SIBINTEK CTF 2025

Задания

09.11.2025

Название: OilBot

Категория: Misc

Очки: динамическое начисление

Описание: Между кнопками и меню — щель в логике. Проскользни и забери флаг.

Флаг: sibintek{c@1lb@ck_1nj3ct10n_m@5t3r}

01

Разведка и анализ.
После запуска бота через `/start` пользователю открывается мини-игра про нефть и её переработку.

Изучив основные функции, пользователь может найти кнопку **Секретные разработки**. При нажатии на неё появляется уведомление: «Эта кнопка недоступна для вас на данный момент».

Исследовав эту кнопку, можно увидеть, что её `callback_data` содержит интересные данные:

```
callback_data = buy:corp:flag:active:false
```

В данных есть параметры, разделенные двоеточием. Параметр `flag` указывает на то, что с этой кнопкой связано получение флага. Также имеется параметр `active:false`, который указывает на "активность" флага, делаем вывод, что нужно каким-то образом изменить данный параметр на `active:true`.

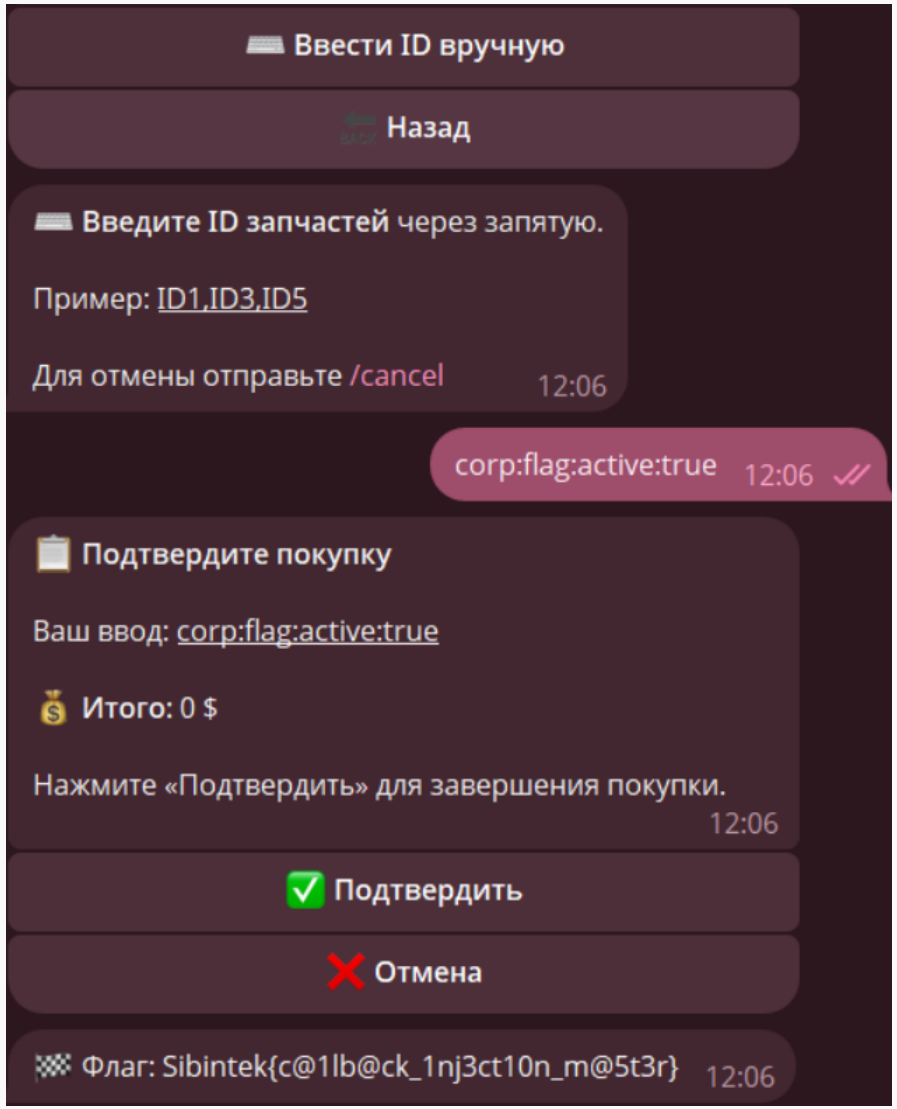
При изучении всех функций бота пользователь может заметить, что при поломке оборудования (событие, которое вызывается случайно) или при отказе от обслуживания в меню **Офис** → **Сервисный центр**, ему предлагают ввести ID комплектующих вручную при их покупке.

Попробовав ввести случайные данные, можно обнаружить, что они попадают в `callback_data` кнопки ✅ **Подтвердить** (кнопка для подтверждения покупки) в формате `buy:<ввод_пользователя>.`

Следовательно, в кнопку ✅ **Подтвердить** можно встроить любую информацию и проверки нет. Можно заметить, что начало `callback_data` у кнопок **Секретные разработки** и ✅ **Подтвердить** одинаковое. Значит, можно попытаться вписать `corp:flag:active:false`, чтобы их `callback_data` стали идентичными, и нажать на ✅ **Подтвердить**.
В результате пользователь получит то же уведомление, что и при нажатии на кнопку **Секретные разработки**.

Далее, попробовав проверить бота на использование динамических параметров из `callback_data`, можно вписать `corp:flag:active:true` и, нажав на кнопку ✅ **Подтвердить**, пользователь получит флаг.

Флаг успешно получен.



Название: Pump Console

Категория: Revrerse

Очки: динамическое начисление

Описание: На технологической площадке возник нестабильный режим работы насосной станции. Операторская консоль визуализирует состояние трёх помп, трубопровода и резервуара: уровень, поток, давление и температуру. Доступ к административному контуру заблокирован политиками доступа и встроенными проверками целостности окружения.

При корректной последовательности действий и соответствии рабочих параметров системе может быть сформирован аварийный пропуск, позволяющий выполнить безопасное переключение режима. Консоль при этом не раскрывает внутренние токены в интерфейсе и не подсказывает прямых шагов — все необходимые сведения содержатся в самом приложении.

Флаг: Sibintek{AES_is_my_fr1end}

01

Быстрый обзор кода и точка входа к «секрету».

- Главное окно обрабатывается функцией `WndProc = sub_140004A90`.

- Кнопка Admin (`WM_COMMAND`, `wParam == '1' / 0x6C`) ведёт в ветку, где либо показывается «Access Denied», либо вызывается генератор аварийного «пропуска» — функция `sub_140002810`.

- В `sub_140002810` формируются плейн-данные и копируются в новую страницу памяти через `VirtualAlloc` → `memcpy`. Это идеальная точка для перехвата.

Возникает следующая идея для решения: пропатчить четыре/пять условных переходов `jnz`, которые ведут нас к «Access Denied», чтобы всегда доходить до вызова `sub_140002810`, а уже там поставить брейкпоинт около `VirtualAlloc` и считать готовые данные (флаг/токен) из памяти.

02

Патчим проверки в обработчике Admin.

Работаем в `sub_140004A90` (адреса указаны как Virtual Address при базе образа `0x140000000`; при ASLR (Address Space Layout Randomization) используйте соответствующие Relative Virtual Address).

Глушим ветки, ведущие к отказу:

1. Антидебаг-денай (две длинные `jnz` на `loc_1400050A7`):
   - `0x140005012`: `jnz loc_1400050A7`
   - `0x140005020`: `jnz loc_1400050A7`
2. Политика окружения (два коротких `jnz` на `loc_140005042` — промежуточная ветка, собирающая «плохие» условия):
   - `0x140005034`: `jnz short loc_140005042`
   - `0x14000503C`: `jnz short loc_140005042`
3. Финальный отказ по policy:
   - `0x14000505A`: `jnz short loc_140005093` → **NOP x2**

После этих патчей выполнение при нажатии **Admin** всегда идёт сюда:

```
0x14000505C  call sub_140002810
0x140005061  log("Admin session: volatile keystore prepared (RAM-only)")
```

03

Ставим брейкпоинт там, где появляется плейн.
Целевая функция — `sub_140002810`.

Критический участок (VA):

```
``
...                 ; завершён XOR/потоковая дешифрация, в RDI —
временный буфер плейна
0x140002B6B  call    VirtualAlloc
0x140002B71  mov     [rsp+...], rax      ; RAX = адрес новой страницы
(DST)
0x140002B7E  mov     rdx, rdi            ; RDX = временный буфер (SRC)
0x140002B84  call    memcpy              ; КОПИРУЕМ ПЛЕЙН НА СТРАНИЦУ
0x140002B89  mov     rax, [rsp+...]      ; RAX = та самая страница с
плейном
```

Как ловить:

- Поставить BreakPoint на `loc_7FF6C7752B57`.
  При остановке:
- `RAX` указывает на начало страницы с уже готовым ASCII;
- в Dump по `RAX` переключаемся в ANSI/ASCII и считываем флаг/токен.
- Либо поставить BP на импорт `VirtualAlloc` и после возврата «шагнуть» до `0x140002B89`.

04

Автодамп флага без брейкпоинтов.

После того как мы пропатчили `Admin` и нажали кнопку, плейн уже лежит в адресном пространстве процесса. Можно вообще не отлаживать: открыть процесс, пройти по его коммит-областям и найти ASCII-строку, начинающуюся с `Sibintek{`` и заканчивающуюся ``}`

```
#include <windows.h>
#include <psapi.h>
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
bool FindAscii(const std::vector<BYTE>& buf, size_t& off, std::string&
out) {
    const char* pat = "Sibintek{";
    for (size_t i = 0; i + 8 < buf.size(); ++i) {
        if (memcmp(&buf[i], pat, 8) == 0) {
            // дочитаем до '}' или до не-печатаемого
            size_t j = i; while (j < buf.size() && buf[j] >= 0x20 &&
buf[j] <= 0x7E) { if (buf[j] == '}') { out.assign((char*)&buf[i], j - i
+ 1); off = i; return true; } j++; }
        }
    }
    return false;
}
```

```
int main() {
    DWORD pid;
    printf("Enter target PID: ");
    if (scanf_s("%lu", &pid) != 1 || pid == 0) {
        printf("Invalid PID\n");
        return 1;
    }

    HANDLE ph = OpenProcess(PROCESS_VM_READ | PROCESS_QUERY_INFORMATION, FALSE,
pid);
    if (!ph) {
        DWORD err = GetLastError();
        printf("OpenProcess failed (%lu)\n", err);
        if (err == 87) printf(" → PID does not exist or invalid parameter\n");
        if (err == 5)  printf(" → Access denied (try running as admin)\n");
        return 2;
    }

    printf("Successfully opened process %lu\n", pid);
    if (!ph) { printf("OpenProcess failed (%lu)\n", GetLastError()); return 2;
}

    SYSTEM_INFO si; GetSystemInfo(&si);
    BYTE* addr = (BYTE*)si.lpMinimumApplicationAddress;
    BYTE* end = (BYTE*)si.lpMaximumApplicationAddress;

    MEMORY_BASIC_INFORMATION mbi;
    std::vector<BYTE> buf;
    while (addr < end) {
        if (VirtualQueryEx(ph, addr, &mbi, sizeof(mbi)) != sizeof(mbi)) { addr
+= 0x1000; continue; }
        bool readable = (mbi.State == MEM_COMMIT) && !(mbi.Protect &
PAGE_GUARD) &&
            !(mbi.Protect & PAGE_NOACCESS);
        if (readable) {
            SIZE_T sz = mbi.RegionSize;
            buf.resize(sz);
            SIZE_T rd = 0;
            if (ReadProcessMemory(ph, addr, buf.data(), sz, &rd) && rd > 0) {
                size_t off; std::string a; std::wstring w;
                if (FindAscii(buf, off, a)) { printf("[ASCII] %p : %s\n", addr
+ off, a.c_str()); break; }

            }
        }
        addr += mbi.RegionSize;
    }
    CloseHandle(ph);
    return 0;
}
```

05

Практическая ценность при анализе ПО.

Задача повторяет типичный сценарий реверса GUI-валидаторов и «индустриальных панелей»:

- пропатчить небольшой набор ветвлений анти-отладки/политик,
- - найти место материализации секрета (выделение/копирование/дешифр),
- - поставить точный runtime-брейкпоинт и извлечь секрет без изучения всех крипто-деталей.

Те же приёмы применимы к DRM-обвязкам, офлайн-лицензаторам, диагностическим тулзам со «скрытым» режимом и инженерным HMI-консолям.