)[ebx] ; "dcrtinshzm"
sub     esp, 4
push    [ebp+s1]        ; int
push    eax             ; s
push    [ebp+ptr]       ; int
call    sub_124D
add     esp, 10h
sub     esp, 0Ch
lea     eax, (aSuccess - 4000h)[ebx] ; "Success!\n"
push    eax             ; s
call    _puts
add     esp, 10h
```

Глядя на этот листинг становится понятно, что действительно вызываются две функции. Судя по всему, одна из них для инициализации ключа, вторая для расшифровки. Таким образом, наш флга лежит в памяти еще до того, как программа спросила пароль. И тут появляется огромное количество возможных решений: к примеру, сдампить процесс, в дебаггере посмотреть содержимое кучи или, самый простой, - воспользоваться утилитой `ltrace`, чтобы отследить все библиотечные вызовы - они тут есть, в этом можно убедиться, если посмотреть, что импортирует программа. Есть второй, более сложный путь решения, - увидеть, что пароль сравнивается с помощью функции `strcmp` и поменять переход `jnz` на `jz` и, таким образом, при вводе неправильного пароля переходить на ветку, где программма печатает флаг. Ниже приведено решение с использованием `ltrace`:



Как бонус, при решении через `ltrace` можно также получить и пароль - это хорошо видно на скриншоте: `reversegod`.
Отдельно стоит упомянуть решение "в лоб" - просто посидеть и прореверсить весь алгоритм. Это не так сложно - в данном случае был использован алгоритм, применявшийся в шифровальных машинах Энигма. Но для простой задачи это очень времязатратная операция, поэтому всегда стоит ставить в соответствие временные затраты и количество баллов, которые можно получить за задачу.

Medium

1 <Название>

Теги: C++, ELF64, strip, dynamic

> <условие задачи>

Суть задания - разбор очень простого бинарного формата файла. Анализируем исполняемый файл и понимает, что он парсит файл кошелька довольно простым образом:

- Первым байтом файла является размер зашифрованного логина, который лежит после этого байта.
- Данный байт является инициализирующим значение для генератора гаммы с которой XOR'ится логин.
- Такой же алгоритм используется для пароля.
- Достаточно вытащить пароль и логин из кошелька и открыть его с помощью предоставленного исполняемого файла.
- Теперь мы можем просматривать все поля кошелька, однако нам нужно загрузить свой кошелёк с определённым балансом.
- Разбираем формат кошелька дальше, это не сложно делать, т.к. внутри бинарника есть функция вывода информации о кошельке, что позволяет достаточно просто и быстро определить смещения на поля и увидеть как они обрабатываются.
- Баланс сохраняется в 4 байта после пароля и "шифруется"путём XOR'а с константой 0xdeadbeef
- После загрузки кошелька с верным балансом, скорее всего, будет получено уведомление о том, что количество последних операций должно быть выше 16.
- Операции сохраняются ещё проще. После баланса идёт количество операций, а следующий байт это размер операции, все операции сохраняются таким образом и шифруются XOR'ом с размером.

- После создания операций в необходимом количестве будет получена ошибка, указывающая, что данный кошелёк не приватный. За это отвечает поле "Info".
- Поле Info ксорится с ключом, который формируется из логина и пароля
- Помещаем в поле Info строку "Private"(на это явно указано в сообщении об ошибке от сервера)
- Загружаем кошелёк и покупаем токен.
- Токен - это и есть флаг.

Таким образом, можно написать простой питоновский скрипт, который автоматизирует всю работу и просто "выдаст флаг на блюдечке с голубой коемочкой":

Листинг 2: Генератор кошелька

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

import sys
import random
import struct
import re

from pwn import *

BLANCE_KEY = 0xdeadbeef
ALPH = 'qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM0123456789'

p32 = lambda val : struct.pack( "!L", val  )
templates = ['debiting from this account %d bitcoins to %s account',
             'crediting from %s to a wallet %d bitcoins']


def idg(size = 16, chars = ALPH):
        return ''.join(random.choice(chars) for _ in range(size))


def GenRandomValue(seed):
    return (seed >> 1) & 0xff


def GenGamma(seed, sz):
    gamma = []

    for i in range(0, sz):
```

```
        value = GenRandomValue(seed)
        seed += value
        gamma.append(value)

    return gamma


def XorStringWithRandomGamma(string):
        res = ''

        seed = len(string)
        gamma = GenGamma(seed, seed)

        for i in range(len(gamma )):
                res += chr(ord(string[i]) ^ gamma[i])

        return res


class Wallet:
        login = None
        password = None
        balance = None
        last_operations = []
        info = None

        def __init__( self, Username, Password, Info ):
                self.login = Username
                self.password = Password
                self.balance = 13371337
                self.gen_random_operations()

                self.info = Info

        def gen_random_operations(self):
                global templates
                for i in range(0, 17):

                        template_id = random.randint(0, 1)

                        if template_id == 0:
                                self.last_operations.append(templates[template
                                % (random.randint(100, 512), idg()))
                        else:
```

```
                        self.last_operations.append(templates[template
                        % (idg(), random.randint(100, 512)))

    def pack_operations(self):
            for i in range(len(self.last_operations)):
                    operation = list(self.last_operations[i])

                    for j in range(len(operation)):
                            operation[j] = chr(ord(operation[j]) ^ len(ope

                    self.last_operations[i] = ''.join(operation)

    def pack_info( self, key ):
            self.info = list(self.info)

            for i in range( len(self.info )):
                    self.info[i] = chr(ord(self.info[i]) ^ ord(key[i % len

            self.info = ''.join(self.info)

    def pack_wallet( self ):
            self.pack_operations()

            res = ''
            # Pack login and password
            res += chr(len(self.login))
            res += XorStringWithRandomGamma(self.login)
            res += chr(len(self.password))
            res += XorStringWithRandomGamma(self.password)

            # Write balance
            res += p32(self.balance ^ BLANCE_KEY)

            # Pack all operations
            if len(self.last_operations) > 0:
                    res += chr(len(self.last_operations ))

                    for operation in self.last_operations:
                            res += chr(len( operation ))
                            res += operation
            else:
                    res += "\x00\x00\x00\x00"

            self.pack_info(self.login + self.password)
```

```
                # Pack info
                if len(self.info) > 0:
                        res += chr(len(self.info ))

                        res += self.info
                else:
                        res += "\x00\x00\x00\x00"

                return res.encode('hex')


if __name__ == "__main__":
    if len(sys.argv) > 2:
                host = sys.argv[1]
                port = int(sys.argv[2])
    else:
                print "Usage: " + sys.argv[0] + " <host> <port>"
                sys.exit(-1)

    login = 'AAAABBBBCCCCDDDD'
    password = login * 2

    wallet = Wallet(login, password, 'Private')
    enc_wallet = wallet.pack_wallet()

    r = remote(host, port)

    log.info("Upload wallet")
    r.sendline("2")

    log.info("Send wallet data")
    r.sendline(enc_wallet)

    log.info("Send login")
    r.sendline(login)

    log.info("Send password")
    r.sendline(password)

    log.info("Set as default")
    r.sendline("Y")

    log.info("Buy token")
```

```
r.sendline("3")

r.interactive()
```

В конце концов получаем флаг `oren_ctf_REvil!`

Hard

1 <Название>

Теги: <Теги>

<условие задачи>

2 <Название>

Теги: <Теги>

<условие задачи>