Swift Access Control

1. Container ACLs - Simple Examples

1.1 Swift command line tools

(1) demo:demo authorizes read access of container1 to demo:swift-user1, auth v2.0:

```
swift -V 2.0 -A http://example.swift.com:5000/v2.0 -U demo:demo -K password post
    -r 'demo:swift-user1' container1
(2) demo:demo authorizes write access of container1 to all users in demo, auth v2.0:
    swift -V 2.0 -A http://example.swift.com:5000/v2.0 -U demo:demo -K password post
    -w 'demo:*' container1
```

1.2 curl

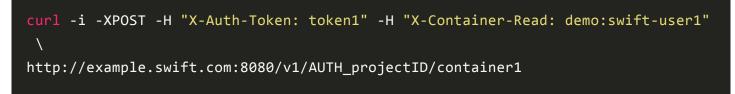
```
curl -X <PUT|POST> -i \
-H "X-Auth-Token: <TOKEN>" \
-H "X-Container-Read: <ACL>"\
<STORAGE_URL>/<container>
```

Example demo:demo grants read access to demo:swift-user1 :

1. Get token via keystone auth

```
curl -d '{"auth":{"tenantName": "demo","passwordCredentials":{"username": "demo"
,"password": "password"}}}' -H "Content-type:application/json" http://example.sw
ift.com:35357/v2.0/tokens
```

2. Get token via keystone auth



token1 and the url include 'AUTH_projectID' can be got from the response of step one.

The two methoeds were tested with packstack allinone default installation. That means it works with keystone, rather thatn tempauth.

[pipeline:main] pipeline = healthcheck cache authtoken keystone staticweb tempurl proxy-server

2. ACLs

Besides the mentioned example, if the swift is serving public web content, it can use the ACL syntax for managing allowed referrers. The syntax is '.r:' followed by a list of allowed referrers. For example, this command allows all referring domains access to the object:

```
swift -V 2.0 -A http://example.swift.com:5000/v2.0 -U demo:demo -K password post
    -r '.r:*' container1
```

The minus sign - indicates referrer hosts that should be denied access, for example:

```
.r:.example.com,
.r:-thief.example.com
```

This would allow all hosts ending with .example.com except for the specific host thief.example.com.

3. TempURL

TempURL is a middleware for granting temporary access to particular objects.

TempURL is a part of Swift, so the code is already on the cluster. To enable it, make sure that it's in proxy server's middleware pipeline before the authentication filters, such as authtoken, tempauth or keystoneauth.

```
[pipeline:main]
pipeline = healthcheck cache tempurl authtoken keystone staticweb proxy-server
```

The default methods is GET, PUT and HEAD, it can be configured to enable other methods.

```
[filter:tempurl]
use = egg:swift#tempurl
methods=GET HEAD PUT DELETE POST
```

Simple steps to use tempurl:

1. Set the Temp-URL-Key - auth v2.0

```
swift -V 2.0 -A http://example.swift.com:5000/v2.0 -U demo:demo -K password post
-m Temp-URL-Key:thisIsAKey
```

2. Create the temporary URL, by bin/swift-temp-url

An sample temp URL looks like this:

http://example.swift.com/v1/AUTH_12345/container1/object? temp_url_sig=da39a3ee5e6b4b0d3255bfef95601890afd80709& temp_url_expires=1323479485

A URL's signature is derived from four things:

- <method>: the HTTP verb (e.g. GET, POST)
- <seconds>: how long the request should be allowed
- <path>: the URL's path
- <key>: the temp url key

./swift-temp-url <method> <seconds> <path> <key>

There will be a respone of temp_url_sig, then it can be used in the temp url.

crul http://example.swift.com/v1/AUTH_12345/container1/object?temp_url_sig=da39a
3ee5e6b4b0d3255bfef95601890afd80709&temp_url_expires=1323479485

4. Swift Auth System

1. TempAuth

The TempAuth has the concept of admin and non-admin users within an account:

- Admin users can do anything within the account.
- Non-admin users can only perform operations **per container** based on the container's X-Container-Read and X-Container-Write ACLs.

Swift makes calls to the tempauth system, giving the auth token to be validated. For a valid token, the auth system responds with an overall expiration in seconds from now. Swift will cache the token up to the expiration time, which is by default 1 day, and configurable.

The code can be found in *middleware/tempauth.py*

self.token_life = int(conf.get('token_life', 86400))

Container ACLs use the "V1" ACL syntax.

In addition to container ACLs, TempAuth allows account-level ACLs. Any auth system may use the special header X-Account-Access-Control to specify account-level ACLs in a format specific to that auth system. (Following the TempAuth format is strongly recommended.) These headers are visible and

settable only by account owners (those for whom swift_owner is true). Behavior of account ACLs is authsystem-dependent.

In the case of TempAuth, if an authenticated user has membership in a group which is listed in the ACL, then the user is allowed the access level of that ACL.

Account ACLs use the "V2" ACL syntax, which is a JSON dictionary with keys named "admin", "read-write", and "read-only". (Note the case sensitivity.)

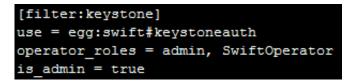
2. Keystone Auth

The keystoneauth middleware performs authorization and mapping the keystone roles to Swift's ACLs.

Link to Configuring Swift to use Keystone

If support is required for unvalidated users (as with anonymous access) or for tempurl middleware, authtoken will need to be configured with delay_auth_decision set to 1.

By default the only users able to give ACL are the ones who has the Keystone role specified in the operator_roles setting. Which locates in *proxy-server.conf*.



This user who have one of those role will be able to give ACLs to other users on containers.

Users with the Keystone role defined in reseller_admin_role (ResellerAdmin by default) can operate on any account. The auth system sets the request environ reseller_request to True if a request is coming from a user with this role. This can be used by other middlewares.

3. Extending Auth

Writing a new wsgi middleware, and plugging it into the proxy server. Link to Creating Auth Server and Middleware

4. Swauth

Not currently under active development, but maintenance.

Link to Swauth github