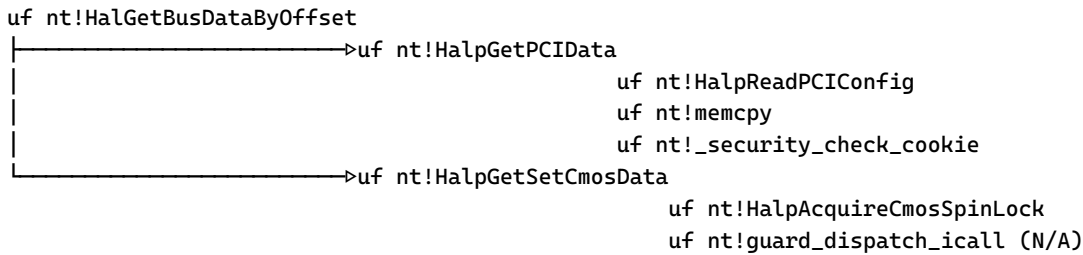


Disassemble Memory File

Memory.DMP can be used to determine call graphs, outside the scope of analysis tools like kd.exe. With a full disassembly, the context around a particular symbol is conveniently explored.



Blueprint

The tool must display the callers or callees for a given symbol. In overwhelming cases, the symbol is a function. The symbol can be a microinstruction, belonging to a function body, or it can be a global variable, in which case the containing bodies are identified, all dependencies are displayed.

The tool must work offline. It can be migrated to an USB and used on-premise, where support cases do not allow internet connection.

The tool must track a collection of memory files that are disassembled, for various OSs. A symbol is located in a precise index, specified in command line.

The tool must reuse the disassembly file, when displaying the tree.

The tool must generate the disassembly once. If the process takes a long time, the tool must show a warning. The tool must store metadata about disassembly to allow statistics to be collected.

The tool must perform fast, even if confined to commodity systems.

The tool can show collateral dependencies, like import functions or function pointers used as arguments.

1. `UfSymbol.ps1` renders the call graph based on a disassembly file. The file is generated once, reused at rendering stage. The disassembly is separated into individual function bodies. The root body contains the symbol requested by the user. A dependency graph contains the callers or the callees for each function. CLI switches determine the depth of the tree, target OS for rendering.
 - Generic functions have many callers; ie. 1118 matches for `nt!KeBugCheckEx` at `-Depth 1`.
2. Known functions are not disassembled: they can be minute like `KeYieldProcessorEx`, `ExAllocatePool2` or familiar as `IoCompleteRequest`, `atol`.

Sample output renders the call tree for `nt!KiSystemStartup`.

```
PS > (Measure-Command {
    $Image = 'D:\DataLake\2025-04-28\MEMORY.DMP'
    & '.\UfSymbol.ps1' -Symbol nt!KiSystemStartup -Image $Image -Depth 4 -Down | Out-Default
}).TotalSeconds
```

File "D:\DataLake\2025-01-28\MEMORY.DMP" of 1194.36 Mb has been processed in 4570 seconds.

D:\Processing\53c6f2af-38db-4219-9f41-f794c7897f5a\53c6f2af-38db-4219-9f41-f794c7897f5a.disassembly

D:\Processing\53c6f2af-38db-4219-9f41-f794c7897f5a\53c6f2af-38db-4219-9f41-f794c7897f5a.meta

D:\Processing\53c6f2af-38db-4219-9f41-f794c7897f5a\53c6f2af-38db-4219-9f41-f794c7897f5a.retpoline

3. The disassembly stage can take a long time. The 1st line shows a comparison with a previously decompiled *memory* file. The decompilation is executed on all cores but 1. Besides the `.disassembly` file, `.meta` and `.retpoline` are created. The `.meta` file contains:
 - *OS* and *computer* where the BSOD occurred.
 - *image* path and *hash*. The hash identifies duplicates, resulting in a decompilation bypass.
 - *system* where disassembly took place, *number* of CPUs allotted, *CPU model*, *duration* and *image size*.
 - The default modules used to disassemble the image:
 - for a `.dmp` file *nt*, *pci*, *acpi* and *hal* functions are disassembled
 - *base name* for all others.

4. The `.retpoline` file is an indirection table for bodies compiled with `/guard:cf`. Wherever `call nt!guard_dispatch_icall` is found, the source pointer is resolved in the memory file and displayed.

For `nt!KiSystemStartup` call tree:

- 1302 callees are identified with `-Depth 4`, 5318 at depth 6.
- Complete decompilation and identification takes **5215** seconds on an “Intel(R) Core(TM) i3-7100U CPU @ 2.40GHz” with 3 CPUs.

```
uf nt!KdInitSystem
├──uf nt!KeQueryPerformanceCounter
│   ├──uf nt!HalpTimerGetInternalData
│   └──uf nt!HalpTimerScaleCounter
│       ├──uf nt!ExAllocatePool2
│       ├──uf nt!_security_check_cookie
│       └──uf nt!MmGetPagedPoolCommitPointer
└──uf nt!KdRegisterDebuggerDataBlock
...
uf nt!PpmUpdatePerformanceFeedback
uf nt!guard_dispatch_icall (nt!_security_cookie
    nt!HalpOriginalPerformanceCounter
    nt!HalpPrivateDispatchTable+0x1b0=nt!HalpProcessorPrepareForIdle
    nt!HalpPrivateDispatchTable+0x1c0=nt!HalpProcessorResumeFromIdle
    nt!HalpTimerReferencePage
    nt!HalpPrivateDispatchTable+0x418=nt!HalpLbrResumeRecording
    nt!HalpPrivateDispatchTable+0x2f8=nt!HalpTimerClockStop
    nt!PopCsConsumption+0x140)
```

5215.506918

5. With `.\UfSymbol.ps1 -Setup`, a text based guide is launched. Configuration options are set in `UfSymbol.json` file:
 - directory where disassemblies are stored - internally called `$Database`.
 - symbol path
 - time limit for decompilation warning. Where `.meta` file previously generated has a duration larger than the limit, then the file and duration are printed.
 - statistics deactivation like duration, system, CPU model, file size from future `.meta` files.
6. `.\UfSymbol.ps1 -List OS | Complete` lists the `.meta` files in table form.

| computer | os basename ↑ | image |
|----------|---------------------------------------|---|
| INHOUSE1 | dbgeng 10.0.26100.2454 | C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\dbgeng.c |
| INHOUSE1 | Windows 11 Enterprise 22000 | D:\DataLake\2025-01-28\MEMORY.DMP |
| DEPLOY1 | Windows 10 Enterprise LTSC 2019 17763 | C:\Windows\Memory.DMP |
| DEPLOY2 | Windows 10 Pro 22631 | D:\DataLake\2025-04-28\MEMORY.DMP |

7. `UfSymbol.ps1` can be copied/migrated to an USB drive. The local database is rendered in place.
8. **powershell Core** is required given the performance benefits in the interpreter engine. Inbox **Desktop 5.1** has bottlenecks.
9. **Hotpaths** are moved to inflight `CSharp` assembly. Decompilation is **8 times** faster.

Notes

- Decompilation-ready processing is useful in support cases where the `Memory.DMP` file cannot be provided. Implementation differences between OS versions are also visible.

- A `.dmp` file contains the dependencies from all modules, can trip the decompiler with inappropriate function bodies. This shortcoming does not apply to user mode.
- An executable solves all functions, cannot solve dependencies.
- Initially, the tool's objective was GUI rendering through SVG. With broad trees being prevalent, a point-and-click is deemed impractical.
- **Complete Memory Dump** contains more functions compared to **Kernel Memory**.
- Decompile through `kd.exe` can be superseded by `dbgeng.dll` COM interfaces. Direct access to `dbgeng.h` gives control to the process: trimming of function bodies occurs ad hoc. Parallel `kd.exe` execution binds trimming to disassembly completion.
 - Speed-up with `kd.exe` is memory file dependent. `IDebugControl::Execute` is not interruptible, `IDebugOutputCallbacks::Output` must retrieve the entire text before it is validated.
 - `IDebugControl::WaitForEvent` fails when clients are created by multiple threads.
error message:
Can't set dump file contexts
MachineInfo::SetContext failed - Thread: 000001A2CDA07900 Handle: 1 Id: 1 - Error == 0x8000FFFF
 - `IDebugControl::Execute` is serialized with a *critical section*.
disassembly:
0:017> k
Child-SP RetAddr Call Site
00 00000035'6ed8d300 00007ffa'976b15e0 dbgeng!DebugClient::ExecuteWide+0x23
01 00000035'6ed8d350 00007ffa'37eccdc5 dbgeng!DebugClient::Execute+0xf0

0:000> uf dbgeng!DebugClient::ExecuteWide

00000001'80101bca 488d0dafa97a00 lea rcx,[dbgeng!g_EngineLock (00000001'808b0580)]
00000001'80101bd1 48ff15b0e65600 call qword ptr [dbgeng!_imp_EnterCriticalSection (00000001'80670288)]

00000001'80101c06 e8990ffdfc call dbgeng!PushOutCtl (00000001'800d2ba4)
00000001'80101c23 e8e8f2ffff call dbgeng!Execute (00000001'80100f10)
00000001'80101c2f e81807fdff call dbgeng!PopOutCtl (00000001'800d234c)
00000001'80101c45 e896c2fcff call dbgeng!FlushCallbacks (00000001'800cdee0)

00000001'80101c50 488d0d29e97a00 lea rcx,[dbgeng!g_EngineLock (00000001'808b0580)]
00000001'80101c57 48ff1512e65600 call qword ptr [dbgeng!_imp_LeaveCriticalSection (00000001'80670270)]
 - The compiler uses long strings to decorate `C++` methods. The `uf` command uses the functions' address.
- Inbox `dbgeng.dll` version `10.0.19041.3636` identifies fewer functions compared with `10.0.26100.2454`.
- Where (N/A) appears in rendering:
 - indirection table has no corresponding target symbol - ie. register is used.
rax ← qword ptr [rcx+20h]:
uf nt!IoCsqRemoveIrp
fffff803'2c9d0980 48895c2410 mov qword ptr [rsp+10h],rbx
fffff803'2c9d0985 4889742418 mov qword ptr [rsp+18h],rsi
fffff803'2c9d098a 57 push rdi
fffff803'2c9d098b 4883ec20 sub rsp,20h
fffff803'2c9d098f 488b4120 mov rax,qword ptr [rcx+20h]
fffff803'2c9d0993 488bf2 mov rsi,rdx
fffff803'2c9d0996 4883613800 and qword ptr [rcx+38h],0
fffff803'2c9d099b 488d542430 lea rdx,[rsp+30h]
fffff803'2c9d09a0 488bd9 mov rbx,rcx
fffff803'2c9d09a3 c644243000 mov byte ptr [rsp+30h],0
fffff803'2c9d09a8 e833f70400 call nt!guard_dispatch_icall (fffff803'2ca200e0)
 - function body is missing either due to absent module, or a large body has been decompiled and trimmed.
- `.retpoline` build is not parallelized. Only $2E+3$ *poi* sources have to be decoded.

```
PS > $prefix = "https://raw.githubusercontent.com/armaber/scripts/refs/heads/disasm/";
"functions.ps1", "UfSymbol.ps1" | foreach {
    Invoke-WebRequest $prefix/DisassembleImage/$PSItem -OutFile $PSItem;
}
Get-Help .\UfSymbol.ps1 -Full;
```