Attacking Assumptions Behind the Image Load Callbacks

ROMHACK 2025

Denis Nagayuk
(diversenok)

WHO AM I

Denis Nagayuk

- **Security Researcher** at

HUNT & HACKETT

Interests:

- Windows Internals
- System Programming
- Reverse Engineering

Social media:

diversenok

on Twitter, GitHub, Discord,

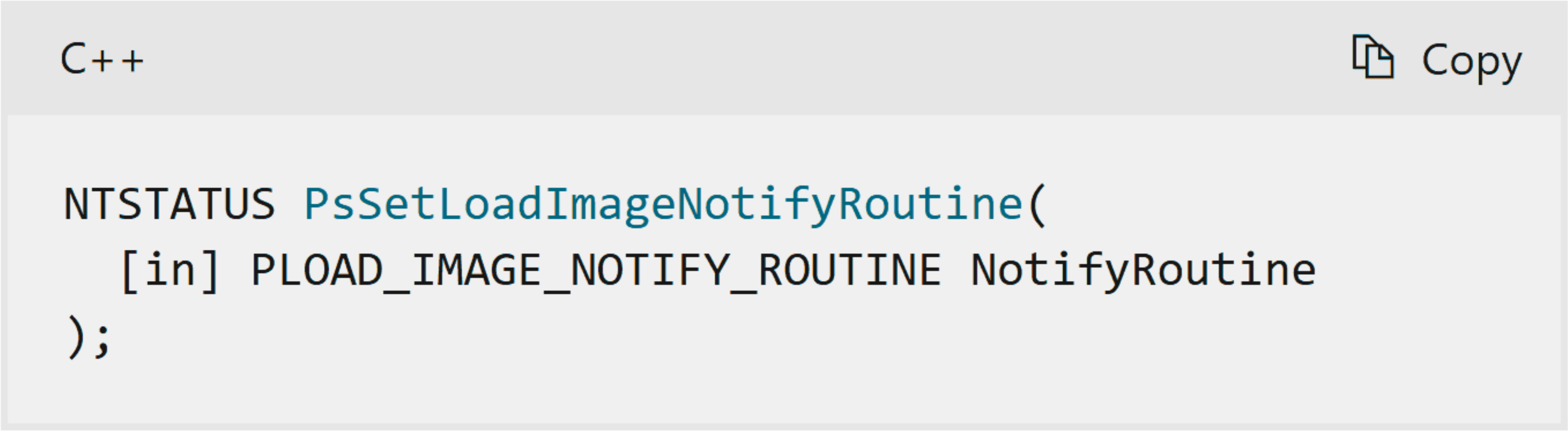
diversenok.github.io

Contributions:

NtDoc, System Informer, phnt, etc.

TOPIC

- Built-in Windows feature
- Loading **DLLs** maps executable **images** into memory
- The operation triggers a **kernel callback** that notifies interested drivers
- Considered a reliable mechanism



```
NTSTATUS PsSetLoadImageNotifyRoutine(
  [in] PLOAD_IMAGE_NOTIFY_ROUTINE NotifyRoutine
);
```

MOTIVATION

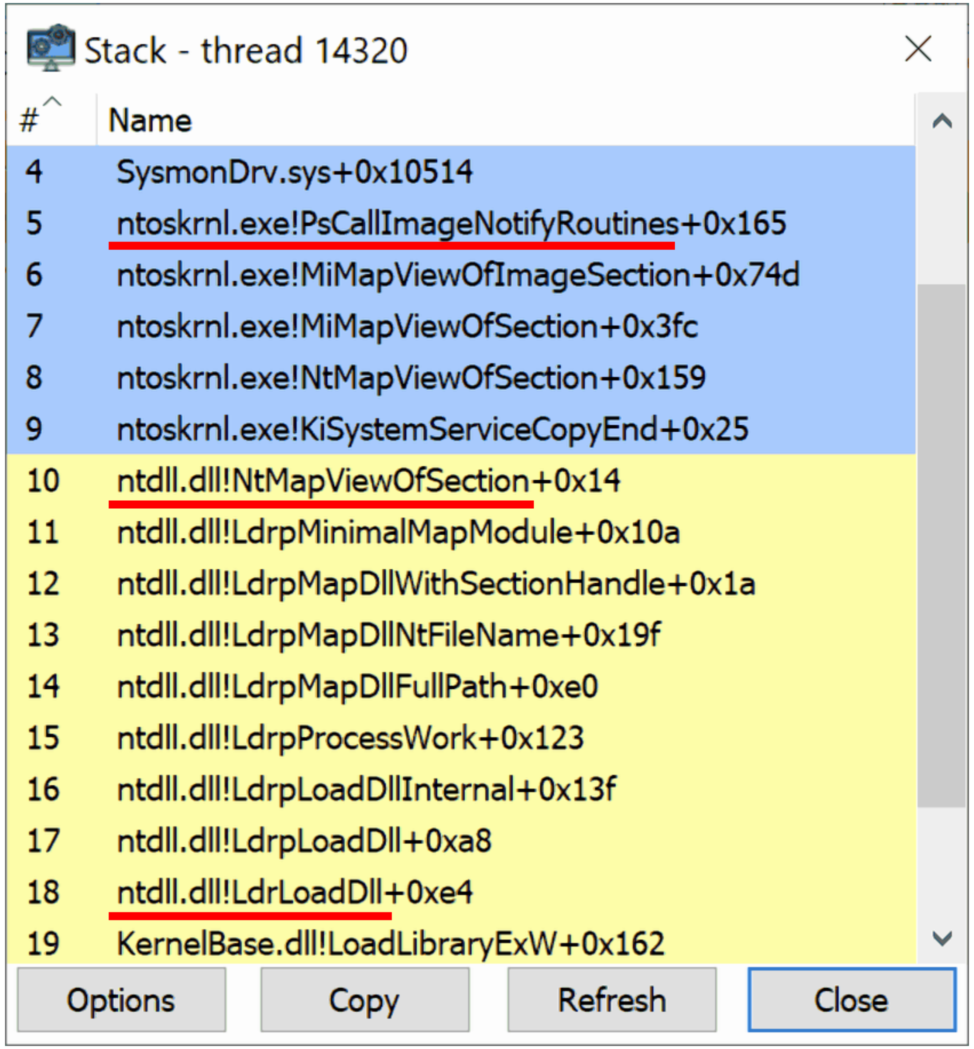
Defenders:

- Log & analyze
  - Sysmon Event ID 7
  - Many AV/EDR products
- Enforce custom security (code integrity) policy
  - System Informer
  - EDRs with custom PPL implementations

Attackers:

- Break all of the above

APPLICATION PERSPECTIVE



Stack - thread 14320

```
#   Name
4   SysmonDrv.sys+0x10514
5   ntoskrnl.exe!PsCallImageNotifyRoutines+0x165
6   ntoskrnl.exe!MiMapViewOfImageSection+0x74d
7   ntoskrnl.exe!MiMapViewOfSection+0x3fc
8   ntoskrnl.exe!NtMapViewOfSection+0x159
9   ntoskrnl.exe!KiSystemServiceCopyEnd+0x25
10  ntdll.dll!NtMapViewOfSection+0x14
11  ntdll.dll!LdrpMinimalMapModule+0x10a
12  ntdll.dll!LdrpMapDllWithSectionHandle+0x1a
13  ntdll.dll!LdrpMapDllNtFileName+0x19f
14  ntdll.dll!LdrpMapDllFullPath+0xe0
15  ntdll.dll!LdrpProcessWork+0x123
16  ntdll.dll!LdrpLoadDllInternal+0x13f
17  ntdll.dll!LdrpLoadDll+0xa8
18  ntdll.dll!LdrLoadDll+0xe4
19  KernelBase.dll!LoadLibraryExW+0x162
```

Works transparently:

1. LoadLibrary
2. LdrLoadDll
3. NtMapViewOfSection*
4. PsCallImageNotifyRoutines

* *Section* refers to a memory mapping object

SCOPE

Multiple types of memory:

- ✖ MEM_PRIVATE – NtAllocateVirtualMemory
- ✖ MEM_MAPPED – NtCreateSection with SEC_COMMIT

} Obvious bypasses

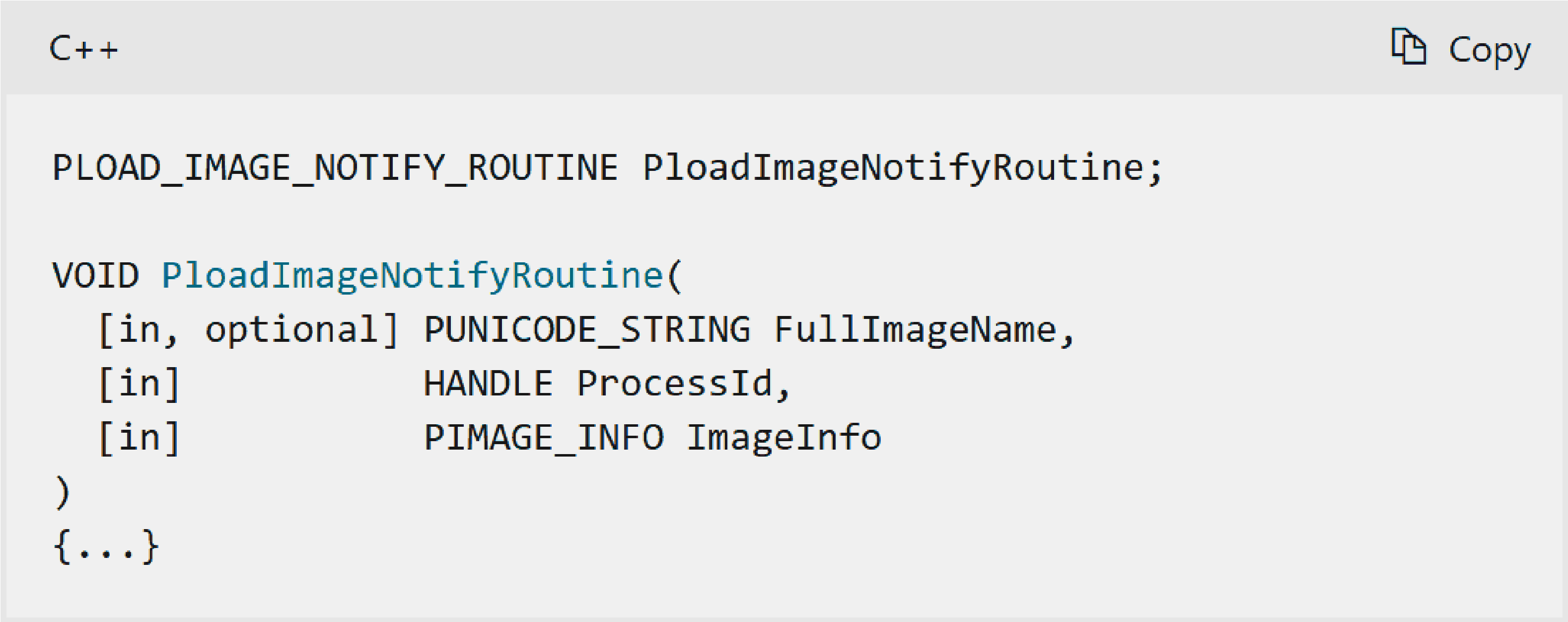
- ✓ MEM_IMAGE – NtCreateSection with SEC_IMAGE

Need to know what it is and what isn't:

- Notifies about **images**, not *any* executable code
- Want to block non-image code? See *Arbitrary Code Guard* (ACG)

DRIVER PERSPECTIVE

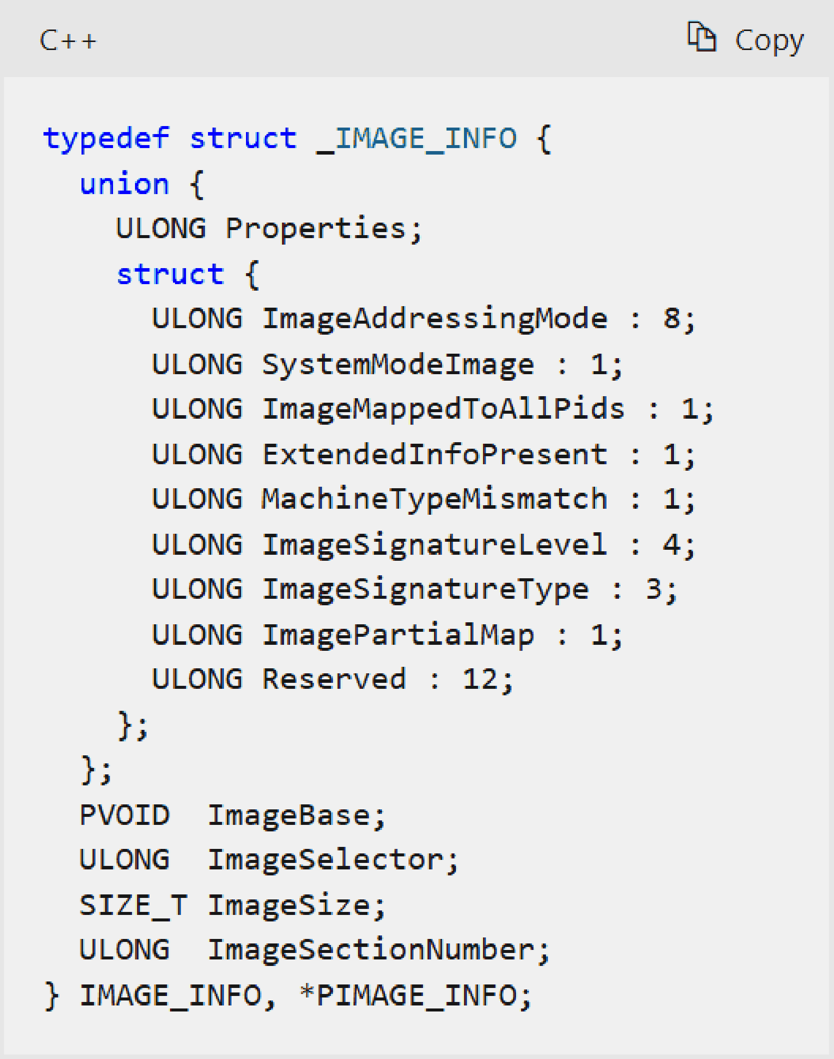
- Registration is **documented** on MSDN
- The driver provides a function, the system invokes it



```
PLOAD_IMAGE_NOTIFY_ROUTINE PloadImageNotifyRoutine;

VOID PloadImageNotifyRoutine(
  [in, optional] PUNICODE_STRING FullImageName,
  [in]           HANDLE ProcessId,
  [in]           PIMAGE_INFO ImageInfo
)
{...}
```

AVAILABLE INFORMATION



```cpp
typedef struct _IMAGE_INFO {
  union {
    ULONG Properties;
    struct {
      ULONG ImageAddressingMode : 8;
      ULONG SystemModeImage : 1;
      ULONG ImageMappedToAllPids : 1;
      ULONG ExtendedInfoPresent : 1;
      ULONG MachineTypeMismatch : 1;
      ULONG ImageSignatureLevel : 4;
      ULONG ImageSignatureType : 3;
      ULONG ImagePartialMap : 1;
      ULONG Reserved : 12;
    };
  };
  PVOID  ImageBase;
  ULONG  ImageSelector;
  SIZE_T ImageSize;
  ULONG  ImageSectionNumber;
} IMAGE_INFO, *PIMAGE_INFO;
```

- Process ID [1]
- Full image **name** (in NT format) [1]
- Base **address** + size
- Signing level (MS binaries only)
- Some flags
- File object pointer [2]

[1] As the function parameter

[2] In -Ex version

BEHAVIOR ESSENTIALS

- **Post-operation**

  We do get a base address we can read

- **No cancellation**

  Can still unmap, bearing compatibility issues (no status code change)

- **Synchronous**

  Unlike ETW

SYNCHRONOUS BUT RACY

Pitfall alert:

- While the calling thread is **stuck** in kernel mode, the section is **already mapped** and usable by other threads.

Thoughts:

- Might come in handy if we can prolong callback execution…

RESEARCH QUESTIONS

1. What **OS mechanisms** are involved?
2. What **API surface** and opportunities do we have for interacting with them?
3. What **assumptions** does the callback delivery and payload rely on?
4. How can we **violate** these assumptions?
5. How can we **mitigate** the damage?

FILES AND SECTIONS

Mapping is a three-step process:

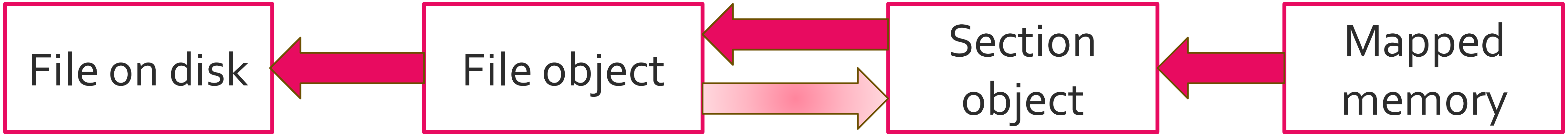
1. **Open** a **file** object – NtOpenFile/NtCreateFile
2. **Create** a **section** object from the file object – NtCreateSection with SEC_IMAGE
3. **Map** the **section** – NtMapViewOfSection

- Step 2 requires a file
- Step 3 requires a section, but not the file (i.e., we can close it after 2)

INDIRECTION

Extra levels of **indirection**:

- Makes sense from the design perspective
- More **points of influence**
- More **caching**, more opportunities for mismatch



ASSUMPTIONS

OS-level assumptions:

1. The file still **exists**
2. Its **name** is possible to **query**
3. The name is **correct**

Driver-level assumptions:

4. The file is possible to **open**
5. Opening **yields** the correct file
6. The file is possible to **read**
7. The content **corresponds** to memory

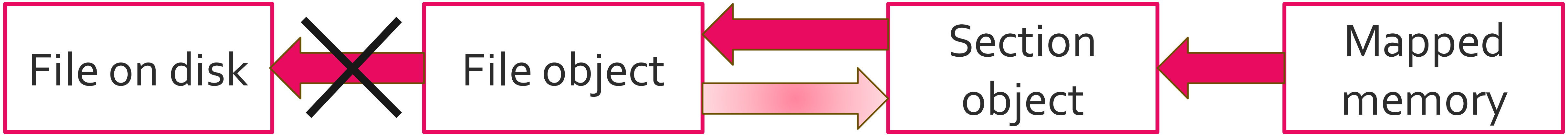
ASSUMPTION 1: FILE STILL EXISTS

Idea: Indirection gives greater control over file lifetime

Caveat: Cannot detach the file object from the section object

Solution: Make sure it doesn’t correspond to anything on disk

Effect: No file, no name to report



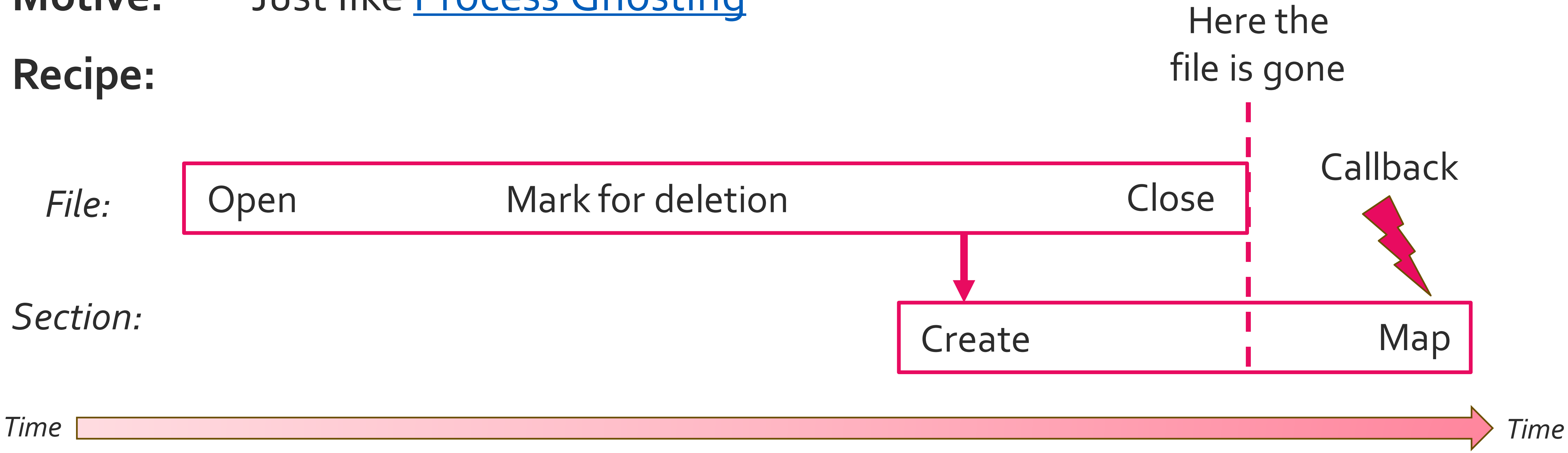
ATTACK 1A: EARLY DELETION

Problem: Cannot delete a file in use by a section (STATUS_CANNOT_DELETE)

Solution: Mark for deletion **before** creating a section

Motive: Just like [Process Ghosting]()

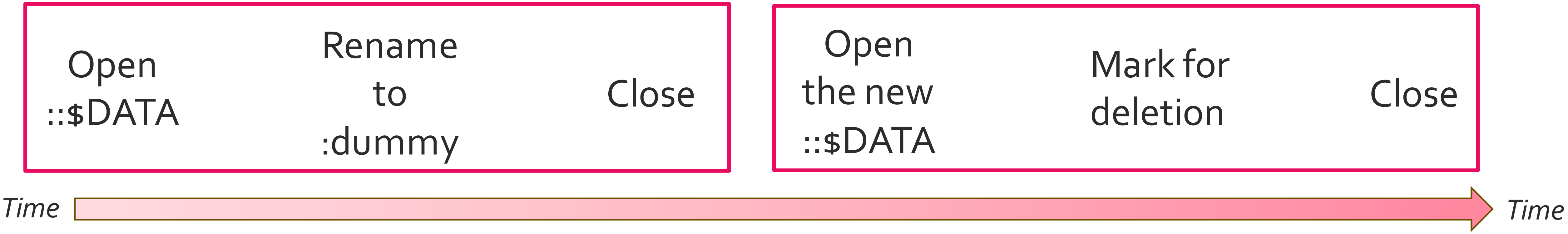
Recipe:



ATTACK 1B: SELF-DELETION

An NTFS-specific trick from Jonas Lyk for deleting locked files via stream rotation:

- Locking applies **per-stream**
- Streams can be **renamed**
- **Deleting** the primary ::$DATA stream deletes **all other** streams



Open ::$DATA

Rename to :dummy

Close

Open the new ::$DATA

Mark for deletion

Close

Time

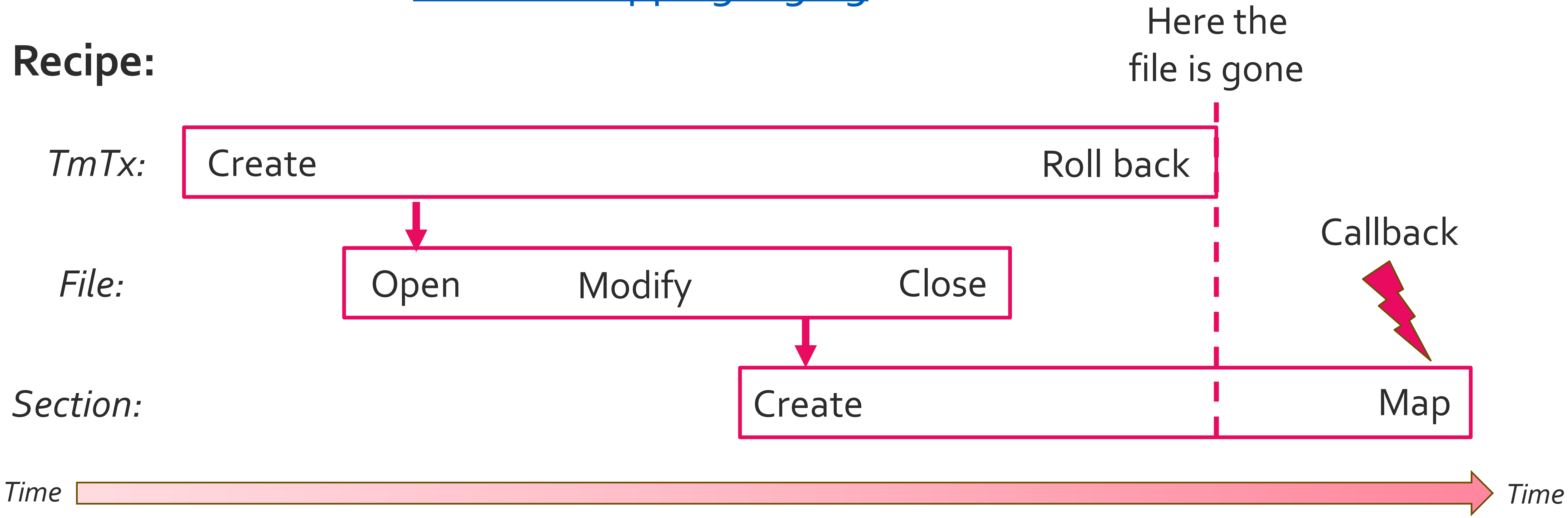
Time

ATTACK 1C: INACTIVE TRANSACTION

Idea: Transacted operations have a scope; can roll back everything.

Motive: Just like [Process Doppelgänging]

Recipe:

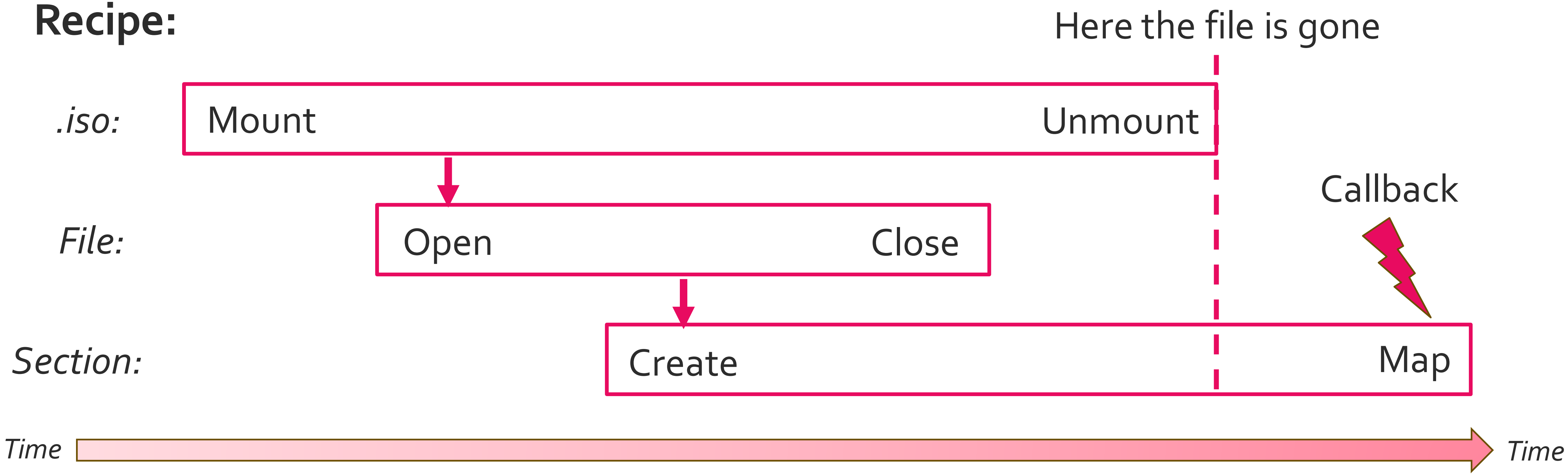


ATTACK 1D: UNMOUNTED VOLUME

Idea: Files belong to a volume

Caveat: Need a disposable volume, preferably without admin

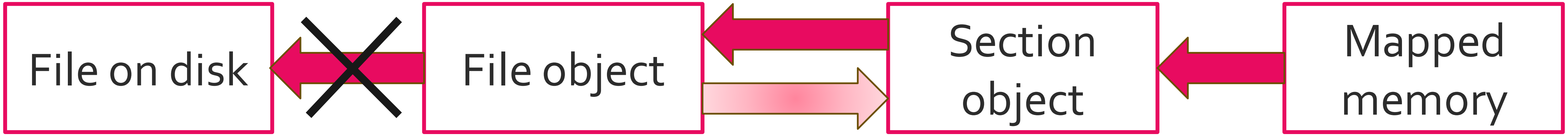
Recipe:



OBSERVING EFFECTS

- Attempting to **query** the **name** yields:
  - STATUS_FILE_DELETED for attacks 1A and 1B
  - STATUS_TRANSACTION_NOT_ACTIVE for attack 1C
  - STATUS_VOLUME_DISMOUNTED for attack 1D

Sysmon **ignores** these events



ASSUMPTION 2: NAME EXISTS

Consider **restrictions** on filenames:

- **Special characters**
  - Blocked by APIs
  - Patched volumes give STATUS_FILE_CORRUPT_ERROR
- **Length**
  - Overflow something?

LONG NAMES

What is the **limit** anyway?

260 aka. MAX_PATH?

- No, it’s a legacy Win32 limit

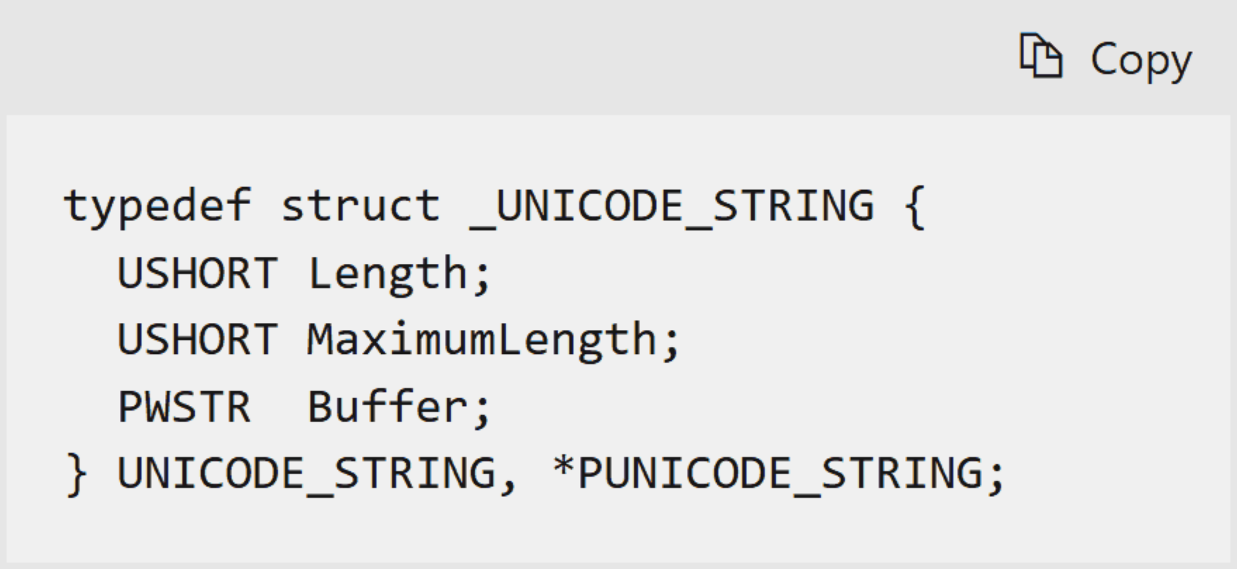
32767?

- Yes, but no
- Also, why this number?

LONG LONG NAMES

The limit comes from how Windows **addresses strings** (UNICODE_STRING):

- USHORT (0..65535) bytes in length or max **32767** wide characters



```
typedef struct _UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

ATTACK 2A: NAME OVERFLOW

A filename consists of **two parts**:

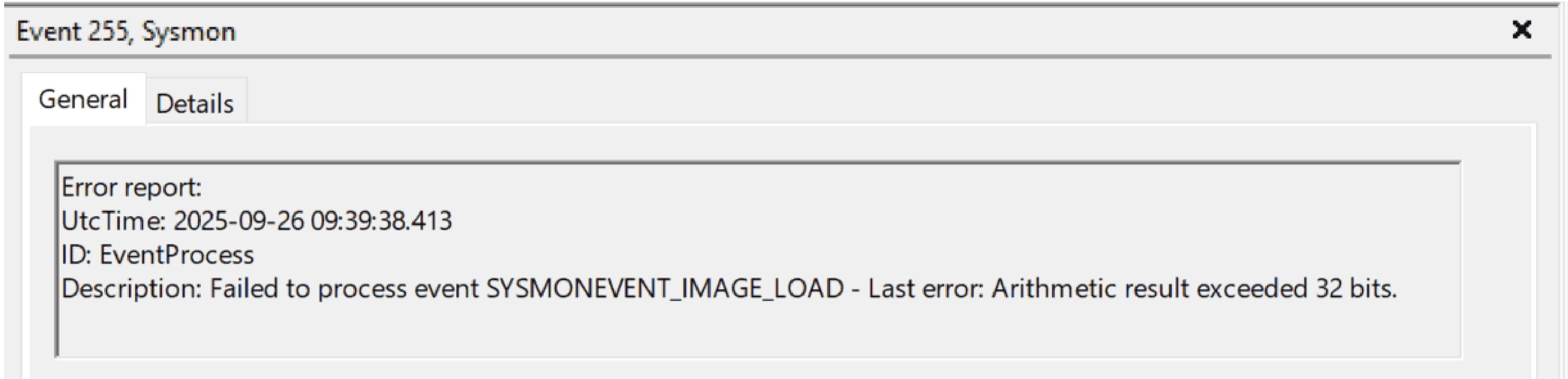
- A **volume name** – \Device\HarddiskVolume1
- A path **on the volume** – \Windows\system32

Filesystem drivers deals with the on-volume path

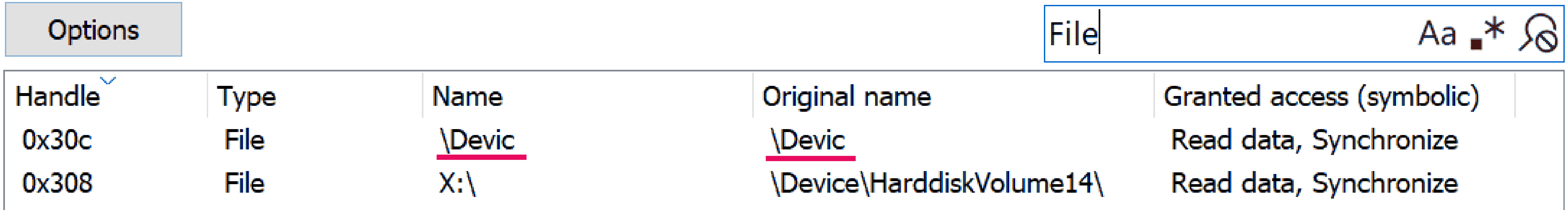
- NTFS allows this part to be **up to** UNICODE_STRING limit
- The **full name** (after concatenation) **might not fit**!
- The file **exists** but impossible to open by full name. Only relative.

HOW NAME OVERFLOW LOOKS

- **Sysmon** either fails with integer overflow or reports a broken name



```
Error report:
UtcTime: 2025-09-26 09:39:38.413
ID: EventProcess
Description: Failed to process event SYSMONEVENT_IMAGE_LOAD - Last error: Arithmetic result exceeded 32 bits.
```



Handle	Type	Name	Original name	Granted access (symbolic)
0x30c	File	\Devic	\Devic	Read data, Synchronize
0x308	File	X:\	\Device\HarddiskVolume14\	Read data, Synchronize

The handle looks cursed...

ASSUMPTION 3: THE NAME IS CORRECT

- Problem with filenames: they are **non-owning** references

After rename:

- **Cached** names (strings) become **outdated**
- But what about **queries** against file and section objects?

RENAME TRACKING

Sections: Always **ask** the underlying file object

File objects: **Depends** on the filesystem…

Experiment: Open, rename, query name

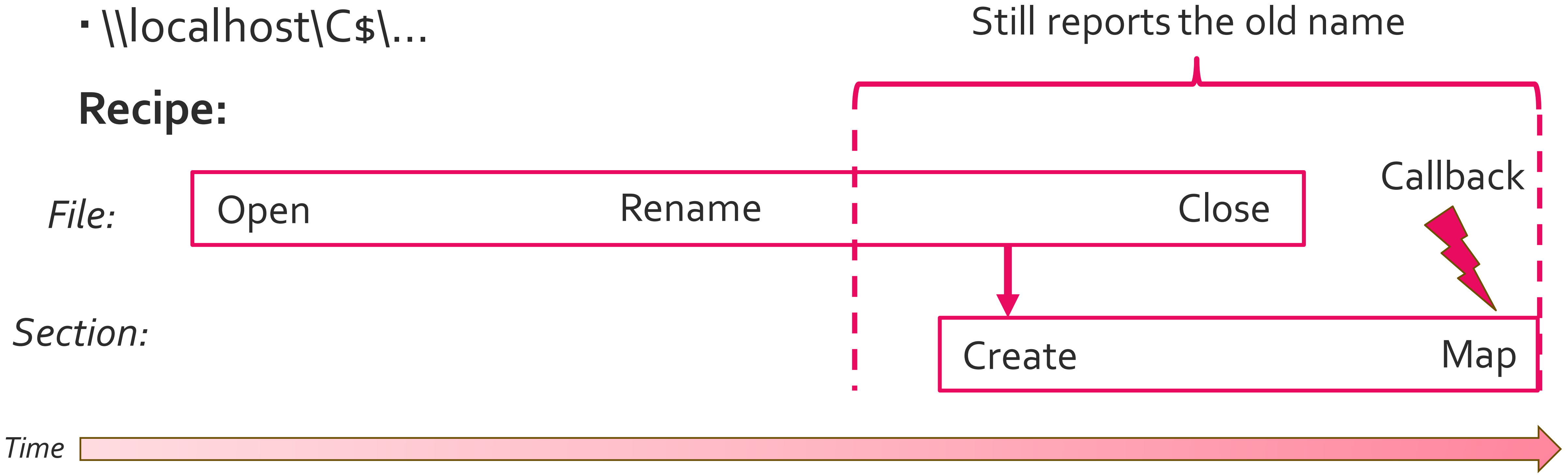
Rename on…	**NTFS**	**\Device\Mup**
The **same** handle	✓ Updated	✗ **Outdated**
Another handle	✓ Updated	✗ **Outdated**

ATTACK 3A: MUP RENAME

LanmanRedirector (a Multiple UNC Provider) **does not** track renames

- \Device\Mup\localhost\C$\...
- \\localhost\C$\...

Recipe:



HARD LINKS

Hard links:

- Allow **multiple names** for the same on-disk file
- Creation is similar to renaming but leaves the old name behind

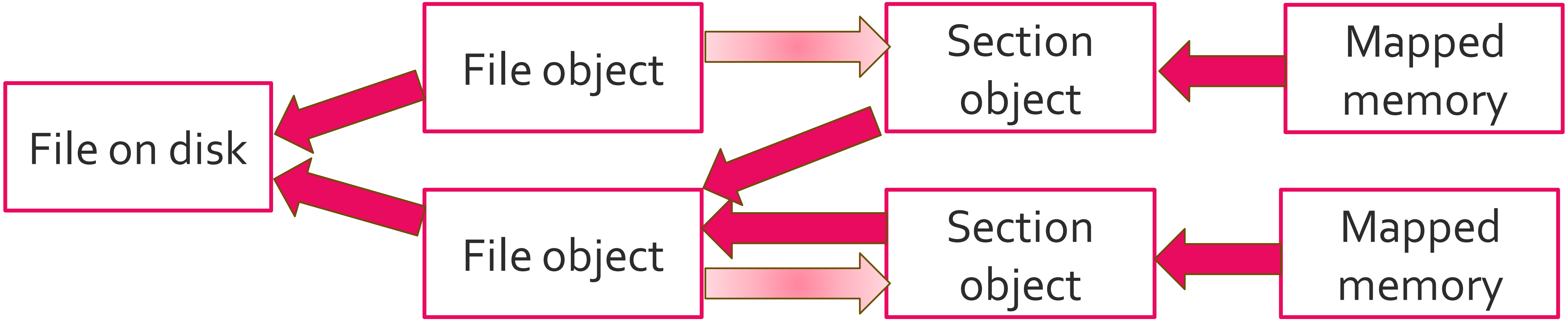
Question:

- **Two** hard links, refer to the same content. We **map both**.
- Which **name** will we get?

HARD LINKS & SECTIONS

Answer:

- Whichever **file object** happened to get **cached** in the section
- Usually not a problem (still the same content)



HARD LINK DELETION

Annoying issue:

- Can hardlink locked files but **cannot delete** (undo)
- Trying to set FileDispositionInformation returns STATUS_CANNOT_DELETE

Solution:

- FileDispositionInformation**Ex** (since RS1) **can**

Rules:

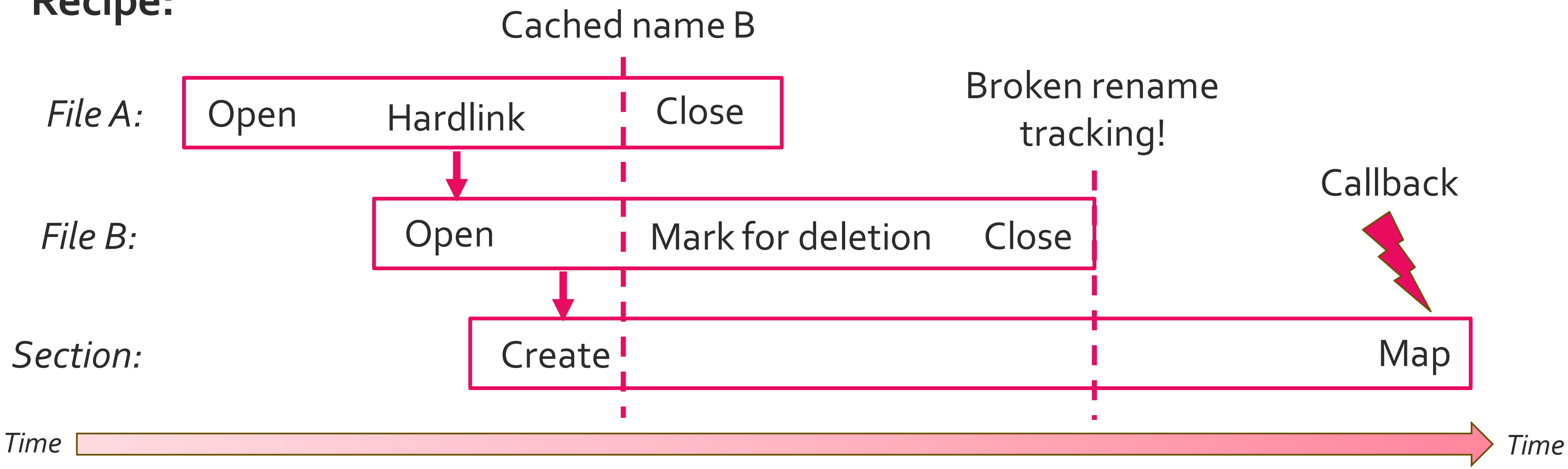
- **Non-Ex** is -**Ex** plus FILE_DISPOSITION_FORCE_IMAGE_SECTION_CHECK
- Omitting **allows deleting** hard links, **up until** the **last** one.

ATTACK 3B: HARD LINK DELETION

Primitive 1: Can **choose** which hard link **name** to return from section

Primitive 2: Can **delete names** until there is only one left

Recipe:



THE X64DBG BUG REPORT

A user opened an **issue** (#2990) in **x64dbg**.

- x64dbg **failed to resolve** a file reported by an **image load** debug event
- Looks like the event returns a **stale name**

The user accidentally discovered an attack on rename tracking.

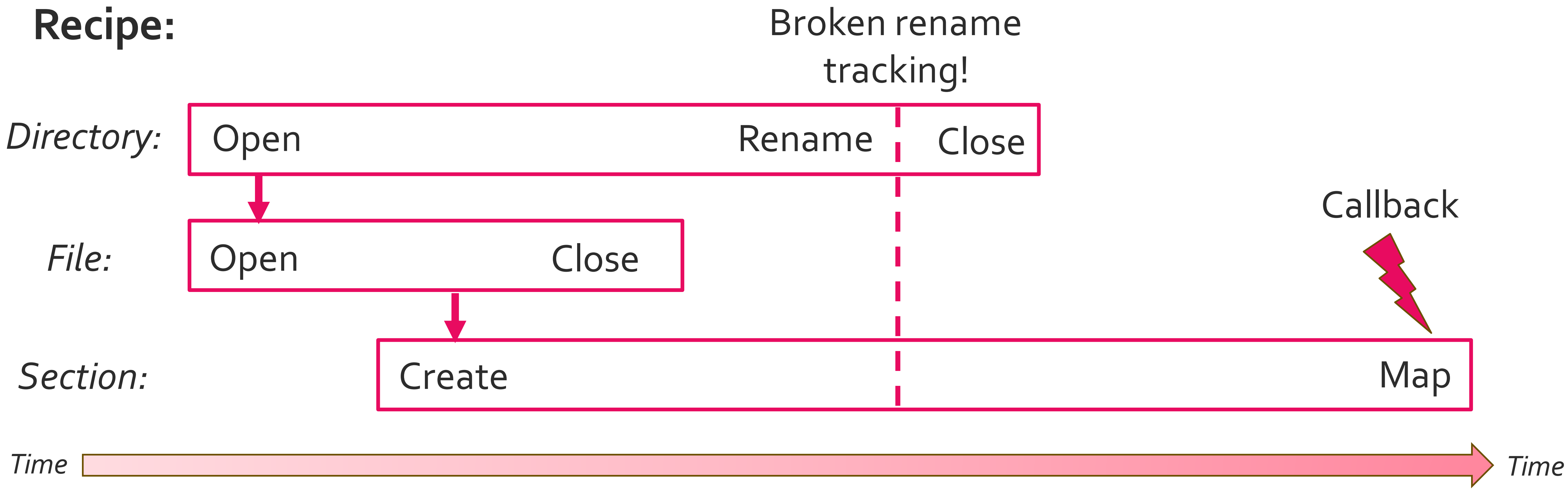
See the discussion on ***winsiderss*** Discord.

ATTACK 3C: PARENT RENAME

Problem: Cannot **rename** a **parent** directory if there are file handles inside

Solution: Keep a section handle instead

Recipe:



ASSUMPTION 4: POSSIBLE TO OPEN

Now to driver assumptions. Anything can prevent opening?

- **Security descriptors**
  - A user-mode concern; even admins can bypass
- **Sharing mode**
  - Drivers can bypass
  - Can also be self-inflicted
    - Like an antivirus that fails to scan a file if somebody has a DELETE handle to it.
- **EFS**
  - Remember the trick for encrypting Defender's executable so it cannot start?

OPLOCKS

The favorite **mechanism** for winning **race conditions**.

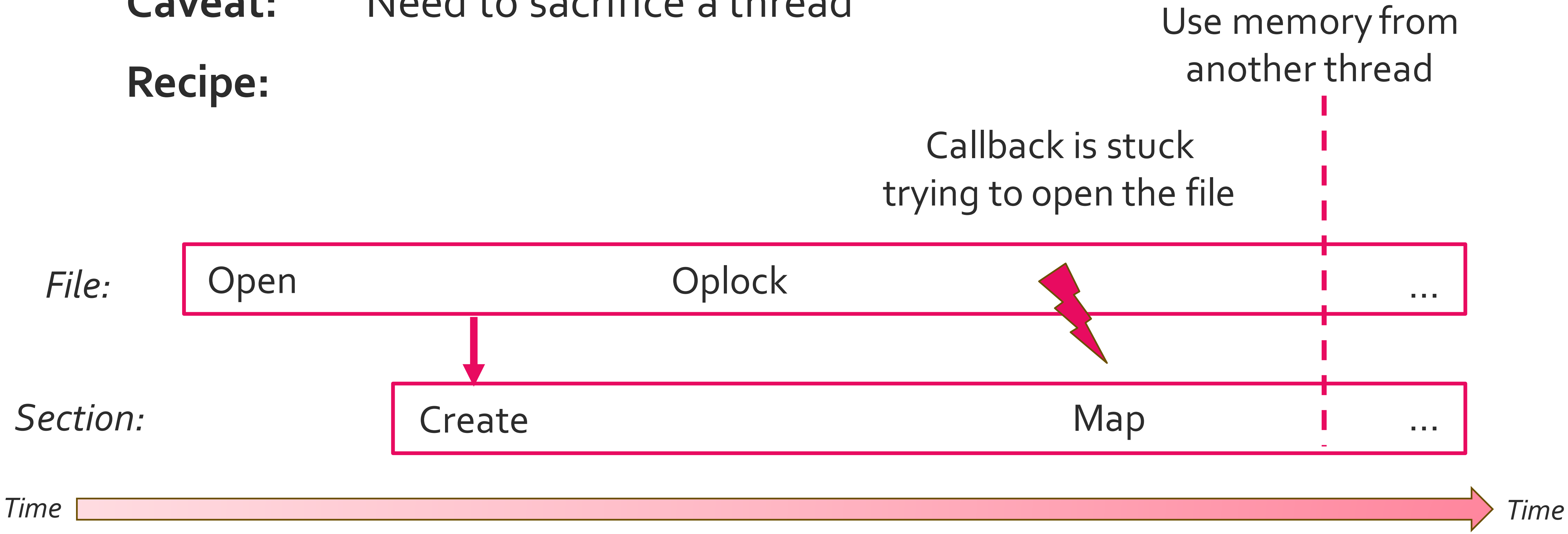
- Oplocks can **postpone** open until **acknowledgement** (indefinitely)
- Many different types
  - Covering open, write, delete
- Batch oplocks
  - Everything beyond FILE_READ_ATTRIBUTES | FILE_WRITE_ATTRIBUTES | SYNCHRONIZE

ATTACK 4A: OPLOCKS

Idea: Abuse post-operationness

Caveat: Need to sacrifice a thread

Recipe:

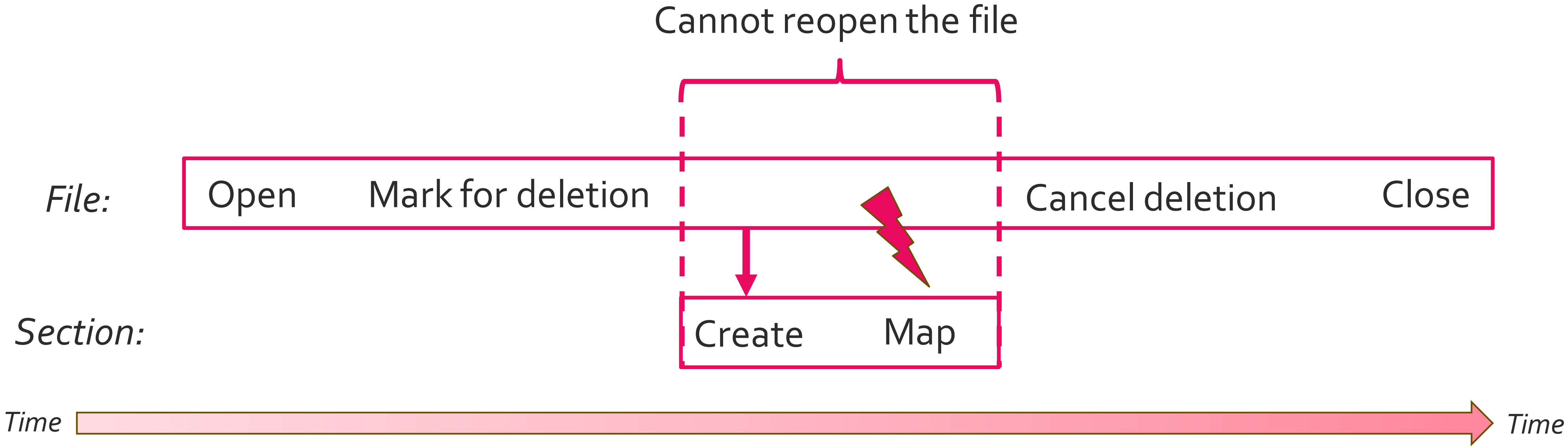


Use memory from another thread
Callback is stuck trying to open the file
File:
Open
Oplock
...
Section:
Create
Map
...
Time
Time

ATTACK 4B: DELETE-PENDING

Idea: Opening files marked for deletion fails with STATUS_DELETE_PENDING

Recipe:



ASSUMPTION 5: OPENS THE RIGHT FILE

The correct name is **not enough.** Also need it to be:

- Not **ambiguous**
- Not **redirected**

Ways to redirect:

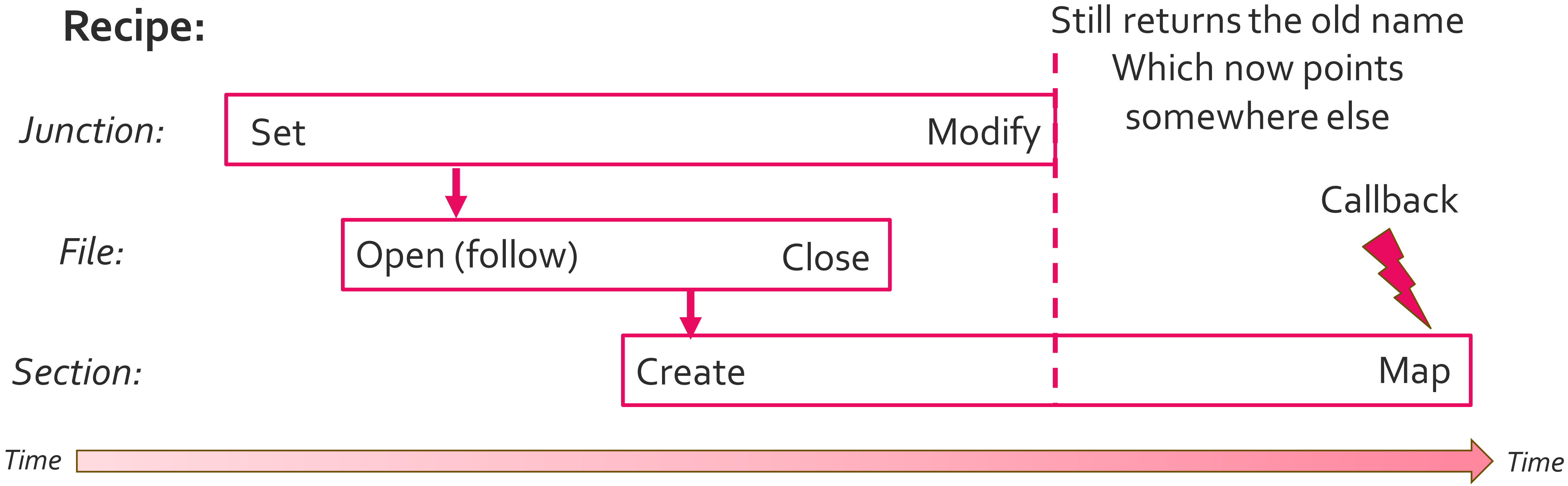
- **Junctions**
- Namespace symlinks
- Filesystem symlinks

ATTACK 5A: MUP & JUNCTIONS

Problem: We receive the name after reparse point resolution

Solution: Maybe on NTFS, but not on \Device\Mup

Recipe:

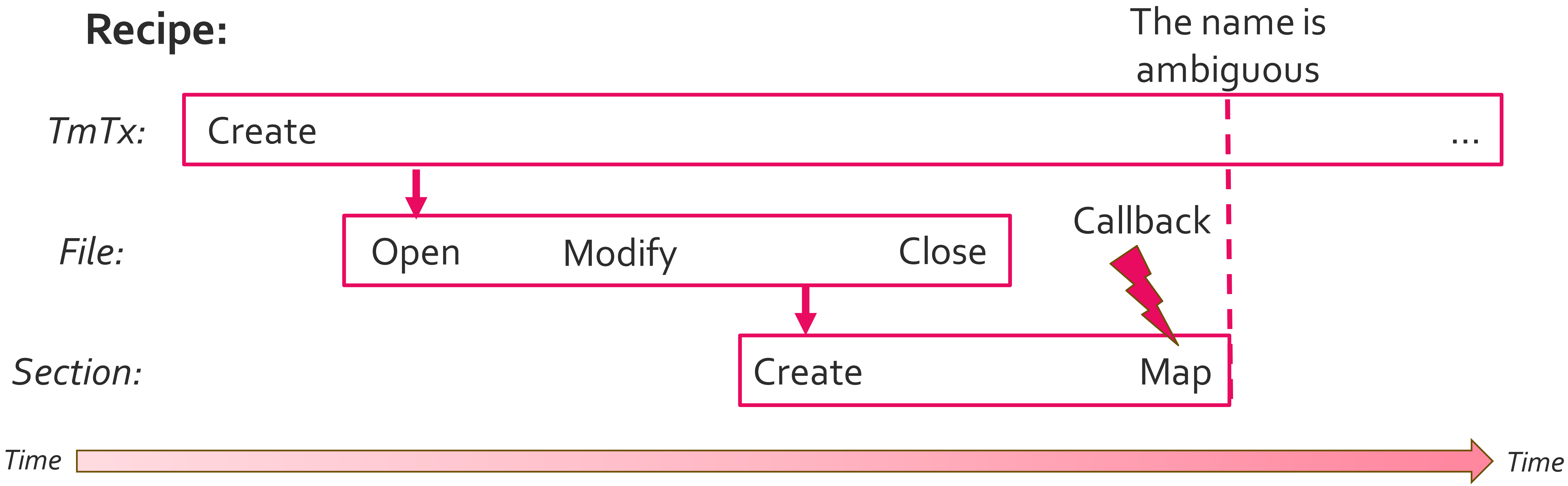


ATTACK 5B: TRANSACTIONAL AMBIGUITY

Idea: Transactions allow one “file” to be in two states at once.

Problem: A filename is not enough without transactional context.

Recipe:



ASSUMPTION 6: POSSIBLE TO READ

Want to **hash** the file? Need to **read** after opening.

Memory:

- Race and set PAGE_GUARD

File:

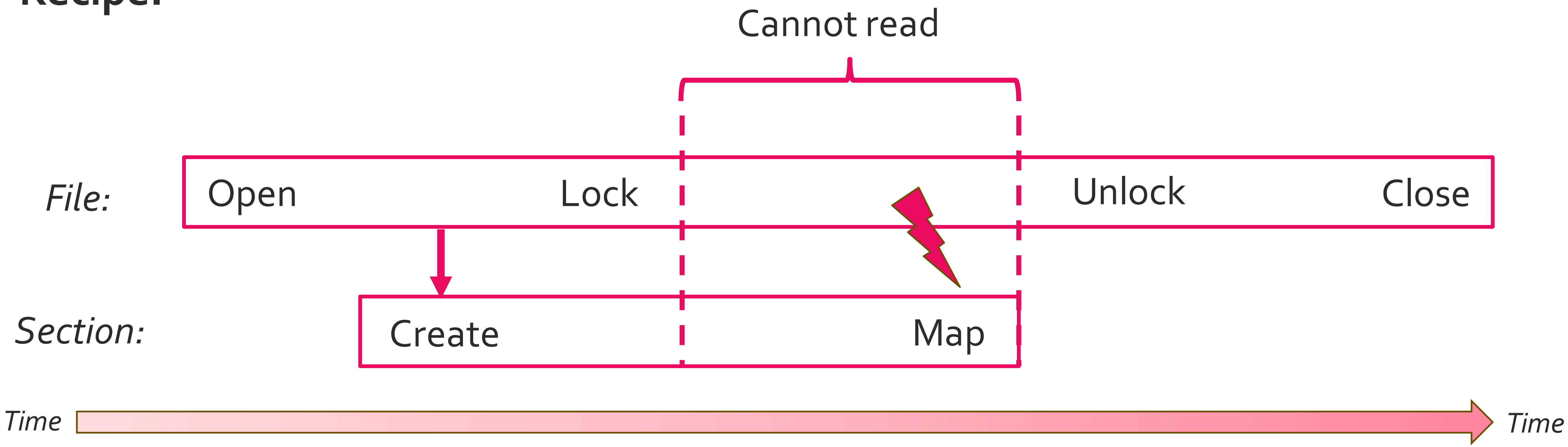
- Somehow cause an error?

ATTACK 6A: BYTE LOCKS

Idea: NtLockFile can grab ranges for exclusive access

Caveat: Blocks NtReadFile (STATUS_FILE_LOCK_CONFLICT) but not mapped I/O

Recipe:



ASSUMPTION 7: FILE & MEMORY MATCH

Mapped image memory is **copy-on-write** – extra caching.

More attacks:

- False Immutability (by Gabriel Landau)
- Process Herpaderping (by Johnny Shaw)

This margin is too narrow… Maybe another time

THE SYSTEM INFORMER CASE

Remember **Process Hacker**?

- An open-source Windows internals-oriented task manager
- Had a driver for extra capabilities
- The driver requires admin, but MS and vendors were not happy.

System Informer – an updated version under *Winsider Seminars & Solutions*. The new driver respects PPL for modifications but still offers great insight.

DRIVER HARDENING

Goal: Need to protect from abuse

Problem: Cannot use PPL (not antimalware)

Extras: Want to support plugins

Need to **re-invent** protections:

- Process & thread handle filtration via **Ob-** callbacks
- Custom code integrity for plugins via **image load** notifications
- Two-phase restart
- Mitigations

ATTACKS THAT WORKED

Challenge: Plant an unsigned plugin to be recognized as signed

Solutions: Name desync, content desync, open redirection.

Example:

1. Prepare an ***unsigned.dll*** with a section that cached ***signed.dll***'s name.
2. Start System Informer
3. It will load ***unsigned.dll*** but validate signature for ***signed.dll***
4. Since there are no unsigned plugins, the driver allows sensitive operations.

MITIGATIONS: FUNDAMENTALS

- Always look for the **-Ex version** of the structure!
  - It gives a **file object**

No, it doesn’t solve all the problems

- The object is in cleanup phase and barely usable

Johnny Shaw and I looked into **reopening** the file from this object (so we don’t have to deal with filenames), but no luck – need a handle, not an object pointer (and cannot upgrade).

MITIGATIONS: QUERYING NAMES

- **Try** querying **harder**:
  - NtQueryVirtualMemory *with* MemoryMappedFilenameInformation does not have a UNICODE_STRING limit
  - Also distinguishes deleted/unmounted/etc. via returned status
  - FltGetFileNameInformationUnsafe can return a different result
- Explicitly **choose** what to do with non-existing files
  - Ignore? Abort? Assume the worst?

MITIGATIONS: OPENING FILES

- Avoid access checks via Zw- and Io- functions
- Bypass sharing mode via IO_IGNORE_SHARE_ACCESS_CHECK
- Be aware of transactions (check them on the file object)
- Use FILE_COMPLETE_IF_OPLOCKED and check for STATUS_OPLOCK_BREAK_IN_PROGRESS
- Use OBJ_DONT_REPARSE if need to avoid redirection (bearing compatibility issues)
- Compare file objects similar to NtAreMappedFilesTheSame

MITIGATIONS: READING CONTENT

- Just don’t depend on it. **Validate memory**, not file.
- Be ready to switch to mapped I/O on STATUS_FILE_LOCK_CONFLICT
- Do image coherency checks (see System Informer’s code)

NOT A VULNERABILITY

Look at Windows **Code Integrity**

- Provides **signing levels** and validation for **PPL**, PP, and kernel **drivers**
- Does not suffer from these attacks
- Validates memory, cares little about data on disk
- Not too optimistic to grant “success” on signature validation anomalies

TAKEAWAY

Interesting and powerful mechanism with lots of caveats.

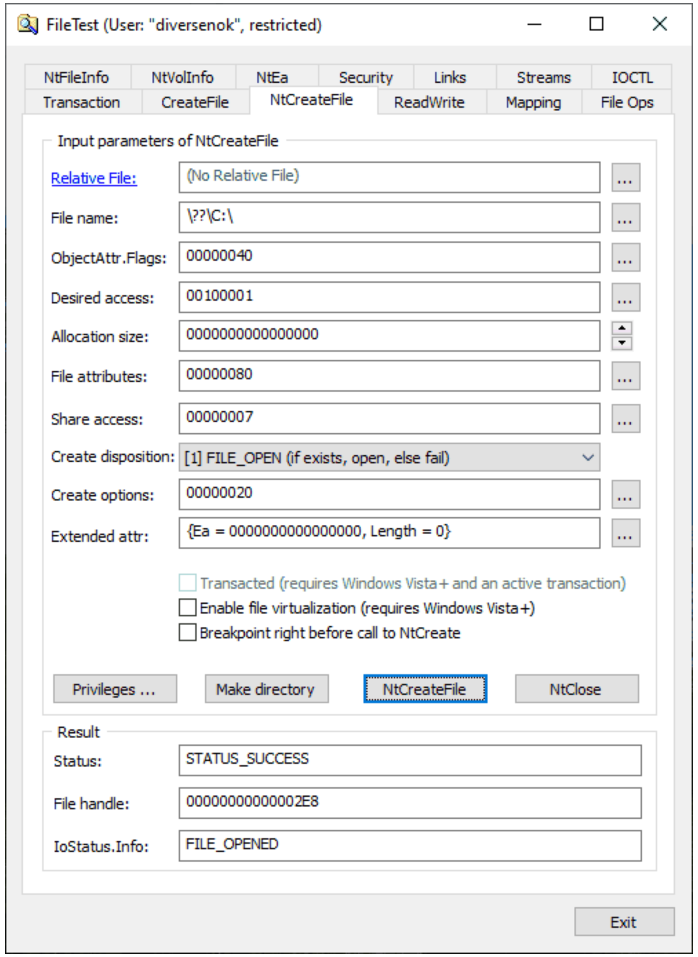
More possibilities than expected.

Tools:

- FileTest – covers 95% of what you need
- Sysmon event ID 7 “Image loaded” for experiments

Thanks:

Johnny Shaw – in-depth dives into mitigations & driver hardening



FileTest (User: "diversenok", restricted)

Input parameters of NtCreateFile

Relative File: (No Relative File)

File name: \??\C:\

ObjectAttr.Flags: 00000040

Desired access: 00100001

Allocation size: 0000000000000000

File attributes: 00000080

Share access: 00000007

Create disposition: [1] FILE_OPEN (if exists, open, else fail)

Create options: 00000020

Extended attr: {Ea = 0000000000000000, Length = 0}

□ Transacted (requires Windows Vista+ and an active transaction)

□ Enable file virtualization (requires Windows Vista+)

□ Breakpoint right before call to NtCreate

Result

Status: STATUS_SUCCESS

File handle: 00000000000002E8

IoStatus.Info: FILE_OPENED

MORE CONTENT

My blog post: Bypassing FileBlockExecutable in Sysmon 14: A Lesson In Analyzing Assumptions
https://www.huntandhackett.com/blog/bypassing-sysmon

Gabriel Landau's blog post: Introducing a New Vulnerability Class: False File Immutability
https://www.elastic.co/security-labs/false-file-immutability

Johnny Shaw's blog post: Process Herpaderping
https://jxy-s.github.io/herpaderping/

James Forshaw's blog post: Windows Exploitation Tricks: Trapping Virtual Memory Access
https://googleprojectzero.blogspot.com/2021/01/windows-exploitation-tricks-trapping.html

Gergely Kalman's talk: The forgotten art of filesystem magic.
https://gergelykalman.com/slides/the_forgotten_art_of_filesystem_magic.pdf

Gergely Kalman's talk: The missing guide to the security of filesystems and file APIs
https://gergelykalman.com/slides/the_missing_guide_to_filesystem_security_v1.pdf

Thank you for your attention!

Attacking Assumptions Behind the Image Load Callbacks :: Denis Nagayuk (diversenok)

ROMHACK 2025

@diversenok

Want me to look at your security product?
Send a message to denis.nagayuk@huntandhackett.com