

# ClipboardFusion crash dump analysis

Deep analysis of ClipboardFusion.exe.11752.dmp — call stack, root cause, and the code change required to stop this class of crash.

### Fatal finding

ClipboardFusion 6.3.99.107 died with a native access violation (0xC0000005) inside the .NET 10 runtime while copying a clipboard bitmap. The process then fail-fast-ed with System.ExecutionEngineException (HRESULT 0x80131506). The crashing managed function is **Q75.e7s.GetClipboardDataBitmap**, invoked from the “show clipboard preview popup” hotkey path. The defect is that CF\_DIB / CF\_DIBV5 HGLOBAL handles are passed to Marshal.Copy without GlobalLock. Marshal.Copy treats the value as a raw pointer and memcpy’s GlobalSize() bytes from it. That AV occurs inside a CoreCLR QCall, which is fatal and cannot be caught by the existing try/catch.

## 1. Incident identity

|                       |                                                                       |
|-----------------------|-----------------------------------------------------------------------|
| Dump file             | D:\ClipboardFusion.exe.11752.dmp (11,937,139 bytes, Windows minidump) |
| Dump written          | 17 September 2026, 21:28:16 local                                     |
| WER Application Error | 17 September 2026, 21:28:08 local (Event ID 1000)                     |
| .NET Runtime event    | 17 September 2026, 21:28:08 local (Event ID 1023)                     |
| Process ID            | 11752 (0x2DE8) — matches the dump filename                            |
| Process start         | 17 September 2026, 21:17:24 local (~11 minutes uptime)                |
| CPU time at dump      | 13 s user / 11 s kernel                                               |
| Crashing thread       | OS thread 16120 (0x3EF8), SOS thread 7, STA UI thread                 |
| Product               | ClipboardFusion 6.3.99.107 (pre-release / 99.x build)                 |
| Binary                | C:\Program Files\ClipboardFusion\ClipboardFusion.exe                  |
| Managed assembly      | ClipboardFusion.dll (obfuscated WinForms app, TFM net10.0)            |

## 2. Environment

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| OS                | Windows 11 (NT 10.0), build 26200, workstation, AMD64                    |
| CPU               | 24 logical processors, family 6 model 0xC6 (Intel-class)                 |
| .NET runtime      | Microsoft.NETCore.App 10.0.12 (CoreCLR 10.0.1226.42308)                  |
| Desktop framework | Microsoft.WindowsDesktop.App 10.0.12                                     |
| coreclr.dll       | C:\Program Files\dotnet\shared\Microsoft.NETCore.App\10.0.12\coreclr.dll |
| coreclr commit    | 95017c711e6afc1085133d440e42b4bd78155701                                 |
| KERNELBASE.dll    | 10.0.26100.9444                                                          |
| GC at crash       | Workstation GC, 1 heap, ~50 MB allocated / ~54 MB committed              |

|                 |                                         |
|-----------------|-----------------------------------------|
| Managed threads | 53 CLR threads (25 background, 27 dead) |
|-----------------|-----------------------------------------|

### 3. Exception record (authoritative)

The minidump ExceptionStream is the source of truth for the original fault. The thread CONTEXT saved in the dump is later, after CoreCLR had already started fail-fast processing.

| Field            | Value              | Meaning                                                               |
|------------------|--------------------|-----------------------------------------------------------------------|
| ExceptionCode    | 0xC0000005         | EXCEPTION_ACCESS_VIOLATION — invalid memory read or write             |
| ExceptionFlags   | 0x00000001         | EXCEPTION_NONCONTINUABLE (fatal; not a catchable first-chance AV)     |
| ExceptionAddress | 0x00007FFED3EA2EB1 | coreclr.dll + 0x1A2EB1 (base 0x00007FFED3D00000)                      |
| Param[0]         | 0                  | Read violation (attempted to read, not write)                         |
| Param[1]         | 0x000001B8C0FA8408 | The virtual address that could not be read (unmapped / not committed) |
| ThreadId         | 0x3EF8 (16120)     | UI thread running the WinForms message loop                           |
| CLR wrapper      | 0x80131506         | COR_E_EXECUTIONENGINE — System.ExecutionEngineException               |

Table 1. Minidump exception stream and matching Windows Error Reporting / .NET Runtime events.

Windows Error Reporting recorded the same facts independently:

```

Application Error (ID 1000)
  Faulting application : ClipboardFusion.exe 6.3.99.107
  Faulting module      : coreclr.dll 10.0.1226.42308
  Exception code       : 0xc0000005
  Fault offset        : 0x00000000001a2eb1
  Faulting process id  : 0x2DE8 (11752)

.NET Runtime (ID 1023)
  Application: ClipboardFusion.exe
  CoreCLR Version: 10.0.1226.42308
  .NET Version: 10.0.12
  Description: The process was terminated due to an internal error in the
  .NET Runtime at IP 0x00007FFED3EA2EB1 (0x00007FFED3D00000)
  with exit code 0x80131506.

```

### 4. What the thread was doing when it died

At dump time the crashing thread's RIP is KERNELBASE!RaiseFailFastException (KERNELBASE + 0x10A1A8, 0x188 bytes into that export). That is the *aftermath*, not the original fault. CoreCLR converts an AV inside a QCall/FCall into a fail-fast ExecutionEngineException; WER then snapshots the process. The original AV address in the exception record is still inside coreclr.dll, in the native implementation behind System.Buffer.MemmoveInternal.

The managed stack on OS thread 0x3EF8 is complete and unambiguous. Frames are listed innermost-first. Obfuscated type names (Q75.e7s, Kej.Me0, ...) are from the shipped binary; several method names were left intact by the obfuscator and are used below.

#### 4.1 Crashing managed call stack

| # | Module                 | Function                                                        | Role                                      |
|---|------------------------|-----------------------------------------------------------------|-------------------------------------------|
| 0 | coreclr.dll            | Buffer.MemmoveInternal (QCall) @ coreclr+0x1A2EB1               | Native memcpy of clipboard bytes; AV here |
| 1 | System.Private.CoreLib | Buffer.MemmoveInternal(ref byte, ref byte, nuint) Buffer.cs:132 | Pins dest/src then calls the QCall        |
| 2 | ClipboardFusion.dll    | Q75.e7s.GetClipboardDataBitmap(IntPtr, bool, string) +483       | Reads CF_DIB / CF_DIBV5 via Marshal.Copy  |

| #  | Module               | Function                                                    | Role                                                 |
|----|----------------------|-------------------------------------------------------------|------------------------------------------------------|
| 3  | ClipboardFusion.dll  | Kej.Me0.re3()                                               | Builds the preview content (image path)              |
| 4  | ClipboardFusion.dll  | kEt8.VEth.BeginInvokeCustomVoid(CHpw)                       | Marshals work onto the UI thread                     |
| 5  | ClipboardFusion.dll  | Kej.Me0.ShowClipboardContentsPopup()                        | Shows the on-screen clipboard preview popup          |
| 6  | ClipboardFusion.dll  | Kej.Me0.ProcessHotKeyShowPreviewPopup(object)               | Hotkey handler for “show preview popup”              |
| 7  | ClipboardFusion.dll  | WEIL.fEIQ.WndProcMessageReceivedAfterMessageHandlers        | Custom WndProc after CF message handlers             |
| 8  | ClipboardFusion.dll  | kEt8.VEth.WndProc(ref Message)                              | Application window procedure                         |
| 9  | System.Windows.Forms | Control.ControlNativeWindow.WndProc / NativeWindow.Callback | WinForms native window thunk                         |
| 10 | user32 / WinForms    | DispatchMessage → Application.Run(Form)                     | STA UI message loop                                  |
| 11 | ClipboardFusion.dll  | sNH.zNp.Main / StartWithNewThreadAsync                      | App startup; this is the UI thread started from Main |

Table 2. SOS clrstack on the crashing thread. Frame 2 is the product function that must be fixed.

Trigger in product terms: the user (or a bound hotkey) asked ClipboardFusion to show the clipboard-contents preview popup while the clipboard held a bitmap (CF\_DIB, CF\_DIBV5, or a CF\_BITMAP that Windows would synthesize as a DIB). The UI thread entered GetClipboardDataBitmap and never returned.

## 4.2 Matching SOS exception object

```
Exception object : 00000198b7c00178
Exception type   : System.ExecutionEngineException
Message         : <none>
HResult          : 80131506
_xcode          : 0xE0434352 (CLR exception wrapper)
StackTrace       : <none> (fail-fast; no managed catch frame ran)
```

Thread 0 (the debugger’s current thread, OS 0xD10) is unrelated: it is blocked in Task.Wait inside obfuscated Main, and SOS reports a leftover System.BadImageFormatException (“Bad method token”) on that thread. Several other threads show the same BadImageFormatException. That is consistent with control-flow / anti-tamper obfuscation leaving invalid metadata tokens in the dump, not with the crash itself. The thread that actually fail-fasted is 0x3EF8, and its exception is ExecutionEngineException.

## 5. Root cause

### 5.1 The defective sequence

GetClipboardDataBitmap was decompiled from the installed ClipboardFusion.dll. Control-flow flattening and string encryption are present, but the clipboard API sequence is intact. Reconstructed, the DIB path is:

```
// Q75.e7s.GetClipboardDataBitmap (decompiled from 6.3.99.107)
// Tries CF_DIB, then CF_DIBV5, then falls back to CF_BITMAP.

IntPtr hGlobal = GetClipboardData(CF_DIB); // or CF_DIBV5
if (hGlobal != IntPtr.Zero)
{
    uint size = GlobalSize(hGlobal); // size of the MOVEABLE block
    if (size != 0)
    {
        byte[] dib = new byte[size];
        Marshal.Copy(hGlobal, dib, 0, dib.Length); // BUG: hGlobal is not a pointer
        var img = FromBitmapDib(dib);
        if (img != null) return img;
    }
}
// fallback: GetClipboardData(CF_BITMAP) → FromHBitmap(...)
```

GetClipboardData(CF\_DIB / CF\_DIBV5) returns an **HGLOBAL** allocated with **GMEM\_MOVEABLE** (the documented clipboard contract: a memory object containing **BITMAPINFO** followed by bits). **GlobalSize(hGlobal)** is the correct API for that handle. **Marshal.Copy(IntPtr source, byte[] dest, int index, int length)** is **not**. It treats source as a raw address and performs a byte memmove of length bytes from that address into the managed array.

The same class already does this correctly for text. **GetClipboardDataText** calls **GlobalLock**, reads via **Marshal.PtrToStringUni / PtrToStringAnsi**, and **GlobalUnlocks** in a finally block. The bitmap path skipped that lock. That is an inconsistency in the same helper type, not a runtime bug.

## 5.2 Why this becomes a fatal AV in coreclr.dll

On .NET 10, **Marshal.Copy** of a large byte count bottoms out in **System.Buffer.Memmove** → **Buffer.MemmoveInternal** (**Buffer.cs** line 132), which pins both references and **QCalls** into **coreclr** (**Buffer\_MemMove / equivalent memcopy**). An access violation inside that **QCall** is treated as a corrupted runtime state: **CoreCLR** raises **ExecutionEngineException** and calls **KERNELBASE!RaiseFailFastException**. The try/catch in **GetClipboardDataBitmap** and the one in **ProcessHotKeyShowPreviewPopup** never run.

What the memcopy actually over-reads depends on the **HGLOBAL** flavour Windows handed back:

- **GMEM\_MOVEABLE (normal clipboard DIB):** the **HANDLE** is not the data pointer. Copying **GlobalSize()** bytes from the handle value reads a heap header / handle entry and then walks off the committed region. Small images may appear to work if adjacent slack happens to be readable; large images (screenshots, high-DPI, multi-monitor) reliably AV. The faulting address **0x1B8C0FA8408** sits in a gap between loaded modules in the **0x1B8...** native heap range — exactly this pattern.
- **Synthesized CF\_DIB from CF\_BITMAP:** Windows can manufacture a **DIB HGLOBAL** on **GetClipboardData**. The size can be several tens of megabytes for a desktop screenshot. That matches “crashes when the clipboard holds a large image”.
- **Corrupt or delay-rendered data:** **GlobalSize** can report a huge length that is not backed by committed pages. **new byte[size]** may succeed; the subsequent memcopy will not.

### Why the existing catch blocks are useless here

**AccessViolationException** from unmanaged memcopy inside a **QCall** is not a normal managed exception on modern .NET. It is promoted to **ExecutionEngineException** and fail-fasts the process (exit **0x80131506**). You cannot “log and continue”. The only fix is to never hand **Marshal.Copy** an address that is not a locked, size-checked pointer.

## 5.3 Call-stack walk, frame by frame

- **ProcessHotKeyShowPreviewPopup** — user-visible entry. Decompilation shows it only calls **ShowClipboardContentsPopup()** inside a try/catch that logs via **AE9v.gE9U**. That catch cannot intercept a fail-fast.
- **ShowClipboardContentsPopup** → **re3** — builds the popup. **re3** is the image-loading helper that calls **GetClipboardDataBitmap**. It was invoked via **BeginInvokeCustomVoid**, i.e. on the UI thread, which is correct for **OpenClipboard** but means a crash here kills the whole app.
- **GetClipboardDataBitmap** — opens the clipboard (disposable **GPtE** wrapper around **OpenClipboard/CloseClipboard**), iterates **CF\_DIB** then **CF\_DIBV5**, **GlobalSize + Marshal.Copy**, then **FromBitmapDib**. Crash IP is +483 bytes into the jitted method, which is the **Marshal.Copy / memmove** site, not **FromBitmapDib** and not the **CF\_BITMAP** fallback.
- **Buffer.MemmoveInternal** — **CoreLib** source at tag **v10.0.12**, line 132: fixed (byte\* pDest = &dest) fixed (byte\* pSrc = &src) **MemmoveInternal(pDest, pSrc, len)**. The native half lives in **coreclr.dll** at the recorded **ExceptionAddress**.
- **RaiseFailFastException** — dump-time **RIP**. Confirms the runtime had already decided the process was unrecoverable.

## 6. Related defect in the same helper

TryCopyDataFromClipboard in the same type has the identical Marshal.Copy-from-HGLOBAL pattern (no GlobalLock). GetClipboardFormatSize uses GlobalSize correctly to measure, which is fine; copying is the dangerous half. Any code path that pulls raw clipboard bytes (images, custom formats, files) through that helper can fail-fast the process the same way.

```
// Q75.e7s.TryCopyDataFromClipboard - same bug, different caller
foreach (var item in q7e(formats)) // item.Item1 is HGLOBAL
{
    if (item.Item1 == IntPtr.Zero || GlobalSize(item.Item1) < data.Length)
        continue;
    Marshal.Copy(item.Item1, data, 0, data.Length); // again, no GlobalLock
    CloseClipboard();
    return true;
}
```

## 7. What to fix in the product code

### Primary fix — lock, bound, copy, unlock

Treat CF\_DIB / CF\_DIBV5 values as HGLOBALs, never as pointers. Mirror the already-correct GetClipboardDataText pattern. Cap the copy. Prefer parsing the BITMAPINFOHEADER (40 or 124 bytes) first and copying min(header-derived size, GlobalSize). Never call Marshal.Copy on a value you have not locked and VirtualQuery'd.

### 7.1 Recommended implementation (unobfuscated)

```
const uint MaxDibBytes = 64u * 1024u * 1024u; // tune; reject absurd DIBs

static bool TryCopyHGlobal(IntPtr hGlobal, out byte[] bytes)
{
    bytes = null;
    if (hGlobal == IntPtr.Zero) return false;

    uint size = GlobalSize(hGlobal);
    if (size == 0 || size > MaxDibBytes) return false;

    IntPtr p = GlobalLock(hGlobal);
    if (p == IntPtr.Zero) return false;
    try
    {
        // Optional but strongly recommended: VirtualQuery(p) must report
        // MEM_COMMIT covering at least 'size' bytes with PAGE_READONLY or better.
        bytes = new byte[checked((int)size)];
        Marshal.Copy(p, bytes, 0, bytes.Length); // p is a real pointer now
        return true;
    }
    finally
    {
        GlobalUnlock(hGlobal);
    }
}

// In GetClipboardDataBitmap, replace the DIB branch with:
IntPtr h = GetClipboardData(format); // CF_DIB or CF_DIBV5
if (TryCopyHGlobal(h, out var dib))
{
    var img = FromBitmapDib(dib); // keep existing parser, but add bounds checks
    if (img != null) return img;
}
```

### 7.2 Hardening that should ship with the lock

- **Size cap.** Refuse GlobalSize above a documented limit (for example 64 MB). A 30000×30000×32 DIB header can ask for gigabytes; new byte[size] plus memcpy will either OOM or AV. Fail the preview, do not take down the process.
- **Header-first parse in FromBitmapDib.** Read biSize, biWidth, biHeight, biBitCount, biCompression, biSizeImage, biClrUsed. Reject biSize not in {40, 108, 124}, reject non-positive dimensions, compute stride as ((width \* bitCount +

31) / 32) \* 4, and require header + color-table + bits ≤ buffer.Length. Do not trust biSizeImage when it is 0 or larger than the remaining buffer (BI\_RGB commonly leaves it 0).

- **VirtualQuery before memcpy.** Marshal.Copy cannot be caught if the region is not committed. A 20-line VirtualQuery loop is cheaper than a process restart.
- **Do not use CF\_BITMAP as an HGLOBAL.** The existing FromHBitmap fallback is the right API for CF\_BITMAP (it is an HBITMAP / GDI object). Never GlobalSize / GlobalLock / Marshal.Copy a CF\_BITMAP handle.
- **Keep OpenClipboard for the whole copy.** The GPtE using-block already does this. Do not CloseClipboard until after GlobalUnlock. Do not copy on a background thread without its own OpenClipboard.
- **Fix TryCopyDataFromClipboard the same way.** Any raw-format copy is the same landmine.
- **Prefer IDataObject where possible.** OleGetClipboard + GetData(CF\_DIB) or Clipboard.GetImage() go through OLE and avoid a class of handle vs pointer mistakes. Keep the native path only where you need formats WinForms will not give you.
- **Do not wrap Marshal.Copy in try/catch and call it done.** After this bug, that catch is a false sense of safety. The process still dies.

### 7.3 Suggested regression tests

- Preview popup hotkey with CF\_DIB of a small bitmap (smoke).
- Preview popup with a full-desktop screenshot at 150% / 200% DPI and multi-monitor virtual desktop (the size that historically trips this).
- Preview popup with CF\_DIBV5, CF\_BITMAP only, PNG ("PNG" / Format17), and an empty clipboard.
- Preview popup with a malformed DIB: biSizeImage = 0xFFFFFFFF, biWidth = int.MaxValue, truncated HGLOBAL smaller than the header claims.
- Delay-rendered clipboard owner (SetClipboardData(format, NULL) + WM\_RENDERFORMAT) that supplies a MOVEABLE DIB.
- Concurrent clipboard changes from another process while the popup is opening (OpenClipboard should serialize this; assert no crash).
- The same matrix through TryCopyDataFromClipboard / history capture, not only the preview popup.

## 8. What this is not

- **Not a coreclr.dll bug.** The runtime is memcpy'ing the pointer and length it was given. The bad pointer/length pair is produced by ClipboardFusion.dll.
- **Not a WinForms / WebView2 / Skia crash.** Those modules are loaded (preview UI almost certainly uses them after a successful bitmap load) but they are not on the crashing stack.
- **Not the BadImageFormatException ("Bad method token") seen on other threads.** That tracks with the obfuscator. The crashing thread's exception object is ExecutionEngineException.
- **Not a GC heap corruption that happened earlier and was only detected here** as the primary story. The AV is a read of 0x1B8C0FA8408 during an explicit memmove of clipboard bytes. Secondary heap damage is possible if this copy ever wrote into the GC heap with a bad dest, but dest is a fresh byte[] and param[0] is a read.

## 9. Analysis method and limits

- Windows minidump parsed with Python minidump (ExceptionStream, modules, thread CONTEXT) plus SOS via dotnet-dump 10.0.745401 (clrthreads, pe, clrstack -a, ip2md, eeheap, dumpobj).
- Managed code decompiled with ILSpy 11.0 from C:\Program Files\ClipboardFusion\ClipboardFusion.dll 6.3.99.107. Names are obfuscated; control-flow flattening is present. The GetClipboardData / GlobalSize / Marshal.Copy sequence is not in doubt.

- coreclr.dll function name at +0x1A2EB1 was not resolved against private PDBs (Microsoft symbol GUID 07FB51EC-FD57-48BE-9DB9-32401E0670CD, age 1). The managed frame plus Buffer.cs:132 plus the QCall contract are sufficient to identify it as the native memmove behind MemmoveInternal.
- The dump is a minidump (MemoryListStream, ~12 MB), not a full dump. Clipboard HGLOBAL contents and the destination byte[] payload were not fully resident, so the exact DIB dimensions were not recovered. The invalid-read address and the source function do not depend on that.
- No product source tree was available. Recommendations are against the decompiled 6.3.99.107 binary and public Win32 / .NET contracts.

## 10. Priority and expected payoff

This is a user-facing hard crash of a tray application, on a hotkey that exists specifically to inspect whatever is currently on the clipboard — including images. It will reproduce whenever the preview (or any other caller of GetClipboardDataBitmap / TryCopyDataFromClipboard) meets a MOVEABLE DIB whose GlobalSize is larger than the readable span of the unlocked handle. Shipping GlobalLock + a size cap + header bounds checks should eliminate this fail-fast class. Until that ships, a practical workaround is to unbind the “show clipboard contents preview” hotkey and avoid image preview if crashes continue.

## Appendix A — notable loaded modules

| Base               | Module                       | Notes                                    |
|--------------------|------------------------------|------------------------------------------|
| 0x00007FF604470000 | ClipboardFusion.exe          | Native host, 6.3.99.107                  |
| 0x000001B8B9590000 | ClipboardFusion.dll          | Managed product code (crash site)        |
| 0x00007FFED3D00000 | coreclr.dll 10.0.12          | AV at +0x1A2EB1                          |
| 0x00007FFED2BA0000 | clrjit.dll                   | JIT; not on crash stack                  |
| 0x00007FFF848C0000 | KERNELBASE.dll               | Dump-time RIP = RaiseFailFastException   |
| 0x00007FFF85120000 | user32.dll                   | GetClipboardData / message loop          |
| 0x00007FFF49580000 | gdiplus.dll                  | Image decode after a successful copy     |
| 0x000001B8B7080000 | SkiaSharp.dll + libSkiaSharp | UI rendering; not on crash stack         |
| 0x000001B8BED50000 | WebView2                     | Settings / online UI; not on crash stack |
| 0x00000198B5830000 | Sentry.dll                   | Crash reporting present in the process   |

Table 3. Subset of ModuleListStream entries relevant to the crash.

## Appendix B — reconstructed user-visible sequence

- ClipboardFusion is running in the tray (started 21:17:24).
- Something places a bitmap on the clipboard (or a format Windows will promote to CF\_DIB).
- At 21:28:08 the show-preview-popup hotkey is processed on the UI thread.
- GetClipboardDataBitmap opens the clipboard, obtains a CF\_DIB or CF\_DIBV5 HGLOBAL, allocates a managed buffer of GlobalSize bytes, and Marshal.Copy's from the unlocked handle.
- coreclr memcpy reads 0x1B8C0FA8408 and AVs.
- The runtime fail-fasts. WER writes ClipboardFusion.exe.11752.dmp.

## Appendix C — code owners: files / symbols to change

In the unobfuscated source tree these shipped names should map approximately to:

- **Q75.e7s** — native clipboard helper (GetClipboardDataBitmap, GetClipboardDataText, TryCopyDataFromClipboard, Open/CloseClipboard P/Invokes). This is the file that must change.
- **Kej.Me0** — main application object: ProcessHotKeyShowPreviewPopup, ShowClipboardContentsPopup, WndProc integration.
- **h1v.FromBitmapDib / FromHBitmap** — DIB / HBITMAP parsers. Add bounds checks in FromBitmapDib even after the lock is fixed.
- **GPtE** — OpenClipboard IDisposable. Keep using it; do not copy after dispose.

### Bottom line for engineering

Crash function: Q75.e7s.GetClipboardDataBitmap. Crash instruction: native memmove in coreclr while servicing Marshal.Copy. Cause: CF\_DIB HGLOBAL used as a pointer. Fix: GlobalLock + size cap + header validation; apply the same pattern to TryCopyDataFromClipboard. Do not rely on try/catch around the copy.