Authentication in OpenStack

Prepared for: Open Stack Summit – Nov 2010

Proposed By: Jorge Williams,
Khaled Hussein,
Ziad Sawalha / Rackspace Architecture

Date: Nov 10, 2010

Our Assumptions

- OpenStack will be deployed by different ‘operators’
  - Enterprises
  - Institutions
  - Service providers
  - Developers, etc…
- OpenStack must be designed to work well in any kind of environment.
  - Some big integration points:
    - **A**uthentication: Who are you?
    - **A**uthorization: What are you allowed to do?
    - **A**ccounting: What are you doing? What resources are you consuming? In what quantity?

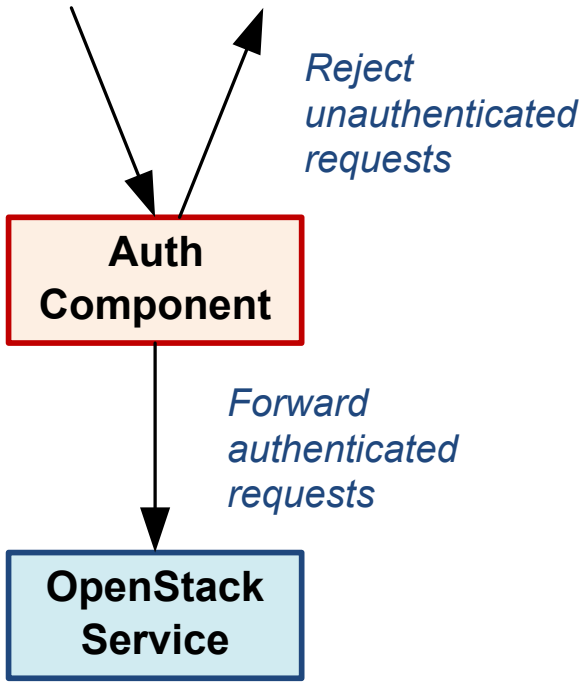
Authentication

- In this blueprint, we attack the problem of Authentication only
  - Relatively easy to approach and generalize
  - It's natural to attack this problem first.
    - We can't even begin looking at Authorization or Accounting if users can't authenticate.
  - We need to solve the problem sooner rather than later
    - Different Auth mechanism in Compute and Object Store
    - Not having a unified authentication mechanism is a problem today

The Goal

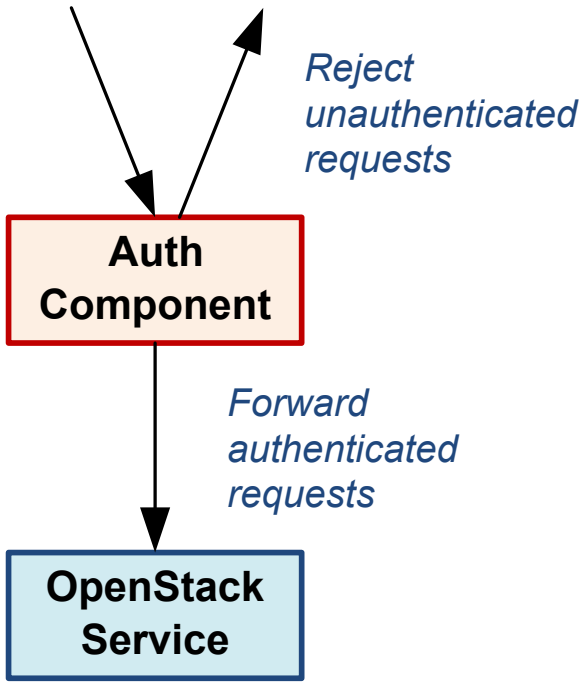
- Each deployment will integrate with and leverage the environment it is deployed in.
- For Authentication this means support for...
  - Separate Authentication protocols
    - HTTP Basic
    - Digest Access
    - Public Key
    - Token (OAuth etc)
  - Obtaining user information from a variety of places
    - Identity management systems
    - LDAP directories
    - Relational Databases
    - Flat Files

Proposed Solution: Authentication Component



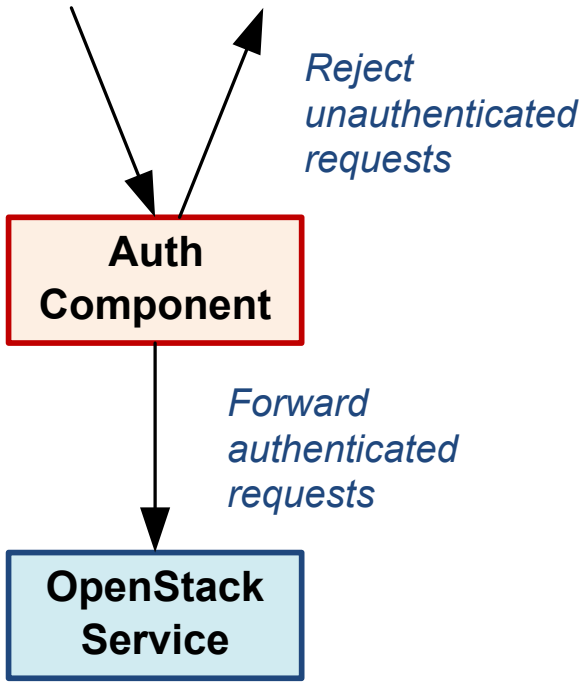
- A pluggable software module
  - Implements an authentication protocol
  - Obtains user information via a backend service (an IDM system)
- A Reverse HTTP Proxy
  - Intercepts HTTP calls
  - Checks a user's identity
    - If the identity is verified it augments the call with user information and passes it to the service.
    - Otherwise the message is rejected before the call reaches the service.

Proposed Solution: Authentication Component



- This is not a new radical idea
  - Existing systems take a similar approach
    - repoze.who
  - We can certainly leverage these sort of systems
  - ...we want to standardize on a protocol, rather than a software library

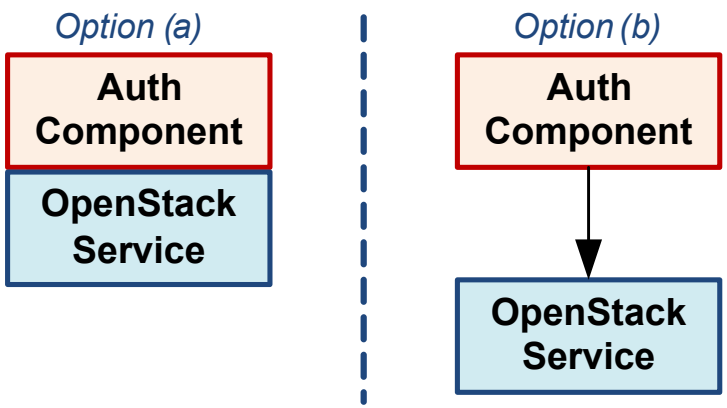
Proposed Solution: Authentication Component



- We want to standardize:
  - The interactions between the open stack component and the authentication component.
  - The handling of exceptional cases
    - The Auth Component is improperly configured?
    - Someone bypasses the Auth Component
- We don't define other interactions with the auth component.
  - How is an unauthenticated request handled?
  - How do we receive credentials?
  - These questions are addressed by the underlying authentication protocol.
- We don't define authorization
  - Can be a controversial subject
  - Places much higher requirements on OpenStack service
  - We don't have a clear solution...yet

Deployment Strategies

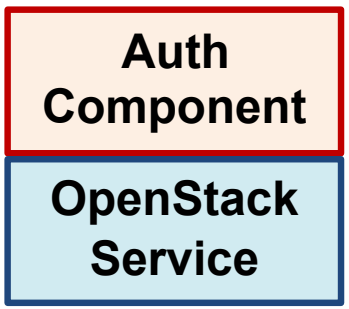
- Components may be:
  - Integrated into the service itself: Option (a)
  - Deployed separately as a true HTTP reverse proxy: Option (b)



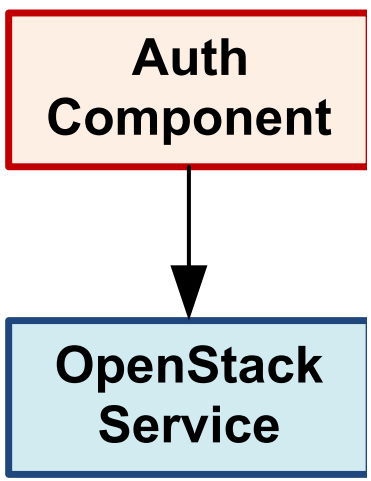
Deployment Strategies

- Option (a) Benefits
  - Low Latency
  - Ease of implementation
  - Appropriate for simple configurations

Option (a)



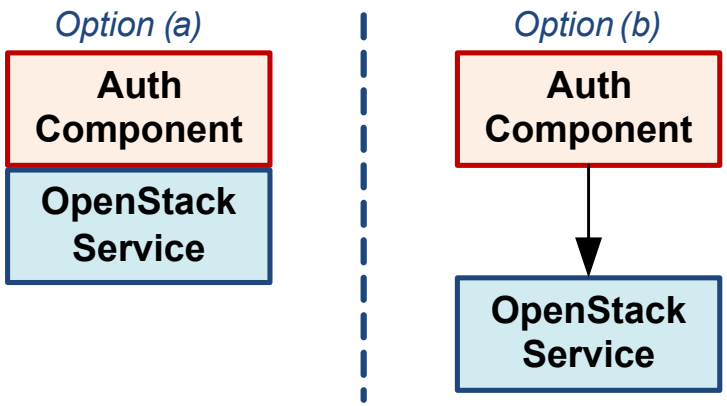
Option (b)



- Option (b) Benefits
  - Horizontally scalable
    - Important for computationally expensive tasks (digital signatures)
  - Multiple Authentication components can be deployed.
  - Authentication components can be written in different languages
  - Appropriate for complex/dynamic configurations

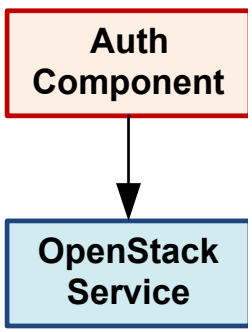
Deployment Strategies

- OpenStack services MUST support both strategies
- Authentication Components can be written to Option (a) or Option (b) or both.
- To support Option (a), in Python, components MUST be written as WSGI middleware



Exchanging User information

- The component passes an X-Authorization header with the Username.

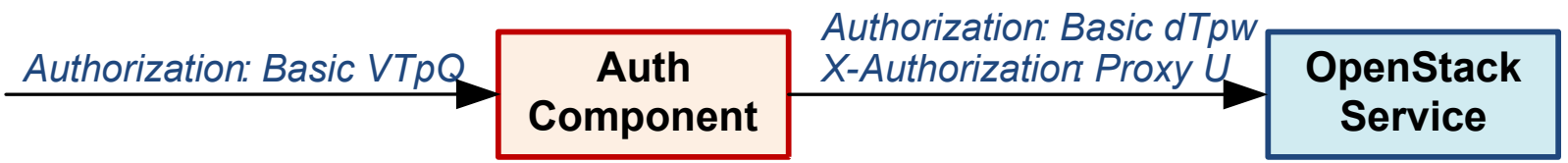



```
X-Authorization: Proxy JoeUser
```

- "Proxy" denotes that the authentication occurred via a Proxy
- We are considering using standard Authorization header, but we need to examine client support.
- Components may pass additional headers
  - We only require X-Authorization

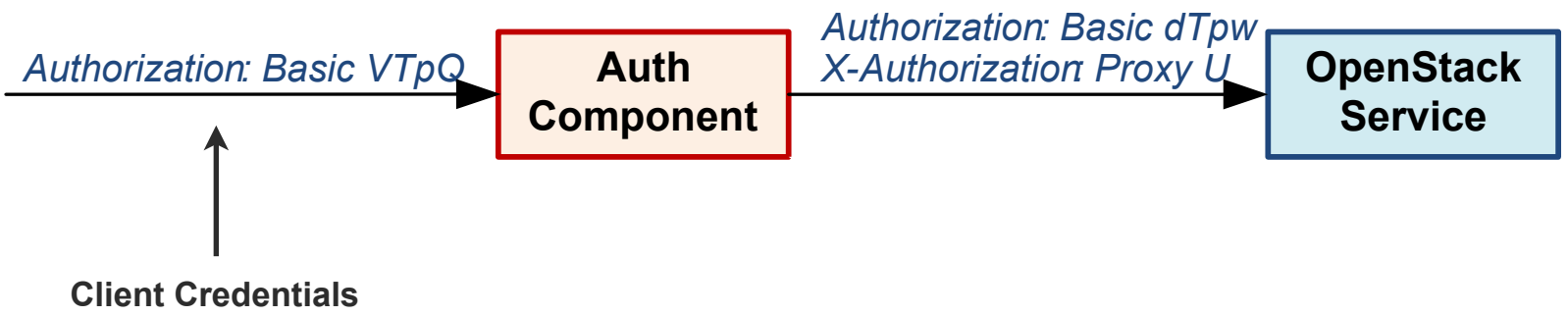
Reverse Proxy Authentication

- How can a service know requests are coming from a trusted proxy?
  - The authentication proxy, must authenticate
  - We call this Reverse Proxy Authentication
- Any HTTP authentication schemes may be used...
- ...But all services must support HTTP Basic Authentication



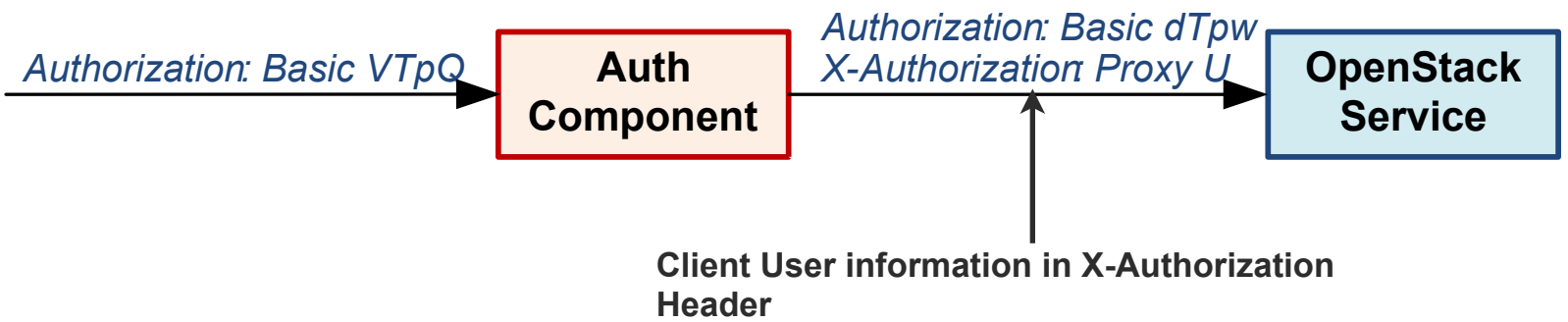
Reverse Proxy Authentication

- How can a service know requests are coming from a trusted proxy?
  - The authentication proxy, must authenticate
  - We call this Reverse Proxy Authentication
- Any HTTP authentication schemes may be used...
- ...But all services must support HTTP Basic Authentication



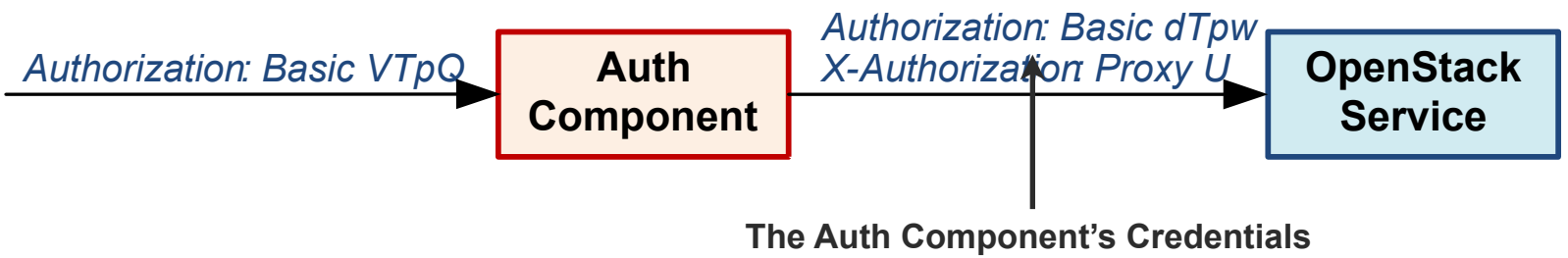
Reverse Proxy Authentication

- How can a service know requests are coming from a trusted proxy?
  - The authentication proxy, must authenticate
  - We call this Reverse Proxy Authentication
- Any HTTP authentication schemes may be used...
- ...But all services must support HTTP Basic Authentication



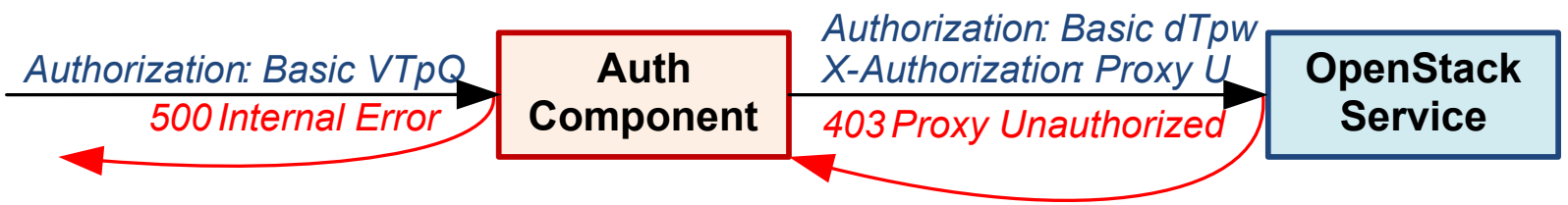
Reverse Proxy Authentication

- How can a service know requests are coming from a trusted proxy?
  - The authentication proxy, must authenticate
  - We call this Reverse Proxy Authentication
- Any HTTP authentication schemes may be used...
- ...But all services must support HTTP Basic Authentication



Reverse Proxy Authentication

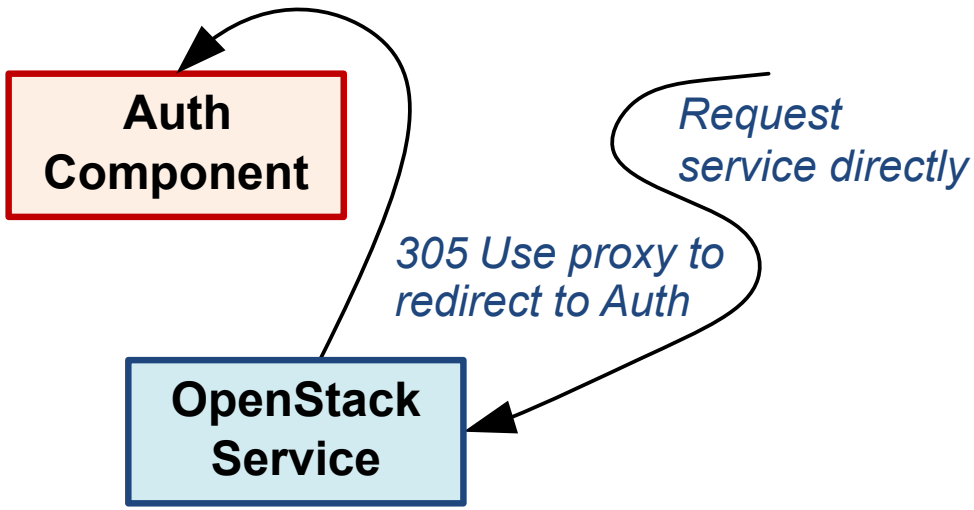
- What happens when authentication fails?
  - May Receive an 401 unauthorized or...
  - ...a 403 forbidden request?
- A 500 Internal error should be returned to the client
  - It is an internal error from the clients point of view
  - Avoid confusion when debugging issues



Handling Direct Connections

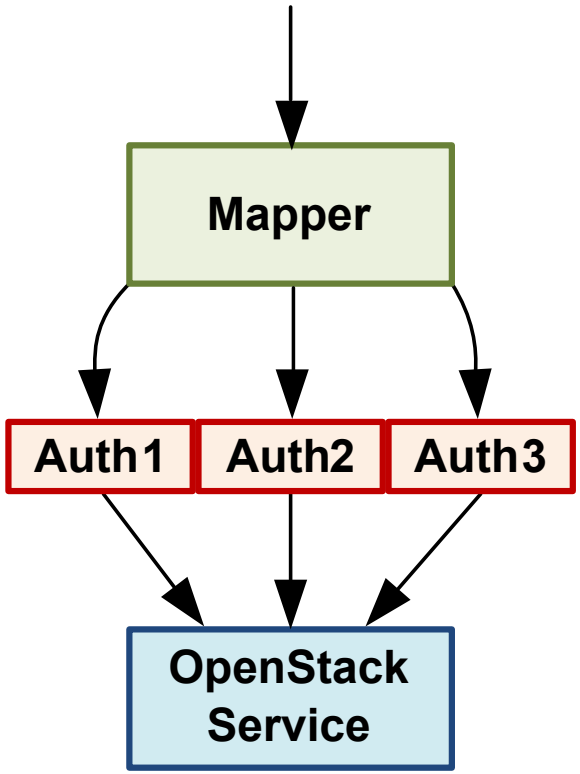
- Clients should not be able to by-pass the proxy
  - We can detect someone doing this by missing X-Authorization header
- In this case services MUST redirect to the Auth Component
  - 305 Use Proxy...

```
HTTP/1.1 305 Use Proxy
Date: Thu, 28 Oct 2010 07:41:16 GMT
Location: http://sample.auth.openstack.com/path/to/resource
```



Using Multiple Authentication Components

- Many reasons to support this
  - One authentication scheme for internal users...another for external users.
  - Support for multiple resellers with different authentications systems.
- Use Option (b) to deploy multiple authentication components
- Have a mapper that chooses the right one
  - The mapper makes a routing decision based on the resource being hit.
  - Can use a traditional reverse proxy as a mapper
    - Zeus
    - Pound
  - Mappers may be used to give anonymous or guest access
    - This is implemented with an anonymous or guest auth component.
- Neither the service or auth component need be aware of the mapper!



The Default Component

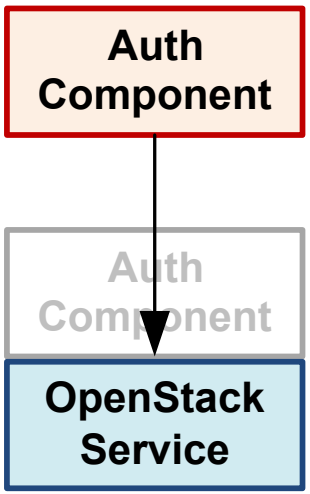
- Services MUST ship with the same default authentication component
  - The component MUST be integrated into the service by default: Option (a)
- The component supports HTTP Basic Auth
  - Reads a flat file for user information.
  - Passwords stored in flat file (/etc/openstack/users.ini) as SHA-1 digests.
    - The user submits a password, the SHA-1 of the password is compared against the file.

```
[users]

user:5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
user2:2aa60a8ff7fcd473d321e0146afd9e26df395147
user3:1119cfd37ee247357e034a08d844eea25f6fd20f
.
.
```

The Default Component

- As with any other component, it must be possible to replace/disable the default component:



Default Component: Reference Implantation

- An example of how to develop an auth component
  - Not meant for production...yet
- Composed of a number of components
  - authcomp.py : Sample authentication component, supports Option (a) and Option (b)
  - service.py : Sample service that supports Option (a) and Option (b)
  - embedded.py : Example deployment supporting Option (a)
  - helper.py : Utility Code

Authenticating requests in authcomp.py

```
def __call__(self, environ, start_response):
 if not environ.get("HTTP_AUTHORIZATION"):                    ❶
  # The user needs to be authenticated. We reject the request and
  # return 401 before the Service (MyApp) receives the request.
  return respond401(environ, start_response)                  ❷
 else:
  # Let's authenticate the user against the users.ini file.
  import base64
  auth_header = environ['HTTP_AUTHORIZATION']
  auth_type, encoded_creds = auth_header.split(None, 1)
  username, password  = base64.b64decode(encoded_creds).split(':', 1)
  if self.validateCreds(username, password):                  ❸
   # The Auth Component has to authenticate itself to the service.
   environ['HTTP_AUTHORIZATION'] = "Basic dTpw"               ❹
   # The Auth Component passes the username to the Service
   environ['HTTP_X_AUTHORIZATION'] = "Proxy %s" % username    ❺
   return self.app(environ, start_response)                   ❻
  else:
   return respond401(environ, start_response)                 ❼
```

Verifying a request in service.py

```
def __call__(self, environ, start_response):
 if not environ.get("HTTP_X_AUTHORIZATION"):
  return respond305(environ, start_response, \                    ❶
   createurl(environ, self.proxy_host, self.proxy_port))

 # Now let's authenticate the Auth Component itself
 if not environ.get("HTTP_AUTHORIZATION"):
  return respond401(environ, start_response)

 import base64
 auth_header = environ['HTTP_AUTHORIZATION']
 auth_type, encoded_creds = auth_header.split(None, 1)
 # Not very secure, but you get the picture
 if str(encoded_creds) != "dTpw":
  return respond401(environ, start_response, False)               ❷
 else:
  # Here is where the service processes the request
  response_headers = [('content-type', 'text/html')]
  response_status = '200 OK'
  user_auth_header = environ["HTTP_X_AUTHORIZATION"]
  auth_type, username = user_auth_header.split(None, 1)
  response_body = "Welcome %s to OpenStack <BR>" % username       ❸
  start_response(response_status, response_headers)
  return [response_body]
```

Questions??