Talos Vulnerability Report

TALOS-2022-1559

Abode Systems, Inc. iota All-In-One Security Kit XCMD testWifiAP OS command injection vulnerabilities

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-33192,CVE-2022-33195,CVE-2022-33193,CVE-2022-33194

SUMMARY

Four OS command injection vulnerabilities exist in the XCMD testWifiAP functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9X and 6.9Z. A XCMD can lead to arbitrary command execution. An attacker can send a sequence of malicious commands 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.

The `iota` device receives command and control messages (referred to in the application as XCMDs) via an XMPP connection established during the initialization of the `hpgw` application. As of version 6.9Z there are 222 XCMDs registered within the application. Each XCMD is associated with a function intended to handle it. As discussed in TALOS-2022-1552 there is a service running on UDP/55050 that allows an unauthenticated attacker access to execute these XCMDs.

An XCMD, by virtue of being commonly transmitted over XMPP, is an XML payload structured in a specific format. Each XCMD must contain a root node `<p>`, which must contain a child element, `<mac>` with an attribute `v` containing the target device MAC Address. There must also be a child element `<cmd>` which must contain an attribute `a` naming the XCMD to be executed. From there, various XCMDs require various child elements that contain information relevent only to that handler.

The `testWifiAP` XCMD is used to validate the existing wireless network configuration, and it does not expect any parameters. The XCMD appears as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<p>
  <mac v="B0:C5:CA:00:00:00"/>
  <cmds>
    <cmd a="testWifiAP"/>
  </cmds>
</p>
```

The handler associated with `testWifiAP` is located at offset `0x104830` of the `/root/hpgw` binary included in version 6.9Z, and its decompilation is included here.

```
int __fastcall testWifiAP(xml_node_t *xcmd, xstrbuf_t *response)
{
  const char *err_str;
  wifi_config_t wifi_config;

  // [1] Copy the current wireless configuration into `wifi_config`
  fetch_wifi_config(&wifi_config);

  // [2] Pass the wireless configuration into the vulnerable function
  if ( do_test_wifiap(&wifi_config) )
  {
    err_str = strtable_get("XCMD_ERR_WIFI_AUTH", 18);
    xml_construct_error_response(response, 0, 82, err_str);
  }
  else
  {
    init_xml_response(response);
  }
  return 0;
}
```

The handler itself is very straightforward. At `[1]` it collects the current wireless configuration and passes it to a delegate function we've named `do_test_wifiap`. It is the `do_test_wifiap` that contains the vulnerability, but this function is only executed when the `testWifiAP` XCMD is received.

The function we refer to as `fetch_wifi_config` is located at offset `0x1C722C`. Its decompilation is included here.

```
void __fastcall fetch_wifi_config(wifi_config_t *config)
{
  fetch_config_value("WL_SSID", config->ssid, 191);
  fetch_config_value("WL_SSID_HEX", config->ssid_hex, 191);
  fetch_config_value("WL_AuthMode", config->auth_mode, 191);
  fetch_config_value("WL_WPAPSK", config->wpapsk, 191);
  fetch_config_value("WL_WPAPSK_HEX", config->wpapsk_hex, 191);
  fetch_config_value("WL_EncrypType", config->encryp_type, 191);
  fetch_config_value("WL_DefaultKeyID", config->default_key_id, 191);
  fetch_config_value("WL_Key", config->key, 191);
}
```

These WiFi configuration values may be modified a few ways: by the user via mobile application or abode web application; through the `setWifiAP` XCMD; and via either the `/action/wirelessPost` or `/action/configPost` endpoints of the device's local web interface. None of these mechanisms implement useful sanitization or sanity checking on the values. For the purposes of the following vulnerabilities, we assume that the remote attacker has manipulated these parameters prior to triggering the vulnerable `testWifiAP` XCMD.

These configuration values are passed to the function we've named `do_test_wifiap` which is located at offset `0x1c7d28`. The function is reminiscent of the vulnerable `web_wireless_connect` function discussed in TALOS-2022-1556. The relevant portions of the decompilation are included below.

```
int __fastcall do_test_wifiap(wifi_config_t *config)
{
  int result;
  const char *static_command;
  int retry;
  int con_suc;
  const char *str_err;
  int wireless_enabled;
  char cmd_output[32];
  char command[256];

  wireless_enabled = 0;

  // [1] Ensure that wireless is enabled, otherwise exit early
  get_config_as_integer("WL_Enable", (int)&wireless_enabled);
  if ( !wireless_enabled )
    return -3;

  // [2] Ensure that one of `config->ssid` or `config->ssid_hex` is provided
  if ( !config->ssid[0] && !config->ssid_hex[0] )
  {
    log(7, 31, "No SSID!");
    return -2;
  }

  ...

  memset(command, 0, 0x80u);
  // [3] Identify whether the SSID is provided via the `config->ssid` or `config-
>ssid_hex` configuration value
  if ( config->ssid_hex[0] )
  {
    // [4] If via `config->ssid_hex`, inject directly into the OS command
    log(7, 31, "with hex string");
    vsnprintf_nullterm(command, 0x7Fu, "driver/wpa_cli -i %s set_network 0 ssid %s",
"wlan0", config->ssid_hex);
  }
  else
  {
    // [5] If via `config->ssid`, inject directly into the OS command
    log(7, 31, "with acii string");
    vsnprintf_nullterm(command, 0x7Fu, "driver/wpa_cli -i %s set_network 0 ssid
'\"%s\"'", "wlan0", config);
  }
  log(7, 1, command);

  // [6] Execute the command constructed at [4] or [5]
  popen_write(command);


  memset(command, 0, 0x80u);
  if ( strcmp(config->auth_mode, "WPA") && strcmp(config->auth_mode, "WPA2") )
  {

    // [7] If `config->auth_mode` is WPAPSK or WPA2PSK
    if ( !strcmp(config->auth_mode, "WPAPSK") || !strcmp(config->auth_mode,
"WPA2PSK") )
    {
```

```
      // [8] then inject `config->wpapsk` directly into the OS command
      vsnprintf_nullterm(command, 0x7Fu, "driver/wpa_cli -i %s set_network 0 psk
'\"%s\"'", "wlan0", config->wpapsk);
      log(7, 1, command);
      p_command = command
    }

    // [9] Otherwise, if `config->auth_mode` is SHARED or WEP
    else if ( !strcmp(config->auth_mode, "SHARED") || !strcmp(config->auth_mode,
"WEP") )
    {
      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");

      // [10] Construct an OS command by injecting `config->default_key_id` and
`config->key`
      vsnprintf_nullterm(
        command,
        0x7Fu,
        "driver/wpa_cli -i %s set_network 0 wep_key%s '\"%s\"'",
        "wlan0",
        config->default_key_id,
        config->key);
      log(7, 1, command);

      // [11] Execute the command constructed at [10]
      popen_write(command);
      memset(command, 0, 0x80u);

      // [12] Then construct a second OS command by injecting `config-
>default_key_id` directly
      vsnprintf_nullterm(
        command,
        0x7Fu,
        "driver/wpa_cli -i %s set_network 0 wep_tx_keyidx %s",
        "wlan0",
        config->default_key_id);
      log(7, 1, command);

      // [13] Execute the command constructed at [12]
      popen_write(command);

      if ( strcmp(config->auth_mode, "SHARED") )
        goto LABEL_19;
      log(7, 1, "driver/wpa_cli -i wlan0 set_network 0 auth_alg SHARED");
      p_command = "driver/wpa_cli -i wlan0 set_network 0 auth_alg SHARED";
    }
    else
    {
      log(7, 1, "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE");
      p_command = "driver/wpa_cli -i wlan0 set_network 0 key_mgmt NONE";
    }

    // [14] Execute the command constructed at [8]
    popen_write(p_command);
  }
  ...
}
```

CVE-2022-33192 - OS Command Injection via `WL_SSID` and `WL_SSID_HEX` Configurations

The conditional referenced at `[3]` is responsible for determining which of the two configuration values to inject into the OS command. Either value is equally exploitable, but `WL_SSID_HEX` will be prioritized if it has been set. At `[4]`, the selected configuration value is injected, without neutralization or sanity checking, into an OS command which is executed via `popen` at `[5]`.

If either of `WL_SSID_HEX` or `WL_SSID` have been maliciously formatted prior to the receipt of a `testWifiAP` XCMD, then receipt of the `testWifiAP` XCMD and subsequent execution of the vulnerable `do_test_wifi` function will result in arbitrary command execution with root privileges.

CVE-2022-33193 - OS Command Injection via `WL_WPAPSK` Configuration

The conditional referenced at `[7]` is responsible for determining whether the `WL_AuthMode` requires the use of the `WL_WPAPSK` configuration value. Note that the use of `WL_WPAPSK_HEX` is never used in this function. If the `WL_AuthMode` is either “WPAPSK” or “WPAP2PSK” then at `[8]` the configuration value is injected, without neutralization or sanity checking, into an OS Command which is executed via `popen` at `[14]`.

If `WL_WPAPSK` has been maliciously formatted prior to the receipt of a `testWifiAP` XCMD, then receipt of the `testWifiAP` XCMD and subsequent execution of the vulnerable `do_test_wifi` function will result in arbitrary command execution with root privileges.

CVE-2022-33194 - OS Command Injection via `WL_Key` and `WL_DefaultKeyID` Configurations

The conditional referenced at `[9]` will be entered if the `WL_AuthMode` is set to “SHARED” or “WEP”. If that is the case, then the values from `WL_Key` and `WL_DefaultKeyID` will be injected at `[10]`, without neutralization or sanity checking, into an OS Command which is executed via `popen` at `[11]`.

If either of `WL_Key` or `WL_DefaultKeyID` have been maliciously formatted prior to the receipt of a `testWifiAP` XCMD, then receipt of the `testWifiAP` XCMD and subsequent execution of the vulnerable `do_test_wifi` function will result in arbitrary command execution with root privileges.

CVE-2022-33195 - OS Command Injection via `WL_DefaultKeyID` Configuration

The conditional referenced at `[9]` will be entered if the `WL_AuthMode` is set to “SHARED” or “WEP”. If that is the case, then the `WL_DefaultKeyID` value will be injected at `[12]`, without neutralization or sanity checking, into an OS Command which is executed via `popen` at `[13]`.

If `WL_DefaultKeyID` has been maliciously formatted prior to the receipt of a `testWifiAP` XCMD, then receipt of the `testWifiAP` XCMD and subsequent execution of the vulnerable `do_test_wifi` function will result in arbitrary command execution with root privileges.

TIMELINE

2022-07-13 - Initial Vendor Contact

2022-07-14 - Vendor Disclosure

2022-10-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1558

NEXT REPORT

TALOS-2022-1560