Windows Driver Security

The Do’s, the Don’t’s and the “oh please don’t”s

Yarden Shafir
CrowdStrike

About Me

- Software Engineer at CrowdStrike
- Previously a Security Researcher at SentinelOne
- Circus Artist – Aerial Arts Performer & Instructor
- Windows Internals Instructor
- Former Pastry Chef
- Blogging about Windows Security stuff
    - Windows-internals.com
- Twitter: @yarden_shafir

Processor Ring Levels

- Processors allow 4 ring levels (CPLs)
  - Ring 0 is the most privileged, 3 is the least
  - Only 2 levels are used: 0 and 3
  - Kernel code runs at ring 0, User code at ring 3
- Ring 0 code can do anything
  - Essentially “owns” a machine
  - Achieving ring 0 code execution is a goal for attackers

Windows Memory Model – x64

- Address ranges:
  - User memory range is 0x0 – 0x7fffffffffff
  - Kernel memory range is 0xffff800000000000 – 0xffffffffffffffff
  - Everything in the middle is invalid memory (for now)
- User-Mode (ring 3) code can't access kernel address space
  - But ring 0 code can access all address space
- User-mode processes are separated from each other
  - Each process has its own separate address space and cannot access other processes
- Kernel code mostly shares the same address space

What’s Running In The Kernel?

- The Windows kernel – ntoskrnl.exe
  - Manages the whole system
- Other Windows drivers
  - Managing graphics, file system, networking…
- 3rd party drivers
  - Hardware drivers
  - Wi-Fi drivers
  - Security products
  - Gaming software
  - And much more…

User-Kernel Communication

- Drivers can create devices for user-mode code to talk to
    - Happens through requests called IOCTLs
    - Every driver can implement its own requests
    - When a process issues an IOCTL request it sends:
        - IOCTL code – identifies the request
        - Input buffer + length
        - Output buffer + length
- Every process can talk to any device – unless specified otherwise
    - There are default security settings set by the system
    - Driver creating the device can overwrite these with its own security settings

Security Descriptors

- Describe the security of an object
  - Who can read, write, issue IOCTLs…
    - For example: can limit who can read and write to a file
    - Devices can (and should) have a security descriptor too
- A device without an SD will receive requests from any process
  - Can cause security issues if driver is not aware of this
  - Not a lot of cases where this is intentional – usually caused by bad programming
- Two ways to create a security descriptor for a device:
  - INF file with security descriptor specs
    - INF file can specify security descriptor for device, service access, registry or file access
  - IoCreateDeviceSecure

Avoiding Common Vulnerability Classes

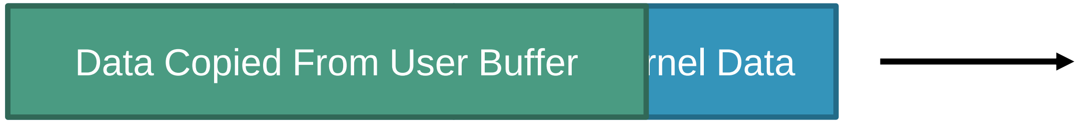
- User-mode callers can’t be trusted – can always be malicious
- It’s important to validate all input from user-mode
- Also check what data you return to the user-mode caller
- User-mode buffers can be changed by malicious code while the kernel is reading them
- Notice what actions you’re doing on behalf of a user-mode caller – and which caller that is

What are You Reading? Where are You Writing?

- Both input buffer and output buffer should be user-mode addresses
- If InputBuffer is a kernel buffer -> driver will read arbitrary data
- If OutputBuffer is a kernel buffer -> driver will write to arbitrary address
- Checking base address is not enough – check size and end address too
- Use ProbeForRead and ProbeForWrite to validate user mode addresses
    - If the buffer contains more pointers – validate those too!

Validate Sizes and Avoid Overflows

- What can happen if you copy all the data from a user buffer to a kernel buffer without checking the size?

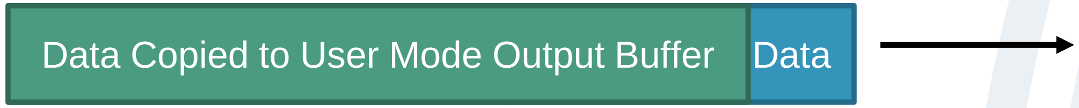


Data Copied From User Buffer

rnel Data

Kernel Data Corruption

- Or copy kernel data back to a user buffer without checking the size of the data?



Data Copied to User Mode Output Buffer

Data

Kernel Information Leak

Only Read User-Mode Memory Once

- Imagine this scenario:
  - Driver reads a size from an input buffer
  - Validates that size is correct
  - Malicious caller changes size to bad value
  - Driver reads size from again and uses it to copy data – overflow!
- This is called double-fetch or Time of Check vs. Time of Use bug
- Copy user-mode data to local kernel buffer to avoid bugs

Summary

- User-Kernel communication can be dangerous
- Secure your devices – who should they ne interacting with? How?
- Don’t trust the user – user-mode code can always be malicious
  - Validate addresses
  - Validate sizes
- User can change data behind the driver’s back

Questions?