SIBINTEK CTF 2025

Задания

09.11.2025

Название: DNK Delivery

Категория: Web

Очки: динамическое начисление

Описание: Обычные пути ведут к обычным результатам.
Что если пойти другим путём?

Флаг: Sibintek{D3L1V3RY_D0CUM3NT_ACCES5}

01

Разведка и анализ.
После запуска приложения и регистрации/входа в систему, открывается личный кабинет пользователя.

В личном кабинете доступны отчёты пользователя с ID 574, 575, 576. При попытке просмотреть отчёт, система перенаправляет на URL вида:

```
/delivery/report?id=574&auth_token=a1b2c3d4e5f6...
```

Из этого видно, что доступ к отчётам защищён токеном (`auth_token`), передаваемым в GET-параметре.

02

Анализ клиентского кода.

Открываем DevTools (F12) и переходим во вкладку **Sources**. Изучаем подключённые JS-файлы:

Открываем DevTools (F12) и переходим во вкладку **Sources**. Изучаем подключённые JS-файлы:

1. `/static/js/delivery-utils.js` — утилиты для работы с доставками
2. `/static/js/main.js` — основной скрипт

В файле `delivery-utils.js` (или в HTML-коде страницы) обнаруживаем:

```
function generateDeliveryAccessToken(reportId, userId) {
  var tokenData = String(reportId) + DELIVERY_SECURITY_KEY +
String(userId);
  return md5(tokenData);
}

function getReportLink(reportId, userId) {
  var token = generateDeliveryAccessToken(reportId, userId);
  return '/delivery/report?id=' + reportId + '&auth_token=' +
token;
}
```

Ключевые выводы:
- Секретный ключ `DELIVERY_SECURITY_KEY` раскрыт в клиентском коде
- Токен генерируется по формуле: `MD5(reportId + DELIVERY_SECURITY_KEY + userId)`
- Функции доступны глобально через `window.generateDeliveryAccessToken` и `window.getReportLink`

В консоли DevTools вызываем функцию для текущего пользователя (user_id = 2):

```
generateDeliveryAccessToken(574, 2)
// Результат: "a1b2c3d4e5f6..." (совпадает с токеном в URL)
```

Токен успешно воспроизводится, что подтверждает механизм генерации.

03

Поиск скрытого отчёта.
При входе под обычным пользователем видны отчёты с ID 574, 575, 576. Логично предположить, что могут существовать отчёты с меньшими ID (например, 1, 2, 3...).

Попробуем вручную сгенерировать токен для отчёта с **ID = 1**, используя текущий user_id = 2:

```
generateDeliveryAccessToken(1, 2)
// Результат: "f8e9a7b6c5d4..."
```

Переходим по ссылке:

```
/delivery/report?id=1&auth_token=f8e9a7b6c5d4...
```

Ответ сервера: `403 Forbidden – Invalid access token`

Токен не подошёл. Это означает, что для отчёта **с ID=1** используется другой алгоритм валидации или другой **user_id**.

Скорее всего 1 отчет будет сделан админом, для доступа к админ-отчёту (id=1) требуется токен, сгенерированный с user_id=1 (ID администратора), а не текущего пользователя.

04

Эксплуатация уязвимости.
Возвращаемся в консоль DevTools и генерируем токен для отчёта с **ID = 1** и **user_id = 1**:

```
generateDeliveryAccessToken(1, 1)
```

Результат (пример):

```
7c8f3d9e2a4b1f0e5c6d8a9b3f2e1d0c
```

Формируем финальный URL:

```
/delivery/report?id=1&auth_token=7c8f3d9e2a4b1f0e5c6d8a9b3f2e1d0c
```

Переходим по ссылке (можно ввести в адресную строку или выполнить в консоли). Ответ сервера: `Успешное отображение страницы отчёта`

Название: DNK War

Категория: Stego

Очки: динамическое начисление

Описание: В компании ДНК Сибирь сотрудники отдела информационной безопасности заметили подозрительную активность - некоторые работники обмениваются странными файлами через корпоративную сеть. Один из таких файлов был перехвачен системой мониторинга, но открыть его не удалось.

Специалисты ИБ подозревают, что сотрудники используют стеганографию для сокрытия конфиденциальной информации. Ваша задача - выяснить, что именно скрывается внутри.

Флаг: SIBINTEK{THE_B3ST_G4M3}

01

Анализ файла в HEX-редакторе.
Получаем файл без расширения, который не открывается стандартными программами. Первым делом открываем файл в HEX-редакторе (например, HxD, 010 Editor или xxd в Linux).

```
$ xxd task
00000000: 0000 0000 0000 0000 d095 d189 d0b5 20d0  .............. .
00000010: bed0 b4d0 bdd0 b020 d0ba d0b0 d180 d182  ....... ........
00000020: d0b0 0014 9c00 0001 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 6b65 793d 434f 4f4c 4b45 5900 0000 0000  key=COOLKEY.....
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

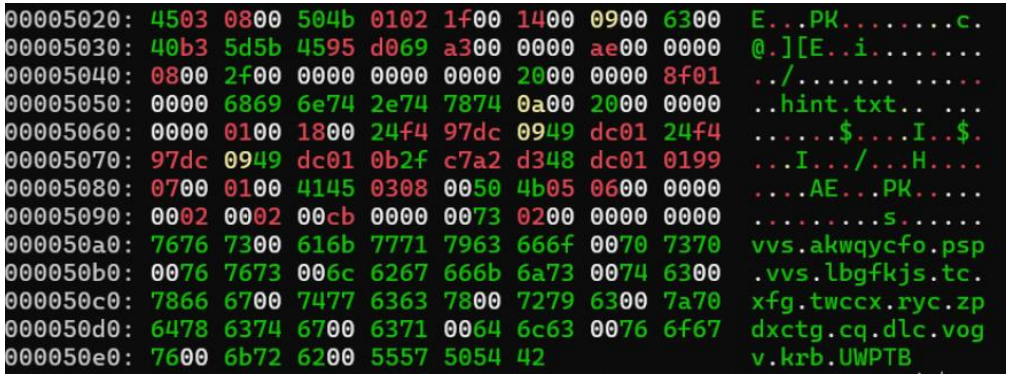
При анализе начала файла обнаруживаем:

- Отсутствуют стандартные сигнатуры файлов
- В начале файла присутствует строка: `key=COOLKEY`

Это указывает на то, что:
1. Сигнатуры файла намеренно удалены
2. Где-то в файле присутствует зашифрованное сообщение
3. Ключ `COOLKEY` может быть использован для дешифрования

02

Поиск зашифрованного сообщения.
Прокручиваем HEX редактор в самый конец файла.



В конце файла обнаруживаем необычный текст:

```
vvs akwqycfo psp vvs lbgfkjs tc xfg twccx ryc zpdxctg cq dlc
vogv krb UWPTB
```

Анализируя этот текст, замечаем:
- Текст состоит из читаемых символов
- Присутствуют повторяющиеся паттерны
- Структура предложения сохранена

Пробуем стандартные шифры, подходящие под эти критерии, при этом не забываем про наличие у нас ключа. Приходим к тому, что это шифра Виженера по его характерным признакам.

03

Дешифрование шифра Виженера.
Используем найденный ранее ключ `COOLKEY` для дешифрования сообщения. Можно использовать онлайн-инструменты (например, dcode.fr) или написать скрипт на Python.

Пример кода для дешифрования:

```
def vigenere_decrypt(ciphertext, key):
    result = []
    key = key.upper()
    key_length = len(key)
    key_index = 0

    for char in ciphertext:
        if char.isalpha():
            shift = ord(key[key_index % key_length]) - ord('A')
            if char.isupper():
                decrypted_char = chr((ord(char) - ord('A') -
shift) % 26 + ord('A'))
            else:
                decrypted_char = chr((ord(char) - ord('a') -
shift) % 26 + ord('a'))
            result.append(decrypted_char)
            key_index += 1
        else:
            result.append(char)

    return ''.join(result)

ciphertext = "Xvs tcggaqzr tzk hvs oknvqis qg hvs tqkgh xaq
psxxskg at hvs xogy obr GQPQK"
key = "COOLKEY"
plaintext = vigenere_decrypt(ciphertext, key)
print(plaintext)
```

После дешифрования получаем:

```
The password for the archive is the first two letters of the
task and SIBIR
```

Расшифрованное сообщение подсказывает нам пароль для архива:
- Первые две буквы названия таска: **DN**
- Плюс слово: **SIBIR**
- Итоговый пароль: **DNSIBIR**

04

Определение типа файла (полиглот).
Теперь понимаем, что файл является полиглотом - файлом, который может быть открыт как несколько разных типов файлов одновременно.

Для того чтобы убедиться, проходимся binwalk, видим следующий фрагмент:

```
$ binwalk task
...
19787         0x4D4B          Zip archive data, encrypted at
least v2.0 to extract, compressed size: 333, uncompressed size:
606, name: crypto.py
20186         0x4EDA          Zip archive data, encrypted at
least v2.0 to extract, compressed size: 163, uncompressed size:
174, name: hint.txt
20617         0x5089          End of Zip archive, footer length:
22
```

Пробуем открыть файл как ZIP архив:

```
unzip task
```

Или переименовываем файл:

```
cp task task.zip
```

При попытке извлечения запрашивается пароль. Вводим полученный пароль: **DNSIBIR**

05

Извлечение содержимого архива.
После ввода правильного пароля архив успешно распаковывается.

Внутри архива обнаруживаем два файла:
1. `crypto.py` - скрипт с зашифрованными данными
2. `hint.txt` - файл с подсказкой

Содержимое `crypto.py`:

```python
def chaos_encrypt(text, key):
    state = key
    result = []

    for i, char in enumerate(text):
        state = ((state << 1) ^ ((state >> 7) & 1) ^ ((state >>
5) & 1)) & 0xFF
        sbox_val = custom_sbox(ord(char))
        xored = sbox_val ^ state
        bit_perm = bit_permute(xored, i)
        final = bit_perm ^ (i * 7 % 256)
        result.append(f"{final:02x}")

    return ''.join(result)


def custom_sbox(byte):
    return ((byte ^ 0x63) * 13 + 47) % 256


def bit_permute(byte, pos):
    shift = (pos % 3) + 1
    return ((byte << shift) | (byte >> (8 - shift))) & 0xFF
```

06

Расшифровка подсказки.
Анализируем `crypto.py` и видим, что это скрипт шифрования. Нам нужно написать обратные функции для дешифрования.

Создаем файл `decoder.py` с обратными операциями:

```python
def inverse_sbox(byte):
    return (((byte - 47) % 256) * 197 % 256) ^ 0x63


def inverse_bit_permute(byte, pos):
    shift = (pos % 3) + 1
    return ((byte >> shift) | (byte << (8 - shift))) & 0xFF


def chaos_decrypt(hex_string, key):
    state = key
    result = []

    hex_bytes = [hex_string[i:i + 2] for i in range(0, len(hex_string),
2)]

    for i, hex_byte in enumerate(hex_bytes):
        state = ((state << 1) ^ ((state >> 7) & 1) ^ ((state >> 5) &
1)) & 0xFF
        encrypted = int(hex_byte, 16)
        after_xor = encrypted ^ (i * 7 % 256)
        after_perm = inverse_bit_permute(after_xor, i)
        after_state = after_perm ^ state
        original = inverse_sbox(after_state)
        result.append(chr(original))

    return ''.join(result)


key = ord('D')
encrypted =
"b319040df04d844ed9dbcb56ef6ad2df821dda9fa212ca0a2e9602e57e5d9db29b88ee
e919a35d438a291a0932c310e48b32448405b07b"
decrypted = chaos_decrypt(encrypted, key)
print(decrypted)
```

Запускаем декодер:

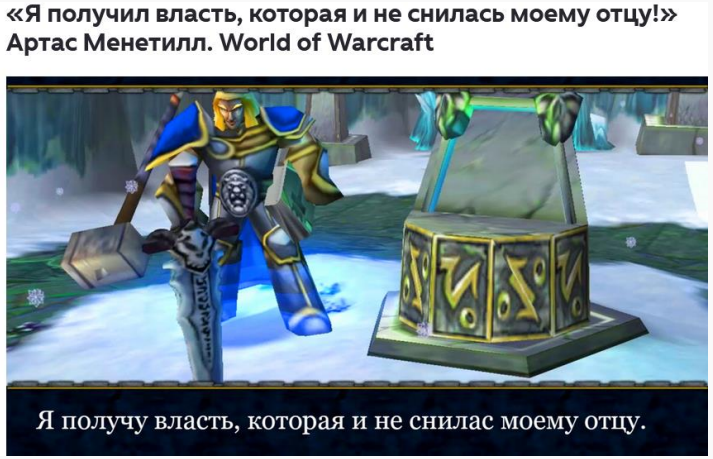
```
python3 decoder.py
```

Получаем расшифрованное сообщение:

```
I gained power that my father could never have imagined
```

Эта фраза является цитатой из игры Warcraft III.
Её произносит Артас Менетил после того, как становится Королём-личом.

Найти информацию об этом можно, например, https://dzen.ru/a/XW6KfMflDACt0JGS



Подсказка указывает на то, что нужно работать с файлом карты Warcraft III.

07

Восстановление сигнатур файла.
Теперь понимаем, что исходный файл `task` - это карта Warcraft III (.w3x файл) со стертыми сигнатурами.

Стандартная сигнатура .w3x файла (карты Warcraft III):

```
HEX: 48 4D 33 57
ASCII: HM3W
```

Открываем файл в HEX редакторе и восстанавливаем сигнатуру в начале файла:

Было:

```
00 00 00 00 ...
```

Стало:

```
48 4D 33 57 ...
```

Таже у файлов Blizzard имеется цифровая подпись в виде второй сигнатуры:

```
HEX: 4D 50 51
ASCII: MPQ
```

Заменяем таким же образом.
Сохраняем файл с правильным расширением: `map.w3x`

08

Открытие карты в редакторе Warcraft III.
Для открытия карты необходим редактор карт Warcraft III (World Editor). Если у вас установлена игра, редактор находится в папке с игрой.

Открываем восстановленный файл `map.w3x` в World Editor:
1. Запускаем World Editor
2. File → Open
3. Выбираем файл `map.w3x`

09

Поиск флага на карте.
После открытия карты в редакторе переходим в режим просмотра местности (Terrain Editor).

На карте обнаруживаем текст, выложенный объектами или написанный на местности:

```
SIBINTEK{THE_B3ST_G4M3}
```

Это и есть искомый флаг.