Webcams and Secure Contexts

Many browsers, most notably Google Chrome, now implement the "Secure Contexts" specification (https://www.w3.org/TR/powerful-features/). This specification requires that browsers block access to a webcam from web sites that are not secured by HTTPS. This can present a challenge to the use of webcams for photo capture in a typical LAN-based deployment of DerbyNet.

Background

What Is HTTPS?

HTTPS is a secure protocol for connecting to web sites. It requires that a web site prove its identity to the browser – when you visit your bank's web site, you want to be sure it's really your bank and not a fake web site trying to get your information. Proof of a web site's identity is achieved by obtaining an "SSL certificate" for the site, usually by buying one from one of several commercial Certificate Authorities. Browsers are pre-configured with a list of trusted certificate authorities, and are able to conclusively test whether a certificate is valid. Many browsers mark the use of HTTPS by showing a small padlock or similar icon.

Secure Contexts

The Secure Contexts proposal suggests that browsers should prevent web sites from accessing your webcam (and other information) if the web site is not secured by HTTPS. It's still just a proposal, so some browsers implement it and some do not. There's an unresolved discussion whether an exception should be made when a browser connects to a web site hosted on the same machine (i.e., to localhost).

DerbyNet and Webcams

DerbyNet is most commonly deployed from a laptop on a local-area network (i.e., not on the public internet). The check-in page for DerbyNet supports the use of a webcam to capture racer and/or car photos as racers check in. Of course, this won't be possible if the browser prevents access to the webcam from the check-in page.

Some Strategies

Do You Need Webcam Access?

Some groups don't bother trying to capture car or racer photos at check-in. If that describes your group, then this issue doesn't affect you.

Try A Different Browser

As mentioned above, the Secure Contexts proposal is just a proposal, and browsers vary in their adherence to it. Google Chrome is probably the browser that most aggressively implements the proposal. (Given that the proposal originates with Google, that's probably unsurprising.)

Some browsers implement Secure Contexts, but allow the user to declare a web site to be trustworthy even if it doesn't use HTTPS.

As of this writing (October, 2018), on Windows, the latest versions of Chrome and Opera block webcam access without HTTPS. Mozilla Firefox allows the user to choose to allow webcam access without HTTPS, although the permission isn't persistent. Microsoft Edge allows full access to the webcam.

Host DerbyNet On An Internet Server

If your event has reliable internet access, you have the option of hosting your DerbyNet server somewhere on the internet, on a web server that already supports HTTPS.

If you or your group already have a web site, you could install DerbyNet there. If you don't already have an SSL certificate for your site, Let's Encrypt (http://letsencrypt.org) can provide one at no charge.

Capture Photos Outside the Browser

Secure Contexts just affects browsers, not the webcam itself. An alternative workflow might be to use a different application (i.e., not the browser) to capture photos, and then upload the photo(s) through the check-in page. (The check-in page supports "drag and drop" for photo uploading: open the photo capture dialog, then drag a photo onto it.)

DerbyNet also includes scripts in the "extras" folder for doing automated photo capture, with barcode scanners used to identify the racers.

Create a Self-Signed Certificate

While the usual practice for commercial web sites is to buy an SSL certificate from a vendor that's trusted by the browser, it's also possible to create your own SSL certificate, and install in the server. (A locally-produced certificate like this is said to be "self-signed.")

Browsers will not initially trust this certificate, but can (usually) be forced to accept it. Getting a browser to trust a new certificate may require administrator access, which can be an issue if using a machine you don't own.

Creating a Self-Signed Certificate on Windows[1]

1. In the UniController application in the UniServerZ folder, click on the Apache menu and select "Apache SSL > Server Certificate and Key generator." In the dialog that comes up, choose any names you like, and click "Generate." A new self-signed certificate will be created, and SSL serving will be enabled for the Apache server.
2. Again in the Apache menu, select "Change Apache Root-folders > Select new Server Root-Folder (ssl)". In the browser that appears, change the root folder from "ssl" to "www."
3. Stop the Apache server if it was running.
4. Start the Apache server.

1 Kudos to Carl Hunter for first noticing UniServer's support for this.

Creating a Self-Signed Certificate on Mac OSX

1. Open Terminal from the Applications/Utilities folder.
2. Type the following commands:

```
cd /etc/apache2

sudo mkdir ssl

cd ssl

sudo openssl req -x509 -nodes -days 3650 -newkey rsa:2048 \
        -keyout derbynet.key -out derbynet.crt
```

3. Edit the Apache configuration file, /etc/apache2/httpd.conf, and add these lines:

```
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/derbynet.crt
SSLCertificateKeyFile /etc/apache2/ssl/derbynet.key
```

4. Restart Apache

```
apachectl restart
```

Creating a Self-Signed Certificate on Debian/Raspbian

1. Install the ‘ssl-cert’ package to generate a self-signed certificate:

```
sudo apt-get install ssl-cert
```

2. Edit /etc/nginx/sites-available/default, and uncomment (remove the leading ‘#’ from) these lines:

```
listen 443 ssl default_server;

listen [::]:443 ssl default_server;

include snippets/snakeoil.conf;
```

3. Restart nginx:

```
sudo nginx -s restart
```