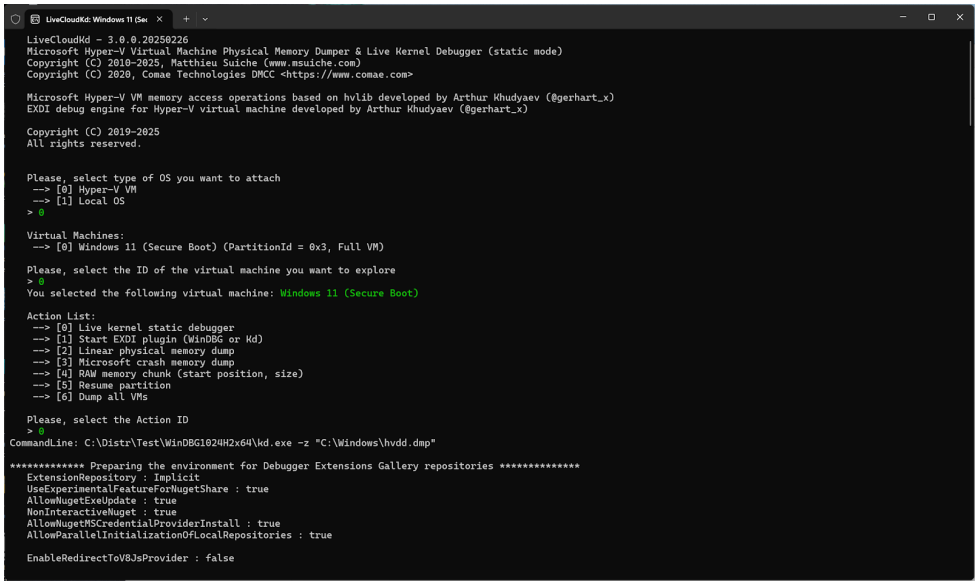
Hyper-V Internals

hvinternals.blogspot.com/2025/08/hyper-v-utility-livecloudkd-evolution.html



Hyper-V utility. LiveCloudKd: evolution and architecture technical analysis

«LiveCloudKd: Pioneering Virtual Machine Introspection for Memory Forensics Back in 2010, I released LiveCloudKd—a forensics tool that was the first of its kind to enable Virtual Machine introspection for Hyper-V v1 environments. Mark Russinovich and Ken Johnson were impressed enough after my presentation at BlueHat in Seattle that they incorporated similar functionality into LiveKd 5.0 shortly afterward. What made my approach particularly clever was that I implemented it entirely in user-mode by exploiting several architectural quirks in the Hyper-V vmwp.exe process. I discovered that the process lacked proper isolation and—even more astonishing—Memory Block handles were actually raw kernel-mode pointers rather than proper opaque handle values. While Microsoft hadn't officially documented vmwp.exe, vid.dll, or the drivers winhv.sys and vid.sys, I managed to leverage leaked headers from Microsoft's Singularity research project to reverse engineer the virtualization interface. Through careful analysis and strategic use of the Vid.dll APIs, I created a bridge between user-mode tools and the previously inaccessible memory space of virtual machines, fundamentally changing how we approach virtualization forensics and memory analysis in hypervisor environments».

Matt Suiche (Mar 6, 2025)

Big thanks Matt (https://www.x.com/msuiche) for introduction!

This article is about the functions of LiveCloudKd, which is now version 3.0 and features some new additions, including support for virtual machines on new Windows Preview OSes and memory reading and writing on local OS. After that version was issued, I decided to describe it, because logically utility was finished, but can be improved of course on others architectures or, probably, platforms.

Technically, the current version of LiveCloudKd looks similar as LiveKd 5.0 (https://learn.microsoft.com/en-us/archive/blogs/markrussinovich/livekd-for-virtual-machine-debugging and https://media.blackhat.com/bh-dc-11/Suiche/BlackHat_DC_2011_Suiche_Cloud_Pocket-wp.pdf), but with some improvements. It is open source: https://github.com/msuiche/LiveCloudKd and fork - https://github.com/gerhart01/LiveCloudKd, has improved performance and includes memory write functions (including writing options to local OS memory). These functions are available under administrative mode (thanks Joseph Bialek (https://x.com/JosephBialek or @josephbialek@infosec.exchange) for advice). Later I asked MSRC, if the local writing mode being enabled (https://x.com/gerhart_x/status/1888665136413515996), but there was no response from them. Therefore, I think it is not so important whether this function is enabled or disabled.

Active development starts again in 2018, and currently is added some small features and bugfixes. Many features have been added over time, but only the most thoroughly tested ones have been included in the public version. Features such as container support, the WHPX library, and nested virtualization work in limited mode (only certain OS versions have been tested), so they haven't been added to the public version, but the source code of prototypes is available at https://github.com/gerhart01/LiveCloudKd/blob/master/hvmm/hvmm/vid.c#L253.

Also, considering that Microsoft copies key part of the LiveCloudKd debugger algorithm for securekernel debugging (https://x.com/gerhart_x/status/1915104209948791211. There was discussion about this, but it finished very fast https://x.com/gerhart_x/status/1947173362448380058), sharing of those functions becomes unsafe and likely will not be done in later future. Microsoft has Microsoft Defender antivirus, therefore they can decompile all packed and protected software. They are really dangerous in that case and they like Copilot 😉 (who doesn't know – it is means "Copy a lot").

Initially, Microsoft blocked the ability to work directly with Hyper-V virtual machines (https://x.com/gerhart_x/status/1014249371695828992), leaving access only at the kernel mode level or through injection into the protected vmwp.exe process, but only for Microsoft-signed libraries. I had to develop a separate vidaux.dll library, which is injects into the vmwp.exe process (it is not particularly safe, owing to live migration operations for virtual machines at a minimum) and uses vid.dll functions through named pipes; however, currently only kernel mode functionality remains. You can still find old LiveCloudKd builds with vidaux.dll support—they still contain the old LiveCloudKdSdk.dll library and can worked with Windows Server 2012 and 2016. Some of its code is available in the Bin2Dmp source code (https://github.com/MagnetForensics/Bin2Dmp/tree/master/Bin2Dmp). On Windows Server 2012, the old mode with vid.dll still works without a driver, if anyone is still using it. At one point, there were certain problems related to changes in the dbgeng.dll library, but currently the latest versions of WinDBG and kd.exe are supported with hooks, as well as the EXDi interface.

LiveCloudKd enables you to connect to a Hyper-V guest virtual machine using Microsoft Debugging Tools kd.exe or WinDBG and allows you to examine the kernel of a running Hyper-V virtual machine without needing to enable debug mode and disable Secure Boot on either the guest or host. You can execute debugger commands, inspect memory, and modify it.

It is particularly useful for examining the internal structures of a virtual machine, whether you are troubleshooting, researching, or conducting memory forensics. You can dump the physical memory of a virtual machine into a raw file or a complete crash dump, which is useful for subsequent analysis. It supports a range of Windows versions, such as Windows 10, Windows 11, and Windows Server 2016 through 2025.

Evolution from 2010 to 2025

LiveCloudKd development spans fifteen years of continuous adaptation to evolving Windows and Hyper-V architectures. The initial 2010 release followed research on Hyper-V v1 (Windows Server 2008, Windows Server 2008 R2 – 2025 already is v2), implementing an entirely user-mode solution, that exploited vmwp.exe design flaws. This groundbreaking work led to Microsoft inviting Matt Suiche to speak at BlueHat Security Briefings and subsequently influenced LiveKd 5.0 development.

The period from 2012 to 2016 saw continued development for integrating vid.dll API functions, including

VidDmMemoryBlockQueryTopology

VidQueryMemoryBlockMbpCount

VidReadMemoryBlockPageRange

The tool employed brute-force enumeration of vmwp.exe memory space to collect kernel pointers, with each virtual machine having one vmwp.exe process containing one partition handle and multiple memory block handles.

Period from 2018 to 2024 contains migration of functionality in kernel mode, migrate some core functionality to hvlib.dll library (Hyper-V memory manager library), including results of Hyper-V memory researches (prototypes of different Hyper-V memory types), Hyper-V scheduler research, that can possible to create dynamic LiveCloudKd debugging for Windows kernel and securekernel (interesting, that securekernel.exe is part of Hyper-V, if you see cabs_HyperV-OptionalFeature-VirtualMachinePlatform-Client-Disabled-FOD-Package~31bf3856ad364e35~amd64~~.cab archive from Windows Insider Preview distributive), improvements with WinDBG integration, added WinDBG with modern UI integration.

Recent developments include the version 3.x (2024–2025), which features are Windows Server 2025 support, enhanced EXDi plugin options, and improved Windows securekernel address space viewing. Local memory interface support was added also.

The current version runs on x64 Windows, ARM64 version still needs to change disassembler engine, because distorm3 engine doesn’t support ARM64.

And finally, one uses it by running it on the Hyper-V host alongside WinDBG, where it uses Hyper-V memory manager library for Hyper-V guest OS memory access. It offers options for static Hyper-V guest OS kernel debugging (eq standard dump viewing), producing memory dumps, and connecting via an EXDi plugin for more advanced debugging techniques. Compared to LiveKd, it is significantly faster—approximately 1,000 times quicker at reading memory—and permits writing to the Hyper-V virtual machine's memory. Additionally, whilst certain specific Hyper-V virtual machine configurations may cause BSODs, the driver is sufficiently stable (but additional stress tests still won't interfere).

The utility has the following capabilities:

1. Attaching to a virtual machine or the local OS using WinDBG, kd, or WinDBG with modern UI, either natively or via the EXDi plugin to access the NT kernel or secure kernel.
2. Dumping Hyper-V virtual machines to raw dump or Microsoft dump format.
3. Dumping local memory to raw dump or Microsoft dump format.
4. Editing Hyper-V virtual machine memory or local memory using WinDBG, kd, or WinDBG with modern UI.
5. Attaching kd, WinDBG, or WinDBG with modern UI directly to Hyper-V with guest Windows or local virtual machines, or launch the EXDi plugin for these purposes.
6. Dumping partition memory blocks from Windows or other virtual machines running inside Hyper-V.

Btw, what AI systems think about LiveCloudKd? It’s also actual now. Answer from one:

«Virtualization-Based Security (VBS) and Hyper-V are Microsoft’s fortress for securing Windows, but what happens when you need to peek under the hood of a running VM? Whether you’re chasing a kernel exploit, analyzing malware in a sandbox, or just curious about securekernel internals, LiveCloudKd is a game-changer. Originally built by Matt Suiche and now maintained by Gerhart, this tool lets you debug Hyper-V guest VMs and dump their memory without mucking around with bootloader settings. Let’s dive into how LiveCloudKd works, why it’s a must-have for security researchers, and how to wield it effectively with WinDbg

It achieves this by leveraging Hyper-V's Memory Manager plugins (hvlib.dll and hvmm.sys), giving you direct access to a VM's memory without the guest OS knowing you're there. Think of it as a stealthy backdoor for virtualization forensics (note – yes, like a LiveKd from Sysinternals Suite).

Why Should You Care?

If you're in exploit development or red teaming, LiveCloudKd is a Swiss Army knife. Need to inspect a Docker container running in Hyper-V isolation? – (note - yes it's support it, but in not public builds and not tested enough for integrated in release build). Want to dump the memory of a VM running Credential Guard to hunt for secrets? Or maybe you're reverse-engineering a rootkit hiding in a VBS-protected process? LiveCloudKd gets you there without tripping over Hyper-V's security or fighting with bcdedit. Plus, it's faster than snapshot-based forensics and doesn't require pausing the VM».

LiveCloudKd offers three methods for read and write memory access:

- ReadInterfaceWinHv/WriteInterfaceWinHv: uses Hyper-V hypercalls to read memory in the same manner as LiveKd. Whilst slower, it is stable and therefore ideal for production virtual machines.
- ReadInterfaceHvmmDrvInternal/WriteInterfaceHvmmDrvInternal: a kernel driver (hvmm.sys) accesses memory directly. This method is faster but may cause crashes during live migration or on virtual machines with dynamic memory enabled. It is therefore useful for virtual machine clones on dedicated hosts for inspection, or within special debugging environments.
- ReadLocalMemory/WriteLocalMemory: kernel driver hvmm.sys get access to memory using Microsoft kernel API functions MmMapIoSpaceEx for writing or ZwMapViewOfSection for reading mode.

You can see detailed description of the tool's functions at the end of this article.

The tool enumerates running virtual machines, enables you to select one, and establishes a debugging session or memory dump. The EXDi plugin is particularly effective—it enables debugging without altering the virtual machine's state, which is crucial for secure kernel analysis or examining VTL1 code. For memory forensics, it can generate raw dumps or structured crash dumps, which can be analysed with olatility to extract process lists, kernel objects, or LSASS credentials.

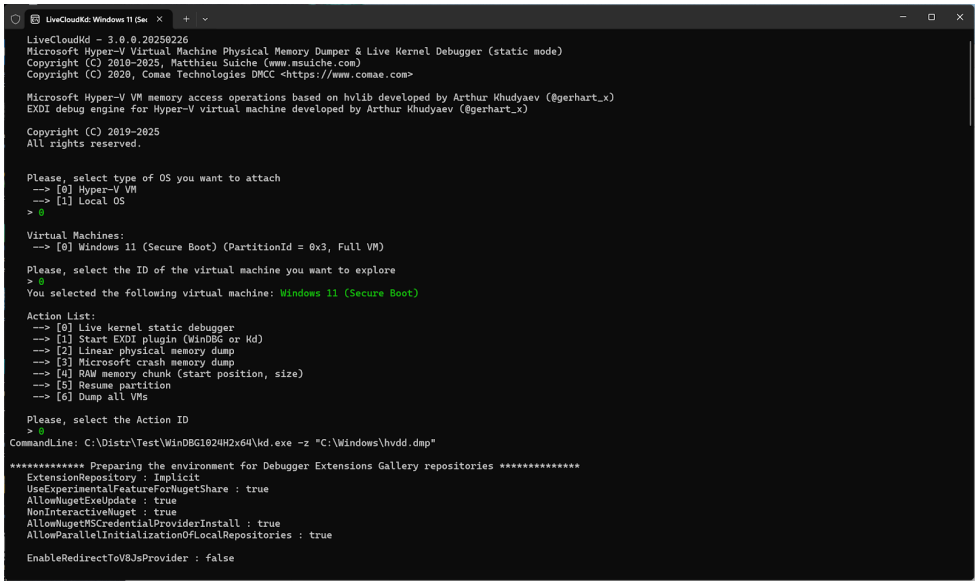
To demonstrate how to launch LiveCloudKd and explore its capabilities:

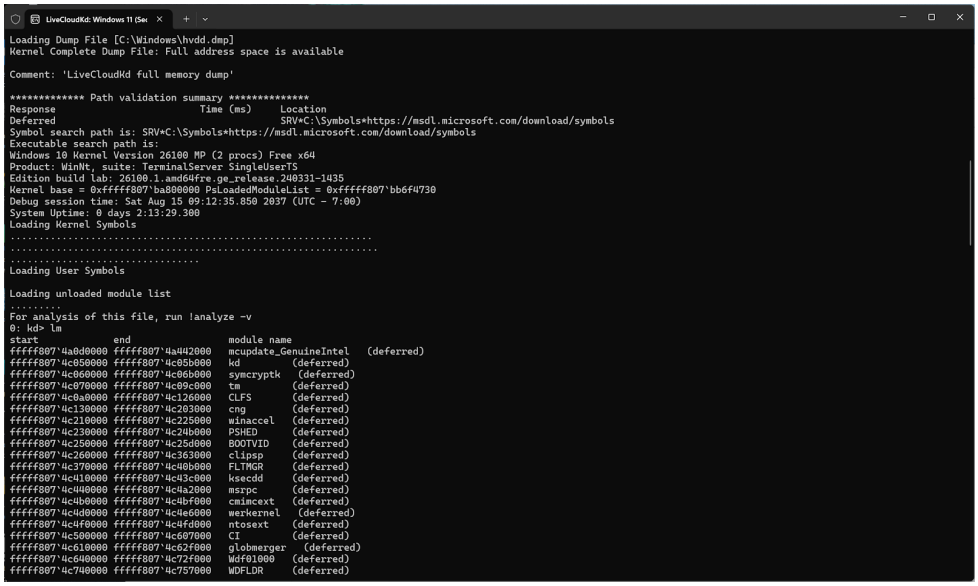
1. Launch a Hyper-V virtual machine and attach LiveCloudKd.exe to it.

2. LiveCloudKd.exe can be attached not only to fully launched virtual machines, but also at early boot stages after pausing the virtual machine. Additionally, you can edit memory for this virtual machine and execute standard WinDBG commands.

3. Various WinDBG plugins and scripts can be utilized whilst working with live virtual machines.

Functionality and performance (how make dump of VM)





For example:

- You can view lists of processes or virtual memory layouts. You can perform the same operations as with a dump file, but with updatable memory (for instance, try to obtain a list of updated processes in the virtual machine by flushing the cache from WinDBG). Complete WinDBG commands usage, for example, can be found in Dmitry Vostokov's dumps anthology (https://www.dumpanalysis.org/advanced-software-debugging-reference).

- You can create standard dump files of the operating system or processes (lsass.exe, for instance) using the following commands:

The current version of LiveCloudKd has the following options:

Usage: LiveCloudKd.exe [/a {0-6}][/b][/m {0-2}][/n {0-9}][/o path][/p][/v {0-2}][/w][/y <path to directory with WinDBG>][/?]

/a Pre-selected action:

0 - Live kernel debugging

1 - Start EXDi plugin (WinDBG)

2 - Produce a linear physical memory dump

3 - Produce a Microsoft full memory crash dump

4 - Dump guest OS memory chunk

5 - Dump raw guest OS memory (without KDBG scanning)

6 - Resume virtual machine

/b Close LiveCloudKd automatically after exiting from kd or WinDBG.

/m Memory access type:

0 - Hypercalls (HvReadGPA and HvWriteGPA)

1 - Raw memory

2 - Local OS

/n Pre-selected virtual machine number.

/o Destination path for the output file (Actions 2-5).

/p Pause partition.

/v Verbose output level.

/w Run WinDBG instead of kd (kd is the default).

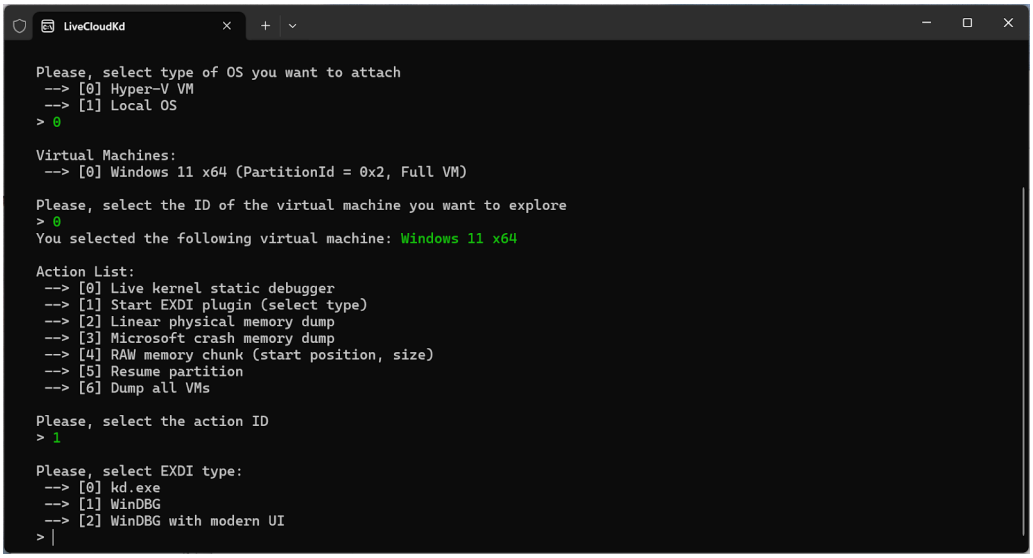
/y Set path to WinDBG or WinDBG with modern UI (for starting the EXDi plugin).

/? Print this help message.

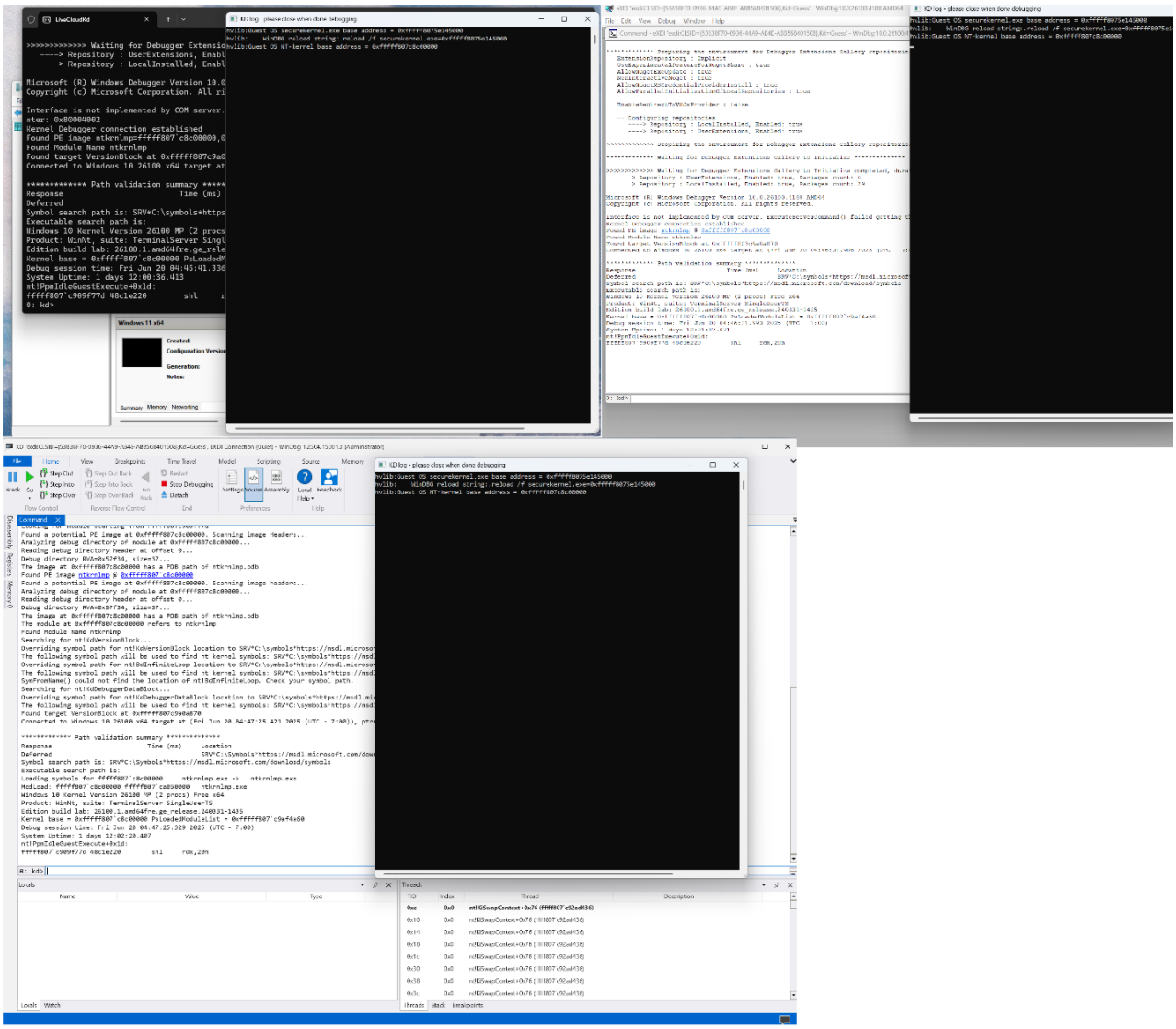
LiveCloudKd works with Hyper-V partitions in two modes: using native Hyper-V calls (HvReadGpa and HvWriteGpa), or using the Hyper-V memory layout, which can be accessed through ring 0. Additional information can be found in article in blog:

https://hvinternals.blogspot.com/2019/09/hyper-v-memory-internals-guest-os-memory-access.html

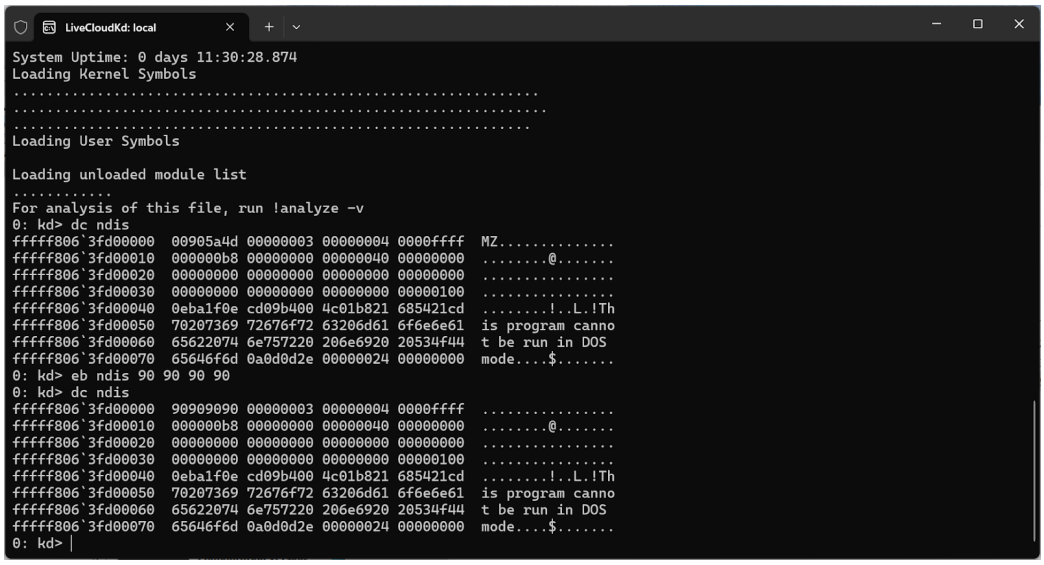
LiveCloudKd also supports a local interface in both read and write modes. I think, that LiveKd remains sufficiently stable for read operations. However, if you wish to explore the local OS and modify it (for instance, to explore some Windows features), you can use LiveCloudKd for these purposes as well.



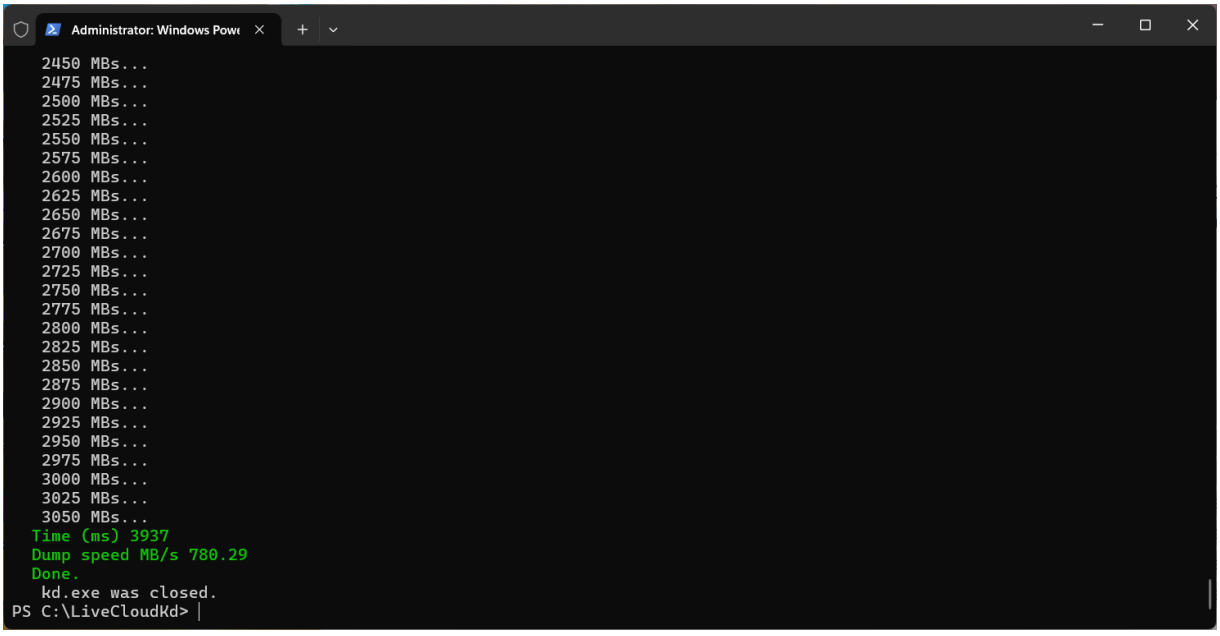
After launch, you can choose to access either local memory or a Hyper-V virtual machine to which you wish to attach.



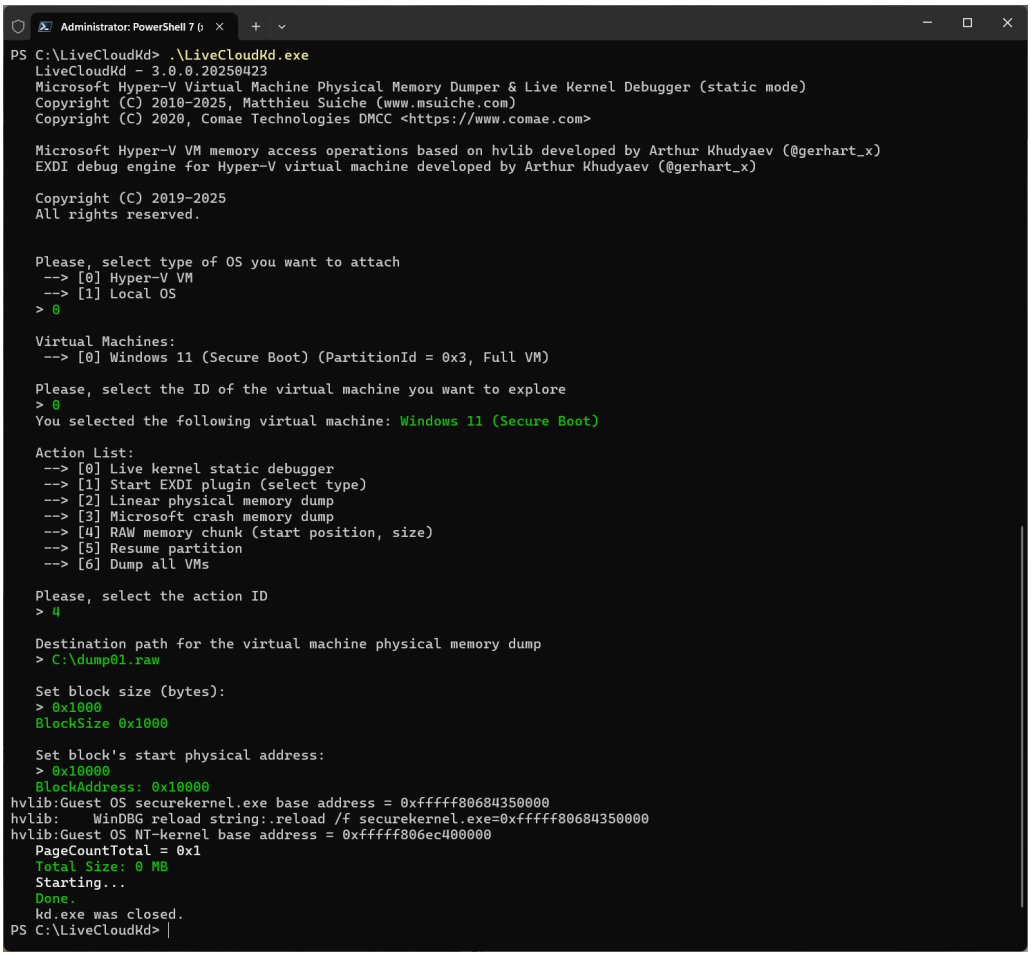
Additionally, you can edit local Windows memory using standard WinDBG commands. To verify the success of this operation, you can either restart the debugger or flush the debugger's cache using the command '.cache flushall', and then execute the same command again.



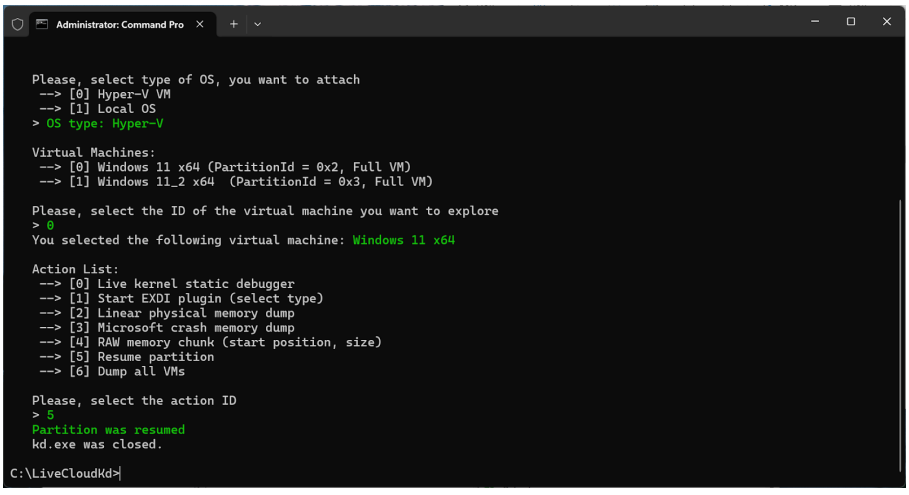
You can create a dump for a Hyper-V virtual machine using the 'Microsoft crash memory dump' option. This option works, when LiveCloudKd and the hvlib engine can verify the correct Hyper-V guest OS version and KDBG structure, which are needed to generate the correct dump header.



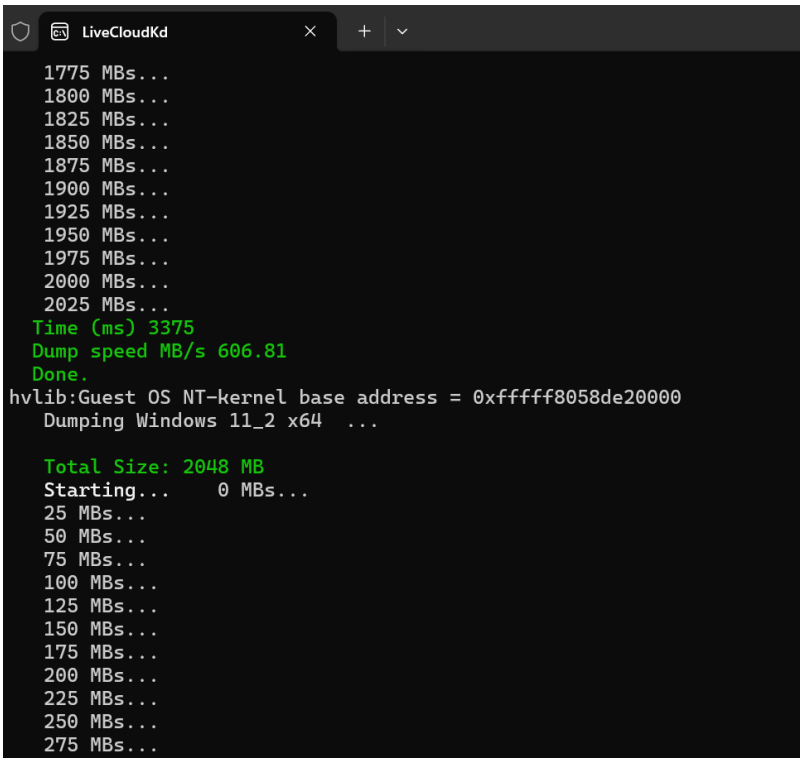
In other cases, use the 'Linear physical memory dump' option. It can dumps all virtual machines in specifying directory, including Linux, Apple MAC OS (for example: https://github.com/Qonfused/OSX-Hyper-V), Cisco Hyper-V virtual machine (https://x.com/gerhart_x/status/1380488855481159680), Android virtual machines, and others, which can be launched on Hyper-V.



Option 5 allows you to resume a partition if it was suspended early:



Option 6 allows you to dump all virtual machines into one directory.



Parameter descriptions:

[HKEY_LOCAL_MACHINE\SOFTWARE\LiveCloudKd\Parameters]

"LogLevel"=dword:00000002 – This is the log-level option, ranging from 0 to 4. Level 0 provides only standard output, whilst level 4 is the most verbose.

"ReloadDriver"=dword:00000000 – Reloads the driver after application start. This can be used if a previous launch of LiveCloudKd has failed.

"ReadMethod"=dword:00000001 – Specifies the read memory method. See the READ_MEMORY_METHOD enumeration.

ReadInterfaceHvmmDrvInternal

ReadInterfaceWinHv

ReadInterfaceHvmmLocal

"WriteMethod"=dword:00000001 – set of write memory methods. See enum WRITE_MEMORY_METHOD.

WriteInterfaceHvmmDrvInternal

WriteInterfaceWinHv

WriteInterfaceHvmmLocal

"VSMScan"=dword:00000001 – scan for securekernel. Can be disabled, but after version 3.0 was enabled by default.

"ForceFreezeCPU"=dword:00000000 – freeze cpu, when memory reading of writing.

"PausePartition"=dword:00000000 – pause partition, when memory reading of writing.

"WinDbgPath" = "C:\\Distributive\\WinDBG1025H2x64\\" – path to WinDBG folder (with slashes at the end or not).

Available VmOpsConfig values:

READ_MEMORY_METHOD ReadMethod;

WRITE_MEMORY_METHOD WriteMethod;

SUSPEND_RESUME_METHOD SuspendMethod;

ULONG64 LogLevel;

BOOLEAN ForceFreezeCPU;

BOOLEAN PausePartition;

HANDLE ExdiConsoleHandle;

BOOLEAN ReloadDriver;

BOOLEAN NestedScan;

BOOLEAN UseDebugApiStopProcess;

BOOLEAN SimpleMemory;

BOOLEAN ReplaceDecypheredKDBG;

BOOLEAN FullCrashDumpEmulation;

BOOLEAN EnumGuestOsBuild;

BOOLEAN VSMScan;

Technically it can be using, if Hyper-V memory manager SDK is used. LiveCloudKd uses that parameters in some cases.

Code for activate VmOpsConfig->PausePartition option:

```
if (VmOpsConfig->PausePartition & VmOpsConfig->ForceFreezeCPU)

{

log_u(log_er, L"PausePartition and ForceFreezeCPU was selected. Switch to use onlu ForceFreezeCPU instead\n");

VmOpsConfig->PausePartition = FALSE;

}
```

```
if (VmOpsConfig->PausePartition & VmOpsConfig->ForceFreezeCPU & ((VmOpsConfig->ReadMethod != ReadInterfaceHvmmLocal)))

{

log_u(log_er, L"hvlib:PausePartition and ForceFreezeCPU was selected. Switch to use only ForceFreezeCPU instead\n");

VmOpsConfig->PausePartition = FALSE;

}
```

Code for attaching to local Windows memory:

```
if (ActionType == ACTION_TYPE_LOCAL_KERNEL_ATTACH)

{

g_VmOperationsConfig.ReadMethod = ReadInterfaceHvmmLocal;

g_VmOperationsConfig.WriteMethod = WriteInterfaceHvmmLocal;

g_MemoryReadInterfaceType = ReadInterfaceHvmmLocal;

g_MemoryWriteInterfaceType = WriteInterfaceHvmmLocal;

DumpLocalMachine();
```

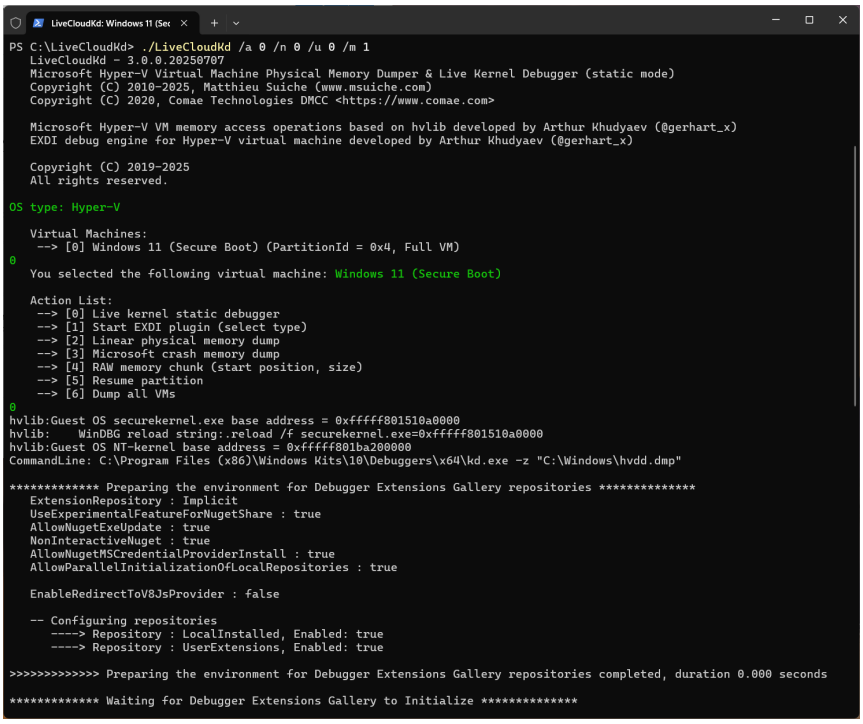
return TRUE;

}

If you wish to launch the utility automatically, you can use the following parameters:

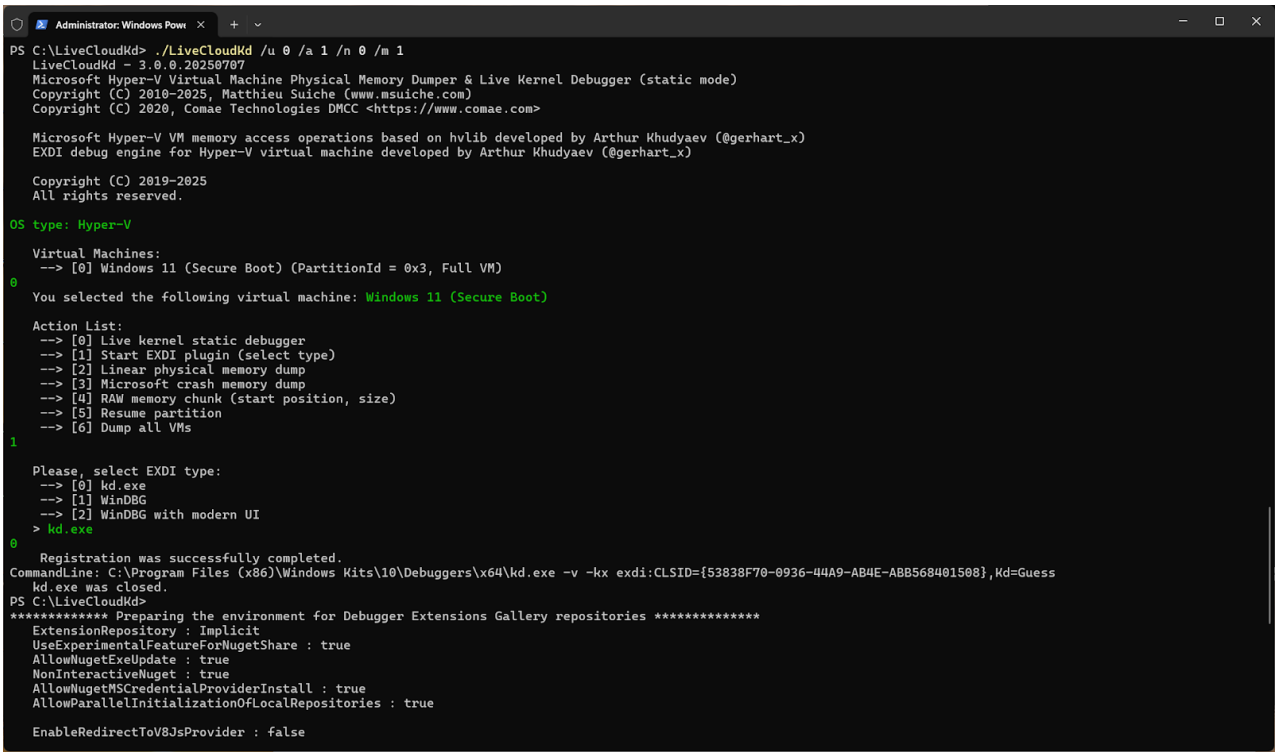
- Memory access type;
- Action type;
- Virtual machine number;
- OS type (Hyper-V or local).

LiveCloudKd /a 0 /n 0 /u 0 /m 1

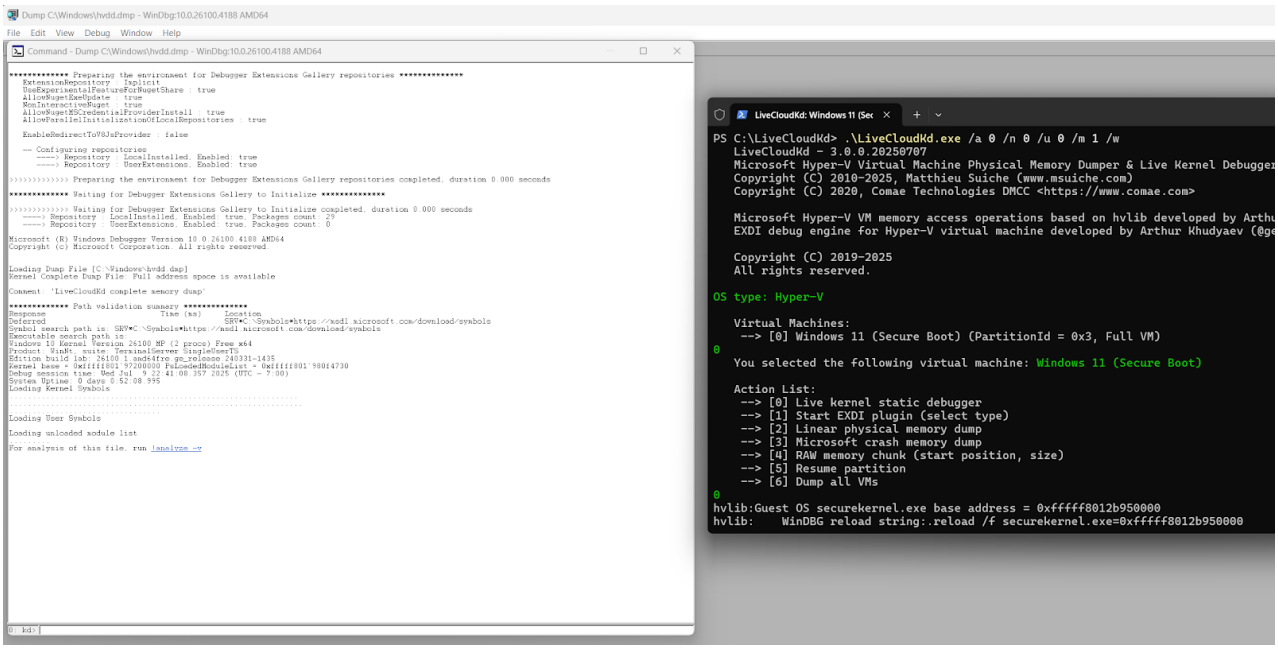


If you use Start EXDI plugin option, you see three options for launching:

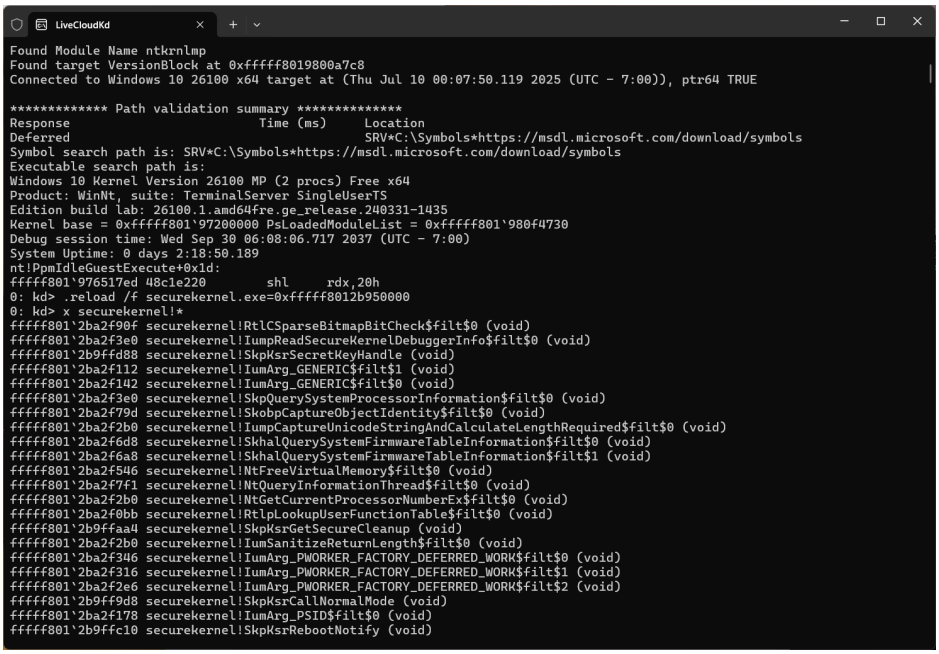
LiveCloudKd /u 0 /a 1 /n 0 /m 1

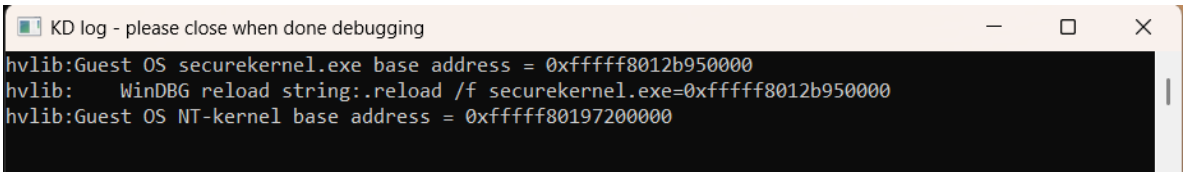


LiveCloudKd.exe /a 0 /n 0 /u 0 /m 1 /w

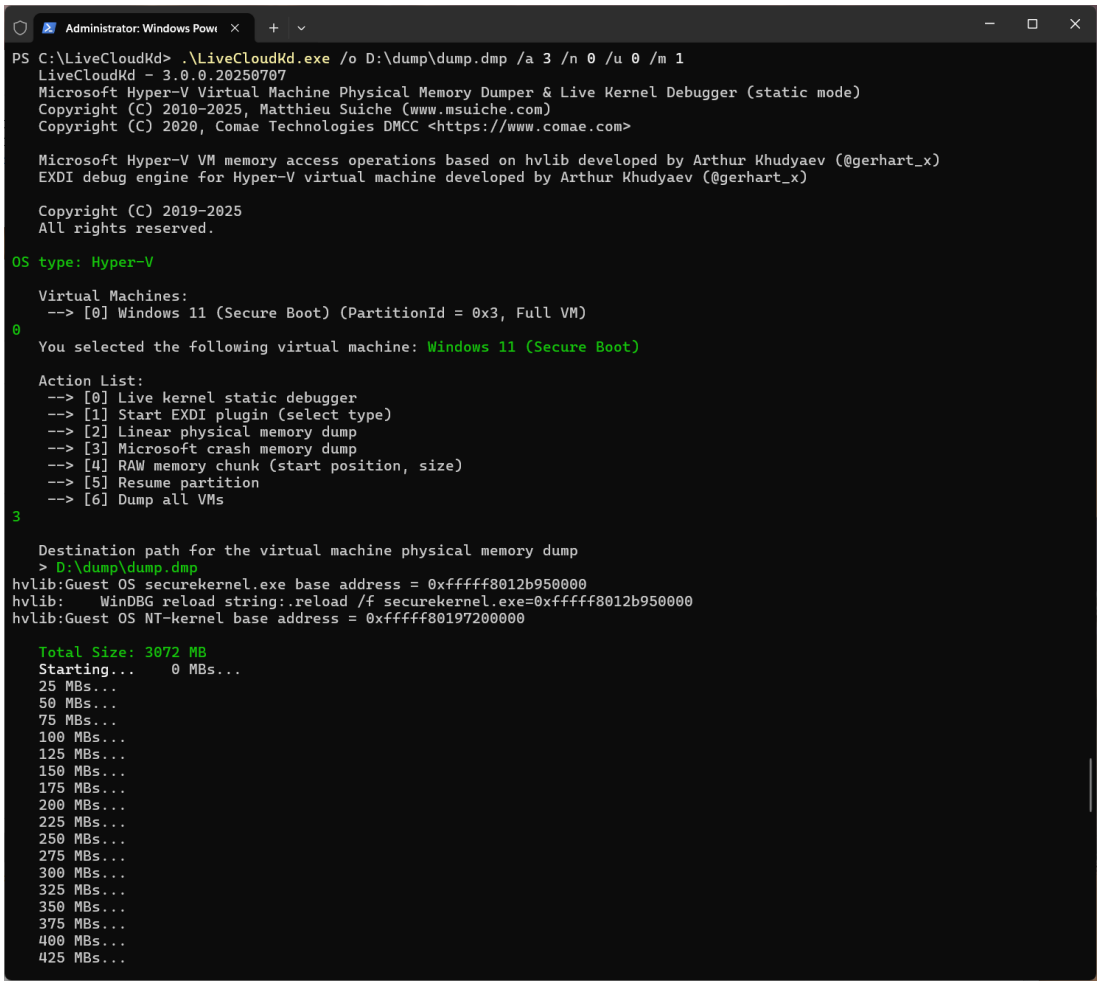


Also, you can see information about loaded securekernel in EXDi log:

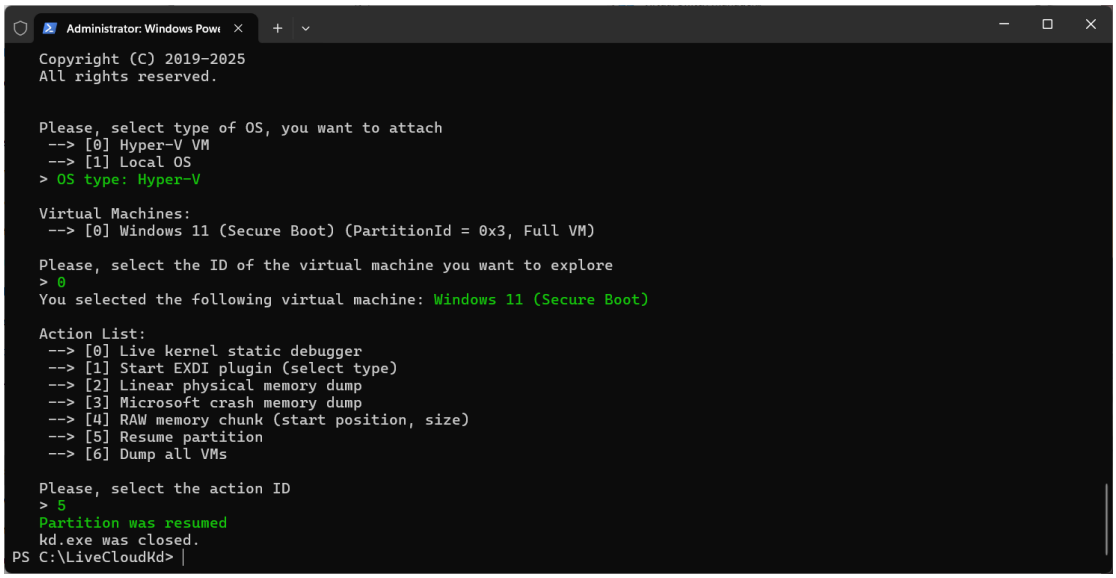




LiveCloudKd.exe /a 3 /o D:\dump\dump.dmp /y C:\WinDBG /n 0 /u 0 /m 1



If you use the 'p' key and the partition has been paused, you can return it to its usual state using the 'Resume partition' option.



Memory dumping mechanisms (most of the implementation was created by Matt Suiche, but required modifications for newer WinDBG versions).

LiveCloudKd implements comprehensive memory dumping capabilities through multiple action modes. Linear physical memory dumps provide complete virtual machine memory in raw format, whilst Microsoft full memory crash dumps generate standard Windows crash dump files compatible with WinDbg. The tool also supports guest OS memory chunk dumps for targeted analysis and raw guest OS memory dumps for forensic investigation.

EXDi integration and debugging capabilities

The Extended Debugger Interface (EXDi) integration represents a significant advancement in LiveCloudKd's debugging capabilities. This integration enables WinDBG to debug VMs without enabling kernel debugging in bootloader configurations.

EXDi capabilities include live debugging of Hyper-V VMs, VTL0/VTL1 (Virtualization Trust Level) context switching for secure kernel debugging, software breakpoints supporting up to 0x1000 breakpoints, single-step debugging functionality, and secure kernel debugging for VBS-enabled VMs. The EXDi plugin architecture allows integration with hardware debuggers, JTAG interfaces, and custom debugging implementations.

It is enough deep technology and can be described later.

Live debugging integration

The live debugging functionality represents one of LiveCloudKd's most sophisticated features. The tool implements Export Address Table (EAT) hooking of WinDBG/kd.exe to create live debugging sessions, redirecting memory read/write operations to Hyper-V VM memory while presenting VM memory as a virtual crash dump file to debuggers.

Dynamic debugging option with that EXDi plugin started developed from the beginning of 2020 and was briefly mentioned at Black Hat 2020 by Saar Amar (https://x.com/AmarSaar or @amarsaar@infosec.exchange) in the presentation 'Breaking VSM by Attacking Secure Kernel: Hardening Secure Kernel through Offensive Research' (https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2020_08_BlackHatUSA/Breaking_VSM_by_Attacking_SecureKernel.pdf). Additionally, this debugger usage was described by Yarden Shafir (https://x.com/yarden_shafir): https://windows-internals.com/secure-kernel-research-with-livecloudkd. Whilst the functionality remains effective, there is scope for improvements in both functionality and stability.

Annex:

Complete dump format implementation

The crash dump format follows Microsoft's specifications:

- Signature: 'EGAP' (0x50414745—PAGE reversed)
- ValidDump: 'PMUD' for 32-bit systems, '46UD' for 64-bit systems
- DumpType: DUMP_TYPE_FULL (1) for complete dumps
- Comment: 'Hyper-V Memory Dump. (c) 2010 MoonSols SARL'

Memory layout is described through PHYSICAL_MEMORY_DESCRIPTOR64:

- Supports up to 32 memory runs (Run[0x20] variable)
- Each run contains BasePage and PageCount
- Enables sparse memory representation

Context Management

The tool manages processor contexts for all virtual CPUs:

- x64 contexts: 3,000-byte buffer in the dump header

- Segment register fixup: Ensures kernel-mode selectors are properly configured
    - CS: KGDT64_R0_CODE (0x10)
    - SS: KGDT64_R0_DATA (0x18)
    - DS/ES: KGDT64_R3_DATA | RPL_MASK

KDBG injection: Placed at the KdDebuggerDataBlockPa offset

The memory dumping process leverages the hvlib.dll library for memory access abstraction, uses page-based operations with boundary checking to prevent read and write errors, and implements virtual machine state management with suspend and resume capabilities to ensure memory consistency during dump operations.

Key implementation details:

· Magic handle: 0x1337 identifies VM memory mappings

· Header injection: Crash dump header prepended at offset 0

· Dynamic context updates: Real-time CPU state reflection

This integration enables real-time debugging of virtual machines without traditional kernel debugging setup, supports standard WinDBG commands for memory analysis, and maintains compatibility with the Windows Symbol Server for symbol resolution. The debugging integration works seamlessly with both legacy and modern versions of WinDBG.

ReadInterfaceHvmmDrvInternal Implementation. You can see it in driver source code:

https://github.com/gerhart01/LiveCloudKd/blob/master/hvmm/hvmm/vid.c#L477

This high-performance method, that is used in hvmm kernel driver:

- Packs memory requests into GPA_INFO structures
- Obtains partition handles via nt!ObReferenceObjectByHandle
- Accesses FILE_OBJECT through FsContext pointers
- Uses nt!MmMapLockedPagesSpecifyCache for virtual address mapping
- Achieves ~1000x performance improvement over LiveKd

In addition requires custom signed driver or test signing mode on ARM64 Windows Server or Client version.

ReadInterfaceWinHv Implementation

Uses official Windows Hypervisor Platform APIs:

- Calls HvReadGPA/HvWriteGPA through winhvr.sys
- Provides maximum compatibility across Windows versions
- Implements proper page boundary checking
- Slower, but more stable than driver methods
- No special privileges required beyond VM access

ReadInterfaceVidNative Implementation

Maintains compatibility with Windows Server 2012-2016:

- Direct vid.dll function calls:
    - VidReadMemoryBlockPageRange
    - VidWriteMemoryBlockPageRange
    - VidQueryMemoryBlockMbpCount
    - VidDmMemoryBlockQueryTopology
- Requires memory block handle enumeration
- Limited by Microsoft's API restrictions since Windows Server 2019

Hooking and patching mechanisms

LiveCloudKd's hooking mechanisms represent advanced exploitation techniques for debugging integration. The tool implements sophisticated API interception through Import Address Table (IAT) modification and memory-mapped file redirection.

API Hooking Architecture

Actualizing with

https://media.blackhat.com/bh-dc-11/Suiche/BlackHat_DC_2011_Suiche_Cloud_Pocket-wp.pdf

The hooking system uses a FUNCTION_TABLE structure allocated at address 0xFFF000000 in the target debugger process. This table contains:

- Original function pointers for all hooked APIs
- VM metadata including partition handles and memory topology
- Context information for up to MAX_PROCESSORS CPUs
- KDBG structure

Hook functions are injected at specific offsets:

- CREATEFILE_OFFSET: MyCreateFile hook (0x300 bytes)
- CREATEFILEMAPPINGA_OFFSET: MyCreateFileMappingA (0x100 bytes)
- CREATEFILEMAPPINGW_OFFSET: MyCreateFileMappingW (0x100 bytes)
- MAPVIEWOFFILE_OFFSET: MyMapViewOfFile (0x1B00 bytes)
- UNMAPVIEWOFFILE_OFFSET: MyUnmapViewOfFile (0x100 bytes)
- GETFILESIZE_OFFSET: MyGetFileSize (0x100 bytes)
- VIRTUALPROTECT_OFFSET: MyVirtualProtect (0xD00 bytes)

Memory Redirection Mechanism

The core innovation lies in MyMapViewOfFile(), which intercepts memory mapping requests:

- Detects special handle 0x1337 indicating VM memory access
- Allocates local buffers aligned to page boundaries
- Translates file offsets to VM physical addresses

- Reads memory through SdkReadPhysicalMemory with configured access method
- Injects processor contexts and KDBG structures at runtime
- Maintains mapping table for proper cleanup

The hook implements sophisticated context injection:

```
if ((ContextPageIndex[i] >= Position) && (ContextPageIndex[i] < (Position +
NumberOfPages))) {

// Inject saved context at correct memory location

// Adjust segment selectors for kernel mode if needed

}
```

Write-Through Implementation

MyVirtualProtect() enables memory modification in live debugging:

- Detects VirtualProtect calls on mapped VM memory
- Validates address ranges against mapping table
- Reads current memory content from VM
- Applies modifications from debugger
- Writes changes back through VidWriteMemoryBlockPageRange
- Updates local cache for consistency

This creates a crash dump debugging experience for VM introspection, enables real-time debugging without traditional kernel debugging setup, and maintains compatibility with standard WinDBG debugging workflows.

Component integration and workflow

The various source files work together through a sophisticated integration architecture. The main livecloudkd.c orchestrates VM selection, debugging coordination, and WinDBG integration, while memoryblock.c manages Hyper-V memory blocks and VM memory topology discovery. Partition.c handles partition discovery and validation, implementing brute-force enumeration of vmwp.exe memory space.

The dump.c component implements memory dump generation for various output formats, while hooker.c manages EAT hooking for WinDBG integration. Kd.c handles kernel debugger integration and launching, file.c manages file operations for dump output and configuration, and misc.c provides utility functions and helper routines.

This integrated architecture enables LiveCloudKd to provide comprehensive VM introspection capabilities, from basic memory dumping to sophisticated live debugging scenarios. The tool's modular design allows for flexible deployment across different Windows versions and debugging requirements while maintaining high performance and reliability.

Appendix 1:

Detailed function description

This section provides a description of functions, implemented in LiveCloudKd's source files, detailing their purpose, implementation, and role in the overall architecture.

livecloudkd.c Functions

wmain()

Purpose: Main entry point for LiveCloudKd executableParameters: Standard argc/argv command line argumentsImplementation:

- Parses command-line arguments through ParseArguments()
- Imports NT functions via ImportGlobalNtFunctions()
- Enumerates available Hyper-V partitions using SdkEnumPartitions()
- Presents interactive VM selection menu
- Executes selected action (live debug, memory dump, etc.)
- Manages cleanup through SdkCloseAllPartitions()

ParseArguments()

Purpose: Processes command-line arguments and sets global configurationParameters: argc (count), argv (argument array)Implementation:

- Parses switches: /w (WinDbg), /x (WinDbgX), /l (Live mode), /e (EXDi), etc.

- Sets memory access method through g_MemoryReadInterfaceType
- Configures output paths and pre-selected VMs
- Validates argument combinations

WriteEXDiPartitionId()

Purpose: Writes selected VM ID to registry for EXDi pluginParameters: VmId (selected virtual machine ID)Return: BOOLEAN success statusImplementation:

- Creates/opens registry key at HKLM\SOFTWARE\LiveCloudKd\Parameters
- Writes VmId as DWORD value
- Enables EXDi plugin to identify target VM

dump.c Functions

DumpMemoryBlock()

Purpose: Dumps specified memory region from VMParameters:

- PartitionEntry: Target VM handle
- DestinationFile: Output file path
- Start: Beginning physical address
- Size: Number of bytes to dump
- DumpMode: Type of dump (standard/raw)
- Creates destination file with CreateDestinationFile()
- Allocates BLOCK_SIZE buffer (1MB)
- Reads memory in chunks using SdkReadPhysicalMemory()
- Writes to file with WriteFileSynchronous()
- Handles read failures by writing zeros

DumpVirtualMachine()

Purpose: Creates complete linear physical memory dumpParameters: PartitionEntry, DestinationFileImplementation:

- Retrieves VM memory size via SdkGetData(InfoMmMaximumPhysicalPage)
- Dumps entire physical memory space page by page
- Shows progress every 10MB
- Handles sparse memory regions

DumpCrashVirtualMachine()

Purpose: Creates Microsoft-compatible crash dumpParameters: PartitionEntry, DestinationFile

Implementation:

- Fills crash dump header using DumpFillHeader()
- Writes header followed by memory contents
- Injects CONTEXT structures at appropriate locations
- Inserts KDBG data block at correct offset
- Ensures WinDbg compatibility

DumpLiveVirtualMachine()

Purpose: Enables live debugging sessionParameters: PartitionEntry, VmIdImplementation:

- Creates temporary crash dump file in Windows directory
- Populates FUNCTION_TABLE with VM metadata
- Hooks debugging functions for live access
- Launches appropriate debugger (kd/WinDbg/WinDbgX)
- Cleans up temporary files on exit

DumpFillHeader()

Purpose: Determines appropriate dump header formatParameters: PartitionEntry, Header (output), HeaderSize (output)Return: Success statusImplementation:

- Checks machine type (x86/x64)
- Calls appropriate header filling function
- Currently only supports x64 (returns error for x86)

DumpFillHeader64()

Purpose: Creates 64-bit crash dump headerParameters: PartitionEntryReturn: Populated DUMP_HEADER64 structureImplementation:

- Fills signature fields (DUMP_SIGNATURE, DUMP_VALID_DUMP64)
- Sets Windows version from NtBuildNumber
- Populates kernel structures (DirectoryTableBase, PfnDatabase, etc.)
- Creates fake bug check code ('MATT')
- Builds physical memory descriptor
- Copies CONTEXT from guest
- Adjusts segment registers for kernel mode

hooker.c Functions

HookKd()

Purpose: Hooks debugger process for VM memory redirectionParameters: ProcessHandle, ProcessIdReturn: Success statusImplementation:

- Takes snapshot of target process modules
- Finds dbgeng.dll in module list
- Allocates memory in target process for hook table
- Writes hook functions to target process
- Patches IAT entries to redirect API calls

- Supports both legacy and modern API names

PatchIAT()

Purpose: Modifies Import Address Table entriesParameters: ModuleBase, ImportModuleName, FunctionName, AddressImplementation:

- Parses PE headers to find import descriptors
- Locates target DLL in import table
- Walks thunk tables to find function
- Replaces function pointer with hook address

MyCreateFile()

Purpose: Hooked CreateFile to intercept hvdd.dmp accessParameters: Standard CreateFileW parametersImplementation:

- Checks if filename ends with "hvdd.dmp"
- Stores handle in FunctionTable.CrashDumpHandle
- Calls original CreateFileW for actual operation

MyMapViewOfFile()

Purpose: Core memory access hookParameters: Standard MapViewOfFile parametersImplementation:

- Detects special handle (0x1337) for VM memory
- Allocates local buffer for requested size
- Copies crash dump header if at offset 0
- Reads VM physical memory using configured method
- Injects CONTEXT and KDBG structures at correct offsets
- Tracks mapped regions for later unmapping

MyCreateFileMappingA/W()

Purpose: Intercepts file mapping creationParameters: Standard CreateFileMapping parametersImplementation:

- Checks if handle matches crash dump file
- Returns magic handle (0x1337) for VM access
- Passes through other requests unchanged

MyUnmapViewOfFile()

Purpose: Cleans up mapped VM memoryParameters: Base address to unmapImplementation:

- Searches tracked mappings for address
- Frees allocated memory
- Clears tracking entry

MyGetFileSize()

Purpose: Reports VM memory size as file sizeParameters: File handle, high-order sizeImplementation:

- Checks for crash dump handle
- Returns calculated VM memory size
- Includes header size in calculation

MyVirtualProtect()

Purpose: Handles memory protection changes with write-throughParameters: Standard VirtualProtect parametersImplementation:

- Detects operations on mapped VM memory
- Reads current page content from VM
- Applies changes from source buffer

- Writes modified pages back to VM
- Updates local cache

kd.c functions

LaunchKd()

Purpose: Launches kd.exe with hooked memory accessParameters: DumpFile path, PartitionEntryImplementation:

- Creates kd.exe process in debug mode
- Waits for initial breakpoint
- Duplicates partition handle to target process
- Calls HookKd() to install hooks
- Manages debug event loop
- Handles thread creation events

LaunchWinDbg()

Purpose: Launches WinDbg with EXDi integrationParameters: PartitionEntryImplementation:

- Builds EXDi connection string
- Ensures EXDi COM registration
- Creates WinDbg process with appropriate arguments
- Handles path resolution

LaunchWinDbgX()

Purpose: Launches WinDbg Preview with EXDiParameters: PartitionEntryImplementation:

- Similar to LaunchWinDbg but for Store version

- Handles different executable location
- Notes command-line limitations

LaunchWinDbgLive()

Purpose: Launches WinDbg in live debugging modeParameters: PartitionEntryImplementation:

- Uses different EXDi CLSID for live mode
- Configures for real-time VM debugging

EXDiRegistration()

Purpose: Ensures EXDi COM component registrationReturn: Success statusImplementation:

- Checks registry for COM registration
- Runs regsvr32.exe if needed
- Verifies successful registration

CheckEXDiRegistration()

Purpose: Verifies EXDi COM registration statusReturn: Registration statusImplementation:

- Opens COM class registry key
- Checks for InprocServer32 entry

file.c functions

CreateDestinationFile()

Purpose: Creates output file for memory dumpsParameters: Filename, Handle (output)Return: Success status

Implementation:

- Creates file with FILE_FLAG_NO_BUFFERING
- Ensures proper alignment for direct I/O
- Returns handle for subsequent operations

WriteFileSynchronous()

Purpose: Performs synchronous write with verificationParameters: Handle, Buffer, NbOfBytesToWriteReturn: Success status

Implementation:

- Issues WriteFile operation
- Handles ERROR_IO_PENDING case
- Verifies all bytes written
- Ensures data reaches disk

misc.c Functions

ImportGlobalNtFunctions()

Purpose: Loads NTDLL functions dynamicallyReturn: Success statusImplementation:

- Loads ntdll.dll
- Resolves function pointers:
    - NtAllocateVirtualMemory
    - NtDuplicateObject
    - NtOpenProcess
    - NtQueryObject
    - NtQuerySystemInformation
- Stores in global g_NtDll structure

GetMmNonPagedPoolLimit()

Purpose: Calculates non-paged pool memory limitsParameters: MmNonPagedPoolStart, MmNonPagedPoolEnd (outputs)Return: Success statusImplementation:

- Queries system basic information
- Calculates PFN database size
- Determines non-paged pool boundaries
- Used for memory range validatio

GetConsoleTextAttribute()

Purpose: Retrieves current console text colorParameters: Console handleReturn: Text attribute valueImplementation:

- Gets console screen buffer info
- Extracts attribute field
- Used for color preservation

White(), Red(), Green()

Purpose: Colored console output functionsParameters: Format string and variadic argumentsImplementation:

- Saves current console color
- Sets specific color (white/red/green)
- Outputs formatted text
- Restores original color

Data Structures and File Formats

Crash Dump Header Structures

DUMP_HEADER64

Purpose: 64-bit Windows crash dump headerSize: 8192 bytes (2 pages)Key Fields:

- Signature: 'EGAP' (PAGE backwards)

- ValidDump: '46UD' for 64-bit dumps
- MajorVersion/MinorVersion: Windows version
- DirectoryTableBase: CR3 value
- KdDebuggerDataBlock: Pointer to KDBG
- PhysicalMemoryBlock: Memory run descriptors
- ContextRecord: Processor context
- BugCheckCode: 'MATT'

PHYSICAL_MEMORY_DESCRIPTOR64

Purpose: Describes physical memory layoutFields:

- NumberOfRuns: Count of memory regions
- NumberOfPages: Total page count
- Run[]: Array of memory runs

X64_CONTEXT

Purpose: 64-bit processor contextSize: 1232 bytes

Contains: All CPU registers including:

- General purpose (RAX-R15)
- Segment registers
- Debug registers
- XMM registers
- Control registers

Function Table Structure

FUNCTION_TABLE

Purpose: Central data structure for hooking system

Key Fields:

- Function pointers for hooked APIs
- VM metadata (partition handle, memory size)
- Context information for all CPUs
- Memory mapping tracking
- KDBG location and contents