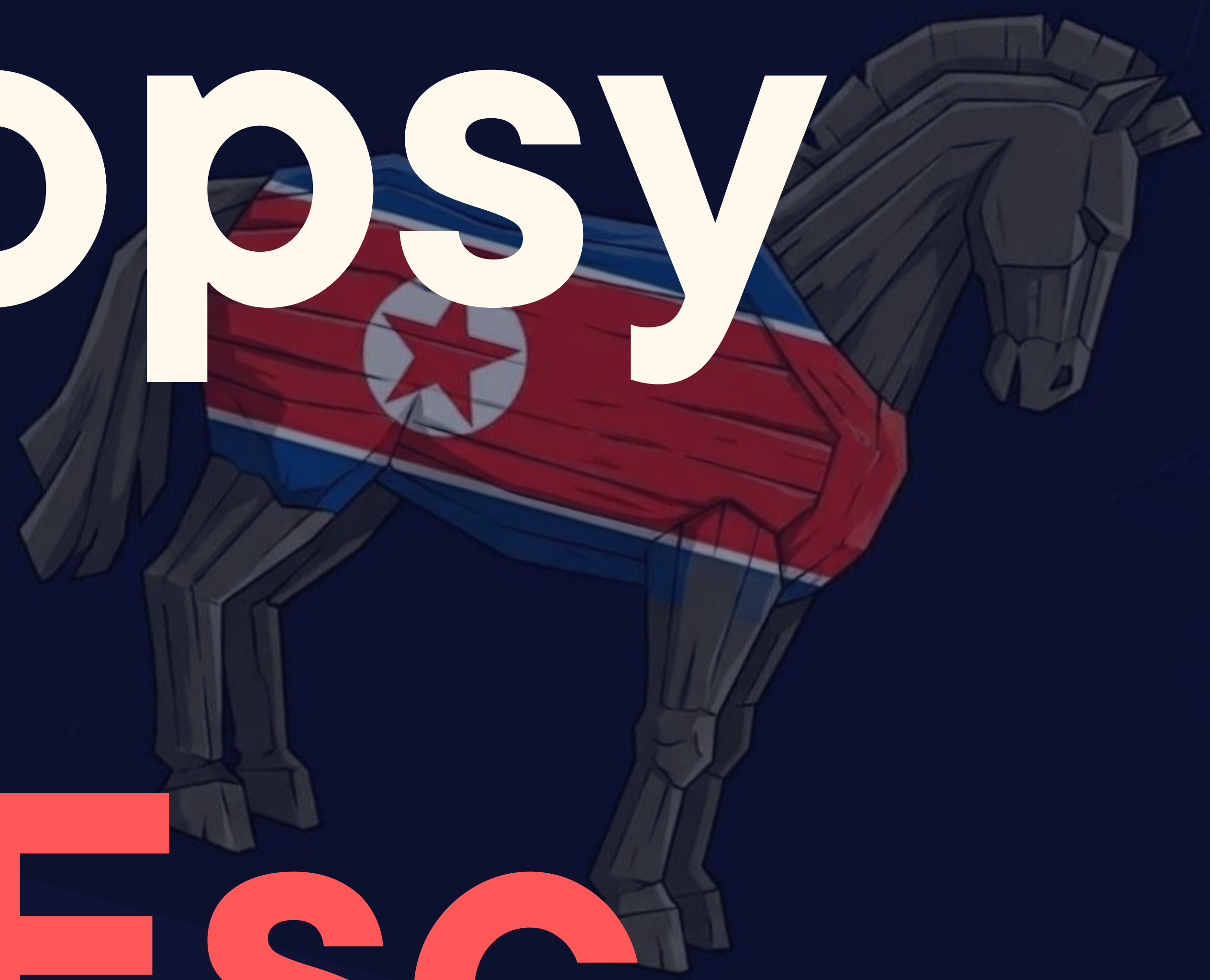
#! Anatomy of a Bug

Autopsy of a PrivEsc



The Lazarus 0-Day

The Patient

- **Name:** Windows AppLocker (appid.sys)
- **CVE:** CVE-2024-21338
- **Diagnosis:** Admin-to-Kernel (LPE)
- **Vector:** Logic Flaw in Input Processing
- **Severity:** 🔥 7.8

What happened?

- **Campaign:** Lazarus deployed a Rootkit.
- **Technique:** Direct Kernel Object Manipulation (DKOM).
- **Action:** Unlinking callbacks to blind EDR.
- **Result:** Undetected persistence in Ring 0.

#1 The Context

Beyond BYOVD

#1 The Context

Beyond BYOVD

👉 The "Loud" Way (BYOVD): Attackers usually bring old, vulnerable drivers to hack the kernel. "Bring Your Own Vulnerable Driver".

👉 The "Silent" Way (LOTL): Lazarus exploited appid.sys, which is already running by default. True "Living off the land".

👉 Advantage: Zero noise compared to BYOVD—no driver drop, no load event.

👉 The Goal: Full Kernel Access (Ring 0) to blind EDR/AV solutions like CrowdStrike & Windows Defender.

#2 The Vulnerability

Blind Trust

#2 The Vulnerability

Blind Trust

👉 **The Core Flaw:** The driver reads a value from the **user-input buffer** and treats it as a **function pointer to be executed.**

👉 **The Danger:** It performs an **indirect call** to this address **without verifying the caller's origin.**

👉 **The Implication:** It **implicitly trusts** that the provided pointer is safe, **failing to check** if the request came from an **untrusted User-Mode source.**

👉 **Result:** **Arbitrary Indirect Call in Kernel Mode.**

#3 The Mindset

Access Control Lists (ACLs)

#3 The Mindset

Access Control Lists (ACLs)

👉 Attackers check: "Who is allowed to talk to this Device?"

👉 The Tool: Using WinObj (SysInternals), we inspect the Device Object's permissions (\Device\AppID).

👉 The Block: Administrators lack Write permission in the ACL and get STATUS_ACCESS_DENIED.

👉 The Key: The account LOCAL SERVICE has explicit Write permission.

#4 The Kill-Chain

Weaponizing Trust

#4.1 The Kill-Chain

Privilege Escalation: The Upgrade (Admin → SYSTEM)

👉 **Step 1:** As Admin, we possess SeDebugPrivilege. This allows us to access the memory of any program like winlogon.exe (SYSTEM).

👉 **Step 2:** We steal (duplicate) the SYSTEM token.

👉 **Why SYSTEM?** The SYSTEM token inherently holds SeAssignPrimaryTokenPrivilege.

👉 **Result:** We impersonate SYSTEM to gain the right to assign new identities.

#4.2 The Kill-Chain

Privilege Escalation: The Downgrade (SYSTEM → LOCAL SERVICE)

👉 Step 3: Using our stolen SYSTEM token, we duplicate the LOCAL SERVICE token from a running svchost.exe Process.

👉 Step 4: We spawn a new process and assign it the LOCAL SERVICE token.

👉 Result: We successfully open a handle to \Device\AppID. The door is open.

#4.3 The Kill-Chain

The Trigger & The Obstacle

👉 **The Vector:** We send the dispatch IOCTL 0x22A018 to the open handle.

👉 **The Execution:** The driver passes our input to AppHashComputeImageHashInternal, which executes the unchecked function pointer.

👉 **The Problem:** kCFG & SMEP (protection mechanisms).

👉 We cannot jump to User-Mode Shellcode (SMEP) nor to arbitrary Kernel addresses (kCFG).

👉 **Solution:** We need a valid "Gadget".

#4.4 The Kill-Chain

The Gadget: ExpProfileDelete()

👉 We choose ExpProfileDelete() because it is a valid CFG target (originally intended for Power Profile cleanup).

👉 The Mechanism: Internally, it calls ObfDereferenceObject() to decrease an object's reference count.

👉 The Abuse: We weaponize this logic to perform an arbitrary decrement on our target address: *Address = *Address - 1

#4.5 The Kill-Chain

Deep Dive: What is PreviousMode?

👉 **Definition:** A field in the Kernel-Internal _KTHREAD structure acting as the Trust Flag for System Calls.

👉 **User Mode (PreviousMode = 1):** Untrusted. The Kernel validates every memory pointer via ProbeForRead/Write.

👉 **Kernel Mode (PreviousMode = 0):** Trusted. The Kernel assumes the caller is safe. No checks performed.

👉 **The Goal:** If we flip this byte to 0, we gain Kernel Privileges.

#4.6 The Kill-Chain

The Corruption (Kernel Mode)

👉 **The Setup:** We point the **ExpProfileDelete()** gadget to the address of **our own PreviousMode field.**

👉 **The Math:**

- 👉 **Current Value: 0x01 (User).**
- 👉 **Gadget Action:** Decrement.
- 👉 **Result: 0x00 (Kernel).**

👉 **The Aftermath:** The Kernel now **treats our thread as trusted (Ring 0).**

👉 Syscalls **bypass security constraints,** granting us **unrestricted arbitrary Read/Write access to the entire system memory.**

#4.7 The Kill-Chain

The Big Picture: Full Execution Flow

👉 1. Access (the dance):

(Goal: Gain Write-Access to \Device\AppID)

👤 Admin Steal →

⚙️ SYSTEM Assign →

🛡️ LOCAL SERVICE

👉 2. Trigger (the bug):

(Goal: Execute the Gadget via the Logic Flaw)

📂 Open Handle IOCTL 0x22A018 →

💥 Unchecked Callback

👉 3. Exploit (the corruption):

(Goal: Decrement PreviousMode to 0)

🔧 Gadget (ExpProfileDelete()) -1 →

🎯 PreviousMode

👉 4. End Game

💀 Kernel Mode → Blind EDR

#5 Developer’s Takeaway

#5 Developer's Takeaway

Never Trust the Caller

👉 **Implicit Trust is Deadly:** The driver assumed the code execution was safe. It wasn't.

👉 **Validate Context:** When exposing sensitive functions, always verify the execution context.

👉 **Gadgets are everywhere:** Even with protections like kCFG, legitimate code (ExpProfileDelete()) can still be weaponized if logic flaws exist.

#6 The Fix.

#6 The Fix.

Microsoft added a check to ensure the IOCTL is not called from User Mode.

```
// Pseudo-code based on binary diffing
NTSTATUS AipSmartHashImageFile(...) {

    // 🛡️ THE FIX: Verify caller mode
+   if (ExGetPreviousMode() != KernelMode) {
+       return STATUS_ACCESS_DENIED;
+   }

    // "Vulnerable"/trust logic continues only if KernelMode...
    AppHashComputeImageHashInternal(...);
}
```

Status

Fixed by Microsoft (Feb 2024).
Monitor your IOCTLs.
Validate Execution Context.

#! Anatomy of a Bug

Technical Credits:

Jan Vojtěšek, Avast & Lazarus Group

Author: @tralsesec

#Windows #Lazarus #ExploitDev #CVE-2024-21338 #LPE #Kernel #PrivilegeEscalation