Второй отборочный этап

Задачи второго этапа

Второй отборочный тур проводится индивидуально в сети Интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Для всех участников предлагается единый набор задач в формате CTF (Capture The Flag).

Решение задач предполагает нахождение специальной последовательности символов — флага. Задания распределены по категориям. Задания каждой категории для решения требуют определенных знаний и навыков таких как: понимание основ работы операционных систем, веб-сервисов, знание различных форматов файлов, знание алгоритмов и умение реализовывать их на одном из языков программирования, знание основ криптографии и криптоанализа и т. д.

Задачи формата CTF не предполагают подробного описание условия задачи, т. к. цель, фактически, всегда одна — найти флаг. В описание лишь дается то, где его искать, например, файл или ссылка на веб-сервис. Любое другое описание может либо содержать дополнительную информацию, необходимую для решения задачи, либо служить подсказкой, чтобы задать правильное направление ходу мыслей решающего. Помимо этого, задание относится к одной из категорий, что позволяет понять какие знания и умения потребуются для его решения. Таким образом, достаточными исходными данными являются категория задачи и данные, в которых необходимо найти флаг.

Участники не были ограничены в выборе языка программирования или каких-либо вспомогательных программ для решения задач. На решение задач второго отборочного этапа участникам давалось 3 суток. Использовалась динамическая система оценивания — количество баллов за задание зависит от числа участников, которые его решили. Таким образом, чем больше участников решило задание, тем меньше оно будет стоить. При этом время решения заданий учитывается только при ранжировании участников, набравших одинаковое число баллов. Максимальное количество баллов за задание — 1000.

Категория Reverse

Задания, для решения которых необходимо уметь внимательно и аккуратно вникать в логику работы программы, чаще всего, не имея ее исходного кода. Но случается, что участникам дается обфусцированный исходный код, разобрать работу которого задача также не из самых простых. Форматы файлов могут быть самыми различными: PE, ELF, Mach-O, APK или даже просто байт-код программы для некоей виртуальной машины, спецификация которой дается участникам.

В заданиях этой категории обычно флаг спрятан где-то внутри программы в зашифрованном виде. И либо необходимо восстановить и переписать алгоритм шифрования, чтобы расшифровать флаг, либо выполнить какие-от действия (например,

записать определенное значение в реестр), чтобы программа в дальнейшем сама расшифровала и вывела на экран флаг.

Предположим, что есть следующее задание: после запуска программа предлагает ввести флаг и отвечает верный он или нет. В данном случае алгоритм решения будет следующий:

1. найти место, где происходит сравнение, в результате которого выдается вердикт;
2. тут введенные атакующим данные могут просто сравниваться с зашитым в теле программы верным флагом; Или же, верный флаг будет в некоем преобразованном виде (например, закодирован в base64);
3. чтобы сравнить вводимое значение с верным, программа сначала проделает все те же манипуляции, какие были проделаны изначально с верным флагом, с введенным флагом;
4. остается только применить преобразования в обратном порядке к зашитому в программе верному флагу в преобразованном виде.

Примеры используемых в решениях задач инструментов:

- Дизассемблер (IDA, Ghidra, radare2).
- Отладчик (gdb, lldb, WinDbg).
- Hex-редактор (Hiew, WinHex, Beye).
- Декомпиляторы (Hex-Rays, Ghidra, JADX, jd-gui, dotPeek).
- Инструменты для динамического анализа (Frida, qemu).

Задача II.1.1.1. Rainbow (608 баллов)

https://cyberchallenge.rt.ru/challenges#Rainbow

Как известно, на вкус и цвет все фломастеры разные. Поэтому мы принесли вам целую упаковку.

Описание

Дан скрипт на языке Python, осуществляющий проверку переданного через аргументы командной строки флага.

```
1  import sys
2
3  from unicorn import *
4  from unicorn.x86_const import *
5  from unicorn.arm_const import *
6  from unicorn.mips_const import *
7
8  STACK_ADDR = 0x1000
9  MEMSIZE = 0x2000
10
11 FLAG = b'\x06\x0eiFs9xk\x13\x0c2ZoBZk2ZrC'
12 FLAG_ADDR = MEMSIZE - len(FLAG)
13
14 user_flag = ""
15
```

```
def x86_encrypt():
    global user_flag
    code = b"\x51\x89\xf7\x31\xc0\xac\x50\xe2\xfa\x8b\x4c\x24\x50\x31\xd2\x31\xc0 \\
    \x09\xc8\xb3\x04\xf6\xf3\x89\xc1\x58\x48\xc1\xe0\x18\x5b\x4b\xc1\xe3\x10 \\
    \x09\xd8\x5b\x4b\xc1\xe3\x08\x09\xd8\x5b\x4b\x09\xd8\xab\xe2\xe6"

    try:
        mu = Uc(UC_ARCH_X86, UC_MODE_32)

        mu.mem_map(0, MEMSIZE)
        mu.mem_write(0, code)
        mu.mem_write(FLAG_ADDR, bytes(user_flag))

        mu.reg_write(UC_X86_REG_ESP, STACK_ADDR)
        mu.reg_write(UC_X86_REG_ESI, FLAG_ADDR)
        mu.reg_write(UC_X86_REG_ECX, len(FLAG))

        mu.emu_start(0, len(code))

        user_flag = mu.mem_read(FLAG_ADDR, len(FLAG))

    except UcError as e:
        print("ERROR: %s" % e)

def mips_encrypt():
    global user_flag
    code = b"\x00\x80\x40\x25\x00\xa0\x48\x25\x24\x0a\x00\x02\x01\x2a\x00\x1a\x00 \\
    \x00\x48\x12\x85\x0a\x00\x00\x39\x4a\xfa\xce\xa5\x0a\x00\x00\x21\x08\x00 \\
    \x02\x21\x29\xff\xff\x1d\x20\xff\xfa\x00\x00\x00\x00"

    try:
        mu = Uc(UC_ARCH_MIPS, UC_MODE_MIPS32 + UC_MODE_BIG_ENDIAN)

        mu.mem_map(0, MEMSIZE)
        mu.mem_write(0, code)
        mu.mem_write(FLAG_ADDR, bytes(user_flag))

        mu.reg_write(UC_MIPS_REG_4, FLAG_ADDR)
        mu.reg_write(UC_MIPS_REG_5, len(FLAG))

        mu.emu_start(0, len(code))

        user_flag = mu.mem_read(FLAG_ADDR, len(FLAG))

    except UcError as e:
        print("ERROR: %s" % e)

def arm_encrypt():
    global user_flag
    code = b"\x3e\x00\x90\xe8\x61\x64\xa0\xe1\x62\x14\xa0\xe1\x63\x24\xa0\xe1\x64 \\
    \x34\xa0\xe1\x65\x44\xa0\xe1\x06\x50\xa0\xe1\xbe\x6a\x0b\xe3\xfe\x6a\x4c \\
    \xe3\x06\x00\x51\xe1\x66\x18\x21\xc0\x06\x10\x21\xd0\x06\x20\x22\xe0\x06 \\
    \x30\x23\xe0\x06\x40\x24\xe0\x06\x50\x25\xe0\x3e\x00\x80\xe8"

    try:
        mu = Uc(UC_ARCH_ARM, UC_MODE_ARM)

        mu.mem_map(0, MEMSIZE)
        mu.mem_write(0, code)
        mu.mem_write(FLAG_ADDR, user_flag.encode('utf-8'))
```

```
76
77          mu.reg_write(UC_ARM_REG_R0, FLAG_ADDR)
78
79          mu.emu_start(0, len(code))
80
81          user_flag = mu.mem_read(FLAG_ADDR, len(FLAG))
82
83      except UcError as e:
84          print("ERROR: %s" % e)
85
86  def wrong():
87      print("Wrong flag")
88      sys.exit(0)
89
90  def correct():
91      print("Correct flag! Congratz!")
92      sys.exit(0)
93
94  def main():
95      global user_flag
96      if len(sys.argv) < 2:
97          print("usage: python3 rainbow-checker.py <flag>")
98          return;
99
100     user_flag = sys.argv[1]
101     if len(user_flag) != len(FLAG):
102         wrong()
103
104     arm_encrypt()
105     mips_encrypt()
106     x86_encrypt()
107
108     if user_flag == FLAG:
109         correct()
110     else:
111         wrong()
112
113 if __name__ == "__main__":
114     main()
```

Решение

Открыв данный скрипт в любом текстовом редакторе, легко заметить, что переданное пользователем через аргументы командной строки значение флага преобразуется с помощью последовательного вызова 3-х функций с довольно говорящими именами: arm_encrypt, mips_encrypt и x86_encrypt. Далее, полученная из пользовательского флага последовательность байт, сравнивается с заданной последовательностью байт в скрипте.

Все 3 функции, участвующие в преобразовании введенного пользователем флага, идеологически похожи — каждая из них эмулирует выполнения машинного кода для определенной аппаратной платформы (архитектуры процессора). Какой именно платформы — легко понять из имени функции, или же прочитав ее код.

Таким образом, становится понятен план дальнейших действий для решения задания — нужно дизассемблировать машинный код, эмулируемый каждой из функций, а затем выполнить над заданным в скрипте массивом байт (с которым в резуль-

тате сравнивается значение, полученное из пользовательского флага) преобразования обратные тем (и в обратном порядке), что выполняются машинным кодом.

Для дизассемблирования кода под разные аппаратные платформы можно воспользоваться любым поддерживающим их дизассемблером или же написать для этого простой скрипт на Python с использованием модуля capstone (https://www.capstone-engine.org/lang_python.html).

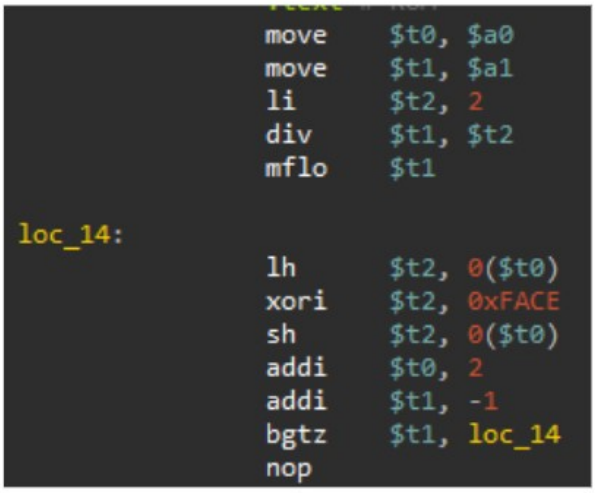

Рис. II.1.1: Пример дизассемблированного кода для MIPS

В результате, решение задачи на языке Python выглядит следующим образом:

```
flag = bytearray(b'\x06\x0eiFs9xk\x13\x0c2ZoBZk2ZrC')

def x86_decr():
        global flag

        for i in range(len(flag)):
            flag[i] = flag[i] + 1

        parts = [bytes(flag[i:i+4]) for i in range(0, len(flag), 4)]
    parts.reverse()

        flag = bytearray(b"".join(parts))

def mips_decr():
        global flag

        for i in range(int(len(flag)/2)):
        flag[i*2] ^= 0xfa
            flag[i*2+1] ^= 0xce

def arm_decr():
        global flag

        parts = []
        for i in range(int(len(flag)/4)):
        parts.append([flag[i*4], flag[i*4+1], flag[i*4+2], flag[i*4+3]])

        parts[0] = bytes([parts[0][3]^0xba, parts[0][0]^0xfe, parts[0][1]^0xca,
        ↪  parts[0][2]^0xbe])
        for i in range(1, len(parts)):
```

```
30              parts[i] = bytes([parts[i][3]^0xca, parts[i][0]^0xbe, parts[i][1]^0xba,
                ↪  parts[i][2]^0xfe])
31
32      parts.insert(0, parts[len(parts)-1])
33          parts = parts[:len(parts)-1]
34
35          flag = bytearray(b"".join(parts))
36
37  x86_decr()
38  mips_decr()
39  arm_decr()
40
41  print(bytes(flag).decode('utf-8'))
```

Ответ: CC{n07_7h47__Py7h0N}.

Задача II.1.1.2. A Bit Classics (919 баллов)

https://cyberchallenge.rt.ru/challenges#A%20Bit%20Classics

Вот она — ваша уникальная возможность познакомиться с классикой и пройти ее до конца!

Описание

Дан образ ROM для NES (Nintendo Entertainment System) с игрой Battletoads.

Решение

Текст задания довольно недвусмысленно намекает на то, что флаг ожидает решающего где-то в конце игры.

Для решения понадобится эмулятор NES. Удобнее всего использовать эмулятор FCEUX, т.к. в нем есть встроенный отладчик и hex-редактор.

Весь текст используемый в игре хранится в 2 представлениях: в открытом виде и в сжатом с помощью алгоритма Хаффмана. Все монологи и диалоги — это второй случай. Процедура декодирования текста начинается в 6-ом 32-килобайтовом блоке ROM по смещению 0xE60F от его начала (т. е. в самом по смещению 0x3661F от начала исходного файла). Кодовое дерево для алгоритма Хаффмана, используемое в этой игре, можно подсмотреть, например, тут: http://rainwarrior.ca/projects/nes/battletoads_huffman.png

Сжатый текст же начинается по смещению 0xE6F9 от начала того же 6-го блока и заканчивается на смещении 0xFE87. Теперь, если взять сжатый текст и применить к нему алгоритм Хаффмана с правильным кодовым деревом, то получим... какой-то бессмысленный набор букв. Все дело в том, что в отличии от оригинального образа ROM, тут есть немного дополнительного кода.

Чтобы понять, где именно находится дополнительный код, можно найти в Интернете оригинальный образ и бинарно сравнить его с образом из задания. Окажется, что в код внесены следующие 2 небольших изменения:

```
3662B: 20 E0 FE  jsr $fee0
                 EA  nop
...
37EF0: 49 1F        eor #$1F
       85 1B        sta $1B
       E6 1C        inc $1C
          60        rts
```

Получается, что перед декомпрессией данных в текст, к каждому байту применяется операция XOR с байтом 0x1F.

В результате, добавив эту операцию к решению, получим текст всех диалогов/монологов в игре, среди которого можно заметить следующий патченный фрагмент (из финального монолога главного антагониста игры):

```
I LOST! THIS CAN'TA БВЕ! I'M CC!THERE-ARE-NO-A ГHARD-GAMES-FOR-NES! A DARK
IS STRONGERA ДTHAN LIGHT! GRRRR!
```

Убрав служебные символы и заменив «!» на «{» и «}», получим искомый флаг.

Ответ: CC{THERE-ARE-NO-HARD-GAMES-FOR-NES}.

Категория Crypto

Криптография — наука о том, как преобразовать исходные данные таким образом, чтобы обеспечить их защиту от посторонних, а также защитить от подмены или сделать ее невозможной.

Задания данной категории предполагают применение знаний в математике и криптоанализе для решения криптографических головоломок, будь то простой шифр замены или же некорректно использованный шифр RSA.

Обычно алгоритм решения задач этой категории выглядит следующим образом:

1. Понять, что за шифр использовался.
2. Узнать возможные атаки на этот шифр.
3. Выяснить, начальные условия для какой из возможных атак выполняются в данном случае.
4. Применить выбранную атаку.

Примеры используемых в решениях задач инструментов:

- CrypTool.
- xortool.
- RsaCtfTool.
- Языки программирования (python, perl, js и т. д.).

Задача II.1.2.1. Encrypccino (619 баллов)

Кажется, кто-то забыл пароль от своих зашифрованных файлов. Очень странно, ведь эта программа специально была создана для того, чтобы пароли от зашифрованных файлов было легко запоминать.

`https://cyberchallenge.rt.ru/files/6dd5bcadcd8d03cdb99defe24ad0e00d/encrypccino.tar.gz`

Описание

Дан исходный код программы на языке Python, которая позволяет шифровать и расшифровывать файлы. При шифровании файлов пользователю выдается случайная парольная фраза из двух словарных слов, которые необходимо запомнить, чтобы иметь возможность в будущем расшифровать текст. Словарь words.txt содержит порядка 370 тысяч слов, что означает, что количество возможных комбинаций превышает 10^11, что делает перебор парольной фразы затруднительным.

Помимо программы, даны зашифрованные файлы prose.txt.enc и flag.txt.enc, а также один исходный файл prose.txt. Необходимо расшифровать файл с флагом.

Решение

Изучим принцип работы программы. Для шифрования используется алгоритм AES-256 в режиме CBC. Данные шифруются два раза: сначала первым парольным словом, затем вторым. В этом и заключается уязвимость алгоритма, поскольку можно провести атаку типа «встреча посередине». Для этого можно для каждого словарного пароля, число которых невелико, с одной стороны, зашифровать один раз исходный файл prose.txt, а с другой стороны, попробовать расшифровать один раз файл prose.txt.enc. Нахождение пересечения между этими двумя множествами позволяет восстановить используемую парольную фразу. Расшифровка файла flag.txt.enc с данной фразой позволяет получить флаг.

Ответ: CC{all_i_w4nt3d_w4s_a_D0UBL3_3ncrypcc1n0}.

Задача II.1.2.2. I'm one yet many (670 баллов)

Кажется, кто-то запустил эту программу и сохранил ее вывод в output.txt. Жаль, что все зашифровано...

`https://cyberchallenge.rt.ru/files/3e7c928b7d476fc23706160ad7f15022/output.txt`

```python
#!/usr/bin/env python3
import os


def encrypt(key, plaintext):
    assert len(key) >= len(plaintext)
    return bytes(x ^ y for x, y in zip(key, plaintext)).hex()


def main():
    key = os.urandom(10 * 1024)
    with open("flag.txt", "rb") as f:
        flag = f.read()
    print(encrypt(key, flag))
    with open("data.bin", "rb") as f:
```

```
        for line in f.readlines():
            print(encrypt(key, line))


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

Описание

Программа на языке Python генерирует случайный ключ достаточной длины, после чего построчно шифрует бинарный файл с неизвестным содержимым и флаг. Результат сохраняется в файл output.txt, который дан участникам. В качестве метода шифрования используется гаммирование. Ключ переиспользуется для каждой строки файла.

Решение

Заметим, что ключ переиспользуется, то есть «одноразовый блокнот» на самом деле является многоразовым. Однако, само по себе это ничего не значит, поскольку содержимое исходного файла и его характеристики неизвестны, а значит, его можно считать случайным. Тем не менее, при считывании содержимого файла построчно с помощью метода readlines у каждой строки line последним символом является символ перевода строки «\n». Поскольку длины строк в бинарном файле распределены случайно, из одной зашифрованной строки можно извлечь один символ секретного ключа. Таким образом, можно восстановить ключ целиком и расшифровать флаг.

Пример скрипта, выполняющего расшифрование:

```
#!/usr/bin/env python3
def xor(a, b):
    return bytearray(x ^ y for x, y in zip(a, b))


def main():
    key = bytearray(b"\x00" * 10240)
    with open("output.txt") as f:
        data = [bytes.fromhex(line.rstrip()) for line in f.readlines()]
    for line in data:
        key[len(line) - 1] = line[-1] ^ ord("\n")
    flag = data[0]
    print(xor(key, flag))


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

Ответ: CC{w0w_s0_crypt0_xe7eimaix7eic0iephohng7ahvaT9Aech0oB2ieSeis2duuphaef 4wiulao6AiGhu0aiQuaeNg0wi7maitouyeighohree7Nueliaitaece1ugh}.

Задача II.1.2.3. Firmware (885 баллов)

Кажется, это устройство можно прошивать только оригинальными прошивками от производителя.

http://firmware.nti.2020.tasks.cyberchallenge.ru/

`https://cyberchallenge.rt.ru/files/a8952a6a13ff498e2e47ddc5001d0e16/service.tar.gz`

`https://cyberchallenge.rt.ru/files/9881ec2c46905084ebf3536df35a5623/firmware.tar`

Описание

Дана ссылка на веб-сервис административной панели модельного устройства. Через данный сервис возможно перепрошить данное устройство при наличии файла с прошивкой. Однако, производитель устройства разрешил перепрошивку только с использованием оригинальных прошивок от производителя. Для этого в устройстве была реализована проверка криптографической подписи. Исходные коды сервиса и пример оригинальной прошивки предоставлены, но закрытый ключ для подписи производитель сохранил в секрете. Для получения флага необходимо обойти проверку и перепрошить устройство самодельной прошивкой.

Решение

Изучим исходный код сервиса. Прошивка представляет из себя tar-архив, содержащий набор бинарных файлов и файл с подписью. Используется асимметричный алгоритм цифровой подписи RSA PSS над хэш-суммой прошивки, которая формируется как побитовое «или» всех хэш-сумм SHA-256 имен и содержимого файлов архива. Сам алгоритм RSS PSS используется корректно, однако, способ комбинации хэш-сумм имеет недостаток.

Хэш-суммы можно интерпретировать как вектора линейного пространства размерности 256 над Z2, а их комбинацию — как операцию сложения векторов. Пусть у нас есть подпись для хэш-суммы h_0, соответствующей файлам оригинальной прошивки. В случае внесения изменений в прошивку, потребуется подпись для хэш-суммы h_1, которую без закрытого ключа получить затруднительно. Однако, если мы найдем набор файлов, соответствующей хэш-сумме $(h_0 + h_1)$, и добавим к уже существующим файлам измененной прошивки, то результирующая хэш-сумма архива снова будет равна $h_1 + (h_0 + h_1) = h_0$, поэтому оригинальная подпись подойдет и к такому модифицированному архиву.

Таким образом, для проведения атаки достаточно сгенерировать множество файлов, соответствующих хэш-сумме $(h_0 + h_1)$. Для этого мы можем составить базис линейного пространства и решить соответствующую линейную систему уравнений. Базис можно составить, генерируя случайным образом файлы, подсчитывая их SHA-256 хэш, и добавляя соответствующий вектор-столбец к матрице. В случае, если ранг матрицы увеличился на 1, то мы продолжаем этот процесс до достижения полной размерности линейного пространства. Если же ранг матрицы не увеличился, то мы отбрасываем этот вектор и пробуем случайным образом сгенерировать новый. Получим 256 базисных векторов, из которых можно линейной комбинацией собрать нужный целевой вектор, соответствующий хэш-сумме $(h_0 + h_1)$, решив систему линейных уравнений.

Осталось лишь добавить необходимые файлы в tar-архив и загрузить в веб-сервис, чтобы получить флаг.

Ответ: CC{s0_you_h4v3_h4ck3d_th3_f1rmw4re_c0ngr4ts}.

Категория Forensics

Компьютерная криминалистика — прикладная наука о поиске и исследовании доказательств совершения различных действий, связанных с компьютерной информацией.

Чтобы справляться с задачами компьютерной криминалистики, надо иметь представление об основах работы ОС, понимать строение файловой системы и взаимодействие различных процессов. В ходе работы необходимо анализировать образы дисков, дампы памяти, дампы сетевых пакетов, а также логи и т.п.

Как правило, в заданиях данной категории необходимо разобраться, что происходило на машине жертвы, с который был получен дамп или логи для задания. Любая обнаруженная аномалия может привести к флагу. Например, в дампе памяти обнаружилось, что в списке запущенных процессов есть графический редактор Paint, далее, нужно добраться до изображения, открытого в нем, и увидеть на нем флаг. Универсального рецепта нет, с каждым типом дампа или другого файла нужно работать по-разному.

Примеры используемых в решениях задач инструментов:

- Виртуальная машина (VirtualBox, VMWare и т.п.).
- Volatility.
- WireShark.
- Binwalk.
- foremost.
- Hex-редакторы.
- Дизассемблер.
- Отладчик.

Задача II.1.3.1. Hidden Treasure (919 баллов)

https://cyberchallenge.rt.ru/challenges#Hidden%20Treasure

У нас на одном из компьютеров в бухгалтерии творится какая-то чертовщина. Наверное, Windows надо переустановить.

P.S. Вот дамп памяти с этой машины. Пароль от архива: cyb3rch4llenge.

Описание

Дан дамп памяти с машины, на которой установлена Windows 7. Необходимо найти следы компрометации машины и в конечном итоге получить флаг.

Решение

Для исследования дампов оперативной памяти удобно воспользоваться фреймворком volatility. Для начала, нужно определиться с профилем, т. е. понять какая ОС была установлена на машине, с которой сняли дамп памяти. Сделать это можно с помощью следующей команды:

```
> volatility -f infected.vmem imageinfo
```

Далее, полезно будет узнать какие процессы были запущены в системе в момент снятия дампа:

```
> volatility -f infected.vmem -profile Win7SP1x86 pstree
```

И тут сразу наше внимание должен привлечь следующий процесс:

```
Name                                Pid   PPid   Thds   Hnds  Time
------------------------------ ------ ------ ------ ------ ----
. 0x864ce3f0:FlagReader.exe     2516   1328    1              7   2020-12-08 18:01:11
↪  UTC+0000
```

Чтобы понять, что делает этот процесс, нужно извлечь его из памяти в отдельный файл и исследовать его в дизассемблере/декомпиляторе:

```
> volatility -f infected.vmem -profile Win7SP1x86 procdump -pid 2516 -D
```

Проанализировав код FlagReader.exe, легко увидеть, что все, что он делает, — это читает строку из файла flag.txt (по всей видимости именно там хранился искомый нами флаг) в буфер на стеке длиной 1024 байта (т. е. 1 Кб). Далее, программа в бесконечном цикле просто итеративно читает байты в буфере.

Получить содержимое flag.txt с помощью команд volatility filescan и dumpfiles, к сожалению, не получится — содержимого файла по каким-то причинам нет в дампе. Но теперь мы знаем, что FlagReader.exe прочитал содержимое этого файла и сохранил в буфере на стеке, и можно поискать его там.

Чтобы найти адреса начала и конца стека в дампе, нам понадобится найти структуру TEB (Thread Environment Block), описывающую единственный поток процесса FlagReader.exe. Для этого сначала воспользуемся командой volatility threads:

```
> volatility -f infected.vmem -profile Win7SP1x86 threads -p 2516

ETHREAD: 0x84d9a648 Pid: 2516 Tid: 2520

...

TEB: 0x7ffde000
```

Адреса начала и конца стека хранятся в TEB по смещениям [TEB+4] и [TEB+8], т. е. нам нужно получить значения по адресам 0x7ffde004 и 0x7ffde008. Для этого воспользуемся командой volatility volshell:

```
> volatility -f infected.vmem -profile Win7SP1x86 volshell

>>> cc(pid=2516

Current context: FlagReader.exe @ 0x864ce3f0, pid=2516, ppid=1328 DTB=0x3e803280

>>> dd(0x7ffde000)

7ffde000: 001bfb6c 001c0000 001be000 00000000
```

Таким образом, получается, что искомый нами стек в адресном пространстве процесса FlagReader.exe начинается с виртуального адреса 0x001c0000 и заканчивается

на 0x001be000 (стек растет от старших адресов к младшим). Чтобы отыскать этот регион памяти в исходном дампе, потребуется найти соответствующие виртуальным адресам физические адреса. Для этого воспользуемся командой volatility memmap:

```
> volatility -f infected.vmem -profile Win7SP1x86 memmap > map.txt
```

И перейдя к секции, относящейся к процессу FlagReader.exe, найдем нужные нам адреса:

```
FlagReader.exe pid:   2516
Virtual    Physical     Size     DumpFileOffset
---------- ---------- ---------- --------------
...
0x001be000 0x2fe83000   0x1000             0xa000
0x001bf000 0x30976000   0x1000             0xb000
0x001c0000 0x19618000   0x1000             0xc000
```

Во 2-ом столбце указаны нужные нам физические адреса, по которым мы можем найти страницы памяти стека в исходном дампе памяти. Однако тут удобнее воспользоваться данными из 4-го столбца — там указаны смещения на соответствующие регионы памяти в дампе памяти процесса. Получить дамп памяти процесса FlagReader.exe можно следующей командой:

```
> volatility -f infected.vmem -profile Win7SP1x86 procdump -pid 2516 -D
```

Теперь нам остается только открыть полученный дамп памяти процесса в любом hex-редакторе и перейти по смещению 0xa000 от начала файла. Следующие 0x2000 байт и будут искомым нами стеком.

Пробежавшись глазами по данным, хранящимся в стеке, несложно заметить, что искомый флаг среди них отсутствует. По крайней мере в открытом виде. Тем не менее, судя по коду FlagReader.exe, он только читает содержимое flag.txt, но никак его не меняет. Остается только предположить, что что-то извне вмешалось в работу FlagReader.exe, или же изменило данные в его памяти.

В таком случае, можно проверить, не внедрялся ли инородный код в процесс FlagReader.exe командой volatility malfind:

```
> volatility -f infected.vmem --profile Win7SP1x86 malfind -p 2516
Process: FlagReader.exe Pid: 2516 Address: 0x60000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00060000  60 9c b9 02 00 00 00 89 e6 83 c6 04 8b 06 25 ff   `.............%.
0x00060010  ff ff 00 35 15 77 69 00 3d 56 34 12 00 75 ea 81   ...5.wi.=V4..u..
0x00060020  36 5c 74 ad be 81 76 04 73 45 bc af 81 76 08 f8   6\t...v.sE...v..
0x00060030  ef a3 27 81 76 0c 3a be 2d ec 83 c6 10 49 75 df   ..'.v.:.-....Iu.

0x00060000 60                   PUSHA
0x00060001 9c                   PUSHF
0x00060002 b902000000           MOV ECX, 0x2
0x00060007 89e6                 MOV ESI, ESP
0x00060009 83c604               ADD ESI, 0x4
0x0006000c 8b06                MOV EAX, [ESI]
0x0006000e 25ffffff00           AND EAX, 0xffffff
0x00060013 3515776900           XOR EAX, 0x697715
0x00060018 3d56341200           CMP EAX, 0x123456
0x0006001d 75ea                 JNZ 0x60009
0x0006001f 81365c74adbe         XOR DWORD [ESI], 0xbead745c
```

```
0x00060025 8176047345bcaf    XOR DWORD [ESI+0x4], 0xafbc4573
0x0006002c 817608f8efa327    XOR DWORD [ESI+0x8], 0x27a3eff8
0x00060033 81760c3abe2dec    XOR DWORD [ESI+0xc], 0xec2dbe3a
0x0006003a 83c610               ADD ESI, 0x10
0x0006003d 49                   DEC ECX
0x0006003e 75df                 JNZ 0x6001f
```

И обнаруживаем, что в памяти процесса FlagReader.exe присутствует дополнительный RWX регион памяти, содержащий некий шеллкод. Прочитав данный шеллкод, легко понять, что его единственная задача — это найти, начиная с вершины стека, последовательность байт начинающуюся с «CC{» (т. е. искомый нами флаг) и зашифровать ее с помощью операции XOR.

Теперь, зная ключ шифрования, мы можем попытаться найти точное расположение флага на стеке и расшифровать его. Применив операцию XOR с ключом «5C 74 AD» к подстроке «CC{», получим, что зашифрованный флаг должен начинаться с последовательности байт «1f 37 d6». Такая последовательность встречается на стеке ровно 1 раз по смещению 0x1730 от его начала.

Раз теперь нам известен ключ шифрования, мы можем просто применить операцию XOR к зашифрованному флагу и получить искомый флаг. И несмотря на то, что получившиеся первые 4 байта действительно похожи на начало флага, дальше мы получаем нечитаемые символы. Значит, внедрением шеллкода вмешательство в работу FlagReader.exe не ограничилось.

Теперь стоит поискать, кто (какой процесс) мог внедрить шеллкод в процесс FlagReader.exe. Для этого еще раз внимательно взглянем на дерево запущенных процессов:

```
Name                                 Pid   PPid   Thds   Hnds Time
-------------------------------- ------ ------ ------ ------ ----
 0x85df3528:wininit.exe              404    336      7     89 2020-12-08 18:00:30 UTC+0000
. 0x86046258:services.exe            504    404     16    223 2020-12-08 18:00:30 UTC+0000
.. 0x8612db18:svchost.exe            772    504     19    377 2020-12-08 18:00:31 UTC+0000
.. 0x861d3918:svchost.exe           1192    504     16    331 2020-12-08 18:00:32 UTC+0000
.. 0x861ab260:svchost.exe           1076    504     12    482 2020-12-08 18:00:31 UTC+0000
.. 0x8626c710:svchost.exe           1528    504     23    312 2020-12-08 18:00:32 UTC+0000
.. 0x8611a9a0:svchost.exe            720    504     10    266 2020-12-08 18:00:31 UTC+0000
.. 0x860df538:svchost.exe            624    504     15    363 2020-12-08 18:00:31 UTC+0000
.. 0x86190030:svchost.exe            936    504     44    782 2020-12-08 18:00:31 UTC+0000
.. 0x86168d40:svchost.exe            888    504     26    449 2020-12-08 18:00:31 UTC+0000
…
. 0x864ce3f0:FlagReader.exe         2516   1328      1      7 2020-12-08 18:01:11 UTC+0000
 0x861e0cf8:svchost.exe             2608   2580      1      8 2020-12-08 18:01:18 UTC+0000
```

Один из запущенных процессов svchost.exe явно выделяется, т.к. у него совершенно другой родительский процесс, и запущен он был явно позже, чем все остальные. Сдампим этот процесс с помощью команды volatility procdump и откроем получившийся файл в любом дизассемблере/декомпиляторе.

Уже в первых инструкциях этого файла мы видим, что он ищет запущенный процесс FlagReader.exe, а, значит, мы на правильном пути. Даная программа замаскировалась под системный процесс svchost.exe, используя технику Process Hollowing, в чем можно убедиться, воспользовавшись плагином для volatility hollowfind.

Далее, с помощь WinAPI функции WriteProcessMemory она внедряет шеллкод, прочитанный из файла C:\sc.bin, в память процесса FlagReader.exe. И в конце эта программа ищет зашифрованный флаг на стеке и меняет местами его dword'ы в определенном порядке.

Для получения флага остается лишь выполнить перестановки в зашифрованном флаге в обратном порядке и применить операцию XOR с уже известным ключом.

Ответ: CC{d0n7_cryp7_m3_m1st3r_7r0j4n}.

Задача II.1.3.2. Forks And Knives (736 баллов)

https://cyberchallenge.rt.ru/challenges#Forks%20And%20Knives

Один знакомый моего брата друга отца тещиной сестры сосед по комнате сказал, что в этом файле что-то спрятано. Но я ничего не нашел. Честно говоря, я даже не понял, что это...

Описание

Дан образ файловой системы, в которой спрятан флаг.

Решение

Сначала нужно понять, образ какой файловой системы (ФС) нам дан. Помочь в этом могут утилиты file, binwalk и/или любой hex-редактор. В данном случае мы имеем образ файловой системы APFS. Эта ФС сейчас используется на компьютерах Apple в связке с операционной системой macOS.

Удобнее всего работать с этой файловой системой именно в macOS, поэтому дальнейшее решение предполагает ее наличие. Тем не менее, задание решается и на Linux с использование пакета apfs-fuse, который также позволит замонтировать этот образ. Также в этом пакете имеется утилита apfs-dump, которая позволяет работать со внутренними структурами ФС.

Чтобы понять, куда двигаться дальше, нужно замонтировать данный образ. Сделать это можно командой:

```
% hdiutil attach fs.dmg
```

После этого в папке /Volumes/DiskDump/ окажется содержимое данного образа. Перемещаясь по файловой системе, можно заметить, что в домашней директории пользователя john (/Users/john/) есть файл с историй выполненных команд — .bash_history. Среди прочего там есть следующие команды:

```
cd  /Downloads/

chmod +x hide

./hide flag.txt /bin/zsh
```

Получается, что кто-то запустил файл с именем hide, который находился в папке Downloads (папка для скачиваемых файлов по умолчанию), передав ему через аргументы имя файлы flag.txt, который, по всей видимости, содержит флаг.

Заглянув в директорию Downloads на нашем образе ФС, мы не обнаружим там файлов hide и flag.txt, вероятно, они уже были удалены на момент снятия дампа ФС. Однако, там есть другой интересный файл — secret.txt. В нем есть некий текст, который, тем не менее, вряд ли поможет в решении. Но! Раз этот файл лежит в папке Downloads, вероятнее всего он был скачан из Интернета.

В macOS есть одна интересная особенность — при скачивании файла через любой браузер, ссылка на этот файл сохраняется в его же расширенных атрибутах. Расширенные атрибуты файлов можно посмотреть с помощью утилиты xattr:

```
% xattr -l Users/john/Downloads/secret.txt
```

Среди них нас интересует тот, который называется com.apple.metadata: kMDItem WhereFroms. Там мы видим следующую ссылку: https://yadi.sk/d/id6VzPUYcEq3ow?w=1.

Ссылка ведет на Яндекс.Диск, где кроме файла secret.txt находится и файл с именем hide. Теперь нужно его скачать и открыть в любом дизассемблере/декомпиляторе.

Проанализировав код файла, можно понять, что данная программа читает содержимое файла переданного через первый аргумент (flag.txt в нашем случае), и записывает его в ресурсный форк файла, переданного через второй аргумент (<второй_аргумент>/..namedfork/rsrc), предварительно сжав его с помощью zlib и зашифровав простейшим XOR'ом со значением 0xDEADFACE.

Остается только достать содержимое ресурсного форка файл /bin/zsh, обратившись к нему по пути /bin/zsh/..namedfork/rsrc, либо из расширенного атрибута этого файла с именем com.apple.ResourceFork. И, далее, применив к этим данным XOR со значением 0xDEADFACE и разжав их с помощью zlib, получим искомый флаг.

Ответ: CC{d0n7_h1d3_y0ur_l1gh7_und3r_4_bush3l}.

Категория PWN

Задания данной категории чаще всего нацелены на эксплуатацию бинарных уязвимостей в исполняемых файлах под различные архитектуры и ОС. По большей части задания затрагивают узявимости переполнения буфера (куча, стек), форматной строки, UAF, и другие повреждения памяти.

Задача II.1.4.1. verificator (885 баллов)

https://cyberchallenge.rt.ru/about

В интернете полным-полно статей на тему того, как детектировать изолированное окружение, так что у вас не должно возникнуть трудностей с решением этой задачи.

```
nc verificator.nti.2020.tasks.cyberchallenge.ru 20104
```

Описание

В данной задаче был дан исходный код программы на языке программирования C. Участнику предлагалось передать на вход программе шеллкод (иначе говоря — «полезную нагрузку») в форме кода для процессора на базе архитектуры x86_64. Код запускается в эмуляторе «unicorn», и в случае, если эмулятор не обнаружит ни одного системного вызова, то код исполнится внутри самой программы.

Решение

Для того, чтобы решить задачу, участникам было необходимо «обмануть» эмулятор и научиться отличать, когда их код исполняется внутри эмулятора, а когда в естественном окружении. Для этого можно было воспользоваться свойством эмулятора unicorn, который при старте программы зануляет все регистры. После проверки одного из незануленных регистров, например rcx, в случае если регистр равен нулю, то необходимости исполнять программу больше нет. Если регистр не равен нулю, значит, наш шеллкод исполняется уже не в изолированном окружении, и можно исполнить нужные нам для получения доступа на сервер системные вызовы и получить флаг.

Ответ: CC{congrats_with_your_first_step_in_emulation_detection}.

Задача II.1.4.2. alloc8or (885 баллов)

https://cyberchallenge.rt.ru/about

Современные аллокаторы содержат внутри себя множество различных защитных техник, но только не этот. Я думаю, что вы должны безо всякого труда проэксплуатировать уязвимость в данной программе.

```
nc alloc8or.nti.2020.tasks.cyberchallenge.ru 20102
```

Описание

В данной задаче был дан исходный код программы на языке программирования C, который содержал код основной программы в стиле «сохрани заметку», которая взаимодействовала с пользователем, а также вспомогательный код для управления памятью в основной программе.

Решение

Для решения задачи, необходимо было найти уязвимость выхода за границы буфера в функции «update» и проэксплуатировать ее. Для эксплуатации от участников требовалось прочитать код программы, которая осуществляет управление памятью, понять, что она уязвима к атаке класса «unlink», с помощью чего получить примитив записи и чтения в произвольную часть памяти программы через перезапись указателя struct storage_s *storage в глобальной памяти. Далее, необходимо было воспользоваться полученным примитивом и получить исполнение кода на системе через чтение и запись в секцию .got.

Ответ: CC{simplE_unl1nk_is_really_not_that_hard_as_it_seemed_yesterday}.

Категория Web

Задания данной категории предполагают поиск уязвимостей веб-приложения (веб-сайта) и дальнейшую их эксплуатацию. Для этого необходимо знать основы разработки веб-приложений, понимать базовые принципы их проектирования, иметь представление о том, как работает тот или иной веб-фреймворк или CMS. Кроме того, важно понимать природу возникновения веб-уязвимостей, понимать и уметь эксплуатировать типовые уязвимости в веб-приложениях. Список наиболее распространенных веб-уязвимостей периодически публикуется в рамках проекта OWASP Top 10 (urlhttps://owasp.org/www-project-top-ten/). Примеры используемых в решениях задач инструментов:

- браузер и различные расширения для него;
- Burp Suite;
- nmap;
- sqlmap;
- httpie.

Решения задач данной категории, как правило, начинаются с исследования данного веб-приложения: какие страницы есть на сайте, какой фреймворк или CMS были использованы и т. п. Далее нужно определить поверхность атаки, т. е. понять, как пользователь может влиять на работу веб-приложения. Иными словами, как он может передать данные и как они будут обработаны сайтом. Исходя из полученных данных, уже можно определить возможные проблемные места и проверить их на типовые (и не только) уязвимости.

Задача II.1.5.1. Hipsterverse / Secrets (505 баллов)

hipsterverse.nti.2020.tasks.cyberchallenge.ru

Пс, новые технологии не интересуют? Тут новый сервис нарисовался, на последнем ректе, сдобрен амплифаем и все это под соусом граф-ку-эль, посмотришь?

Слушай, забыл спросить, а как вообще в этом вашем граф-ку-эль делаются запросы?

Описание

Предоставленное веб-приложения является сервисом для заметок.

Аутентифицированный пользователь может оставлять как обычные, так и зашифрованные заметки. Такие заметки шифруются ключом, который пользователь сам вводит в браузере, он хранится только в браузере пользователя.

Неаутентифицированный пользователь может только создать учетную запись, а далее аутентифицироваться.

Решение

Чтобы начать взаимодействие с приложением пользователю нужно создать учетную запись. Сделать это можно, перейдя по ссылке Create account на главной странице.

При исследовании веб приложений часто используют средства разработчика в браузере. С их помощью можно посмотреть ошибки приложения, запросы к серверу и даже список исполняемых файлов.

Для начала, пользователю нужно повзаимодействовать с приложением, чтобы посмотреть, как приложение общается с сервером. Для этого нужно открыть средства разработчика, нажав клавишу `F12`, и выбрать вкладку `Network` в браузере `Chrome`.

После того как пользователь создаст заметку, он может заметить запрос к серверу по пути graphql, а это может значить, что приложение использует технологию `GraphQL`.

Если приложение использует `GraphQL`, значит, приложение поддерживает абстракции запросов (`query`) и мутаций (`mutations`).

Чтобы осуществить поиск по исходному коду, нужно в средствах разработчика перейти на вкладку `Sources`, нажать `Ctrl + P` и ввести `query`. Внутри найденного файла `query.ts` можно найти запрос `secretFlagQuery517470ab`.

Остается только перейти на вкладку `Network`, скопировать запрос к `/graphql` и изменить тело запроса на:

```
{"body": "{"query":"query MyQuery {secretFlagQuery517470ab}","variables":{}}"}
```

Так пользователь получит первый флаг CC{7f3e4ecd-a483-4f88-80f0-7fc523682f81}.

Ответ: CC{7f3e4ecd-a483-4f88-80f0-7fc523682f81}.

Задача II.1.5.2. Hipsterverse / Our notes (524 баллов)

hipsterverse.nti.2020.tasks.cyberchallenge.ru

Пс, новые технологии не интересуют? Тут новый сервис нарисовался, на последнем ректе, сдобрен амплифаем и все это под соусом граф-ку-эль, посмотришь?

Какие вот у всех секреты? Почему все что-то да скрывают? Пишут свои заметочки... Что они там пишут? Вот бы узнать, что там админ пишет (admin@cyberchallenge.rt.ru).

Описание

Предоставленное веб-приложения является сервисом для заметок.

Аутентифицированный пользователь может оставлять как обычные, так и зашифрованные заметки. Такие заметки шифруются ключом, который пользователь сам вводит в браузере, он хранится только в браузере пользователя.

Неаутентифицированный пользователь может только создать учетную запись, а далее аутентифицироваться.

Решение

Чтобы начать взаимодействие с приложением пользователю нужно создать учетную запись. Сделать это можно, перейдя по ссылке `Create` account на главной странице.

При исследовании веб приложений часто используют средства разработчика в браузере. С их помощью можно посмотреть ошибки приложения, запросы к серверу и даже список исполняемых файлов.

Для начала, пользователю нужно повзаимодействовать с приложением, чтобы посмотреть, как приложение общается с сервером. Для этого нужно открыть средства разработчика, нажав клавишу `F12`, и выбрать вкладку `Network` в браузере `Chrome`.

После того как пользователь создаст заметку, он может заметить запрос к серверу по пути `/graphql`, а это может значить, что приложение использует технологию `GraphQL`.

Если перезагрузить страницу и посмотреть все запросы к `/graphql`, то можно заметить запрос `ListTodos`, а также, что один из параметров запроса — фильтр по владельцу заметки.

Если повторить этот запрос, но удалить этот фильтр, то можно получить все заметки, созданные всеми пользователями в этом приложении.

В описании заметки есть поле владельца, но это только его идентификатор, сейчас нет возможности понять, какой пользователь действительно является админом.

Для того, чтобы это установить, нужно опять посмотреть файл `queries.ts`, и найти там запрос `listUsers`, результат которого даст все связи идентификатора и почты.

Таким образом можно понять какая заметка принадлежит какому пользователю и получить флаг CC{0cee90a0-f93e-4c09-8768-a2dea47ae330}.

В некоторых случаях, сервер можно возвращать не все ответы разом, но также добавлять идентификатор пагинации `nexToken`, который нужно добавить к последующему запросу, чтобы получить оставшиеся данные.

Ответ: CC{0cee90a0-f93e-4c09-8768-a2dea47ae330}.

Задача II.1.5.3. Hipsterverse / Our secured notes (885 баллов)

hipsterverse.nti.2020.tasks.cyberchallenge.ru

Пс, новые технологии не интересуют? Тут новый сервис нарисовался, на последнем ректе, сдобрен амплифаем и все это под соусом граф-ку-эль, посмотришь?

Какие вот у всех секреты? Почему все что-то да скрывают? Пишут свои заметочки... Что они там пишут? Вот бы узнать, что там админ пишет (admin@cyberchallenge.rt.ru).

Описание

Предоставленное веб-приложения является сервисом для заметок.

Аутентифицированный пользователь может оставлять как обычные, так и зашифрованные заметки. Такие заметки шифруются ключом, который пользователь сам вводит в браузере, он хранится только в браузере пользователя.

Неаутентифицированный пользователь может только создать учетную запись, а далее аутентифицироваться.

Решение

Найдя все заметки пользователя `admin@cyberchallenge.rt.ru`, можно заметить, что одна из них обычная заметка с открытым текстом, а другая заметка зашифрована.

Найдя файл `encryption.utils.ts`, можно заметить, что для шифрования используется шифр `AES` и библиотека `crypto-js`.

Полученную зашифрованную заметку можно попробовать расшифровать пытаясь угадать пароль. Сделать это можно, перебирая список популярных паролей.

Так можно получить следующий флаг CC{adaa822d-2b3f-4b71-ae26-63a5ebe9d0b9}.

Ответ: CC{adaa822d-2b3f-4b71-ae26-63a5ebe9d0b9}.

Задача II.1.5.4. Hipsterverse / Super Secret Refrigerator Frank (957 баллов)

hipsterverse.nti.2020.tasks.cyberchallenge.ru

Пс, новые технологии не интересуют? Тут новый сервис нарисовался, на последнем ректе, сдобрен амплифаем и все это под соусом граф-ку-эль, посмотришь?

У них же что-то было уже написано? Куда они все это дели? Я только авс вижу. Неужели они все переписали? Не, не может быть.

Описание

Предоставленное веб-приложения является сервисом для заметок.

Аутентифицированный пользователь может оставлять как обычные, так и зашифрованные заметки. Такие заметки шифруются ключом, который пользователь сам вводит в браузере, он хранится только в браузере пользователя.

Неаутентифицированный пользователь может только создать учетную запись, а далее аутентифицироваться.

Решение

Открыв вкладку `Network` в средствах разработчика, можно заметить еще один запрос к серверу `AuthProxy`. Запрос содержит в себе только один параметр `url`. Исхо-

дя из названия запроса, можно предположить, что север делает последующий запрос по переданному адресу.

Если попробовать поменять параметр `url` на любой другой сайт, сервер вернет ошибку с информацией о том, что действие запрещено.

Можно предположить, что проверка осуществляется регулярным выражением, но с такой проверкой есть известная проблема: если проверять адрес одним способом, а делать запрос другим, то итоговый адрес может отличаться между двумя проверками.

Обойти проверку регулярным выражением можно, например, вот так https://example.com?@analytics.internal.ru?action=AppLoaded.

Для того, чтобы получить запрос, можно использовать сервис https://requestbin.com/. В запросе от сервера:

```
/?@analytics.internal.ru?action=AppLoaded&flag=CC%7Bdde9b81e-942c
↪  -4ebf-b5bc-1df79c432742%7D
```

Можно найти флаг CC{dde9b81e-942c -4ebf-b5bc-1df79c432742}.

Ответ: CC{dde9b81e-942c -4ebf-b5bc-1df79c432742}.

Задача II.1.5.5. Christmas (756 баллов)

christmas.nti.2020.tasks.cyberchallenge.ru

Скоро Рождество, и мы запустили для вас сервис (http://christmas.nti.2020.tasks.cyberchallenge.ru/), где вы можете проверить, насколько хорошо вы вели себя в этом году.

P.S. У вас есть еще время исправиться.

Описание

Предоставленное веб-приложения является сервисом для проверки, хорошо ли себя вел ребенок, по его имени.

Единственная доступная операция — отправить имя на сервер через форму и получить в ответ, что ребенок хорошо себя вел (на экране будет подарок) или плохо (на экране будет уголек).

Решение

Для начала необходимо было найти исходный код сервиса, он состоял из одного файла и был доступен по ссылке http://christmas.nti.2020.tasks.cyberchallenge.ru/index.phps. Расширение `PHPS` (`PHP Source`) интерпретируется некоторыми веб-серверами как файл с исходным кодом, который необходимо отобразить с подсветкой синтаксиса без интерпретации.

Прочита исходный код сервиса можно было увидеть, что в сервисе происходит десериализация данных полученных от пользователя, что приводит к `POI`.

Если передать в куки user класс User со значением поля `$name` — класс `PS`, то в методе `good_name` `PS` будет приведен к строке (при вызове функции `hash`) и, соответственно, у него будет вызван метод `__toString`. Который в свое время вызовет метод `run`, в котором содержится уязвимость `shell injection`. Таким образом, в сервисе можно получить `RCE`, и далее любым удобным способом получить флаг, например, выполнив команду `cat /flag.txt`.

Ответ: CC{F457_chr157m45_ch3ck_f457_d357ruc7}.

Категория PPC

PPC (Professional programming and coding) — задачи на программирование или автоматизацию обработки большого количества данных. Примеры используемых в решениях задач инструментов:

- Python.
- Sublime Text.
- Notepad++.
- Vim.
- Emacs.

Задача II.1.6.1. LogQR (460 баллов)

На брендированных QR-кодах обычно очень маленькие логотипы, мы исправили это недоразумение.

Описание

В задаче был дан QR-код поверх которого был нанесен логотип Ростелеком. Так как логотип закрывал значительную часть QR-кода, считать код обычными распознавалками было нельзя.

Решение

Логотип был нанесен на QR-код таким образом, что мета-информация QR-кода и биты отвечающие за данные, которые хранились в коде, не были перекрыты. Чтобы

это понять, можно было воспользоваться статьей на википедии о том, как устроен QR-код. После этого можно было выписать битовое представление данных из QR-кода и получить флаг.

Ответ: CC{l0g0_c4nt_m3ss_up_QR}.

Задача II.1.6.2. Anamorphism (736 баллов)

Напишите на языке Haskell функцию anagram, которая находит все множества анаграмм, которые содержат в себе наибольшее количество слов.

Функция должна иметь следующий тип:

```
anagram :: [String] -> [[String]]
```

Пример:

```
anagram ["abc", "hehe", "foo", "cba", "eheh", "hhee", "oof", "ofo"]
```

```
[["eheh","hehe","hhee"],["foo","ofo","oof"]]
```

Входной список слов непустой, имеет длину не более 50000, а каждое слово имеет длину не более 25 символов.

Множества в выходном списке можно выдавать в любом порядке, как и слова внутри отдельных множеств.

Для вашего удобства мы импортировали Data.List, поэтому вы можете использовать все функции оттуда.

Сервис для сдачи решений:

```
nc anamorphism.nti.2020.tasks.cyberchallenge.ru 20101
```

В этой версии задачи вам нужно уместить код в 104 символов.

Описание

Задача на программирование на языке Haskell с ограничением на длину символов.

Решение

Одно из возможных решений, укладывающееся в ограничение и проходящее все тесты:

```
anagram=(\z->[snd<$>x|x<-z,l x==maximum(l<$>z)]).groupBy
((.fst).(==).fst).sort.(zip=<<map sort);l=length
```

Авторское решение этой задачи состоит из 62 символов. Пытливому читателю предлагается найти его самостоятельно.