Peeling Back the Windows Registry Layers: A Bug Hunter's Expedition

Mateusz Jurczyk

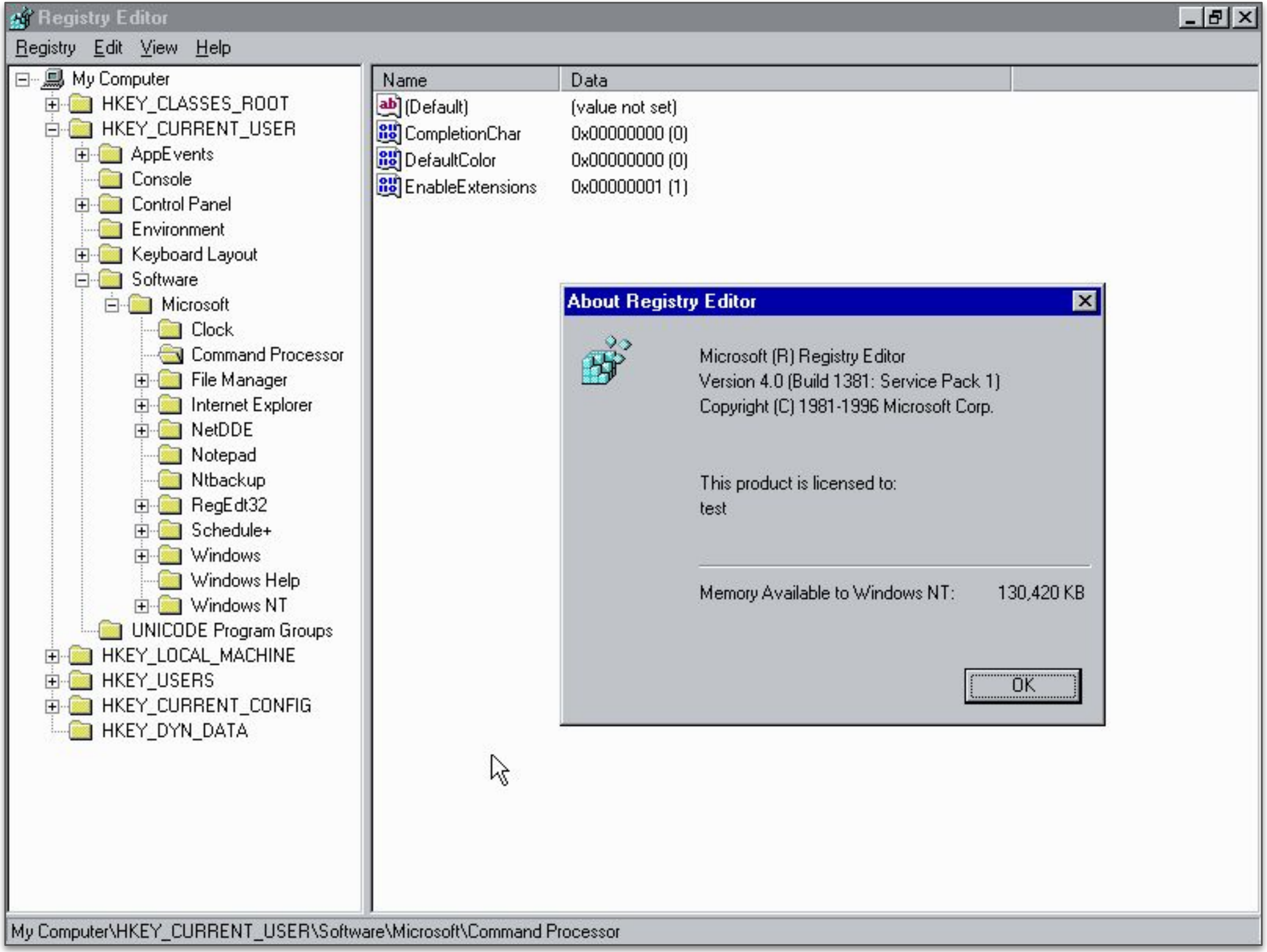
REcon, June 2024

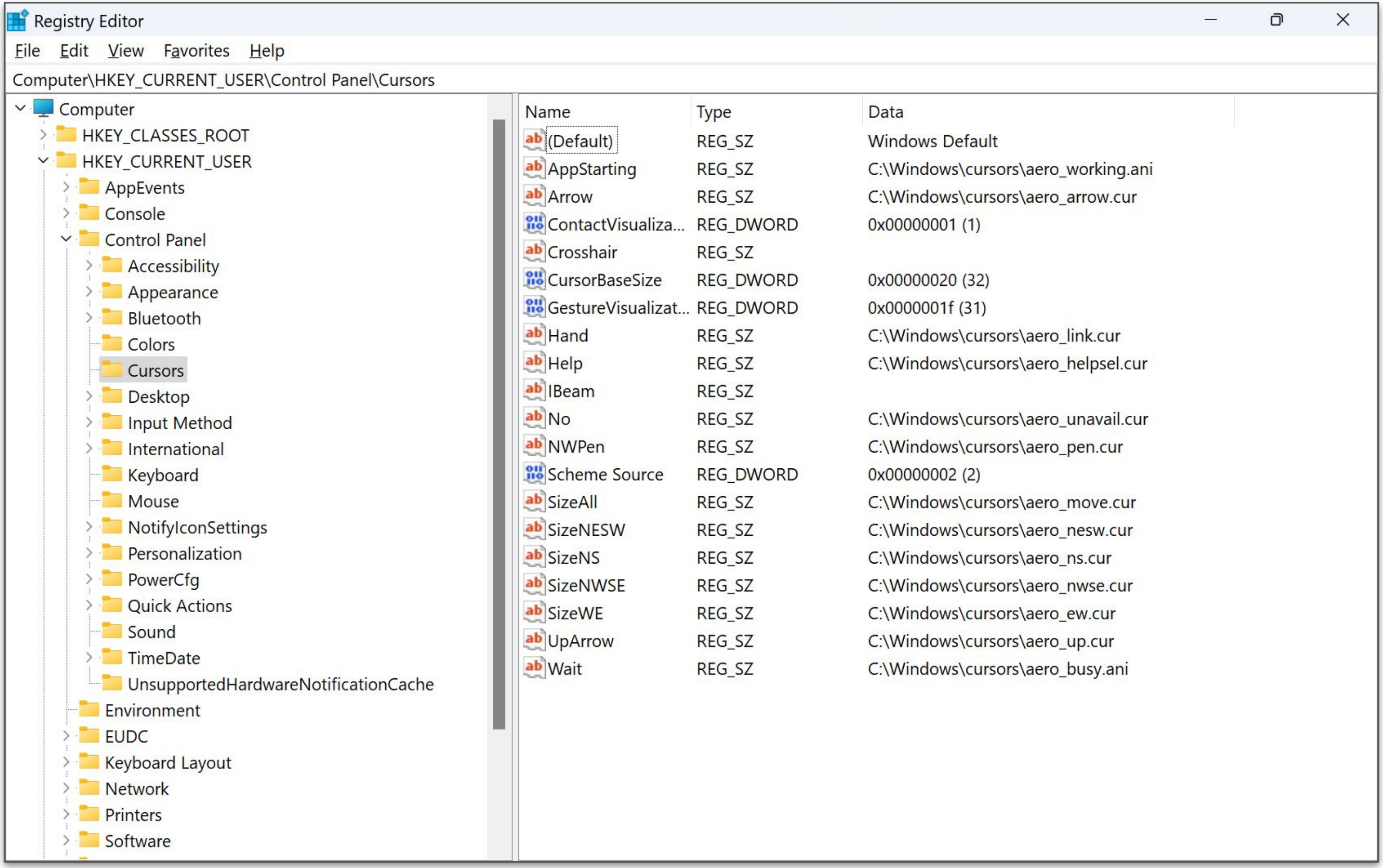
The registry fundamentals

- A hierarchical database for storing system/application settings in Windows
- Essential concepts: **hives**, **keys** and **values**
- Built-in tools for management: **Regedit.exe** (GUI), **Reg.exe** (CLI)
- Documented Registry API for software developers
- Most of the implementation is in the kernel

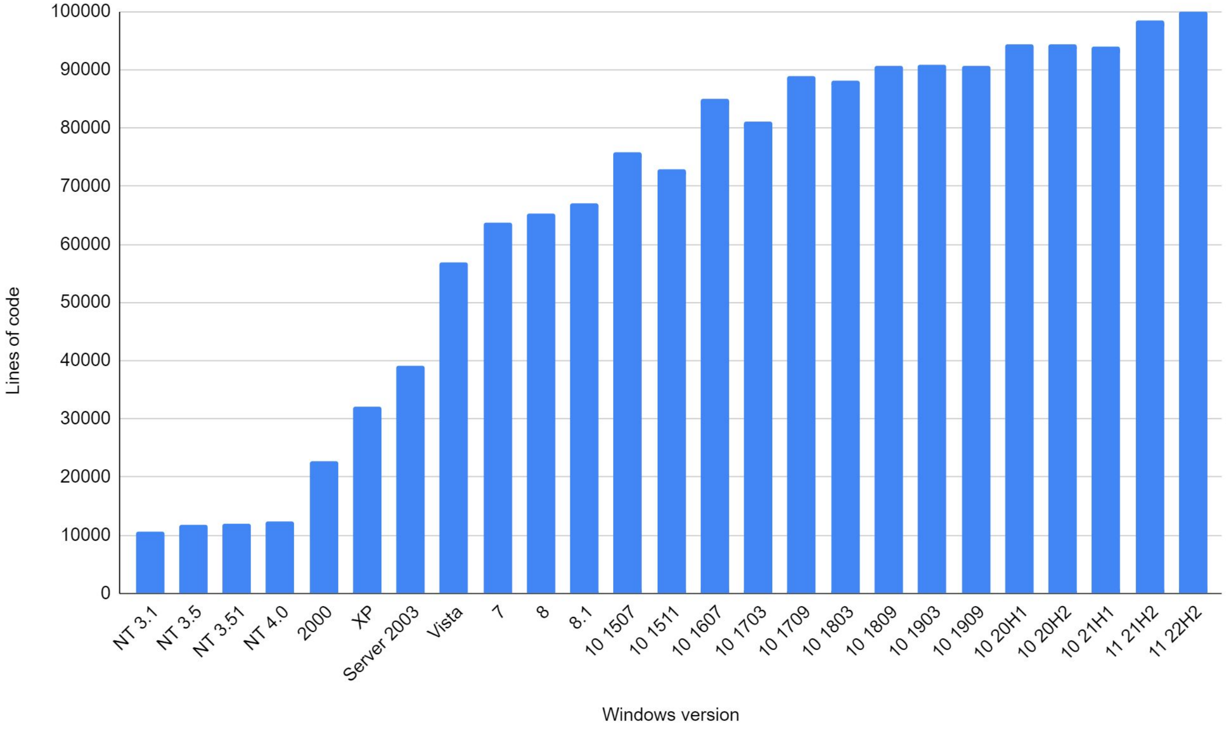
A bit of history

- First introduced in **Windows 3.1** (1992) to replace INI files
- Current code and design directly rooted in **Windows NT 3.1** (1993) and **Windows NT 4.0** (1996)
- Started out small, then extended and improved over the next 30 years
  - Performance improvements: *faster subkey lookups, optimized key renaming*
  - Backwards compatibility: *registry virtualization*
  - New features: *big values, registry callbacks, transactions, application hives, differencing hives*


Registry Editor
Registry Edit View Help
My Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
AppEvents
Console
Control Panel
Environment
Keyboard Layout
Software
Microsoft
Clock
Command Processor
File Manager
Internet Explorer
NetDDE
Notepad
Ntbackup
RegEdt32
Schedule+
Windows
Windows Help
Windows NT
UNICODE Program Groups
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG
HKEY_DYN_DATA
Name Data
(Default) (value not set)
CompletionChar 0x00000000 (0)
DefaultColor 0x00000000 (0)
EnableExtensions 0x00000001 (1)
About Registry Editor
Microsoft (R) Registry Editor
Version 4.0 (Build 1381: Service Pack 1)
Copyright (C) 1981-1996 Microsoft Corp.
This product is licensed to:
test
Memory Available to Windows NT: 130,420 KB
OK
My Computer\HKEY_CURRENT_USER\Software\Microsoft\Command Processor


Registry Editor
File Edit View Favorites Help
Computer\HKEY_CURRENT_USER\Control Panel\Cursors
Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
AppEvents
Console
Control Panel
Accessibility
Appearance
Bluetooth
Colors
Cursors
Desktop
Input Method
International
Keyboard
Mouse
NotifyIconSettings
Personalization
PowerCfg
Quick Actions
Sound
TimeDate
UnsupportedHardwareNotificationCache
Environment
EUDC
Keyboard Layout
Network
Printers
Software
Name Type Data
(Default) REG_SZ Windows Default
AppStarting REG_SZ C:\Windows\cursors\aero_working.ani
Arrow REG_SZ C:\Windows\cursors\aero_arrow.cur
ContactVisualiza... REG_DWORD 0x00000001 (1)
Crosshair REG_SZ
CursorBaseSize REG_DWORD 0x00000020 (32)
GestureVisualizat... REG_DWORD 0x0000001f (31)
Hand REG_SZ C:\Windows\cursors\aero_link.cur
Help REG_SZ C:\Windows\cursors\aero_helpsel.cur
IBeam REG_SZ
No REG_SZ C:\Windows\cursors\aero_unavail.cur
NWPen REG_SZ C:\Windows\cursors\aero_pen.cur
Scheme Source REG_DWORD 0x00000002 (2)
SizeAll REG_SZ C:\Windows\cursors\aero_move.cur
SizeNESW REG_SZ C:\Windows\cursors\aero_nesw.cur
SizeNS REG_SZ C:\Windows\cursors\aero_ns.cur
SizeNWSE REG_SZ C:\Windows\cursors\aero_nwse.cur
SizeWE REG_SZ C:\Windows\cursors\aero_ew.cur
UpArrow REG_SZ C:\Windows\cursors\aero_up.cur
Wait REG_SZ C:\Windows\cursors\aero_busy.ani

Lines of decompiled kernel code



Registry as an attack surface: the good

👍 Ability to load custom hives as an unprivileged user

👍 Access to sensitive data: system configuration, user credentials

👍 Error prone parts of the design: self-healing, size-bound, heavily optimized

👍 A mixture of complex C code from different eras: from 30 years ago to now

👍 A variety of potential bug classes and attack vectors

Registry as an attack surface: the bad*

- 👎 Very hard to fuzz effectively
- 👎 Source code not available, and documentation is poor for specific areas
- 👎 Public symbols incomplete, lack some type definitions
- 👎 Lots of reverse engineering required: significant time and energy investment
- 👎 Not all bugs are good, as usual

How the research started

- Started in May 2022 as a test of my new coverage-based fuzzer for the Windows kernel
- Found one bug: **CVE-2022-35768**
    - *Windows Kernel multiple memory problems when handling incorrectly formatted security descriptors in registry hives*
- The initial success prompted me to have a deeper look into the kernel
- It quickly turned into a challenge to reverse and review *all* of the code...

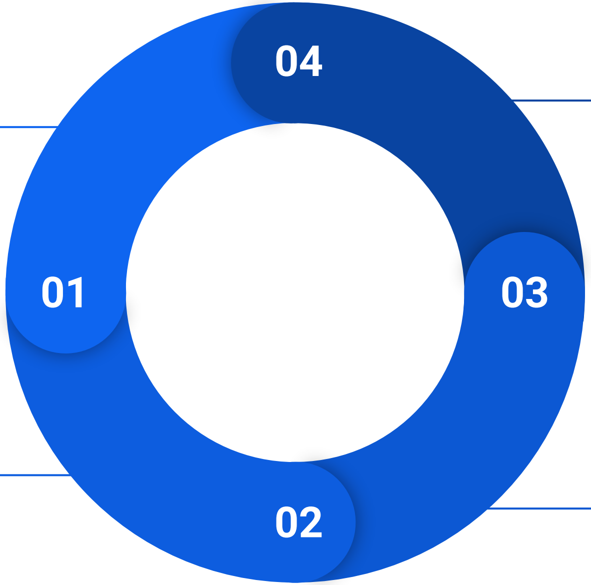
The research process

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.



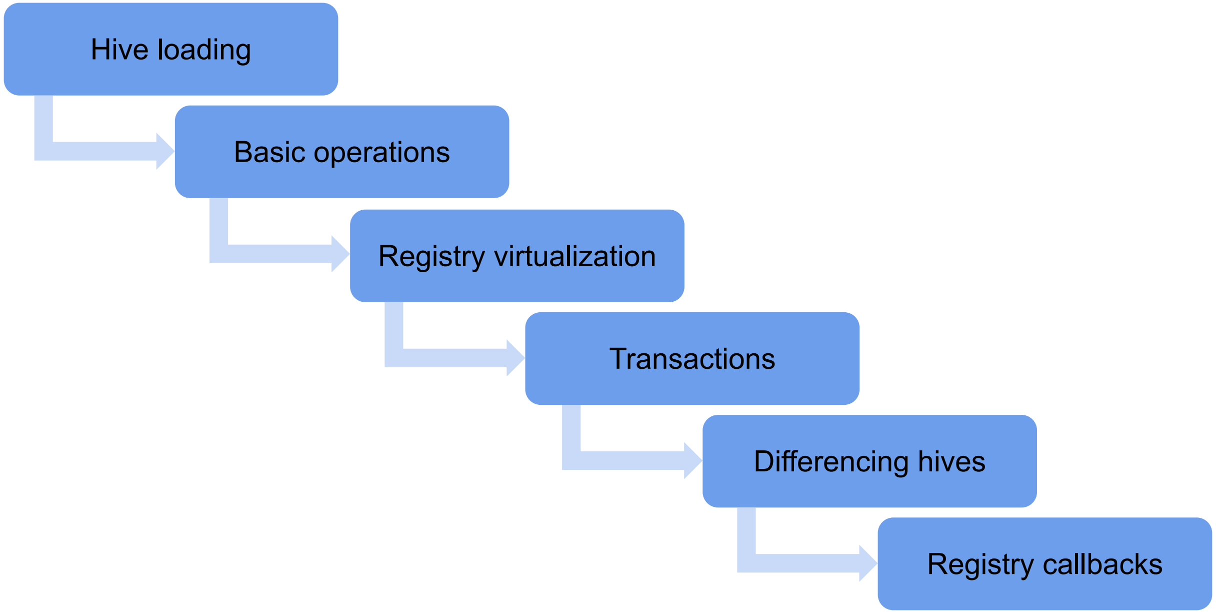
Test, reproduce, report bugs

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

Compare with prior knowledge

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

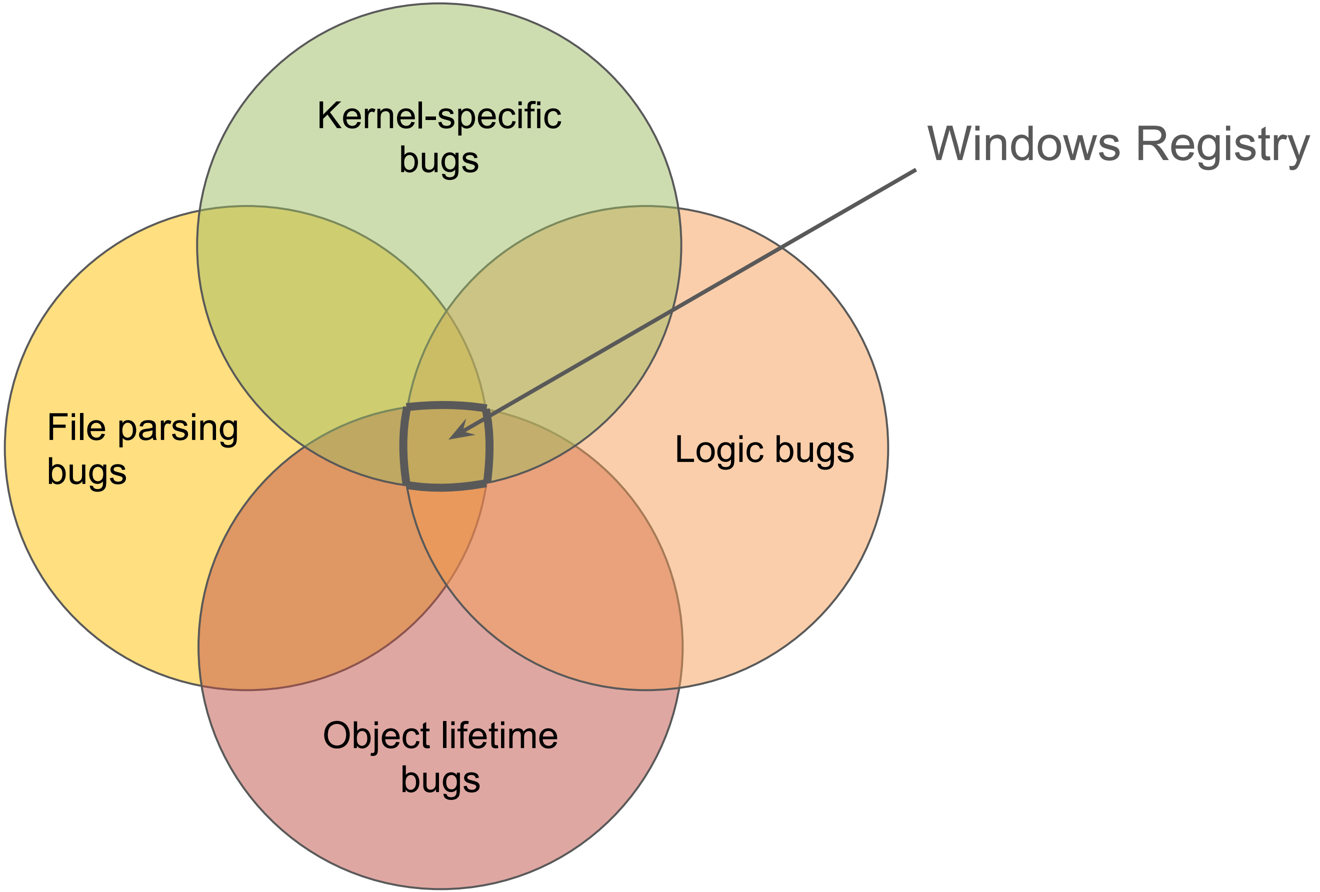
Research progression: major features



How it went

- The audit lasted for ~20 months between May 2022 – December 2023
- Results:
  - **39 issues** reported in the Project Zero bug tracker (under a 90 day deadline)
  - **20 issues** reported outside the tracker (no deadline, low/unclear severity)
  - **= 50 CVEs** assigned by Microsoft across 15 monthly bulletins

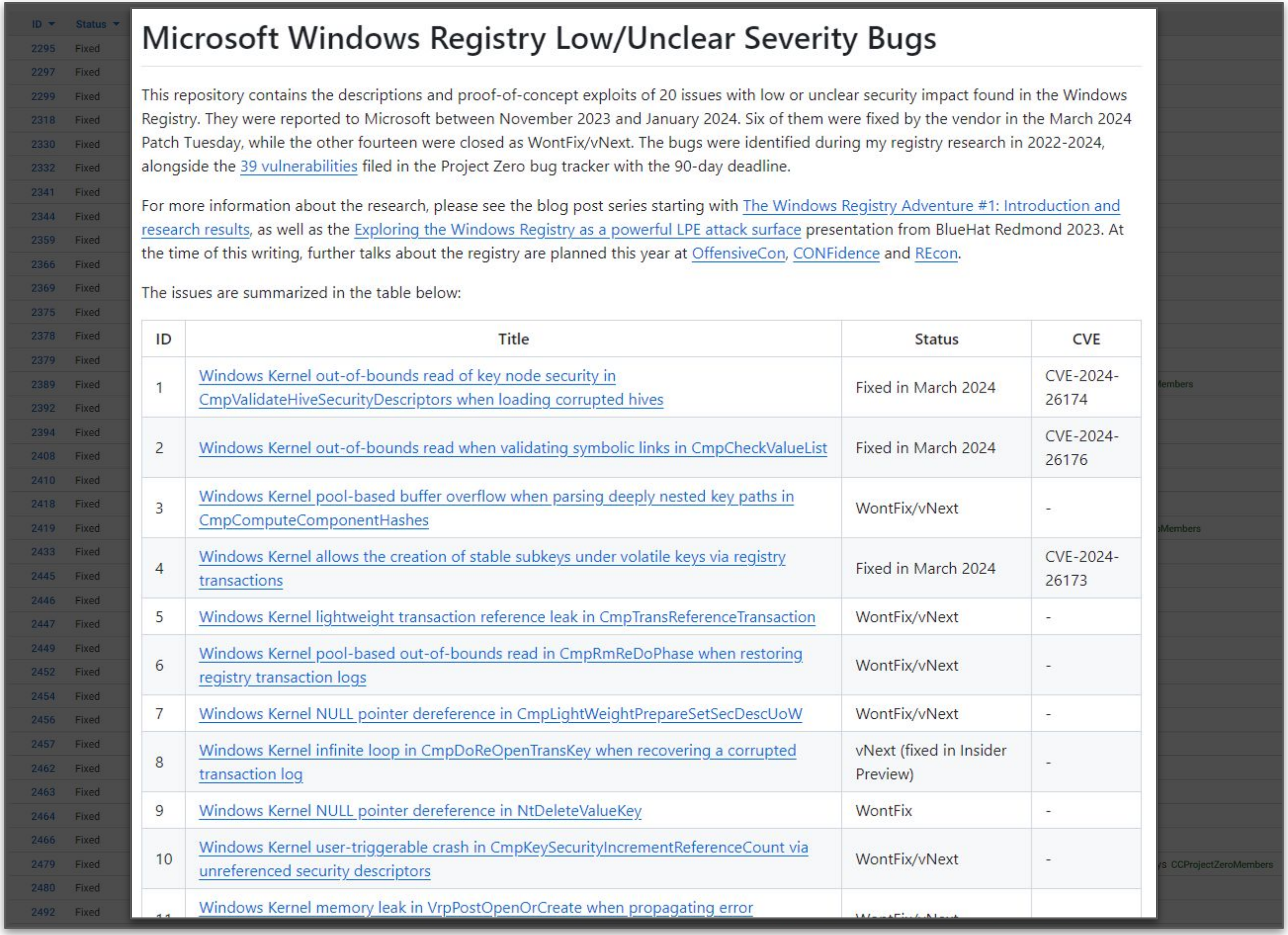
Bug classes



Kernel-specific bugs
Windows Registry
File parsing bugs
Logic bugs
Object lifetime bugs

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
2457	Fixed	----	2023-May-31	Microsoft	Kernel	mjurczyk	Windows Kernel doesn't reset security cache during self-healing, leading to refcount overflow and UAF CCProjectZeroMembers
2462	Fixed	----	2023-Jun-26	Microsoft	Kernel	mjurczyk	Windows Kernel passes user-mode pointers to registry callbacks, leading to race conditions and memory corruption CCProjectZeroMembers
2463	Fixed	----	2023-Jun-27	Microsoft	Kernel	mjurczyk	Windows Kernel paged pool memory disclosure in VrpPostEnumerateKey CCProjectZeroMembers
2464	Fixed	----	2023-Jun-27	Microsoft	Kernel	mjurczyk	Windows Kernel out-of-bounds reads and paged pool memory disclosure in VrpUpdateKeyInformation CCProjectZeroMembers
2466	Fixed	----	2023-Jul-7	Microsoft	Kernel	mjurczyk	Windows Kernel containerized registry escape through integer overflows in VrpBuildKeyPath and other weaknesses CCProjectZeroMembers
2479	Fixed	----	2023-Aug-10	Microsoft	Kernel	mjurczyk	Windows Kernel time-of-check/time-of-use issue in verifying layered key security may lead to information disclosure from privileged registry keys CCProjectZeroMembers
2480	Fixed	----	2023-Aug-22	Microsoft	Kernel	mjurczyk	Windows Kernel bad locking in registry virtualization leads to race conditions CCProjectZeroMembers
2492	Fixed	----	2023-Oct-6	Microsoft	Kernel	mjurczyk	Windows registry predefined keys may lead to confused deputy problems and local privilege escalation CCProjectZeroMembers

https://bugs.chromium.org/p/project-zero/issues/list?q=finder%3Amjurczyk%20opened%3E2022-05-01%20opened%3C2024-01-01&can=1



Microsoft Windows Registry Low/Unclear Severity Bugs

This repository contains the descriptions and proof-of-concept exploits of 20 issues with low or unclear security impact found in the Windows Registry. They were reported to Microsoft between November 2023 and January 2024. Six of them were fixed by the vendor in the March 2024 Patch Tuesday, while the other fourteen were closed as WontFix/vNext. The bugs were identified during my registry research in 2022-2024, alongside the 39 vulnerabilities filed in the Project Zero bug tracker with the 90-day deadline.

For more information about the research, please see the blog post series starting with The Windows Registry Adventure #1: Introduction and research results, as well as the Exploring the Windows Registry as a powerful LPE attack surface presentation from BlueHat Redmond 2023. At the time of this writing, further talks about the registry are planned this year at OffensiveCon, CONFidence and REcon.

The issues are summarized in the table below:

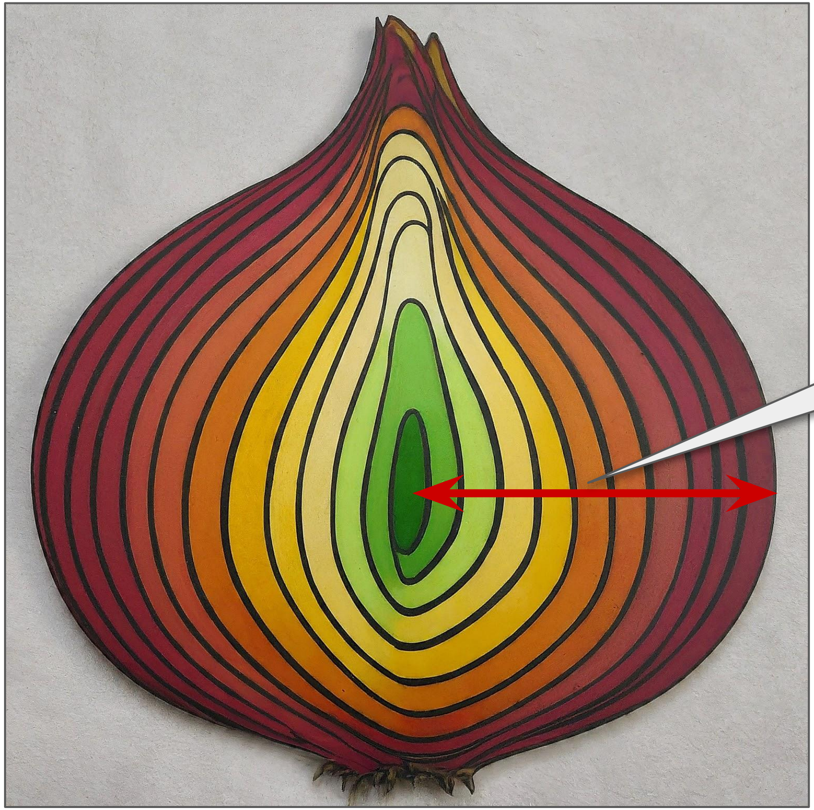
ID	Title	Status	CVE
1	Windows Kernel out-of-bounds read of key node security in CmpValidateHiveSecurityDescriptors when loading corrupted hives	Fixed in March 2024	CVE-2024-26174
2	Windows Kernel out-of-bounds read when validating symbolic links in CmpCheckValueList	Fixed in March 2024	CVE-2024-26176
3	Windows Kernel pool-based buffer overflow when parsing deeply nested key paths in CmpComputeComponentHashes	WontFix/vNext	-
4	Windows Kernel allows the creation of stable subkeys under volatile keys via registry transactions	Fixed in March 2024	CVE-2024-26173
5	Windows Kernel lightweight transaction reference leak in CmpTransReferenceTransaction	WontFix/vNext	-
6	Windows Kernel pool-based out-of-bounds read in CmpRmReDoPhase when restoring registry transaction logs	WontFix/vNext	-
7	Windows Kernel NULL pointer dereference in CmpLightWeightPrepareSetSecDescUoW	WontFix/vNext	-
8	Windows Kernel infinite loop in CmpDoReOpenTransKey when recovering a corrupted transaction log	vNext (fixed in Insider Preview)	-
9	Windows Kernel NULL pointer dereference in NtDeleteValueKey	WontFix	-
10	Windows Kernel user-triggerable crash in CmpKeySecurityIncrementReferenceCount via unreferenced security descriptors	WontFix/vNext	-
11	Windows Kernel memory leak in VrpPostOpenOrCreate when propagating error		

https://github.com/googleprojectzero/p0tools/tree/master/WinRegLowSeverityBugs

Lessons learned

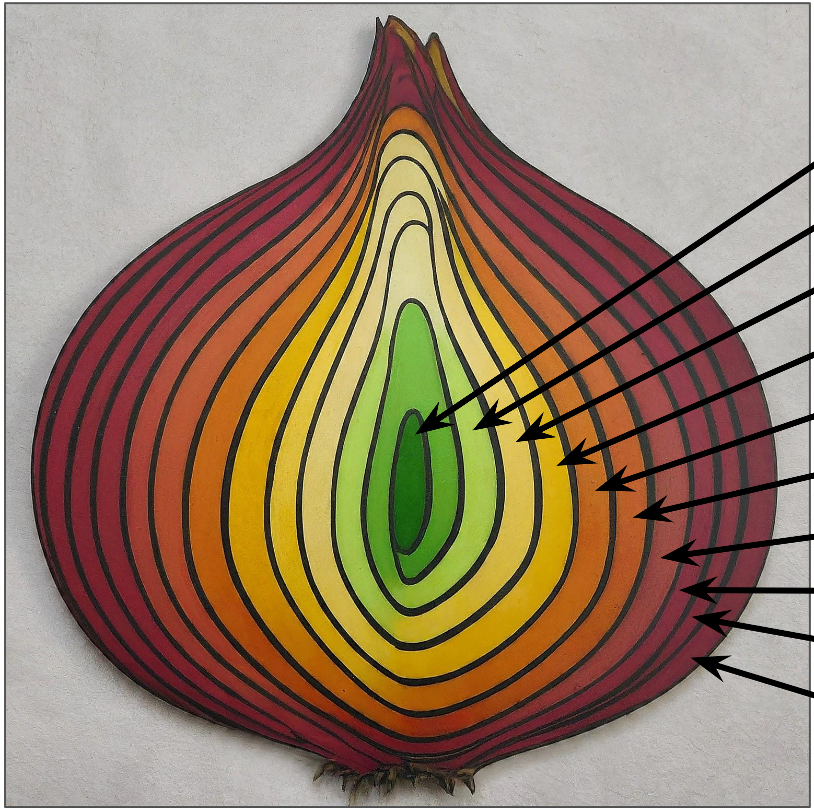
- Software continues to have bugs 🤯
- Different bugs lie at different points on the code understanding scale 📏
- Security research is akin to peeling back the layers of an onion 🧅

A taxon(ion)omy of bugs



Level of context required
(easiest to hardest to find?)

A taxonomy of bugs



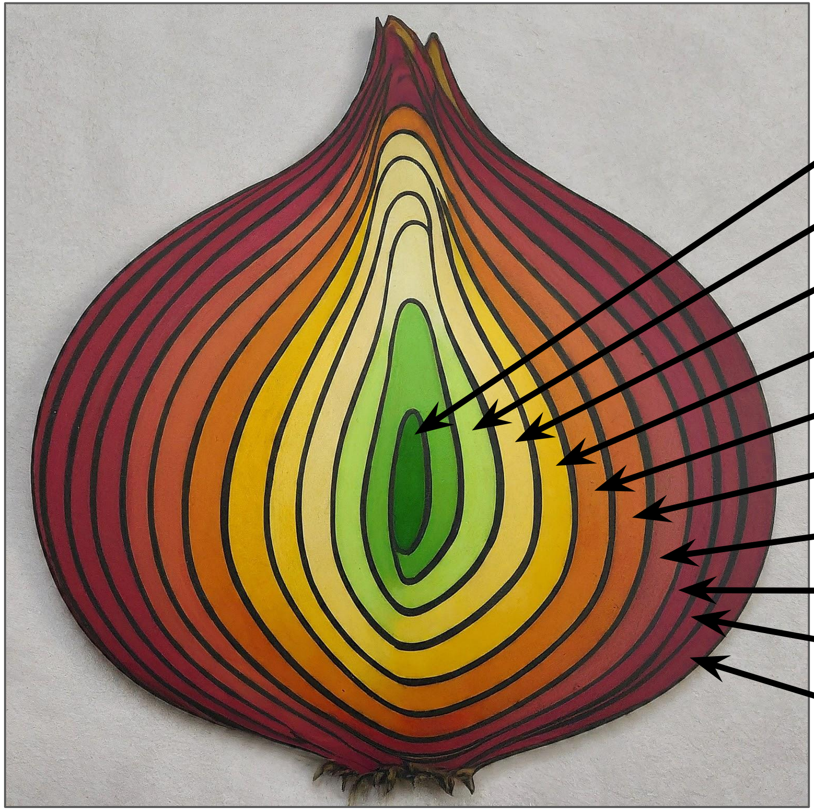
🥇 Logic bugs

🥈 Cross-feature bugs

🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

A taxonomy of bugs



🥇 Logic bugs

🥈 Cross-feature bugs

🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Fuzzable bugs

- Virtually zero knowledge of the target required, only its behavior and examples of inputs:
  - How to build it (optional)
  - How to run it and pass input data
  - How it fails/crashes

Registry fuzzing – easy in theory

- Hives are a binary format
- Input samples readily available in Windows
- Initial harness easy to write
  - RegLoadAppKey + RegCloseKey
- Simple bug detection
  - Catching BSoDs / unexpected reboots

Registry fuzzing – hard in practice

- Hives are a binary format
- Input samples readily available in Windows
- Initial harness easy to write
  - RegLoadAppKey + RegCloseKey
- Simple bug detection
  - Catching BSoDs / unexpected reboots

Registry fuzzing – hard in practice

- ~~Hives are a binary format~~
- Input samples readily available in Windows
- Initial harness easy to write
  - RegLoadAppKey + RegCloseKey
- Simple bug detection
  - Catching BSoDs / unexpected reboots

- Structurally very simple
- Most interesting bugs occur on a higher level
- Bitflipping can only trigger the lowest hanging fruit

Registry fuzzing – hard in practice

- ~~Hives are a binary format~~
- ~~Input samples readily available in Windows~~
- Initial harness easy to write
  - RegLoadAppKey + RegCloseKey
- Simple bug detection
  - Catching BSoDs / unexpected reboots

Default hives don't contain any interesting/non-standard constructs

Registry fuzzing – hard in practice

- ~~Hives are a binary format~~
- ~~Input samples readily available in Windows~~
- ~~Initial harness easy to write~~
  - ~~RegLoadAppKey + RegCloseKey~~
- Simple bug detection
  - Catching BSoDs / unexpected reboots

- This only covers a very small part of the registry
- Dozens of other operations required to properly test the code

Registry fuzzing – hard in practice

- ~~Hives are a binary format~~
- ~~Input samples readily available in Windows~~
- ~~Initial harness easy to write~~
  - ~~RegLoadAppKey + RegCloseKey~~
- ~~Simple bug detection~~
  - ~~Catching BSoDs / unexpected reboots~~

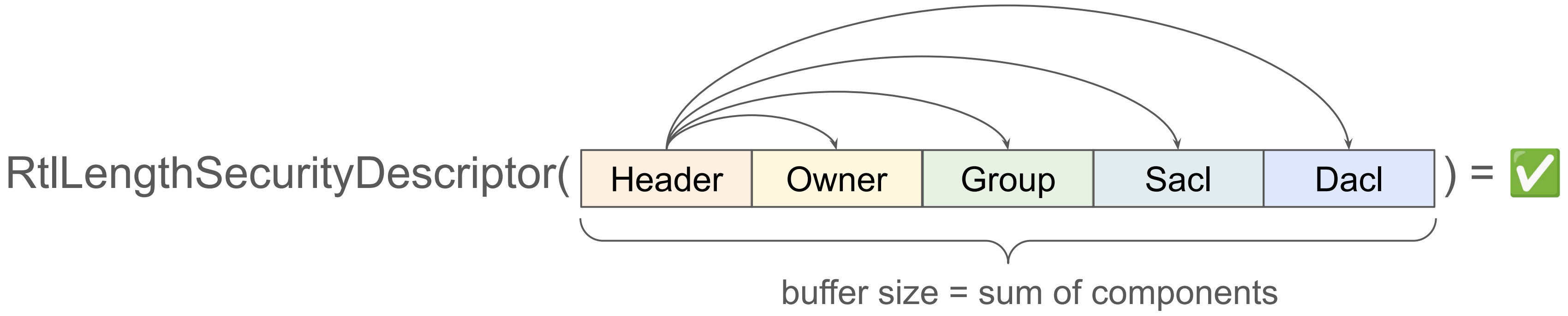
Most registry bugs don't trigger hard crashes

- Hive memory corruption
- Logic bugs

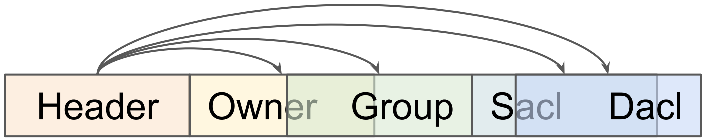
Registry fuzzing – hard in practice

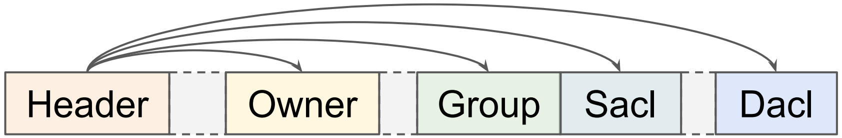
- This might explain why only 1 bug was fuzzed out
    - Miscalculation of a security descriptor buffer size
    - Trivial leak of OOB kernel pool data into the hive file
    - Likely wouldn't have been found manually
- ... and why the other 49 survived for so long

CVE-2022-35768

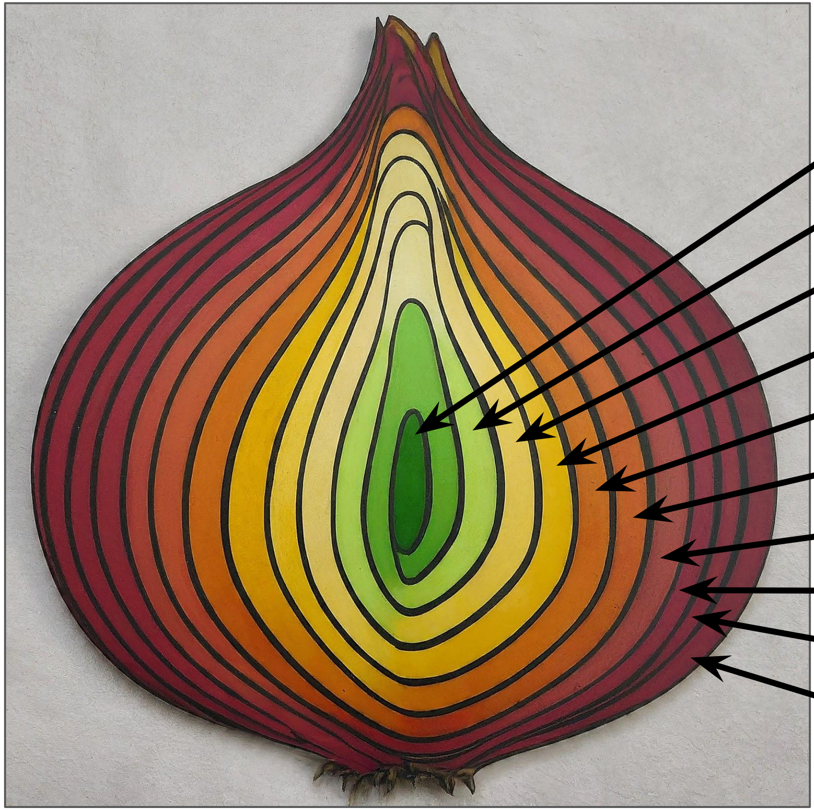


CVE-2022-35768

RtlLengthSecurityDescriptor() = ❌ too large

RtlLengthSecurityDescriptor() = ❌ too small

A taxonomy of bugs



- 🥇 Logic bugs
- 🥈 Cross-feature bugs
- 🥉 Object-lifetime bugs
- 4. Concurrency-related bugs
- 5. Cross-function bugs
- 6. Local bugs of omission
- 7. Information disclosure
- 8. Obvious, local bugs
- 9. Greppable
- 10. Fuzzable

Patterns for grepping

- General:
  - Buffer operations: calls to **memcpy** etc.
  - Dynamic allocations: calls to **malloc** etc.
  - Integer arithmetic: especially next to allocations, on 16-bit types etc.
- Kernel-specific:
  - Pointer probing: **ProbeForRead** / **ProbeForWrite** calls, references to **MmUserProbeAddress**
- Registry-specific:
  - Calls to the hive allocator: **HvAllocateCell**, **HvReallocateCell**, **HvFreeCell**
  - Operating on key handles: references to **CmKeyObjectType**

Example: long strings

- Under certain circumstances*, registry paths may be over 64 KiB long
- Windows stores strings, including registry paths, in UNICODE_STRING
    - 16-bit Length and MaximumLength fields
- Manually calculating the unicode buffer size may indicate insecure code
- Relatively easy to grep for 16-bit arithmetic in x86 assembly:

```
(add|sub)\s+[a-z][xi],
```

Examples 🐛

```
PAGE:00000001407F00CF                 mov     ecx, r9d        ; PoolType
PAGE:00000001407F00D2                 mov     [rbp+var_40.Buffer], rax
PAGE:00000001407F00D6                 movzx   eax, dx
PAGE:00000001407F00D9                 add     ax, ax
PAGE:00000001407F00DC                 test    r8d, r8d
PAGE:00000001407F00DF                 mov     r8d, 'bNMC'     ; Tag
PAGE:00000001407F00E5                 cmovz   ax, dx
PAGE:00000001407F00E9                 add     ax, 12h
PAGE:00000001407F00ED                 add     ax, [rbx]
PAGE:00000001407F00F0                 movzx   r15d, ax
PAGE:00000001407F00F4                 mov     edx, r15d       ; NumberOfBytes
PAGE:00000001407F00F7                 call    ExAllocatePoolWithTag
```

CmRealKCBToVirtualPath (CVE-2022-37990)

```
PAGE:00000001407F2039                 movzx   eax, word ptr [r15]
PAGE:00000001407F203D                 mov     ecx, 1          ; PoolType
PAGE:00000001407F2042                 add     ax, 2
PAGE:00000001407F2046                 mov     r8d, 'bNMC'     ; Tag
PAGE:00000001407F204C                 add     ax, [rbp+DestinationString.Length]
PAGE:00000001407F2050                 movzx   edx, ax         ; NumberOfBytes
PAGE:00000001407F2053                 mov     [rbp+RemainingPath.MaximumLength], dx
PAGE:00000001407F2057                 call    ExAllocatePoolWithTag
```

CmpVEExecuteVirtualStoreParseLogic (CVE-2022-38038)

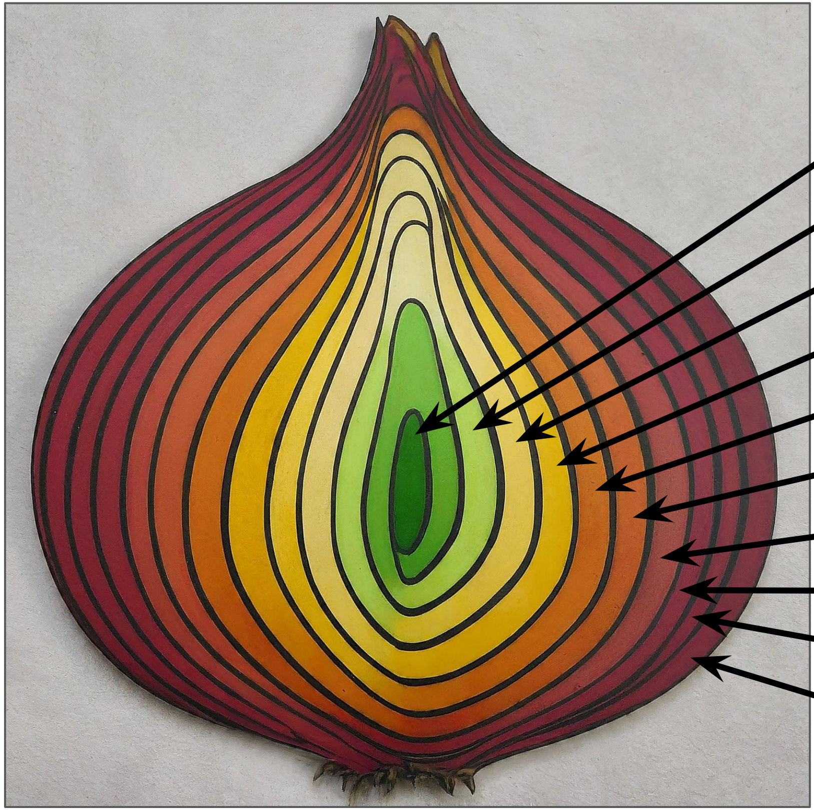
```
PAGE:000000014077B34D                 movzx   edx, bp         ; NumberOfBytes
PAGE:000000014077B350                 mov     ecx, 1          ; PoolType
PAGE:000000014077B355                 mov     r8d, '66MC'     ; Tag
PAGE:000000014077B35B                 call    ExAllocatePoolWithTag
```

CmpDoWritethroughReparse (CVE-2022-38039)

```
PAGE:000000014080C8BC                 movzx   eax, word ptr [rdx]
PAGE:000000014080C8BF                 mov     rdi, r8
PAGE:000000014080C8C2                 add     ax, 2
PAGE:000000014080C8C6                 movzx   ecx, word ptr [rsi+2]
PAGE:000000014080C8CA                 add     cx, ax
PAGE:000000014080C8CD                 movzx   edx, cx         ; NumberOfBytes
PAGE:000000014080C8D0                 lea     ecx, [rbx+1]    ; PoolType
PAGE:000000014080C8D3                 mov     [r8+2], dx
PAGE:000000014080C8D8                 mov     r8d, 'geRV'     ; Tag
PAGE:000000014080C8DE                 call    ExAllocatePoolWithTag
```

VrpBuildKeyPath (CVE-2023-36576)

A taxonomy of bugs



🥇 Logic bugs

🥈 Cross-feature bugs

🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Obvious, local bugs

- Disclaimer: often "obvious" only after hours of reversing
- Typical root causes:
  - Evident out-of-bounds array accesses
  - Incorrect allocation size
  - Incorrect return value
  - Incorrect reference counting
- Most frequent bug class in the research: **13 of 50**

Example (CVE-2022-34707)

```
BOOLEAN CmpCheckAndFixSecurityCellsRefcount(CMHIVE *CmHive) {
  ...
  for (int i = 0; i < CmHive->SecurityCount; i++) {
    CM_KEY_SECURITY_CACHE_ENTRY *CacheEntry = &CmHive->SecurityCache[i];
    CM_KEY_SECURITY *SecurityNode            = CmHive->Hive.GetCellRoutine(CmHive, CacheEntry->Cell);

    if (SecurityNode->ReferenceCount < CacheEntry->CachedSecurity->RealRefCount) {
      SecurityNode->ReferenceCount = CacheEntry->CachedSecurity->RealRefCount;
    }
  }
  ...
}
```

Example (CVE-2022-34707)

What about inadequately large refcounts?

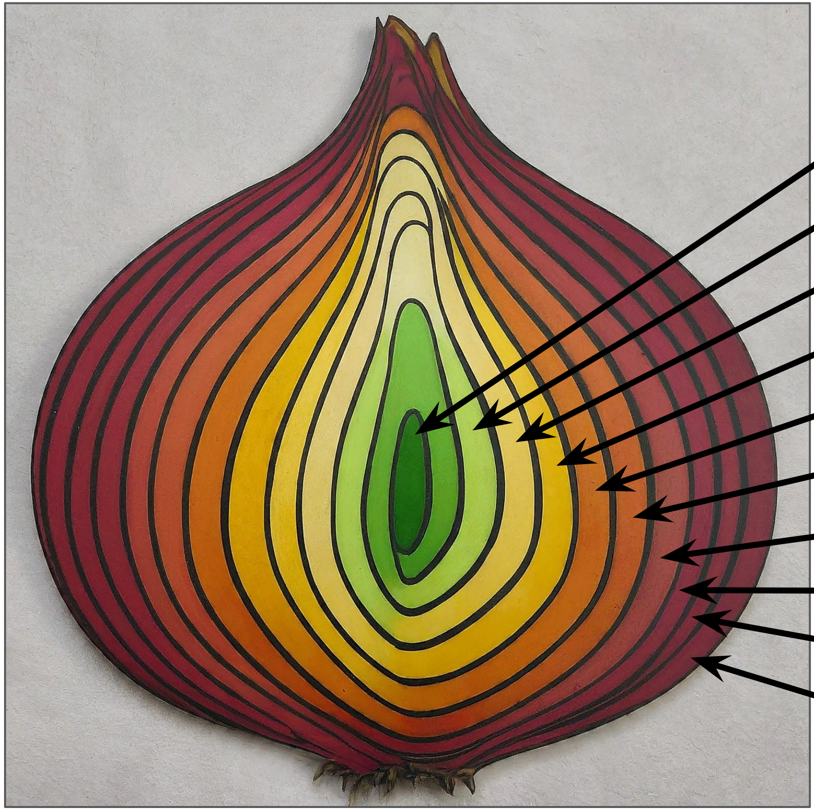
```
BOOLEAN CmpCheckAndFixSecurityCellsRefcount(CMHIVE *CmHive) {
  ...
  for (int i = 0; i < CmHive->SecurityCount; i++) {
    CM_KEY_SECURITY_CACHE_ENTRY *CacheEntry = &CmHive->SecurityCache[i];
    CM_KEY_SECURITY *SecurityNode            = CmHive->Hive.GetCellRoutine(CmHive, CacheEntry->Cell);

    if (SecurityNode->ReferenceCount < CacheEntry->CachedSecurity->RealRefCount) {
      SecurityNode->ReferenceCount = CacheEntry->CachedSecurity->RealRefCount;
    }
  }
  ...
}
```

CVE-2022-34707

- The bug lead to a refcount integer overflow, and a security descriptor use-after-free in the hive mapping
  - A registry-specific memory corruption primitive that hasn't been explored before
- With some work, it can be converted to a KASLR leak and arbitrary read/write
- For details, see my latest OffensiveCon talk on exploitation

Demo

A taxonomy of bugs



- 🥇 Logic bugs
- 🥈 Cross-feature bugs
- 🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Kernel information disclosure

- Disclosing uninitialized kernel stack/pool memory: partially filled arrays, padding structure bytes etc.
- *Could be* fuzzable or greppable, but it's harder, hence its own category
    - Never triggers a system crash, requires a dedicated detector (e.g. Bochspwn Reloaded)
    - Doesn't stand out when reading the code
- Enables a local attacker to leak kernel addresses or other system secrets

Examples 🐛

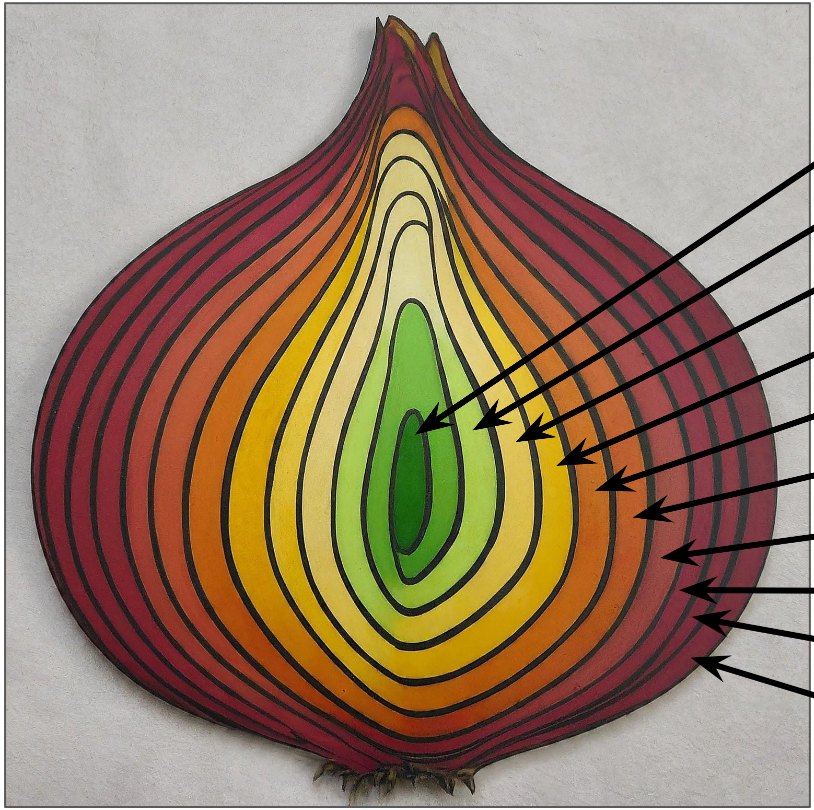
Issue 2418: Windows Kernel disclosure of kernel pointers and uninitialized memory through registry KTM transaction log files
Reported by mjurczyk@google.com on Tue, Jan 31, 2023, 3:43 PM GMT+1 Project Member

Issue 2463: Windows Kernel paged pool memory disclosure in VrpPostEnumerateKey
Reported by mjurczyk@google.com on Tue, Jun 27, 2023, 10:53 AM GMT+2 Project Member

- **Issue 2418** (CVE-2023-28271)
  - The kernel directly saved a kernel structure with pointers and padding bytes to a file
  - Required the use of transactions and observing that the log files are user-readable
- **Issue 2463** (CVE-2023-38140)
  - In principle, a standard kernel memory disclosure bug
  - Existed in a very specific code path, required layered keys and invoking a system call directly

Demo

A taxonomy of bugs



- 🥇 Logic bugs
- 🥈 Cross-feature bugs
- 🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Local bugs of omission

- Bugs that are local in scope, but caused by something that is *not* in the code
- Require a different mindset to identify
  - Consider whether a function does everything it should be doing in every code path
- Good candidates:
  - Missing bounds/correctness checks of some structure fields
  - Missing handling of specific object types in generic functions
  - Missing return value checks
  - Missing state unwinding in error code paths

Example (CVE-2023-28248)

```
VOID CmpCleanupLightWeightUoWData(CM_KCB_UOW *UoW) {
  switch (UoW->ActionType) {
    //
    // Other action types...
    //

    case UoWSetSecurityDescriptor:
      CM_UOW_SET_SD_DATA *SecurityData = UoW->SecurityData;
+     CmpDereferenceSecurityNode(SecurityData->Hive, SecurityData->SecurityCell);
      ExFreePoolWithTag(SecurityData, 'wUMC');
      break;
  }
}
```

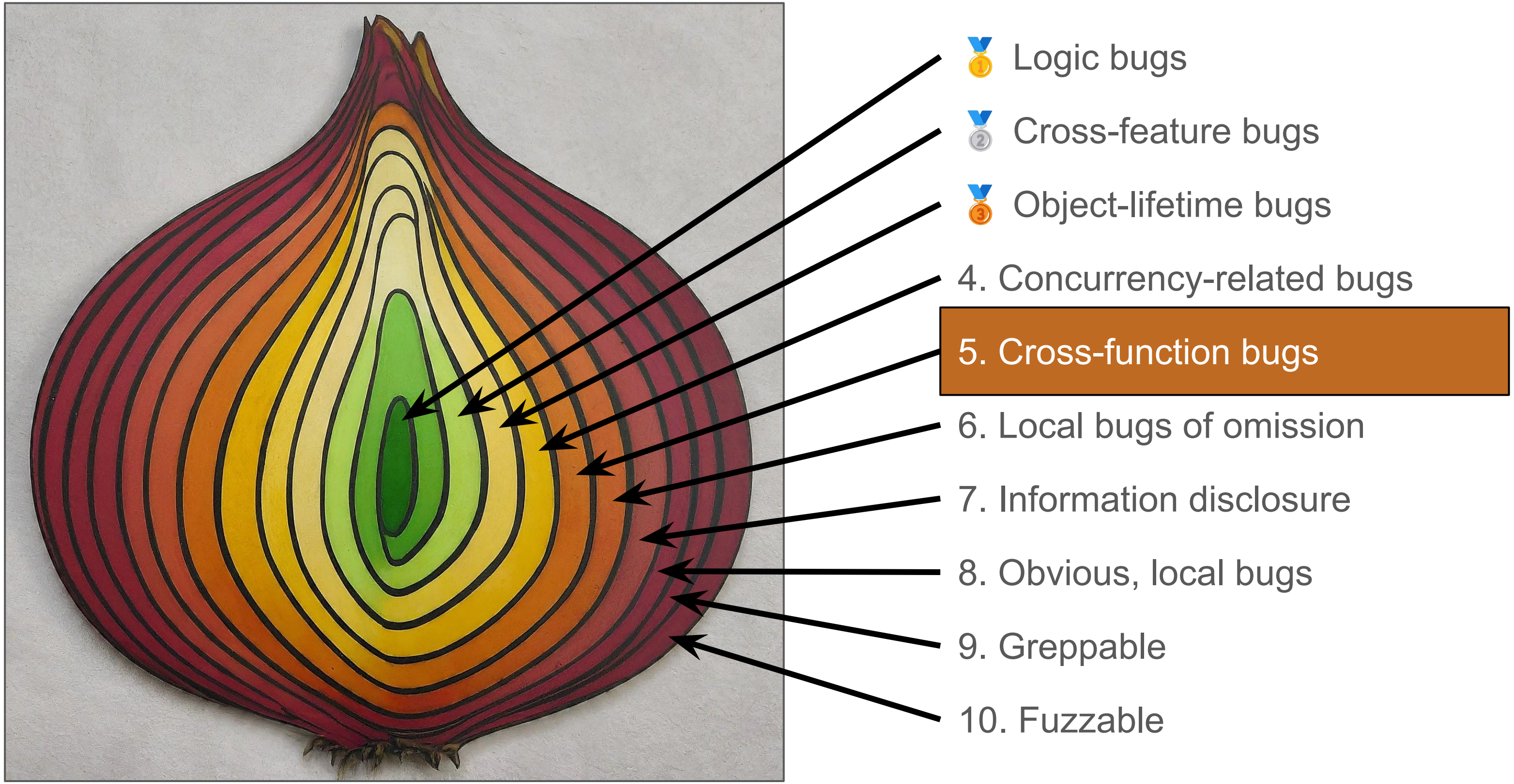
Missing security descriptor dereference

Example (CVE-2023-28248)

- A functionality-neutral issue
- Virtually impossible to find without careful analysis of the logic of the function
- Outcome:
  - The missing call leads to a leak of a single reference
  - The security descriptor refcount is a uint32, and can be incremented multiple times
  - There is no overflow protection, and once the value overflows, we get a UAF
- The proof-of-concept takes ~20 hours to complete

Demo ⏳

A taxonomy of bugs



Cross-function bugs

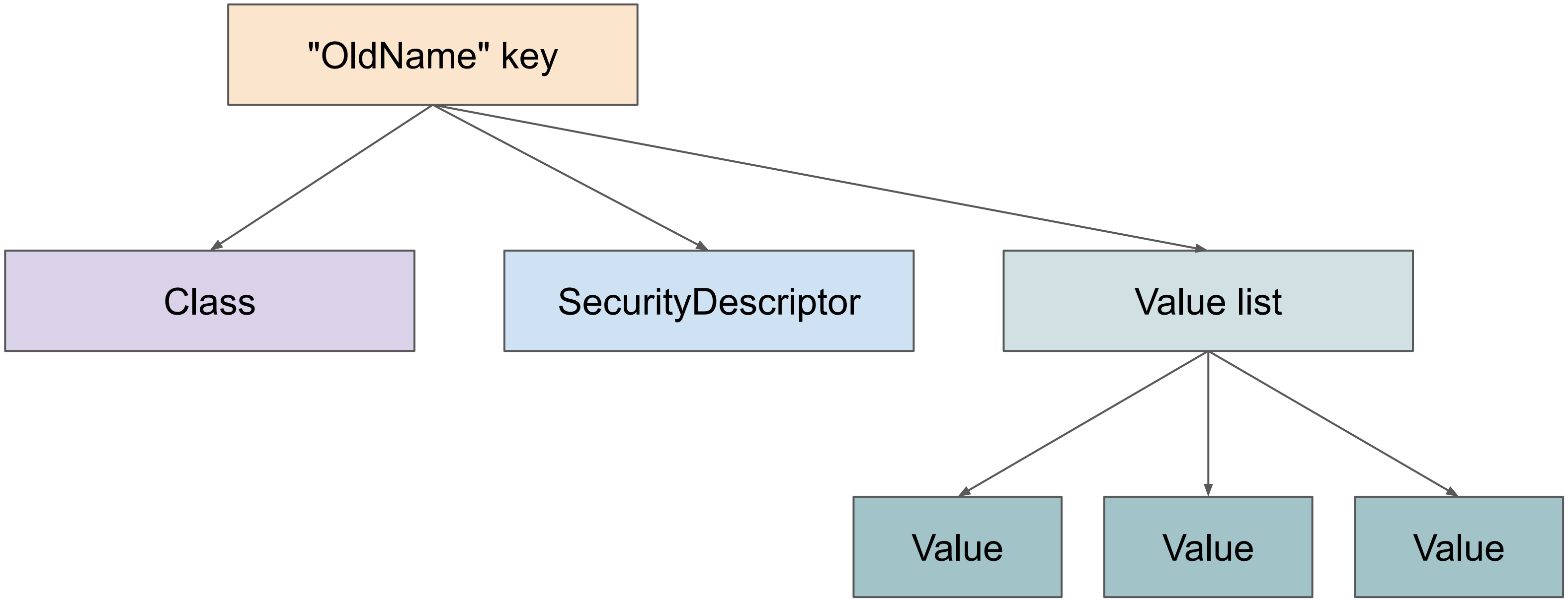
- Bugs that are rooted in (mis)interactions between two or more functions
- Examples observed in the registry:
  - Assumption that certain internal functions never fail
  - Assumption that a failed call implies no internal state change
  - Confusion about what success/failure even means
  - Using the wrong function for the wrong task

Example (CVE-2023-23423)

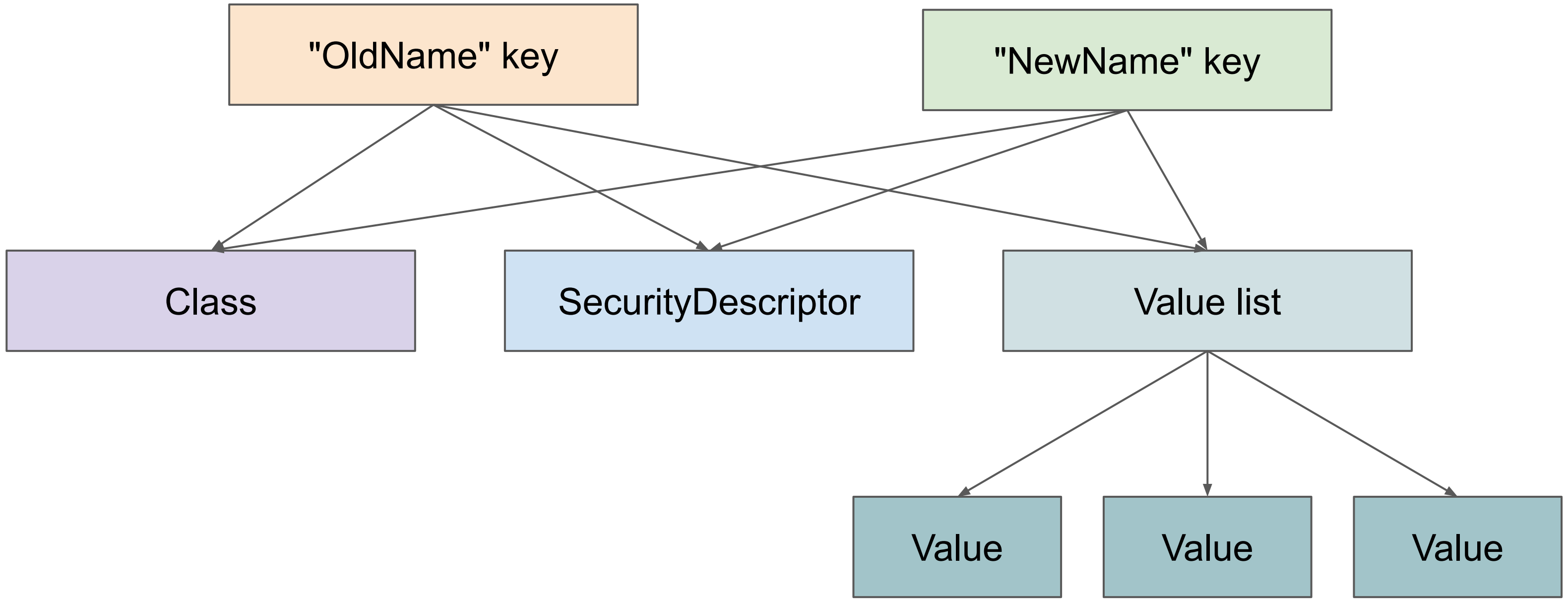
```
NTSTATUS CmpCommitRenameKeyUoW(CM_KCB_UOW *UoW) {
  // ...

  if (!CmpAddSubKeyEx(Hive, ParentKey, NewNameKey) ||
      !CmpRemoveSubKey(Hive, ParentKey, OldNameKey)) {
-    CmpFreeKeyByCell(Hive, NewNameKey);
+    HvFreeCell(Hive, NewNameKey);
    return STATUS_INSUFFICIENT_RESOURCES;
  }

  // ...
}
```

Deep vs. shallow free

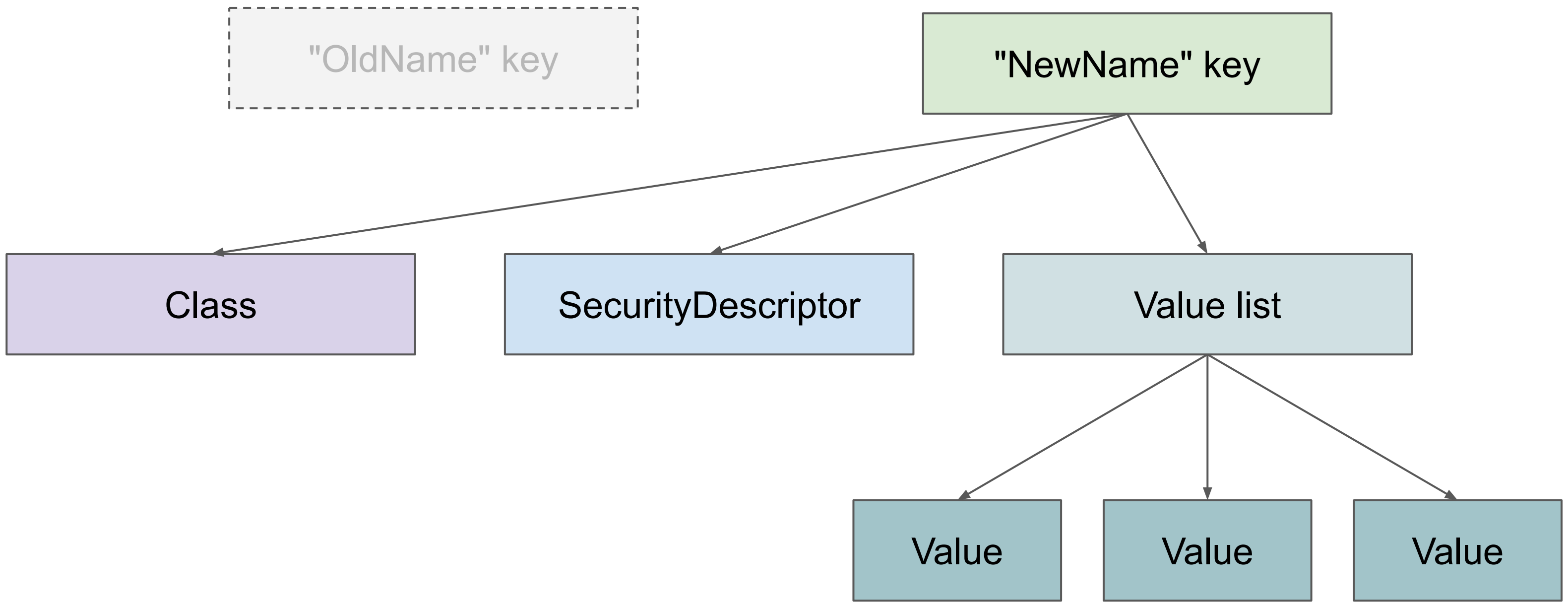
Successful rename case



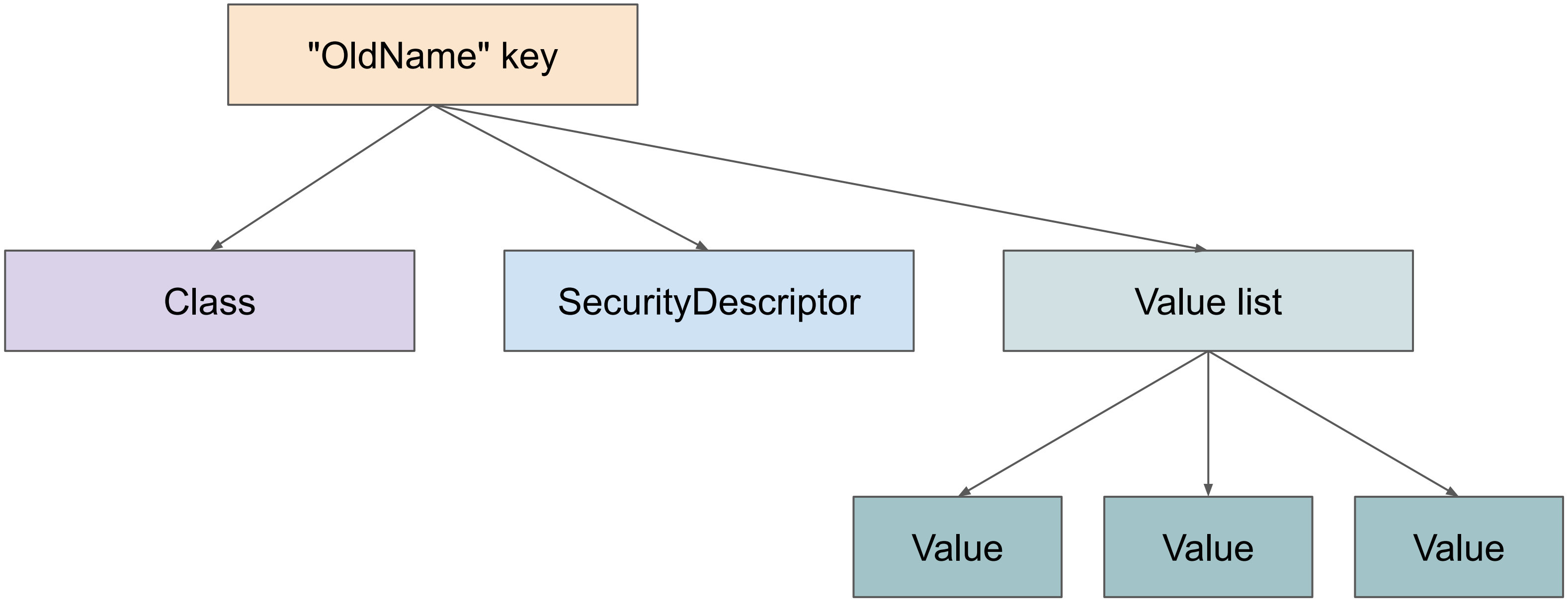
Successful rename case



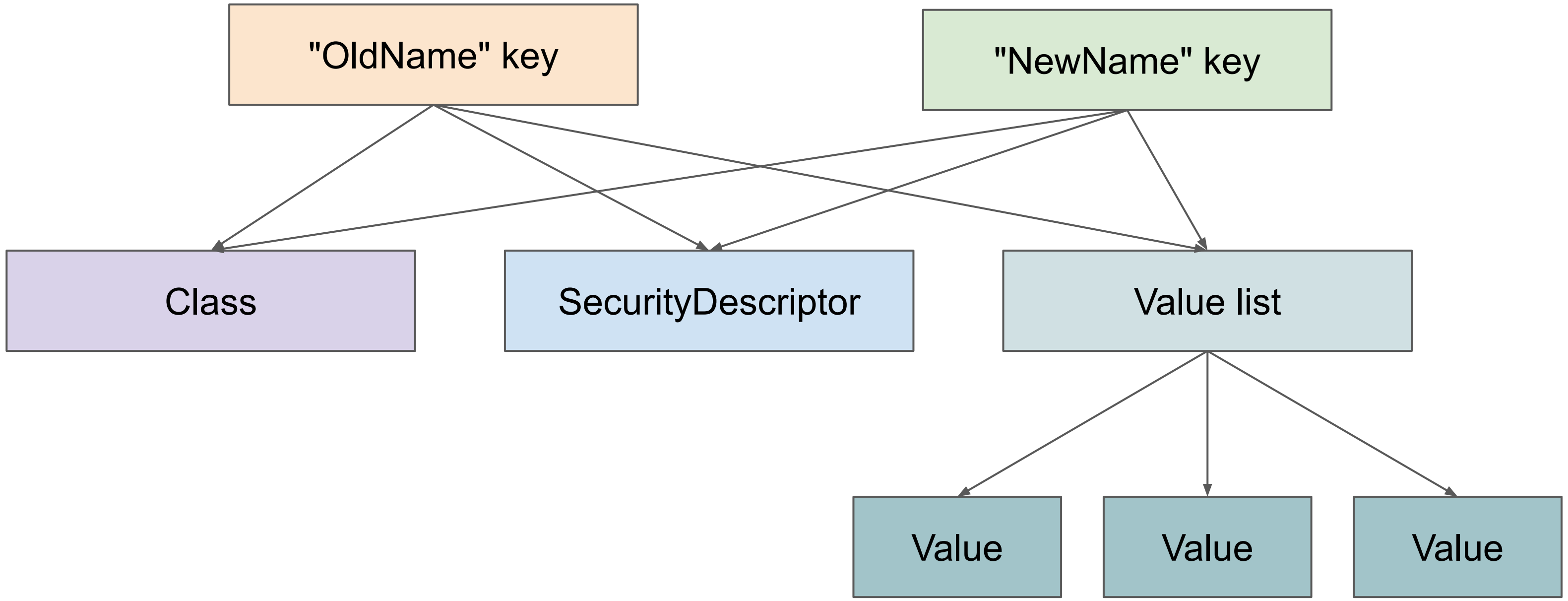
Successful rename case



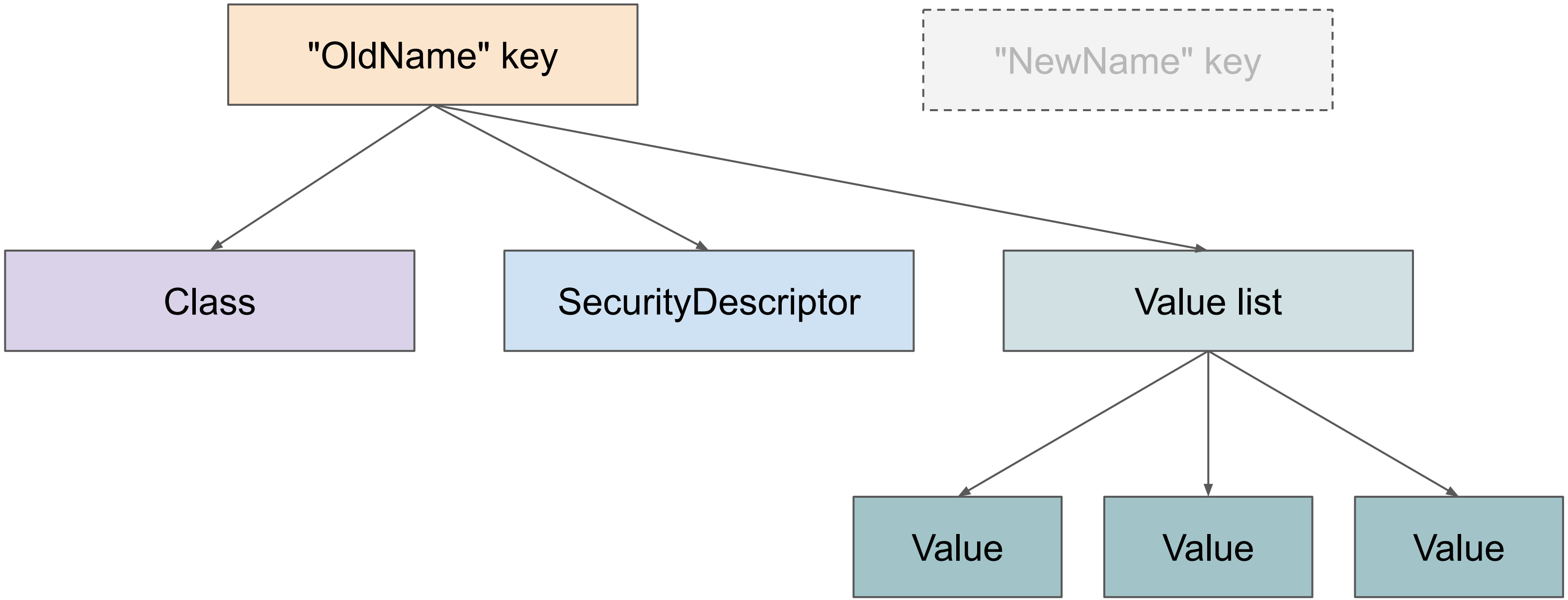
Failed rename case (correct)



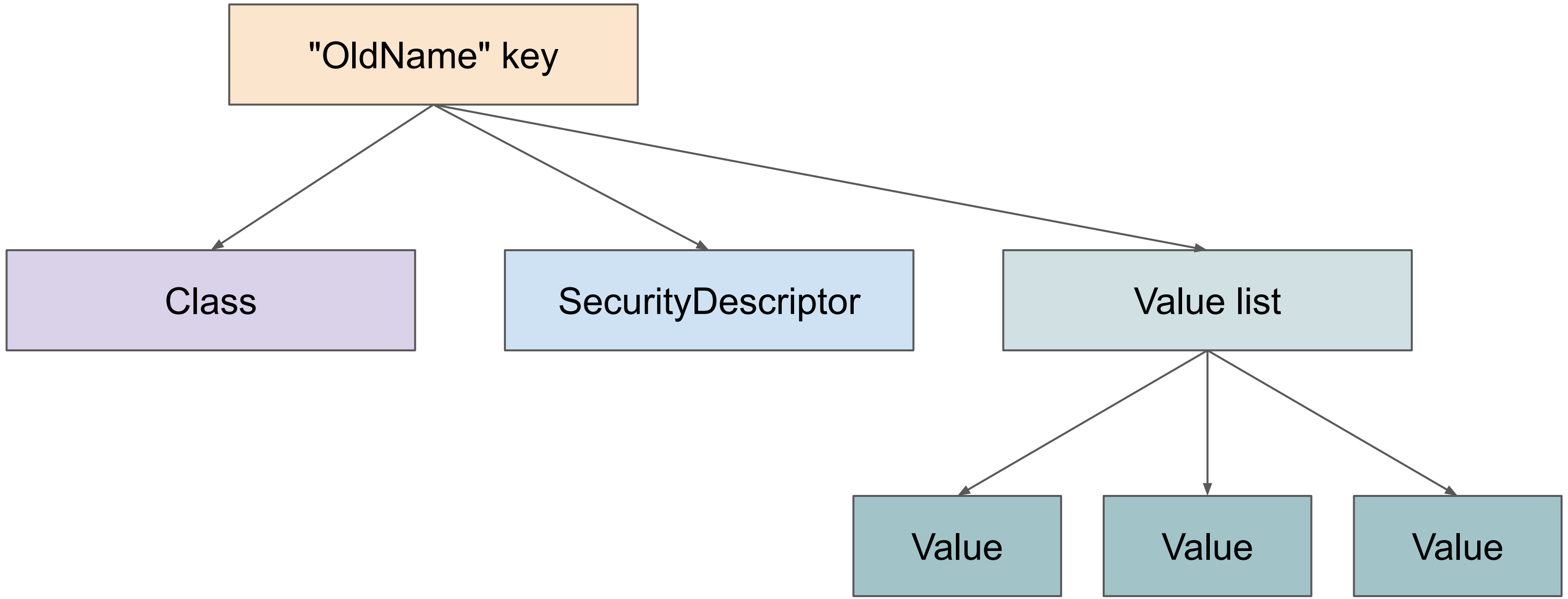
Failed rename case (correct)



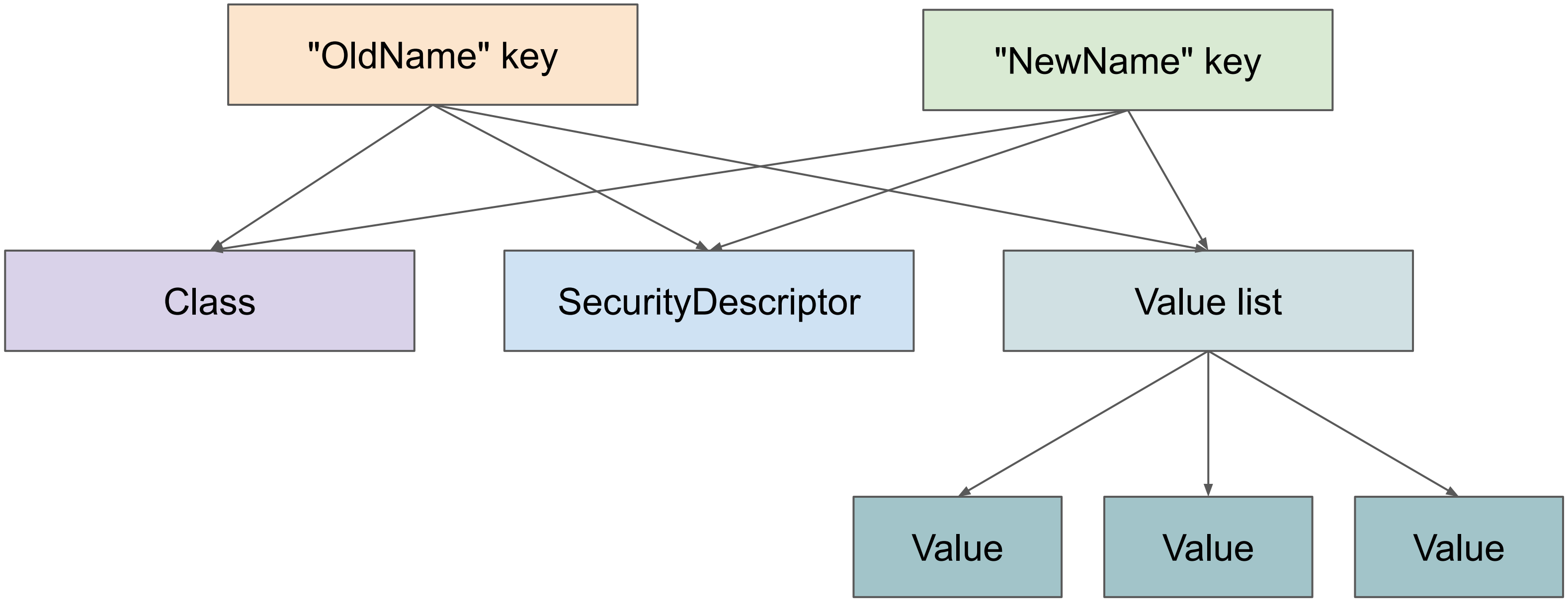
Failed rename case (correct)



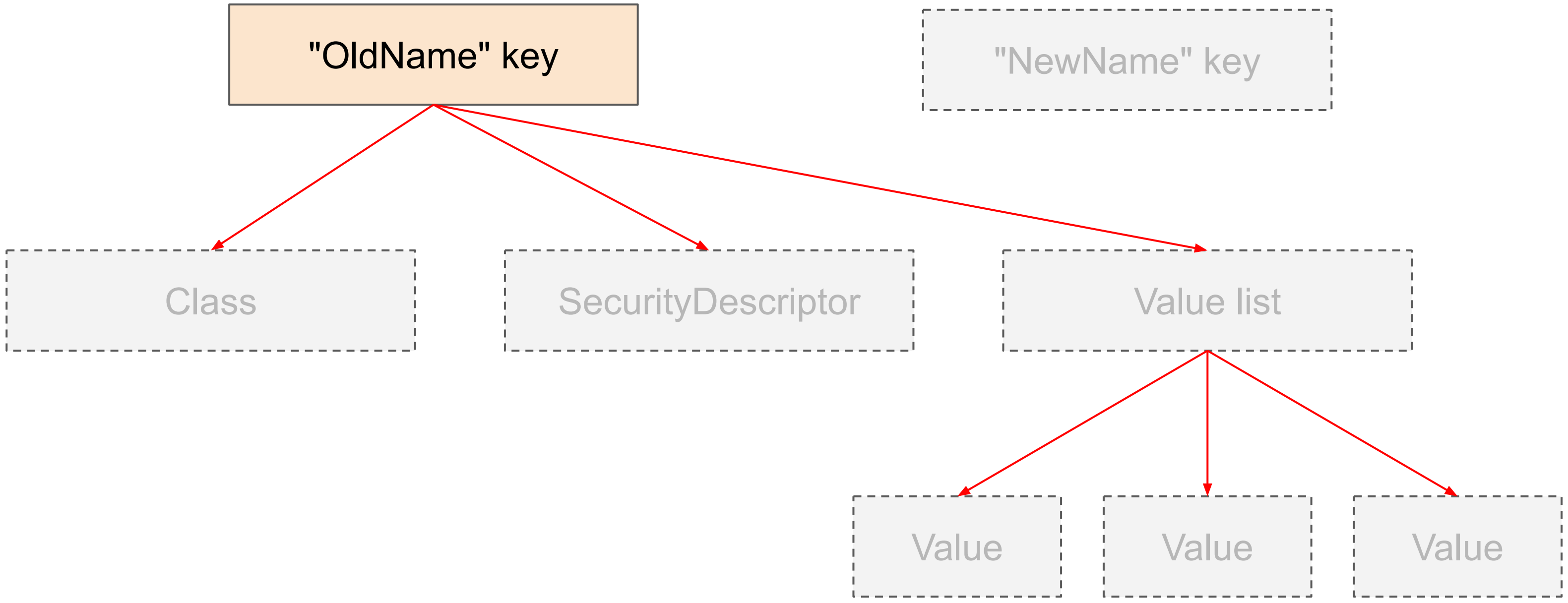
Failed rename case (buggy)



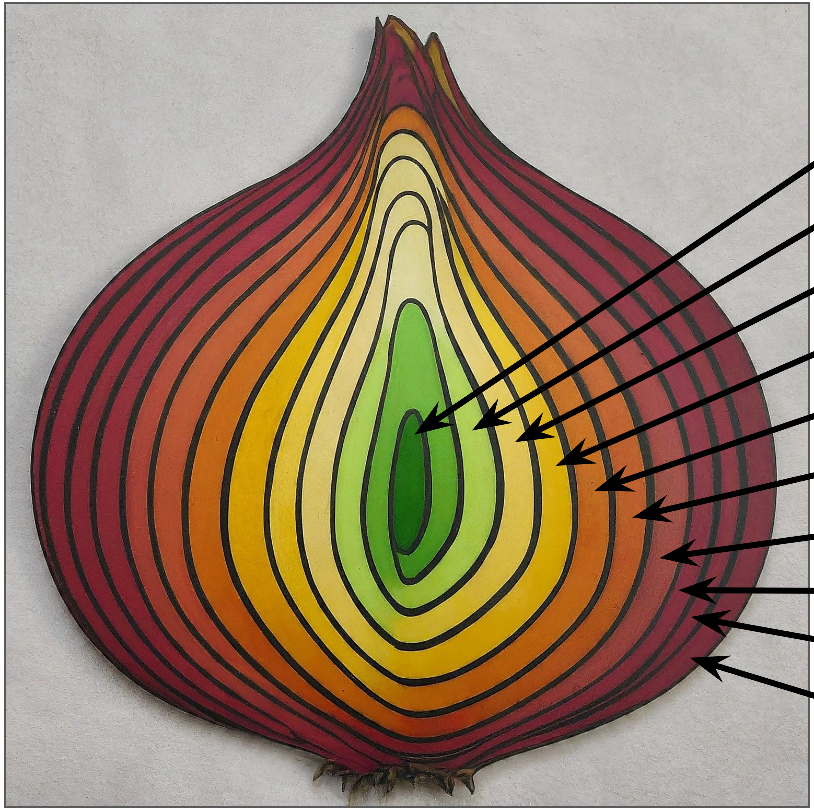
Failed rename case (buggy)



Failed rename case (buggy)



A taxonomy of bugs



🥇 Logic bugs

🥈 Cross-feature bugs

🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Race conditions

- Bugs that require an understanding of how global state can be manipulated in different code paths at the same time
- General problem types:
  - Missing synchronization of access to a resource
  - Bad synchronization: *shared* vs. *exclusive* access
  - Bad synchronization: locking the wrong thing **(two registry reports)**
  - Interactions with user-mode memory: double fetches etc. **(one registry report)**

Example (CVE-2023-38141)

Issue 2462: Windows Kernel passes user-mode pointers to registry callbacks, leading to race conditions and memory corruption Code

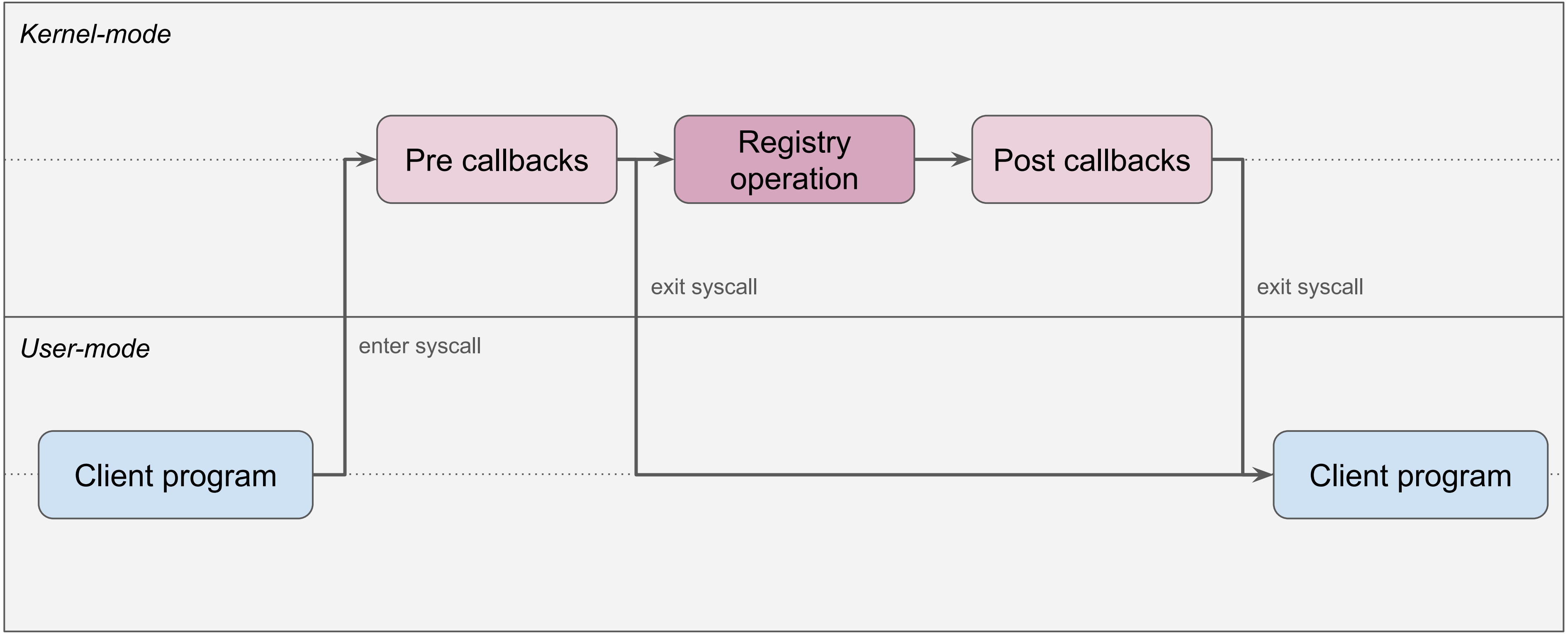
Reported by mjurczyk@google.com on Mon, Jun 26, 2023, 3:13 PM GMT+2 Project Member

The Windows operating system exposes a documented kernel API named Registry Callbacks. It allows drivers and the kernel itself to register callback functions using CmRegisterCallbackEx, which then get invoked every time a registry operation takes place in the system. The callbacks are provided with full information about the type and context of the operations through the REG_NOTIFY_CLASS enum and one of the many corresponding REG_*_INFORMATION structures. Based on this data, the callbacks can decide whether to act on it - for example log the operation, block it, adjust the output data, or intercept it and bypass the Configuration Manager completely. One obvious use case for this interface is antivirus-like software, but it is also utilized by the core Windows kernel as well, e.g. to implement the namespace redirection feature of the VRegDriver (part of containerized registry support for app/server silos), or for ETW logging of registry activity.

There is a fundamental weakness in the way the callback support is currently implemented: many of the operation-specific structures contain pointers to input/output data, and in some cases, these fields point directly to user-mode buffers passed to the registry syscalls as arguments by client applications. This fact is documented in the specification of the registry callback function [1]. According to MSDN, input buffer pointers are safe to use because they are captured by the kernel before being passed to the callbacks on modern versions of Windows (8 and newer), while output buffer pointers are always potentially unsafe and must be accessed within try/except blocks and/or captured in kernel-mode memory before passing them to other kernel functions.

However, there are two issues here:

Registry callbacks?



Operating on input/output pointers

Buffer type	**Windows version**	**Buffer pointer passed to callback routine**	**Safe for callback routine to directly access?**	**Safe to pass to system routines (such as ZwOpenKey)?**
User-mode input	Windows 8 and later	Points to captured data.	Yes	Yes
User-mode input	Windows 7 and earlier	Points to captured data or original user-mode buffer.	No. Must read under try/except.	No. Must allocate kernel memory, copy data from the original buffer under try/except, and pass the copied data to the system routine.
User-mode output	All	Points to original user-mode buffer.	No. Must write under try/except.	No. Must allocate kernel memory, pass kernel memory to the system routine, and copy the results back to the original buffer under try/except.
Kernel-mode input and output	All	Points to original kernel-mode buffer.	Yes	Yes

Operating on input/output pointers

Problem #1: untrue for some operations:

- SetInformationKey
- QueryMultipleValueKey

Buffer type	Windows version	Buffer pointer passed to callback routine	Safe for call routine to d acces?	
User-mode input	Windows 8 and later	Points to captured data.	Yes	Yes
User-mode input	Windows 7 and earlier	Points to captured data or original user-mode buffer.	No. Must read under try/except.	No. Must allocate kernel memory, copy data from the original buffer under try/except, and pass the copied data to the system routine.
User-mode output	All	Points to original user-mode buffer.	No. Must write under try/except.	No. Must allocate kernel memory, pass kernel memory to the system routine, and copy the results back to the original buffer under try/except.
Kernel-mode input and output	All	Points to original kernel-mode buffer.	Yes	Yes

Operating on input/output pointers

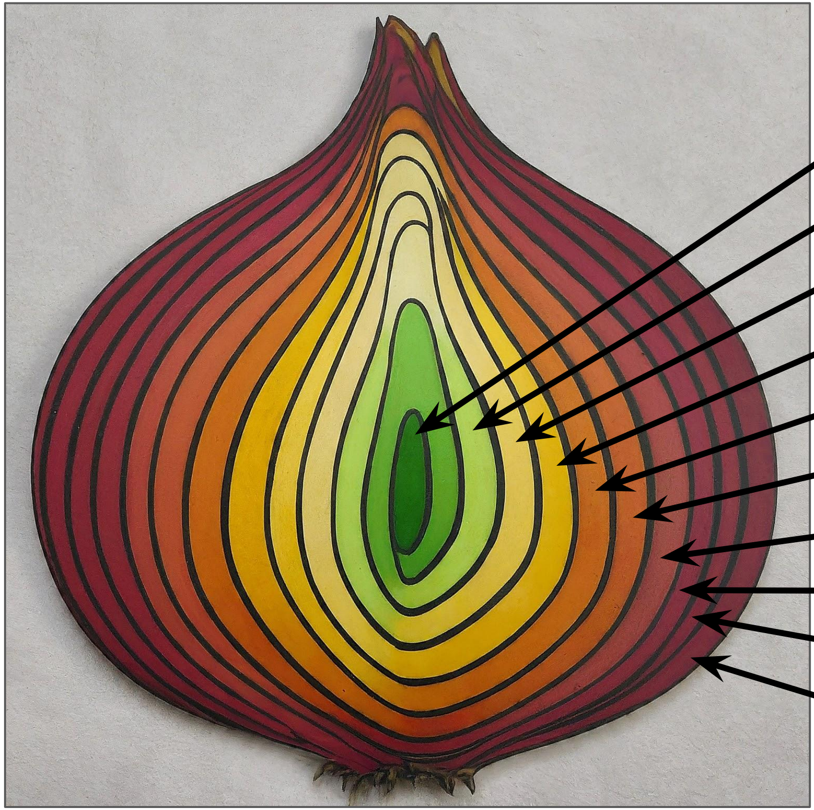
Buffer type	Windows version	Buffer pointer passed to callback routine	Safe for call routine to d access?	
User-mode input	Windows 8 and later	Points to captured data.	Yes	Yes
User-mode input	Windows 7 and earlier	Points to captured data or original user-mode buffer.	No. Must read under try/except.	No. Must allocate kernel memory, copy data from the original buffer under try/except, and pass the copied data to the system routine.
User-mode output	All	Points to original user-mode buffer.	No. Must write under try/except.	No. Must allocate kernel memory, pass kernel memory to the system routine, and copy the results back to the original buffer under
Kernel-mode input and output	All	Points to original kernel-mode buffer.	Yes	

Problem #1: untrue for some operations

- SetInformationKey
- QueryMultipleValueKey

Problem #2: documented, but surprising even for Windows kernel developers

A taxonomy of bugs



🥇 Logic bugs

🥈 Cross-feature bugs

🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Object lifetime bugs

- Bugs that require understanding of how objects are created, managed and destroyed in time
    - Temporal violations, typically use-after-free
- In registry, a key's lifetime may be hard to reason about
    - Referenced by a handle in the period between RegOpenKey and RegCloseKey
    - Within that time, many things can happen:
        - The key can be renamed / deleted
        - Its parent key can be renamed
        - The underlying hive can be unloaded

Key lifetime challenges

- Challenge 1: renaming keys (NtRenameKey)
  - Very complex, combines the delete + create operation in one
- Challenge 2: uncommitted transactions
  - Operations aren't atomic; an intermediate state gets exposed that had previously been hidden
- Put the things together: renaming + transactions = Schrödinger's key ☢️
  - A single key is simultaneously known by two different names, and its subkeys by two paths
  - The Windows NT-era registry was not designed for this

Examples

Issue 2392: Windows Kernel multiple issues with subkeys of transactionally renamed registry keys
Reported by mjurczyk@google.com on Wed, Dec 7, 2022, 11:24 AM GMT+1 Project Member

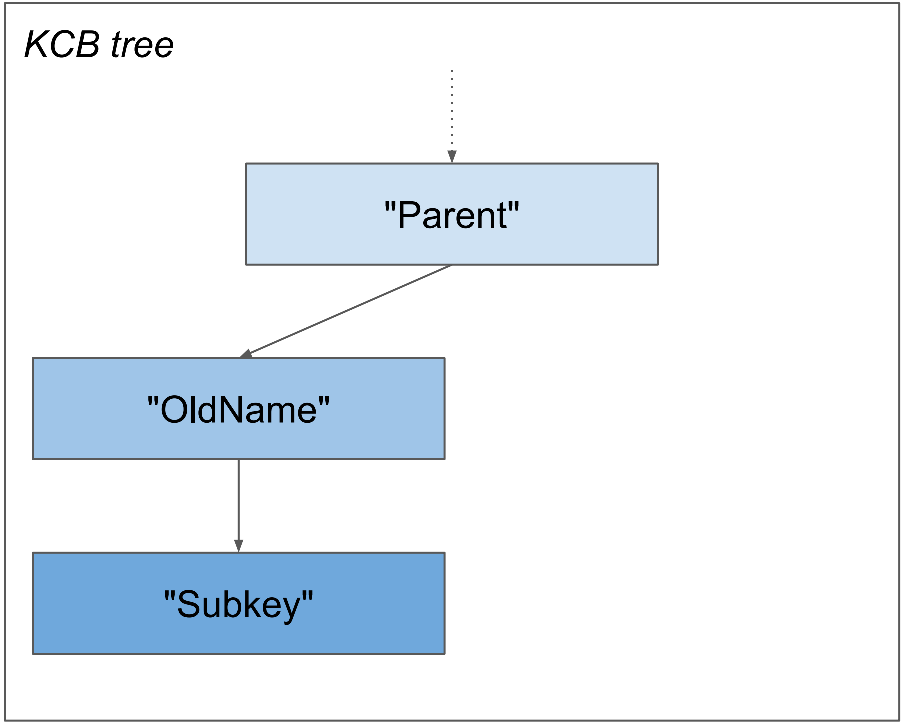
Issue 2394: Windows Kernel multiple issues in the prepare/commit phase of a transactional registry key rename
Reported by mjurczyk@google.com on Wed, Dec 14, 2022, 5:23 PM GMT+1 Project Member

Issue 2408: Windows Kernel insufficient validation of new registry key names in transacted NtRenameKey
Reported by mjurczyk@google.com on Fri, Jan 13, 2023, 2:12 PM GMT+1 Project Member

All reports fixed collectively in March 2023 by disabling transacted renames

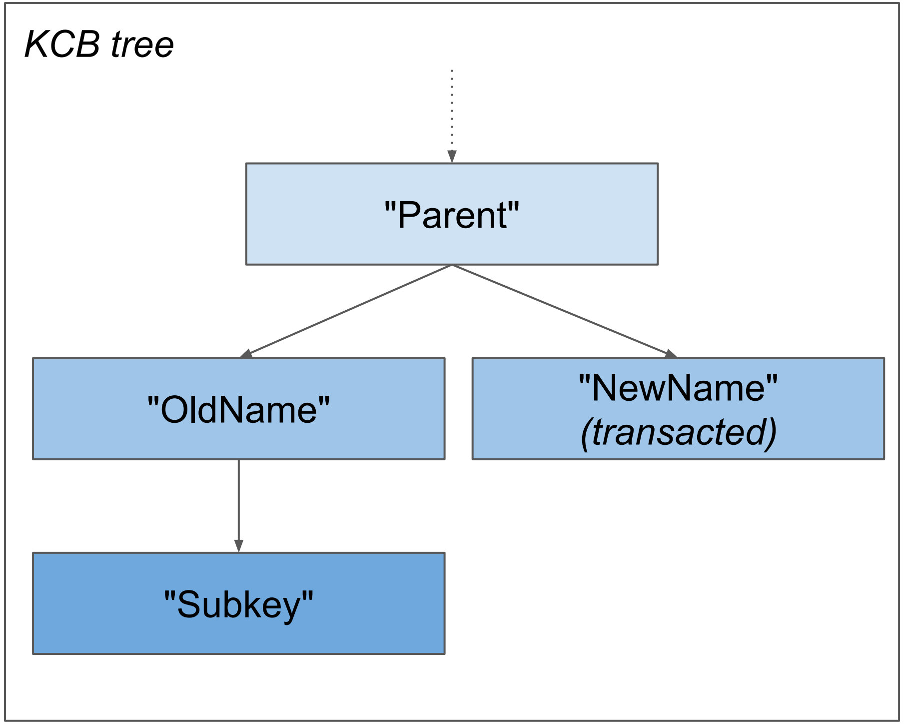
CVE-2023-23420

1. Open a handle to Parent\OldName\Subkey to create its corresponding KCB



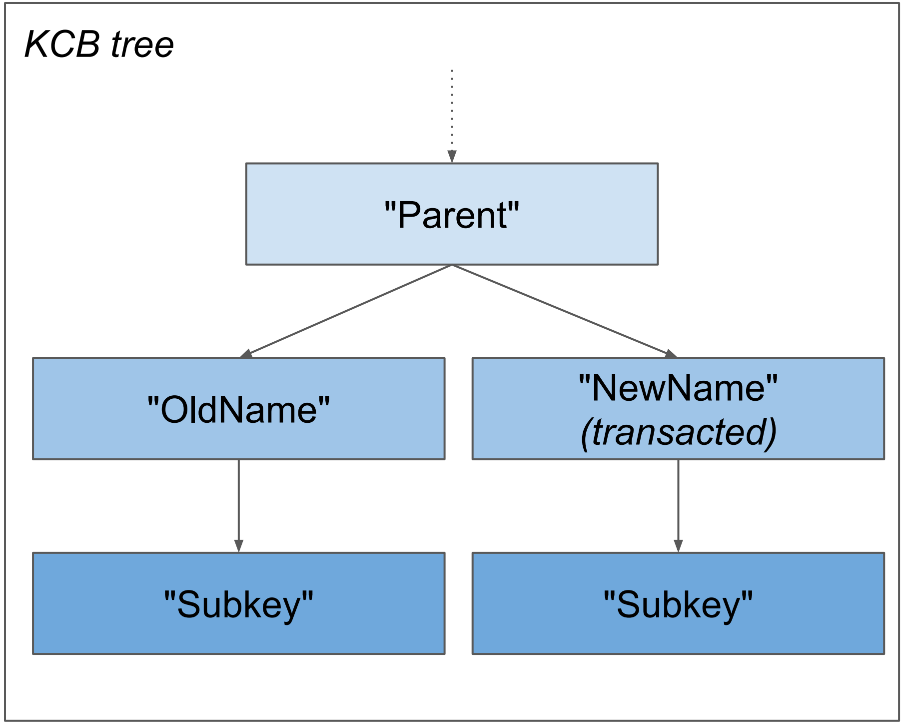
CVE-2023-23420

1. Open a handle to Parent\OldName\Subkey to create its corresponding KCB
2. Transactionally rename "OldName"



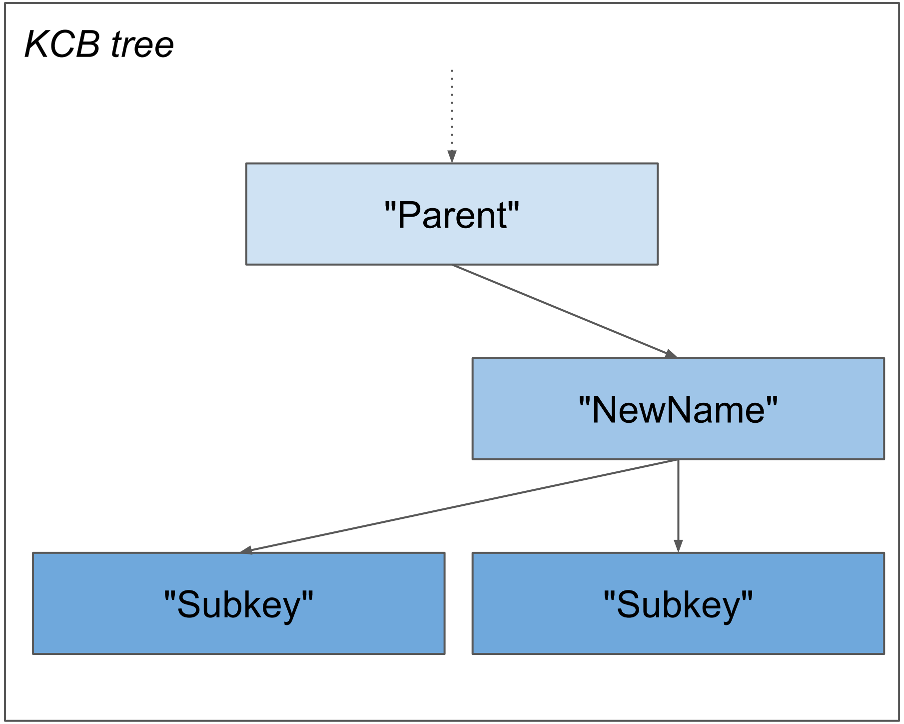
CVE-2023-23420

1. Open a handle to Parent\OldName\Subkey to create its corresponding KCB
2. Transactionally rename "OldName"
3. Open a handle to Parent\NewName\Subkey



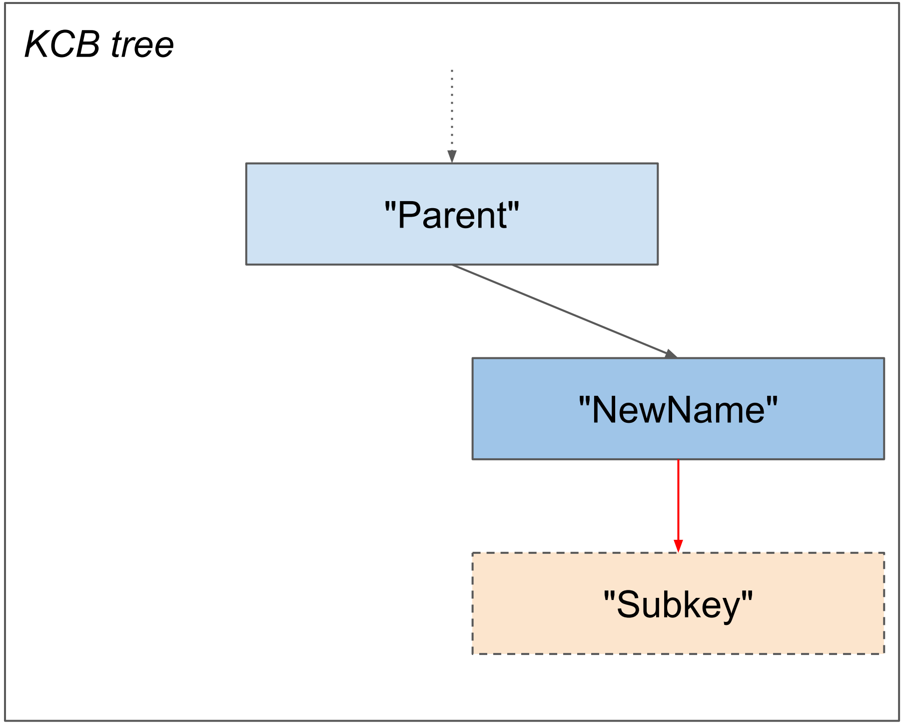
CVE-2023-23420

1. Open a handle to Parent\OldName\Subkey to create its corresponding KCB
2. Transactionally rename "OldName"
3. Open a handle to Parent\NewName\Subkey
4. Commit the transaction, leading to duplicate KCBs of the subkey



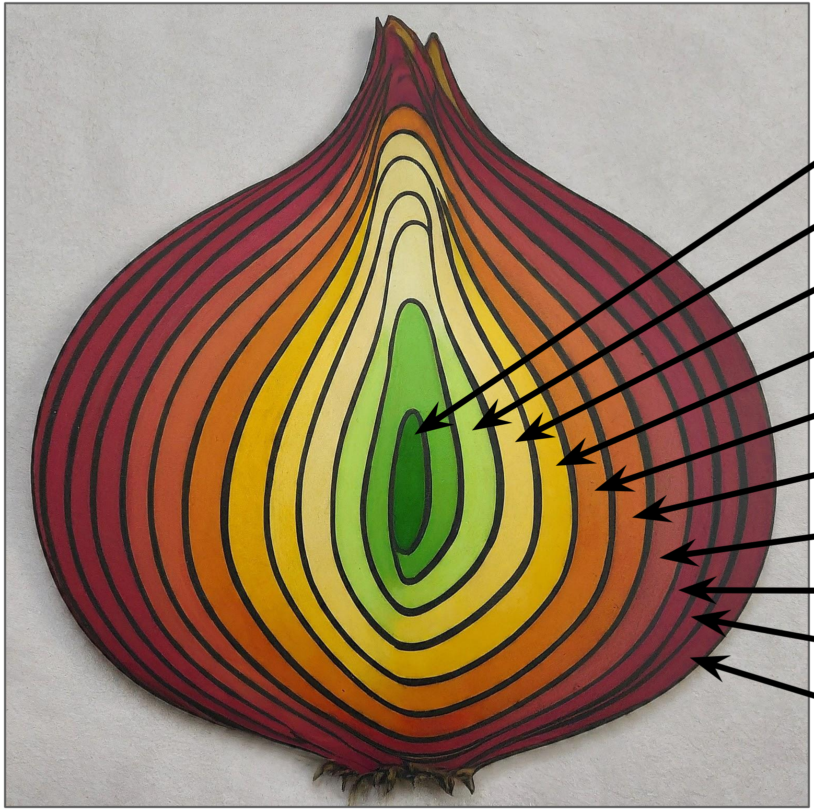
CVE-2023-23420

1. Open a handle to Parent\OldName\Subkey to create its corresponding KCB
2. Transactionally rename "OldName"
3. Open a handle to Parent\NewName\Subkey
4. Commit the transaction, leading to duplicate KCBs of the subkey
5. Delete the subkey and discard one of the KCBs; the other KCB now refers to freed objects



Demo

A taxonomy of bugs



- 🥇 Logic bugs
- 🥈 Cross-feature bugs
- 🥉 Object-lifetime bugs

4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Cross-feature bugs

- The Windows NT 3.1 registry design was elegant, but simple
- Many mechanisms introduced later are "hacks" addressing specific problems:
  - Predefined keys
  - Symbolic links
  - Registry virtualization
  - KTM and lightweight transactions
  - Differencing hives and layered keys
- So how do they all work together?


Predefined Keys
Registry Virtualization
Layered Keys
Transactions
Symbolic Links

Cross-feature bugs

- A bit of a hyperbole – they are not actively hostile
- However, they are often unaware of each others' corner cases and may trip over them:
    - Reimplementing a standard operation without porting all of the checks from the canonical one
    - Accessing weird keys / key placeholders indirectly, where directly wouldn't have been possible
    - Forgetting to opt out of specific options, which are opt-in by default and not immediately obvious

Examples

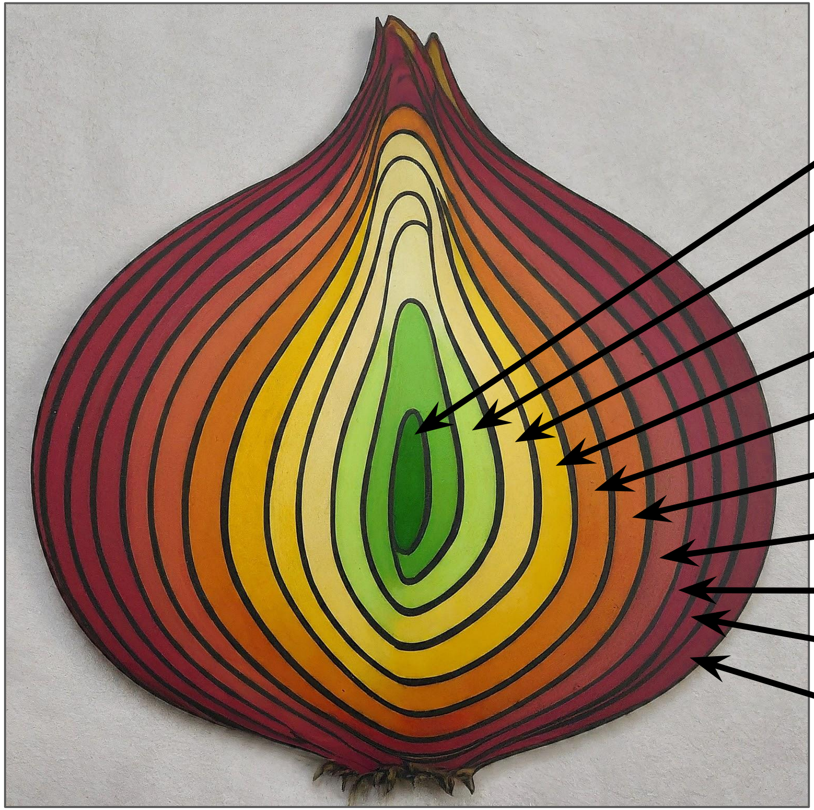
Issue 2389: Windows Kernel registry virtualization incompatible with transactions, leading to inconsistent hive state and memory corruption
Reported by mjurczyk@google.com on Wed, Nov 30, 2022, 3:50 PM GMT+1 Project Member

Issue 2445: Windows Kernel arbitrary read by accessing predefined keys through differencing hives
Reported by mjurczyk@google.com on Wed, Apr 19, 2023, 3:20 PM GMT+2 Project Member

Issue 2446: Windows Kernel may reference unbacked layered keys through registry virtualization
Reported by mjurczyk@google.com on Thu, Apr 20, 2023, 3:44 PM GMT+2 Project Member

Issue 2447: Windows Kernel may reference rolled-back transacted keys through differencing hives
Reported by mjurczyk@google.com on Thu, Apr 27, 2023, 1:01 PM GMT+2 Project Member

A taxonomy of bugs



- 🥇 Logic bugs
- 🥈 Cross-feature bugs
- 🥉 Object-lifetime bugs

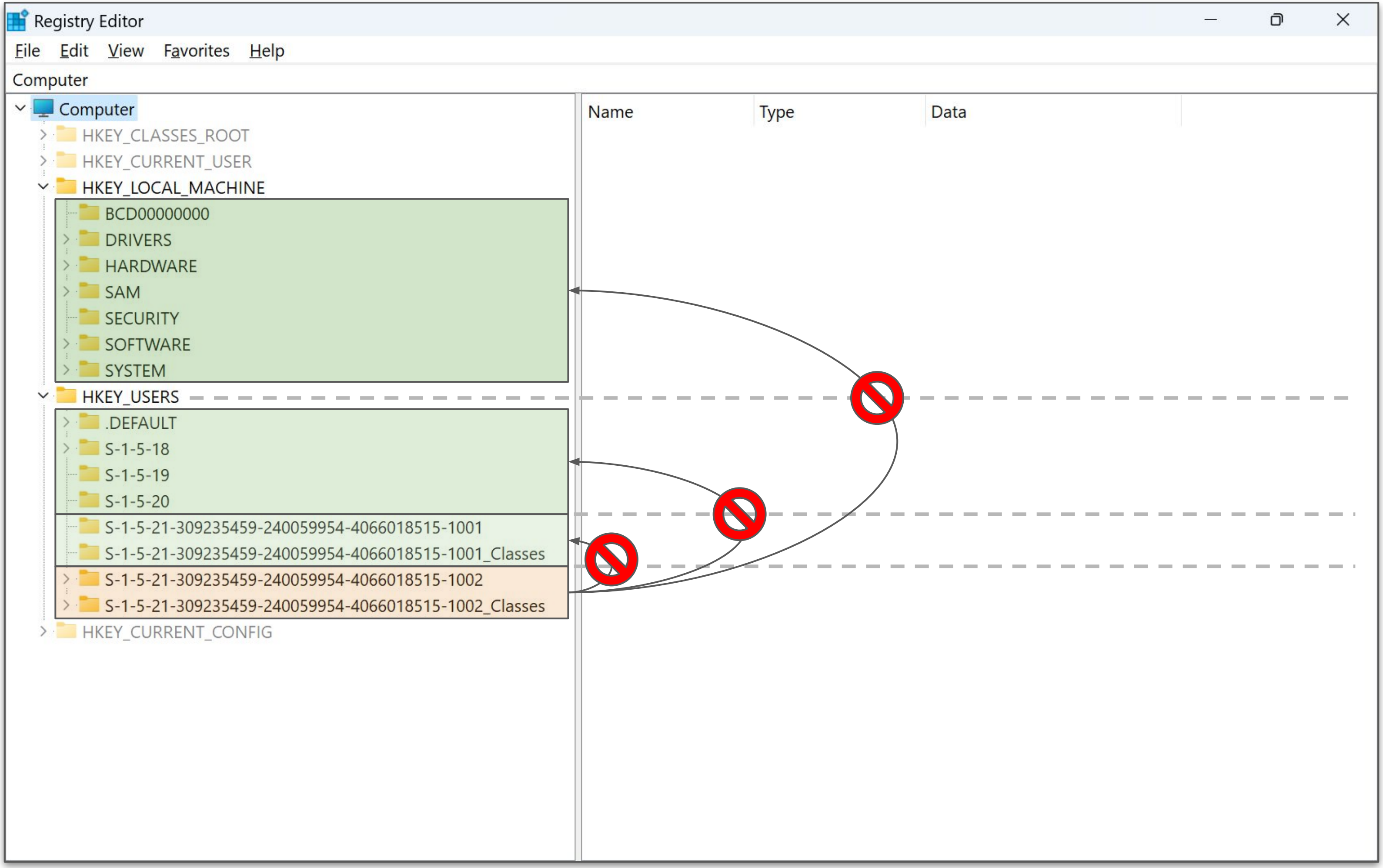
4. Concurrency-related bugs
5. Cross-function bugs
6. Local bugs of omission
7. Information disclosure
8. Obvious, local bugs
9. Greppable
10. Fuzzable

Logic bugs

- The crown jewel of software vulnerabilities 👑
  - Can be very deep and hard to find with automation
  - Often 100% reliable
  - Typically don't involve memory corruption and are easier to exploit
- Particularly relevant to the registry
  - Implements a substantial amount of high-level logic
  - Responsible for enforcing its own security access checks
  - Manages sensitive system configuration that is attractive both to leak and corrupt
  - Shared by both restricted and highly-privileged processes in the system

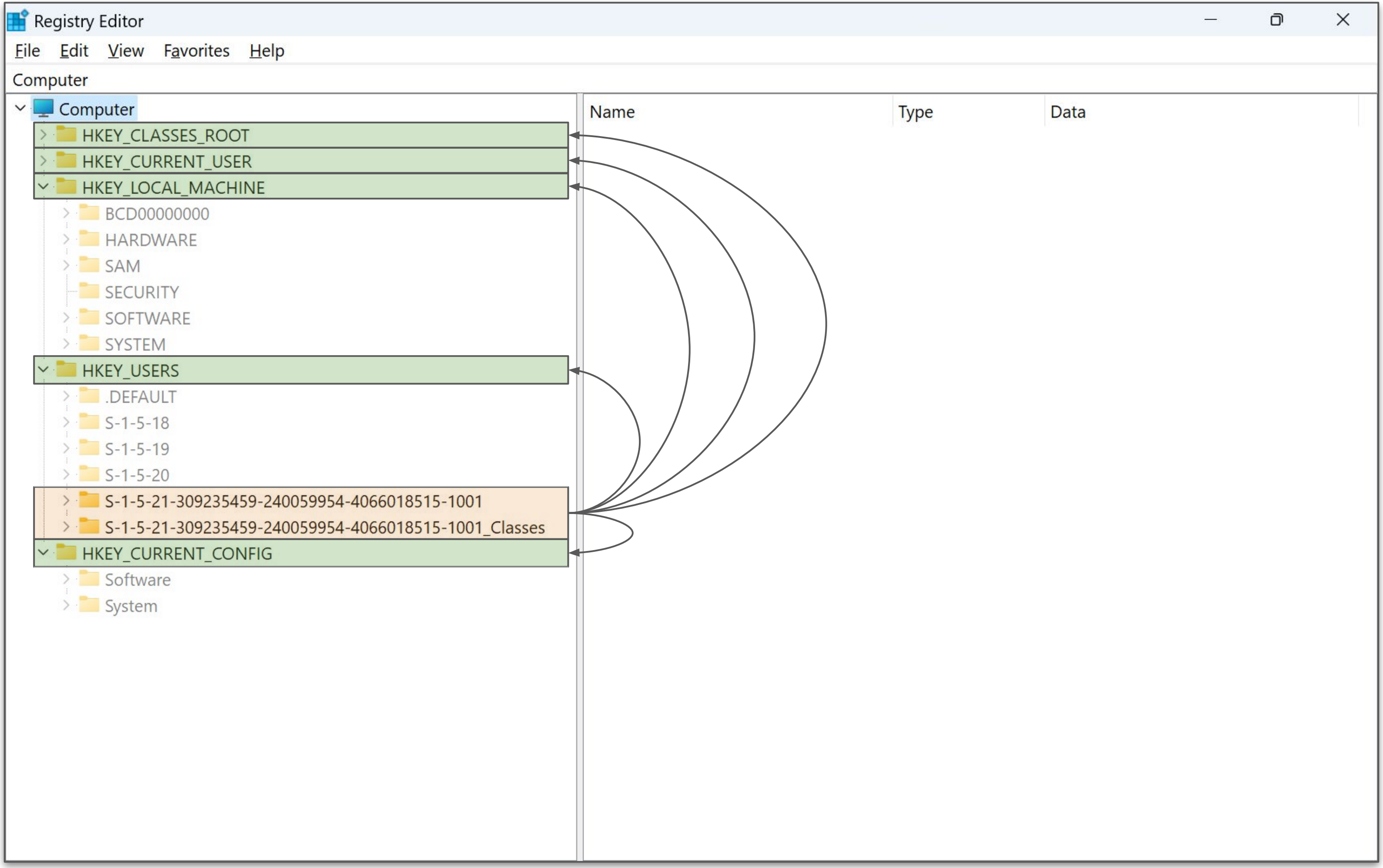
Case study: symbolic links

- Symbolic links with source/destination across different privilege levels are dangerous, as they can lead to confused deputy problems
- This has been previously the case in Windows XP and earlier versions
- Addressed in Windows Server 2003 and later with *hive trust classes*


Registry Editor
File
Edit
View
Favorites
Help
Computer
Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
BCD00000000
DRIVERS
HARDWARE
SAM
SECURITY
SOFTWARE
SYSTEM
HKEY_USERS
.DEFAULT
S-1-5-18
S-1-5-19
S-1-5-20
S-1-5-21-309235459-240059954-4066018515-1001
S-1-5-21-309235459-240059954-4066018515-1001_Classes
S-1-5-21-309235459-240059954-4066018515-1002
S-1-5-21-309235459-240059954-4066018515-1002_Classes
HKEY_CURRENT_CONFIG
Name
Type
Data

Predefined keys

- A special type of key introduced for compatibility reasons in Windows NT 3.5
  - Redirects a key to a controlled HKEY_* top-level key on the Registry API level
  - Used to redirect two Perflib-related keys to their HKEY_PERFORMANCE_* counterparts
- Conceptually equivalent to symbolic links, but not subject to trust classes
- More restricted than regular symlinks:
  - Source: a hive that grants write access to its backing file
  - Destination: one of ~10 possible top-level keys


Registry Editor
File
Edit
View
Favorites
Help
Computer
Computer
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
BCD00000000
HARDWARE
SAM
SECURITY
SOFTWARE
SYSTEM
HKEY_USERS
.DEFAULT
S-1-5-18
S-1-5-19
S-1-5-20
S-1-5-21-309235459-240059954-4066018515-1001
S-1-5-21-309235459-240059954-4066018515-1001_Classes
HKEY_CURRENT_CONFIG
Software
System
Name
Type
Data

Issue 2492: Windows registry predefined keys may lead to confused deputy problems and local privilege escalation

Reported by mjurczyk@google.com on Fri, Oct 6, 2023, 11:44 AM GMT+2 Project Member

Code

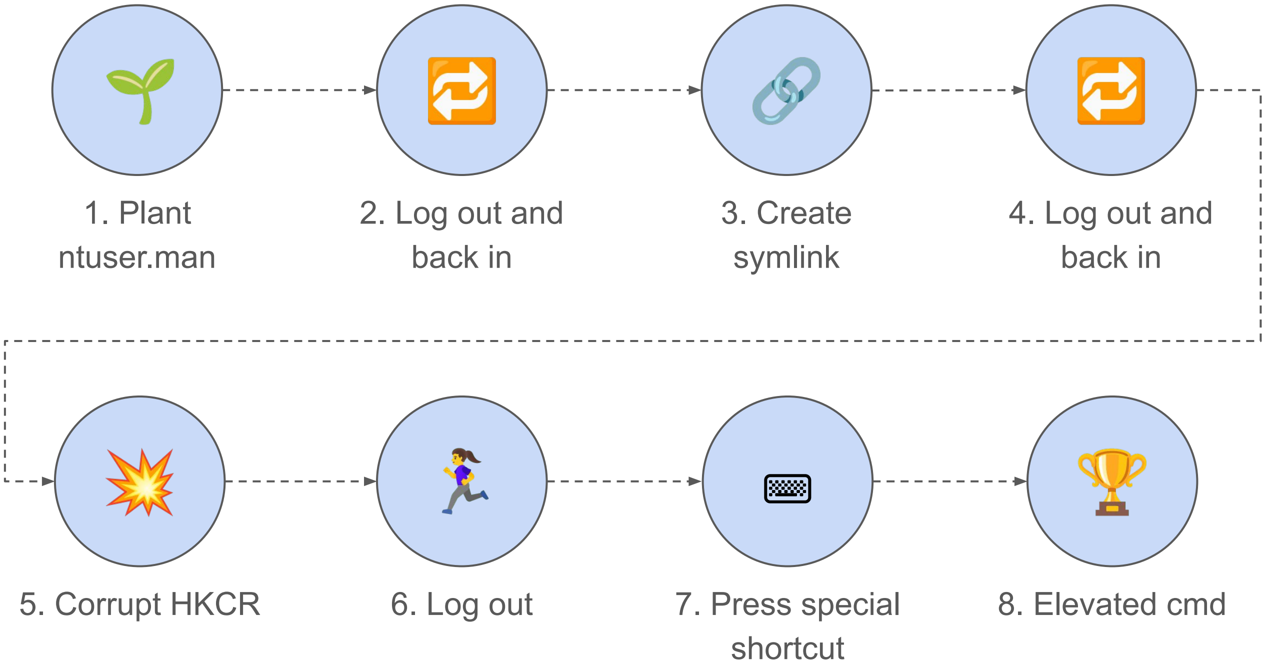
In Windows Registry, predefined-handle keys are a special type of keys similar to symbolic links, but instead of transparently redirecting to an arbitrary registry path, they redirect to an arbitrary predefined registry key (HKLM, HKCU, HKCR etc., see [1] for a full list). The concept of symbolic links makes the system potentially prone to security bugs, in situations where a privileged process (e.g. winlogon or a system service) operates on user-controlled keys. By abusing symbolic links, such processes could be tricked into reading from or writing to a different key that they originally intended to, allowing a local attacker to elevate their privileges in the system. For this reason, there is a mechanism in the Windows registry called "trust classes", which prevents traversing symbolic links originating from untrusted hives (such as user hives) pointing to trusted hives (such as global system hives). Internally, the verification of this security boundary is implemented in the CmpOKToFollowLink kernel function.

The problem discussed in this report is the fact that predefined keys don't have a similar safety mechanism, which means that a local user may redirect any key within their HKEY_CURRENT_USER hive to any of the possible predefined keys, including some system-wide ones. This behavior may potentially allow crossing a security boundary, but successful exploitation depends on finding a privileged process that opens a key inside HKCU and does something "interesting" with it. We have found one such candidate in the form of the System Event Notification Service (SENS), which is implemented by the sens.dll library that also extensively calls into es.dll (probably standing for Event System). This service gets notified about all user logon/logoff events in the system, and when that happens, a series of function calls leads to es!CreateEventSystemKey. This routine opens the HKCU\Software\Microsoft\EventSystem key in the hive of the user that is just logging in, and sets its security descriptor to a very permissive DACL which grants the user full access (KEY_ALL_ACCESS) to the key and all of its subkeys (the specific DACL string is formatted in es!InitializeStringSecurityDescriptorForEventSystemKey).

Plan of attack

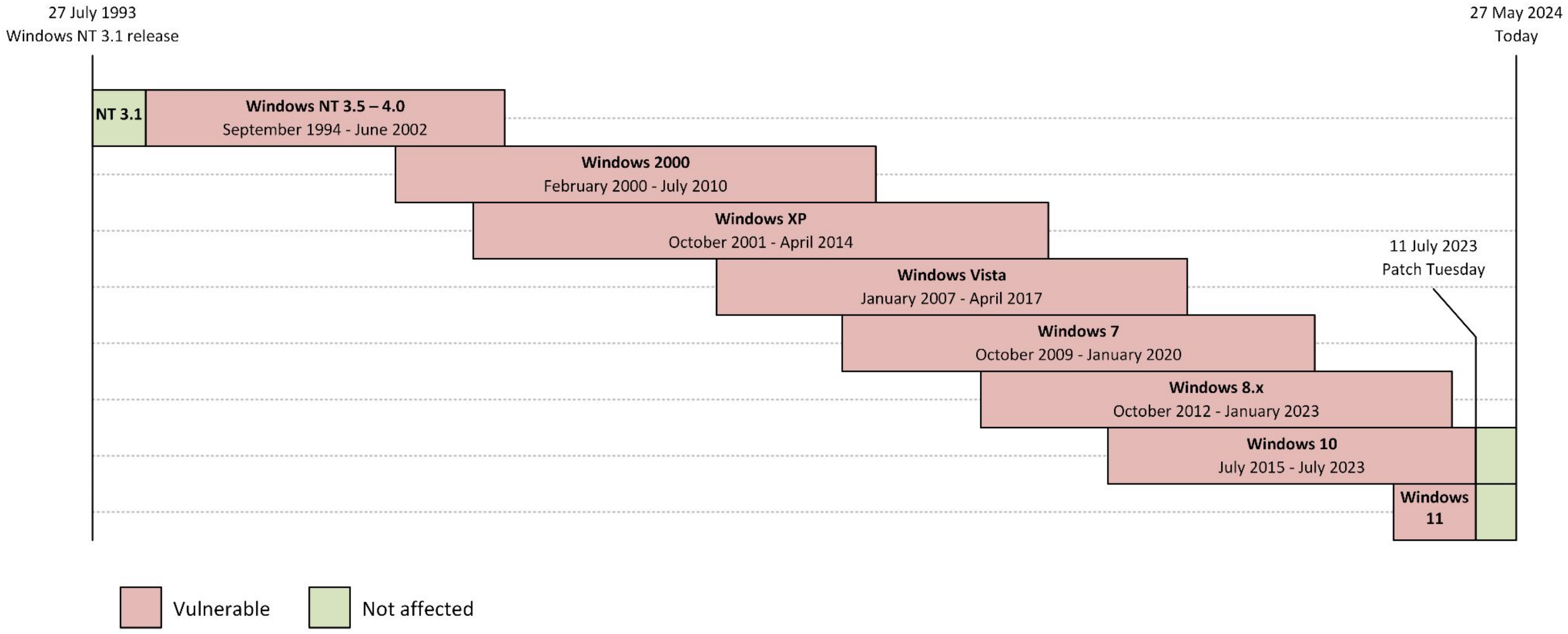
1. HKCU\Software\Microsoft\EventSystem → HKEY_CURRENT_CONFIG
2. *System Event Notification Service* (svchost.exe) unknowingly sets a permissive descriptor on HKCC, granting us write access
3. HKCC\Link → HKCR\TypeLib\{GUID}\2.0
4. Trigger the bug a second time to gain control over the COM object
5. Corrupt a COM object used by a System process, elevate privileges

Attacker's view



Demo

Predefined key timeline



Predefined key summary

- A completely undocumented feature lived in the format for almost 30 years
- Demonstrates the strengths of logic bugs
  - Unfuzzable
  - Breaks high-level security guarantees
- Requires comprehensive knowledge of the target for exploitation
  - Identifying the fundamental problem with the feature
  - Finding the right set of primitives
    - Binary control over HKCU via ntuser.man
    - A system service that performs "abusable" operations on HKCU

Conclusion

Takeaways

- The registry is a fascinating research target
- If you're a researcher: persistent analysis pays off
  - Fuzzing is often only scratching the surface
  - For some targets, the really good bugs come from a deep understanding of software
- If you're a vendor: some features shouldn't live forever
  - Legacy code is a security hazard and should be periodically reevaluated
  - Attack surface reduction and well-placed mitigations have an outsized impact on security