Talos Vulnerability Report

TALOS-2022-1568

# Abode Systems, Inc. iota All-In-One Security Kit web interface /action/wirelessConnect OS command injection vulnerabilities

OCTOBER 20, 2022

## CVE NUMBER

CVE-2022-33207,CVE-2022-33205,CVE-2022-33206,CVE-2022-33204

## SUMMARY

Four OS command injection vulnerabilities exists in the web interface /action/wirelessConnect functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9X and 6.9Z. A specially-crafted HTTP request can lead to arbitrary command execution. An attacker can make an authenticated HTTP request to trigger these vulnerabilities.

## CONFIRMED VULNERABLE VERSIONS

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

abode systems, inc. iota All-In-One Security Kit 6.9X
abode systems, inc. iota All-In-One Security Kit 6.9Z

## PRODUCT URLS

iota All-In-One Security Kit - https://goabode.com/product/iota-security-kit

## CVSSV3 SCORE

10.0 - CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

## CWE

CWE-78 - Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)

## DETAILS

The iota All-In-One Security Kit is a home security gateway containing an HD camera, infrared motion detection sensor, Ethernet, WiFi and Cellular connectivity. The iota gateway orchestrates communications between sensors (cameras, door and window alarms, motion detectors, etc.) distributed on the LAN and the Abode cloud. Users of the iota can communicate with the device through mobile application or web application.

These vulnerabilities are reminiscent of a vulnerability identified and reported previously by Bitdefender. The vulnerability was tracked as CVE-2020-8105. From screenshots, the version of the firmware they tested appears to be 6.8E. Abode released firmware version 6.9X on December 15th, 2021 as the patch for CVE-2020-8105.

As brief background, CVE-2020-8105 was identified in the function responsible for handling HTTP requests destined for `/action/wirelessConnect` on the local web interface of the device. The vulnerability was an OS Command Injection that could be exploited through any of three parameters: `ssid`, `wpapsk` and `key`. It is our belief that the released patch for this vulnerability was to disable the web interface by default and encode the `ssid` and `wpapsk` parameters as hex-ascii in order to neutralize them before injecting into the OS command. Included in this patch were two new parameters, `ssid_hex` and `wpapsk_hex`, which are treated as if they have already been encoded as hex-ascii.

The web interface, now disabled-by-default, can be re-enabled through the use of either TALOS-2022-1552 or TALOS-2022-1553 in order to make these vulnerabilities accessible over the network.

The vulnerable `web_wireless_connect` function is located at offset `0x19AC94` in the `/root/hpgw` binary included in firmware version 6.9Z. For brevity, we have included only the relevant portions of the decompiled function below, with annotations.

```
int __fastcall wirelessConnect(mg_connection *conn, mg_request_info *ri)
{
  int payload_len;
  int v5;
  const char *v6;
  int decoded_ssid_len;
  int bufflen;
  const char *cmd;
  int count;
  int success;
  int error_len;
  const char *error;
  const char *err_str;
  const char *task;
  const char *v17;
  int wireless_enabled;
  xstrbuf_t decoded_ssid;
  xstrbuf_t decoded_wpapsk;
  char cmd_buffer[128];
  char cmd_output[128];
  char scratchpad[128];
  char ssid[192];
  char ssid_hex[192];
  char wpapsk_hex[192];
  char auth_mode[192];
  char wpapsk[192];
  char encryp_type[192];
  char default_key_id[192];
  char key[192];
  char payload[544];

  log(7, 1, "\x1B[33mweb_wireless_connect ri->uri:[%s]\x1B[0m", ri->uri);
  wireless_enabled = 0;

  // [1] Ensure that wireless is enabled, otherwise exit early
  get_config_as_integer("WL_Enable", (int)&wireless_enabled);
  if ( !wireless_enabled )
    goto LABEL_38;

  ...

  // [2] Extract the first 512 bytes of the POST body and extract all necessary
parameters
  payload_len = http_collect_payload(conn, ri, payload, 512);
  memset(ssid, 0, sizeof(ssid));
  memset(ssid_hex, 0, sizeof(ssid_hex));
  memset(wpapsk_hex, 0, sizeof(wpapsk_hex));
  memset(auth_mode, 0, sizeof(auth_mode));
  memset(wpapsk, 0, sizeof(wpapsk));
  memset(encryp_type, 0, sizeof(encryp_type));
  memset(default_key_id, 0, sizeof(default_key_id));
  memset(key, 0, sizeof(key));
  mg_get_var(payload, payload_len, "ssid", ssid, 191);
  mg_get_var(payload, payload_len, "ssid_hex", ssid_hex, 191);
  mg_get_var(payload, payload_len, "auth_mode", auth_mode, 191);
  mg_get_var(payload, payload_len, "wpapsk", wpapsk, 191);
  mg_get_var(payload, payload_len, "wpapsk_hex", wpapsk_hex, 191);
  mg_get_var(payload, payload_len, "encryp_type", encryp_type, 191);
  mg_get_var(payload, payload_len, "default_key_id", default_key_id, 191);
  mg_get_var(payload, payload_len, "key", key, 191);

  // [3] Ensure that one of `ssid` or `ssid_hex` is provided, all other parameters
appear optional
  if ( !ssid[0] )
  {
    v5 = (unsigned __int8)ssid_hex[0];
    if ( !ssid_hex[0] )
    {
      log(7, 1, "No SSID!");
      v6 = strtable_get("WEB_ERR_PARAM_EMPTY", 19);
      return web_error(conn, v5, v6, "ssid and ssid_hex");
    }
  }

  memset(cmd_buffer, 0, sizeof(cmd_buffer));
  memset(v22, 0, sizeof(v22));
  xstrbuf(&decoded_ssid);
  mg_urldecode(ssid, &decoded_ssid);

  // [4] Identify whether the SSID was provided via the `ssid` or `ssid_hex`
parameter
  if ( ssid_hex[0] )
  {
    // [5] If via `ssid_hex`, inject directly into OS command without neutralization
    log(7, 31, "with hex string");
    vsnprintf_nullterm(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 ssid
%s", "wlan0", ssid_hex);
  }
  else
  {
    // [6] If via `ssid`, neutralize the value before injecting into OS command
    memset(scratchpad, 0, sizeof(scratchpad));
    decoded_ssid_len = strlen(decoded_ssid.buff);
    hexlify(decoded_ssid.buff, decoded_ssid_len, scratchpad);
    log(7, 31, "with ascii string");
    vsnprintf_nullterm(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 ssid
'\"%s\"'", "wlan0", scratchpad);
  }
  xstrbuf_free_d(&decoded_ssid, (int)"web_wireless_connect");
  log(7, 1, cmd_buffer);

  // [7] Execute the command constructed at [5] or [6]
  popen_write(cmd_buffer);

  memset(cmd_buffer, 0, sizeof(cmd_buffer));
  if ( strcmp(auth_mode, "WPA") && strcmp(auth_mode, "WPA2") )
  {

    // [8] If `auth_mode` is WPAPSK_HEX or WPA2PSK_HEX
    if ( !strcmp(auth_mode, "WPAPSK_HEX") || !strcmp(auth_mode, "WPA2PSK_HEX") )
    {

      // [9] then inject `wpapsk_hex` directly into the command without
neutralization
      vsnprintf_nullterm(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 psk
%s", "wlan0", wpapsk_hex);
LABEL_16:
      log(7, 1, cmd_buffer);

      // [10] Execute the command constructed at [9]
      popen_read(cmd_buffer, cmd_output, 127);

      if ( strlen(cmd_output) > 1 && !strncmp(cmd_output, "OK", 2u) )
        log(7, 1, "web_wireless_connect set_network 0 psk OK");
      goto LABEL_25;
    }

    // [11] Otherwise, if `auth_mode` is WPAPSK or WPA2PSK
    if ( !strcmp(auth_mode, "WPAPSK") || !strcmp(auth_mode, "WPA2PSK") )
    {
      // [12] Then neutralize `wpapsk` before injecting it into the command
      xstrbuf(&decoded_wpapsk);
      mg_urldecode(wpapsk, &decoded_wpapsk);
      memset(scratchpad, 0, sizeof(scratchpad));
      bufflen = strlen(decoded_wpapsk.buff);
      hexlify(decoded_wpapsk.buff, bufflen, scratchpad);
      vsnprintf_nullterm(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 psk
'\"%s\"'", "wlan0", scratchpad);
      xstrbuf_free_d(&decoded_wpapsk, (int)"web_wireless_connect");
      goto LABEL_16;
    }
    if ( strcmp(auth_mode, "SHARED") && strcmp(auth_mode, "WEP") )
    {
      log(7, 1, "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");
      cmd = "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE";
      goto LABEL_24;
    }

    log(7, 1, "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");
    popen_write("driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");

    // [12]  If `auth_mode` is SHARED or WEP then inject `default_key_id` and `key`
into the command directly
    vsnprintf_nullterm(
      cmd_buffer,
      0x7Fu,
      "driver/wpa_cli -i %s set_network 0 wep_key%s '\"%s\"'",
      "wlan0",
      default_key_id,
      key);
    log(7, 1, cmd_buffer);

    // [13] Execute the command constructed at [12]
    popen_write(cmd_buffer);
    memset(cmd_buffer, 0, sizeof(cmd_buffer));

    // [14] Construct a final command using only the unsanitized `default_key_id`
    vsnprintf_nullterm(
      cmd_buffer,
      0x7Fu,
      "driver/wpa_cli -i %s set_network 0 wep_tx_keyidx %s",
      "wlan0",
      default_key_id);
    log(7, 1, cmd_buffer);

    // [15] Execute the final vulnerable command constructed at [14]
```

```
      popen_write(cmd_buffer);
      if ( !strcmp(auth_mode, "SHARED") )
      {
        log(7, 1, "driver/wpa_cli -i wlan0 set_network 0 auth_alg SHARED");
        cmd = "driver/wpa_cli -i wlan0 set_network 0 auth_alg SHARED";
LABEL_24:
        popen_write(cmd);
      }
    }
...
}
```

We observe at `[1]`, these vulnerabilities are only exploitable if wireless functionality has been enabled. This is not particularly hard to configure as a remote attacker via TALOS-2022-1552, but most devices will not be immediately exploitable. If `WL_Enable` is true, then at `[3]` the first 512 bytes of the request body are extracted into the `payload` variable and eight parameters are extracted from the POST body—`ssid`, `ssid_hex`, `auth_mode`, `wpapsk`, `wpapsk_hex`, `encryp_type`, `default_key_id` and `key`. A value must be supplied for one, or both, of `ssid` and `ssid_hex`. All other parameters appear to be optional.

## CVE-2022-33204 - Authenticated OS Command Injection via `ssid_hex` parameter

The conditional referenced at `[4]` is responsible for determining whether the supplied `ssid*` parameter needs to be encoded prior to injection: The `ssid` parameter will be neutralized, but the `ssid_hex` parameter will be injected directly. There is nothing to enforce that the `ssid_hex` parameter has already been safely neutralized. At `[7]` the vulnerable command constructed using `ssid_hex` at `[5]` is executed by the root user via a call to `popen`.

A maliciously-formatted `ssid_hex` parameter successfully submitted via authenticated HTTP request to the `/action/wirelessConnect` endpoint will result in arbitrary command execution.

## CVE-2022-33205 - Authenticated OS Command Injection via `wpapsk_hex` parameter

The conditionals referenced at `[8]` and `[11]` are responsible for determining whether the supplied `wpapsk*` parameter needs to be encoded prior to injection. If `auth_mode` is “WPAPSK” OR “WPA2PSK” then the `wpapsk` parameter will be neutralized, but if the `auth_mode` is “WPAPSK_HEX” or “WPA2PSK_HEX” then `wpapsk_hex` parameter will be used directly. There is nothing to enforce that the `wpapsk_hex` parameter has already been safely neutralized. At `[10]`, the vulnerable command constructed using `wpapsk_hex` at `[9]` is executed by the root user via a call to `popen`.

A maliciously-formatted `wpapsk_hex` parameter successfully submitted via authenticated HTTP request to the `/action/wirelessConnect` endpoint will result in arbitrary command execution.

## CVE-2022-33206 - Authenticated OS Command Injection via `key` and `default_key_id` parameters

If the conditional referenced at `[12]` identifies that `auth_mode` is “SHARED” or “WEP”, then at `[13]` both the `key` and `default_key_id` are injected directly into an OS command. There is nothing to enforce that either of these parameters have been safely neutralized prior to use. At `[14]`, the vulnerable command is executed via a call to `popen`.

A maliciously-formatted `key` or `default_key_id` parameter successfully submitted via authenticated HTTP request to the `/action/wirelessConnect` endpoint will result in arbitrary command execution.

## CVE-2022-33207 - Authenticated OS Command Injection via `default_key_id` parameters

If the conditional referenced at `[12]` identifies that `auth_mode` is “SHARED” or “WEP”, then at `[15]` the `default_key_id` parameter will be injected directly into an OS command. There is nothing to enforce that the `default_key_id` has already been safely neutralized. At `[16]`, the vulnerable command is executed by the root user via a call to `popen`.

A maliciously-formatted `default_key_id` parameter successfully submitted via authenticated POST request to the `/action/wirelessConnect` endpoint will result in arbitrary command execution.

### TIMELINE

2022-07-14 - Vendor Disclosure
2022-10-20 - Public Release

### CREDIT

Discovered by Matt Wiseman of Cisco Talos.

---

VULNERABILITY REPORTS

PREVIOUS REPORT
TALOS-2022-1567

NEXT REPORT
TALOS-2022-1581