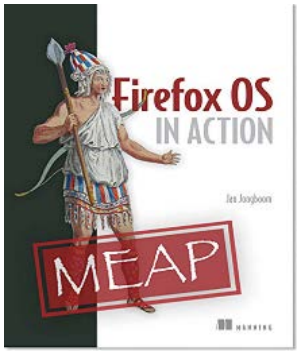

The Marketplace and how to deal with CSP on Firefox OS devices

By Jan Jongboom

For developers Firefox OS is amazing. Like on the normal web, users will have a modern browser that supports all the latest web standards. But if you've decided to develop in Firefox OS, you should know that the marketplace poses some constraints on your application that you will run into when publishing your app. In this article, based on Firefox OS in Action, I'll cover some of those constraints.

Firefox OS, the new mobile operating system by Mozilla, contains the Marketplace, which is a curated app store that holds apps for Firefox OS. It ships on all Firefox OS devices, and it's the primary channel where users will find your application. There are both free and paid apps, and the marketplace also supports in-app payments. It's not required to distribute applications through the marketplace, as you can also host your applications on your own server, or create a competing marketplace.

The reason for distributing through the marketplace is two-fold. First there is the free exposure and increased discoverability of your app. Second, users can only install privileged apps from the official marketplace. So if you want to have access to sensitive information, like the list of contacts in the phone, you'll need to distribute through the marketplace.

However, the marketplace poses some constraints on your application that you will run into when publishing your app. If you're aware of them before starting your first real app, it'll save you a headache.

The submission process

As a basis for this article, we use a note taking application that is available from http://firefoxosinaction.com/notes2/finished.zip.

An application for Firefox OS is essentially a ZIP file that contains all assets. In our case the ZIP file would contain six files:

- note.html
- note.css
- list.js
- detail.js
- manifest.webapp
- The logo that we specify in the manifest file

To create an application package, you'll need to create a ZIP file that holds all these files. Do not ZIP the folder that contains the files, but the files themselves. The manifest.webapp file should be in the root of the ZIP file. Create the ZIP file and give it a convenient name (for example note.zip).

The next step is to go the Marketplace Developer Hub located at https://marketplace.firefox.com/developers/. Look for the button “Submit an app.” You will be

prompted with a Sign in/Sign up button and the developer agreement. You don't need to pay any fee to participate in the developer program.

On the next page you can select the platforms on which your application will work. In almost any case your application will work on Firefox OS and on Firefox Mobile for Android, so you can select both options. Next click the “Packaged” option and locate the ZIP file you just created, as shown in Figure 1.

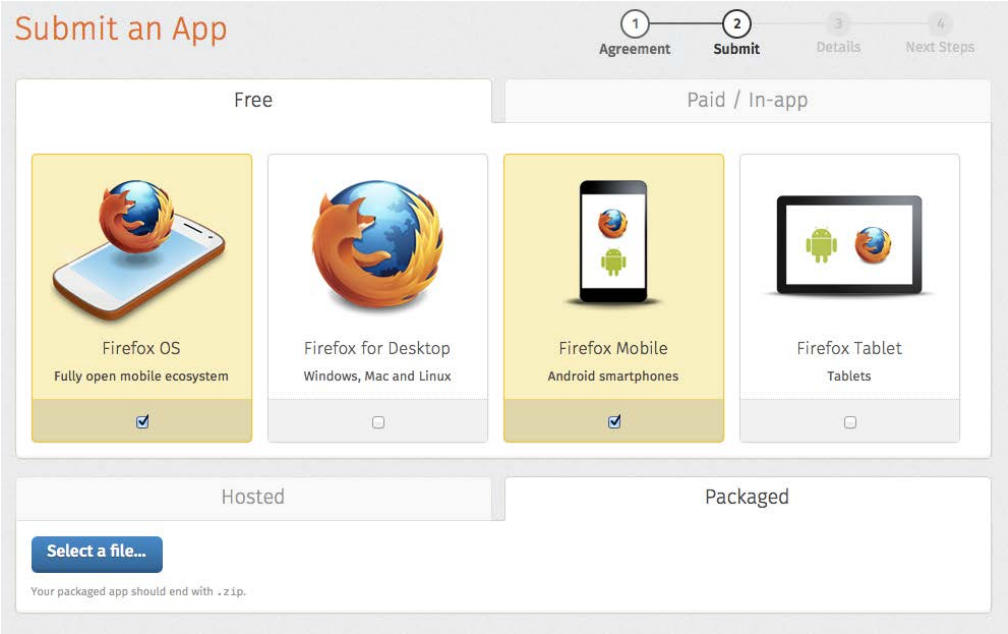

Figure 1 - Submitting an app. We selected Firefox OS and Firefox Mobile. Click the 'Select a file...' button to locate the ZIP file.

After the upload is completed, you'll get a report with warnings. Consider fixing the warnings before publishing your application. In general, you'll be warned about long app names and lack of icons. For now we can leave it at that.

On the next page you can fill out information on your app, e.g., description for users, screenshots, the categories that you want to appear in. If you're not submitting a finished application, toggle off the “Publish my app in the Firefox Marketplace as soon as it's reviewed.” option. Otherwise, your application will be published immediately after it has been reviewed.

There will be one more step -- to get a content rating for your app. After that your app will be in the review queue, and a Mozillian will review your application, which can take a few days. You'll get e-mail updates regarding the status of your app. On the “Status & Versions” page, you can see the expected review time. When all is well, your application will be in the marketplace, for the whole world to download (Figure 2).

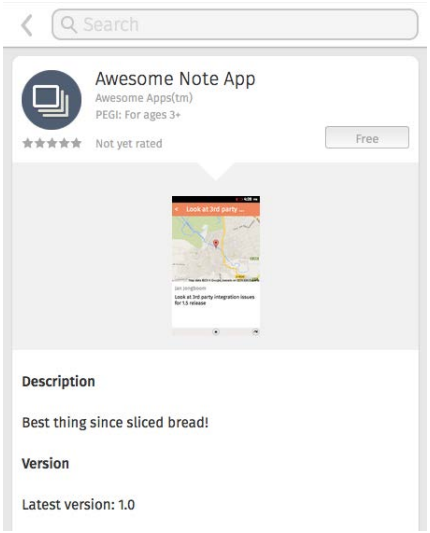

Figure 2 - The Awesome Note App in the Firefox OS marketplace

Dealing with the Content Security Policy

As I said before, while publishing apps in the marketplace is simple there are some constraints that you will run into. One important one is the Content Security Policy (CSP). The CSP is designed to avoid content injection that can be used to create cross-site scripting (XSS) attacks.

Here is an example of an XSS attack. There might be a piece of code like this in your application.

```
getDataFromServer(function(data) {
  someElement.innerHTML = data;
});
```

This might seem innocent. It's just a way to show some HTML that the server generated to a user. However, if the content is user-generated, an attacker can inject evil JavaScript code that can steal data. In our note application we have the Contacts permission, so we could inject some code to steal all contact information. For example, the data variable can hold:

```
non-evil content, la la, <script>
var n = navigator.mozContacts.getAll()</script>
```

and then upload all the contacts to a remote server. These kind of scenarios are prevented by the CSP. It's not possible to do code injection from JavaScript. Executing the code above will throw a CSP error.

Implications when developing a Firefox OS Application

The CSP is applied for all privileged applications. And as a rule of thumb: all code that cannot be reviewed by Mozilla, or that can be replaced at a later point (because the file lives on a server) will not pass the CSP. Here is a simple application that demonstrates the things you cannot do when writing privileged Firefox OS applications.

HTML

```
<!DOCTYPE html>

<html lang="en-us">

<head>

     <meta charset="utf-8">

     <title>Note App</title>

     <script src="http://www.firefoxosinaction.com/some.js"></script> #a

</head>


<body>

  <div onclick="alert('hello world')">Click me</div> #b


  <div style="display: none;">Inline styles</div> #c


  <script> #d

    console.log('inline javascript');

  </script>

</body>

</html>

#a Loading scripts from a server

#b Inline event handlers

#c Inline styles

#d <script> tags with content. You can only use <script src="..."/>
```

JavaScript

```
someElement.innerHTML = '<script>alert("yolo")</script>'; #a


var a = document.createElement('script'); #b

a.src = 'some-file.js';

document.body.appendChild(a)


eval('3 + 7'); #c


setTimeout('alert("here we go!")'); #d


new Function('var a = 7; return a * 2;'); #e
```

```
#a Injecting script tags through innerHTML

#b Creating new script tags through createElement

#c Calling eval

#d Functions that implicitly use eval, like setTimeout with a string instead of a
function

#e Calling new Function()
```

When developing your own applications, it's easy to work around these limitations. It gets more complicated when you're using a third-party library or a framework. Generally they are violating the CSP, but in most big frameworks there is a CSP compatibility mode. If you run into a CSP issue (and you will), search for "[library] csp". Most likely there will be either a CSP compatibility mode, or there will be another developer that created a workaround.

AJAX requests

Besides CSP, you will also run into trouble when creating AJAX requests to your server because of origin issues. Using AJAX, it's not possible to make requests from one domain to another domain. If your page is running on http://firefoxosinaction.com, and you run this code, it will fail:

```
$.get('http://janjongboom.com/hello', function(data) { alert(data) });
```

This is because it violates the same origin policy in the browser. You can only make AJAX requests to the same host on which you're running (in this case firefoxosinaction.com). The same thing applies to Firefox OS. Therefore you cannot make AJAX calls from your application by default.

In general, this is very good because it protects the user from evil websites that steal user data. If any website could access http://facebook.com/my-private-messages, that would be a massive security hole. However, there are times when you don't want to be protected from this. Think of an open API or public data that other hosts should have access too as well.

Generally there are three ways to circumvent this policy.

1. Use JSONP. This will not work in Firefox OS because it creates a <script> tag, which violates the CSP.
2. Configure your server to send Cross Origin Request Sharing (CORS) headers.
3. Use the systemXHR permission.

CORS

CORS is a technique where the server specifies that it's OK to make AJAX requests from other domains. If you have a public API it makes sense to allow all domains to make requests, but you can also limit usage to a subset. You can for example specify that only requests from my-other-domain.com may go through, but nothing else.

Normally an application on Firefox OS gets a random origin, in the form of app://<UUID>. This origin will be different for each install. This is not viable when you're implementing CORS because you can't whitelist your own application, as the origin changes every time. Luckily you can create a custom origin in the manifest file. Add before the last }:

```
"origin": "app://yourdomain.com"
```

This URL should always start with app://, and then include a domain that you control.

Next thing is to configure your web server to serve up CORS headers. For Apache you can add a line in your .htaccess file that reads:

```
Header set Access-Control-Allow-Origin "yourdomain.com"
```

To only open up your application to one domain. You can also enable CORS for all domains:

```
Header set Access-Control-Allow-Origin "*"
```

To configure other web servers, look at http://enable-cors.org.

THE SYSTEMXHR PERMISSION

The second option does not require fiddling with your server, but let's say you bypass the cross-origin protection via a permission in Firefox OS. First we need to announce that we want to use the feature through the manifest. Add the following code to the manifest file:

```
"permissions": {
  "systemXHR": {
    "description": "Put a good explanation here"
  }
}
```

Now when you're creating your AJAX request you'll need to specify that it should be a system request. It depends on the technique and library how to configure your requests. Here is how to do it using bare XMLHttpRequest objects, and via jQuery. For other libraries, search for "[library] mozsystem".

Bare XMLHttpRequest

```
var req = new XMLHttpRequest({ mozSystem: true }) #a

#a Pass an object into the constructor
```

jQuery

```
$.ajaxSetup({ #a
  xhrFields: {
    mozSystem: true
  }
})

#a Execute this before making any AJAX requests
```

Those were the main pitfalls that you run into when building a Firefox OS application. If you realize the limitations of CSP and AJAX calls on Firefox OS you'll save yourself quite a few headaches.