Secure Contexts and DerbyNet

Many browsers, most notably Google Chrome, now implement the “Secure Contexts” recommendation (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.

In addition, the Web Serial API, which may be used for connecting a track timer through a supporting browser, is available only in a secure context.

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 the site obtaining an “SSL certificate,” 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 site has a valid certificate. 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 frequently deployed from a laptop on a local-area network (i.e., not on the public internet). The replay kiosk uses a webcam to capture video of a heat; the check-in page supports the use of a webcam to capture racer and/or car photos as racers check in. Neither feature will be available if the browser prevents access to the webcam.

Additionally, the browser-based interface for a connected track timer depends on the Web Serial API, which is available only within a secure context.

Some Strategies

Host DerbyNet On An Internet Server

If your event venue 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.

Free hosting for DerbyNet is available at https://hosting.derbynet.org/signup.php, with full support for 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 for capturing photos at check-in 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. (A video describing this system is available at https://youtu.be/aR5vDUBemx4.)

Use 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 itin the server. A locally-produced certificate like this is said to be "self-signed."

Accepting Self-Signed Certificates

Browsers will not initially trust this certificate, but usually allow the user to choose to accept it. Typically the browser will display a message saying something like, "This Connection is Not Private." It's often necessary to click through to an "Advanced" or "Details" view in order to authorize accepting the certificate.

Since user intervention is required for a self-signed certificate, it doesn't actually matter what name the certificate attests to.

> ***NOTE (December, 2019):*** Some recent versions of Chrome browser on MacOS do not offer to accept the certificate. *There's a secret passphrase built into the error page. Just make sure the page is selected (click anywhere on the background), and type **thisisunsafe**.* (https://ericasberry.com/blog/2019/10/10/this-is-unsafe/)

Creating a Self-Signed Certificate on Windows[1]

The Apache web server included in the Windows installation of DerbyNet can be configured to use a self-signed certificate with just a few clicks.

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."

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

3. Stop the Apache server if it was running.
4. Start the Apache server.

Using a Self-Signed Certificate on MacOS

The MacOS installer for DerbyNet configures Apache with a self-signed certificate.

Using a Self-Signed Certificate on Debian/Raspbian

The derbynet-server package configures nginx with a self-signed certificate from the ‘ssl-cert’ package.