Talos Vulnerability Report

TALOS-2022-1585

Abode Systems, Inc. iota All-In-One Security Kit web interface /action/wirelessConnect format string injection vulnerabilities

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-35887,CVE-2022-35884,CVE-2022-35886,CVE-2022-35885

SUMMARY

Four format string injection vulnerabilities exist in the web interface /action/wirelessConnect functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9Z and 6.9X. A specially-crafted HTTP request can lead to memory corruption, information disclosure and denial of service. 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

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

CWE

CWE-134 - Use of Externally-Controlled Format String

DETAILS

The iota All-In-One Security Kit is a home security gateway containing an HD camera, infrared motion detection sensor, Ethernet, Wi-Fi 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.

The `iota` device generates a significant volume of diagnostic logs, which it displays on its read-only physical UART console. These logs are formatted and put on the serial line inside of a function (located at offset `0xA3270`) which we refer to simply as `log`. The `log` function operates as a wrapper to `vsnprintf` and `puts` with the added functionality of prefixing supplied log messages with severity and task strings. The `log` function is variadic and crafts the final log message by passing the supplied `format` and variadic arguments to `vsnprintf`. If an attacker can inject content into the `format` parameter, they could potentially leak stack memory and write arbitrary memory.

```
/* Examples:
        log(6, 13, "Initialized SSL");  ->  [DBG!][NET ]Initialized SSL
        log(3, 13, "SSL init error: %d", error);  ->  [ERR!][NET ]SSL init error:
{error}
*/
void log(unsigned int severity, unsigned int task, const char *format, ...)
{
  char log_buffer[520];
  va_list var_args;

  va_start(var_args, format);
  if ( severity <= g_LOG_LEVEL && ((g_TASK_LOGGING_ENABLED_BITFIELD >> task) & 1) !=
0 )
  {
    // Prefix the message with the SEVERITY tag
    if ( severity >= MAX_SEVERITY )
      severity = MAX_SEVERITY;
    memcpy(log_buffer, g_SEVERITY_PREFIX[severity], 6u);

    // Prefix the message with the TASK tag
    if ( task >= MAX_TASK )
      task = MAX_TASK;
    memcpy(&log_buffer[6], g_TASK_PREFIX[task], 0xCu);

    // Populate remainder of message with format and var_args
    vsnprintf(&log_buffer[12], 499u, format, var_args);

    // Put crafted message on to UART
    puts(log_buffer);
  }
}
```

It is important to note that all output from format string injections stemming from misuse of the `log` function will only be available to a physically present attacker who has partially disassembled the device and connected to the UART console.

This log function is misused in four locations within the `/action/wirelessConnect` HTTP handler, `web_wireless_connect`, located at offset `0x19AC94` within the `hpgw` binary included in firmware 6.9Z.

The web interface, now disabled-by-default, can be re-enabled, and the web interfaces' administrator account's password may be changed through the use of TALOS-2022-1552 and TALOS-2022-1553 in order to make these vulnerabilities accessible over the network.

CVE-2022-35884 - `ssid_hex`

The first misuse of the `log` function occurs when logging the construction of an OS command meant to configure the device's Wi-Fi AP SSID. The SSID can be provided via either the `ssid` or `ssid_hex` HTTP parameter, with `ssid_hex` taking priority if both are provided. The logic for constructing the log message when using the `ssid` parameter sufficiently sanitizes the controllable input, so we focus on the logic for handling the `ssid_hex` parameter. Below is a partial decompilation of the `web_wireless_connect` function, with annotations.

```
int __fastcall web_wireless_connect(mg_connection *conn, mg_request_info *ri)
{
  ...
  memset(ssid, 0, sizeof(ssid));
  memset(ssid_hex, 0, sizeof(ssid_hex));
  ...
  mg_get_var(payload, payload_len, "ssid", ssid, 191);

  // [1] Extract the `ssid_hex` parameter
  mg_get_var(payload, payload_len, "ssid_hex", ssid_hex, 191);
  ...
  if ( !ssid[0] )
  {
    if ( !ssid_hex[0] )
    {
      log(7u, 1u, "No SSID!");
      v6 = strtable_get("WEB_ERR_PARAM_EMPTY", 19);
      return web_error(conn, 0, v6, "ssid and ssid_hex");
    }
  }
  memset(cmd_buffer, 0, sizeof(cmd_buffer));
  ...
  if ( ssid_hex[0] )
  {
    log(7u, 0x1Fu, "with hex string");
    // [2] Craft the OS Command using user-supplied `ssid_hex` parameter
    snprintf(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 ssid %s",
"wlan0", ssid_hex);
  }
  ...

  // [3] Call `log` with the injected `cmd_buffer` variable as `format`
  log(7u, 1u, cmd_buffer);
  ...
}
```

First, at `[1]`, the `ssid_hex` parameter is extracted from the request and stored in a local variable. If the `ssid_hex` parameter has content, then at `[2]` the content of that parameter is used to construct an OS command. At `[3]`, prior to execution of that command, the command string is logged, with the `cmd_buffer` containing the attacker-controlled `ssid_hex` parameter being passed to the `log` function as its `format` parameter. This results in an attacker-controlled format string being passed to `vsprintf`.

As an example, supplying an `ssid_hex` parameter of `%x.%x.%x.%x.%x.%x.%x.%x` results in the following log being generated:

```
[DBG ][WEB ]driver/wpa_cli -i wlan0 set_network 0 ssid
1.727fd5a0.0.0.1.0.0.0.0.0.0.76697264.772f7265
```

CVE-2022-35885 - `wpapsk_hex`

This misuse of the `log` function occurs when logging the construction of an OS command meant to configure the WPAPSK of the wireless network. The password can be provided via either the `wpapsk` or `wpapsk_hex` HTTP parameters, with `wpapsk_hex` taking priority if both are provided. The logic for constructing the log message when using the `wpapsk` parameter sufficiently sanitizes the controllable input, so we focus on the logic for handling the `wpapsk_hex` parameter. Below is the relevant portion of a decompilation of the `web_wireless_connect` function, with annotations.

```
int __fastcall web_wireless_connect(mg_connection *conn, mg_request_info *ri)
{
  ...
  memset(wpapsk, 0, sizeof(wpapsk));
  memset(wpapsk_hex, 0, sizeof(wpapsk_hex));
  memset(auth_mode, 0, sizeof(auth_mode));
  ...
  mg_get_var(payload, payload_len, "auth_mode", auth_mode, 191);
  mg_get_var(payload, payload_len, "wpapsk", wpapsk, 191);

  // [1] Extract the `wpapsk_hex` parameter
  mg_get_var(payload, payload_len, "wpapsk_hex", wpapsk_hex, 191);
  ...
  // [2] Validate that `auth_mode` is WPAPSK_HEX or WPA2PSK_HEX
  if ( !strcmp(auth_mode, "WPAPSK_HEX") || !strcmp(auth_mode, "WPA2PSK_HEX") )
  {
      // [3] Craft the OS Command using user-supplied `wpapsk_hex` parameter
      snprintf(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0 psk %s",
"wlan0", wpapsk_hex);
      // [4] Call `log` with the injected `cmd_buffer` variable as `format`
      log(7u, 1u, cmd_buffer);
      ...
  }
}
```

First, at `[1]`, the `wpapsk_hex` parameter is extracted from the request and stored in a local variable. It is then confirmed at `[2]` that the `auth_mode` parameter is either `WPAPSK_HEX` or `WPA2PSK_HEX`. If so, then at `[3]` the content of that parameter is used to construct an OS command. At `[4]`, prior to execution of that command, the command string is logged, with the `cmd_buffer` containing the attacker-controlled `wpapsk_hex` parameter being passed to the `log` function as its `format` parameter. This results in an attacker-controlled format string being passed to `vsprintf`.

Supplying `%x.%x.%x.%x.%x.%x...` as the `wpapsk_hex` parameter results in the following log message being generated:

```
[DBG ][WEB ]driver/wpa_cli -i wlan0 set_network 0 psk
723fd3a0.723fd660.76b6ecf4.0.1.0.0
```

CVE-2022-35886 - `default_key_id` / `key`

This misuse of the `log` function occurs when logging the construction of an OS command meant to configure WEP key management for the wireless network. The WEP key is provided via the `key` HTTP parameter and the key identifier is provided via the `default_key_id` HTTP parameter. The WEP key management functionality is only triggered if the `auth_mode` HTTP parameter is one of `WEP` or `SHARED`. Below is the relevant portion of a decompilation of the `web_wireless_connect` function, with annotations.

```
int __fastcall web_wireless_connect(mg_connection *conn, mg_request_info *ri)
{
  ...
  memset(default_key_id, 0, sizeof(default_key_id));
  memset(key, 0, sizeof(key));
  memset(auth_mode, 0, sizeof(auth_mode));
  ...
  mg_get_var(payload, payload_len, "auth_mode", auth_mode, 191);

  // [1] Extract the `default_key_id` and `key` HTTP parameters
  mg_get_var(payload, payload_len, "key", key, 191);
  mg_get_var(payload, payload_len, "default_key_id", default_key_id, 191);
  ...

  // [2] Validate that `auth_mode` is SHARED or WEP
  if ( !strcmp(auth_mode, "SHARED") || !strcmp(auth_mode, "WEP") )
  {
    log(7u, 1u, "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");
    popen_write("driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");

    // [3] Craft the OS Command using user-supplied `default_key_id` and `key`
parameters
    snprintf_nullterm(
      cmd_buffer,
      0x7Fu,
      "driver/wpa_cli -i %s set_network 0 wep_key%s '\"%s\"'",
      "wlan0",
      default_key_id,
      key);

    // [4] Call `log` with the injected `cmd_buffer` variable as `format`
    log(7u, 1u, cmd_buffer);
  }
  ...
}
```

First, at [1], the default_key_id and key parameters are extracted from the request and stored in local variables. It is then confirmed at [2] that the auth_mode parameter is either WEP or SHARED. If so, then at [3] the content of the vulnerable parameters are used to construct an OS command. At [4], prior to execution of that command, the command string is logged, with the cmd_buffer containing the attacker-controlled parameters being passed to the log function as its format parameter. This results in an attacker-controlled format string being passed to vsprintf.

Supplying %x.%x.%x.%x.%x.%x... as both the default_key_id and key parameters will result in the following log message being generated:

```
[DBG ][WEB ]driver/wpa_cli -i wlan0 set_network 0
wep_key7237d3c0.7237d960.7237da20.6d61535f.1.0.0
'"0.22303122.4c220a2c.35657669.76697264.772f7265.635f6170.2d20696c."'
```

CVE-2022-35887 - default_key_id

The final misuse of the log function within web_wireless_connect occurs almost immediately after the previous vulnerability, when logging the construction of an OS command meant to configure WEP key management for the wireless network. The WEP key index is provided via the default_key_id HTTP parameter. The WEP key management functionality is only triggered if the auth_mode HTTP parameter is one of WEP or SHARED. Below is the relevant portion of a decompilation of the web_wireless_connect function, with annotations.

```
int __fastcall web_wireless_connect(mg_connection *conn, mg_request_info *ri)
{
  ...
  memset(default_key_id, 0, sizeof(default_key_id));
  memset(auth_mode, 0, sizeof(auth_mode));
  ...
  mg_get_var(payload, payload_len, "auth_mode", auth_mode, 191);

  // [1] Extract the `default_key_id` and `key` HTTP parameters
  mg_get_var(payload, payload_len, "default_key_id", default_key_id, 191);
  ...

  // [2] Validate that `auth_mode` is SHARED or WEP
  if ( !strcmp(auth_mode, "SHARED") || !strcmp(auth_mode, "WEP") )
  {
    ...

    memset(cmd_buffer, 0, sizeof(cmd_buffer));
    snprintf_nullterm(cmd_buffer, 0x7Fu, "driver/wpa_cli -i %s set_network 0
wep_tx_keyidx %s", "wlan0", default_key_id);
    log(7u, 1u, cmd_buffer);
  }
  ...
}
```

First, at `[1]`, the `default_key_id` parameter is extracted from the request and stored in a local variable. It is then confirmed at `[2]` that the `auth_mode` parameter is either `WEP` or `SHARED`. If so, then at `[3]` the content of the `default_key_parameter` parameter is used to construct an OS command. At `[4]`, prior to execution of that command, the command string is logged, with the `cmd_buffer` containing the attacker-controlled parameters being passed to the `log` function as its `format` parameter. This results in an attacker-controlled format string being passed to `vsprintf`.

Supplying `%x.%x.%x.%x.%x.%x...` as the `default_key_id` parameter will result in the following log message being generated:

```
[DBG ][WEB ]driver/wpa_cli -i wlan0 set_network 0 wep_tx_keyidx
7237d3aa.7237d960.7237da20.6d61535f.1.0.0
```

TIMELINE

2022-07-20 - Vendor Disclosure

2022-10-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.

VULNERABILITY REPORTS | PREVIOUS REPORT | NEXT REPORT

TALOS-2022-1584 | TALOS-2022-1600