ECP

- Overview
- Direct vs. Delegated Authentication
- Code Availability
    - Links to Known Client Implementations
- Authentication
- Discovery
- Implementing Clients
- What About Non-Web/HTTP Protocols?

Overview

ECP is a SAML acronym that stands for "Enhanced Client or Proxy". The name is historical; the basic point of an enhanced client is that it's not a browser. The ECP profile is an adaptation of the SAML profile used for Browser SSO with the parts that were designed around the limitations of a browser removed.

Thus, it is the profile of SAML authentication designed for clients other than browsers, such as:

- desktop applications
- server-side code running in a web application
- just about anything else that's not a browser

The ECP profile that was included in the original SAML 2.0 standard, like the Browser SSO profile, is designed around HTTP applications and a session-oriented model. While it is not designed for a browser client, it is specific to HTTP, including traditional web sites and also web services, whether using SOAP, plain XML, REST-style services, JSON, or whatever other "reinvent the wheel" format emerges. It assumes a session-based security approach; that is, instead of securing each HTTP request, it assumes the SAML login establishes a session between the client and the web server, usually via a cookie. In more advanced cases, the session might be based on a key.

A new Version 2.0 of the ECP profile is now standardized, and is the best choice to base a new implementation or project around, as it is backwardly-compatible with the original and is described a bit more clearly.

Direct vs. Delegated Authentication

The basic definition of the ECP profile is designed around the same set of "actors" as the Browser SSO profile:

- a subject controlling a client
- an IdP that can authenticate the subject
- an SP that can accept an assertion from the IdP to establish a session with the subject.

In other words, it's a "single tier" profile for a user accessing a service. The bulk of this topic focuses on this simpler use case.

A more advanced variant of the ECP work can be combined with some other profiles to solve a more complicated problem, that of multi-tier delegation. In the delegated case, the "client" accessing the web service is acting on behalf of an earlier client that authenticated to it with SAML. With delegation, the web service can recognize both the original client/subject and the "delegate" that is accessing it on behalf of that subject, and control or limit access based on both identities.

This companion wiki space contains an in-depth technical discussion of this more advanced scenario and includes links to early versions of extensions and libraries in support of delegation.

Code Availability

Getting down to it, is ECP usable now? Yes and no. There is little "real world" support for the profile in SAML implementations other than Shibboleth. Supposedly many commercial products do support it, but we have no direct knowledge of any that actually do.

Within the Shibboleth System, the SP includes support for ECP (both basic and delegated variants). The IdP supports extensions that provide the basic and delegated variants. Since IdP 2.3, the basic variant is part of the core. The version included with IdP 2.4 includes some of the new 2.0 features added to the profile.

Client support is (no surprise) the main impediment. We do not expect to see support for ECP added to HTTP clients or development tools unless we as a project undertake that work. To date, most of the work we are aware of has been in Java, mainly in support of the more complicated delegated case, which is a great deal more work to support than the basic version is. However, these libraries would be good starting points for anybody trying to obtain a basic ECP client library.

Links to Known Client Implementations

- Java
    - https://wiki.jasig.org/display/UPM31/Delegated+Authentication+Integration+Library
    - https://forge.switch.ch/redmine/projects/idwsfecp
    - https://github.com/lindqvist/simple-ecp-client
    - shib-http-client - Minimalistic wrapper around the Apache HTTPClient adding Shibboleth support
- Bash
    - simple demonstration client using curl and xsltproc
- Python
    - Production client that includes support for making proxies, using Kerberos or username/password with the IdP, and storing in MyProxy. Works with Python 2.4+.

    - simple demonstration client using Python 2.6+ and the Python lxml toolkit
- Perl
    - http://www.cilogon.org/ecp
- Swift
    - https://github.com/OpenClemson/SwiftECP

Authentication

With a typical IdP, authentication of clients is handled using a web form of some kind to collect a username and password. This approach is not appropriate for most ECP deployments and is not supported by the Shibboleth ECP extension.

Instead, some form of container- or web server-based authentication is required "in front" of the ECP endpoint, essentially the same way the "RemoteUser" login handler that ships with the IdP works. This could be basic authentication with LDAP, Kerberos, etc., or could even be client certificate authentication with TLS. Anything you can use to get REMOTE_USER set will work, but you will often need to set this up in parallel with your IdP's "standard" login process for use by browsers.

The actual exchange between the ECP client and the IdP is based on SOAP, and future ECP support in the IdP will provide for a wider range of authentication options using SOAP and HTTP, and will not require extra setup work; the same back-end verifiers will be used for both the ECP and Browser profiles.

Note that the above refers to the "basic" ECP use case; in the delegation case, the authentication to the IdP is done using a service's certificate (using TLS) and the user's SSO assertion is attached as a SOAP header.

Discovery

When an SP supports more than one choice of IdP, deployers are usually well versed in the challenges created by the need for IdPDiscovery. This problem goes away with ECP; instead, the ECP client needs to be "provisioned" with information about the IdP(s) to support, and will usually store the location of the IdP and could even store the IdP's actual TLS certificate so that the IdP can be strongly verified. This is analogous to the kinds of things sites do to deploy 802.1x wireless authentication.

Implementing Clients

If you're interested in implementing an HTTP client with ECP support (presumably based on an existing, robust HTTP client), here are some basic notes. This is **not** intended to be a substitute for reading the specification itself, just some basic things to think about.

To build this into an HTTP client, there has to be a fair amount of support in these areas:

- cookies
- injecting custom request headers
- basic authentication, preferably without being prompted for it
- access to HTTP response status codes and headers
- raw POST of XML request bodies (not form encoded, in other words)

A basic summary of what the client is responsible for:

1. Send special HTTP headers on every request to advertise ECP support to the SP.
2. Detect the special PAOS content type in a response to trigger the profile.
3. Save off the response location provided by the SP in a SOAP header, as well as optional RelayState.
4. Check for an optional set of "supported IdPs" from the SP in a SOAP header, filtered against IdP(s) the client supports.
5. Route the SOAP body from the SP to the IdP along with the user's credentials, making sure to strongly verify the IdP's identity while doing so. (Note: a list of 100+ useless CAs that come with browsers and so forth are **not** suitable to authenticate an IdP.)
6. Compare the response location provided by the IdP in a SOAP header to the one from the SP and scream bloody murder if they don't match.
7. Route the SOAP body from the IdP back to the SP, along with any RelayState information saved from earlier.
8. Check for a 200 or 302 response from the SP, indicating success or a redirect to the original resource. You can follow a redirect, or simply preserve the original request from the beginning of the process internally and replay it.

At each step, cookies might be involved (particularly with the SP) and need to be handled properly.

Again, this is **not** a specification, merely a summary. Implementers need to read the specification.

What About Non-Web/HTTP Protocols?

If you're working in the non-web space, you'll find that there are really no good solutions for plugging in security today in most protocols, with password-oriented options and client certificates via TLS usually the only options. PAM works well as a back end verification layer, but it is quite password-centric.

The only real frameworks for authentication in most IETF protocols are SASL and GSS-API, and neither is well-proven with technologies beyond a very basic set, primarily Kerberos (in the GSS case) and use of passwords directly in the SASL case. However, there are a variety of proposals circulating to add GSS-API and SASL mechanisms for federated protocols that might stimulate the work required to fix clients and servers.

Most of the proposals are to shoehorn web-based authentication into the flow by launching a browser from the application client and using existing Web SSO protocols like SAML.

Project Moonshot, and the ABFAB IETF WG, are working on the use of EAP (the technology used in wireless authentication) within GSS-API, essentially using EAP and RADIUS in place of SAML.

The Shibboleth Project has also proposed a GSS-API/SASL mechanism that is directly adapted from the ECP profile, and is designed to be fully compatible with IdPs that support it.