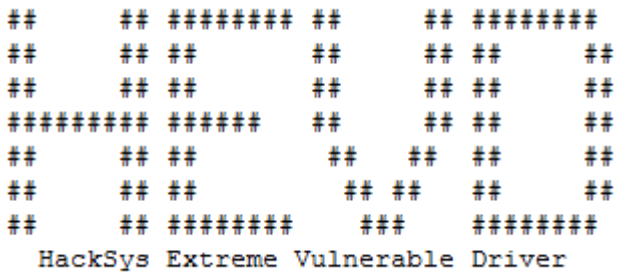
HackSys Extreme Vulnerable Driver



Ashfaq Ansari
ashfaq@payatu.com
PAYATU TECHNOLOGIES PVT. LTD.

Table of Content

Introduction

HackSys Extreme Vulnerable Driver is intentionally vulnerable Windows Kernel driver developed for security enthusiasts to learn and polish their exploitation skills at Kernel level.

HackSys Extreme Vulnerable Driver caters wide range of vulnerabilities ranging from simple **Buffer Overflow** to complex **Use After Free** and **Pool Overflow**. This allows the researchers to explore the different exploitation techniques for every implemented vulnerabilities.

Previous Work

Damn Vulnerable Windows Driver

KdExploitMe (https://github.com/clymb3r/KdExploitMe/)

Why HackSys Extreme Vulnerable Driver?

I was giving a series of talks on **Windows Kernel Exploitation** at **null Pune Chapter**. So, I thought, it's better to write a driver which has all the major vulnerabilities implemented in it. The idea to write the driver was to provide the attendees a better view of **what's happening behind the vulnerable code** and also this will be of great help during my workshops and trainings.

This driver also implements the mitigations for the each implemented vulnerabilities if compiled with SECURE flag. This allows the developers to understand how these vulnerabilities can be mitigated easily.

Vulnerabilities Implemented

- Pool Overflow
- Use After Free
- Type Confusion
- Stack Overflow
- Integer Overflow
- Stack Overflow GS
- Null Pointer Dereference
- Arbitrary Memory Overwrite

To Do

- Memory Disclosure
- Use of Uninitialized Variable
- Time Of Check To Time Of Use (Race Condition)

Pool Overflow

```
// Code snipped for brevity

NTSTATUS TriggerPoolOverflow(IN PVOID pUserModeBuffer, IN SIZE_T
userModeBufferSize) {
    PVOID pKernelBuffer = NULL;
    NTSTATUS status = STATUS_SUCCESS;
    PAGED_CODE();

    __try {
        pKernelBuffer = ExAllocatePoolWithTag(NonPagedPool,
                                              (SIZE_T)POOL_BUFFER_SIZE,
                                              (ULONG)POOL_TAG);

        if (!pKernelBuffer) {
            status = STATUS_NO_MEMORY;
            return status;
        }

        ProbeForRead(pUserModeBuffer,
                     (SIZE_T)POOL_BUFFER_SIZE,
                     (ULONG)__alignof(UCHAR));


        #ifdef SECURE
            RtlCopyMemory(pKernelBuffer, pUserModeBuffer, (SIZE_T)BUFFER_SIZE);

        #else
            DbgPrint("[+] Triggering Pool Overflow\n");
            RtlCopyMemory(pKernelBuffer, pUserModeBuffer, userModeBufferSize);

        #endif

        if (pKernelBuffer) {
            ExFreePoolWithTag(pKernelBuffer, (ULONG)POOL_TAG);
            pKernelBuffer = NULL;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

PoolOverflow.c

```
[+] Starting Pool Overflow Exploitation
        [+] Creating The Exploit Thread
                [+] Exploit Thread Handle: 0x50
        [+] Setting Thread Priority
                [+] Priority Set To THREAD_PRIORITY_HIGHEST
        [+] Getting Device Driver Handle
                [+] Device Name: \\.\HackSysExtremeVulnerableDriver
                [+] Device Handle: 0x54
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For Buffer
                        [+] Memory Allocated: 0x00480D18
                        [+] Allocation Size: 0x220
                [+] Mapping Null Page
                        [+] Memory Allocated: 0x00000000
                        [+] Allocation Size: 0x2000
                [+] Preparing Buffer Memory Layout
                        [+] TypeIndex Of Event Object Set To: 0x0
                [+] Preparing OBJECT_TYPE At Null Page
                        [+] DeleteProcedure Value: 0x1132770
                        [+] DeleteProcedure Address: 0x00000060
                [+] EoP Payload: 0x01132770
                [+] Preparing NonPaged Kernel Pool Layout
                        [+] Spraying With Event Objects
                        [+] Creating Holes By Coalescing
        [+] Triggering Pool Overflow
                [+] Triggering Payload
                        [+] Freeing Event Objects
[+] Completed Pool Overflow Exploitation
[+] Checking Current Process Privileges
        [+] Trying To Get Process ID Of: csrss.exe
                [+] Process ID Of csrss.exe: 344
        [+] Trying To Open csrss.exe With PROCESS_ALL_ACCESS
                [+] Process Handle Of csrss.exe: 0xEB08
        [+] Successfully Elevated Current Process Privileges
[+] Enjoy As SYSTEM [0.000000]s
```

Use After Free

```
// Code snipped for brevity

VOID UaFObjectCallback() {
    PAGED_CODE();
    DbgPrint("[+] UseAfter Free Callback called\n");

}

NTSTATUS CreateUaFObject() {
    NTSTATUS status = STATUS_SUCCESS;
    PUSE_AFTER_FREE pUseAfterFree = NULL;
    PAGED_CODE();

    __try {
        DbgPrint("[+] Creating UaF Object\n");
        pUseAfterFree = (PUSE_AFTER_FREE)ExAllocatePoolWithTag(NonPagedPool,
                                                               sizeof(USE_AFTER_FREE),
                                                               (ULONG)POOL_TAG);
        if (!pUseAfterFree) {
            status = STATUS_NO_MEMORY;
            return status;
        }
        RtlFillMemory((PVOID)pUseAfterFree->buffer,
                      sizeof(pUseAfterFree->buffer),
                      0x41);
        pUseAfterFree->buffer[sizeof(pUseAfterFree->buffer) - 1] = '\0';
        pUseAfterFree->pCallback = &UaFObjectCallback;

        g_UseAfterFreeObject = pUseAfterFree;
```

```
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }
    return status;
}

NTSTATUS UseUaFObject() {
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PAGED_CODE();

    __try {
        if (g_UseAfterFreeObject) {

            if (g_UseAfterFreeObject->pCallback) {
                g_UseAfterFreeObject->pCallback();
            }
            status = STATUS_SUCCESS;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }
    return status;
}

NTSTATUS FreeUaFObject() {
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PAGED_CODE();

    __try {
        if (g_UseAfterFreeObject) {
            #ifdef SECURE
                ExFreePoolWithTag((PVOID)g_UseAfterFreeObject, (ULONG)POOL_TAG);
                g_UseAfterFreeObject = NULL;
            #else
                ExFreePoolWithTag((PVOID)g_UseAfterFreeObject, (ULONG)POOL_TAG);
            #endif

            status = STATUS_SUCCESS;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}

NTSTATUS CreateFakeObject(IN PFAKE_OBJECT pFakeObject) {
    NTSTATUS status = STATUS_SUCCESS;
    PFAKE_OBJECT pKernelFakeObject = NULL;

    PAGED_CODE();

    __try {
        pKernelFakeObject = (PFAKE_OBJECT)ExAllocatePoolWithTag(NonPagedPool,
                                                                sizeof(FAKE_OBJECT),
                                                                (ULONG)POOL_TAG);
        if (!pKernelFakeObject) {
            status = STATUS_NO_MEMORY;
            return status;
        }
```

```
        ProbeForRead((PVOID)pFakeObject,
                     sizeof(FAKE_OBJECT),
                     (ULONG)__alignof(FAKE_OBJECT));

        RtlCopyMemory((PVOID)pKernelFakeObject,
                      (PVOID)pFakeObject,
                      sizeof(FAKE_OBJECT));

        pKernelFakeObject->buffer[sizeof(pKernelFakeObject->buffer) - 1] = '\0';
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

UseAfterFree.c

```
[+] Starting Use After Free Exploitation
        [+] Creating The Exploit Thread
                [+] Exploit Thread Handle: 0x50
        [+] Setting Thread Priority
                [+] Priority Set To THREAD_PRIORITY_HIGHEST
        [+] Getting Device Driver Handle
                [+] Device Name: \\.\HackSysExtremeVulnerableDriver
                [+] Device Handle: 0x54
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For Buffer
                        [+] Memory Allocated: 0x0037FE40
                        [+] Allocation Size: 0x58
                [+] Preparing FAKE_OBJECT structure
                        [+] pFakeObject Value: 0x01132720
                        [+] pFakeObject Address: 0x0037FE40
                        [+] FAKE_OBJECT Size: 0x58
                        [+] EoP Payload: 0x01132720
                [+] Preparing NonPaged Kernel Pool Layout
                        [+] Spraying With Reserve Objects
                        [+] Creating Holes
                [+] Working With Vulnerable UaF Object In NonPaged Pool
                        [+] Allocating UaF Object
                        [+] Freeing UaF Object
                        [+] Filling Freed Chunks
                        [+] Freeing Reserve Objects
        [+] Triggering Kernel Use After Free
[+] Completed Use After Free Exploitation
[+] Checking Current Process Privileges
        [+] Trying To Get Process ID Of: csrss.exe
                [+] Process ID Of csrss.exe: 344
        [+] Trying To Open csrss.exe With PROCESS_ALL_ACCESS
                [+] Process Handle Of csrss.exe: 0xEB28
        [+] Successfully Elevated Current Process Privileges
[+] Enjoy As SYSTEM [0.000000]s
```

Type Confusion

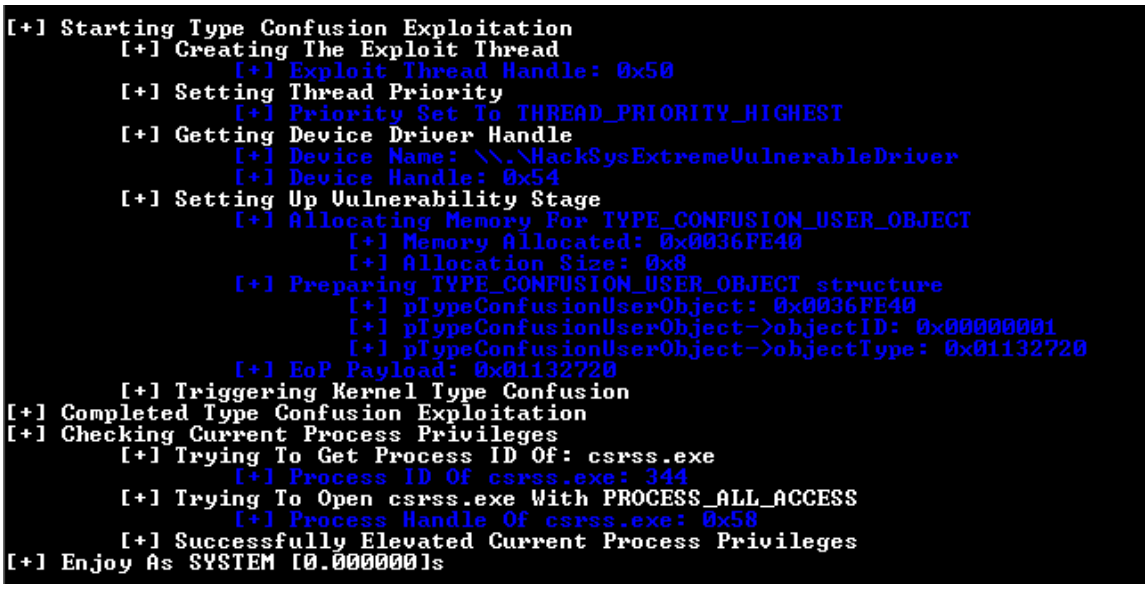
```
// Code snipped for brevity

VOID TypeConfusionObjectCallback() {
    PAGED_CODE();

    DbgPrint("[+] Type Confusion Object Callback called\n");
}
```

```
NTSTATUS TypeConfusionObjectInitializer(PTYPE_CONFUSION_KERNEL_OBJECT
pTypeConfusionKernelObject) {
    NTSTATUS status = STATUS_SUCCESS;
    PAGED_CODE();

    pTypeConfusionKernelObject->pCallback();

    DbgPrint("[+] Type Confusion Object Initialized\n");

    return status;
}

NTSTATUS TriggerTypeConfusion(IN PTYPE_CONFUSION_USER_OBJECT
pTypeConfusionUserObject) {
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PTYPE_CONFUSION_KERNEL_OBJECT pTypeConfusionKernelObject = NULL;
    PAGED_CODE();

    __try {
        ProbeForRead(pTypeConfusionUserObject,
                     sizeof(TYPE_CONFUSION_USER_OBJECT),
                     (ULONG)__alignof(TYPE_CONFUSION_USER_OBJECT));

        pTypeConfusionKernelObject = (PTYPE_CONFUSION_KERNEL_OBJECT)
                                     ExAllocatePoolWithTag(NonPagedPool,
                                     sizeof(TYPE_CONFUSION_KERNEL_OBJECT),
                                     (ULONG)POOL_TAG);

        if (!pTypeConfusionKernelObject) {
            status = STATUS_NO_MEMORY;
            return status;
        }

        pTypeConfusionKernelObject->objectID = pTypeConfusionUserObject->objectID;

        pTypeConfusionKernelObject->objectType = pTypeConfusionUserObject-
>objectType;

        #ifdef SECURE
            pTypeConfusionKernelObject->pCallback = &TypeConfusionObjectCallback;

            status = TypeConfusionObjectInitializer(pTypeConfusionKernelObject);
        #else
            status = TypeConfusionObjectInitializer(pTypeConfusionKernelObject);
        #endif

        ExFreePoolWithTag((PVOID)pTypeConfusionKernelObject, (ULONG)POOL_TAG);

        pTypeConfusionKernelObject = NULL;
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

TypeConfusion.c

```
// Code snipped for brevity

typedef struct _TYPE_CONFUSION_USER_OBJECT {
    ULONG objectID;
    ULONG objectType;
} TYPE_CONFUSION_USER_OBJECT, *PTYPE_CONFUSION_USER_OBJECT;

typedef struct _TYPE_CONFUSION_KERNEL_OBJECT {
    ULONG objectID;
    union {
        ULONG objectType;
        FunctionPointer pCallback;
    };
} TYPE_CONFUSION_KERNEL_OBJECT, *PTYPE_CONFUSION_KERNEL_OBJECT;
```

TypeConfusion.h



Stack Overflow

```
// Code snipped for brevity

NTSTATUS TriggerStackOverflow(IN PVOID pUserModeBuffer, IN SIZE_T
userModeBufferSize) {
    NTSTATUS status = STATUS_SUCCESS;
    ULONG kernelBuffer[BUFFER_SIZE] = {0};

    PAGED_CODE();

    __try {
        ProbeForRead(pUserModeBuffer, sizeof(kernelBuffer),
(ULONG)__alignof(kernelBuffer));

        #ifdef SECURE

            RtlCopyMemory((PVOID)kernelBuffer, pUserModeBuffer,
sizeof(kernelBuffer));
```

```
        #else

            RtlCopyMemory((PVOID)kernelBuffer, pUserModeBuffer,
userModeBufferSize);

        #endif
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

StackOverflow.c

```
[+] Starting Stack Overflow Exploitation
        [+] Creating The Exploit Thread
                [+] Exploit Thread Handle: 0x50
        [+] Setting Thread Priority
                [+] Priority Set To THREAD_PRIORITY_HIGHEST
        [+] Getting Device Driver Handle
                [+] Device Name: \\.\HackSysExtremeVulnerableDriver
                [+] Device Handle: 0x54
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For Buffer
                        [+] Memory Allocated: 0x003F0D38
                        [+] Allocation Size: 0x824
                [+] Preparing Buffer Memory Layout
                        [+] RET Value: 0x01132670
                        [+] RET Address: 0x003F1558
                [+] EoP Shellcode: 0x01132670
        [+] Triggering Kernel Stack Overflow
[+] Completed Stack Overflow Exploitation
[+] Checking Current Process Privileges
        [+] Trying To Get Process ID Of: csrss.exe
                [+] Process ID Of csrss.exe: 344
        [+] Trying To Open csrss.exe With PROCESS_ALL_ACCESS
                [+] Process Handle Of csrss.exe: 0x58
        [+] Successfully Elevated Current Process Privileges
[+] Enjoy As SYSTEM [0.000000]s
```

Integer Overflow

```
// Code snipped for brevity

NTSTATUS TriggerIntegerOverflow(IN PVOID pUserModeBuffer, IN SIZE_T
userModeBufferSize) {
    ULONG arrayCount = 0;
    NTSTATUS status = STATUS_SUCCESS;
    ULONG bufferTerminator = 0xBAD0B0B0;
    ULONG kernelBuffer[BUFFER_SIZE] = {0};
    SIZE_T bufferTerminatorSize = sizeof(bufferTerminator);

    PAGED_CODE();

    __try {
        ProbeForRead(pUserModeBuffer, sizeof(kernelBuffer),
(ULONG)__alignof(kernelBuffer));

        #ifdef SECURE
            if (userModeBufferSize > (sizeof(kernelBuffer) - bufferTerminatorSize))
            {
                status = STATUS_INVALID_BUFFER_SIZE;
```

```
                return status;
            }
        #else
            DbgPrint("[+] Triggering Integer Overflow\n");

            if ((userModeBufferSize + bufferTerminatorSize) > sizeof(kernelBuffer))
            {
                status = STATUS_INVALID_BUFFER_SIZE;
                return status;
            }
        #endif

        while (arrayCount < (userModeBufferSize / sizeof(ULONG))) {
            if (*(PULONG)pUserModeBuffer != bufferTerminator) {
                kernelBuffer[arrayCount] = *(PULONG)pUserModeBuffer;
                pUserModeBuffer = (PULONG)pUserModeBuffer + 1;
                arrayCount++;
            }
            else {
                break;
            }
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

IntegerOverflow.c

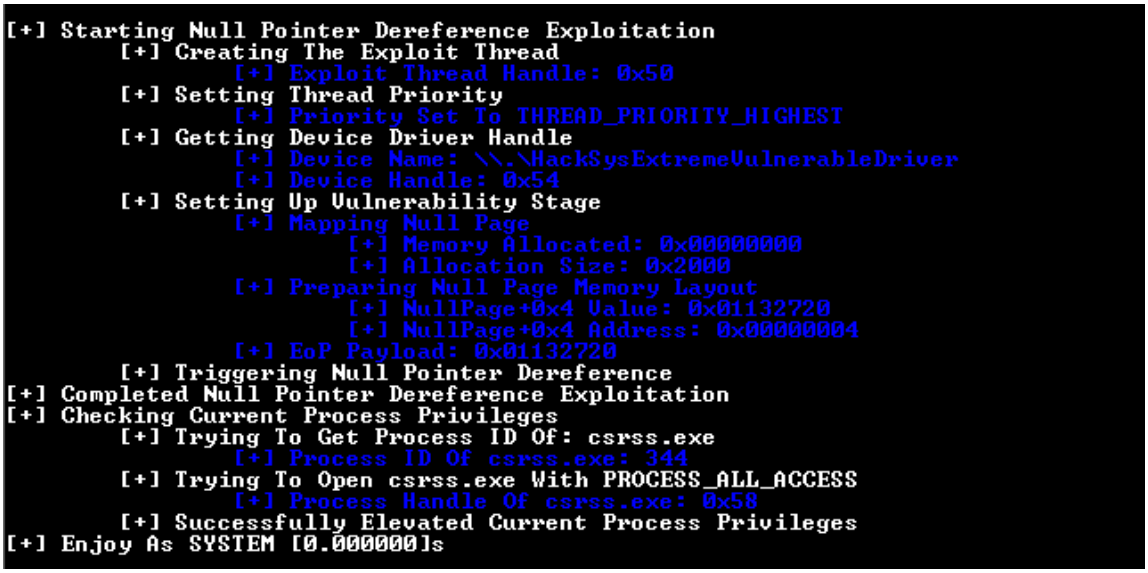
```
[+] Starting Integer Overflow Exploitation
        [+] Creating The Exploit Thread
                [+] Exploit Thread Handle: 0x50
        [+] Setting Thread Priority
                [+] Priority Set To THREAD_PRIORITY_HIGHEST
        [+] Getting Device Driver Handle
                [+] Device Name: \\.\HackSysExtremeVulnerableDriver
                [+] Device Handle: 0x54
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For Buffer
                        [+] Memory Allocated: 0x00320D18
                        [+] Allocation Size: 0x830
                [+] Preparing Buffer Memory Layout
                        [+] RET Value: 0x01132670
                        [+] RET Address: 0x00321540
                [+] EoP Shellcode: 0x01132670
        [+] Triggering Integer Overflow
[+] Completed Integer Overflow Exploitation
[+] Checking Current Process Privileges
        [+] Trying To Get Process ID Of: csrss.exe
                [+] Process ID Of csrss.exe: 344
        [+] Trying To Open csrss.exe With PROCESS_ALL_ACCESS
                [+] Process Handle Of csrss.exe: 0x58
        [+] Successfully Elevated Current Process Privileges
[+] Enjoy As SYSTEM [0.000000]s
```

Null Pointer Dereference

```
// Code snipped for brevity

VOID NullPointerDereferenceObjectCallback()
```

```
    PAGED_CODE();

    DbgPrint("[+] Null Pointer Dereference Object Callback called\n");
}

NTSTATUS TriggerNullPointerDereference(IN PVOID pUserModeBuffer) {
    ULONG userValue = 0;
    ULONG magicValue = 0xBAD0B0B0;
    NTSTATUS status = STATUS_SUCCESS;
    PNULL_POINTER_DEREFERENCE pNullPointerDereference = NULL;

    PAGED_CODE();

    __try {
        ProbeForRead(pUserModeBuffer,
                     sizeof(NULL_POINTER_DEREFERENCE),
                     (ULONG)__alignof(NULL_POINTER_DEREFERENCE));

        pNullPointerDereference = (PNULL_POINTER_DEREFERENCE)
                                  ExAllocatePoolWithTag(NonPagedPool,
                                  sizeof(NULL_POINTER_DEREFERENCE),
                                  (ULONG)POOL_TAG);

        if (!pNullPointerDereference) {
            status = STATUS_NO_MEMORY;
            return status;
        }

        // Get the value from user mode
        userValue = *(PULONG)pUserModeBuffer;

        // Validate the value
        if (userValue == magicValue) {
            pNullPointerDereference->value = userValue;

            pNullPointerDereference->pCallback =
&NullPointerDereferenceObjectCallback;
        }
        else {
            ExFreePoolWithTag((PVOID)pNullPointerDereference, (ULONG)POOL_TAG);
            pNullPointerDereference = NULL;
        }

        #ifdef SECURE

            if (pNullPointerDereference) {
                pNullPointerDereference->pCallback();
            }
        #else

            DbgPrint("[+] Triggering Null Pointer Dereference\n");

            pNullPointerDereference->pCallback();
        #endif
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();
    }

    return status;
}
```

NullPointerDereference.c



Arbitrary Memory Overwrite

```
// Code snipped for brevity

NTSTATUS TriggerArbitraryOverwrite(IN PWRITE_WHAT_WHERE pUserModeWriteWhatWhere) {
    NTSTATUS status = STATUS_SUCCESS;

    PAGED_CODE();

    __try {
        ProbeForRead((PVOID)pUserModeWriteWhatWhere,
                     sizeof(WRITE_WHAT_WHERE),
                     (ULONG)__alignof(WRITE_WHAT_WHERE));

        #ifdef SECURE
            ProbeForRead((PVOID)pUserModeWriteWhatWhere->Where,
                         sizeof(PULONG),
                         (ULONG)__alignof(PULONG));

            ProbeForRead((PVOID)pUserModeWriteWhatWhere->What,
                         sizeof(PULONG),
                         (ULONG)__alignof(PULONG));

            *(pUserModeWriteWhatWhere->Where) = *(pUserModeWriteWhatWhere->What);

        #else
            DbgPrint("[+] Triggering Arbitrary Overwrite\n");

            *(pUserModeWriteWhatWhere->Where) = *(pUserModeWriteWhatWhere->What);
        #endif
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {
        status = GetExceptionCode();

    }

    return status;
}
```

ArbitraryOverwrite.c

```
// Code snipped for brevity

typedef struct _WRITE_WHAT_WHERE {
        PULONG What;
        PULONG Where;
} WRITE_WHAT_WHERE, *PWRITE_WHAT_WHERE;
```

ArbitraryOverwrite.h

```
[+] Starting Arbitrary Memory Overwrite Exploitation
        [+] Creating The Exploit Thread
                [+] Exploit Thread Handle: 0x50
        [+] Setting Thread Priority
                [+] Priority Set To THREAD_PRIORITY_HIGHEST
        [+] Getting Device Driver Handle
                [+] Device Name: \\.\HackSysExtremeVulnerableDriver
                [+] Device Handle: 0x54
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For WRITE_WHAT_WHERE Structure
                        [+] Memory Allocated: 0x0046FE40
                        [+] Allocation Size: 0x8
                [+] Gathering Information About Kernel
                        [+] Loaded Kernel: ntoskrnl.exe
                        [+] Kernel Base Address: 0x82845000
                        [+] HalDispatchTable: 0x829673F8
                        [+] HalDispatchTable+0x4: 0x829673FC
                [+] Preparing WRITE_WHAT_WHERE structure
                        [+] pWriteWhatWhere: 0x0046FE40
                        [+] pWriteWhatWhere->What: 0x007EF8D0
                        [+] pWriteWhatWhere->Where: 0x829673FC
                [+] EoP Payload: 0x01132720
        [+] Triggering Arbitrary Memory Overwrite
                [+] Triggering Payload
[+] Completed Arbitrary Memory Overwrite Exploitation
[+] Checking Current Process Privileges
        [+] Trying To Get Process ID Of: csrss.exe
                [+] Process ID Of csrss.exe: 344
        [+] Trying To Open csrss.exe With PROCESS_ALL_ACCESS
                [+] Process Handle Of csrss.exe: 0x58
        [+] Successfully Elevated Current Process Privileges
[+] Enjoy As SYSTEM [0.000000]s
```

Test Bed

This driver has been successfully tested on **Windows XP SP3 (x86)**, **Windows 2003 SP3 (x86)** and **Windows 7 SP1 (x86)**, but it can support **Windows 8/8.1 (x86)** too. **Windows 8/8.1** support has not been tested now.

Exploits

Yes, exploits have been provided with this project. The exploit has been tested on **Windows 7 SP1 (x86)** and will need tweaking to support other versions of Windows OS.

Source Code

https://github.com/hacksysteam/HackSysExtremeVulnerableDriver

Building Driver

1. Install **Windows Driver Kit** https://www.microsoft.com/en-in/download/details.aspx?id=11800
2. Change **%localSymbolServerPath%** in **Build_HEVD_Secure.bat** and **Build_HEVD_Vulnerable.bat** driver builder
3. Run the appropriate driver builder **Build_HEVD_Secure.bat** or **Build_HEVD_Vulnerable.bat**

Installing Driver

Use **OSR Driver Loader** (https://www.osronline.com/article.cfm?article=157) to install **HackSys Extreme Vulnerable Driver**

Sessions Conducted

- Windows Kernel Exploitation 1 http://null.co.in/event_sessions/156-windows-kernel-exploitation
- Windows Kernel Exploitation 2 http://null.co.in/event_sessions/186-windows-kernel-exploitation-2
- Windows Kernel Exploitation 3 http://null.co.in/event_sessions/226-windows-kernel-exploitation-3
- Windows Kernel Exploitation 4 http://null.co.in/event_sessions/234-windows-kernel-exploitation-4
- Windows Kernel Exploitation 5 http://null.co.in/event_sessions/309-windows-kernel-exploitation-5

Workshops Conducted

- Windows Kernel Exploitation Humla Pune http://null.co.in/event_sessions/280-windows-kernel-exploitation
- Windows Kernel Exploitation Humla Mumbai http://null.co.in/event_sessions/327-windows-kernel-exploitation

References

- https://www.blackhat.com/docs/us-14/materials/us-14-Tarakanov-Data-Only-Pwning-Microsoft-Windows-Kernel-Exploitation-Of-Kernel-Pool-Overflows-On-Microsoft-Windows-8.1.pdf
- http://www.attackingthecore.com/
- http://poppopret.blogspot.com/

- https://cwe.mitre.org/data/definitions/843.html