DSEC-2011-0001

Digit Security Security Advisory

Securstar - DriveCrypt

Multiple Local Kernel Code Execution/Denial of Service

Wednesday 20th July, 2011

(generated on: Wednesday 20th July, 2011)

Multiple Local Kernel Code Execution/Denial of Service - multiple vulnerabilities in the IOCTL interface.

Tel: +44 (0)3300 881337
info@digit-security.com
digit-security.com

Contents

Vulnerability Summary

Vendor:	Securstar
Product:	DriveCrypt
Affected Versions:	5.X
Vendor URL:	http://www.securstar.com/

Author:	Neil 'mu-b' Kettle
CVE Reference:	Not Yet Assigned
BID #:	Not Yet Assigned
Severity:	High
Local/Remote:	Local
Vulnerability Class:	Denial of Service/Memory Disclosure/Privilege Escalation
Impact:	An attacker exploiting this vulnerability may execute arbitrary code with kernel mode privileges, or cause a Denial of Service attack via a page fault caused by an invalid pointer dereference.

1 Detailed Vulnerability Information

1.1 Introduction

Multiple vulnerabilities have been discovered in Securstar DriveCrypt kernel drivers, the vulnerabilities exist due to several somewhat systemic issues in the validation of user-supplied pointers and trust thereof, use of user-supplied parameters to privileged kernel functionality and finally, the lack of bounds checking in unbounded copy operations resulting in buffer overflows.

Securstar documentation describes DriveCrypt as:

"DRIVECRYPT securely and easily protects all proprietary data on notebooks and desktop computers 100% of the time without users having to think about security. Any organization, from a small company to a large international firm with thousands of users in the field, can effectively protect business plans, client lists, product specifications, confidential corporate memos, stock information, and much more with this disk encryption product." [1]

1.2 Technical Background

1.2.1 User-Supplied Pointer Issues

Numerous vulnerabilities exists due to a complete lack of validation of user-supplied pointers contained within structures passed as arguments to the IOCTL interface exported from the globally accessible "\\.\DCR" device.

The following code is the minimum required to reach the defective code within the DriveCrypt kernel driver,

```
#include <stdio.h>
#include <stdlib.h>

#include <windows.h>
#include <ddk/ntapi.h>

#define DCR_IOCTL   0x00073800
...
int
main (int argc, char **argv)
{
  CHAR req[44];
  DWORD rlen;
  HANDLE hFile;
```

```
  hFile = CreateFileA ("\\\\.\\DCR", FILE_EXECUTE,
                       FILE_SHARE_READ|FILE_SHARE_WRITE, NULL,
                       OPEN_EXISTING, 0, NULL);
  memset (&req, 0, sizeof req);
  *((DWORD *) &req) = 0x153;
  *((DWORD *) &req[40]) = 0xDEADBEEF;
  result = DeviceIoControl (hFile, DCR_IOCTL,
                            &req, sizeof req, &req,
                            sizeof req, &rlen, 0);
...
}
```

The vulnerability is present in the IOCTL handler for the "\\.\DCR" device, part of which is given in Figure 1.

In the code given in Figure 1, the user controls the value of `edx` at offset `0x000212B6` which is then dereferenced at offset `0x000212B9`, the value of which is stored in `var_C`. Observing the code and applying transitivity, we have a user controlled value in register `edx` at offset `0x000212FB` which is then written to with value `al`.

It should be noted that the above code is **always** executed with any IOCTL request with value `0x00073800` irrespective of the size parameter passed to the user-land call to DeviceIoControl. As such, it is highly likely that any random request made to the driver will result in a denial of service through a page fault.

1.2.2 User-Supplied Parameters in Kernel Functions

Several vulnerabilities exist due to a lack of validation of user-supplied pointers and values contained within structures passed as arguments to the IOCTL interface which are, in turn, passed to several kernel functions (`ZwXxx` routines).

The following is quoted from [2], *"[H]andles received from user mode [...] should not be passed to ZwXxx routines. Doing so makes a second transition into the kernel. When the ZwXxx routine runs, the previous processor mode is kernel; all access checks [...] are disabled. [...] Similarly, calls to ZwCreateFile or ZwOpenFile with file names provided to the driver will successfully create or open files that should be denied to the caller."*

In the code given in Figure 2, the user controls the value of `edx` at offset `0x000234E2` which is then dereferenced at offset `0x000234E5`, the value of which is stored in `var_B44`. Observing the code and applying transitivity, we have a user controlled value for the `FileHandle`, `Buffer`, `Length`, and `ByteOffset` parameters for the call to `ZwReadFile` at offset `0x000235B1`.

```
.text:00024F0D mov     edx, [ebp+arg_C]
.text:00024F10 push    edx
.text:00024F11 mov     eax, [ebp+Irp]
.text:00024F14 push    eax
.text:00024F15 mov     ecx, [ebp+DeviceObject]
.text:00024F18 push    ecx
.text:00024F19 mov     edx, [ebp+Irp]
.text:00024F1C mov     eax, [edx+0Ch]
.text:00024F1F push    eax              <- user buffer
.text:00024F20 call    sub_21290
...
.text:00021290 00021290:
.text:000212B6 mov     edx, [ebp+arg0]  <- user buffer
.text:000212B9 mov     eax, [edx+28h]
.text:000212BC mov     [ebp+var_C], eax <- user controlled
                                           pointer
.text:000212BF mov     [ebp+var_BBC], offset aDriverBuiltOnA;
                                 "Driver built on Apr  3 2009."
.text:000212C9 mov     ecx, [ebp+var_C]
.text:000212CC add     ecx, 13h
.text:000212CF mov     [ebp+var_BC0], ecx
.text:000212D5 mov     edx, [ebp+var_BC0]
.text:000212DB mov     [ebp+var_BC4]
.text:000212E1 loc_212E1:
.text:000212E1 mov     eax, [ebp+var_BBC]
.text:000212E7 mov     cl, [eax]
.text:000212E9 mov     [ebp+var_BC5], cl
.text:000212EF mov     edx, [ebp+var_BC0]
.text:000212F5 mov     al, [ebp+var_BC5]
.text:000212FB mov     [edx], al         <- user controlled
                                           overwrite
.text:000212FD mov     ecx, [ebp+var_BBC]
.text:00021303 add     ecx, 1
.text:00021306 mov     [ebp+var_BBC], ecx
.text:0002130C mov     edx, [ebp+var_BC0]
.text:00021312 add     edx, 1
.text:00021315 mov     [ebp+var_BC0], edx
.text:0002131B cmp     [ebp+var_BC5], 0
.text:00021322 jnz     short loc_212E1
```

Figure 1: IOCTL handler

```
.text:000234E2 loc_234E2:
.text:000234E2 mov     edx, [ebp+arg0]  <- user buffer
.text:000234E5 mov     eax, [edx+8]
.text:000234E8 mov     [ebp+var_B44], eax
.text:000234EE mov     [ebp+var_B50], 0CE000001h
.text:000234F8 mov     [ebp+var_B48], 0
.text:00023502 mov     ecx, [ebp+var_B44]
.text:00023508 mov     edx, [ecx+8]
.text:0002350B mov     [ebp+var_B58], edx
.text:00023511 mov     eax, [ebp+var_B44]
.text:00023517 mov     ecx, [eax+0Ch]
.text:0002351A mov     [ebp+var_B54], ecx
.text:00023520 mov     edx, [ebp+var_B44]
.text:00023526 cmp     dword ptr [edx+1Ch], 0
.text:0002352A jnz     loc_235EF
...
.text:00023581 loc_23581:              ; Key
.text:00023581 push    0
.text:00023583 lea     edx, [ebp+var_B58]
.text:00023589 push    edx             ; ByteOffset
.text:0002358A mov     eax, [ebp+var_B44]
.text:00023590 mov     ecx, [eax+18h]
.text:00023593 push    ecx             ; Length
.text:00023594 mov     edx, [ebp+P]
.text:0002359A push    edx             ; Buffer
.text:0002359B lea     eax, [ebp+var_B4C]
.text:000235A1 push    eax             ; IoStatusBlock
.text:000235A2 push    0               ; ApcContext
.text:000235A4 push    0               ; ApcRoutine
.text:000235A6 push    0               ; Event
.text:000235A8 mov     ecx, [ebp+var_B44]
.text:000235AE mov     edx, [ecx]
.text:000235B0 push    edx             ; FileHandle
.text:000235B1 call    ds:ZwReadFile
.text:000235B7 mov     [ebp+var_B50], eax
.text:000235BD mov     eax, [ebp+var_B44]
.text:000235C3 mov     ecx, [eax+18h]
.text:000235C6 push    ecx             ; Length
.text:000235C7 mov     edx, [ebp+P]
.text:000235CD push    edx             ; void *
.text:000235CE mov     eax, [ebp+var_B44]
.text:000235D4 mov     ecx, [eax+10h]
.text:000235D7 push    ecx             ; Address
.text:000235D8 call    sub_26800
```

Figure 2: IOCTL handler #2

```
.text:00023938 mov     edx, [ebp+arg0]  <- user buffer
.text:0002393B mov     eax, [edx+8]
.text:0002393E mov     [ebp+var_B98], eax
.text:00023944 mov     ecx, [ebp+ arg0] <- user buffer
.text:00023947 mov     edx, [ecx+18h]
.text:0002394A mov     [ebp+var_B94], edx
.text:00023950 mov     eax, [ebp+var_B94]
.text:00023956 push    eax
.text:00023957 mov     ecx, [ebp+var_B98]
.text:0002395D push    ecx
.text:0002395E call    sub_203E0
...
.text:000203E0 push    ebp
.text:000203E1 mov     ebp, esp
.text:000203E3 sub     esp, 3A4h
.text:000203E9 mov     eax, [ebp+arg_0]
.text:000203EC push    eax
.text:000203ED lea     ecx, [ebp+SourceString]
.text:000203F3 push    ecx
.text:000203F4 call    sub_20380
.text:000203F9 mov     edx, [ebp+arg_4]
.text:000203FC push    edx
.text:000203FD lea     eax, [ebp+var_108]
.text:00020403 push    eax
.text:00020404 call    sub_20380         <- strcpy
```

Figure 3: IOCTL handler #3

It should be noted that similar code is accessible that permits a user-land process to perform calls to `ZwCreateFile` and `ZwWriteFile`. As such, the "\\.\DCR" device exports all the functionality necessary for a user-land process to perform unrestricted (that is, irrespective of permissions) file I/O through its IOCTL interface.

1.2.3 Buffer Overflows

A stack based buffer overflow exists in the IOCTL interface of the IOCTL interface exported from the globally accessible "\\.\DCR" device.

In the code given in Figure 2, the user controls the value of `ecx` at offset `0x00023944` which is then dereferenced at offset `0x00023947`, the value of which is stored in `var_B94`. Observing the code and applying transitivity, we have a user controlled value for the second parameter to the call to the `sub_20380` function at offset `0x00020404`. The `sub_20380` function sub-

sequently performs an `strcpy` from the user-supplied buffer in the second parameter into the buffer given as the first parameter thus causing a stack based buffer overflow.

1.3 Exploit Information

Proof of concept exploit code can be obtained from http://www.digit-labs.org.

2 Vendor Response

Patches are available from the vendor to resolve these issues released with the following remarks,

"Although DriveCrypt has never been hacked, as a precaution in this version we have optimized some disk access operations and fixed a potential common vulnerability present in almost all disk encryption solutions. Also we have fixed some minor bugs and increased the encryption speed." [3]

However, customers should heed the following quotes,

With respect to user-supplied pointers being used without validation, *"I still have mixed feelings about it. I don't expect to call my driver functions myself with bad buffers, and they were never intended to be called by anyone else. Of course if attackers are using it to crash the computer perhaps I should think again."*

With respect to user-supplied parameters being passed to kernel functions without validation, *"The user mode app still leverages the driver for some of the I/O, but in a way which cannot be exploited as easily as before, without some prior transient elevation to admin level. I am still checing [sic] a couple of aspects to be sure it is reasonably secure, IE less easy to exploit."*

It should furthermore be noted that no attempt has been made by Digit-Security to verify the patches provided by Securstar.

3 Recommendations

It is recommended that affected systems are updated to the latest patch level available from Securstar.

References

[1] Securstar GmbH. Securstar, Encryption Software Solutions - Products - Drivecrypt. http://www.securstar.com/products_drivecrypt.php, 2011.

[2] Microsoft Corporation. Common Driver Reliability Issues. http://msdn.microsoft.com/en-us/library/ms809962.aspx, 2009.

[3] Securstar GmbH. Securstar, Encryption Software Solutions - Press Area. http://www.securstar.com/press.php?id_press=405, 2011.