Talos Vulnerability Report

TALOS-2022-1584

Abode Systems, Inc. iota All-In-One Security Kit ghome_process_control_packet format string injection vulnerability

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-33938

SUMMARY

A format string injection vulnerability exists in the ghome_process_control_packet functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9Z and 6.9X. A specially-crafted XCMD can lead to memory corruption, information disclosure and denial of service. An attacker can send a malicious 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 6.9X
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 vulnerable instance of the `log` function occurs within the `ghome_process_control_packet` function located at offset `0x1D27B0` in the `hpgw` binary included in firmware version 6.9Z. An annotated decompilation of this function is included, for reference, below.

```
int __fastcall ghome_process_control_packet(
        ghome_struct_t ghome_packet,
        const struct sockaddr addr,
        socklen_t addr_len)
{
  char *payload;
  int sock_fd;
  char *cmd_offset;
  int result;
  char *host;
  const char *v8;
  const char *v9;
  xstrbuf_t xcmd_result;
  char cmd[64];
  char id[64];
  char log_msg[280];

  payload = ghome_packet.payload;
  ghome_packet.payload[ghome_packet.payload_len] = 0;
  sock_fd = ghome_packet.sock_fd;
  log(7u, 0x10u, "GHome recv ctrl len=%d\n%s", ghome_packet.payload_len,
ghome_packet.payload);
  cmd_offset = strstr(payload, "<cmd");
  memset(cmd, 0, sizeof(cmd));
  memset(id, 0, sizeof(id));
  result = (int)cmd_offset;
  if ( cmd_offset )
  {
    // [1] Extract the `a=` and `id=` fields from the GHOME packet into the `cmd`
and `id` variables
    xml_fetch_value(cmd_offset, "a=", cmd);
    xml_fetch_value(cmd_offset, "id=", id);
    if ( !strcmp(id, g_PREV_ID) )
    {
      return 1;
    }
    else
    {
      strncpy_0(g_PREV_ID, id, 63);
      log(7u, 0x10u, "\x1B[33mGHome cmd=%s, id=%s\x1B[0m", cmd, id);
      memset(log_msg, 0, 0x100u);
      xstrbuf(&xcmd_result);
      xcmd_run(1, payload, 0, &xcmd_result);
      host = inet_ntoa(*(struct in_addr *)&addr.sa_data[2]);

      // [2] Construct a log message using the attacker-supplied `cmd` and `id`
variables
      snprintf(
        log_msg,
        0xFFu,
        "reply '%s' id=%s to %s:%d",
        cmd,
        id,
        host,
        (unsigned __int16)(*(_WORD *)addr.sa_data << 8) | (unsigned __int16)
(unsigned __int8)addr.sa_data[1]);

      // [3] Make the vulnerable call to `log` passing the `log_msg` variable as the
```

```
`format` parameter
      log(7u, 0x10u, log_msg);
      ...
```

This function is called on every UDP payload received on port 55050. The function expects that the payload will be an XML formatted payload, though it is not particularly strict in its parsing. At `[1]` the function parses out the `a` and `id` fields of the `cmd` tag and uses them at `[2]` to craft a log message. After the attacker-supplied values have been injected into the log message buffer, the buffer is used as the `format` parameter of the `log` function. Supplying a `cmd` tag with the `a` and `id` attributes set to `%x.%x.%x.%x.%x.%x.%x.%x` will result in the following log message:

```
[DBG ][FIND]reply '714be54c.714be4b8.714bf958.e6ab.0.0.482a40.200'
id=102.252e7825.78252e78.2e78252e.252e7825.78252e78.78252e.0.0.0 to 10.1.1.207:59051
```

TIMELINE

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

CREDIT

Discovered by Matt Wiseman of Cisco Talos.

© 2022 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our Privacy Policy.