SAMP: Web Profile

Mark Taylor (GAVO/Bristol)

IVOA Interop Meeting
Naples

16 May 2011

$Id: websamp.tex,v 1.8 2011/05/15 09:52:08 mbt Exp $

Outline

SAMP for web applications

- The Problem *(recap)*
- Web Profile Solution
  - General features
  - Initial proposal details (apart from security)
    - How it works
    - Current status
  - Security
    - Existing approach
    - Issues, questions, proposals, discussion
- Next steps

Target Capability

- SAMP works well for *desktop clients*
- Would like it to work for *web clients* (code running in a browser)
  - In-browser technologies:
    - JavaScript (a.k.a. JScript, ECMAScript)
    - Adobe Flash
    - MS Silverlight
    - Java applet *(if signed, works already)*
  - Example capabilities:
    - Provide a button which sends a table/image/spectrum to a suitable desktop viewer
    - Receive information from desktop clients, e.g. highlight catalogue rows
    - Communicate with other web pages loaded in the same browser
  - Many persuasive use cases!

Technical Barriers

Browsers impose security restrictions ("sandbox") on web clients:

- can't read local files
- can't access URLs on localhost or external hosts *(cross-domain restrictions)*
- can't run an HTTP server to receive callbacks

$\Rightarrow$ Untrusted web clients can't exercise user privileges

- ☺ to damage the user's system
- ☹ to send/receive SAMP messages using the Standard Profile

Alternative Profile

- Alternative profiles explicitly permitted in SAMP
  - SAMP = generic core + specific profile(s)
  - Profile = hub discovery + RPC encoding/transport + callback arrangements
  - Until now (SAMP v1.11/1.2), only Standard Profile defined
  - Door left open for other possibilities
- Web Profile:
  - Need something that will allow a sandboxed application to find and communicate with hub

Web Profile Details

Web Profile is like *Standard Profile* (uses XML-RPC), but:

- Hub Discovery:
  - Hub server resides on well-known port (`http://localhost:21012/`)
    - $\Rightarrow$ only one instance per machine
- Hub Communications:
  - Hub XML-RPC HTTP server uses one or more *cross-domain workarounds*
  - These are configured to allow *unrestricted access* to server from sandboxed clients
- Callbacks:
  - Reverse HTTP/"Long poll" pattern
    - Client pulls callback instructions from hub, rather than hub pushing to client
    - Client may make repeated periodic short-timeout polls, or blocking long-timeout requests
    - Hub response contains XML-RPC (`<methodName>`, `<params>`) pairs
- Data URL Dereferencing:
  - Hub provides proxy service for external URLs

Cross-Domain Workarounds

Cross-domain access from within the browser sandbox

- Common requirement (Flickr, Twitter, YouTube, Amazon, . . .)
- HTTP server somehow declares sandboxed clients may access its resources
- Several client- and browser-specific options exist:
  - **CORS:** implement Cross-Origin Resource Sharing standard
    - Server reads/writes HTTP headers to signal cross-domain policy to browser
    - W3C standard (`http://www.w3.org/cors/`)
    - JavaScript support in XMLHttpRequest Level 2 (Firefox 3.5+, Chrome 2.0+, Safari 4.0+)
    - JScript support in XDomainRequest (IE8+)
  - **Flash:** serve `/crossdomain.xml` resource
    - Server provides XML file(s) describing cross-domain policy to browser
    - Introduced by Adobe Flash
    - Flash support since version 7(?)
    - MS Silverlight support in all(?) versions
    - Java support for (unsigned) applets and JNLP in versions 1.6.0_10+
  - **Silverlight:** serve `/clientaccesspolicy.xml` resource
    - Works like `crossdomain.xml`
    - MS Silverlight support (preferred alternative to `crossdomain.xml`)

Cross-Domain Workarounds

What workarounds work with what clients?

- CORS (Cross-Origin Resource Sharing)
  - JavaScript in modern browsers (Firefox, Chrome, Safari, IE)
  - More browsers in future?
  - Other HTML5-friendly technology?
- Flash (/crossdomain.xml)
  - Flash clients
  - JavaScript in older browsers (JS can use Flash for HTTP)
  - Silverlight
  - Unsigned Java applets

Status: Implementation

- Hubs:
  - JSAMP hub (v1.2) *(tested and working)*
  - SAMPy hub (v1.2.1) *(tested and working)*
- In-browser clients:
  - JavaScript *(tested, works with most browsers)*
    - Client library `http://www.star.bris.ac.uk/~mbt/websamp/`
    - Uses CORS for browsers that support it, Flash for others
    - Tested with several non-ancient browsers; believed to work on most except Opera
    - Currently undocumented and scrappy
  - Flash *(indirectly tested, working)*
  - Silverlight *(not tested)*
    - Expected to work
  - Unsigned Java applet/Unsigned JNLP *(so far, not working)*
    - Not clear what the problem is
- Desktop Clients (useful for testing only):
  - Java client library in JSAMP *(tested, working)*

Status: Standardisation

Standardisation desirable

- Decided in Nara to adopt Web Profile as a standard
- Either new Recommendation-track document, or part of SAMP standard
- . . . subject to further consideration of security issues

Progress towards acceptance in SAMP:

- At least 2 interoperating implementations ☑
  - Hubs: Java, Python
  - Clients: JavaScript, JavaScript/Flash, Java application
- Validation tool ☑
  - JSAMP test suite (tests client-hub interaction, but not from a browser and does not test cross-domain capabilities)
  - A JavaScript test suite would be a good idea
- Documented in Working Draft
  - WD-SAMP-1.3-20110512 just published
  - New section 5. Web Profile; otherwise, almost the same as REC-SAMP-1.2
  - Needs further internal/external scrutiny
  - Some security issues TBD . . .

Security

Is subverting browser security measures such a good idea . . . ?

- Cross-domain workarounds (try to) remove all restrictions to web apps contacting Hub HTTP server
- What can hostile web apps do by contacting the Hub HTTP Server?
  - Register with SAMP — *dangerous!*
    - SAMP clients can get full access to user resources (e.g. filesystem I/O)
  - Anything else — *harmless*
    - hub offers no useful/dangerous services to *unregistered* applications
    - denial of service attacks are possible — but web pages can mount those anyway
- So, security needs to be applied *only at registration time*
  - Only allow trusted clients to register
  - But . . . what's a trusted client?

Security

How to determine if a registering client is trustworthy?

1. Only accept clients from local host — *Yes*
2. Require explicit consent of user — *Yes*
3. Attempt secure authentication — *???*

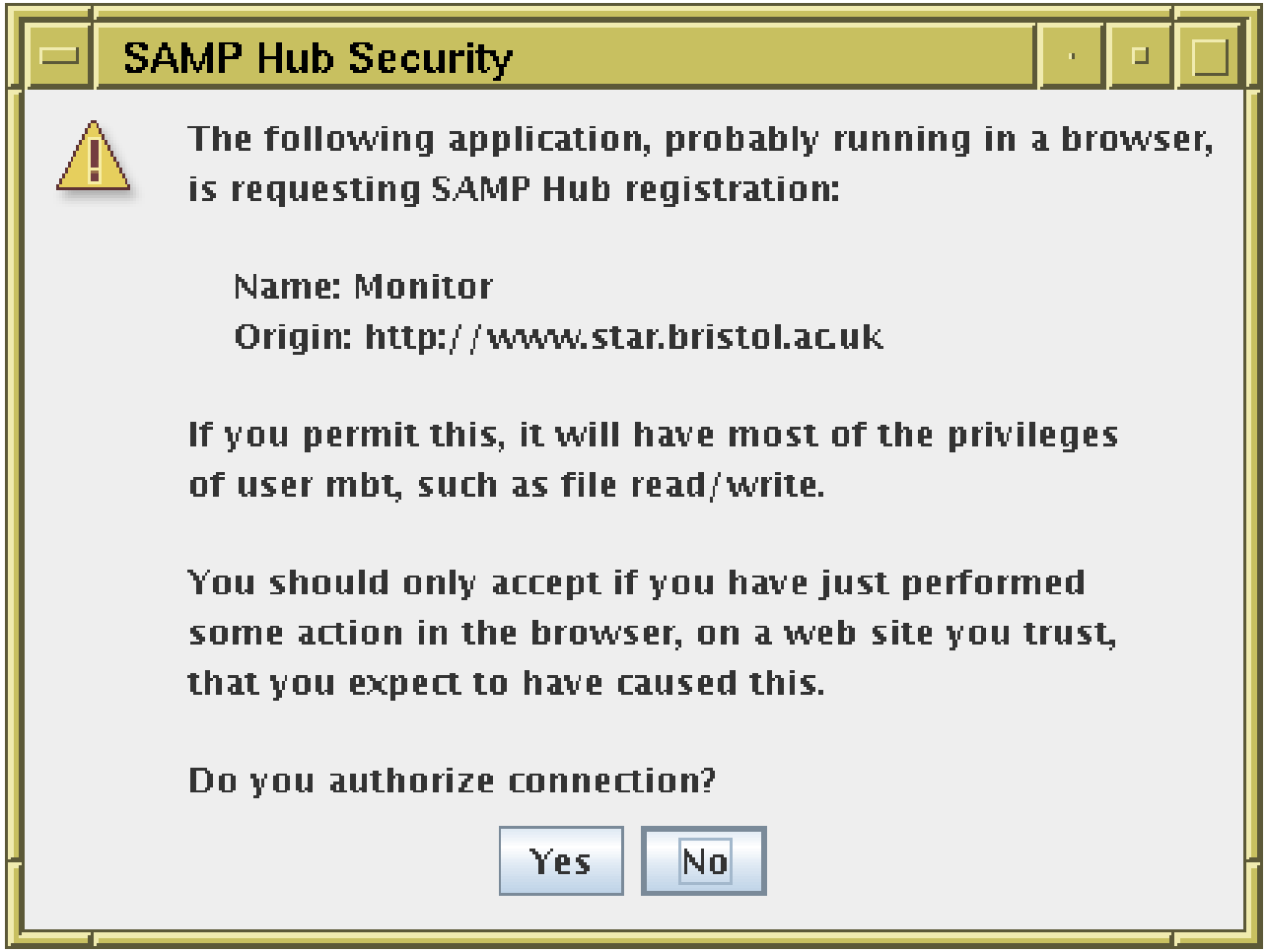
Registration Control: Local Clients Only

HTTP connections from remote hosts rejected

- Web browser assumed to run on same host as SAMP hub
- Remote host requests can't come from browser, must be bogus
- The only registrations allowed by this criterion are:
  - Web apps in hub-owner's browser
    - OK — intended)
  - Non browser-based processes of hub-owner
    - OK — not intended but have user privileges anyway, so no extra risk
  - Processes of other users on the local host
    - possibly problematic, but hostile local users rare, and mitigated by *Explicit User Consent*

Registration Control: Explicit User Consent

- Popup dialogue asks user if application may register
  - If not explicitly allowed, registration is denied



Registration Control: Explicit User Consent

- Popup dialogue asks user if application may register
  - If not explicitly allowed, registration is denied
- But how does the user know which application is asking?
  - Application *Name*
    - Always present
    - Supplied by application with reg request — unrestricted client-chosen string
  - Application *Origin* (e.g. `http://example.com:8080`, identifies server)
    - Only present if CORS is in use (not Flash/Silverlight)
    - HTTP header inserted by browser, cannot be faked by CORS client
    - Can it be faked by Flash/Silverlight client? Not sure
  - *Timing* of dialogue appearance
    - Only popped up immediately following a user action in the browser
    - User accepts iff he trusts the web page just interacted with
    - Intuitive and familiar way of doing things (signed applet, signed WebStart)
    - *Possibility* of simultaneous legitimate and hostile requests — but unlikely
    - Vulnerable to phishing attacks — astro/VO phishing sites not currently known?

  — Can we do better?

Registration Control: Client Authentication

- Would like to authenticate clients seeking to register
  - User could see this information to decide whether to trust or not
  - Problem: don't have much reliable information about registering client
  - In particular don't have URL/content of web application
  - May have *Origin* (location of server)
    - Guaranteed reliable for CORS, not present for Flash/Silverlight
  - Possibilities:
    - If origin is uses HTTPS:
      - Hub contacts any resource at origin server (e.g. root resource, `https://example.com/`), examines HTTPS certificate
    - Client provides [URL of] signed resource:
      - Signed content is origin string (e.g. "http://example.com")
      - Hub checks that signed content matches origin, and examines signing certificate
  - These don't authenticate authorship of web app, but do authenticate ownership of server it was downloaded from — probably good enough

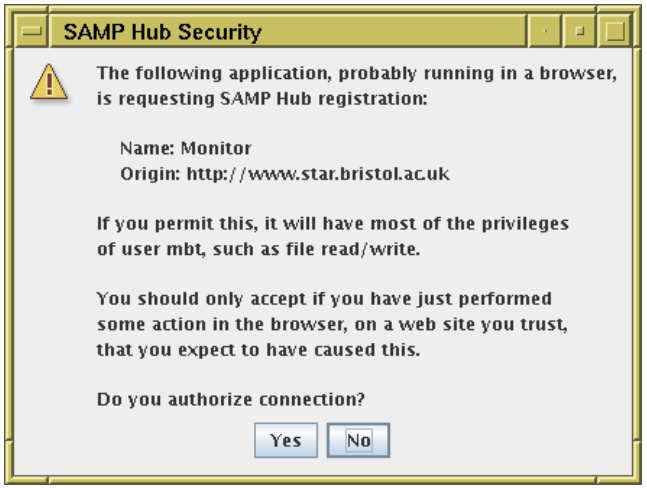
Authentication Usefulness

Even if clients can be authenticated, is this useful?

- Authentication infrastructure is still required
  - Need available [list of] Certificate Authorities for web app providers and users to agree to trust
- If introduced now, trusted signatures wouldn't be used
  - Web app authors would self-sign certificates
  - Web app users would see the warnings and (usually) click "OK"
  - This is what happens now
    - e.g. TOPCAT JNLP, Aladin JNLP, SAI Open Clusters applet, . . .
- Maybe in the future this will change?

Web Profile vs. Self-Signed App

Compare Web Profile with **self-signed** Java applet/JNLP:



(Though note: **Origin** *may be missing)*

- Most (all?) existing astro/VO signed applets/apps are self-signed
  - Authentication mechanism present but unused — self-signing = no authentication
  - In this case Web Profile has similar security to "signed" applet/app — already in use
    - but in absence of CORS, Origin info may be missing
- Apps signed by a suitable Certificate Authority would be more secure
  - What suitable CAs are available for VO providers? (eScience? others?)
  - Few(?) astro users have browsers set up to trust such suitable CAs

Mitigation Options

Possible ways to reduce security exposure (in standard or software):

- Only allow CORS, not Flash/Silverlight cross-domain workarounds
  - Guarantees reliable Origin visible to user, and possibly available for authentication
  - Allows JavaScript on modern browsers; excludes Flash, Silverlight
- Add authentication capabilities to the standard based on Origin
  - Only possible with CORS, not Flash/Silverlight cross-domain workarounds
  - Still vulnerable to hostile users on the local host (not common?)
  - Still requires authentication framework (e.g. VO-blessed CA list)
  - Authentication options:
    - HTTPS — requires web apps to be served using HTTPS
    - Signed resource on server — significant hub implementation work required?
    - Other ideas?
- Is there any other way to do authentication?
  - Self-signed applets/apps will still do the same job, insecurely
- Turn off Web Profile in hubs by default, only use it if user explicitly turns it on
  - In practice will mostly restrict use to SAMP experts
  - Experimental implementations (JSAMP, SAMPy) currently do this
- Throw away the Web Profile as irredeemably insecure

Summary

- Security summary:
  - Cross-origin work arounds not in themselves dangerous
  - Danger is only when client registers
- Existing solution (implemented in JSAMP & SAMPy):
  - Registration controlled by user consent (popup dialogue)
    - User decides based on informal trust of website
    - User knows which website is trying to register by:
      - CORS: dialogue displays identity of website
      - Flash: user infers identity of website from preceding browser activity
  - *My opinion: low-tech, but in practice reliable*
- Adding secure authentication
  - May be possible to do with some effort
  - Probably necesary to restrict to CORS (outlaw Flash, Silverlight)
  - *My opinion: doesn't buy you much with current security infrastructure*

Next Steps

- Do we:
  - Keep Web Profile as it is?
  - Mandate/recomment/implement authentication?
    - Restrict to CORS-only to make this reliable?
  - Deem Web Profile insecure and
    - make sure it's switched off by default in hubs implementations?
    - ditch it?