Talos Vulnerability Report

TALOS-2022-1511

Asuswrt and Asuswrt-Merlin New Gen httpd unescape memory corruption vulnerability

JULY 27, 2022

CVE NUMBER

CVE-2022-26376

SUMMARY

A memory corruption vulnerability exists in the httpd unescape functionality of Asuswrt prior to 3.0.0.4.386_48706 and Asuswrt-Merlin New Gen prior to 386.7.. A specially-crafted HTTP request can lead to memory corruption. An attacker can send a network request to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

Asuswrt-Merlin New Gen prior to 386.7
Asus routers with firmware prior to:
XT8 - 3.0.0.4.386_48706
TUF-AX3000_V2 - 3.0.0.4.386_48750
XD4 - 3.0.0.4.386_48790
ET12 - 3.0.0.4.386_48823
GT-AX6000 - 3.0.0.4.386_48823
XT12 - 3.0.0.4.386_48823
RT-AX58U - 3.0.0.4.386_48908
XT9 - 3.0.0.4.388_20027
XD6 - 3.0.0.4.386_49356
GT-AX11000 PRO - 3.0.0.4.386_48996
GT-AXE16000 - 3.0.0.4.386_48786
RT-AX86U - 3.0.0.4.386_49447

RT-AX68U - 3.0.0.4.386_49479

RT-AX56U - 3.0.0.4.386_49380

RT-AX82U - 3.0.0.4.386_49559

RT-AX55 - 3.0.0.4.386_49559

GT-AX11000 - 3.0.0.4.386_49559

PRODUCT URLS

Asuswrt-Merlin New Gen - https://github.com/RMerl/asuswrt-merlin.ng

CVSSV3 SCORE

5.3 - CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

CWE

CWE-787 - Out-of-bounds Write

DETAILS

The Asuswrt-Merlin New Gen is an open source firmware alternative for Asus routers.

The Asuswrt-Merlin New Gen's httpd component, has a file named `cgi.c` that contains CGI helper functions. One of these functions is `unescape`:

```
void
unescape(char *s)
{
    char s_tmp[65535];
    unsigned int c;

    while ((s = strpbrk(s, "%+"))) {
[1]
        /* Parse %xx */
        if (*s == '%') {
            sscanf(s + 1, "%02x", &c);
[2]
            *s++ = (char) c;
[3]
            strlcpy(s_tmp, s + 2, sizeof(s_tmp));
[4]
            strncpy(s, s_tmp, strlen(s) + 1);
[5]
        }
        /* Space is special */
        else if (*s == '+')
            *s++ = ' ';
    }
}
```

This function takes as argument a string. If URL-encoded, this function will decode it. At `[1]`, a loop takes the next `%` or `+` in the string. If a `%` is found, then at `[2]`, the two characters following it are converted from hex values to a single character. At `[3]` the converted character replaces the `%` character and the string pointer advances. At `[4]` and `[5]`, the string after the already-parsed URL-encoded character is moved left by two positions. This will replace the parsed characters. A string like “A…B%41%42” would go through the following steps:

```
|A|...|B|%|4|1|%|4|2|NULL|                      at    [1]/[2]
|A|...|B|A|4|1|%|4|2|NULL|                      after [3]
|A|...|B|A|%|4|2|NULL|NULL|NULL|                after [5]
```

Eventually, after a second iteration of the loop, the string would end up like this:

```
|A|...|B|A|B|NULL|NULL|NULL|NULL|NULL|          after [5]
```

The `unescape` function assumes, wrongly, that after a `%` there are always at least two characters. If this is not the case, the instruction at `[4]` would cause an out-of-bounds read, and the one at `[5]` could cause the removal of the null terminator. This removal, in the next loop iteration, could cause an out-of-bounds write. Let's take for instance the following string "A…B%a". This would go through the following steps:

```
|A|...|B|%|a|NULL|Q|Q|Q|Q|Q|                          at    [1]/[2]
|A|...|B|\n|a|NULL|Q|Q|Q|Q|Q|                         after [3]
            ^ s points to 'a'                         after [3]
```

After the string there is other data, in this scenario the string "QQQQQ", `s+2`, at `[4]`, which will point to the first `Q`. So after `[4]` the string will look like:

```
|A|...|B|%|a|NULL|Q|Q|Q|Q|Q|                          at    [1]/[2]
|A|...|B|\n|a|NULL|Q|Q|Q|Q|Q|                         after [3]
                   ^ s+2 point to the first Q         at [4]
|A|...|B|\n|Q|Q|Q|Q|Q|Q|Q|                            after [5]
```

The result would be the string "A…B\nQQQQQQQ…".

TIMELINE

2022-04-11 - Vendor Disclosure

2022-04-11 - Initial Vendor Contact

2022-07-27 - Public Release

CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.