Andrey Lovyannikov Follow

Oct 16, 2020 · 9 min read · Listen

Groundhog Day in IoT Valley, or 5 CVEs in 1 Camera

CS-C2SHW

We at BI.ZONE have an IoT cabinet stuffed full of all sorts of devices. Any researcher can grab a box and search for vulnerabilities at their leisure.

I came across an IP camera CS-C2SHW with its Flash carefully dumped by the previous researcher. This proved to be a great way to while away several days!

Disclaimer: I didn't carry out a full-scale analysis of the device, just studied what was lying on the surface for the sake of interest. That's why my description does not in any way claim to be absolutely complete.

What's Inside

To start with, the traffic and Flash dump analysis identified, at least, the following services:

Service name	Port	Protocol	Enabled by Default	Executable File
discovery	44444	UDP	Yes, and can't be disabled	AgentGreen
RTSP	554	UDP, RTSP	Supposedly not Presumably	AgentGreen
control	10000	TCP, protobuf?	Yes, but runs on localhost	vc-hal
http-api	8080	HTTP	Yes, but runs on localhost	AgentUpdater

As seen from the table, there aren't that many services, and most of them are not accessible from the outside. I didn't explore all of them — only the two that kindled my interest.

The first to seize my attention was a proprietary service on the UDP-port 44444, but the service was found to have nothing of interest. Its only function is to discover the camera on the network (hence the name — `discovery`), it doesn't possess any additional capabilities. *This service had a DoS vulnerability, we'll talk about it later.*

With a limited functionality of the `discovery` service and no other suitable candidates, how then is the camera configured? As it turned out, in a genuinely peculiar way: a special QR code has to be generated and shown to the camera. The QR code contains text, which defines the device's configuration. *Spoiler: a bash injection vulnerability was detected in the QR code-based configuration, I will give more details below.*

Further, I analyzed two vectors of interaction with the camera and detected two vulnerabilities: a `discovery` service DoS and a bash injection through the QR code. Both have certain restrictions, but RCE is something I was really after. This feeling of incompleteness prompted me to explore two other mechanisms: Internet availability check and updates.

To check the Internet availability, an HTTP request is sent to the URL set in the configuration file. Analyzing this mechanism led me to a proprietary parser and an HTTP request builder — *and in the parser, I discovered a heap overflow (more details are provided below).* But again with serious restrictions, which affect the "value" of the vulnerability.

The final thing up for analysis was the update mechanism. And that's where everything appeared to be very promising. Updates can be installed via the network (using the URLs set in the HTTPs configuration) or locally, via an SD card. When updates are carried out via the SD card, `AgentUpdater` sends respective protobuf requests to the main service (aka `control` or `vc-hal` file), which conducts a shallow analysis of the transmitted arguments with part of them placed to the command string. As you may have guessed, part of the arguments are also read from the SD card and some are placed to the command string without escaping *(hello, bash injection).*

Now let's delve into the details.

`discovery` Service Vulnerability

The `discovery` service (in `AgentGreen`) is implemented as follows:

1. A set of BPF rules is applied to the inbound traffic.
2. The inbound packet undergoes additional checks and is copied to the local buffer.
3. The UDP is checked for the correct JSON request.

BPF rules:

- MAC address equals `ff:ff:ff:ff:ff:ff` (or any other configured value)
- IP protocol number equals 0x11 (UDP).
- UDP port equals the value set in the config file (44444).
- IP total length is less than 0x21C bytes.

Checks after reading in the local buffer:

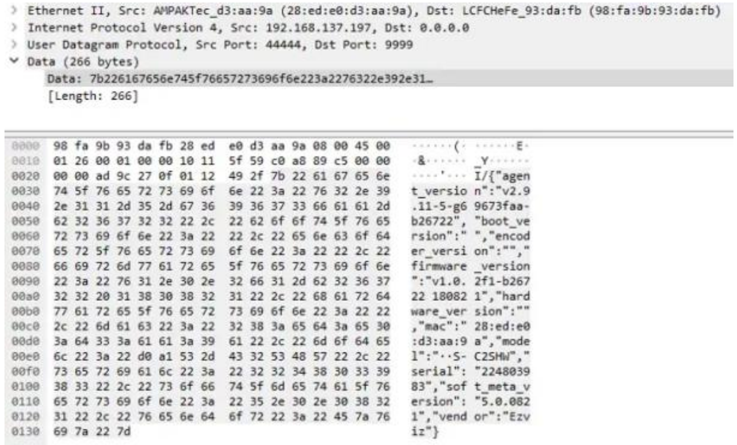
- The inbound buffer length is more or equal to 4 bytes.
- IP protocol number is equal to 0x11 (UDP).
- IP header checksum is correct.

Checks before sending a response:

- UDP contains a correct JSON.
- JSON has `version` and `action` fields.
- The `action` field value equals `discovery`.

The script below sends a request for camera detection, and a response to the request.

send camera discovery request



```
> Ethernet II, Src: AMPAKTec_d3:aa:9a (28:ed:e0:d3:aa:9a), Dst: LCFCHeFe_93:da:fb (98:fa:9b:93:da:fb)
> Internet Protocol Version 4, Src: 192.168.137.197, Dst: 0.0.0.0
> User Datagram Protocol, Src Port: 44444, Dst Port: 9999
v Data (266 bytes)
    Data: 7b226167656e745f76657273696f6e223a2276322e392e31…
    [Length: 266]

0000  98 fa 9b 93 da fb 28 ed  e0 d3 aa 9a 08 00 45 00   ······(· ······E·
0010  01 26 00 01 00 00 10 11  5f 59 c0 a8 89 c5 00 00   ·&······ _Y······
0020  00 00 ad 9c 27 0f 01 12  49 2f 7b 22 61 67 65 6e   ····'··· I/{"agen
0030  74 5f 76 65 72 73 69 6f  6e 22 3a 22 76 32 2e 39   t_versio n":"v2.9
0040  2e 31 31 2d 35 2d 67 36  39 36 37 33 66 61 61 2d   .11-5-g6 9673faa-
0050  62 32 36 37 32 32 22 2c  22 62 6f 6f 74 5f 76 65   b26722", "boot_ve
0060  72 73 69 6f 6e 22 3a 22  22 2c 22 65 6e 63 6f 64   rsion":" ","encod
0070  65 72 5f 76 65 72 73 69  6f 6e 22 3a 22 22 2c 22   er_versi on":"","
0080  66 69 72 6d 77 61 72 65  5f 76 65 72 73 69 6f 6e   firmware _version
0090  22 3a 22 76 31 2e 30 2e  32 66 31 2d 62 32 36 37   ":"v1.0. 2f1-b267
00a0  32 32 20 31 38 30 38 32  31 22 2c 22 68 61 72 64   22 18082 1","hard
00b0  77 61 72 65 5f 76 65 72  73 69 6f 6e 22 3a 22 22   ware_ver sion":""
00c0  2c 22 6d 61 63 22 3a 22  32 38 3a 65 64 3a 65 30   ,"mac":" 28:ed:e0
00d0  3a 64 33 3a 61 61 3a 39  61 22 2c 22 6d 6f 64 65   :d3:aa:9 a","mode
00e0  6c 22 3a 22 d0 a1 53 2d  43 32 53 48 57 22 2c 22   l":"··S- C2SHW","
00f0  73 65 72 69 61 6c 22 3a  22 32 32 34 38 30 33 39   serial": "2248039
0100  38 33 22 2c 22 73 6f 66  74 5f 6d 65 74 61 5f 76   83","sof t_meta_v
0110  65 72 73 69 6f 6e 22 3a  22 35 2e 30 2e 30 38 32   ersion": "5.0.082
0120  31 22 2c 22 76 65 6e 64  6f 72 22 3a 22 45 7a 76   1","vend or":"Ezv
0130  69 7a 22 7d                                        iz"}
```

discovery response in Wireshark

Since there're no checks of the source IP address, we can force the device to send a response to a random IP address. However, we can't control the content at all — hence, this doesn't give us much benefit (even DDoS is hardly realistic, as you need way too many such cameras; besides, it would be extremely easy to filter out such traffic by content).

But the code has another error: the size of incoming data is not duly verified (checking that it's more than or equal to 4 bytes is insufficient). When initializing `std::string`, which contains the UDP payload, we see the following:

discovery request receiving and processing

Thus, with a buffer size of less than 42 bytes, we have an attempt to initialize a string with a negative size. And this attempt, obviously, leads to an exception being generated and the agent's termination. After a short while, the agent reboots.

Accordingly, by exploiting this vulnerability, we can cause **the agent's DoS on all the cameras in the local area network.**

Overall, it's not very useful, but there's a couple of nuances. First, the agent might also control the video recording start/stop (as scheduled). Then, by exploiting this vulnerability, we can prevent the camera from recording. Second, there's a probability that the agent is also responsible for video streaming to the vendor's servers. This supposition rests on nothing more, but the functions' names. But if it's correct, then the video transmission from all the cameras in the network can be terminated remotely. Even if one of these speculations is true, this makes the vulnerability increasingly useful.

Configuring via QR Code

As we have mentioned above, the device's configuration is QR-encoded. This is how network configuration is set and the device is linked to the user's account.

Configuration in the QR code is represented as ASCII strings, where the first string denotes the configuration type:

- Wi-Fi configuration (magic: `VCFRT`) — at least 4 lines, 7 for PPPoE configuration;
- Ethernet configuration (magic: `VCFRTE`).

Wi-Fi configuration lines:

1. Config magic (`VCFRT`)
2. Wi-Fi SSID
3. Wi-Fi password
4. Agent registration hash (device linking to the user)
5. Type of environment (`prod` or `edg`)
6. `PPPoE` value, which means the start of PPPoE settings (optional)
7. PPPoE config value 1 (optional)
8. PPPoE config value 2 (optional)

Ethernet configuration lines:

1. Config magic (`VCFRTE`)
2. Designation of the config type: DHCP (`dhcp`) or static IP address (any other value)
3. IP address
4. Netmask
5. Gateway
6. DNS
7. Agent registration hash (device linking to the user)
8. Type of environment (`prod` or `edg`)

QR code sample (Wi-Fi configuration):

```
VCMFRT
my_sweet_home
1337!telecom
agent_registration_hash_here
prod
```

Wi-Fi settings are written to the WPA supplicant config file.

PPPoE settings are written to the file `/config/pppoe` (set by the parameter `pppoe-settings-file` of the service config file `vc-hal`).

Ethernet configuration for a static IP address is written to the file `/config/ip-static` (set by the parameter `static-ip-settings-file` of the service config file `vc-hal`), and the returned value indicates that the device must be rebooted. The last execution during a reboot is the initialization script `/etc/init.d/S99custom`, which calls the following function:

This function reads the file `/config/ip-static` line-by-line, selects the interface name, terminates all DHCP services and applies the new configuration. From the comments, you can see that strings set via the QR code are read in points A, whereas these strings are inserted to the bash command in points B. Thus, we get a **bash injection**.

The downsides of this vulnerability:

- QR code scanning will only be performed until `agent registration hash` is identified, which links the device to the account. In other words, this vulnerability can only be exploited to attack non-configured devices.
- We can do almost nothing with such type of injection. I haven't found any interesting options of `ifconfig` or `route` to get arbitrary command execution.

HTTP Response Processing Vulnerability

As you may remember, `AgentUpdater` was found to have a self-written parser and an HTTP request builder. Of course, the parser had a vulnerability.

Analysis of HTTP requests creation and parsing algorithms showed that an HTTP response can be parsed fully or partially (with only the headers taken and the request body discarded). The full parsing of an HTTP request may result in **a heap buffer overflow.**

The vulnerable function code is presented below.

As seen from the snippet, the first iteration of the `while (true)` cycle allocates a buffer `req->pld_buf` . The buffer size is equal to the header value `Content-Lenght+1` , unless it equals zero.

Further, for the first iteration (`pld_buf_off==0`), part of the data from the request body is copied to the allocated buffer. The size is represented as `pld_len` , which is equal to the difference between the size of the output buffer (including the header) and the offset where the data is kept.

Fortunately, the function is only executed on responses from the URLs indicated in the config files (i.e. they are strictly set by the vendor). And all of the URLs are HTTPS, and it looks like certificate pinning is used (but that's not entirely certain). Among the said URLs, there's only one HTTP, a ping analog, but during such HTTP request the response is not fully analyzed and the vulnerable code can't be reached.

Thus, the processing of HTTP requests carries a potential RCE vulnerability, but to exploit it, a threat actor would have to wait until the config files are modified (e.g. following the update) or find another vulnerability in the used mbedTLS library. However, an outdated vulnerable version of this library might well be used: I haven't checked.

SD Card Update Mechanism Vulnerability

As mentioned above, the agents send requests for various actions to the `control` service, with the required parameters transmitted in the same request. The `control` service is written in such a way as almost every action is associated with the execution of a certain sh- command. It's obvious that in this case parameters must be transmitted as command string arguments and escaped.

Given that a `system` function is used to execute sh-commands, untrusted input validation is critical. The developers hadn't given this threat due consideration and didn't use any escaping at all.

While this is very bad indeed, it still requires the right opportunity to implement the threat:

- first, to send a request for a vulnerable action
- second, to set an arbitrary (or at least partially controlled) parameter value

Let's consider the device's most obvious communication channels:

- **Network interaction.** The device interacts with the vendor's servers via HTTPS, and we can't control this interaction.
- **QR code.** Successful QR code scanning results in requests being generated to `control`, and a bash injection is even possible via Ethernet's static configuration (see above).
- **SD card.** Though being not the most obvious option, the inserted SD card must generate some event (and, generally, not just a single one).
  And that's where we discovered quite a useful "feature": the device can be updated through an SD card.

Here is a step-by-step of how the device can be updated via an SD card:

1. check that file `vc\version.json` exists
2. check that file `vc\version.json` is a correct JSON file
3. extract from file `vc\version.json` the fields `platform`, `fw-version`, `fw-url` and `fw-sign`
4. check that the `platform` value is equal to the expected value (`hi3518ev200`)
5. check that the `fw-version` value is different from the current version (i.e**. it's possible to roll back to the previous, vulnerable versions**)
6. check that the `fw-url` value is represented as a file on the SD card, which is available for reading
7. start updates with the command bellow (**bash injection through the parameter `fw-sign`**)
8. <some other steps>
9. rename file `vc\version.json` to `vc\version.json.ok` in case of success and to `vc\version.json.fail` in case of an error.

```
fwupgrade_binary "file://<checked fw-url>" "fw-sign"
```

Thus, we can compile such a `vc\version.json`, where `fw-sign` contains a bash code for the exploitation of the above vulnerability. Here's an example of such script:

vulnerable vc\version.json

This code will save the network configuration to the SD card, add a new user with root privileges and launch an SSH server. A threat actor can thereby establish an SSH connection with the camera from the local network with root privileges and randomly affect the processed data (and, possibly, attack other devices).

To sum up: this is an extremely useful vulnerability for backdoor implementation, the functionality of which is only limited by the attacker's imagination.

Instead of Conclusion

Nowadays there are too many IoT devices and only few of them are properly secured. There is a chance that after this post we'll have one more a little bit more secure IoT device and one more security aware vendor (a tiny chance in my personal opinion).

The table below shows a full list of vulnerabilities.

P.S. I contacted Hikvision (why? just compare this camera with EZVIZ C2C (Mini O) by design and name, and try to set the *grep -i ezviz -R ** against the firmware dump) and they said that firmware for CS-C2SHW has been developed by Rostelekom and I should report bugs to them. Anyway I don't think there is a big difference between Rostelecom and EZVIZ firmwares, so it could be a good idea to check EZVIZ C2C cameras for the similar vulnerabilities.

Cve Iot Security Cameras