RDF Policy-based URI Access Control for Content Authoring on the Social Semantic Web

Joe Presbrey

Email: presbrey@mit.edu

Massachusetts Institute of Technology

Supervisor: Tim Berners-Lee

May 26, 2009

This report describes a new RDF policy-based access control project implementing secured, decentralized client authentication and authorization for the widely used and distributed *Apache HTTP Server*[1] (Apache). The project demonstrates scalable support for collaborative authoring of linked data on a WebDAV-enabled filesystem by all users with a *Web ID*[2]. Produced as my MIT Undergraduate Advanced Project, this prototype consists of two Apache modules called `mod_authn_webid` and `mod_authz_webid` that inline the authentication and authorization components of the project within Apache's core request processing mechanism.

Source code is available at:

`http://dig.csail.mit.edu/2009/mod_authn_webid/`

`http://dig.csail.mit.edu/2009/mod_authz_webid/`

and included for reference at the end of this document.

[1] `http://httpd.apache.org/`

[2] `http://esw.w3.org/topic/WebID`

1 Introduction

The Semantic Web enables global sharing and integration of data across organizational boundaries. To foster adoption and growth, a decentralized platform for accountable collaborative editing of linked data at Internet-scale is necessary. Traditional authentication and access control mechanisms intended for single domains under common administrative control are not sufficient as they rely on maintaining trusted, centralized databases for identity authentication and complementarily isolated stores for authorizations.

This project implements a new access control project demonstrating secured, decentralized client authentication and authorization. Section 2 provides background information describing open, widely-deployed technologies such as Linked Data (and the Semantic Web), FOAF, WebDAV, SSL, and the *FOAF+SSL*[3] protocol. Section 3 describes implementation of two project modules introducing Web ID support to Apache called `mod_authn_webid` and `mod_authz_webid` performing authentication and authorization respectively.

2 Background

2.1 Linked Data

The Semantic Web is an online map of machine-readable data published in a shared document format, eg. N3[4] or RDF[5]. Documents expressing data in any number of shared vocabularies are published at universally resolvable, interlinked URIs producing *linked data* representing a shared, structured graph of knowledge. Prefixes are used to abbreviate common URI paths and can be dereferenced by substituting the associated URI. (The prefix and URI for each vocabulary used in

[3] `http://esw.w3.org/topic/foaf+ssl`

[4] `http://www.w3.org/DesignIssues/Notation3`

[5] `http://www.w3.org/RDF/`

this project are shown in Table 1). A query language called SPARQL[6] can be used to query linked data across diverse sources and formats.

Prefix	*URI*
acl:	`http://www.w3.org/ns/auth/acl#`
cert:	`http://www.w3.org/ns/auth/cert#`
foaf:	`http://xmlns.com/foaf/0.1/`
rdf:	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rsa:	`http://www.w3.org/ns/auth/rsa#`

Table 1: Common Namespace Definitions

2.2 FOAF: Friend-of-a-Friend

RDF has been extended using the `foaf` vocabulary to allow the Semantic Web community to define an open-data social network. This ontology defines links between people in RDF documents that describe them and their properties. In this model, a URI refers to FOAF data representing a person, a group, or their agents and their respective relations. A user's Web ID is a URI referencing a person's FOAF data when used as a globally unique identity.

For example, the linked data found when dereferencing my Web ID, shown in Figure 1 as N3, is a FOAF document expressing my personal identity and its properties using `cert`, `rsa`, and `foaf`. The `cert` and `rsa` vocabularies define the cryptographic properties of the account the FOAF+SSL protocol uses to authenticate me to my Web ID. In this example, `foaf:knows` links me to Tim Berners-Lee. Using SPARQL, the set of people I know can be found with the following query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?friend
```

[6] `http://www.w3.org/TR/rdf-sparql-query/`

```
    WHERE {
      <http://presbrey.mit.edu/foaf#presbrey> foaf:knows ?friend
    }
```

```
@prefix cert: <http://www.w3.org/ns/auth/cert#> .
@prefix rsa: <http://www.w3.org/ns/auth/rsa#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<https://presbrey.mit.edu/foaf#presbrey>
    a foaf:Person ;
    foaf:title "Mr." ;
    foaf:givenname "Joe" ;
    foaf:family_name "Presbrey" ;
    foaf:holdsAccount <#ssl-account-1> ;
    foaf:knows <http://www.w3.org/People/Berners-Lee/card#i>, ... .

<#ssl-account-1>
    a foaf:OnlineAccount .

[]
    cert:identity <#ssl-account-1> ;
    a rsa:RSAPublicKey ;
    rsa:modulus [
        cert:hex "9f6995ad34ca245ee42456169e217ffa0632362c4dca2f6e0456d8a554354ed36574ce
                  5007db9bfac212352ab2d55161d7c6d538779f6225b2e0a1597480014c9a086afd0a90
                  bbcd819f96c030d28a62847759c4bff616a3f10202e31b26a2748b7c589cf2542e3957
                  46634e74b13cabdd5cd2c470d098e412f890a124f483bad17a9a9ef412430586ab4ee2
                  5ccb32d5b640d535568350589f154ea8defec5d98a2fcbd61b0801bb408acf608e1786
                  8dbdd7d905c369aa35af9e93d18f40285e87258d9a10b853548e64d89801162e65eb85
                  fd93455e3e1ee12d25d3f4fb2ae2a8ed255cd970d152637f576305167bc6b5300e3f38
                  37369df27973bb3991f637"
    ] ;
    rsa:public_exponent [
        cert:decimal "65537"
    ] .
```

Figure 1: Web ID FOAF document at <https://presbrey.mit.edu/foaf#presbrey>

2.3 WebDAV

The *WebDAV protocol*[7] extends the *HTTP protocol*[8] to provide methods for collaborative content authoring among remote clients on the web. Many web content publishing services allow WebDAV for various uses such as online storage and backup like Apple's iDisk and collaborative calendaring like the CalDAV interface to Google Calendars. Its also common for a webmaster to use WebDAV to upload files and publish modifications to their website. Client support for the protocol is built into widely-deployed interfaces including Mac OS X Finder, Microsoft Web Folders, Adobe Dreamweaver, and many others. An Apache module called *mod_dav* implements server support for the protocol and provides the WebDAV filesystem store used in this project. Table 2 lists the methods handled by the module provided by the HTTP and WebDAV specifications respectively.

Specification	*Methods*
HTTP	OPTIONS, GET, PUT, DELETE
WebDAV	COPY, MOVE, MKCOL, LOCK, UNLOCK, PROPFIND, PROPPATCH

Table 2: HTTP Methods handled by *mod_dav*

For example, the PUT method can be used to upload a file by sending:

```
PUT /uploads/new_file.txt HTTP/1.1
```

to a WebDAV-enabled HTTP server with the file contents in the HTTP request body.

2.3.1 WebDAV Client Authentication

Typical mechanisms for authentication check a username, password, or other credentials with a file, database, or other distinguished authority and require users to obtain and maintain separate

[7] `http://www.ietf.org/rfc/rfc4918.txt`

[8] `http://www.ietf.org/rfc/rfc2616.txt`

accounts for each service.

Because of the non-standard nature of common decentralized, single-sign-on authentication schemes such as OpenID that require a user to authorize login by handling redirects and acting on confirmation pages, WebDAV services deploying such systems would additionally require modification of client software and user upgrades. In addition to decentralizing user accounts, using FOAF+SSL in this project means maintaining compatibility with clients that already support SSL and fully integrating with existing linked data.

2.4 SSL

SSL (secure sockets layer) is a protocol used widely on the Internet for providing secured communications between clients and servers. To negotiate a connection, a client asks the server to decrypt a secret the client has encrypted with the server's public key. Only a server that knows the private key paired to that public key can decrypt the secret sent by the client. This ensures the privacy of the communication channel.

In addition to using cryptography to ensure the privacy of messages, SSL uses cryptographically signed **certificates** encoding identity data with its public key to ensure the authenticity of messages. During SSL connection negotiation, all SSL servers send a certificate that clients can validate to perform SSL server authentication. Similarly, an SSL client can respond with a certificate for the server to validate to perform SSL client authentication when optionally accepted or required by the server.

Apache's *mod_ssl* module uses an open-source distribution of SSL called the *OpenSSL*[9] cryptography toolkit to provide support for the HTTPS scheme, a SSL-secured HTTP. Figure 2 displays the identity properties and an excerpt of the public key of the SSL client certificate issued to me

[9] `http://openssl.org/`

for HTTPS by MIT.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2315330 (0x235442)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=Massachusetts, O=Massachusetts Institute of Technology,
                OU=Client CA v1
        Validity
            Not Before: Jul 28 17:38:43 2008 GMT
            Not After : Jul 30 17:38:43 2009 GMT
        Subject: C=US, ST=Massachusetts, O=Massachusetts Institute of Technology,
                 OU=Client CA v1, CN=Joseph W Presbrey/emailAddress=presbrey@MIT.EDU
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:aa:e7:16:5d:7b:4c:45:14:27:c5:52:a1:59:ee:
                    .....
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Client, S/MIME
            X509v3 Extended Key Usage:
                E-mail Protection, TLS Web Client Authentication
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Key Identifier:
                CA:A7:BE:0A:D0:6B:DE:DD:5A:39:E7:C3:A9:69:11:26:6C:09:3F:BA
    Signature Algorithm: sha1WithRSAEncryption
        51:89:e1:18:72:1e:57:8e:4a:03:c6:4a:81:50:22:35:1b:3b:
        .....
```

Figure 2: SSL Client Certificate: presbrey@MIT.EDU

2.4.1 SSL Client Authentication

SSL authentication is the process of using the identity data in a valid SSL certificate to treat its holder as acting on behalf of a given security principal according to some mapping. The existing

certificate validation mechanism in *mod_ssl* verifies that:

1. its holder can decrypt a secret encrypted with its public key
2. its common name (CN) matches the expected identity name such as the DNS server name
3. the validator's current date and time is between its issue and expiry dates
4. its signing issuer establishes a chain of trust from the certificate to a trusted root certificate authority (CA) in the validator's certificate store
5. it has not been revoked by its issuing CA

Authentication succeeds when all checks pass allowing the client to act on behalf of a given principal defined by an administratively chosen mapping from a certificate identity component. For example, MIT servers may conveniently interpret the SSL_CLIENT_S_DN_Email field (emailAddress of Subject field in Figure 2) of a client certificate as the principal since each certificate MIT issues corresponds to a single user's Kerberos principal with matching email address.

In this conventional scenario, a domain's trusted CA issues and signs all client certificates and must be used to validate all client certificates to prevent tampering with the mapped component from causing servers to map clients with those certificates to unauthorized identities. The limitations inherent to this domain-based model restricts users of SSL client authentication to domain-specific identities.

2.5 FOAF+SSL

In the FOAF+SSL protocol[10], normal SSL negotiations occur inheriting privacy of messages but a trusted CA is not required to validate SSL identity authenticity and thus message authenticity. The

[10] `http://dig.csail.mit.edu/2009/Papers/SPOT/foaf-ssl-spot2009.pdf`

user claims a Web ID by self-issuing a client certificate including the Web ID URI in the **Subject Alternative Name** component of the certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 212 (0xd4)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=GB, ST=LONDON, L=Wimbledon, O=FOAF.ME, CN=FOAF.ME/emailAddress=ca@foaf.me
        Validity
            Not Before: Apr 28 15:05:02 2009 GMT
            Not After : Apr 28 15:05:02 2010 GMT
        Subject: CN=FOAF ME Temp Acct 77397
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:9f:69:95:ad:34:ca:24:5e:e4:24:56:16:9e:21:
                    .....
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                73:F1:A5:7D:FE:27:59:FE:6F:7F:91:A3:B7:A7:E2:3B:BD:FF:75:01
            X509v3 Authority Key Identifier:
                keyid:42:D9:0E:FB:7A:B4:D4:EF:BE:5F:4B:87:7E:BE:D3:AD:4C:64:C3:33
                DirName:/C=GB/ST=LONDON/L=Wimbledon/O=FOAF.ME/CN=FOAF.ME/emailAddress=ca@foaf.m
                serial:B8:8B:38:D3:8A:D1:75:E2
            X509v3 Subject Alternative Name:
                URI:http://presbrey.mit.edu/foaf#presbrey
    Signature Algorithm: md5WithRSAEncryption
        8a:5d:5b:5b:83:4b:6e:be:a5:eb:3b:11:ff:88:ca:82:1b:79:
        .....
```

Figure 3: SSL Client Certificate: <https://presbrey.mit.edu/foaf#presbrey>

The user publishes the definition for the client certificate and its cryptographic signature as linked data in their personal profile document (FOAF) at their Web ID. For example, Figure 1 links the cryptographic properties of the SSL client certificate in Figure 3 using `foaf:holdsAccount`.

Using the Web ID as the principal identifier for the user avoids the per-domain boundaries of the username/ID token of most centralized authentication models. Figure 3 displays the identity properties and an an excerpt of the public key issued to me for use with my Web ID by a FOAF+SSL demo site: `http://foaf.me/`.

After the following two certificate validation checks,

1. certificate holder can decrypt a secret encrypted with its public key

2. validator's current date and time is between its issue and expiry dates

authentication to Web ID is performed by verifying the client certificate's cryptographic signature with user account data linked from the Web ID asserted by the certificate. A successful match allows the client to act on behalf of their Web ID using its URI as the security principal. FOAF+SSL client authentication using certificates validated without a trusted CA allows users to self-issue certificates and decentralizes authentication.

Another decentralized framework for user authentication called OpenID[11] elimintates the need for multiple usernames across different websites by making the same assurance that a user is the person who wrote the web page at the URI used as their identity. Unlike OpenID which limits the user identity to the reputation of HTML tags in a web site or blog, FOAF+SSL explicitly links FOAF data to the identity URI allowing access decisions to be drawn from a social network or other linked data.

3 Implementation

Authentication treats a user as acting on behalf of a given security principal according to some mapping. Authorization determines which actions the security principal may perform. `mod_authn_webid`

[11] `http://openid.net`

and `mod_authz_webid` perform Web ID authentication and authorization respectively. They are both implemented in C as Apache 2 modules to maximize performance by tightly coupling protocol support to the server core.

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule authn_webid_module modules/mod_authn_webid.so
LoadModule authz_webid_module modules/mod_authz_webid.so
LoadModule ssl_module modules/mod_ssl.so

<VirtualHost _default_:443>
    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
    SSLVerifyClient optional_no_ca
    <Directory />
        <IfModule mod_authn_webid.c>
            AuthType WebID
            Require valid-user
        </IfModule>
        <IfModule mod_authz_webid.c>
            Dav On
        </IfModule>
    </Directory>
</VirtualHost>
```

Figure 4: Sample Apache Configuration: httpd.conf

3.1 `mod_authn_webid`: Authentication

WebID authentication is performed for all requests configured to require authentication of type WebID as demonstrated in the sample Apache configuration in Figure 4. *mod_ssl* negotiates all SSL connections and optionally performs SSL client certificate validation less the checks verifying with a CA. During the `check_user_id` phase of the Apache request handling process, the module requests the client certificate data from *mod_ssl* and calculates the `cert:hex` and `cert:decimal` data it intends to verify with the Web ID data.

The procedure can be summarized as follows:

1. look for a Web ID in the Subject Alternative Name extension by importing and using `ssl_ext_lookup` from `mod_ssl`
2. calculate the client certificate's cryptographic signature from the certificate by importing and using `ssl_var_lookup` from `mod_ssl`
3. query the claimed Web ID linked data for valid certificate signatures using Redland RDF library[12]
4. compare the signature calculated to queried signatures to allow or deny authentication

The following SPARQL query is used to list authoritative user account data from the Web ID:

```
SELECT ?mod_hex WHERE {
    ?key rdf:type rsa:RSAPublicKey.
    ?key rsa:public_exponent ?exp.
    ?key rsa:modulus ?mod.
    ?key cert:identity ?a.
    ?exp cert:decimal \"%d\".
    ?mod cert:hex ?mod_hex.
    ?a rdf:type foaf:OnlineAccount.
    %s foaf:holdsAccount ?a.
}
```

The `rsa:modulus` comparison manually extracts only hex digits to support users with whitespace in valid Web ID data that would otherwise fail a `strcmp` literal-string comparison to the OpenSSL binary modulus from the certificate. On success, the module populates the authenticated user field with the WebID and yields request processing back to Apache for the remainder of the request.

3.1.1 Performance

A benchmark of this procedure was performed using the Apache-provided ApacheBench program against 4 different Apache configurations running in single-process mode with the worker MPM on

[12] `http://librdf.org/`

a Quad-Core AMD Opteron 2350 2GHz Xen VM guest with 512MB of RAM. In each of the tests, ApacheBench was timed performing 1000 serial requests for a null-document. Shown in Table 3, the calculated result for each test configuration is average request time over 5 test runs.

Test Configuration	*Request Time (ms/request)*
without client certificate, without mod_authn_foafssl	4.2
without client certificate, with mod_authn_foafssl	4.43
with Web ID client certificate, without mod_authn_foafssl	4.63
with Web ID client certificate, with mod_authn_foafssl	11.31

Table 3: Benchmark results for mod_authn_webid

Caching linked data at Web ID URIs is not yet implemented but would substantially improve the performance of persistent HTTP connections and could help protect against DoS attacks launched against a Web ID target URI.

3.2 `mod_authz_webid`: Authorization

mod_authz_webid authorization checks occur during the `auth_checker` phase of the Apache request handling process. HTTP and WebDAV methods from Table 2 are mapped to one of three simple `acl:mode`s (Read, Write, Control) in the Web Access Control[13] vocabulary[14] according to Table 4. COPY and MOVE requests are treated as special cases due to the destination URI in the HTTP header: `Destination`. For COPYs, the request URI is checked for acl:Read and the destination URI for acl:Write. For MOVEs, both URIs are checked for acl:Write.

Access control list (ACL) acl:Authorization policies are stored in an .htaccess-like metadata file local to the protected resources called `,meta.n3`. In each acl:Authorization, `acl:accessTo` links to the resource for a given policy rule (or `acl:accessToClass` to a class of resources), `acl:agent` links to an authenticated user of the rule (or `acl:agentClass` to a class of users), `acl:mode` maps

[13] `http://esw.w3.org/topic/WebAccessControl`

[14] `http://www.w3.org/ns/auth/acl`

acl:mode	*Methods*
acl:Read	OPTIONS, GET, POST*, PROPFIND * interpretation of POST data must also indicate Read, e.g SPARQL SELECT
acl:Write	PUT, DELETE, PROPPATCH, MKCOL, COPY*, MOVE** * two checks: Request URI acl:Read, Destination URI acl:Write ** two checks: Request URI acl:Write, Destination URI acl:Write
acl:Control	*any acl:Write to an ACL URI*

Table 4: HTTP method to acl:mode mappings

HTTP methods for access, and `acl:defaultForNew` links the rules new resources should inherit. In Figure 5 in N3, an example ACL policy grants all acl:mode access modes for the current directory and all new URIs created to my personal Web ID.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

[]
    a acl:Authorization ;
    acl:accessTo <.> ;
    acl:defaultForNew <.> ;
    acl:agent <http://presbrey.mit.edu/foaf#presbrey> ;
    acl:mode acl:Control, acl:Read, acl:Write .
```

Figure 5: `,meta.n3` policy for private access

Access is authorized by successful SPARQL query of the following:

```
SELECT ?rule WHERE {
    ?rule rdf:type acl:Authorization ;
    acl:accessTo <%s> ;
    acl:mode acl:%s ;
    acl:agent %s .
}
```

The `acl:accessTo` and `acl:agent` predicates are replaced in the SPARQL with `acl:accessToClass` and `acl:agentClass` as needed. Additionally, a generic "anyone" accessor is defined as `acl:agentClass foaf:Agent` and can be queried:

```
SELECT ?rule WHERE {
    ?rule rdf:type acl:Authorization ;
    acl:accessTo <%s> ;
    acl:mode acl:%s ;
    acl:agentClass <http://xmlns.com/foaf/0.1/Agent> .
}
```

4 Related Work

Two high-level language implementations of FOAF+SSL exist in Perl [15] and PHP [16] that can be integrated into servers already supporting these languages for Web ID authentication support. No other known implementations of Web ID authorization currently exist.

5 Conclusion

The FOAF-walking project I produced with the Decentralized Information Group at MIT in 2007 established precendence for crawling a network of FOAF documents to define a white-list for controlling comment contributions to blogs[17]. In many ways, this project follows up this work by allowing RDF and FOAF networking to define a more extensible access control system that can be applied to any URI space.

Implementation of this for Apache demonstrates it is possible to streamline this in a production web server with hopes to drive wide adoption to support the ultimate goal of providing a single sign-on solution that makes life easiest for users while maintaining a fully decentralized architecture in which identities, data storage, and applications can all be independent and managed by different sites.

[15] http://search.cpan.org/~tobyink/CGI-Auth-FOAF_SSL/
[16] https://foaf.me/testLibAuthentication.php
[17] http://dig.csail.mit.edu/breadcrumbs/node/206

mod_authn_webid.c


```
/* mod_authn_webid
 * WebID FOAF+SSL authentication module for Apache 2
 *
 * Joe Presbrey <presbrey@csail.mit.edu>
 *
 * $Id: mod_authn_webid.c 26058 2009−05−14 07:04:26Z presbrey $
 */

#include "apr_strings.h"
#define APR_WANT_STRFUNC
#include "apr_want.h"

#include "ap_config.h"
#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_request.h"

#include "mod_auth.h"
#include "mod_ssl.h"

#include "openssl/ssl.h"
#include "redland.h"

static APR_OPTIONAL_FN_TYPE(ssl_var_lookup) *ssl_var_lookup;
static APR_OPTIONAL_FN_TYPE(ssl_ext_lookup) *ssl_ext_lookup;

typedef struct {
    int authoritative;
} authn_webid_config_rec;

static void *
create_authn_webid_dir_config(apr_pool_t *p, char *dirspec) {
    authn_webid_config_rec *conf = apr_pcalloc(p, sizeof(*conf));

    conf->authoritative = −1;
    return conf;
}

static void *
merge_authn_webid_dir_config(apr_pool_t *p, void *parent_conf, void *newloc_conf
) {
    authn_webid_config_rec *pconf = parent_conf, *nconf = newloc_conf,
    *conf = apr_pcalloc(p, sizeof(*conf));

    conf->authoritative = (nconf->authoritative != −1) ?
        nconf->authoritative : pconf->authoritative;
    return conf;
}

static const command_rec
authn_webid_cmds[] = {
    AP_INIT_FLAG("AuthWebIDAuthoritative", ap_set_flag_slot,
                 (void *)APR_OFFSETOF(authn_webid_config_rec, authoritative),
                 OR_AUTHCFG,
                 "Set to 'Off' to allow access control to be passed along to "
                 "lower modules if the WebID is not known to this module"),
    {NULL}
```

mod_authn_webid.c


```
};

module AP_MODULE_DECLARE_DATA authn_webid_module;

static int
hex_or_x(int c) {
    if (c >= '0' && c <= '9')
        return c;
    c |= 32;
    if (c >= 'a' && c <= 'f')
        return c;
    return 'x';
}

static int
matches_pkey(unsigned char *s, char *pkey) {
    if (s == NULL || pkey == NULL)
        return 0;
    unsigned int s_s = strlen(s);
    unsigned int s_pkey = strlen(pkey);
    unsigned int fc, pc, j, k = 0;

    for (j = 0; j < s_s; j++) {
        if ((fc = hex_or_x(s[j])) == 'x')
            continue;
        pc = hex_or_x(pkey[k]);
        if (fc != pc)
            break;
        k++;
    }
    if (k == s_pkey)
        return 1;
    return 0;
}

static int
authenticate_webid_user(request_rec *request) {
    int r = 0;
    authn_webid_config_rec *conf =
        ap_get_module_config(request->per_dir_config, &authn_webid_module);
    if (!conf->authoritative) r = DECLINED;
    else r = HTTP_UNAUTHORIZED;

    /* Check for AuthType WebID */
    const char *current_auth = ap_auth_type(request);
    if (!current_auth || strcasecmp(current_auth, "WebID") != 0) {
        return DECLINED;
    }
    request->ap_auth_type = "WebID";

    const char *subjAltName;
    char *pkey_n = NULL;
    char *pkey_e = NULL;

    subjAltName = ssl_ext_lookup(request->pool, request->connection, 1, "2.5.29.17
");
    if (subjAltName != NULL) {
        if (strncmp(subjAltName, "URI:", 4) != 0)
            subjAltName = NULL;
```

mod_authn_webid.c

```
        else
            subjAltName = subjAltName+4;
    }

    char *c_cert = NULL;
    BIO *bio_cert = NULL;
    X509 *x509 = NULL;
    EVP_PKEY *pkey = NULL;
    RSA *rsa = NULL;

    BIO *bio = NULL;
    BUF_MEM *bptr = NULL;

    if (NULL != (c_cert = ssl_var_lookup(request->pool, request->server, request
->connection, request, "SSL_CLIENT_CERT"))
        && NULL != (bio_cert = BIO_new_mem_buf(c_cert, strlen(c_cert)))
        && NULL != (x509 = PEM_read_bio_X509(bio_cert, NULL, NULL, NULL))
        && NULL != (pkey = X509_get_pubkey(x509))
        && NULL != (rsa = EVP_PKEY_get1_RSA(pkey))) {

        // public key modulus
        bio = BIO_new(BIO_s_mem());
        BN_print(bio, rsa->n);
        BIO_get_mem_ptr(bio, &bptr);
        pkey_n = apr_pstrndup(request->pool, bptr->data, bptr->length);
        BIO_free(bio);

        // public key exponent
        bio = BIO_new(BIO_s_mem());
        BN_print(bio, rsa->e);
        BIO_get_mem_ptr(bio, &bptr);
        pkey_e = apr_pstrndup(request->pool, bptr->data, bptr->length);
        BIO_free(bio);
    } else {
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID: invalid client certific
ate");
    }

    if (rsa)
        RSA_free(rsa);
    if (pkey)
        EVP_PKEY_free(pkey);
    if (bio_cert)
        BIO_free(bio_cert);

    librdf_world *rdf_world = NULL;
    librdf_storage *rdf_storage = NULL;
    librdf_model *rdf_model = NULL;
    librdf_query *rdf_query = NULL;
    librdf_query_results *rdf_query_results = NULL;

    if (subjAltName != NULL
        && pkey_n != NULL && pkey_e != NULL) {
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID: subjectAltName = %
s", subjAltName);
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID: client pkey.n  = %s"
, pkey_n);
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID: client pkey.e  = %s"
, pkey_e);
```

mod_authn_webid.c


```
        rdf_world = librdf_new_world();
        if (rdf_world != NULL) {
            librdf_world_open(rdf_world);
            rdf_storage = librdf_new_storage(rdf_world, "uri", subjAltName, NULL)
;
        }
        if (rdf_storage != NULL) rdf_model = librdf_new_model(rdf_world, rdf_sto
rage, NULL);
        char *c_query = apr_psprintf(request->pool,
            " PREFIX rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>"
            " PREFIX cert: <http://www.w3.org/ns/auth/cert#>"
            " PREFIX rsa: <http://www.w3.org/ns/auth/rsa#>"
            " SELECT ?mod_hex WHERE {"
            " ?key rdf:type rsa:RSAPublicKey."
            " ?key rsa:public_exponent ?exp."
            " ?key rsa:modulus ?mod."
            " ?exp cert:decimal \"%d\"."
            " ?mod cert:hex ?mod_hex."
            " }", apr_strtoi64(pkey_e, NULL, 16));
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID: query = %s", c_q
uery);
        if (rdf_model != NULL) rdf_query = librdf_new_query(rdf_world, "sparql",
NULL, (unsigned char *)c_query, NULL);

        if (rdf_query != NULL) {
            rdf_query_results = librdf_query_execute(rdf_query, rdf_model);
            if (rdf_query_results != NULL) {
                if (librdf_query_results_get_count(rdf_query_results) > 0) {
                    librdf_node *rdf_node;
                    unsigned char *mod_hex;
                    while (NULL != (rdf_node = librdf_query_results_get_binding_
value_by_name(rdf_query_results, "mod_hex"))) {
                        mod_hex = librdf_node_get_literal_value(rdf_node);
                        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebI
D: modulus = %s", mod_hex);
                        if (matches_pkey(mod_hex, pkey_n)) {
                            request->user = apr_psprintf(request->pool, "<%s>",
subjAltName);
                            r = OK;
                            break;
                        }
                        librdf_free_node(rdf_node);
                        if (librdf_query_results_next(rdf_query_results)) break;
                    }
                }
                librdf_free_query_results(rdf_query_results);
            } else
                ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, request, "WebID: librdf_quer
y_execute returned NULL");
            librdf_free_query(rdf_query);
        } else
            ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, request, "WebID: librdf_new_query
 returned NULL");

        if (rdf_model) librdf_free_model(rdf_model);
        if (rdf_storage) librdf_free_storage(rdf_storage);
        if (rdf_world) librdf_free_world(rdf_world);
    }
```

mod_authn_webid.c


```
    if (conf->authoritative && r != OK) {
        if (subjAltName != NULL) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO | APLOG_TOCLIENT, 0, request, "
WebID authentication failed: <%s>. Request URI: %s", subjAltName, request->uri);
        } else {
            ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID authentication f
ailed. Request URI: %s", request->uri);
        }
    }
    else if (r == OK)
        ap_log_rerror(APLOG_MARK, APLOG_DEBUG, 0, request, "WebID authentication succee
ded: <%s>. Request URI: %s", subjAltName, request->uri);
    return r;
}

static void
import_ssl_func() {
    ssl_var_lookup = APR_RETRIEVE_OPTIONAL_FN(ssl_var_lookup);
    ssl_ext_lookup = APR_RETRIEVE_OPTIONAL_FN(ssl_ext_lookup);
}

static void
register_hooks(apr_pool_t *p) {
    ap_hook_check_user_id(authenticate_webid_user, NULL, NULL, APR_HOOK_MIDDLE);
    ap_hook_optional_fn_retrieve(import_ssl_func, NULL, NULL, APR_HOOK_MIDDLE);
}

module AP_MODULE_DECLARE_DATA
authn_webid_module = {
    STANDARD20_MODULE_STUFF,
    create_authn_webid_dir_config,
    merge_authn_webid_dir_config,
    NULL,
    NULL,
    authn_webid_cmds,
    register_hooks
};
```

mod_authz_webid.c


```
/* mod_authz_webid
 * WebID authorization module for Apache 2
 *
 * Joe Presbrey <presbrey@csail.mit.edu>
 *
 * $Id: mod_authz_webid.c 26098 2009−05−18 07:55:19Z presbrey $
 */

#include "apr_strings.h"

#include "ap_config.h"
#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_protocol.h"
#include "http_request.h"

#include <redland.h>

#define URI__NS_ACL      "<http://www.w3.org/ns/auth/acl#>"
#define URI__NS_RDF      "<http://www.w3.org/1999/02/22−rdf−syntax−ns#>"
#define URI__FOAF_AGENT "<http://xmlns.com/foaf/0.1/Agent>"

#define WEBID_ACL_FNAME          ",meta.n3"
#define WEBID_ACCESS_INVALID    0
#define WEBID_ACCESS_READ       0x1
#define WEBID_ACCESS_WRITE      0x2
#define WEBID_ACCESS_CONTROL    0x4

#define WEBID_M_READ \
    (AP_METHOD_BIT << M_GET | \
     AP_METHOD_BIT << M_POST | \
     AP_METHOD_BIT << M_CONNECT | \
     AP_METHOD_BIT << M_OPTIONS | \
     AP_METHOD_BIT << M_TRACE | \
     AP_METHOD_BIT << M_PROPFIND)

#define WEBID_M_WRITE \
    (AP_METHOD_BIT << M_PUT | \
     AP_METHOD_BIT << M_DELETE | \
     AP_METHOD_BIT << M_PATCH | \
     AP_METHOD_BIT << M_PROPPATCH | \
     AP_METHOD_BIT << M_MKCOL | \
     AP_METHOD_BIT << M_COPY | \
     AP_METHOD_BIT << M_MOVE | \
     AP_METHOD_BIT << M_LOCK | \
     AP_METHOD_BIT << M_UNLOCK | \
     AP_METHOD_BIT << M_VERSION_CONTROL | \
     AP_METHOD_BIT << M_CHECKOUT | \
     AP_METHOD_BIT << M_UNCHECKOUT | \
     AP_METHOD_BIT << M_CHECKIN | \
     AP_METHOD_BIT << M_UPDATE | \
     AP_METHOD_BIT << M_LABEL | \
     AP_METHOD_BIT << M_REPORT | \
     AP_METHOD_BIT << M_MKWORKSPACE | \
     AP_METHOD_BIT << M_MKACTIVITY | \
     AP_METHOD_BIT << M_BASELINE_CONTROL | \
     AP_METHOD_BIT << M_MERGE)
```

mod_authz_webid.c


```
#define SPARQL_URI_MODE_AGENT \
     " PREFIX rdf: " URI__NS_RDF \
     " PREFIX acl: " URI__NS_ACL \
     " SELECT ?rule WHERE {" \
     " ?rule rdf:type acl:Authorization ;" \
     "     acl:%s <%s> ;" \
     "     acl:mode acl:%s ;" \
     "     acl:agent %s ." \
     " }"

#define SPARQL_URI_MODE_AGENTCLASS \
     " PREFIX rdf: " URI__NS_RDF \
     " PREFIX acl: " URI__NS_ACL \
     " SELECT ?rule WHERE {" \
     " ?rule rdf:type acl:Authorization ;" \
     "     acl:%s <%s> ;" \
     "     acl:mode acl:%s ;" \
     "     acl:agentClass ?class ." \
     " %s rdf:type ?class ." \
     " }"

#define SPARQL_URI_MODE_WORLD \
     " PREFIX rdf: " URI__NS_RDF \
     " PREFIX acl: " URI__NS_ACL \
     " SELECT ?rule WHERE {" \
     " ?rule rdf:type acl:Authorization ;" \
     "     acl:%s <%s> ;" \
     "     acl:mode acl:%s ;" \
     "     acl:agentClass " URI__FOAF_AGENT "." \
     " }"

typedef struct {
    int authoritative;
} authz_webid_config_rec;

static void *
create_authz_webid_dir_config(apr_pool_t *p, char *dirspec) {
    authz_webid_config_rec *conf = apr_palloc(p, sizeof(*conf));

    conf->authoritative = 1;
    return conf;
}

static const command_rec
authz_webid_cmds[] = {
    AP_INIT_FLAG("AuthzWebIDAuthoritative", ap_set_flag_slot,
                 (void *)APR_OFFSETOF(authz_webid_config_rec, authoritative),
                 OR_AUTHCFG,
                 "Set to 'Off' to allow access control to be passed along to "
                 "lower modules if the WebID is not authorized by this module"),
    {NULL}
};

module AP_MODULE_DECLARE_DATA authz_webid_module;

static int
http_status_code(request_rec *r, int status_code) {
    authz_webid_config_rec *conf = ap_get_module_config(r->per_dir_config,
```

mod_authz_webid.c

```
                                                        &authz_webid_module);
    if (status_code != OK && !conf->authoritative) {
        return DECLINED;
    } else if (status_code != OK) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
                      "access to %s failed, reason: WebID %s does not meet "
                      "ACL requirements to be allowed access",
                      r->uri, r->user);
    }
    return status_code;
}

static int
query_results(request_rec *r, librdf_world *w, librdf_model *m, char *query) {
    int ret = 0;
    librdf_query *q = NULL;
    librdf_query_results *qr = NULL;

    if ((q = librdf_new_query(w, "sparql", NULL, (unsigned char *)query, NULL)) !
= NULL) {
        if ((qr = librdf_query_execute(q, m)) != NULL) {
            ret = librdf_query_results_get_count(qr);
            librdf_free_query_results(qr);
        } else
            ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_query_execute returned NU
LL");
        librdf_free_query(q);
    } else
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_query returned NULL");
    return ret;
}

static int
check_request_acl(request_rec *r, int req_access) {
    char *dir_path, *acl_path;
    apr_finfo_t acl_finfo;

    const char *req_uri, *dir_uri, *acl_uri, *access;
    const char *port, *par_uri, *req_file;

    librdf_world *rdf_world = NULL;
    librdf_storage *rdf_storage = NULL;
    librdf_model *rdf_model = NULL;
    librdf_parser *rdf_parser = NULL;
    librdf_uri *rdf_uri_acl = NULL,
               *rdf_uri_base = NULL;

    int ret = HTTP_FORBIDDEN;

    // dir_path: parent directory of request filename
    // acl_path: absolute path to request ACL
    dir_path = ap_make_dirstr_parent(r->pool, r->filename);
    acl_path = ap_make_full_path(r->pool, dir_path, WEBID_ACL_FNAME);

    if (apr_filepath_merge(&acl_path, NULL, acl_path, APR_FILEPATH_NOTRELATIVE,
r->pool) != APR_SUCCESS) {
        ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r,
                      "Module bug?  Request filename path %s is invalid or "
                      "or not absolute for uri %s",
```

mod_authz_webid.c


```
                        r->filename, r->uri);
        return HTTP_FORBIDDEN;
    }

    // acl_path: 403 if missing
    if ((apr_stat(&acl_finfo, acl_path, APR_FINFO_TYPE, r->pool) != APR_SUCCESS)
 ||
        (acl_finfo.filetype != APR_REG)) {
        return HTTP_FORBIDDEN;
    }

    // req_uri: fully qualified URI of request filename
    // dir_uri: fully qualified URI of request filename parent
    // acl_uri: fully qualified URI of request filename ACL
    // access: ACL URI of requested access
    port = ap_is_default_port(ap_get_server_port(r), r)
           ? "" : apr_psprintf(r->pool, ":%u", ap_get_server_port(r));
    req_uri = apr_psprintf(r->pool, "%s://%s%s%s%s",
                           ap_http_scheme(r), ap_get_server_name(r), port,
                           (*r->uri == '/') ? "" : "/",
                           r->uri);
    par_uri = ap_make_dirstr_parent(r->pool, r->uri);
    dir_uri = apr_psprintf(r->pool, "%s://%s%s%s%s",
                           ap_http_scheme(r), ap_get_server_name(r), port,
                           (*par_uri == '/') ? "" : "/",
                           par_uri);
    acl_uri = ap_make_full_path(r->pool, dir_uri, WEBID_ACL_FNAME);

    if (req_access == WEBID_ACCESS_READ) {
        access = "Read";
    } else if (req_access == WEBID_ACCESS_WRITE) {
        if ((req_file = strrchr(r->filename, '/')) != NULL &&
            strcmp(++req_file, WEBID_ACL_FNAME) == 0)
            access = "Control";
        else
            access = "Write";
    } else {
        access = "Control";
    }

    /*
    ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r,
                  "%s (%s) %s | URI: %s | DIR: %s (%s) | ACL: %s (%s) | status:
%d",
                  r->method, access, r->uri, req_uri, dir_uri, dir_path, acl_uri
, acl_path, r->status);
    */

    if ((rdf_world = librdf_new_world()) != NULL) {
        librdf_world_open(rdf_world);
        if ((rdf_storage = librdf_new_storage(rdf_world, "memory", NULL, NULL))
!= NULL) {
            if ((rdf_model = librdf_new_model(rdf_world, rdf_storage, NULL)) !=
NULL) {
                if ((rdf_parser = librdf_new_parser(rdf_world, "turtle", NULL, NUL
L)) != NULL) {
                    if ((rdf_uri_base = librdf_new_uri(rdf_world, (unsigned char
*)acl_uri)) != NULL) {
                        if ((rdf_uri_acl = librdf_new_uri_from_filename(rdf_worl
```

mod_authz_webid.c

```
d, acl_path)) != NULL) {
                                if (!librdf_parser_parse_into_model(rdf_parser, rdf_
uri_acl, rdf_uri_base, rdf_model)) {
                                    if (query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_AGENT,
 "accessTo", req_uri, access, r->user)) > 0 || \
                                        query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_AGENTC
LASS, "accessTo", req_uri, access, r->user)) > 0 || \
                                        query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_WORLD,
 "accessTo", req_uri, access)) > 0 || \
                                        query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_AGENT,
 "defaultForNew", dir_uri, access, r->user)) > 0 || \
                                        query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_AGENTC
LASS, "defaultForNew", dir_uri, access, r->user)) > 0 || \
                                        query_results(r, rdf_world, rdf_model,
                                        apr_psprintf(r->pool, SPARQL_URI_MODE_WORLD,
 "defaultForNew", dir_uri, access)) > 0)
                                        ret = OK;
                                } else
                                    ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf
_parser_parse_into_model failed");
                                librdf_free_uri(rdf_uri_acl);
                            } else
                                ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new
_uri_from_filename returned NULL");
                            librdf_free_uri(rdf_uri_base);
                        } else
                            ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_uri re
turned NULL");
                        librdf_free_parser(rdf_parser);
                    } else
                        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_parser retur
ned NULL");
                    librdf_free_model(rdf_model);
                } else
                    ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_model returned N
ULL");
                librdf_free_storage(rdf_storage);
            } else
                ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_storage returned NULL
");
            librdf_free_world(rdf_world);
        } else
            ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "librdf_new_world returned NULL");

        return ret;
}

static int
webid_auth_checker(request_rec *r) {
    int is_initial_req, req_access, req_method, ret;
    const char *req_dest;

    request_rec *r_dest;
    apr_uri_t apr_uri;
```

mod_authz_webid.c


```
    if (r->filename == NULL) {
        ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r,
                      "Module bug?  Request filename is missing for URI %s", r->uri);
        return http_status_code(r, OK);
    }

    if (r->user == NULL || strlen(r->user) == 0) {
        return http_status_code(r, HTTP_FORBIDDEN);
    }

    // req_access: Read, Write, or Control
    is_initial_req = ap_is_initial_req(r);
    req_access = WEBID_ACCESS_INVALID;
    req_method = (AP_METHOD_BIT << r->method_number);

    if (is_initial_req && r->method_number == M_COPY) {
        // allow COPY of a readonly source URI
        // - target URI check happens by subrequest
        req_access = WEBID_ACCESS_READ;

    } else if (req_method == (req_method & WEBID_M_READ)) {
        // check the acl:Read method bitmask
        req_access = WEBID_ACCESS_READ;

    } else if (req_method == (req_method & WEBID_M_WRITE)) {
        // check the acl:Write method bitmask
        // - writes to ACL URIs are acl:Control (handled internally)
        req_access = WEBID_ACCESS_WRITE;

    } else {
        // unhandled methods require acl:Control
        req_access = WEBID_ACCESS_CONTROL;
    }

    ret = HTTP_FORBIDDEN;

    if (is_initial_req && (r->method_number == M_COPY || r->method_number == M_M
OVE)) {
        req_dest = apr_table_get(r->headers_in, "Destination");
        if (req_dest == NULL) {
            const char *nscp_host = apr_table_get(r->headers_in, "Host");
            const char *nscp_path = apr_table_get(r->headers_in, "New-uri");
            if (nscp_host != NULL && nscp_path != NULL)
                req_dest = apr_psprintf(r->pool, "http://%s%s", nscp_host, nscp_pa
th);
        }
        if (req_dest != NULL) {
            if ((apr_uri_parse(r->pool, req_dest, &apr_uri) == APR_SUCCESS) &&
                (apr_uri.scheme != NULL && strcmp(apr_uri.scheme, ap_http_scheme
(r)) == 0) &&
                (apr_uri.hostname != NULL && strcmp(apr_uri.hostname, ap_get_ser
ver_name(r)) == 0)) {
                req_dest = apr_uri_unparse(r->pool, &apr_uri, APR_URI_UNP_OMITSI
TEPART);
                r_dest = ap_sub_req_method_uri(r->method, req_dest, r, NULL);
                if ((ret = check_request_acl(r, req_access)) == OK)
                    ret = check_request_acl(r_dest, WEBID_ACCESS_WRITE);
            } else {
```

mod_authz_webid.c


```
                ret = HTTP_BAD_GATEWAY;
            }
        }
    } else {
        ret = check_request_acl(r, req_access);
    }

    return http_status_code(r, ret);
}

static int
webid_log_transaction(request_rec *r) {
    if (ap_is_HTTP_SUCCESS(r->status)) {
        if (r->method_number == M_PUT) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, "%s %s | status: %d", r->meth
od, r->uri, r->status);
        } else if (r->method_number == M_DELETE) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, "%s %s | status: %d", r->meth
od, r->uri, r->status);
        } else if (r->method_number == M_MKCOL) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, "%s %s | status: %d", r->meth
od, r->uri, r->status);
        } else if (r->method_number == M_COPY) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, "%s %s | status: %d", r->meth
od, r->uri, r->status);
        } else if (r->method_number == M_MOVE) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, "%s %s | status: %d", r->meth
od, r->uri, r->status);
        }
    }
    return DECLINED;
}

static void
register_hooks(apr_pool_t *p) {
    static const char * const aszPost[]={ "mod_authz_user.c", NULL };

    ap_hook_auth_checker(webid_auth_checker, NULL, aszPost, APR_HOOK_MIDDLE);
    ap_hook_log_transaction(webid_log_transaction, NULL, NULL, APR_HOOK_MIDDLE);
}

module AP_MODULE_DECLARE_DATA
authz_webid_module = {
    STANDARD20_MODULE_STUFF,
    create_authz_webid_dir_config, /* dir config creater */
    NULL,                          /* dir merger --- default is to override */
    NULL,                          /* server config */
    NULL,                          /* merge server config */
    authz_webid_cmds,              /* command apr_table_t */
    register_hooks                 /* register hooks */
};
```