State of Win32k Security

Revisiting Insecure Design

Vishal Chauhan (@axsdnied)
Microsoft Security Response Center (MSRC) Engineering Lead
DerbyCon 8.0

Agenda

Setting the stage

- Typical window's exploit
- Why Win32k?
- The design and redesign …

Revisiting insecure design

What's next?

Typical Exploit (via Edge)

- ✓ Multiple Vulnerabilities
- ✓ Multiple Components

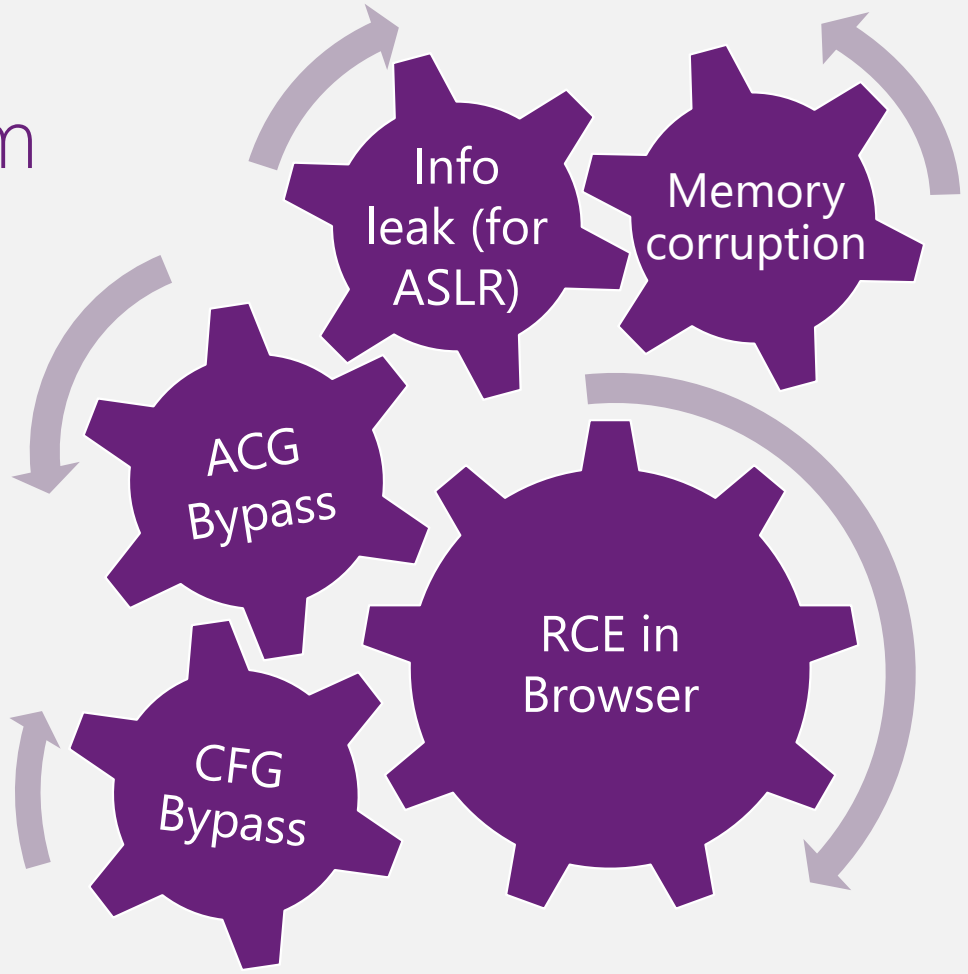
Typical Exploit: Getting on the system

Stage 1 Goal:

Get code execution on the system

Need multiple vulnerabilities

Bypass mitigations



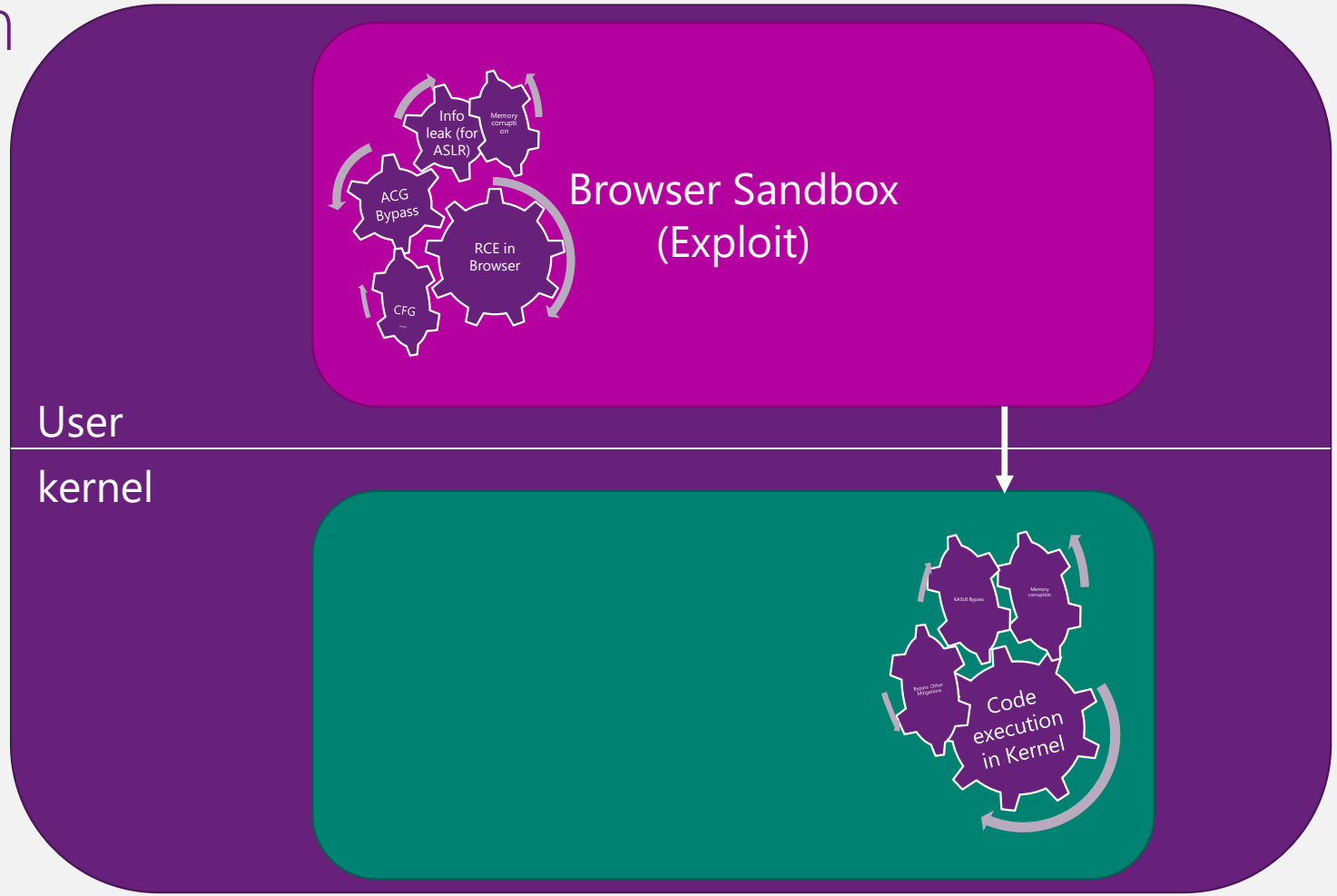
Typical Exploit: Getting on the system

Overall Goal:

Completely own the System

Need Sandbox escape

Bypass additional mitigations



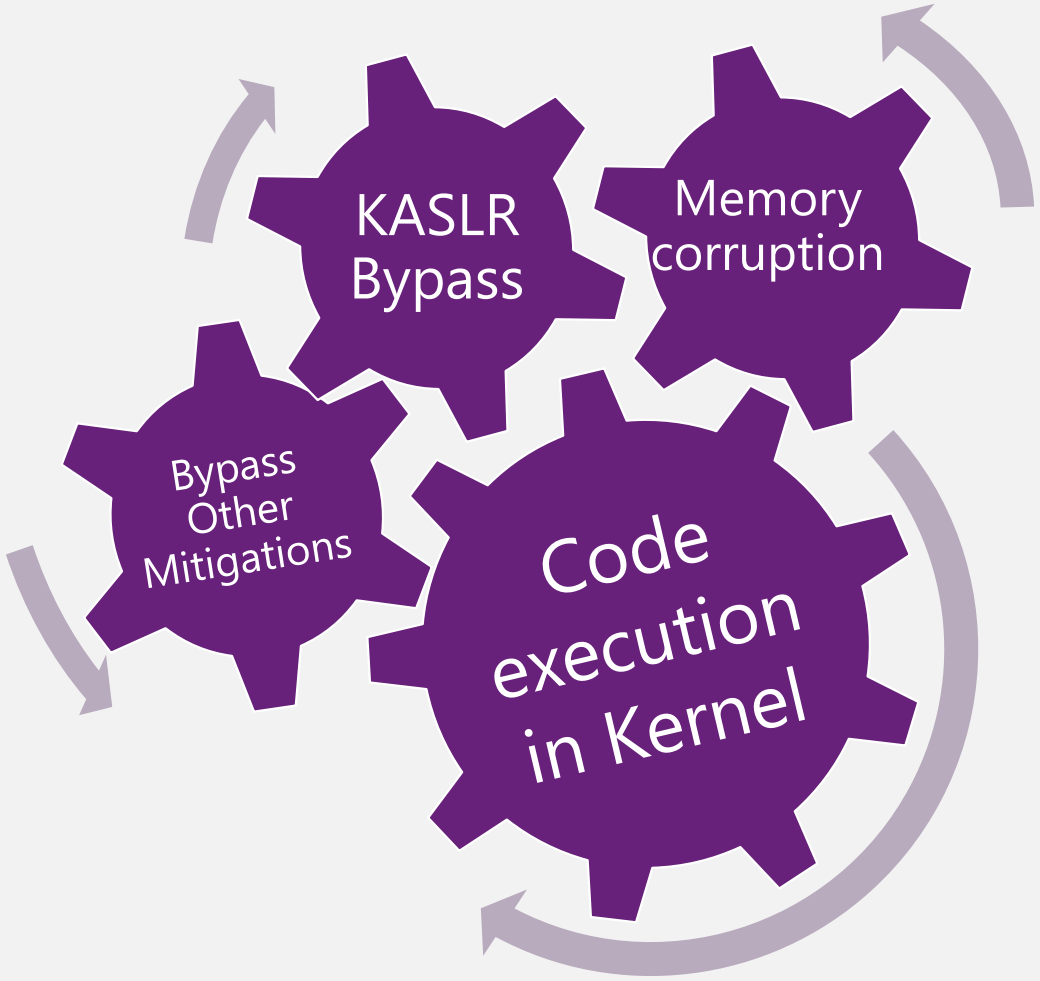
Typical Exploit: Getting on the system

Stage 2 Goal:

Get into kernel

Need multiple vulnerabilities

Bypass mitigations



Where does Win32k.sys fit in?

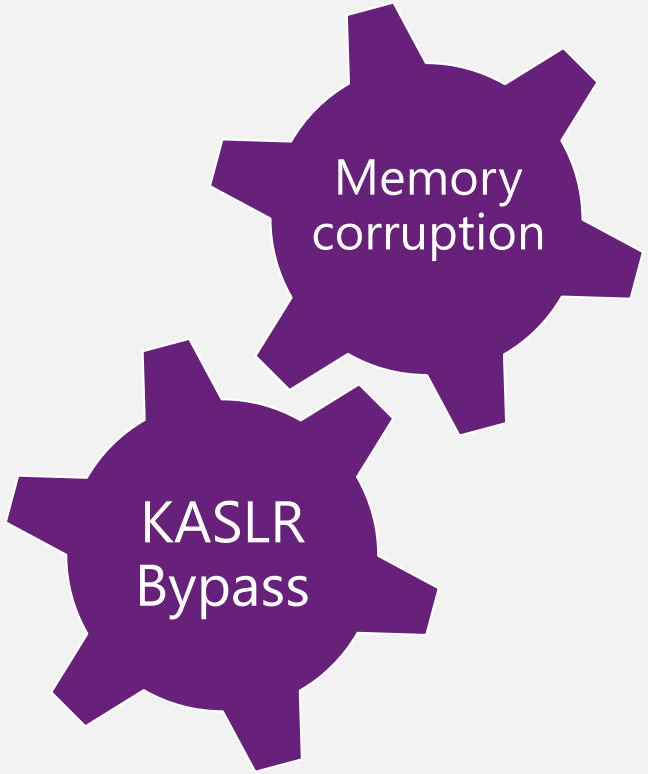
A typical sandbox escape could require

Identifying a memory corruption, like UAF, in win32k code base

Use it to craft a read/write primitive

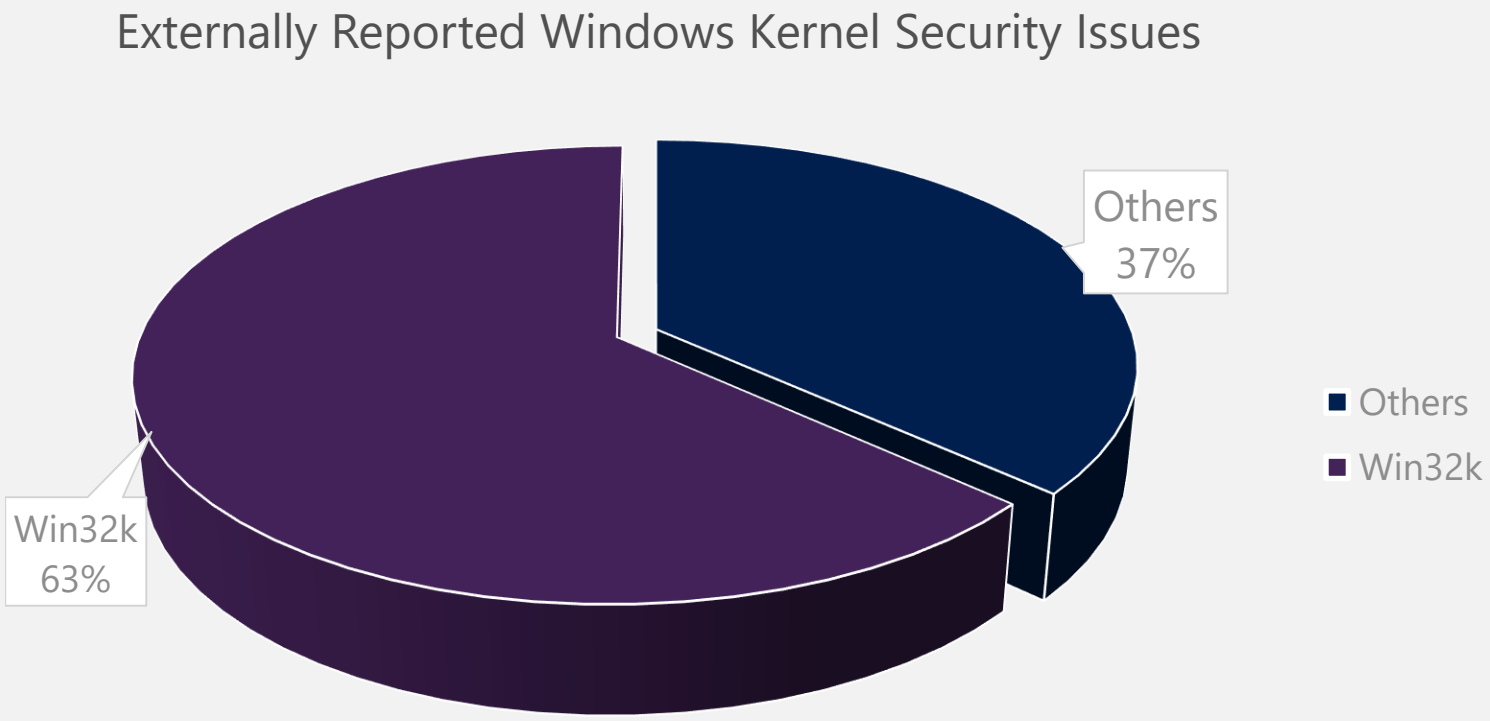
Using by-design leaks in win32k

Bypass KASLR using these leaks



Why Win32k.sys?

Since 2010, >50% of all kernel security bug in windows, are just in win32k.sys



But wait, what is Win32k?

Win32k is a Window's kernel mode driver

Part of its purpose is to host:

GDI: drawing library for graphics output devices
USER: handles input and UI elements

Demo

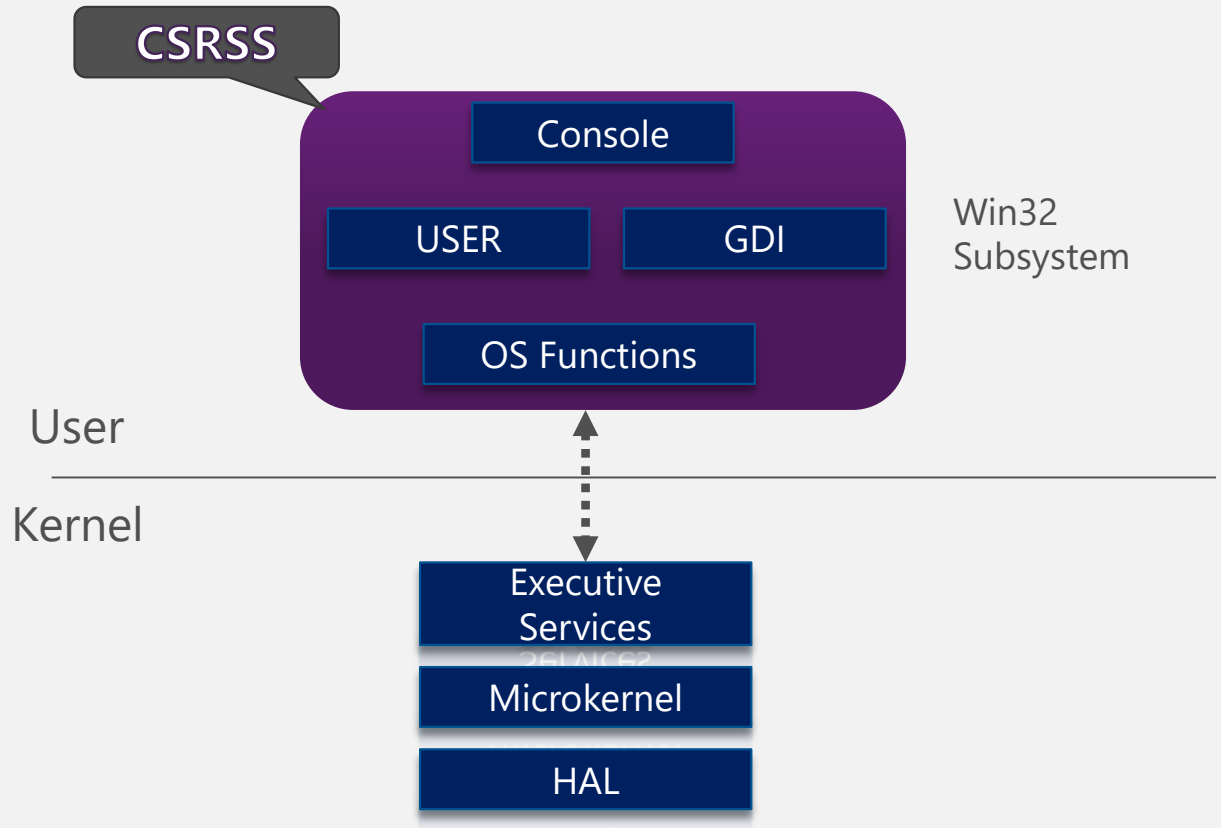
The design and redesign

It's old, *really* old.

> *"10-22-90 MikeHar Ported functions from Win 3.0 sources."*
> —from a file in Win32k

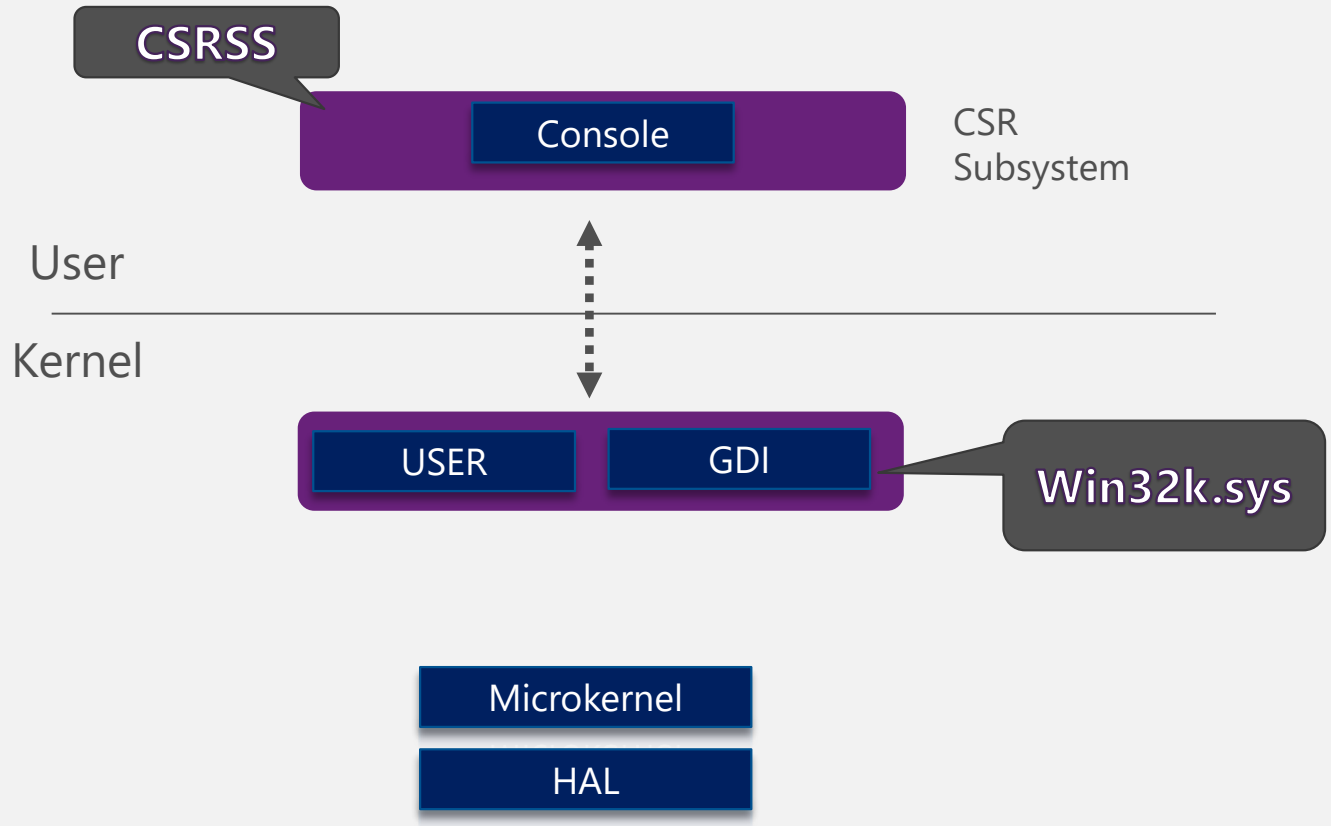
Design

Most of Win32k was in User land



Redesign

After the redesign(NT4), it was moved to Kernel land



Redesign: Why, you ask?

Eliminated the need for shared buffers and paired threads

Results in fewer thread and context switches

Reduces memory requirements

Impact

Great performance especially for graphics

But …

Redesign: What about security?

- ❑ New syscalls
- ❑ User mode Callbacks
- ❑ Shared data between User and Kernel
- ❑ …

Impact

Security took a hit in favor of performance

Revisiting Insecure Design

- ✓ New syscalls
- ✓ User mode Callbacks
- ✓ Shared data between User and Kernel

Revisiting Insecure Design: Syscalls

1100+ syscalls in Win32k Vs 480+ syscalls in NTOS

Wide attack surface

Writing secure Syscall?

Probing
Input validation
Exception handling
Locks

Revisiting Insecure Design: Syscall Filtering

Solution?

No Win32k policy

Now what?

- ✓ Applications do not needs all syscalls
- ✓ Linux has seccomp

Edge Filters out >75% of all Win32k syscalls
Multiple other system components use this filtering
Not available for 3rd party yet

Revisiting Insecure Design: Syscall Filtering

Impact

Reduced attack surface

Cascade effect on dependent syscalls for exploits

Like syscalls used for pool spray

Does it kill all exploits?

Nop, but it does reduce attack surface &
potentially increase exploitation cost

Revisiting Insecure Design: User mode callbacks

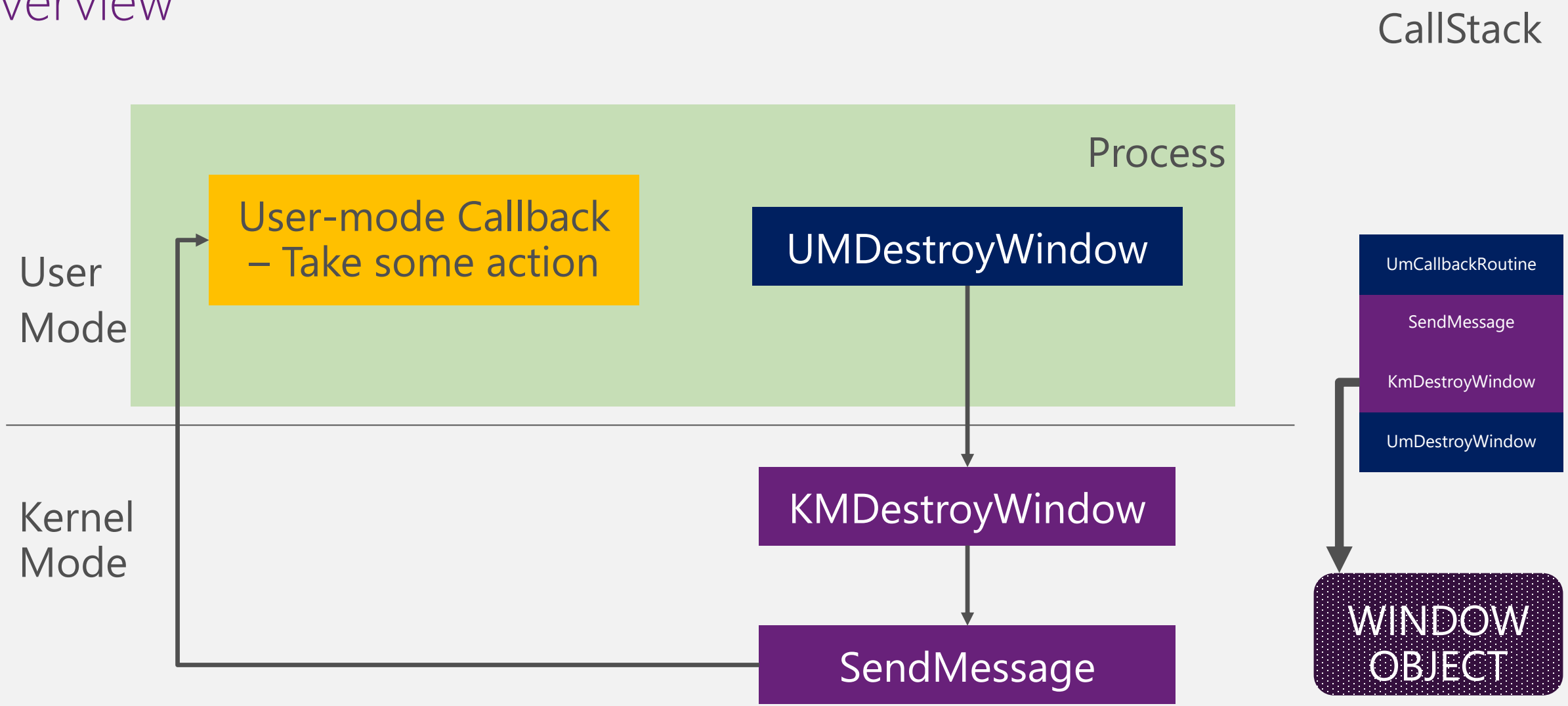
Unique feature of Win32k

Side effect of design-redesign

So what is it?

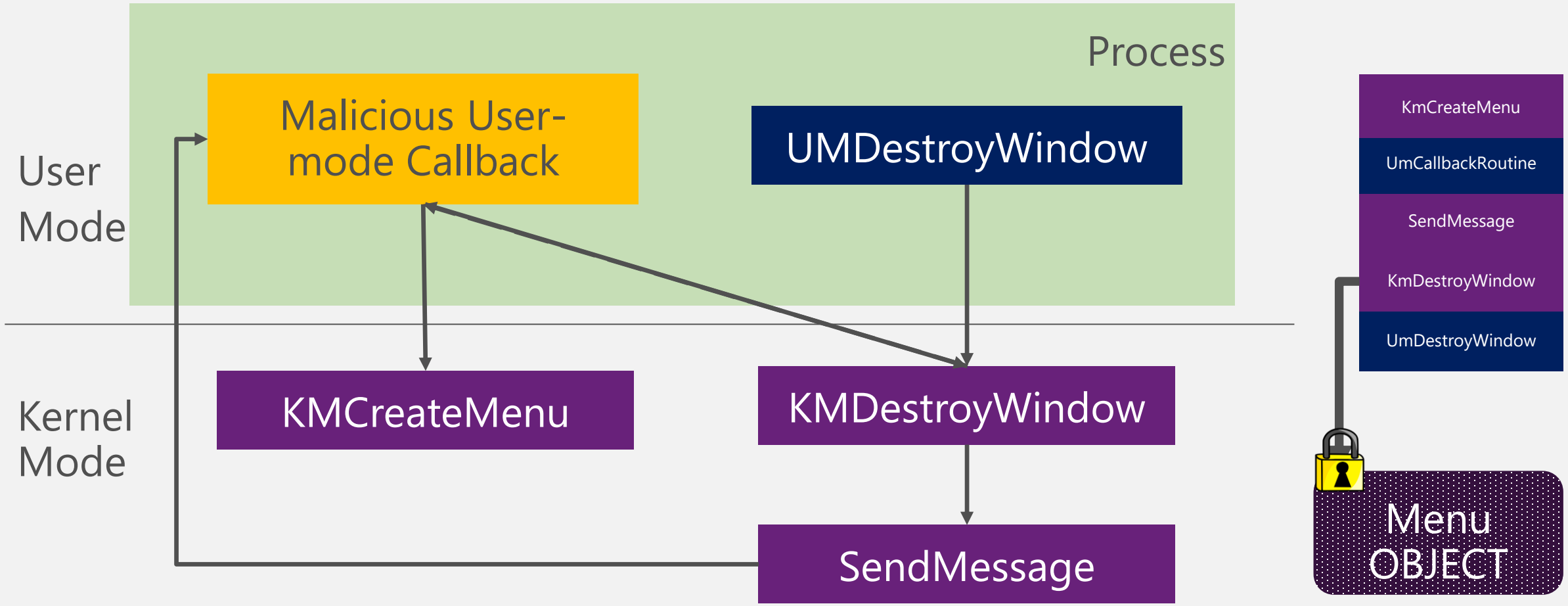
Revisiting Insecure Design: User mode callbacks

Overview



Revisiting Insecure Design: User mode callbacks

Typical Vulnerability

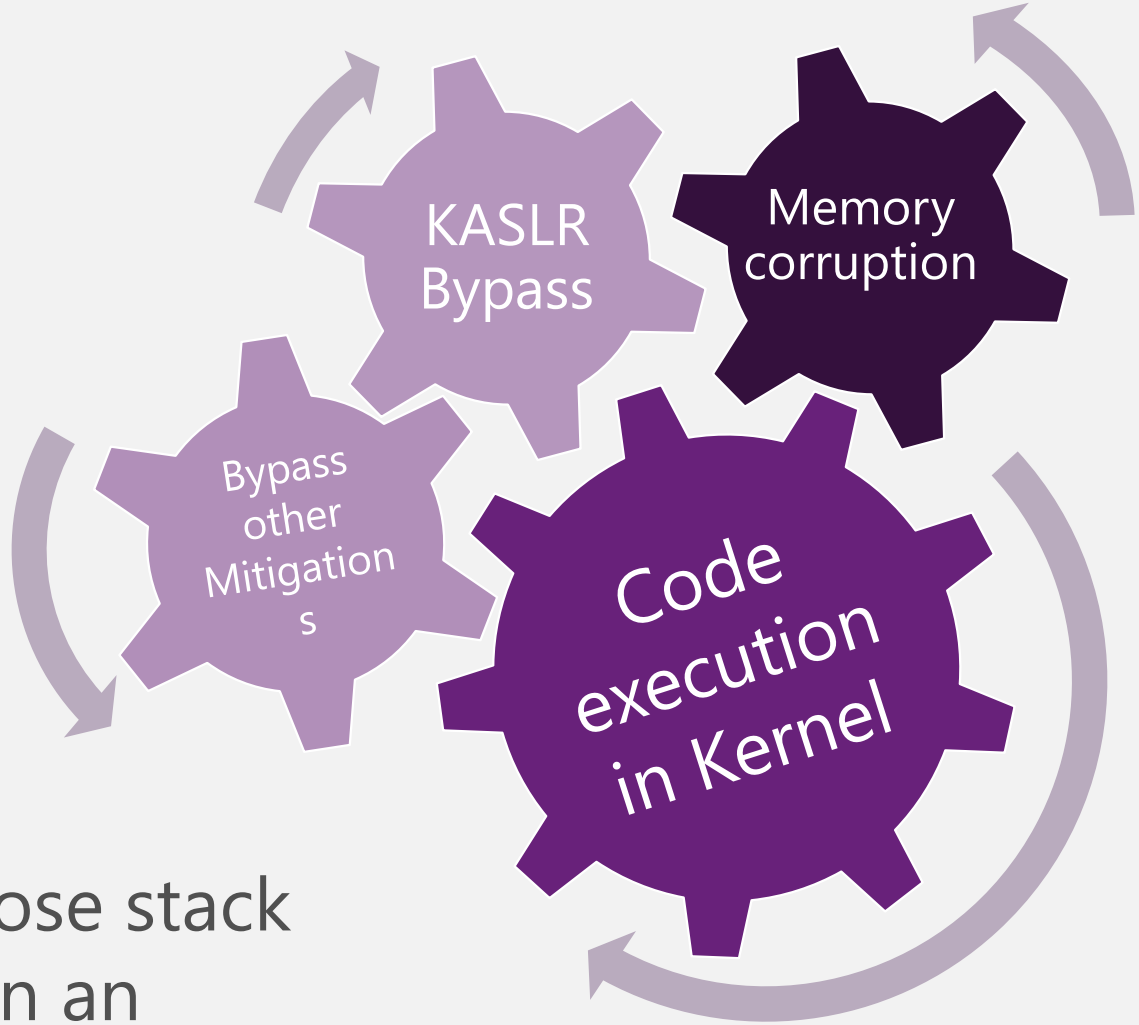


Revisiting Insecure Design: User mode callbacks

Win32k is a stateless system

So anytime you go out of context, it need revalidation OR make sure before hand that state will remain sane

Most of UAF bugs in USER are caused by dangling stack references.

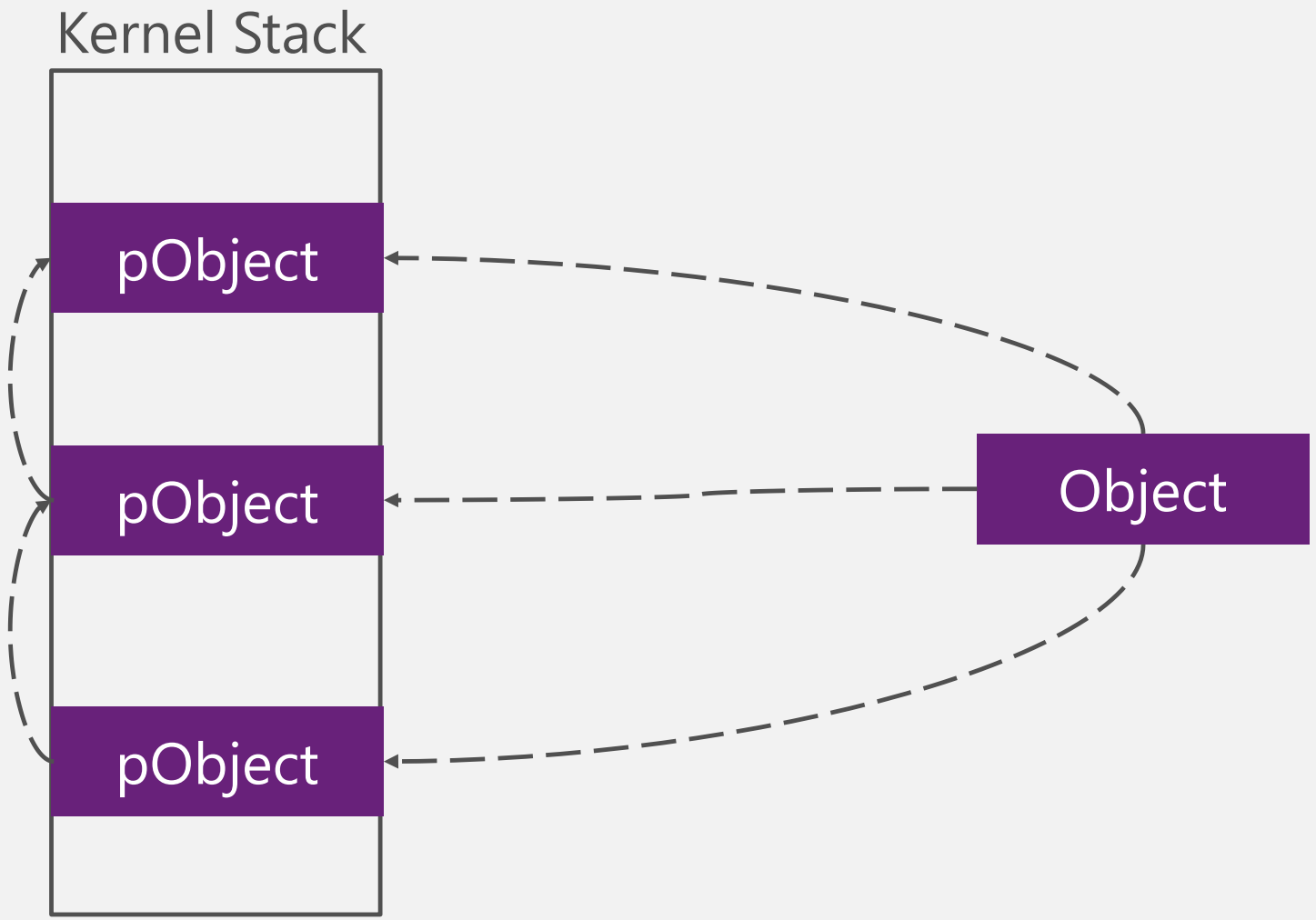


Solution

Stack Reference Tracker (SRT) tracks those stack references and sets them to NULL when an object is freed

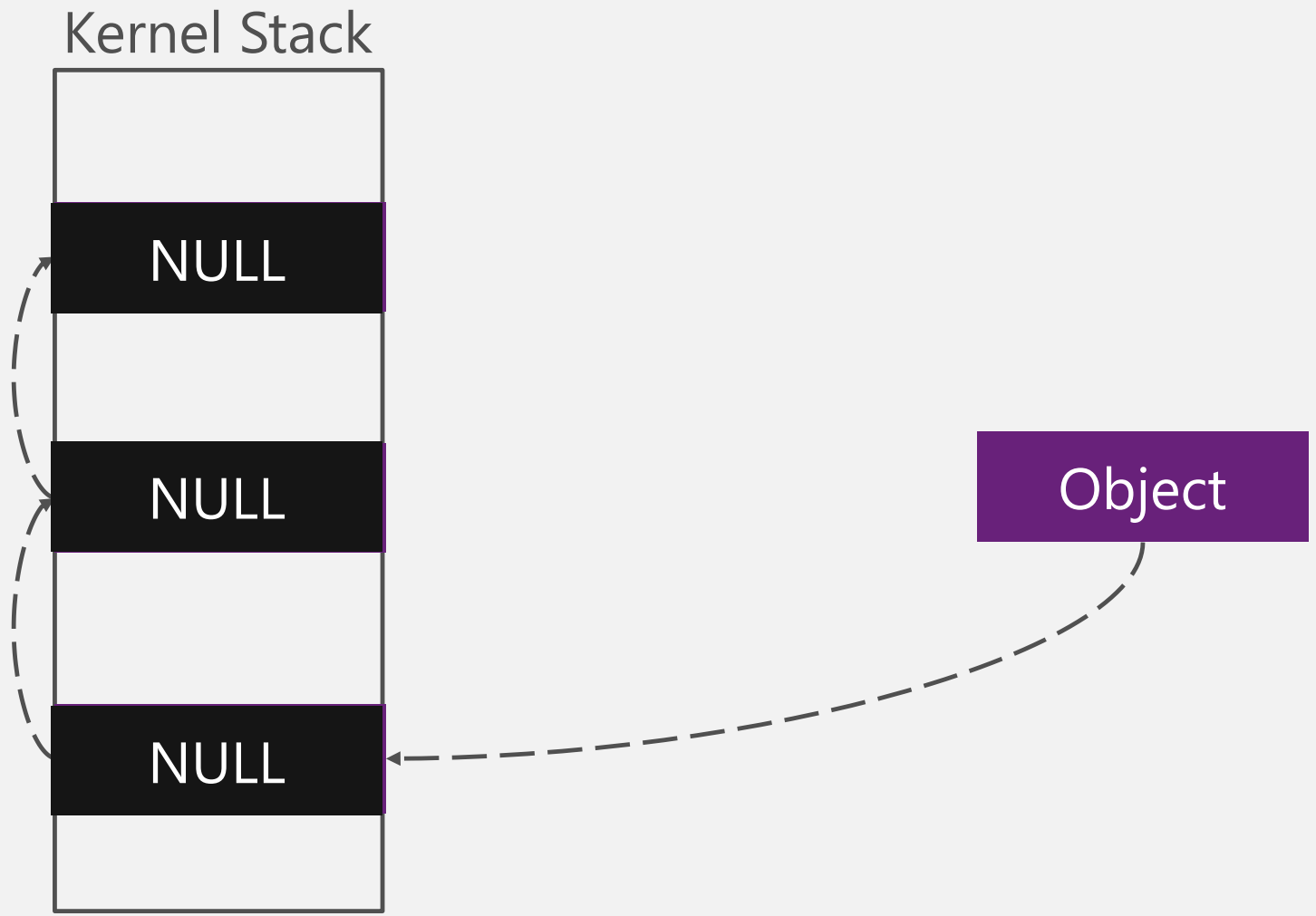
Revisiting Insecure Design: User mode callbacks

Stack reference tracking



Revisiting Insecure Design: User mode callbacks

Stack Reference nullification on Object Destruction



Revisiting Insecure Design: User mode callbacks

USER objects with history of MSRC cases are getting enlightened

Menu

PopUp Menu

Windows Class

+More objects on opportunity

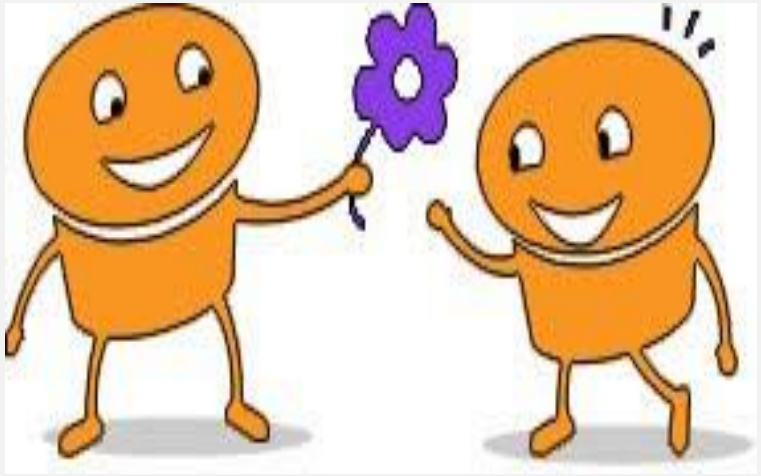
Impact

For objects that are SRT enlightened, user mode callback based UAF becomes unexploitable.

Revisiting Insecure Design: Shared Data

Sharing data is actually fairly common between kernel and user mode

But Win32k has some unique requirements



Handle tables

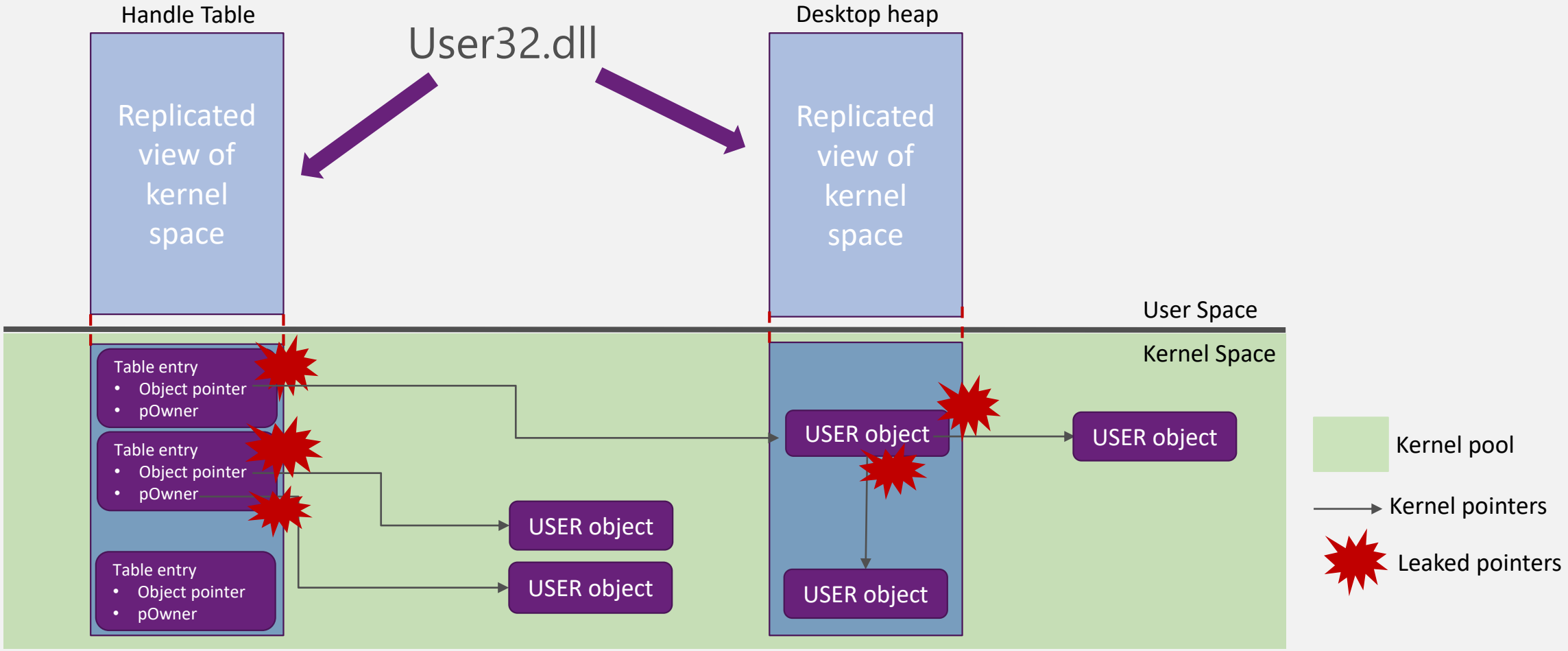
List of referenced to USER/GDI objects

Desktop heap

A subset of USER objects are shared with user

Revisiting Insecure Design: Shared Data

Handle table – list of USER objects
Desktop heap – host the USER objects



Revisiting Insecure Design: Shared Data

But Why?

For Performance

Save kernel context switches

Asynchronous access

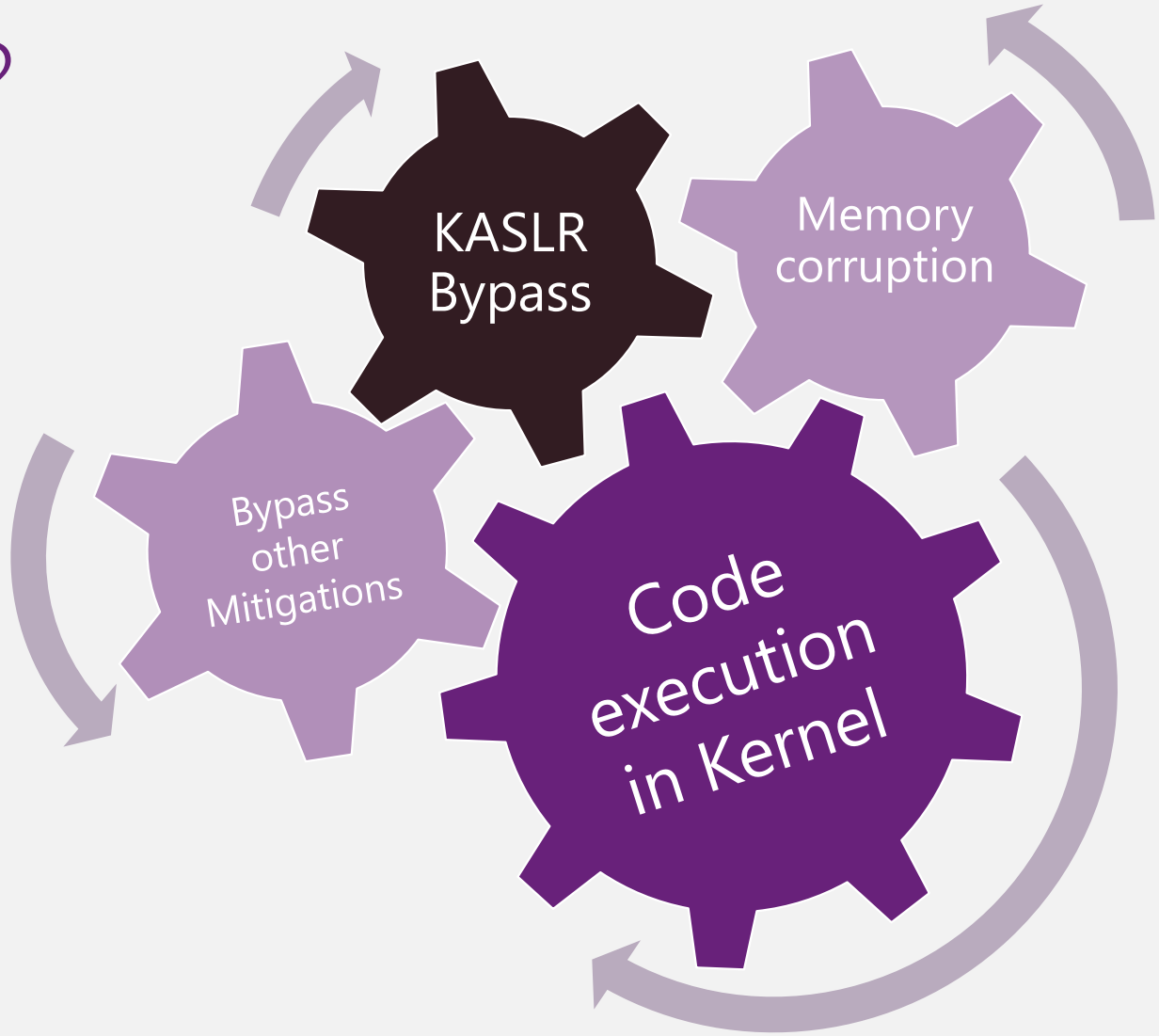
Optimizations for specific operations

Revisiting Insecure Design: Shared Data

Why does it matter?

KASLR Bypass

Major cog in the Exploit Machinery



Revisiting Insecure Design: Shared Data

How do we fix?

Remove any kind of sharing and move all the code to kernel mode. Easy… Right?

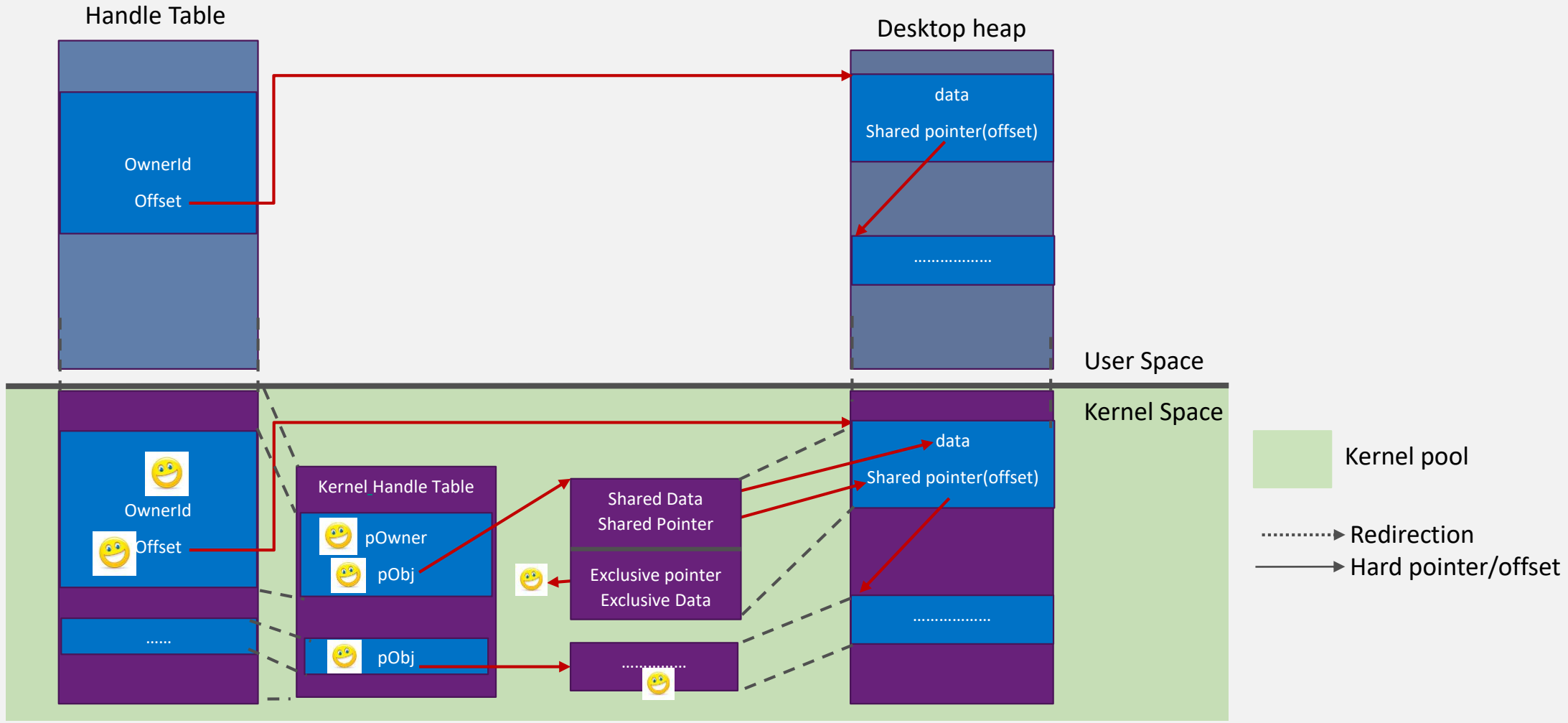
Perf hit

Regression prone

May result in more security bugs

Revisiting Insecure Design: Shared Data

Fix Implemented



What does the new design achieve?

Functionally

Minimal perf hit

No app compatibility issues

Security

No kernel pointer leaks

No leaks of location of kernel views of shared sections

Minimal increase of kernel code footprint

But Wait? What about heap Metadata?

What's Next?

Redesigning complex Software is hard

But security does drive things forward

More mitigation were added to Win32k in RS4 release

GDI objects isolation

No known exploits so far targeting Win32k in RS4

Finger crossed 

What's Really Next?

DirectX is getting more love

Need to tackle that as we move along

References

Win32k Internals

http://pasotech.altervista.org/windows_internals/Win32KSYS.pdf

Win32k Modern Exploits

https://www.blackhat.com/docs/us-17/wednesday/us-17-Schenk-Taking-Windows-10-Kernel-Exploitation-To-The-Next-Level%E2%80%93Leveraging-Write-What-Where-Vulnerabilities-In-Creators-Update.pdf

Desktop Heap

https://blogs.msdn.microsoft.com/ntdebugging/2007/01/04/desktop-heap-overview/

Usermode Callbacks and there exploits

https://media.blackhat.com/bh-us-11/Mandt/BH_US_11_Mandt_win32k_WP.pdf

Win32k Syscall Filtering

https://improsec.com/blog/win32k-system-call-filtering-deep-dive

Windows 10 Segment Heap Internals

https://www.blackhat.com/docs/us-16/materials/us-16-Yason-Windows-10-Segment-Heap-Internals-wp.pdf