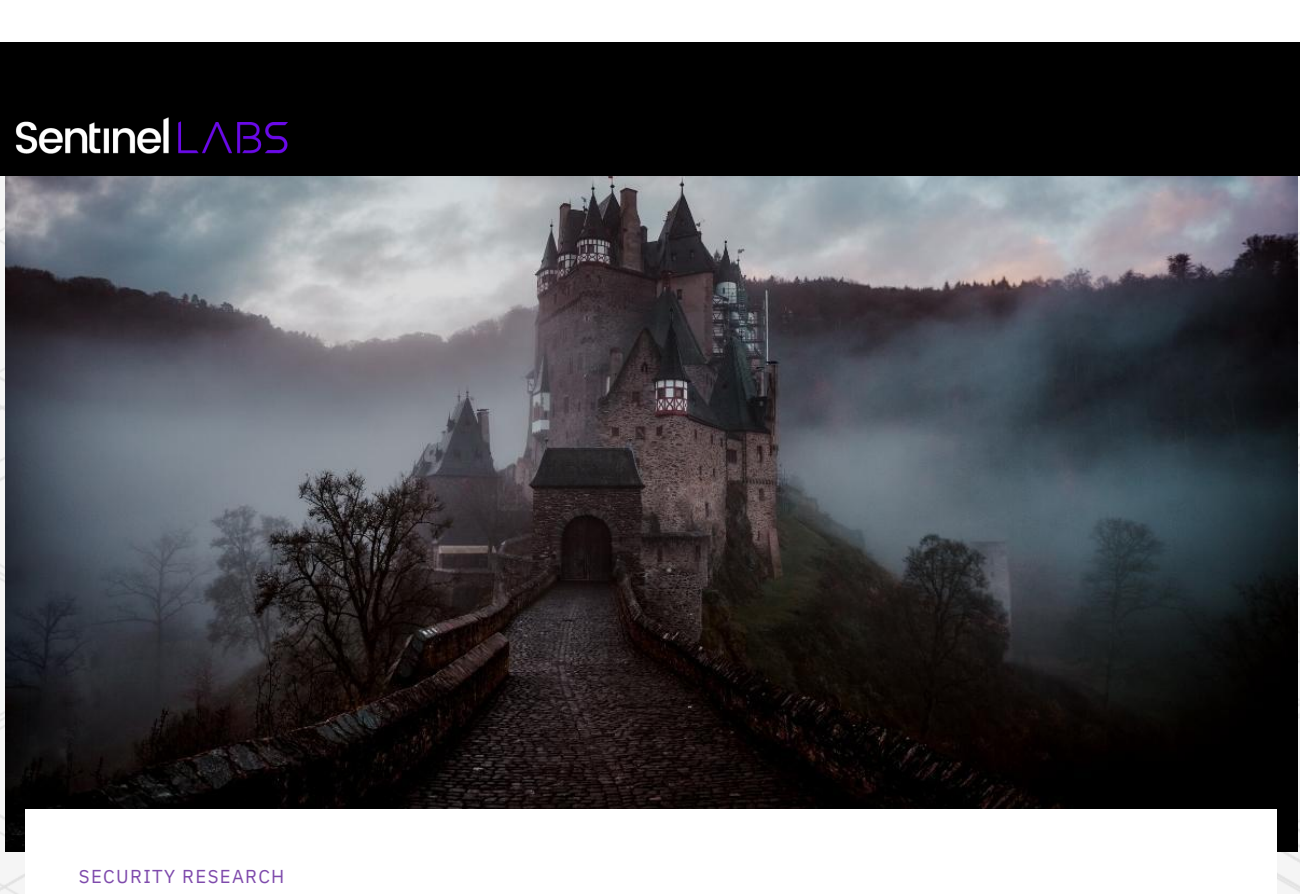


SECURITY RESEARCH

CVE-2021-45608 | NetUSB RCE Flaw in Millions of End User Routers

MAX VAN AMERONGEN / JANUARY 11, 2022

Executive Summary

- SentinelLabs has discovered a high severity flaw in the KCodes NetUSB kernel module used by a large number of network device vendors and affecting millions of end user router devices.
- Attackers could remotely exploit this vulnerability to execute code in the kernel.
- SentinelLabs began the disclosure process on the 9th of September and the patch was sent to vendors on the 4th of October.
- At this time, SentinelOne has not discovered evidence of in-the-wild abuse.

Introduction

As a number of my projects start, when I heard that Pwn2Own Mobile 2021 had been announced, I set about looking at one of the targets. Having not looked at the Netgear device when it appeared in the 2019 contest, I decided to give it a lookover.

While going through various paths through various binaries, I came across a kernel module called NetUSB. As it turned out, this module was listening on TCP port `20005` on the IP `0.0.0.0`.

Provided there were no firewall rules in place to block it, that would mean it was listening on the WAN as well as the LAN. Who wouldn't love a remote kernel bug?

NetUSB is a product developed by KCodes. It's designed to allow remote devices in a network to interact with USB devices connected to a router. For example, you could interact with a printer as though it is plugged directly into your computer via USB. This requires a driver on your computer that communicates with the router through this kernel module.

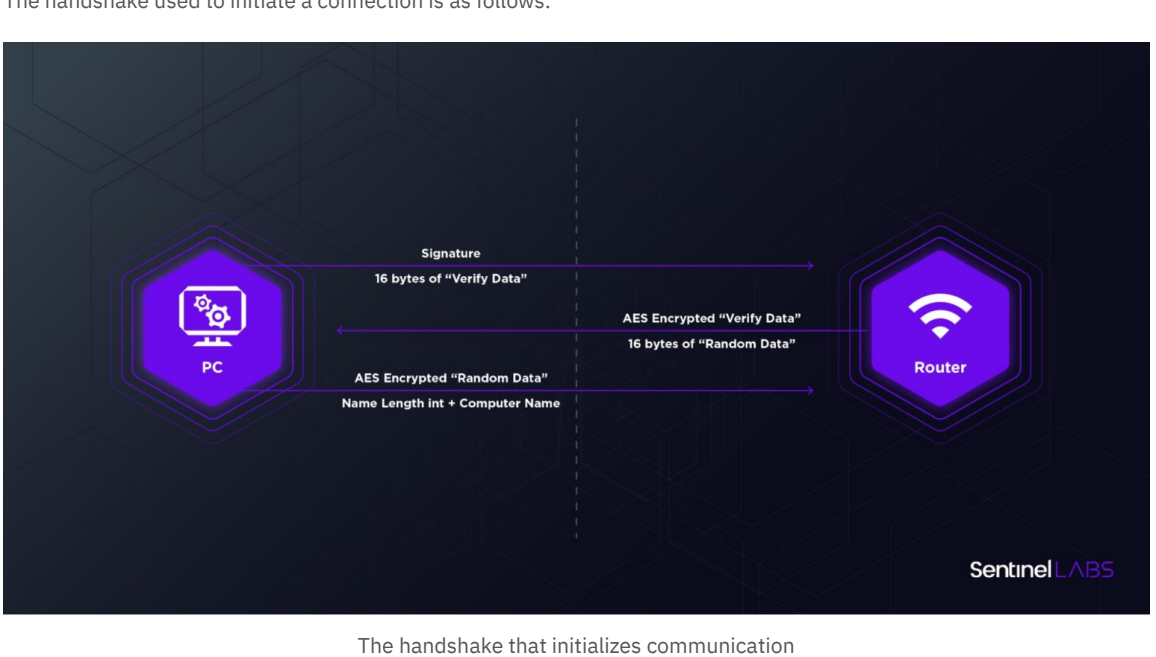
It's licensed to a large number of other vendors for use in their products, most notably:

- Netgear
- TP-Link
- Tenda
- EDiMAX
- DLink
- Western Digital

NetUSB.ko Internals

Back in 2015 a different NetUSB vulnerability was discovered. From that came some great resources (including a very helpful exploit for that vulnerability by bl4sty which helped quickly verify this vulnerability).

The handshake used to initiate a connection is as follows:



The handshake that initializes communication

After the handshake, a command-parsing while-loop is executed that contains the following code:

```
/* payload starts here */
read_success = SoftwareBus_fillBuf(sbus_info,(char *)((int)&cmd + 3),1);
if ((int)read_success == 0) break;
if ((cmd & 0x80000000) == 0) {
  SoftwareBus_dispatchHostCommands(sbus_info,cmd >> 0x18,constant_1,(int3)t_flag);
}
else {
  read_success = SoftwareBus_fillBuf(sbus_info,(char *)((int)&cmd + 2),1);
  if ((int)read_success == 0) break;
  partial_cmd = cmd >> 0x10 & 0xff;
  if ((cmd & 0xf0000) == 0) {
    SoftwareBus_dispatchEP0MsgOut(sbus_info,cmd >> 0x18,partial_cmd,t_flag);
  }
  else {
    SoftwareBus_dispatchNormalEPMsgOut(sbus_info,cmd >> 0x18,partial_cmd);
  }
}
```

The code that takes a command number and routes the message to the appropriate SoftwareBus function

`SoftwareBus_fillBuf` acts in a similar way to `recv` by taking both a buffer and its size, filling the buffer with data read from the socket.

The Vulnerability

The command `0x805f` reaches the following code in the function `SoftwareBus_dispatchNormalEPMsgOut`:

```
uVar17 = SoftwareBus_fillBuf(sbus_info,(char *)&supplied_size,4);
if ((int)uVar17 == 0) {
  return;
}
allocated_region = (char *)__kmalloc(supplied_size + 0x11,0xd0);
if ((soft_bus_info *)allocated_region == (soft_bus_info *)0x0) {
  log_msg = "INFO%04X: Out of memory in USBSoftwareBus";
  log_line = (char *)0x1156;
  goto LAB_0001a8fc;
}
```

The vulnerable segment of code in the kernel module

4 bytes are fetched from the remote PC. The number `0x11` is added to it and then used as a size value in `kmalloc`. Since this supplied size isn't validated, the addition of the `0x11` can result in an integer overflow. For example, a size of `0xffffffff` would result in `0x10` after `0x11` has been added to it.

This allocated region is then used and written to through both dereferencing and through the `SoftwareBus_fillBuf` function:

```
uVar14 = SoftwareBus_fillBuf(sbus_info,(char *)&supplied_size,4);
if ((int)uVar14 == 0) {
  return;
}
allocated_region = (vuln_struct *)__kmalloc(supplied_size + 0x11,0xd0);
if (allocated_region == (vuln_struct *)0x0) {
  pcVar1 = "INFO%04X: Out of memory in USBSoftwareBus";
  uVar2 = 0x1156;
  goto LAB_0001a8fc;
}
allocated_region->cmd_shifted = cmd_shifted_;
allocated_region->partial_cmd = partial_cmd_;
allocated_region->allocated_size = supplied_size;
uVar14 = SoftwareBus_fillBuf(sbus_info,allocated_region->number_of_packets,4);
uVar2 = (undefined4)((ulonglong)uVar14 >> 0x20);
if ((int)uVar14 != 0) {
  uVar14 = SoftwareBus_fillBuf(sbus_info,allocated_region->packet_size,4);
  uVar2 = (undefined4)((ulonglong)uVar14 >> 0x20);
  if ((int)uVar14 != 0) {
    if (sbus_info->field_0x296 == 0) {
      allocated_region->flag = 0;
    }
    else {
      uVar14 = SoftwareBus_fillBuf(sbus_info,allocated_region->flag,1);
      uVar2 = (undefined4)((ulonglong)uVar14 >> 0x20);
      if ((int)uVar14 == 0) goto LAB_0001af2c;
    }
    uVar14 = SoftwareBus_fillBuf(sbus_info,allocated_region->data,supplied_size);
```

Out-of-bounds writes taking place on the small allocated region

Looking at the final call to `SoftwareBus_fillBuf`, the supplied size is used as a maximum value to read from the remote socket. From the previous example, the size `0xffffffff` would be used here (not the overflown value) as the size sent to `recv`.

Along with our report, we sent a suggested mitigation strategy. Before allocating memory with user supplied sizes, an integer overflow check should be performed, as so:

```
if(user_supplied_size + 0x11 < 0x11) return;
```

Exploitability

From an exploit perspective, there are a number of things to consider.

First, the minimum size we can allocate is `0x0` and the maximum is `0x10`. That means that the allocated object will always be in the `kmalloc-32` slab of the kernel heap.

Second, we need to consider the amount of control over the overflow itself. We already know that the data being received over the socket is within control of the attacker, but is the size negotiable in any way? Since a size of `0xffffffff` is not realistically exploitable on a 32-bit system, it's necessary to take a look at how `SoftwareBus_fillBuf` actually works. Underneath this function is the standard socket `recv` function. That means that the size supplied is only used as a maximum receive size and not a strict amount, like `memcpy`.

It's also important to consider how easy it is going to be to lay out the kernel heap for the overflow. Many exploits require the use of heap holes in order to make sure that the vulnerable heap structure will be placed before the object that will be overwritten. In the case of this kernel module, there's a timeout of 16 seconds on the socket for receiving data, meaning the struct can be overflown up to 16 seconds after it is allocated. This removes the need to create a heap hole.

Finally, the selection of target structures that could be overwritten needs to be considered. There are some constraints as to which ones can be used.

- The structure must be less than 32 bytes in size in order to fit into `kmalloc-32`.
- The structure must be sprayable from a remote perspective.
- The structure must have something that can be overwritten that makes it useful as a target (e.g. a Type-Length-Value structure or a pointer)

While these restrictions make it difficult to write an exploit for this vulnerability, we believe that it isn't impossible and so those with Wi-Fi routers may need to look for firmware updates for their router.

Remediation

Since this vulnerability is within a third party component licensed to various router vendors, the only way to fix this is to update the firmware of your router, if an update is available. It is important to check that your router is not an end-of-life model as it is unlikely to receive an update for this vulnerability.

Exploring the Netgear firmware update, the vulnerability was patched by adding a new size check to the function:

```
iVar2 = SoftwareBus_fillBuf(sbus_info,&supplied_size,4);
if (iVar2 == 0) {
  return;
}
if (supplied_size < 0x1000000) {
  allocated_region = (int *)__kmalloc(supplied_size + 0x11,0xd0);
  if (allocated_region != (int *)0x0) {
```

The patch for the vulnerability, as implemented by Netgear

Conclusion

This vulnerability affects millions of devices around the world and in some instances may be completely remotely accessible. Due to the large number of vendors that are affected by the vulnerability, we reported this vulnerability directly to KCodes to be distributed among their licensees instead of targeting just the TP-Link or the Netgear device in the contest. This ensures that all vendors receive the patch instead of just one during the contest.

While we are not going to release any exploits for it, there is a chance that one may become public in the future despite the rather significant complexity involved in developing one. We recommend that all users follow the remediation information above in order to reduce any potential risk.

Disclosure Timeline

- 09 Sep, 2021 - An initial email to KCodes, requesting information on how to send the vulnerability information
- 20 Sep, 2021 - The vulnerability details and a patch suggestion is disclosed to KCodes with a final disclosure date of December 20, 2021
- 04 Oct, 2021 - A proof-of-concept script is requested by KCodes to verify the patch
- 04 Oct, 2021 - A proof-of-concept script is provided
- 17 Nov, 2021 - An email is sent to KCodes to double check that the patch was sent out to all vendors on the 4th of October, and not just Netgear
- 19 Nov, 2021 - KCodes confirms that they had sent the patch to all vendors and that the firmware would be out before the 20th December
- 14 Dec, 2021 - Netgear was found to have released firmware for the R6700v3 device with the changes implemented
- 20 Dec, 2021 - Netgear releases an advisory for the vulnerability
- 11 Jan, 2022 - SentinelLabs publicly disclose details of the vulnerability

SentinelOne's responsible disclosure policy can be found here.

VULNERABILITY

MAX VAN AMERONGEN

Max Van Amerongen is a Vulnerability Researcher at SentinelOne focusing on identifying security holes in critical software. Before joining SentinelOne, he had previously worked at F-Secure Labs where he successfully participated in the Pwn2Own hacking contest a number of times.