



JUNE 18, 2018 BY OR PELES

## VDOO Discovers Significant Vulnerabilities in Axis Cameras

For the past several months, VDOO's security research teams have been undertaking broad-scale security research of leading IoT products, from the fields of safety and security. In most cases, the research was carried out together with the device vendors for the sake of efficiency and transparency.

As part of this research, VDOO researchers found zero-day vulnerabilities in devices of several vendors. These vulnerabilities were disclosed to the vendors, in accordance with responsible disclosure best practices, and will be shared gradually after the disclosure periods are concluded.

One of the vendors for which we found vulnerable devices was Axis Communications. Our team discovered a critical chain of vulnerabilities in Axis security cameras. The

vulnerabilities allow an adversary that obtained the camera's IP address to remotely take over the cameras (via LAN or internet). In total, VDOO has responsibly disclosed seven vulnerabilities to Axis security team.

The vulnerabilities' IDs in Mitre

are: [CVE-2018-10658](#), [CVE-2018-10659](#), [CVE-2018-10660](#), [CVE-2018-10661](#), [CVE-2018-10662](#), [CVE-2018-10663](#) and [CVE-2018-10664](#).

Chaining three of the reported vulnerabilities together, allows an unauthenticated remote attacker that has access to the camera login page through the network (without any previous access to the camera or credentials to the camera) to fully control the affected camera. An attacker with such control could do the following:

- Access to camera's video stream
- Freeze the camera's video stream
- Control the camera – move the lens to a desired point, turn motion detection on/off
- Add the camera to a botnet
- Alter the camera's software
- Use the camera as an infiltration point for network (performing lateral movement)
- Render the camera useless
- Use the camera to perform other nefarious tasks (DDoS attacks, Bitcoin mining, others)

The vulnerable products include 390 models of Axis IP Cameras. The full list of affected products can be found [here](#). Axis uses the ACV-128401 identifier for relating to the issues we discovered.

To the best of our knowledge, **these vulnerabilities were not exploited in the field**, and therefore, did not lead to any concrete privacy violation or security threat to Axis's customers.

We strongly recommend Axis customers who did not update their camera's firmware to do so immediately or mitigate the risks in alternative ways. See instructions in [FAQ](#) section below.

We also recommend that other camera vendors follow [our recommendations](#) at the end of this report to avoid and mitigate similar threats.

## About VDOO

VDOO is a technology driven company that strives to change the reality of unprotected connected devices. VDOO is building a line of products to support device manufacturers in embedding security into their connected devices at the development stage and enable post-development security.

In addition to developing products and services, VDOO invests significant efforts in a wide scope research of connected devices. Security cameras is one focus area of this research.

VDOO research goal is to contribute knowledge and tools to mitigate risks, as well as encourage the devices' manufacturers to implement the right security for their products. We at VDOO believe that an appropriate implementation of the security essentials will dramatically decrease the chances of exploiting vulnerabilities on the device.

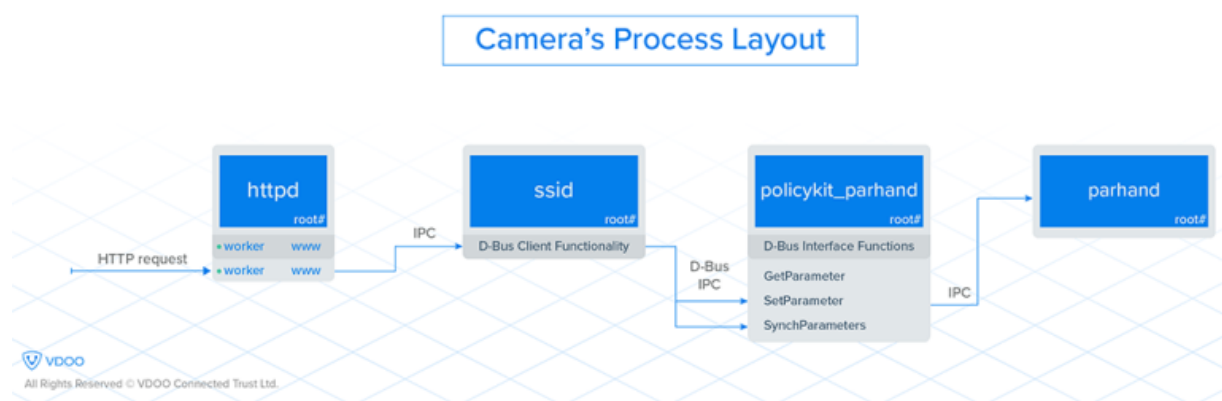
This is the second report from our series of researches focused on video surveillance equipment.

The first report, focusing on Foscam equipment, is available [here](#). For more details about our research approach, click [here](#).

## Technical Overview

The camera runs a Linux operating system and the camera's web-interface is based on the **Apache httpd** web server with proprietary modules developed by Axis. Access to files in the web-server's root directory is governed by Axis's custom authorization

code inside the **mod\_authz\_axisgroupfile.so** module. By using a proprietary module, **mod\_trax.so**, the web-server forwards certain requests to be handled by other processes using special directives (such as **TransferMime**) in Apache configuration files. For example, requests for files ending with **.shtm**, **.shtml** or **.srv** extension are forwarded to the **/bin/ssid** process. The **ssid** process runs as root, serving different functionalities for **.srv** requests than for **.shtm** or **.shtml**. Requests for a **.srv** file are only allowed for privileged users. Some of the system's daemons communicate by using the dbus Inter-Process Communication mechanism. Additionally, the camera has a proprietary system for managing internal parameters. The **/bin/parhand** process (parameter handler) is responsible for storing, fetching and updating parameters and their values. For example, when a user sets a parameter through the web interface, the relevant CGI script (**param.cgi**) forwards the set-parameter request to the **parhand** process, which verifies access-rights, and stores the parameter's value in the relevant configuration file. Some of the parameters (those that are "Shell-mounted") end up in configuration files in shell variable assignment format (e.g., **FOO=Bar**), that are later imported (executed) in some services' startup-scripts. Another process of interest is the **/usr/sbin/policykit\_parhand**, which offers the PolicyKitParhand dbus-interface that also includes functions for setting the values of **parhand**-parameters.



By exploiting three of the seven newly discovered vulnerabilities in a specific sequence, an attacker with network access to the camera can remotely execute shell commands with root privileges.

The attack sequence is as follows:

- **Step 1:** The attacker uses an **authorization bypass vulnerability (CVE-2018-10661)**. This vulnerability allows the attacker the ability to send unauthenticated HTTP requests that reach the **.srv** functionality (that handles **.srv** requests) inside **/bin/ssid**. Normally, this functionality should only be accessible to administrative users.
- **Step 2:** The attacker then utilizes an **interface that allows sending any dbus message** to the device's bus, without restriction (**CVE-2018-10662**), that is reachable from **/bin/ssid's .srv**. Due to the fact that **/bin/ssid** runs as root, these dbus messages are authorized to invoke most of the system's dbus-services' interfaces (that were otherwise subject to a strict authorization policy). The attacker chooses to send dbus messages to one such dbus-service's interface – **PolicyKitParhand**, which offers functions for setting **parhand** parameters. The attacker now has control over any of the device's **parhand** parameter values. (See the next vulnerability).
- **Step 3:** A shell command injection vulnerability (**CVE-2018-10660**) is then exploited. Some **parhand** parameters (of type "Shell-Mounted") end up in configuration files in shell variable assignment format, which are later, included in a service's init-script that runs as root. Due to step-2, the attacker is able to send unauthenticated requests to set **parhand** parameter values. By doing so, the attacker can now exploit this vulnerability by setting one parameter's value with special characters which will cause command injection, in order to execute commands as the root user.

## Technical Deep-Dive

This section provides details for each of the vulnerabilities used in the full attack sequence.

### **CVE-2018-10661 – Authorization bypass vulnerability**

This vulnerability allows an attacker to bypass the web-server's authorization mechanism by sending unauthenticated requests that reach the `/bin/ssid's .srv` functionality; no user credentials are required.

This vulnerability resides in `mod_authz_axisgroupfile.so`: a custom authorization module for Apache `httpd` that was written by the vendor.

As previously stated, the device runs an Apache `httpd` server and requests to paths inside the document-root folder must be granted by `mod_authz_axisgroupfile.so` authorization module in order to proceed.

The only `.srv` file inside the document-root folder is `/sm/sm.srv` (relative path), and the authorization code verifies that the authenticated user has sufficient privileges in order to pass. Upon granting authorization, the web-server is configured to handle requests with a specific handler to paths ending with the `.srv` extension (the '`.srv handler code`').

To summarize the problem, requests to a world-readable file that are followed by a backslash and end with the `.srv` extension (e.g. [http://CAMERA\\_IP/index.html/a.srv](http://CAMERA_IP/index.html/a.srv)) are treated by the authorization code as standard requests to the `index.html` and thus granted access, while the requests are *also* treated as legitimate requests to an `.srv` path, and are thus handled by the `.srv` handler, simultaneously.

This happens due to a feature of the web-servers that deals with trailing pathname strings that follow an actual filename, called `PATH_INFO` (see <https://tools.ietf.org/html/rfc3875#section-4.1.5>)

The following (abstracted) logic occurs on receipt of an HTTP request to `http://CAMERA_IP/index.html/a.srv`:

1. When Apache `httpd` parses the request URI, it sets the following member fields in the `request's request_rec` struct:
  - `uri` = `"/index.html/a.srv"`
  - `filename` = `"/usr/html/index.html"` # Assuming the server's Document Root is `"/usr/html"`

- **path\_info** = “/a.srv”

2. Access to files in the Document Root directory is governed by Axis’s custom authorization code by **so** module (due to the **Require axis-group-file** directive in the **/usr/html** Directory directive in the **httpd** configuration file).

The custom authorization code performs authorization checks based on the **request.filename** only (see IDA screenshot with explanation below), ignoring the existence of the **path\_info** feature, and thus **grants access** to our request to **/index.html/a.srv**, because the request is perceived as intended for the **/usr/html/index.html** file that is world-readable (and does not require any authentication).

3. Now that the request is authorized, the configuration’s `<LocationMatch “.+\.(shtml|shtmlsrv)(|$&)”>` directive, matches its regex pattern against **uri** (our request’s full uri, see above), and, because it ends with **.srv**, the regular expression matches and it proceeds to execute the **.srv handler code**: – “TransferMime /var/run/ssid/ssidsocket” – which transfers the request to the **/var/run/ssid/ssidsocket** unix socket for handling by the **/bin/ssid** process.
4. Later, the **/bin/ssid** process receives the request, checks its (full) URI, and treats the request as a legitimate request to an **.srv** file – allowing the request to reach the **.srv** functionality.

```

; Authorization provider for paths subject
; to 'Require axis-group-file' directive

axis_group_file_provider
STMFD    SPI, {R4-R6,LR} ; Store Block to Memory
MOV      R4, R0 ; Rd = Op2
LDR      R1, [R0,#request_rec.filename] ; request.filename
LDR      R5, =( _GLOBAL_OFFSET_TABLE_ - 0x26D8) ; Load from Memory
LDR      R0, [R0] ; Load from Memory
BL       is_file_world_readable ; Branch with Link
CMP      R0, #1 ; request.filename is world-readable
ADD      R5, PC, R5 ; _GLOBAL_OFFSET_TABLE_
MOVEQ   R1, #0 ; if world-readable: group_name = NULL
BEQ      loc_270C ; Branch

LDR      R2, =(a127001+8 - 0x26F0) ; Load from Memory
LDR      R1, =(aHandleRedirect - 0x26F4) ; Load from Memory
LDR      R0, [R4,#0xA8] ; Load from Memory
ADD      R2, PC, R2 ; "1"
ADD      R1, PC, R1 ; "HANDLE_REDIRECT"
BL       apr_table_setn ; Branch with Link
LDR      R3, =(authz_owner_get_file_group_ptr - 0x13F04) ; Load from Memory
MOV      R0, R4 ; Rd = Op2
LDR      R3, [R5,R3] ; authz_owner_get_file_group
LDR      R3, [R3] ; Load from Memory
BLX     R3 ; authz_owner_get_file_group - returns the file's owner group name
MOV      R1, R0 ; group name

; if request.filename is world readable:
; group_name = NULL // (R1)

```

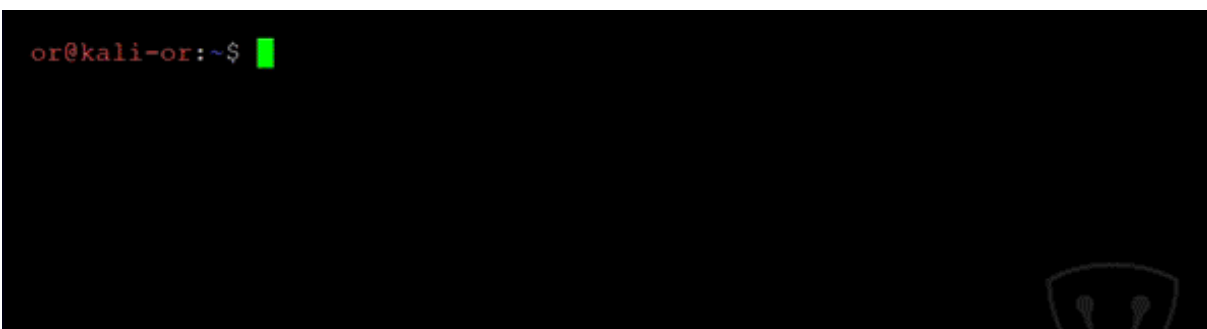
```
; group_name = NULL // (R1)
; else:
;   group_name = the file's owner group name // (R1)
;
loc_270C
MOV     R0, R4 ; struct request_rec *
LDMFD  SPI, {R4-R6,LR} ; Load Block from Memory
B      check_user_authz_by_file_owner_gid ; (request, group_name)
; End of function axis_group_file_provider
```

This IDA screenshot from **mod\_authz\_axisgroupfile.so** shows the **axis\_group\_file\_provider** function, that is registered (by apache's **ap\_register\_auth\_provider** function) as an authorization provider for paths that are subject to 'Require axis-group-file' directives. One can observe (in the upper part of the screenshot) that the **request.filename** component is used for checking whether the file is world-readable. In our example above, **request.filename** is the path of the world-readable **/usr/html/index.html** file, and thus the flow proceeds to calling the **check\_user\_authz\_by\_file\_owner\_gid** function, with the **group\_name** param as **NULL**. When invoked with **group\_name** as **NULL**, the latter function skips all authorization checks and Grants Access to the request.

Thus – the attacker is given an unauthenticated access to the **/bin/ssid's .srv** functionality

## PoC

In order to show we have the ability to reach the **/bin/ssid's .srv** functionality, we send a request with the 'return\_page' query-string parameter. This is a special parameter used for HTTP redirection. We know we have reached the **/bin/ssid's .srv** functionality as it returns a redirect with the parameter's value (the string "it\_worked") back in the response.





# CVE-2018-10662 – Unrestricted dbus access for users of the .srv functionality

Legitimate requests that reach `/bin/ssid`'s **.srv functionality** can choose one of several actions by setting the **action** parameter in the request's query-string. One possible action is **dbus**, which allows the user to invoke any dbus request as root (the uid and gid of the `/bin/ssid` process), without any restriction on the destination or content. Due to the dbus request originating from a root process – unrestricted access is granted to many dbus-services' interfaces. This happens because the authorization mechanism that is intended to limit such requests, **PolicyKit**, is configured to automatically grant access to requests originating from the root user.

```
<config version="0.1">
  <match user="root">
    <return result="yes"/>
  </match>
```

*The beginning of `/etc/PolicyKit/PolicyKit.conf`. "yes" means authorization is granted. See `PolicyKit.conf`'s manual page.*

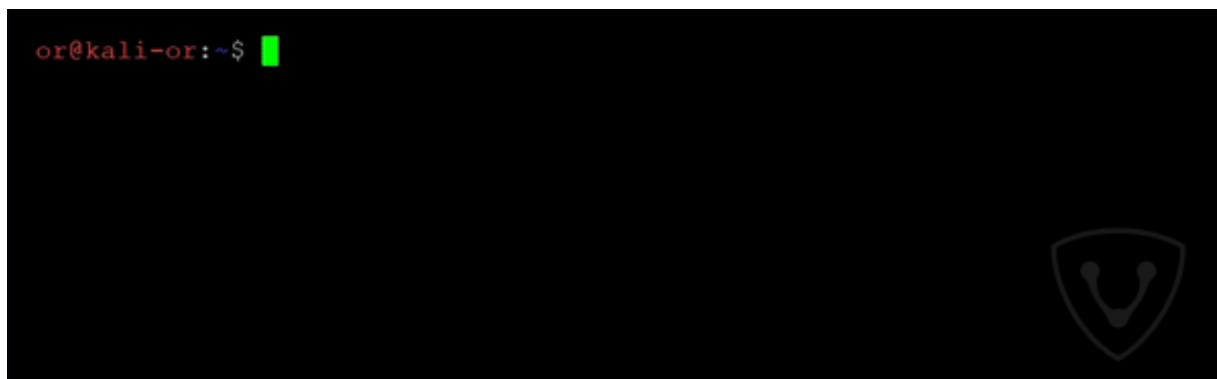
While the dbus interface in `/bin/ssid` only serves the purpose of fetching specific values from some specific dbus-enabled services, it exposes a much broader functionality, which has security consequences without justification.

For example, this interface gives users the ability to control any of the device **parhand** parameters' values. Control can be achieved by sending dbus-requests to invoke **policykit\_parhand** process' dbus-interface (`PolicyKitParhand`) functions. This interface offers the **SetParameter** and **SynchParameter** methods that are evocable by root dbus-clients. By executing **SetParameter** followed by **SynchParameter**, a

user can set the value of any **parhand** parameter and apply the changes.

## PoC

The camera's **parhand** parameter **Image.I0.Overlay.Enabled** controls whether to show an image on top of the camera's video output. As an example, we use the vulnerability to toggle its value from 'no' to 'yes'.



As a result of running these commands on a vulnerable camera, the overlay image (by default – a small Axis Logo) will appear in the top left corner of the video stream. One can log into the web-interface to see it:



## CVE-2018-10660 – Shell command

# injection vulnerability

To take advantage of this vulnerability you must have permissions to change certain **parhand** parameters. This can be achieved by one of:

1. Achieving/having administrator privileges (by using the **cgi** interface)
2. Executing code inside the **upnp** daemon
3. Finding other ways to control certain **parhand** parameters – as was achieved by **CVE-2018-10662** in the example of directly invoking the functions of **policykit\_parhand** (see above).

The **parhand** parameter handler is responsible for fetching, storing, and changing many of the device's internal parameters. When a user sets a parameter through the web interface, the relevant CGI script (`param.cgi`) forwards the set-parameter request to the **parhand** binary, which checks access-rights, and stores the parameter's value in the relevant configuration file.

Some of the parameters are used for feeding shell-scripts, and are defined as *Shell mounted* (`mount = "Shell{...}"` in the **parhand** configuration file). The parameters' values are parsed by the **parhand** ShellParser, which does not sanitize special shell characters and also does not quote the parameters' values. Some of these parameters (for example, the **Time.DST.Enabled** parameter we exploited) end up in configuration files (for example, `/etc/sysconfig/openntpd.conf`) in shell variable assignment format (e.g., `FOO=Bar`). These parameters are later used by shell init-scripts (for example, **parhand-systemctl restart time-source.service**) which run as a result of the setter command, that is executed when applying a new value for a parameter – by running the `sync` command.

The shell scripts directly execute the configuration file (for the purpose of including the configuration parameters), and by setting the parameter's value with a semicolon ("`;`"), we were able to inject arbitrary shell commands with root privileges.

The key factors in this vulnerability are:

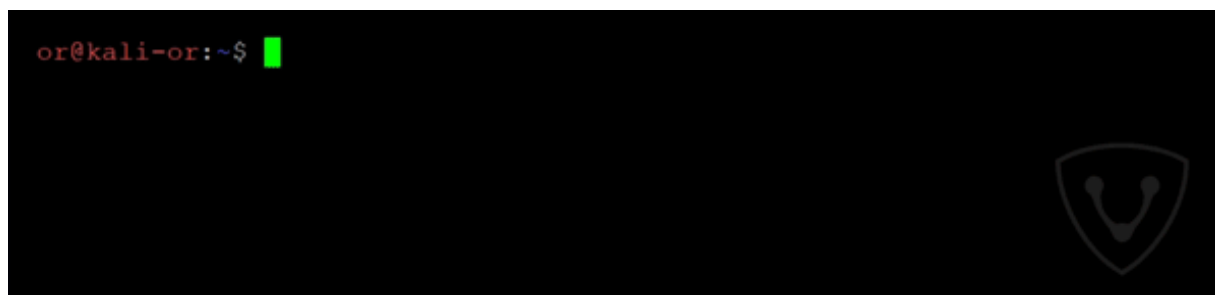
- Lack of input sanitization when parsing values that end up in a shell environment

- An outdated method is employed by the device, using shell scripts to set parameters by storing them in files as shell-assignment expressions and then executing the files.

Please also note that the parameters that can be set by the camera's **upnp** daemon can also be used to exploit this vulnerability to escalate privileges, in case the attacker happens to have the ability to execute code inside the UPnP daemon.

## PoC

Out of the possible options, we chose to trigger this vulnerability by using the **param.cgi** interface, which requires administrator credentials. We inject the **id** command, which prints user and group information of the current user to standard output. In our case, the standard output is redirected to the system log.



As a proof that the PoC worked – after executing the commands, we logged as an admin to the camera's management interface to view the system log [http://CAMERA\\_IP/axis-cgi/admin/systemlog.cgi](http://CAMERA_IP/axis-cgi/admin/systemlog.cgi), and were able to see the **id** command's output (see **uid** and **gid** at the bottom line):

```
[ INFO    ] parhand[867]: Updated configuration file /etc/sysconfig/openntpd.conf.  
[ INFO    ] systemd[1]: Stopping Time zone configuration...  
[ INFO    ] systemd[1]: Starting Time zone configuration...  
[ INFO    ] timezone-set[2781]: Setting time zone configuration  
[ INFO    ] timezone-set[2781]: uid=0(root) gid=0(root)
```

## Additional Vulnerabilities

This section provides details for four more vulnerabilities, that were not used in the attack sequence described above.

## CVE-2018-10664 – Crashing the httpd process

This vulnerability allows an unauthenticated adversary to crash the httpd process – causing (at least) a black screen for viewers that were already logged to the camera using the web interface with default settings. This vulnerability does not require any user credentials.

The following line (followed by a crash dump) is appended to the system log after triggering the vulnerability:

```
[ ERR ] kernel: [ 2819.017996] httpd: httpd: potentially unexpected fatal signal 11.
```

## PoC

This vulnerability is triggered by issuing an HTTP request to a **.cgi** script URL, with a **PATH\_INFO** that ends with the **.srv** extension.

## CVE-2018-10663 Information Leakage vulnerability in the /bin/ssid process

This vulnerability does not require any user credentials. The **'return\_page'** and **'servermanager\_return\_page'** query-string parameters in /bin/ssid's **.srv**

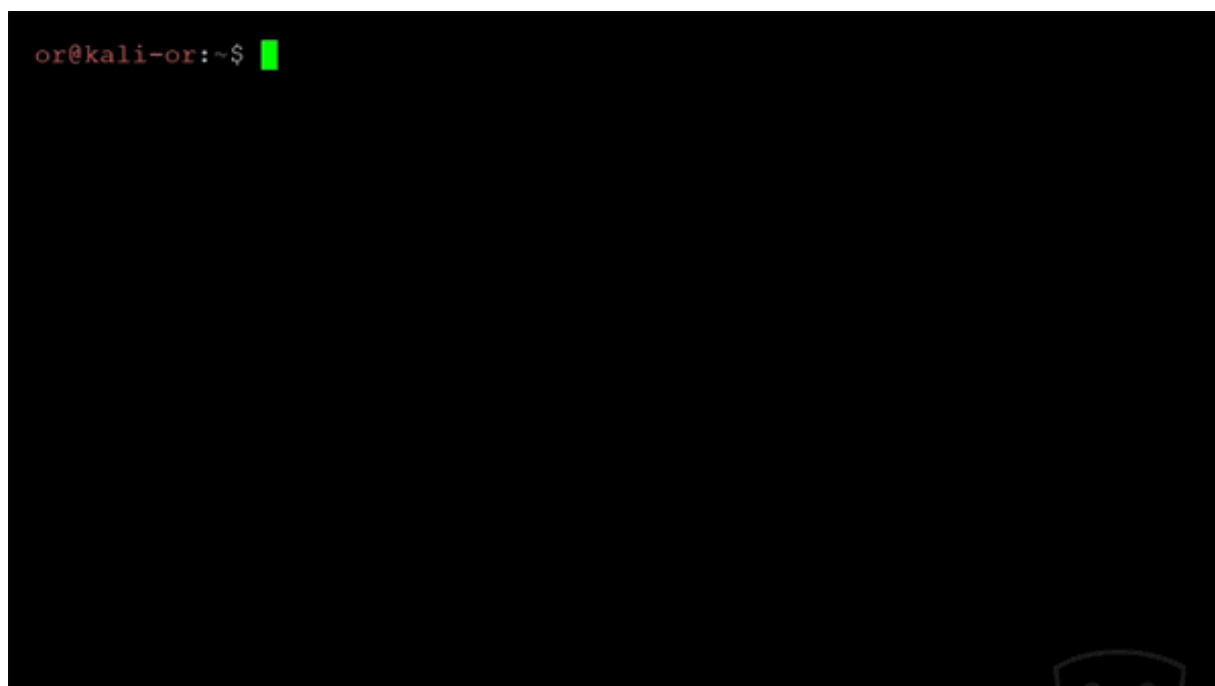
functionality are controlled by the user, and returned back to her in the response to the user's request. When dealt with in the response-building code – these fields' values are trimmed to a size of 0x200 bytes and copied to a malloced 0x200-bytes space by using the safe `__snprintf_chk` function. Then the return value of the `__snprintf_chk` function (supposedly their length) is saved in a struct member variable for later calculating the response's content length.


```
MOV          R1, #0x200 ; Rd = Op2
BL          __snprintf_chk ; Branch with Link
STR         R0, [R7,#8] ; Store to Memory
```

*A (partial) IDA Screenshot showing that the return value of `__snprintf_chk` is saved into a struct member.*

The problem is that the return value of the `__snprintf_chk` function is the “The number of characters that would have been written if n had been sufficiently large...” (taken from `sprintf`'s manual). This makes the calculated content-length larger than the actual data buffer, and as a result – extra bytes from memory are leaked in the response.

## PoC





Watch how the end of the response returned changes. The extra symbols and characters are leaked bytes that are adjacent to the response's buffer in memory.

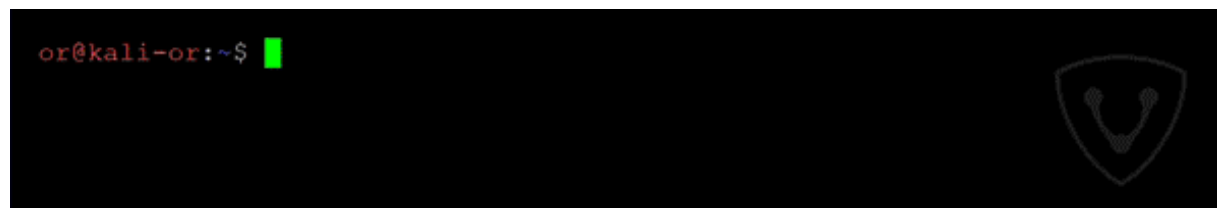
## CVE-2018-10658 Crashing the /bin/ssid process

This vulnerability does not require any user credentials. The unauthenticated user can send (by /bin/**ssid** .srv interface) dbus-request with a specially crafted string to crash the **ssid** service. This crash arises from code inside **libdbus-send.so** shared object or similar, as it produces the following log message:

```
[ INFO ] ssid[2334]: process 2334: arguments to
dbus_message_new_method_call() were incorrect, assertion "iface == NULL ||
_dbus_check_is_valid_interface (iface)" failed in file ../../dbus-1.10.14/dbus/dbus-
message.c line 1373.
```

As the crashes also occur by directly invoking "/usr/bin/**dbus-send**" with a similar string, this may affect other processes that include this code. Note that the /bin/**ssid** process will be respawned.

## PoC



## CVE-2018-10659 Crashing of the /bin/ssid process.

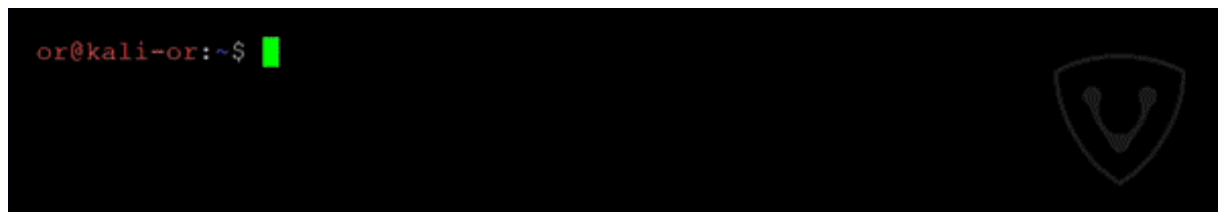
An unauthenticated user can send (by /bin/ssid .srv interface) a specially crafted command that will result in a code path that calls the **UND** undefined ARM instruction (and possibly a similar scenario in MIPS or other architecture's' cameras) that causes the process to crash. Note that the /bin/ssid process will be respawned.

The following log line (followed by the crash-dump) appears after triggering it:

```
[ ERR ] kernel: [ 2390.374778] ssid: ssid: potentially unexpected fatal signal 11.
```

This vulnerability does not require any user credentials.

## PoC



## Recommendations for Device Makers

We would like to relate to some bad architectural practices that were found in the cameras analyzed in this research, that make it easier for an attacker to discover and exploit vulnerabilities. We encourage device makers to take the below recommendations into consideration.



- **Lack of privilege separation:** This violates the concept of privilege separation ([https://en.wikipedia.org/wiki/Privilege\\_separation](https://en.wikipedia.org/wiki/Privilege_separation)), which states that a program should be divided into parts – each part limited to its own needed privileges. While every process in the system runs as root – a code-execution bug in any of the system’s processes will allow the attacker to escalate to root privileges. On the other hand, if less processes were running with high privileges – an attacker would have to discover vulnerabilities in a more restricted set of processes in order to escalate privileges, which is a harder task.
  - As an example, in **CVE-2018-10662**, `/bin/ssid` had an unrestricted dbus interface – that allowed an attacker to call dbus service’s functions. If the process wasn’t running as root, the dbus authorization policy wouldn’t allow many privileged dbus services’ functions to be invoked. But because the `/bin/ssid` process does run as root, all the dbus functions are exposed to the attacker without permission barriers.
- **Lack of proper input sanitization:** When getting input from an external interface, the input should be sanitized from characters that have damage potential. This could have prevented **CVE-2018-10660** – in which shell special characters were not escaped.
- **Minimize Use of Shell Scripts:** The extended use of shell scripts that take user input as parameters is discouraged. This approach enabled **CVE-2018-10660**. In another note – instead of directly executing dbus commands – the parhand infrastructure could be used (together with dbus and getters command).
- **Lack of Binary Firmware Encryption:** Firmware encryption will raise the bar and make it harder for adversaries to analyze the firmware for bugs, and specifically use binary diffing methods between the latest and previous firmware in order to find and analyze the patches and in this way – uncover the vulnerabilities that still exist in the previous version. Moreover, the device contained unstripped binaries with symbols like function names. This aided us in understanding how the code works. On the other hand, it is worth noting that the *security by obscurity* approach for firmware content may contribute to a situation in which issues exists but are not being discovered and remediated since the firmware is encrypted properly. Vendors should consider this tradeoff carefully.

## Acknowledgment

We would like to thank Axis Communications' security team for efficiently and promptly handling this security issue, and for their professional conduct of communication.

## Credit

Or Peles (@peles\_o), VDOO

## FAQ Section

### **Q1. How do I know if my device is vulnerable?**

In order to verify if your device is vulnerable or not – you need to check the [the ACV-128401 affected product list](#). If your camera's firmware is an earlier version or the version that appears in the ACV-128401 affected product list, your device is vulnerable, and we highly recommend on upgrading the device firmware. To check which firmware version your camera uses, you can do the following:

1. Using a web browser, access your camera.
2. Enter your username and password
3. Click "System" → "Options" → "Support" → "System Overview".
4. Look for the firmware version

If you have multiple devices it may be worthwhile to retrieve the firmware using either the Axis Device Manager software or programmatically via the Axis VAPIX API (see section 2.2 in the [official VAPIX documentation](#)).

### **Q2. How can I tell if my device was breached?**

The chances that your device was breached are very low as there are no known malware utilizing these vulnerabilities at the time of publication.

As IoT malware is normally crafted to go undetected, there's no easy way to know for sure. Any suspicious change to the device may indicate the existence of a botnet malware on your device.

A few ways to check:

1. **Your password is not working anymore** (and not because you simply forgot it) – this is a strong indication for a device that has been taken over.
2. **Your device settings were modified** – for example, videos are now sent to a different server.
3. **Spike in network traffic** – if possible, examine your router network statistics. A botnet could increase the amount of network traffic originated from the camera. Any spikes should alert you since unless you are streaming video from the camera, this number should be relatively low.

### **Q3. Is there a way to remediate my device if it was breached?**

At the time of publication, we are not aware of any malware abusing this issue. If you suspect your camera is breached, restore the camera to its factory settings. Doing so will restore the configuration to default settings, allowing you to connect and upgrade the firmware. Keep in mind that if you're using a firmware susceptible to the vulnerabilities detected by VDOO, the device might be targeted and can become infected again shortly. So, after resetting the device, make sure to immediately perform the firmware upgrade, prior to connecting the camera directly to the internet.

### **Q4. How to mitigate the risk if I can't update the camera's firmware?**

In order to reduce the camera's exposure and the ability to manage it remotely, it is recommended to place the device behind a firewall blocking port 80 and 443 (or the ports specified in the camera's configuration) and consider not allowing the camera to initiate any outbound connections. Another option is to put the device behind a reverse proxy that blocks the URLs we are using for the exploit (see above for additional details). Please contact [security@vdoo.com](mailto:security@vdoo.com) if additional help is necessary.

## Q5. How to upgrade the firmware in the camera?

To upgrade to the latest firmware, you can use Axis Device Manager, the camera's web interface or FTP. <https://www.axis.com/en-in/support/technical-notes/how-to-upgrade> to find the vendor's instructions for firmware update.

---

Share this:



Be the first to like this.

---

### Related

Major Vulnerabilities in Foscam Cameras  
In "Technical"

Behind the research glass - an insight into our approach  
In "About Us"

Why good security foundations are better than the best security mitigation  
In "About Us"

 TECHNICAL