Web SAMP and HTTPS: What to do?

Mark Taylor, University of Bristol, UK

Introduction

SAMP, the Simple Application Messaging Protocol, is a standard developed within the Virtual Observatory to allow communication between different software items on the desktop. One popular usage scenario has been enabling one-click transmission of a table or FITS image from a web page, typically an archive search result of some kind, to a desktop application such as TOPCAT, Aladin or ds9. This has worked well for HTTP web pages since the introduction of the SAMP Web Profile in SAMP 1.3 (2012), but the Web Profile will not work over HTTPS. This problem was first spotted in late 2014, but is becoming increasingly apparent as more data providers adopt HTTPS.

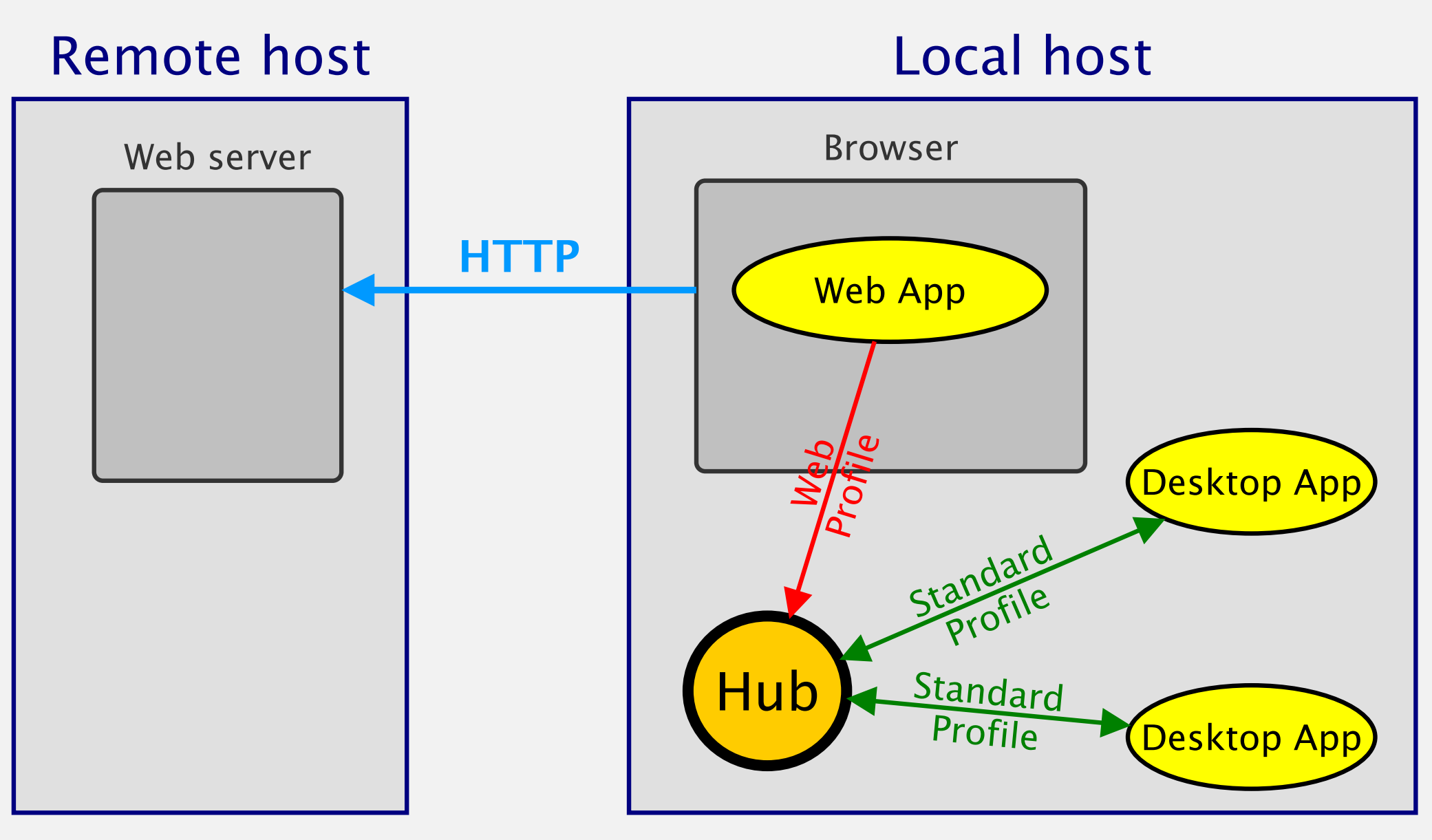
This paper presents a summary of the problem and explores some possible ways forward, for which working prototypes have been developed: specify a new HTTPS-capable Profile, use a SAMP-capable helper application, or abandon SAMP over HTTPS.

What is (Web) SAMP?

SAMP is middleware designed to allow loose interoperability between astronomy applications on a user's desktop. An example pattern of use is to send a catalogue from a catalogue analysis tool such as TOPCAT to an image analysis tool such as Aladin, so that activity in the two tools can be linked: for instance the catalogue positions can be overplotted on sky imagery, and if a user indicates a selection in a colour-magnitude plot in the catalogue tool, the corresponding objects can be highlighted in the image tool.

The architecture is based on message passing via a central *Hub*, a daemon that runs on the user's machine; the hub may be either free-standing or embedded in one of the running SAMP-aware applications. Each SAMP client has to establish two-way communication with this hub, which it does according to one of the *Profiles* defined by the SAMP standard. Initially, only the *Standard Profile* was defined, in which clients locate the hub from a file in the user's home directory and communicate via bi-directional XML-RPC calls.

Web applications (typically HTML + JavaScript) sometimes want to communicate with desktop applications too, but browser sandboxing means that the Standard Profile cannot be used, so in SAMP v1.3 the *Web Profile* was defined, which uses a well-known port, cross-origin workarounds and message polling to provide the required functionality.



How is Web SAMP Used?

Web SAMP is used in a number of web pages, but in practice SAMP interactions from web pages nearly all seem to follow the same pattern: the result of some archive search contains a button like "Send table via SAMP" or "Send FITS image via SAMP". This allows the user to take the result of a query made on the web and insert it directly into a chosen SAMP-capable desktop application such as TOPCAT or ds9 with one click. This is a nice convenience, but it's really just saving the user from having to save from the browser to disk and then reload into the client application. There seem to be very few Web SAMP applications that offer interoperability functions beyond exchanging a table or image*.

* WWT at CXC[1] messages sky position; I'm not aware of other cases

What is HTTPS?

HTTPS (secure HTTP) is HTTP layered over TLS (Transport Layer Security ≈ SSL = Secure Sockets Layer). It enforces host authentication, so that the client is guaranteed to be talking to who it thinks it is talking to. To make this work, the HTTPS server requires a trusted certificate. It also encrypts communications, which is required to support secure user authentication.

Driven by security concerns*, data providers are increasingly replacing their HTTP services with HTTPS.

* and possibly buzzword compliance

What's the Problem?

TL;DR: The Web Profile won't work for HTTPS

The problem is *mixed active content*.

If the browser retrieves a web page from a remote host using HTTPS, then that web page is only permitted to contact other URLs that are also HTTPS, since accessing HTTP pages would potentially compromise the integrity of the content that HTTPS assures.

The SAMP Web Profile relies on web applications using the browser-provided `XMLHttpRequest` API from JavaScript (or something similar) to communicate with the Hub at the fixed local URL `http://localhost:21012/`. If the web application has been downloaded from an HTTPS service, browser sandboxing prevents it from talking to that HTTP URL. This is reported by the browser (if you know where to look) with a message like

```
Blocked loading mixed active content 'http://localhost:21012/'
```

So, can't we just set up the Hub server to run on the localhost over HTTPS rather than HTTP? Unfortunately, this is much harder than it sounds. In order to run an HTTPS URL, a server must be able to present a *trusted certificate*, that is one recognised by the browser. If the well-known URL is of the form `https://localhost[:<port>]/<path>` then the hub needs a certificate for the domain "localhost"; Certificate Authorities are *not permitted* to issue certificates for localhost (or 127.0.0.1). It would be possible to self-sign a certificate for this domain, but then the browser would not recognise it (in the absence of browser security reconfiguration, which normal users probably don't want and shouldn't be encouraged to do). Conceivably one could acquire a certificate from a CA for a domain that DNS-resolves to 127.0.0.1; that's more effort than the SAMP user can be expected to make, but if done by the hub developer, then public/private key pairs would have to be distributed with the hub, which is bad practice and may lead to certificate revocation. In principle it would be possible for the user to acquire a certificate associated with the actual hostname of their machine and install it into the hub, but again that's a lot of work and expense for the user, and there are still problems for the web app to determine what the hostname actually is in order to contact the server. In short, running an HTTPS service for access on a well-known local host URL seems to be impossible.

Can we specify an HTTPS Profile?

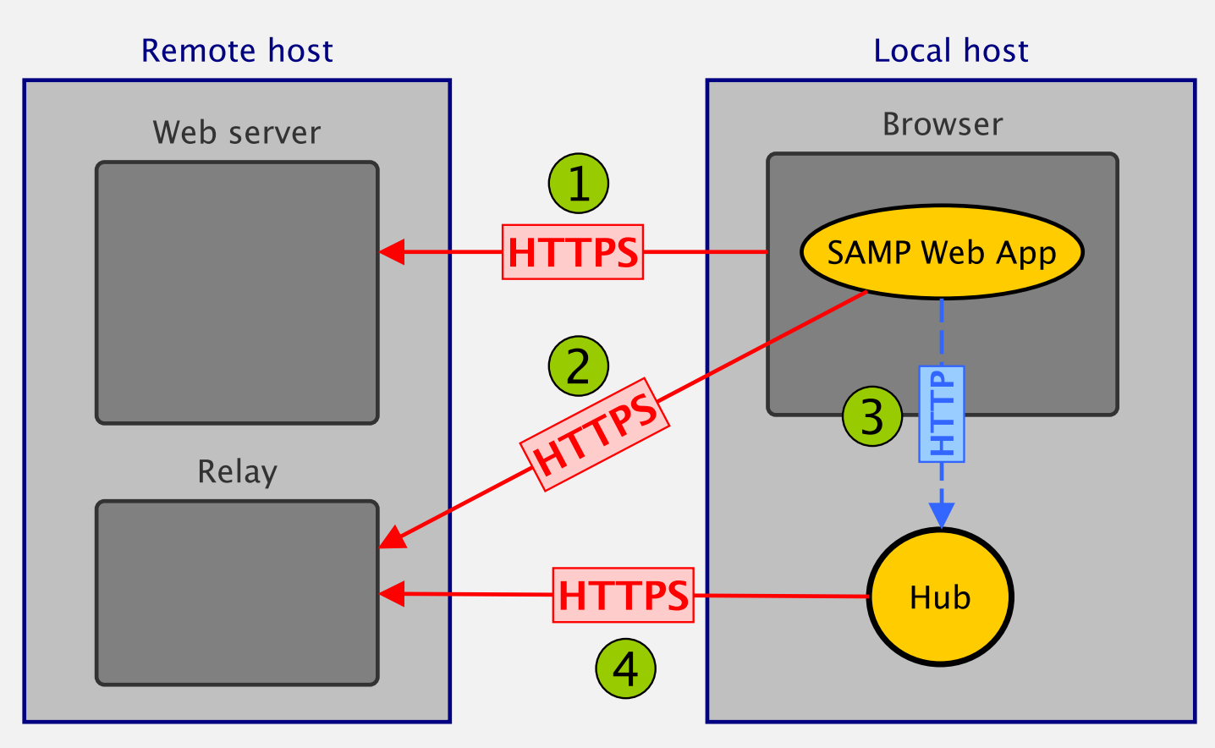
TL;DR: Yes, but it's horrible

Since the web application can't communicate via HTTP, it has to communicate via HTTPS. Since it can't talk to the Hub via HTTPS, it has to talk to somebody else who can forward a message. The solution is for both the web application and the Hub to communicate with a third service which can act as an intermediary.
We therefore set up a *Relay* service on an external server that can serve HTTPS (it has a trusted certificate) and arrange for that to forward messages between the web application and the Hub. The host of the Relay may or may not be the same as the host serving the web application in the first place, it doesn't matter.
Both web-app↔relay and hub↔relay communications have to be achieved using outgoing HTTPS calls from the client host to the Relay server, so this involves both the hub and the web application polling for messages that may have arrived from the other (Web Sockets might be a better solution than polling here, but it doesn't change the basic architecture). As soon as the web application starts running, it can start this polling. One problem remains: how does the Hub know to go looking for messages the web application may be leaving for it, i.e. when and where to start polling? One possibility would be a single centralised relay service for all HTTPS Web SAMP applications, but this would require centralised infrastructure, a single point of failure, and it would scale badly.

A loophole in current browser security policy allows a hacky solution: although browsers block mixed *active* content, they generally permit mixed *passive* content. Active content covers executable-code-like items such as Javascript inclusions, stylesheet references and XMLHttpRequest calls, while Passive content covers inert items such as embedded video, audio and image tags. We can abuse this laxity to allow the web application to smuggle a minimal one-way message to the hub by requesting an image from an HTTP server on a well-known localhost port with a URL whose text contains enough information to request polling. Specifically, the web application contains a small (maybe invisible) embedded image with a tag like:

```
<IMG src="http://localhost:21013/nudge?relay=https://ex.com/samp-relay"/>
```

We call this manoevre the *Nudge*; the same mechanism is not flexible enough to be used for SAMP communication in general (no information is returned to the calling code), but it is enough to bootstrap communication with the relay.



1. Browser loads web app
2. Web app starts polling relay
3. Web app nudges hub
4. Hub starts polling relay

What's wrong with this scheme?

- It's pretty complicated (difficult to implement; lots to go wrong)
- Communication between two processes on the local host has to get routed through a potentially distant remote server (impact on latency, reliability, security)
- Browsers only grudgingly allow mixed passive content:
  - Browsers indicate reduced security, e.g. 🔒 → 🔒⚠
  - This option may be revoked by future browser security policies (that is the stated intention of the W3C *Mixed Content* specification[2])

So, What Now?

Here are three possible ways forward.

1. Standardise HTTPS Profile

The HTTPS Profile outlined above has been prototyped and, though it requires some effort to set up, is known to work. A full specification, prototype hub and relay software, and example working HTTPS-based web applications are available[3]. This implementation has been deployed in production (it requires the user to run a custom TOPCAT) at SSDC[4].

To take this forward, it would be necessary to issue a new version of the SAMP specification that defines the new HTTPS Profile alongside the existing Standard and Web Profiles. This involves drafting the additional text, providing two independent implementations of the new functionality (the existing prototype is in Java; probably a Python one would also be required), pushing the new version through the IVOA Recommendation process, and ensuring that users are working with HTTPS-Profile-capable Hub implementations by embedding the updated hub in popular SAMP clients. Data providers adopting the new profile would need to deploy Relay services alongside their existing web applications, which makes the process of providing SAMP-capable web pages more complex. The standardisation process in particular is time consuming in terms of both effort and elapsed time, especially for a technically complex enhancement like this. There remain also some loose ends to tie up in the existing prototype implementation.

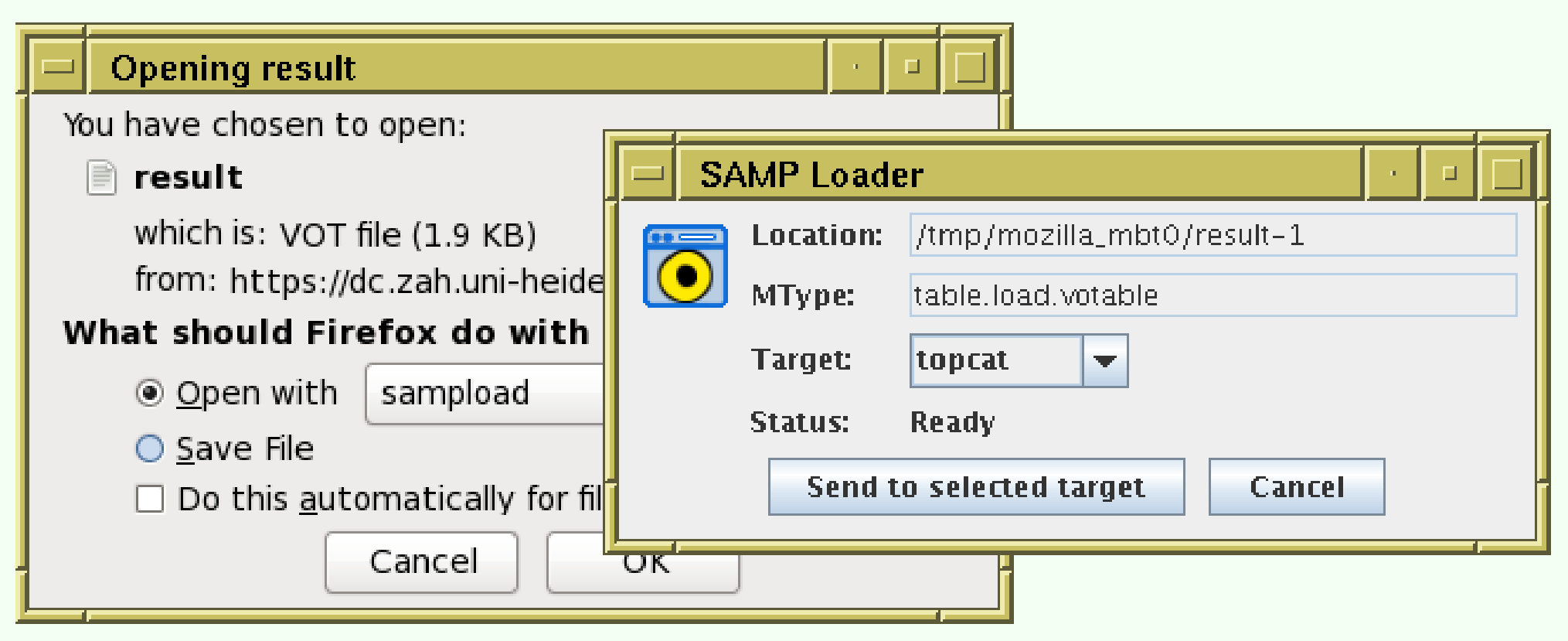
Pro: SAMP works equally from HTTPS and HTTP
Con: Considerable effort required; slow to get working
Con: May stop working if browser security policies change

2. SAMP-Capable Helper Application

Since by far the most common use of Web SAMP is to ask desktop applications to load a VOTable or FITS file, we could get away with something much simpler than a full SAMP client.

One possibility is providing a helper application for use with browsers that accepts a filename on the command line and forwards it to running SAMP-capable desktop clients. The user would either associate the helper in the browser with suitable MIME types (`application/fits`, `application/x-votable+xml`) or choose it at a browser's **Open with ...** prompt, so that it would get invoked on VOTable/FITS download. Since this would work with the browser's standard mechanism for passing files to desktop applications (download to temporary file; pass to application on the command line) no strange or questionable tricks are required.

Sampload, an example such helper application, has been written as a proof of concept. When invoked at download time, this identifies file type by examination, connects to the SAMP Hub using the Standard Profile, then pops up a window offering to send a SAMP `table.load.votable`/`cdf` or `image.load.fits` message to a suitable SAMP client if one is running. Once configured in the browser, it works quite smoothly. No additional infrastructure is required. This utility is available as part of JSAMP v1.3.6[5,6].



Opening result
You have chosen to open:
result
which is: VOT file (1.9 KB)
from: https://dc.zah.uni-heide
What should Firefox do with
(●) Open with: sampload
( ) Save File
☐ Do this automatically for fil
Cancel | OK

SAMP Loader
Location: /tmp/mozilla_mbt0/result-1
MType: table.load.votable
Target: topcat
Status: Ready
Send to selected target | Cancel

Pro: Not much development effort required
Con: Some user effort required (helper download and configuration)

3. Do Nothing

We could finally abandon the idea of using SAMP from HTTPS web pages. Web SAMP will still work from HTTP pages, but not from the increasing number of pages served using HTTPS. Users of those services will just have to save files to local disk and reload them into a suitable local application rather than use a "Send to SAMP" button or similar. It's not as convenient, but doesn't really stop users doing anything that is not otherwise possible.

This does not mean the end of SAMP, which is still a useful technology for non-browser-based applications, e.g. TOPCAT↔Aladin interoperability, especially for more complex interactions than just exchanging table or image files.

Pro: Easy
Con: No SAMP from HTTPS web pages

[1] http://cxc.harvard.edu/csc2/wwt.html
[2] https://www.w3.org/TR/mixed-content/
[3] http://andromeda.star.bristol.ac.uk/websamp/
[4] https://www.ssdc.asi.it/boomerang/
[5] https://github.com/mbtaylor/jsamp
[6] http://andromeda.star.bris.ac.uk/websamp/sampload.html

Input Welcome!

This poster presents my thoughts along with some experiments I have pursued to explore this problem and possible solutions. If you have ideas on this topic, please talk to me, or discuss it within the IVOA or on the `apps-samp@ivoa.net` mailing list. Your opinions are especially welcome if you're a data provider who wants to use SAMP over HTTPS; if you have Web SAMP requirements beyond the simple load-a-table-or-image use case; if you're a developer who is keen or willing to implement the HTTPS profile (in Python?); if you do or don't support some of the options presented here; or if you have a different or better idea for how to move forward on this than those listed here.