Office of Strategic National Alien Planning

Worker Handle Origin
(WHO)

Product Requirements and Design Document

January 9, 2017

Executive Summary

The pleathora of authentication systems in use by OSNAP have resulted in numerous security incidents. To address this, the chief information security officer (CISO) has mandated that a single sign on (SSO) solution must be deployed. The Worker Handle Origin (WHO) system will provide SSO services for OSNAP applications. WHO will expose authentication and user information via a rest-like API.

Document Versioning

12/28/2016 DE - Initial version

01/09/2017 DE - Updates based on POC implementation

Project Description

Single sign on (SSO) solutions reduce the burden on users to keep track of multiple accounts and passwords. By reducing the burden on users, users are more likely to follow organizational security practices and avoid security incidents. The Worker Handle Origin will be OSNAP's SSO solution for future projects. Existing projects will need to be refitted to support WHO or must have a security exception on file and signed by the chief information security officer (CISO).

Single sign on is accomplished using a rest-like API. Public/private cryptography with signed keys will be used before passing data between the client and server. A plain text fallback will be available for systems under active development but the plain text interface may not be used in any production system.

API Specification

This section describes the API used to interact with WHO.

Overview

WHO can be used to authenticate users for OSNAP applications and verify wether a user is authorized to access an application. Calls are made via a REST-like API. The plain text functions are not available in production environments.

API Calls

authenticate

The *authenticate* call is used to validate that a user has provided valid credentials. The request blob will be JSON encrypted using the application's private key. The response will be JSON encrypted using the application's registered public key.

Request:

application Application name requesting authentication

blob Encrypted blob containing

- **username** Username to authenticate
- **userpass** Password to authenticate
- **transaction_id** Nonce to identify this request

Response:

transaction_id Nonce to identify this request

result Authentication result

additional Optional additional data

authorized

The *authorized* call is used to validate that a user is authorized to use a particular application. The request blob will be JSON encrypted using the application's private key. The response will be JSON encrypted using the application's registered public key.

Request:

application Application name requesting authentication

blob Encrypted blob containing

> **username** Username to authenticate
>
> **transaction_id** Nonce to identify this request

Response:

transaction_id Nonce to identify this request

result Authentication result

authenticate_plain

The *authenticate_plain* call is used to validate that a user has provided valid credentials. The request blob will be JSON plaintext. The response will be JSON plaintext.

Request:

application Application name requesting authentication

blob Encrypted blob containing

> **username** Username to authenticate
>
> **userpass** Password to authenticate
>
> **transaction_id** Nonce to identify this request

Response:

transaction_id Nonce to identify this request

result Authentication result

additional Optional additional data

authorized_plain

The *authorized_plain* call is used to validate that a user is authorized to use a particular application. The request blob will be JSON plaintext. The response will be JSON plaintext.

Request:

application Application name requesting authentication

blob Encrypted blob containing

> **username** Username to authenticate
>
> **transaction_id** Nonce to identify this request

Response:

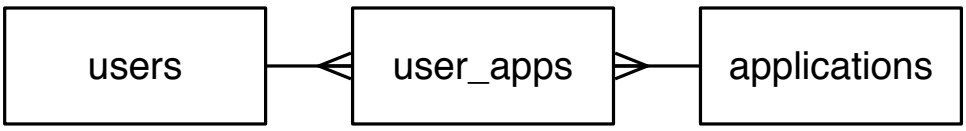
transaction_id Nonce to identify this request

result Authentication result

Data Model

Persistent data will be stored in a relational database. This section discusses the relational database model supporting WHO.

Overview



WHO has fairly modest data needs to provide authentication services. Users are authenticated using a username and password pair. To support encryption, applications requesting service must have a public key registered with WHO.

Table Definitions

users

user_pk	integer	primary key for user records
username	varchar(128)	OSNAP username
passhash	varchar(256)	hashed password

applications

app_pk	integer	primary key for user records
application_name	varchar(128)	OSNAP application name
application_desc	varchar(256)	Longer information about the application
application_key	blob	public key for the application

user_apps

user_fk	integer	primary key for user records
app_fk	integer	primary key for application record

Tech Stack

The Worker Handle Origin (WHO) product will use OSNAP's standard web application technology stack. No deviations to the standard technology are expected to support WHO. Exceptions must be approved by the OSNAP Chief Information Security Officer (CISO) prior to deployment.

Standard Technologies

Apache httpd The Apache http daemon will be used to host the web application.

mod_wsgi mod_wsgi will be used as the gateway between Apache and the application.

Python Python 3 will be used as the application development language.

PyCryptodom Provides cryptographic services for Python

Flask The Flask framework will support development efforts.

Postgres The Postgres RDBMs will be used for persistent storage.

PGSQL If needed, stored procedures will be written using the default procedure language PGSQL.

Software Design

This section provides architectural details regarding the software design supporting WHO.

Overview

As a webservice, there is no page flow associated with WHO. Each path is handled by a separate function. For compatibility an ease of use, the index path for WHO provides a listing of the available WHO calls. Calls can be made using GET or POST requests.

Access to applications is managed centrally at OSNAP. To address this requirement, WHO tracks the association between users and applications. All WHO service calls require the requesting application to be provided. For the encrypted calls, which must be used in production, the application is used to identify the key needed to read the signature.

Each call expects to receive the requesting application name and a blob containing the remaining arguments. The blob will be plaintext or encrypted json. Responses will also be plaintext or encrypted json.

Authenticate

Authentication verifies that a user is who they claim to be by verifying that a given username and password pair matches the registered username and password pair. Additionally, WHO also checks that the user should a associated with the requesting application.

Implementation

Implementation is contained in *app.py*, *authenticate_plain* and *authenticate* respectively for plaintext and encrypted version.

Authorize

Authorization is a quick check to verify that a username has access to an application.

Implementation

Implementation is contained in *app.py*, *authorized_plain* and *authorized* respectively for plaintext and encrypted version.