Talos Vulnerability Report

TALOS-2022-1557

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

OCTOBER 20, 2022

CVE NUMBER

CVE-2022-30541

SUMMARY

An OS command injection vulnerability exists in the XCMD setUPnP functionality of Abode Systems, Inc. iota All-In-One Security Kit 6.9X and 6.9Z. A specially-crafted XCMD can lead to arbitrary command execution. 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

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.

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 then signaling 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 the configuration (e.g. if `webport` is not provided, the system will use the current value `WebPort` configuration key). One might note that the port being forwarded need not actually be related to the web interface.

A sample of this XCMD could 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 set the various configuration values and then signal 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 that appears largely superfluous for code responsible for configuring UPnP data.

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

```
int __fastcall setUPnP(xml_related_t *xml, int *a2)
{
  char *webextport;
  char command[256];
  ...
  // [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 `webextport` 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] Construct an OS command with the value of `webextport`
  vsnprintf_nullterm(command, 0xFFu, "echo %s > /var/in", webextport);

  // [4] Execute the OS command
  popen_write(command);
  init_xml_response(a2);
  return 0;
}
```

For 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 injected directly into an OS command. Finally, at `[4]`, the command is executed.

Submitting an appropriately formatted XCMD to this handler will result in arbitrary command execution.

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=":test; sleep 11 #"/>
    </cmd>
  </cmds>
</p>
```

will result in the execution of the command:

```
echo :test; sleep 11 # > /var/in
```

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.
