API Key Access Control

- Introduction and Context
- Implementation
    - New Classes
    - Configuration
- API Key Usage
    - URL Query String Parameters
    - HTTP Authorization Header with custom scheme
- Appendix A (Access Control Implementation)

Update

A more robust, and proper authN/authZ mechanism is being explored.

Introduction and Context

Certain RESTful admin functions of the IdP e.g. account lockout, by default use IP authentication. Consequently, a Cross-Site Request Forgery (CSRF) attack would exploit the authorisation granted to the IP address of the users network host - from which the browser makes the request - to invoke certain admin functions cross site.

The IdP access policy implementation is highly customisable. Here, we describe a new access control mechanism based on a pre-shared secret or API key.

An API key would not be forgeable by an attacker (unless the key was exposed by other means), and would not (depending on the scheme used e.g. Basic Auth is cached by browsers) be re-sent by the browser when requesting the resource. As a result, this should prevent CSRF attacks.

Despite this, the simple API key mechanism described here **may** not be seen as a robust RESTful API security mechanism for the following reasons:

- Key management is not easy:
    - Keys are not automatically rotated, they are manually managed in the IdP config. This would also require communication with those whom the keys are shared.
    - Keys are likely to be *very* long lived, widening the attack window.
    - Revoking keys requires config changes and a restart of the IdP.
    - The key does not carry fine grained role information - only the admin role. This single role, for the same key, can not be adjusted during its lifetime.
- Keys are sent in clear text, are not signed or encrypted, and so require SSL/TLS everywhere they are communicated.
- High key entropy needs to be enforced so they are not guessable - this can be validated when set.
- The API key always carries the same authentication and authorization authority - even though they should strictly be used for authentication, and authorization determine by another means.e.g. some form of access management.
- As with any pre-shared secret, clients must be trusted to manage the key appropriately i.e. not to disclose the key.
- The raw key is exposed in the IdP's config file.
    - This could be pre-hashed by the deployer, but this adds complexity around algorithm usage etc. which is not currently in scope of the implementation.

Despite this, an API key could still provide a stop-gap solution for the IdP's RESTFul admin endpoints, it:

- should 'help prevent' CSRF attacks that could take advantage of IP address authentication.
- is simple for the deployer to configure.
- is assumed most actors using the IdP's admin endpoints would be 'internal' users or systems, and not uncontrollable end-users.
    - and probably a very limited number, thus simplifying key management.

Implementation

The implementation can be found on my personal Git repository (git@git.shibboleth.net:philsmart/java-support), on the feature branch `feature/access-control-apikey`.

New Classes

There is only one new class and it's associated JUnit test class:

- `APIKeyAccessControl` - an implementation of the `AccessControl` interface.
    - Extracts the API key from either the HTTP Authorization header (using the Authentication Framework [rfc7235] with custom scheme), or the URL query string parameters.
        - Defaults to HTTP Header extraction. This is arguably safer than URL extraction, as URLs are saved in browser histories, server logs, and can be cut and paste into emails etc.
        - The custom authorization scheme should be defined, is not currently.

    - The HTTP Authorization header mechanism only supports preemptive authentication / authorization. It does not support challenge response e.g. `WWW-Authenticate`.
- The API Key has to be created by the deployer, and set in the access-control configuration.
    - A customisable key validation policy predicate can be injected to validate the key against a predefined policy on initialisation. A simple key length policy is configured by default.

Configuration

The APIKeyAccessControl bean is defined in the `access-control-system.xml` file.

```
<bean id="shibboleth.APIKeyAccessControl" abstract="true"
        class="net.shibboleth.utilities.java.support.security.APIKeyAccessControl" />
```

This can then be added to the `shibboleth.AccessControlPolicies` map e.g.

```
<entry key="AccessByApiKey">
            <bean id="AccessByApiKey" parent="shibboleth.APIKeyAccessControl"
            p:apiKeyIn="#{T(net.shibboleth.utilities.java.support.security.APIKeyAccessControl.APIKeyIn).
HEADER}"
            p:apiKey="astrongapikey"/>
</entry>
```

Then configured as a policy on an appropriate admin function e.g. in `general-admin.xml`:

```
<bean parent="shibboleth.AdminFlow"
            c:id="http://shibboleth.net/ns/profiles/lockout-manager"
            p:loggingId="Lockout"
            p:policyName="AccessByApiKey" />
```

API Key Usage

The API Key is either placed - for every request - in the URL query string parameters, or a HTTP Authorization Header.

URL Query String Parameters

The API Key is added to the URL query parameters e.g.:

```
https://localhost:8443/idp/profile/admin/lockout/shibboleth.authn.Password.AccountLockoutManager/ss!127.0.0.1?
shibapikey=thisisanapikey
```

HTTP Authorization Header with custom scheme

The HTTP Authorization header with a custom scheme is used to communicate the API key. The header syntax is:

```
Authorization: SHIB-API-KEY <key>
```

e.g.

```
Authorization: SHIB-API-KEY thisisanapikey
```

Note, a bypass for http headers could exist if the user agent was auto-injecting headers when requesting certain URLs e.g. a http header modification browser plugin.

Appendix A (Access Control Implementation)

```
/*
 * Licensed to the University Corporation for Advanced Internet Development,
 * Inc. (UCAID) under one or more contributor license agreements.  See the
 * NOTICE file distributed with this work for additional information regarding
 * copyright ownership. The UCAID licenses this file to You under the Apache
 * License, Version 2.0 (the "License"); you may not use this file except in
 * compliance with the License.  You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package net.shibboleth.utilities.java.support.security;

import java.util.Enumeration;
import java.util.function.Predicate;

import javax.annotation.Nonnull;
import javax.annotation.Nullable;
import javax.servlet.ServletRequest;
import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.common.net.HttpHeaders;

import net.shibboleth.utilities.java.support.annotation.constraint.NonnullAfterInit;
import net.shibboleth.utilities.java.support.annotation.constraint.NotEmpty;
import net.shibboleth.utilities.java.support.component.AbstractIdentifiableInitializableComponent;
import net.shibboleth.utilities.java.support.component.ComponentInitializationException;
import net.shibboleth.utilities.java.support.component.ComponentSupport;
import net.shibboleth.utilities.java.support.logic.Constraint;
import net.shibboleth.utilities.java.support.primitive.StringSupport;

/**
 * Access control implementation based on pre-shared API keys in the HTTP Authorization header or query
parameters.
 */
public class APIKeyAccessControl  extends AbstractIdentifiableInitializableComponent implements AccessControl{


   /** Class logger. */
    @Nonnull private final Logger log = LoggerFactory.getLogger(APIKeyAccessControl.class);

    /** The API key to match in the request. */
    @NonnullAfterInit @NotEmpty private String apiKey;

    /** The HTTP Authorization scheme name.*/
    @Nonnull @NotEmpty public static final String API_KEY_HEADER_SCHEME="SHIB-API-KEY";

    /** The API key query parameter name.  */
    @Nonnull @NotEmpty public static final String API_KEY_PARAM_NAME="shibapikey";

    /** Whether API key has sufficient entropy, or conforms to a specific policy? */
    @Nonnull private Predicate<String> apiKeyValidationPolicyPredicate;

    /** Where the API key can be found in the request.   */
    public enum APIKeyIn{
        /** The API Key is contained in the HTTP Authorization header. */
        HEADER,
        /** The API Key is contained in a query string parameter. */
        QUERY,
    }

    /** Where the API key can be found in the request. Default is HEADER.*/
    @Nonnull private APIKeyIn apiKeyIn;

    /**
     *
     * Constructor.
     *
     */
    public APIKeyAccessControl() {
        apiKeyIn = APIKeyIn.HEADER;
        apiKeyValidationPolicyPredicate = new LengthBasedAPIKeyPolicyValidator();
    }

    @Override protected void doInitialize() throws ComponentInitializationException {

        if (apiKey == null) {
            throw new ComponentInitializationException("API Key cannot be null");
        }

       if (!apiKeyValidationPolicyPredicate.test(apiKey)) {
           throw new ComponentInitializationException("API Key does not satisfy validation policy");
       }

    }

    /**
     * Set the API key validation policy.
     *
     * @param keyPolicy the API key validation policy.
     */
    public void setApiKeyValidationPolicyPredicate(@Nonnull final Predicate<String> keyPolicy) {
        ComponentSupport.ifInitializedThrowUnmodifiabledComponentException(this);
        apiKeyValidationPolicyPredicate = Constraint.isNotNull(keyPolicy, "API key validation policy must not
be null");
    }

    /**
     * Set the API Key.
     *
     * @param key the API key, can not be {@literal null} or {@literal empty}.
     */
    public void setApiKey(@Nonnull @NotEmpty final String key) {
        ComponentSupport.ifInitializedThrowUnmodifiabledComponentException(this);
        apiKey = Constraint.isNotEmpty(key, "API Key cannot be null or empty");
    }

    /**
     * Sets where to find the API key, either the HTTP header or the query parameters.
     *
     * @param keyIn where to find the API key.
     */
    public void setApiKeyIn(@Nonnull final APIKeyIn keyIn) {
        ComponentSupport.ifInitializedThrowUnmodifiabledComponentException(this);
        apiKeyIn = Constraint.isNotNull(keyIn, "API Key In (Header or Query Parameters) can not be null");
    }

    /** {@inheritDoc} */
    @Nonnull public boolean checkAccess(@Nonnull final ServletRequest request, @Nullable final String operation,
            @Nullable final String resource) {

       final String addr = request.getRemoteAddr()!=null ? request.getRemoteAddr():"unknown";

       if (request instanceof HttpServletRequest) {
           final HttpServletRequest httpRequest = (HttpServletRequest) request;

          String requestApiKey = null;
```

```
            if (APIKeyIn.HEADER == apiKeyIn) {

                requestApiKey = extractApiKeyFromHeaders(httpRequest);

            } else if (APIKeyIn.QUERY == apiKeyIn) {

                requestApiKey = extractApiKeyFromParameters(httpRequest);
            }
            if (apiKey.equals(requestApiKey)) {
                log.debug("{} Granted access to client [{}] using matching API key (Operation: {}, Resource:
{})",
                        new Object[] {getLogPrefix(), addr, operation, resource});
                return true;
            }else {
                log.debug("{} Denied request from client [{}], API keys do not match (Operation: {}, Resource:
{})",
                        new Object[] {getLogPrefix(), addr, operation, resource});
                return false;
            }

        } else {
            log.warn("{} Denied request from client [{}], request is not a HTTP request (Operation: {},
Resource: {})",
                    getLogPrefix(),addr, operation, resource);
            return false;
        }



    }

    /**
     * Gets the API key passed in via the HTTP parameters. Only one API key value is expected.
     *
     * @param httpRequest current HTTP request
     * @return the API key, or {@literal null}
     */
    @Nullable private String extractApiKeyFromParameters(@Nonnull final HttpServletRequest httpRequest) {

        final String[] apiKeyValues = httpRequest.getParameterValues(API_KEY_PARAM_NAME);

        if (apiKeyValues==null || apiKeyValues.length!=1) {
            log.warn("{} One API key parameter expected, has {}",
                    getLogPrefix(),apiKeyValues==null?"none":apiKeyValues.length);
            return null;
        }

        return StringSupport.trimOrNull(apiKeyValues[0]);

    }

    /**
     * Gets the API key passed in via the {@link HttpHeaders#AUTHORIZATION} header. This method checks to
     * ensure that the authentication scheme is {@link #API_KEY_HEADER_SCHEME} and then strips off
     * and returns the follow on key. Extracts the first of such found.
     *
     * @param httpRequest current HTTP request
     *
     * @return the API key, or {@literal null}
     */
    @Nullable private String extractApiKeyFromHeaders(@Nonnull final HttpServletRequest httpRequest) {

        final Enumeration<String> header = httpRequest.getHeaders(HttpHeaders.AUTHORIZATION);

        while (header.hasMoreElements()) {
            final String[] splitValue = header.nextElement().split(" ");
            if (splitValue.length == 2) {
                final String authnScheme = StringSupport.trimOrNull(splitValue[0]);
                if (API_KEY_HEADER_SCHEME.equalsIgnoreCase(authnScheme)) {
                    return StringSupport.trimOrNull(splitValue[1]);
                }
            }
        }


        log.debug("{} No appropriate Authorization header found", getLogPrefix());
        return null;
    }


    /**
     * Get logging prefix.
     *
     * @return  prefix
     */
    @Nonnull private String getLogPrefix() {
        return "Policy " + getId() + ":";
    }

    /**
     * Simple, default, API key validator. Requires the key to be >= 9 characters.
     */
    class LengthBasedAPIKeyPolicyValidator implements Predicate<String>{

        /** {@inheritDoc} */
        @Nonnull public boolean test(@Nullable final String key) {
            if (StringSupport.trimOrNull(key)==null) {
                return false;
            }
            if (key.length()>=9) {
                return true;
            }
            return false;
        }

    }

}
```