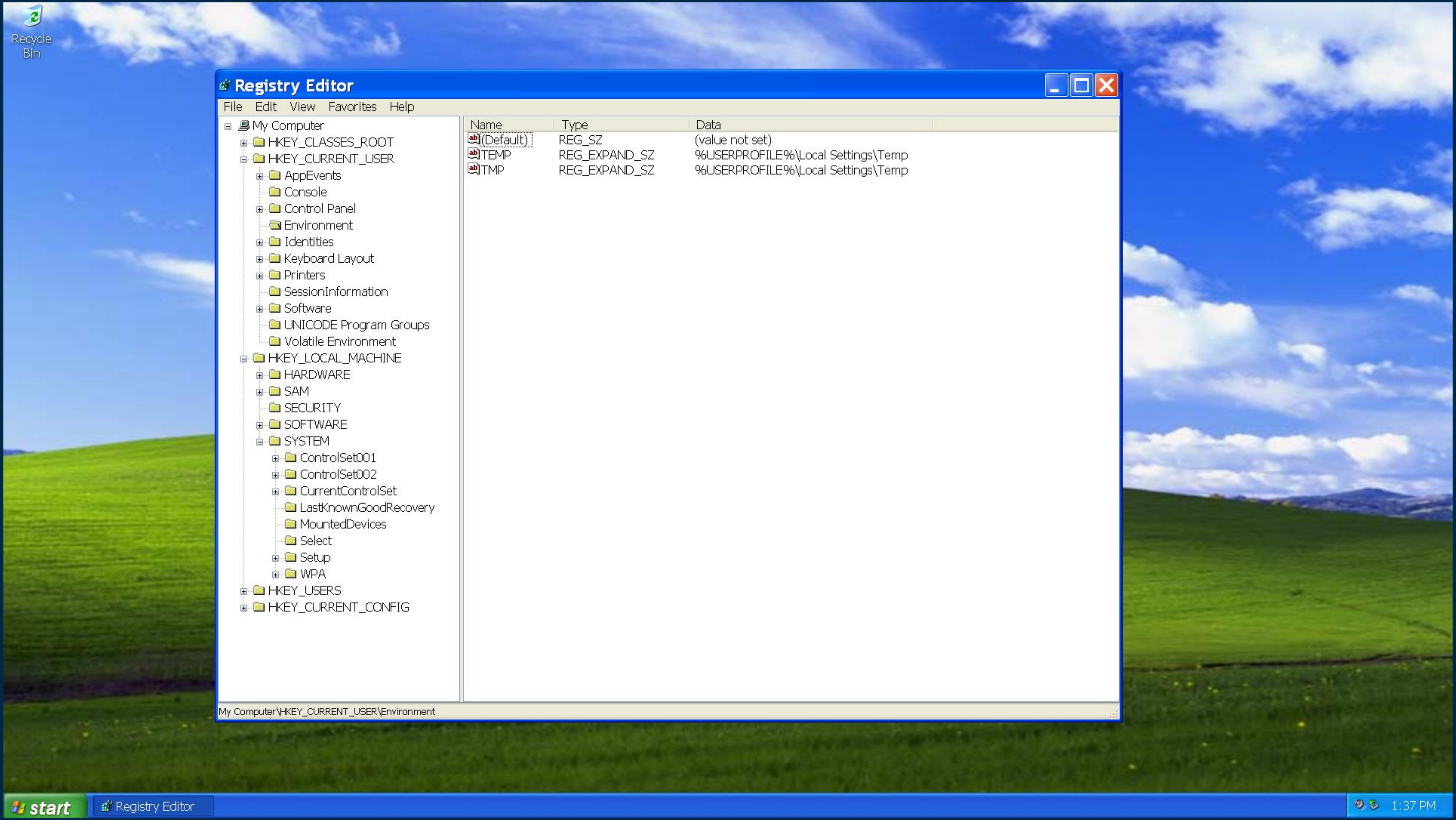
Microsoft

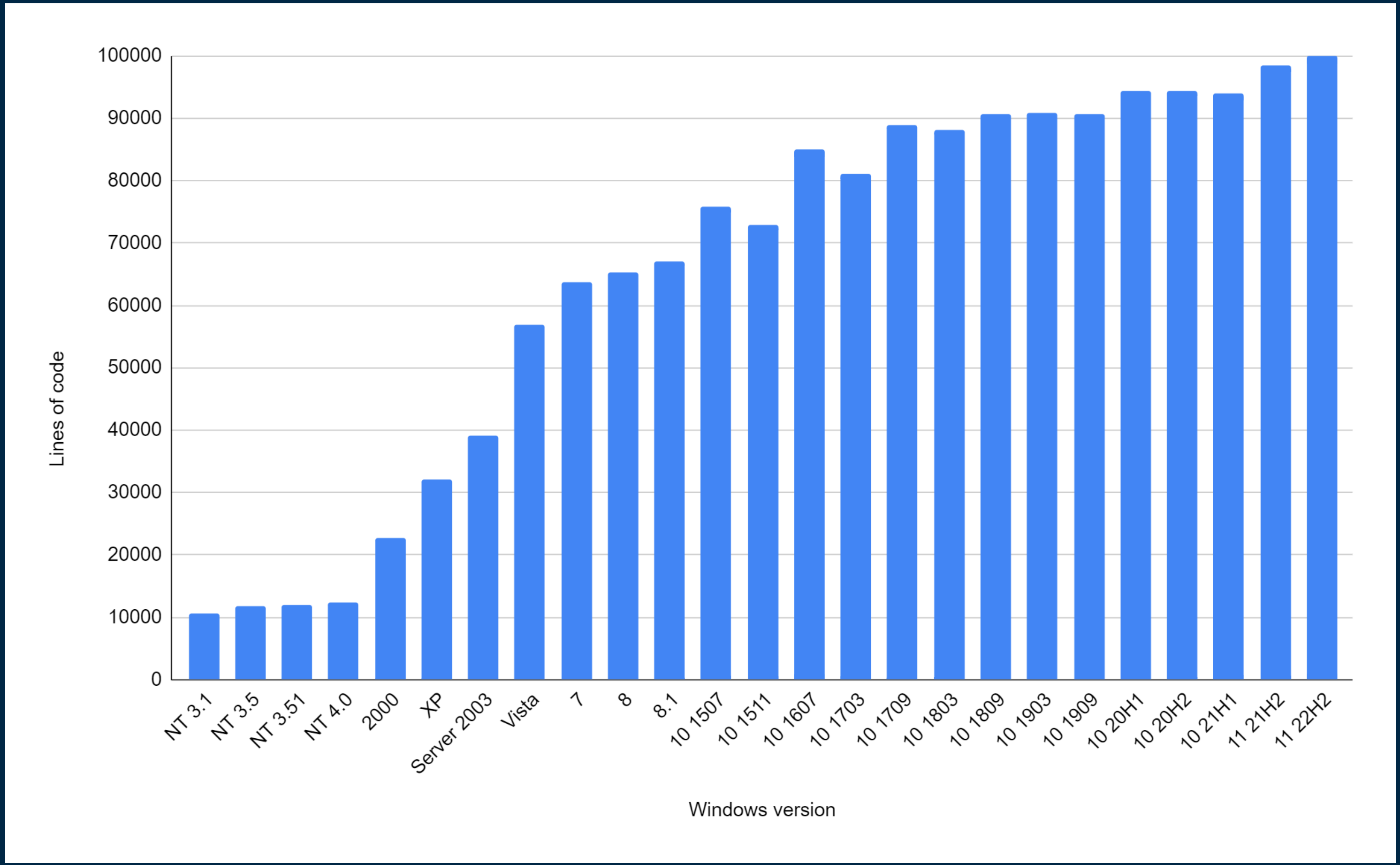
MSRC BLUEHAT OCT '23

Exploring the Windows Registry as a powerful LPE attack surface

Mateusz Jurczyk
Microsoft BlueHat, October 2023


Recycle Bin
Registry Editor
File Edit View Favorites Help
My Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
AppEvents
Console
Control Panel
Environment
Identities
Keyboard Layout
Printers
SessionInformation
Software
UNICODE Program Groups
Volatile Environment
HKEY_LOCAL_MACHINE
HARDWARE
SAM
SECURITY
SOFTWARE
SYSTEM
ControlSet001
ControlSet002
CurrentControlSet
LastKnownGoodRecovery
MountedDevices
Select
Setup
WPA
HKEY_USERS
HKEY_CURRENT_CONFIG
Name Type Data
(Default) REG_SZ (value not set)
TEMP REG_EXPAND_SZ %USERPROFILE%\Local Settings\Temp
TMP REG_EXPAND_SZ %USERPROFILE%\Local Settings\Temp
My Computer\HKEY_CURRENT_USER\Environment
start
Registry Editor
1:37 PM

Registry: Lines of kernel code (decompiled)



Timeline of most important features

Windows NT 3.1 - 4.0 (1993 – 1996)
- Initial implementation
- Predefined Keys
- Early hive versions: 1.1, 1.2
- Major version 1.3 (Fast Leaf support)

Windows 2000 (2000)

Windows XP (2001)
- Key Rename
- Registry Callbacks
- Minor version 1.4 (Big Value support)
- Major version 1.5 (Hash Leaf support)

Windows Server 2003 (2003)

Windows Vista (2007)
- KTM Transactions
- Registry Virtualization
- App Keys

Windows 7, 8, 8.1 (2009 – 2013)

Windows 10 1507, 1511 (2015)

Windows 10 1607 (2016)
- Lightweight Transactions
- Containers: Namespace Redirection
- Containers: Differencing Hives
- Major version 1.6 (Layered Key support)

Windows 10/11 1703 – 22H2 (2017 – 2022)

Why attack the registry?

- Local attack surface potentially allowing privilege escalation in the system
- Stores and operates on sensitive data (system configuration, user credentials)
- Many potential types of issues:
  - Plain memory corruption
  - Logic bugs
  - Information disclosure
  - Inter-process disruption (registry as a shared resource)
- Huge old/new C codebase with layers of complex mechanisms mixed together

Prior publicly known security research

- Evidently a lot of work done internally at Microsoft
- Relatively little prior art in the public space
  - 2010: **5 bugs** reported by Gynvael Coldwind and myself
  - 2014 – 2020: **17 bugs** by James Forshaw
    - A consistent stream of kernel logic issues, many at the intersection of registry and other system mechanisms (security impersonation, file system)
  - 2016: **4 bugs** reported by James and me as a result of some basic hive format fuzzing
  - 201X: Several isolated bugs reported by others (Fortinet, Maxim Suhanov)

This effort

- Started in May 2022 as a test of my new coverage-based fuzzer
- Found one (1) and only bug: **GPZ-2299/CVE-2022-35768**
  - *Windows Kernel multiple memory problems when handling incorrectly formatted security descriptors in registry hives*
  - Checks out as security descriptors are one of the few things well-suited for binary fuzzing
- The initial success prompted me to have a deeper look into the kernel
- It quickly turned into a challenge to review all of the code…

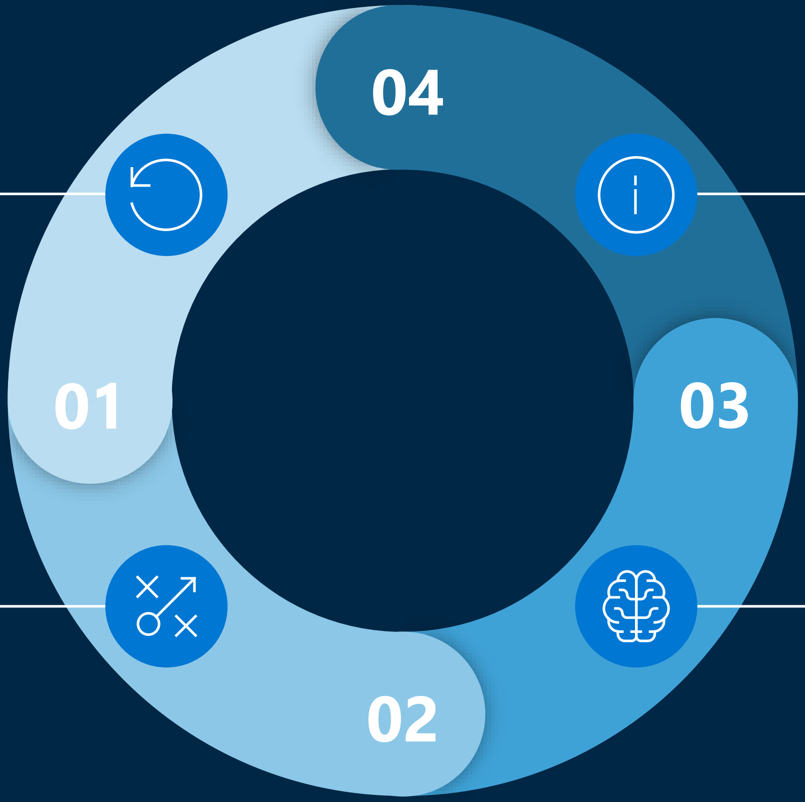
The research process

Reverse engineer

Choose a self-contained part of the registry implementation and try to get it as close to readable C-like code as possible.

Understand the logic

Try to understand the purpose, assumptions, guarantees and underlying intentions of the code.



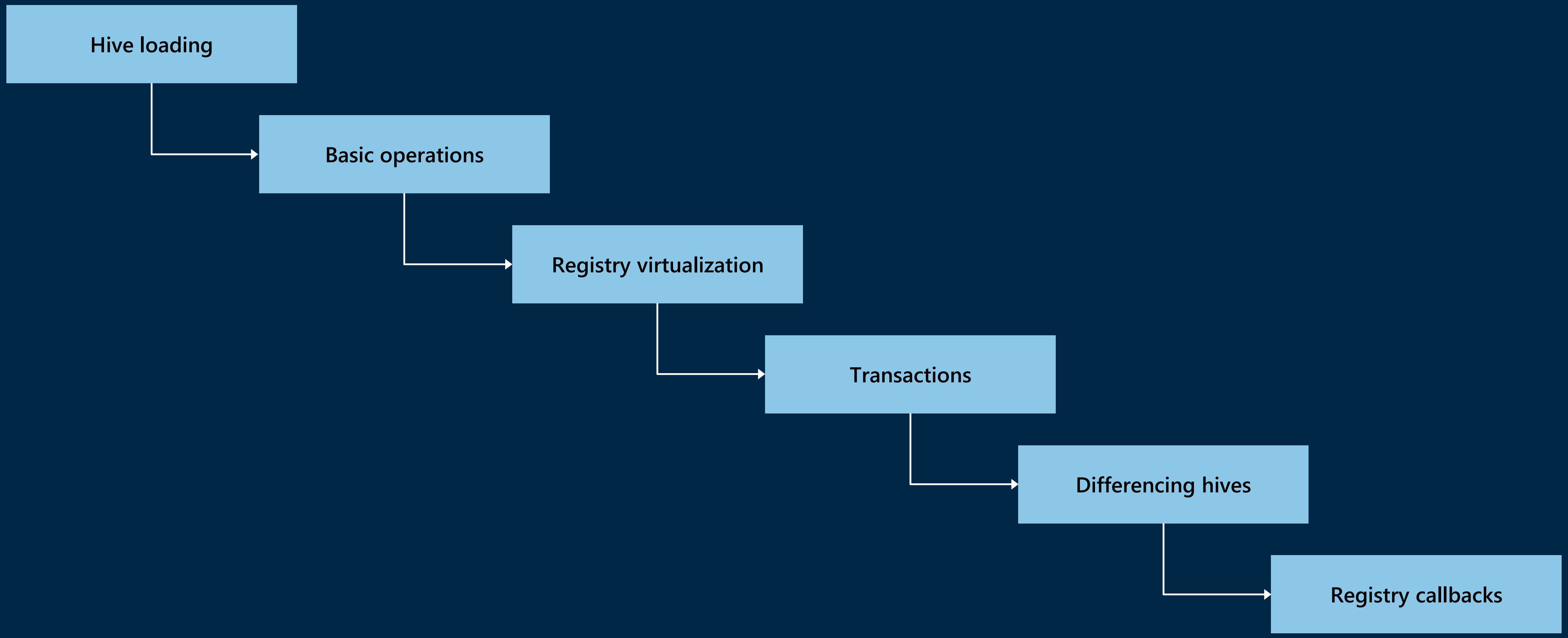
Compare with prior knowledge

Consider if the behaviour of the feature is consistent with what we already know about the registry.

Test, reproduce, report bugs

Test any discovered bugs, create reliable reproducers, write up detailed reports and submit them to Microsoft.

Research progression: Major features



Results as of September 2023

Really hard to quantify actual number of bugs (what is a "bug"?)

- 33 issues marked as Fixed in the Project Zero bug tracker
- ~45 unique problems (in my assessment)
- 39 CVEs assigned by Microsoft
- ?? fixes introduced in the source code

Official classification:

- 33 x Windows Kernel Elevation of Privilege Vulnerability
- 5 x Windows Kernel Information Disclosure Vulnerability
- 1 x Windows Kernel Memory Information Disclosure Vulnerability

ID	Status	Restrict	Reported	Vendor	Product	Finder	Summary + Labels
2295	Fixed	----	2022-May-11	Microsoft	Kernel	mjurczyk	Windows Kernel use-after-free due to refcount overflow in registry hive security descriptors CCProjectZeroMembers
2297	Fixed	----	2022-May-17	Microsoft	Kernel	mjurczyk	Windows Kernel invalid read/write due to unchecked Blink cell index in root security descriptor CCProjectZeroMembers
2299	Fixed	----	2022-May-20	Microsoft	Kernel	mjurczyk	Windows Kernel multiple memory problems when handling incorrectly formatted security descriptors in registry hives CCProjectZeroMembers
2318	Fixed	----	2022-Jun-22	Microsoft	Kernel	mjurczyk	Windows Kernel integer overflows in registry subkey lists leading to memory corruption CCProjectZeroMembers
2330	Fixed	----	2022-Jul-8	Microsoft	Kernel	mjurczyk	Windows Kernel registry use-after-free due to bad handling of failed reallocations under memory pressure CCProjectZeroMembers
2332	Fixed	----	2022-Jul-11	Microsoft	Kernel	mjurczyk	Windows Kernel memory corruption due to type confusion of subkey index leaves in registry hives CCProjectZeroMembers
2341	Fixed	----	2022-Aug-3	Microsoft	Kernel	mjurczyk	Windows Kernel multiple memory corruption issues when operating on very long registry paths CCProjectZeroMembers
2344	Fixed	----	2022-Aug-5	Microsoft	Kernel	mjurczyk	Windows Kernel out-of-bounds reads and other issues when operating on long registry key and value names CCProjectZeroMembers
2359	Fixed	----	2022-Sep-22	Microsoft	Kernel	mjurczyk	Windows Kernel use-after-free due to bad handling of predefined keys in NtNotifyChangeMultipleKeys CCProjectZeroMembers
2366	Fixed	----	2022-Oct-6	Microsoft	Kernel	mjurczyk	Windows Kernel memory corruption due to insufficient handling of predefined keys in registry virtualization CCProjectZeroMembers
2369	Fixed	----	2022-Oct-13	Microsoft	Kernel	mjurczyk	Windows Kernel use-after-free due to dangling registry link node under paged pool memory pressure CCProjectZeroMembers
2375	Fixed	----	2022-Oct-25	Microsoft	Kernel	mjurczyk	Windows Kernel multiple issues in the key replication feature of registry virtualization CCProjectZeroMembers
2378	Fixed	----	2022-Oct-31	Microsoft	Kernel	mjurczyk	Windows Kernel registry SID table poisoning leading to bad locking and other issues CCProjectZeroMembers
2379	Fixed	----	2022-Nov-2	Microsoft	Kernel	mjurczyk	Windows Kernel allows deletion of keys in virtualizable hives with KEY_READ and KEY_SET_VALUE access rights CCProjectZeroMembers
2389	Fixed	----	2022-Nov-30	Microsoft	Kernel	mjurczyk	Windows Kernel registry virtualization incompatible with transactions, leading to inconsistent hive state and memory corruption CCProjectZeroMembers
2392	Fixed	----	2022-Dec-7	Microsoft	Kernel	mjurczyk	Windows Kernel multiple issues with subkeys of transactionally renamed registry keys CCProjectZeroMembers
2394	Fixed	----	2022-Dec-14	Microsoft	Kernel	mjurczyk	Windows Kernel multiple issues in the prepare/commit phase of a transactional registry key rename CCProjectZeroMembers
2408	Fixed	----	2023-Jan-13	Microsoft	Kernel	mjurczyk	Windows Kernel insufficient validation of new registry key names in transacted NtRenameKey CCProjectZeroMembers
2410	Fixed	----	2023-Jan-19	Microsoft	Kernel	mjurczyk	Windows Kernel CmpCleanupLightWeightPrepare registry security descriptor refcount leak leading to UAF CCProjectZeroMembers
2418	Fixed	----	2023-Jan-31	Microsoft	Kernel	mjurczyk	Windows Kernel disclosure of kernel pointers and uninitialized memory through registry KTM transaction log files CCProjectZeroMembers
2419	Fixed	----	2023-Feb-2	Microsoft	Kernel	mjurczyk	Windows Kernel out-of-bounds reads when operating on invalid registry paths in CmpDoReDoCreateKey/CmpDoReOpenTransKey CCProjectZeroMembers
2433	Fixed	----	2023-Mar-7	Microsoft	Kernel	mjurczyk	Windows Kernel KTM registry transactions may have non-atomic outcomes CCProjectZeroMembers
2445	Fixed	----	2023-Apr-19	Microsoft	Kernel	mjurczyk	Windows Kernel arbitrary read by accessing predefined keys through differencing hives CCProjectZeroMembers
2446	Fixed	----	2023-Apr-20	Microsoft	Kernel	mjurczyk	Windows Kernel may reference unbacked layered keys through registry virtualization CCProjectZeroMembers
2447	Fixed	----	2023-Apr-27	Microsoft	Kernel	mjurczyk	Windows Kernel may reference rolled-back transacted keys through differencing hives CCProjectZeroMembers
2449	Fixed	----	2023-May-2	Microsoft	Kernel	mjurczyk	Windows Kernel renaming layered keys doesn't reference count security descriptors, leading to UAF CCProjectZeroMembers
2452	Fixed	----	2023-May-10	Microsoft	Kernel	mjurczyk	Windows Kernel CmDeleteLayeredKey may delete predefined tombstone keys, leading to security descriptor UAF CCProjectZeroMembers
2454	Fixed	----	2023-May-15	Microsoft	Kernel	mjurczyk	Windows Kernel out-of-bounds reads due to an integer overflow in registry .LOG file parsing CCProjectZeroMembers
2456	Fixed	----	2023-May-22	Microsoft	Kernel	mjurczyk	Windows Kernel partial success of registry hive log recovery may lead to inconsistent state and memory corruption CCProjectZeroMembers

Reverse engineering

Reverse engineering the Windows Kernel

- An essential step: It probably took around 70-80% of the total research time
- Analysis primarily done on Windows Server 2019 (originally used for fuzzing), then reproduced on up-to-date Windows 11
- Tools: IDA Pro disassembler with Hex-Rays decompiler
- Extra aid: PDBs for ntoskrnl.exe hosted on the Microsoft Symbol Server
  - Function names, global variable names, some structure layouts, some enum definitions

Before

```
__int64 __fastcall CmGetKCBCacheSecurity(__int64 a1, __int64 a2)
{
  __int64 v2; // rdi
  __int64 v5; // rbp
  __int64 PrevElement; // rax
  __int64 v7; // rbx
  __int64 v8; // [rsp+30h] [rbp+8h] BYREF

  v2 = *(_QWORD *)(a1 + 80);
  if ( a2 )
  {
    v8 = 0i64;
    v5 = a1 + 200;
    while ( 1 )
    {
      PrevElement = CmListGetPrevElement(v5, &v8);
      v7 = PrevElement;
      if ( !PrevElement )
        break;
      if ( (unsigned __int8)CmEqualTrans(*(_QWORD *)(PrevElement + 56), a2) && *(_DWORD *)(v7 + 68) == 9 )
        return *(_QWORD *)(v7 + 88);
    }
  }
  return v2;
}
```

After

```
_CM_KEY_SECURITY_CACHE *__fastcall CmGetKCBCacheSecurity(_CM_KEY_CONTROL_BLOCK *Kcb, _CM_TRANS *Trans)
{
  _CM_KEY_SECURITY_CACHE *CachedSecurity; // rdi
  _LIST_ENTRY *p_KCBUoWListHead; // rbp
  _CM_KCB_UOW *uow; // rax MAPDST
  _LIST_ENTRY *CurrElement; // [rsp+30h] [rbp+8h] BYREF

  CachedSecurity = Kcb->CachedSecurity;
  if ( Trans )
  {
    CurrElement = 0i64;
    p_KCBUoWListHead = &Kcb->KCBUoWListHead;
    while ( 1 )
    {
      uow = CmListGetPrevElement(p_KCBUoWListHead, &CurrElement);
      if ( !uow )
        break;
      if ( CmEqualTrans(uow->Transaction, Trans) && uow->ActionType == UoWSetSecurityDescriptor )
        return uow->TxCachedSecurity;
    }
  }
  return CachedSecurity;
}
```

The hard part

- Dealing with compiler optimizations: inlined functions, mangled arithmetic etc.
  - Sometimes necessary to cross-check the same code in different builds of Windows
- Figuring out the missing pieces
  - Reconstructing internal structures: the parse context, on-disk transaction log records, some structures related to virtualization/transactions, all structures related to differencing hives
  - Reverse-engineering the meaning and names of constants – basically guesswork
    - Both static and dynamic analysis employed to try to deduct their meaning
    - Many of them still poorly/not understood
- Dear Microsoft: please publish more information through the public symbols, every bit helps

Understanding the code

Understanding the code

- Once the code is readable, we can analyze a particular feature as a whole
  - What problem is it trying to solve?
  - Is it implemented the same way I would intuitively do it? If not, why?
  - Is it internally consistent?
  - Does it correctly handle error conditions?
  - Does it behave in accordance with the documentation?
  - Does it make any assumptions that aren't explicitly enforced?
  - What interesting primitives does it enable?
    (even if they're not bugs on their own)
- Exposes deeper, logic bugs and structural weaknesses

Examples of deep-rooted bug classes

Resource exhaustion

Handling partial success in multi-step operations

Constraints of the hive binary format: Expectations vs reality

Resource exhaustion

- Every "create" and "set" operation internally (re)allocates buffers from the hive storage or kernel pools
- A local attacker may try to interfere by exhausting both types of memory:
  - Hive storage: allocation failure very practical and easy to trigger
  - Kernel pools: allocation failure possible but slightly less practical

Hive size limit exhaustion

- Two separate quotas enforced on registry size:
  - The maximum size of a single hive is 4 GiB
  - The cumulative system-wide registry quota is also 4 GiB
- That's two ways to reliably cause `HvAllocateCell` to fail
- Opens up a plethora of interesting, deep error code paths to review
  - Every such path needs to restore the registry to a known-good state
  - Most of them are probably poorly tested, as they almost never trigger in real life
- One of my main focuses throughout the research

Resource exhaustion – Allocation call sites

xrefs to HvAllocateCell

Direction	Type	Address	Text
Up	o	.pdata:0000000140111498	RUNTIME_FUNCTION <rva HvAllocateCell, rva byte
Up	p	CmpCreateTombstone+BE	call HvAllocateCell
Up	p	CmpSetValueKeyExisting+2A3	call HvAllocateCell
Up	p	CmpAddSubKeyEx+1C7	call HvAllocateCell
Up	p	CmpSetValueDataNew+70	call HvAllocateCell
Up	p	CmpSetValueDataNew+101	call HvAllocateCell
Up	p	CmpSetValueDataNew+15F	call HvAllocateCell
Up	p	CmpSetValueDataNew+1C1	call HvAllocateCell
Up	p	CmpCreateChild+3D5	call HvAllocateCell
Up	p	CmpCreateChild+8F4	call HvAllocateCell
Up	p	CmpAddValueToListEx+11F	call HvAllocateCell
Up	p	CmpAddValueKeyNew+6E	call HvAllocateCell
Do...	p	HvDuplicateCell+90	call HvAllocateCell
Do...	p	CmpCopyCell+92	call HvAllocateCell
Do...	p	CmpSetValueDataExisting+2...	call HvAllocateCell
Do...	p	CmpSetSecurityDescriptorIn...	call HvAllocateCell
Do...	p	CmpGetSecurityDescriptorN...	call HvAllocateCell
Do...	p	CmpCreateHiveRootCell+9C	call HvAllocateCell
Do...	p	CmpAddSubKeyEx+223F74	call HvAllocateCell
Do...	p	CmpCreateChild+22359E	call HvAllocateCell
Do...	p	CmRenameKey+D86	call HvAllocateCell
Do...	p	CmRenameKey+135E	call HvAllocateCell
Do...	p	CmpAddValueKeyTombston...	call HvAllocateCell
Do...	p	CmpConcatenateValueLists...	call HvAllocateCell
Do...	p	CmpCopyMergeOfLayeredK...	call HvAllocateCell
Do...	p	CmpSplitLeaf+12B	call HvAllocateCell
Do...	p	CmpCommitRenameKeyUo...	call HvAllocateCell
Do...	p	CmpLightWeightPrepareAd...	call HvAllocateCell
Do...	p	CmpLightWeightPrepareRe...	call HvAllocateCell
Do...	p	CmpCreateRootNode+83	call HvAllocateCell

Line 1 of 30

OK Cancel Search Help

xrefs to CmpAllocatePool

Direction	Type	Address	Text
Up	o	.pdata:00000001400D6CA4	RUNTIME_FUNCTION <rva CmpAllocatePool, rv
Up	p	CmpDoFileWrite+52	call CmpAllocatePool
Up	p	CmpMarkIndexDirty+85	call CmpAllocatePool
Up	p	CmpGetNameControlBlock...	call CmpAllocatePool
Up	p	CmpConstructNameFromK...	call CmpAllocatePool
Up	p	CmpDoParseKey+5A7	call CmpAllocatePool
Up	p	CmpDoParseKey+13B9	call CmpAllocatePool
Up	p	CmpDoParseKey+142A	call CmpAllocatePool
Up	p	CmpDoParseKey+32B1	call CmpAllocatePool
Up	p	CmLoadAppKey+FF	call CmpAllocatePool
Up	p	CmLoadKey+EC	call CmpAllocatePool
Up	p	CmLoadKey+15E	call CmpAllocatePool
Up	p	CmpAddToHiveFileList+59	call CmpAllocatePool
Up	p	CmAllocateExtraParameter...	call CmpAllocatePool
	p	CmpSetSecurityDescriptorIn...	call CmpAllocatePool
Do...	p	CmpCreateSiloKeyLockEntr...	call CmpAllocatePool
Do...	p	CmpCreateGlobalKeyLockE...	call CmpAllocatePool
Do...	p	CmpFinishSystemHivesLoa...	call CmpAllocatePool
Do...	p	CmpFinishSystemHivesLoa...	call CmpAllocatePool
Do...	p	CmpCreateRegistryProcessT...	call CmpAllocatePool
Do...	p	CmpReorganizeHive+227B95	call CmpAllocatePool
Do...	p	CmpStartKcbStack+211F46	call CmpAllocatePool
Do...	p	CmQueryValueKey+209B5E	call CmpAllocatePool
Do...	p	CmpParseKey+205A66	call CmpAllocatePool
Do...	p	CmpLogTransactionAborted...	call CmpAllocatePool
Do...	p	CmpSaveBootControlSet+135	call CmpAllocatePool
Do...	p	CmpLoadHiveVolatile+2B7	call CmpAllocatePool
Do...	p	CmpRefreshHive+254	call CmpAllocatePool
Do...	p	CmpReadBuildVersion+FA	call CmpAllocatePool
Do...	p	CmpReadBuildVersion+14C	call CmpAllocatePool
Do...	p	CmpRecordShutdownStopT...	call CmpAllocatePool

Line 1 of 58

OK Cancel Search Help

Partial success of multi-step operations

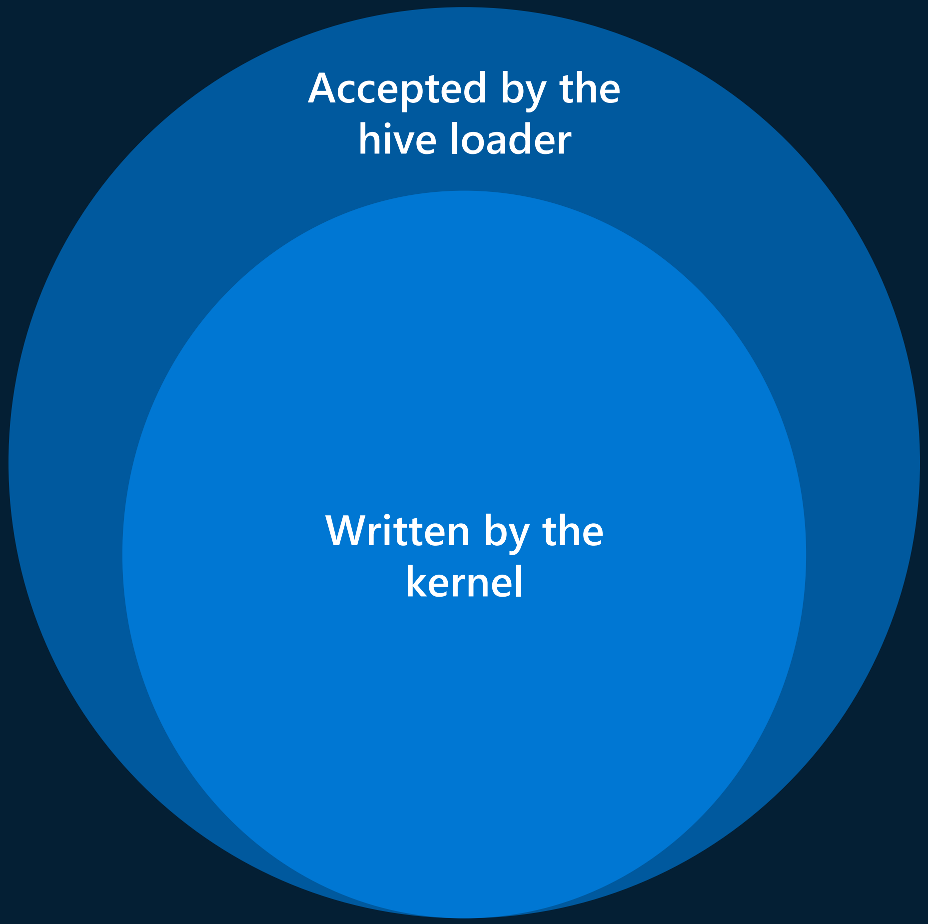
- Error handling gets even more difficult across function boundaries
- Information about outcome is passed back via a `NTSTATUS` return value
  - In theory a 32-bit type, in practice mostly used as a binary success/failure differentiator
  - Semantically more of a "last error encountered" than "overall operation status"
- Not many, but there are some functions implementing multi-step operations
  - `CmpReplicateKeyToVirtual` – recreates a virtualized key in the user's hive (virtual store)
  - `CmpTransMgrCommit` – commits an entire transaction, which may consist of an unlimited number of operations

Error handling-related bugs

- GPZ-2330: Windows Kernel registry use-after-free due to bad handling of failed reallocations under memory pressure
- GPZ-2369: Windows Kernel use-after-free due to dangling registry link node under paged pool memory pressure
- GPZ-2375: Windows Kernel multiple issues in the key replication feature of registry virtualization
- GPZ-2394: Windows Kernel multiple issues in the prepare/commit phase of a transactional registry key rename
- GPZ-2410: Windows Kernel CmpCleanupLightWeightPrepare registry security descriptor refcount leak leading to UAF
- GPZ-2433: Windows Kernel KTM registry transactions may have non-atomic outcomes
- GPZ-2456: Windows Kernel partial success of registry hive log recovery may lead to inconsistent state and memory corruption

Constraints of the hive format

- When loading a hive, the kernel performs extensive sanity checks of its structure
  - `CmpCheckKey`, `CmpCheckValueList`, many other `CmpCheck*` functions
- While quite strict, the checks still allow(ed) some constructs that would never be produced by the kernel itself: by mistake or by design



Odd-but-accepted construct examples

Hive data construct	Accepted by loader	Written by kernel
Large cells >16 KiB not aligned to power of two	✓	✗
Non-compressed ASCII-only key names	✓	✗
Empty subkey lists	✓	✗
Leaf subkey lists longer than 507/1013 elements	✓	✗
Subkey list types incompatible with hive version	✓	✗
Unused security descriptors	✓	✗
Duplicate security descriptors	✓	✗
Values with duplicate names	✓	✗

Odd-but-accepted construct examples

Hive data construct	Accepted by loader	Written by kernel
Large cells >16 KiB not aligned to power of two	✓	✗
Non-compressed ASCII-only key names	✓	✗
Empty subkey lists	✓	✗
Leaf subkey lists longer than 507/1013 elements	✓	✗
Subkey list types incompatible with hive version	✓	✗
Unused security descriptors	✓	✗
Duplicate security descriptors	✓	✗
Values with duplicate names	✓	✗

Interactions between features

Interactions between features

- The original registry design from Windows NT seems defendable
  - Simple, predictable operations that work exactly as advertised
- Then increasingly complex features got introduced, complicating internal state
  - KCB size: **64 bytes** in Windows 2000, **176 bytes** in Windows 10 22H2
- Each of them implements some kind of ✨magic✨ that may not be obvious to other parts of the kernel
- Let's consider how they deviate from the baseline mental model

Special types of keys

Symbolic links (flag 0x10)

- Transparently point to another registry key, if opened with default options
- Makes it harder to reason if opening a key really opened *that* key, or even the intended hive
- Often useful as an exploitation primitive
- Can be created via API, used extensively by Windows itself

Predefined-handle keys (flag 0x40)

- Transparently point to a chosen predefined key (HKLM, HKCU, HKCR, etc.)
- Have no values, the `ValueList` part of key node is reused for a different purpose
- Cannot be operated on by most syscalls – supposed to only ever be opened

Registry virtualization

- Compatibility mechanism to create the illusion of running as administrator
- Redirects and replicates operations within the system-wide **HKLM\Software** hive to a user-accessible copy in **HKCU\Software\Classes\VirtualStore**
- Outcome: creation of a key in a different location than specified, or reading from multiple sources when the caller thinks it's just one key

Transactions

Normal registry: Everything either *is* or *isn't*

Transacted registry: Everything *is*, *isn't*, or *is pending*

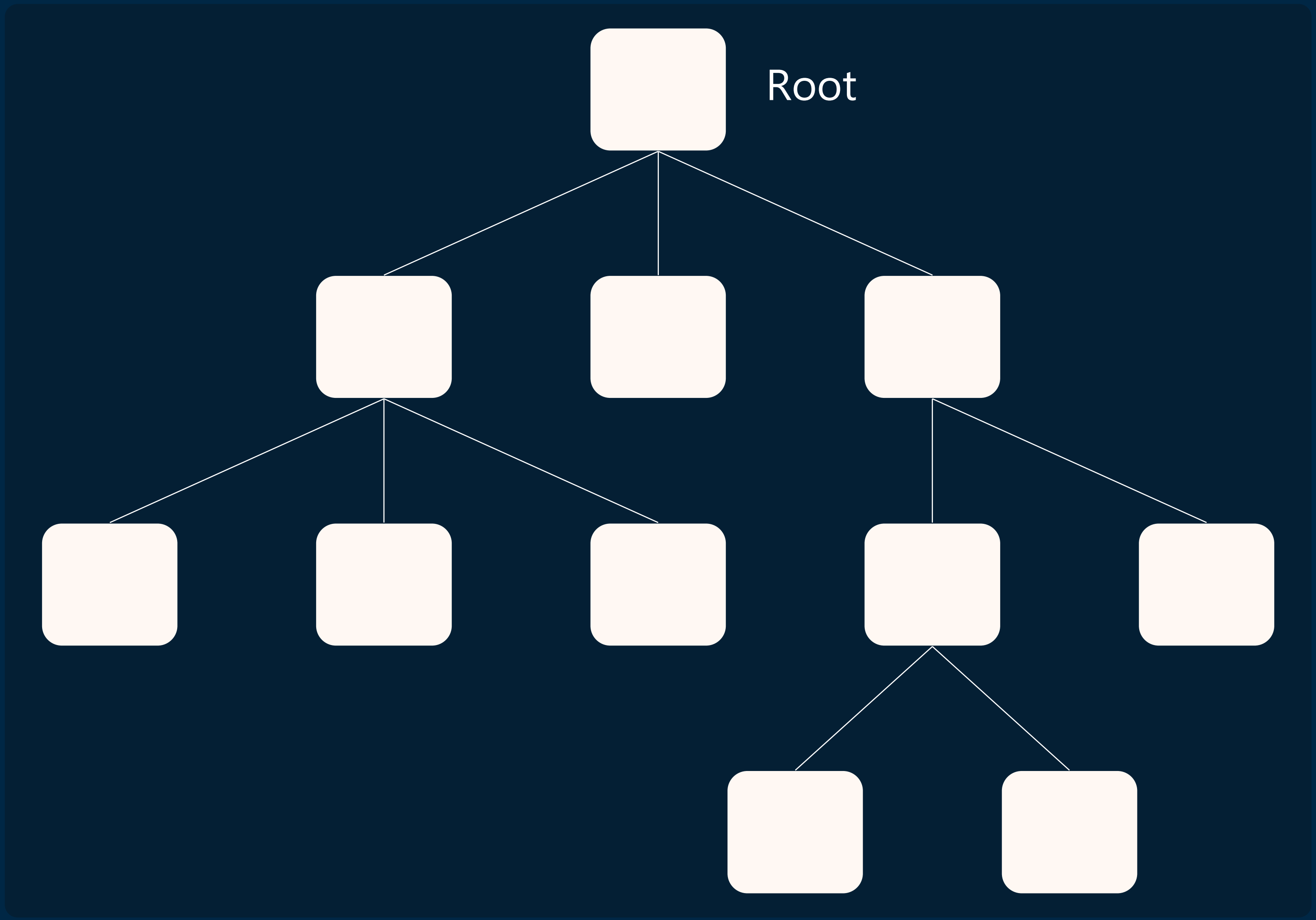
Transactions

- There is no ground truth about the state of the registry, everything is considered in the scope of the specific *alternate reality* (transaction)
- A key may exist in the global view but not in a transaction, and vice versa
  - All aspects of keys may be subject to alternate states: name, subkeys, values, security
- Significant complexity added to all registry code
  - Non-transacted write operations must revert pending transactions concerning the given key
  - Transacted write operations must be careful to avoid collisions with other existing transactions
  - All read operations must correctly incorporate transacted state

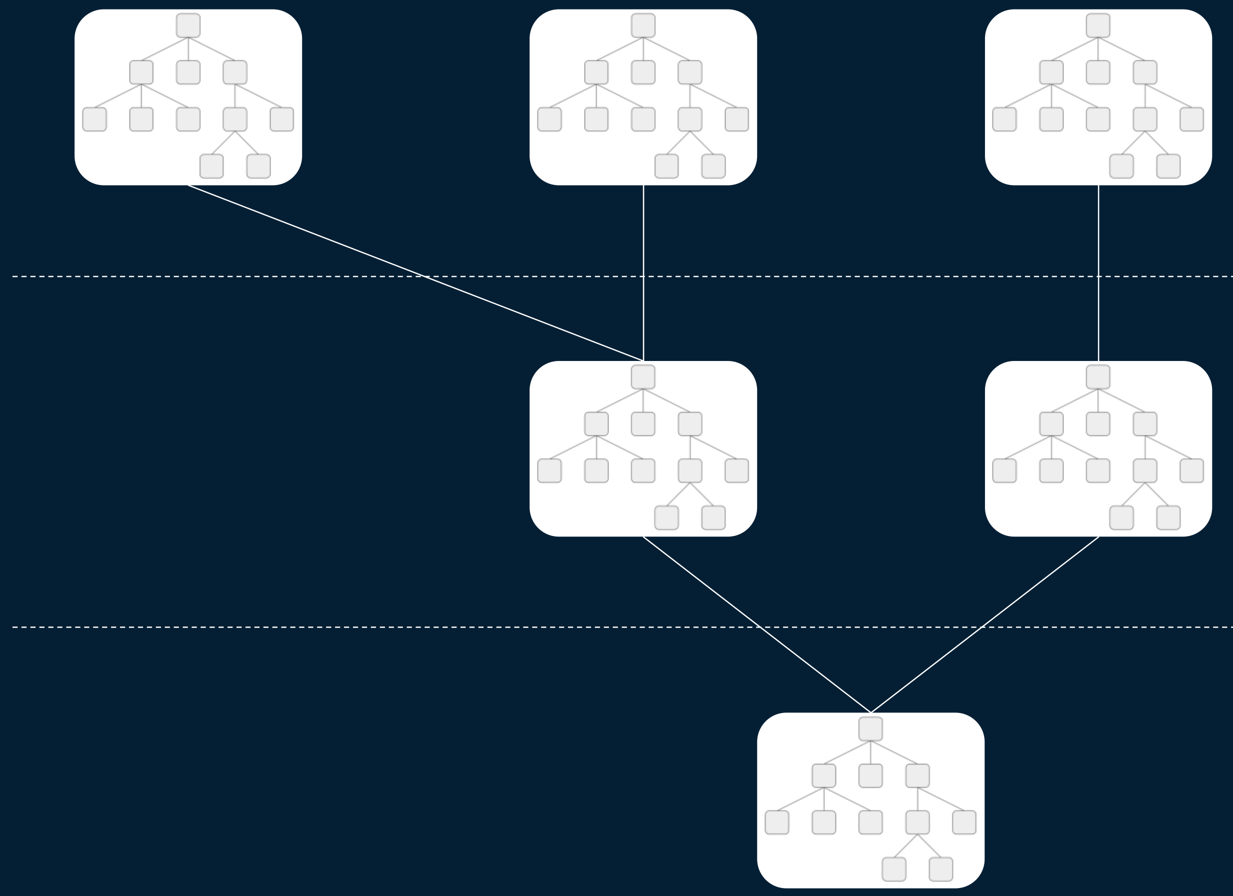
Differencing hives

- Windows 10 1607 added another huge complication to support containers: d*ifferencing hives*
- Normal hives are standalone, self-sufficient databases for storing data
- Delta hives are "patch sets" to be applied to another hive (base or delta)
- They can be stacked on top of each other in case of nested containers
- A key referenced through a differencing hive is a **layered key**

Normal registry tree



Differencing hives tree

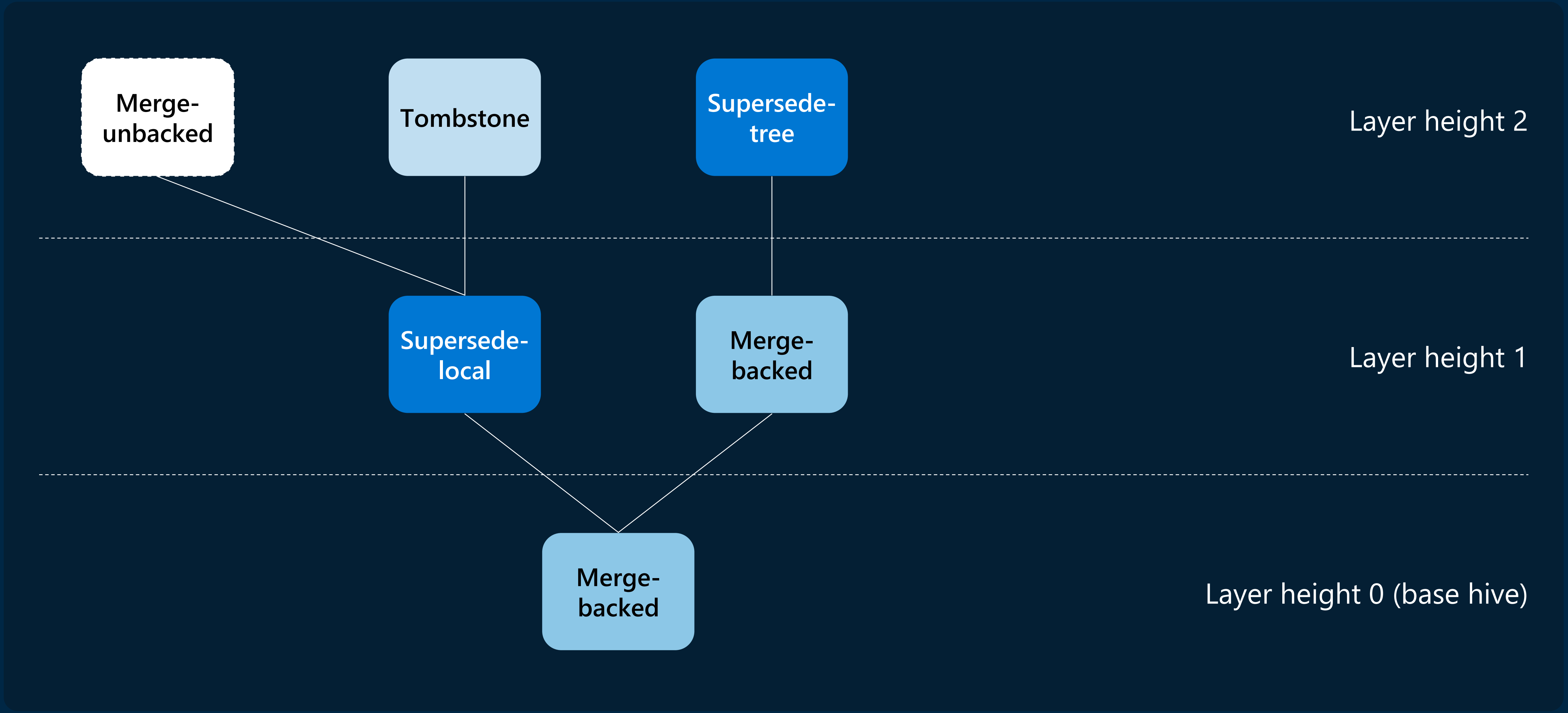


Layer height 2

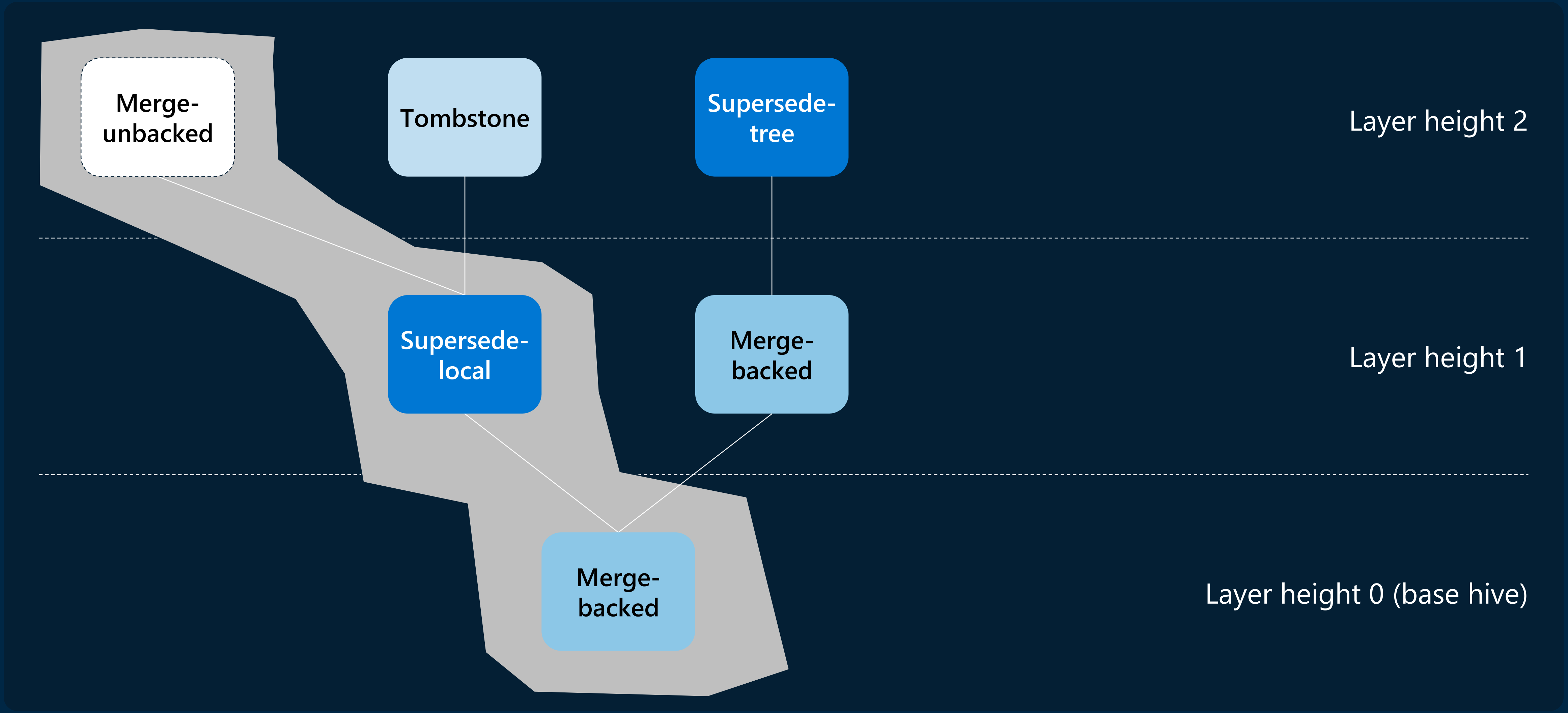
Layer height 1

Layer height 0 (base hive)

Layered key tree



Layered key stack



Layered keys

- Turns everything that we know about the registry upside down
- Every key node is now part of *two* trees in different dimensions
  - Operations like rename have to basically think in 3D
- A key is not represented by a single key: it's now a *key* stack
- The existence of a key node doesn't mean that it exists: see **Tombstones**
- The absence of a key node doesn't mean that it doesn't exist: see **Merge-unbacked** semantics

So how do they all work together?


Predefined Keys
Layered Keys
Transactions
Symbolic Links
Registry Virtualization

Cross-feature bugs

- GPZ-2359: Windows Kernel use-after-free due to bad handling of predefined keys in NtNotifyChangeMultipleKeys
- GPZ-2366: Windows Kernel memory corruption due to insufficient handling of predefined keys in registry virtualization
- GPZ-2375: Windows Kernel multiple issues in the key replication feature of registry virtualization
- GPZ-2389: Windows Kernel registry virtualization incompatible with transactions, leading to inconsistent hive state and memory corruption
- GPZ-2445: Windows Kernel arbitrary read by accessing predefined keys through differencing hives
- GPZ-2446: Windows Kernel may reference unbacked layered keys through registry virtualization
- GPZ-2447: Windows Kernel may reference rolled-back transacted keys through differencing hives
- GPZ-2452: Windows Kernel CmDeleteLayeredKey may delete predefined tombstone keys, leading to security descriptor UAF

Case study: Predefined keys vs the world

- Predefined keys are supposed to be rejected by almost all syscalls
- Otherwise, internal kernel functions are unaware of their semantics and will usually crash when operating on them
- It's necessary to use a safe wrapper to filter them out while referencing a key handle: **`CmObReferenceObjectByHandle`**
- Did it cover all potential scenarios?

Case study: Predefined keys vs the world

Date	GPZ#	Description

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	2359	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	2366	Registry virtualization doesn't go through the safe wrapper

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization doesn't go through the safe wrapper

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[illegible]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	[illegible]	Registry virtualization doesn't go through the safe wrapper
October 2022	2375	Registry virtualization uses unsafe CmpRebuildKcbCache which doesn't refresh the cache of predefined keys

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization uses unsafe CmpRebuildKcbCache which doesn't refresh the cache of predefined keys

Case study: Predefined keys vs the world

Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization uses unsafe CmpRebuildKcbCache which doesn't refresh the cache of predefined keys
April 2023	2445	The "safe" wrappers CmObReferenceObjectByHandle and CmObReferenceObjectByName are insufficient for layered keys

Case study: Predefined keys vs the world

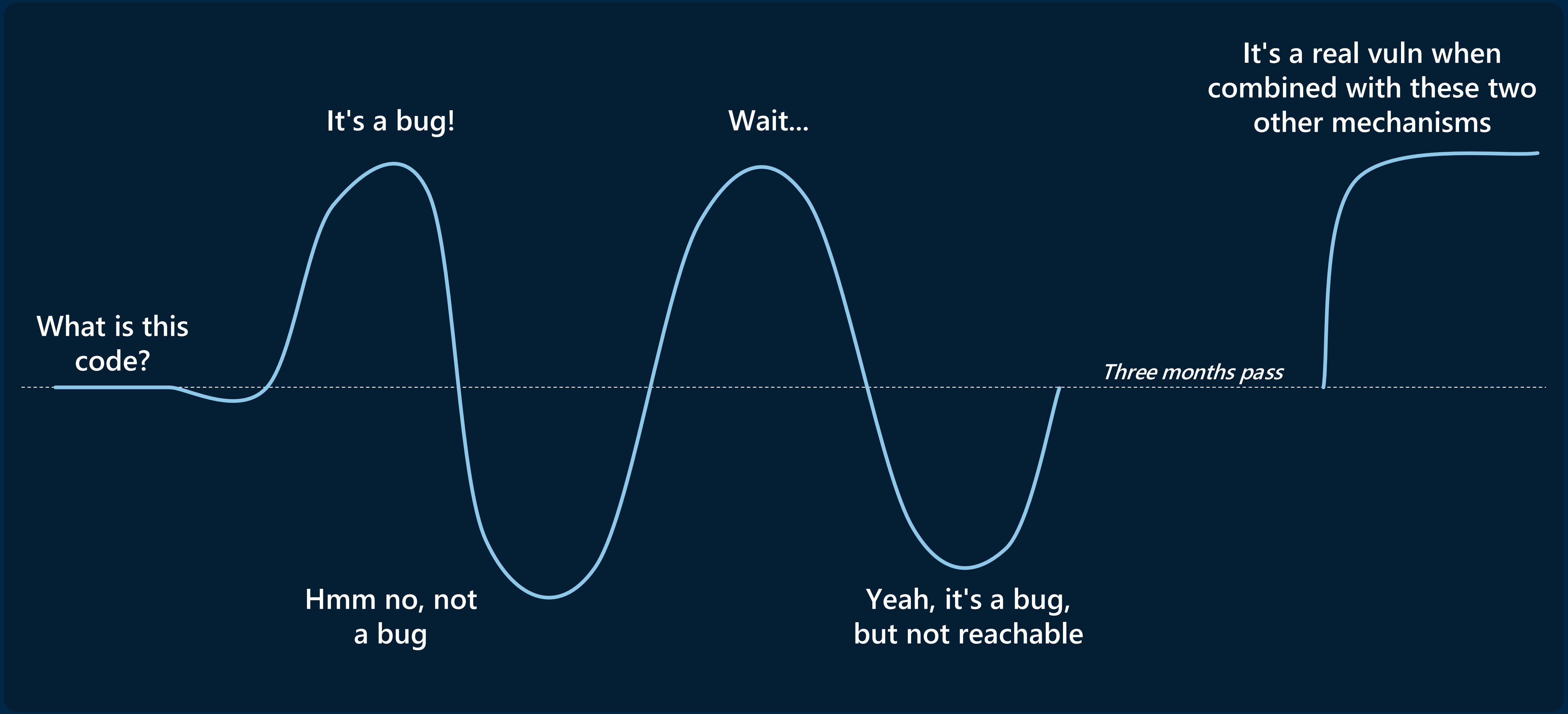
Date	GPZ#	Description
September 2022	[redacted]	NtNotifyChangeMultipleKeys doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization doesn't go through the safe wrapper
October 2022	[redacted]	Registry virtualization uses unsafe CmpRebuildKcbCache which doesn't refresh the cache of predefined keys
April 2023	2445	The "safe" wrappers CmObReferenceObjectByHandle and CmObReferenceObjectByName are insufficient for layered keys
May 2023	2452	CmDeleteLayeredKey bypasses safe wrappers by directly freeing predefined keys via CmpFreeKeyByCell

The ultimate fix

- Finally, GPZ-2445 and GPZ-2452 were fixed in July 2023 by deprecating predefined keys completely
  - Flag 0x40 is cleared in `CmpCheckKey` for every key while loading a hive
- Great to see, as the feature is probably hardly used but was the source of many bugs and much confusion

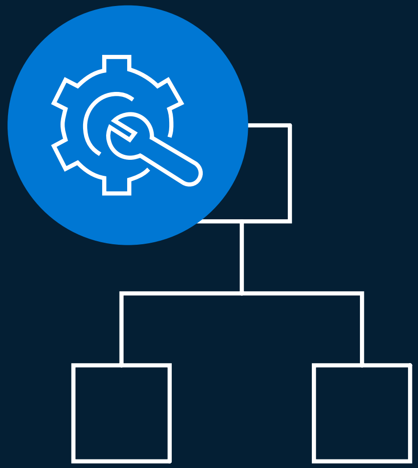
Testing, reproducing, reporting

Testing


It's a real vuln when combined with these two other mechanisms
It's a bug!
Wait...
What is this code?
Three months pass
Hmm no, not a bug
Yeah, it's a bug, but not reachable

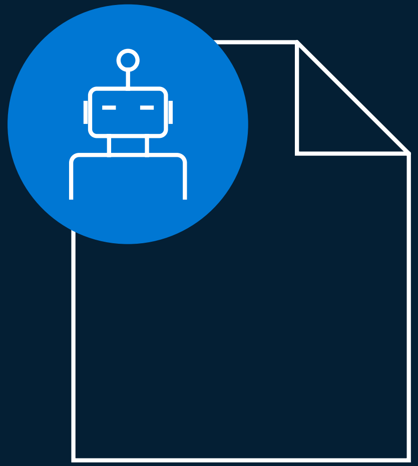
Testing

- This is where you really learn how the registry works
- So many moving pieces that it's almost impossible to be sure of any behavior before testing it
- Tooling:
  - Virtual machines + WinDbg: the !reg extensions are great
  - RegEdit, Process Monitor, Process Explorer
  - Own, custom tools for more advanced stuff: creating symlinks, loading differencing hives etc.



Reproducing the bugs

- I try to trigger a system bugcheck / obvious security violation for every bug
  - Crashes typically don't just happen, deliberate action is needed to demonstrate the corruption
- Registry API is well documented, so writing C++ reproducers is smooth
- Crafting semi-well but unusually formatted hives was the more difficult part
  - Hard to find existing tooling for my specific needs
  - Ended up manually patching the built-in Offline Registry Library (offreg.dll) to produce most of my binary hive PoCs



Reporting

All security bugs filed in the Project Zero bug tracker and submitted to MSRC

All reported bugs successfully fixed within 90+14 days so far

Average time to fix from report until patch publicly available: 81 days

Verifying fixes

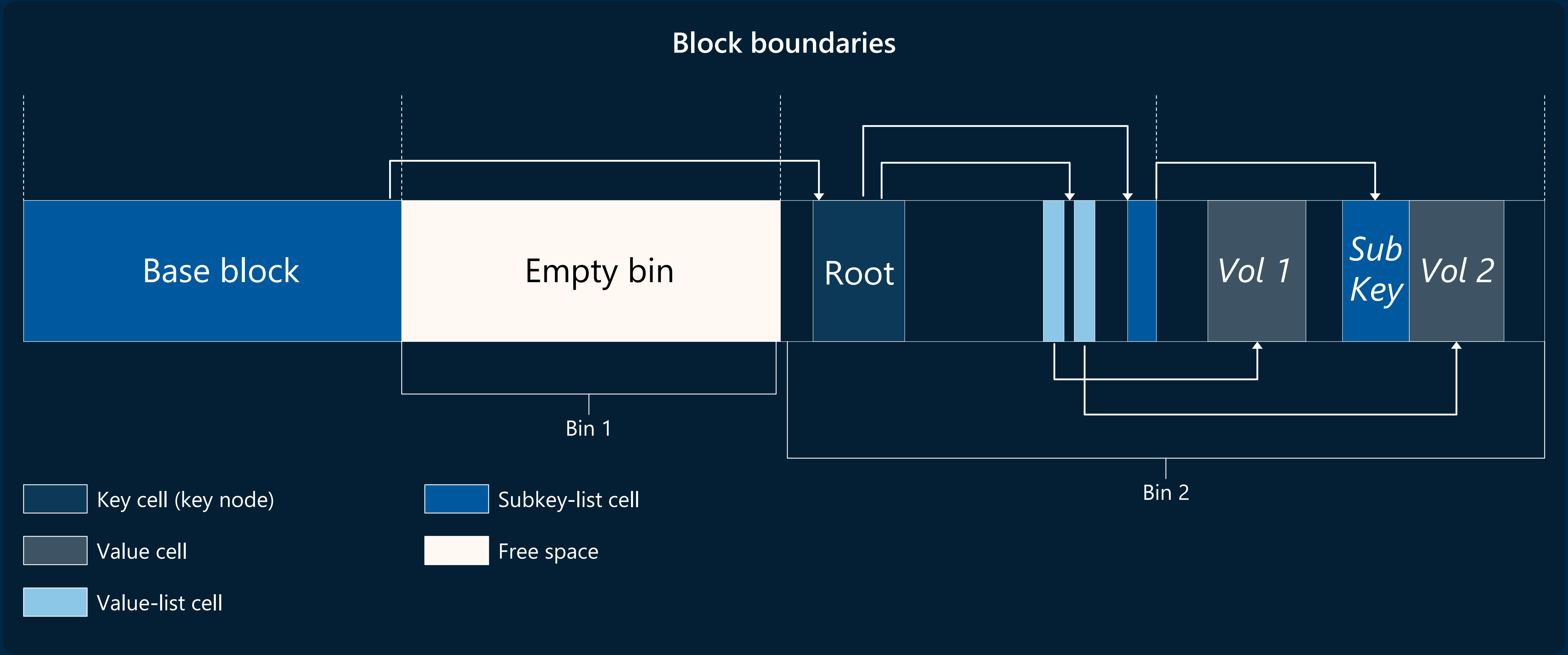
- An optional step, but I try to keep track of all registry changes on a monthly basis
- There's a lot to learn
  - See if the fixes were correct/complete
  - See which avenue was taken – point fix, global code refactoring, something in between?
  - See if any lesser bugs mentioned in the reports were addressed
  - See if any internally found variants I was unaware of were patched
  - See if any (un)related functional changes were made
- Found out about some good work this way
  - Attack surface reduction: KTM transactions, transacted renames, predefined keys
  - Code hardening: Integer overflow checks for security refcounts, rejecting cell index -1 in cell translation code

Bonus: Exploitation

Exploitation

- A huge subject on its own
- Depends largely on the type of bugs and initial primitives
  - Logic issues: usually easiest and most reliable to exploit, but not that many of them in this research (did James find them all?)
  - Pool-based memory corruption: state-of-the-art exploitation techniques apply
  - Hive-based memory corruption: an unexplored class of issues worth investigating further

Registry hive layout



Source: Windows Internals, 7th Edition, Part 2 (A. Allievi, A. Ionescu, M. Russinovich, D. Solomon)

Comparing to heap/pool allocators

Similarities

- Divided into contiguous "free" and "allocated" cells, which are arbitrarily sized chunks of data
- Some cells have a predefined structure (e.g. key nodes), while others contain 100% user-specified data (e.g. value data)

Differences

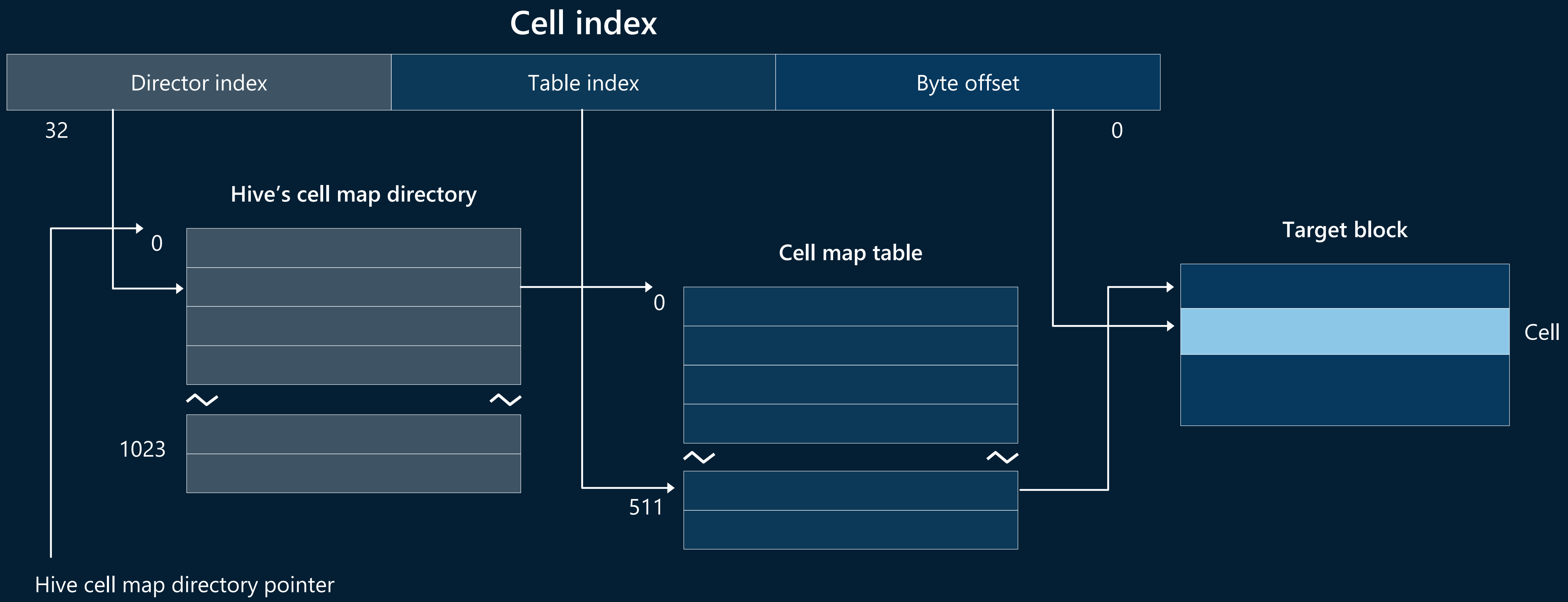
- Exact same data layout maintained on disk and in memory
- No randomization (100% deterministic) or any protection against temporal/spatial violations
- Most bugs allow reliably replacing/corrupting arbitrary objects in the hive

So what do we corrupt?

- Corrupting our own hive gets us nowhere
- At first glance, there are no pointers or anything to take us "outside" of the hive
- The solution: cell indexes
  - 32-bit unsigned integers used to reference cells between each other
  - On disk: simple offsets within the hive file
  - In memory: offsets into a multi-level page table-like structure (cell maps)
  - At runtime, the `HvpGetCellPaged` function is used for the translation:

```
HvpGetCellPaged(uint32 CellIndex) → void* VirtualAddress
```

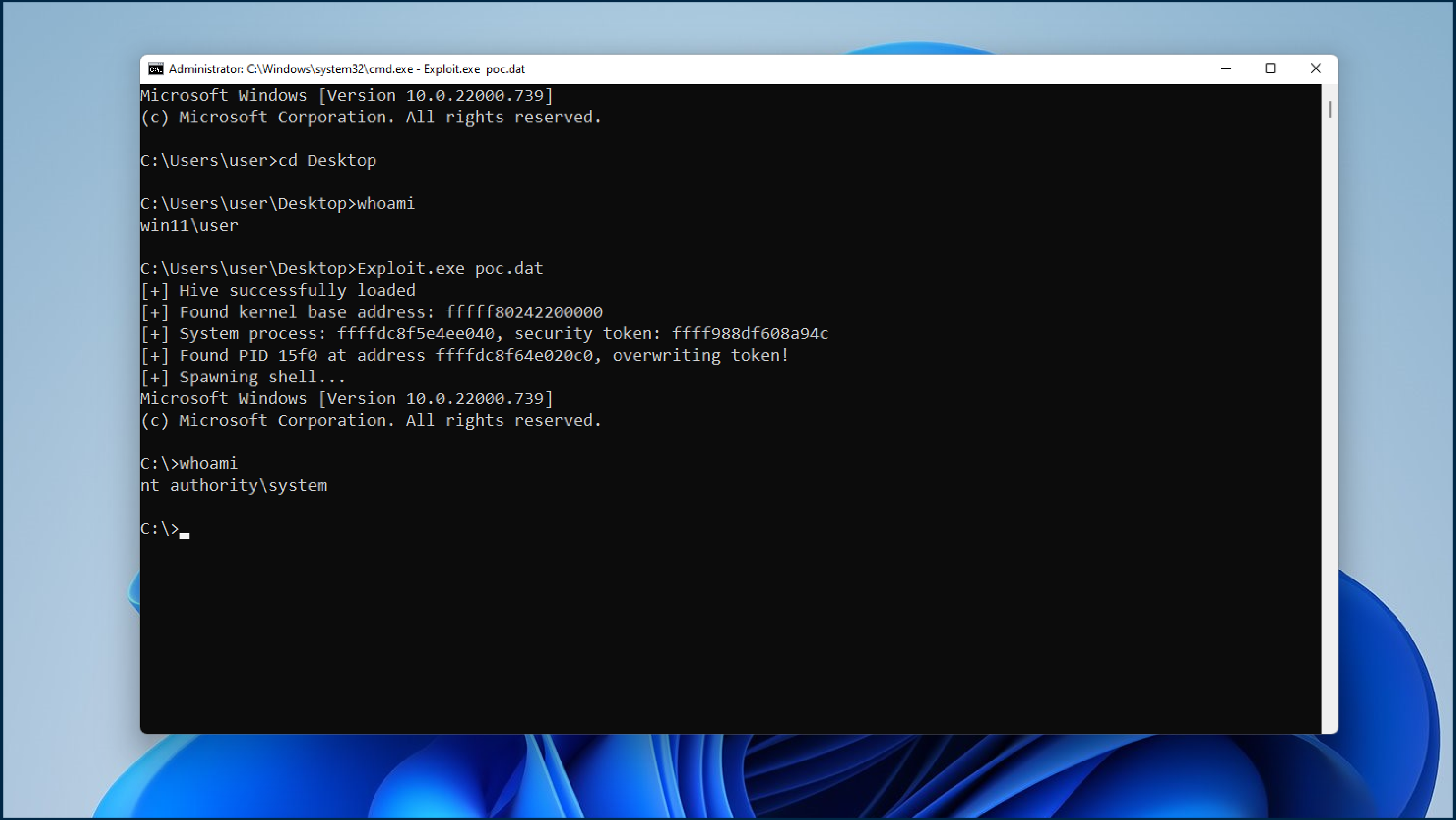
Cell maps and cell indexes



Source: Windows Internals, 7th Edition, Part 2 (A. Allievi, A. Ionescu, M. Russinovich, D. Solomon)

Out-of-bounds cell indexes

- Due to how cell maps are allocated, an OOB index can be abused to point to:
  - An arbitrary address
  - Specific objects in memory
  - Itself (self-referential cell index)
- Not quite Turing-complete, but firmly in the category of a "weird machine"
- Provides an address leak and arbitrary read/write, all with one bug
- Enables a reliable, data-only LPE attack


Administrator: C:\Windows\system32\cmd.exe - Exploit.exe poc.dat
Microsoft Windows [Version 10.0.22000.739]
(c) Microsoft Corporation. All rights reserved.
C:\Users\user>cd Desktop
C:\Users\user\Desktop>whoami
win11\user
C:\Users\user\Desktop>Exploit.exe poc.dat
[+] Hive successfully loaded
[+] Found kernel base address: fffff80242200000
[+] System process: ffffdc8f5e4ee040, security token: ffff988df608a94c
[+] Found PID 15f0 at address ffffdc8f64e020c0, overwriting token!
[+] Spawning shell...
Microsoft Windows [Version 10.0.22000.739]
(c) Microsoft Corporation. All rights reserved.
C:\>whoami
nt authority\system
C:\>

Takeaways

- The registry is a fascinating research target, but has been publicly underexplored throughout its history
- If you're a researcher: deep, persistent analysis pays off
- If you're a software vendor: attack surface reduction, elimination of entire bug classes and well-placed mitigations have an outsized impact on security

Microsoft

MSRC BLUEHAT OCT '23