Tapis Tokens

2021 NSF Cybersecurity Summit

Token-Based Authentication and Authorization Workshop

October 18, 2021

Joe Stubbs (jstubbs@tacc.utexas.edu)

Sean Cleveland (seanbc@hawaii.edu)

Richard Cardone (rcardone@tacc.utexas.edu)

Tapis Tokens

- Background & Use Cases
- Components of Tapis Security Architecture
- Challenges using Token in Tapis

Background & Use Cases

- Tapis Project
- Tapis Services
- Data Management and Code Execution
- Use Cases:
  - Service to Service Requests
  - Cross Site Service Requests

Tapis Project

- 5 year, NSF funded computing framework supporting multi-site computational research
- Used to manage data and execute code on HPC, HTC and cloud systems (>51K researcher accounts, 23 tenants & 15 gateways 2020-2021)
- Agentless, SSH-based communication with storage/compute systems
- Implemented as microservices with REST interfaces
- Users obtain a token by authenticating to Tapis using OAuth2
    - Subsequent APIs calls are authenticated using the token

https://tapis-project.org

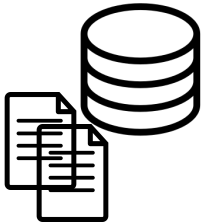
Tapis Services

Tenancy, Authentication and Security

- Tenants
- Sites
- Tokens
- Authenticator
- Security Kernel
- *Postits

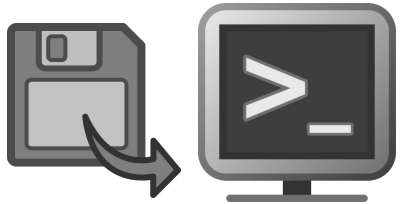
MetaData Management

- Meta
- PgREST



Data Management and Code Executions

- Systems
- Files
- Apps
- Jobs



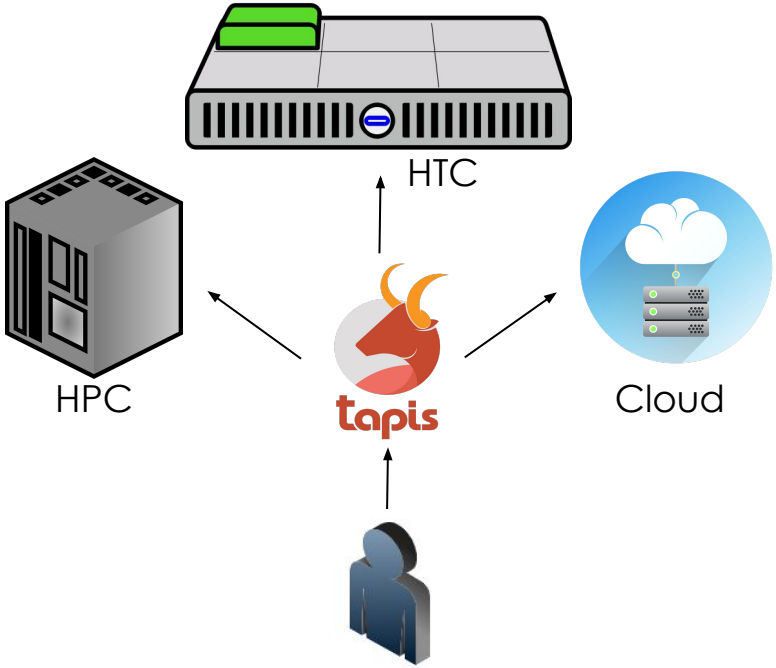
Streaming Data, Events and Functions

- Functions (Actors)
- *Notifications
- Streams

https://tapis-project.github.io/live-docs

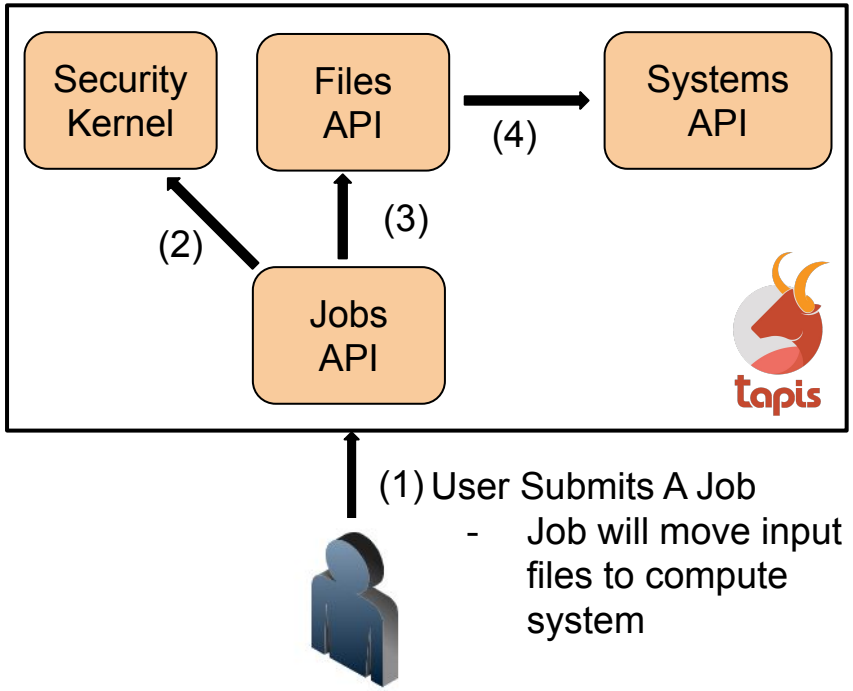
Data Management and Code Execution APIs

/systems /files /apps /jobs



- Register storage and compute systems
- Ingest, move and transform data files and folders
- Register application containers on large systems
- Launch jobs to invoke applications & Capture metadata about the workflow

Use Case: Service to Service Requests



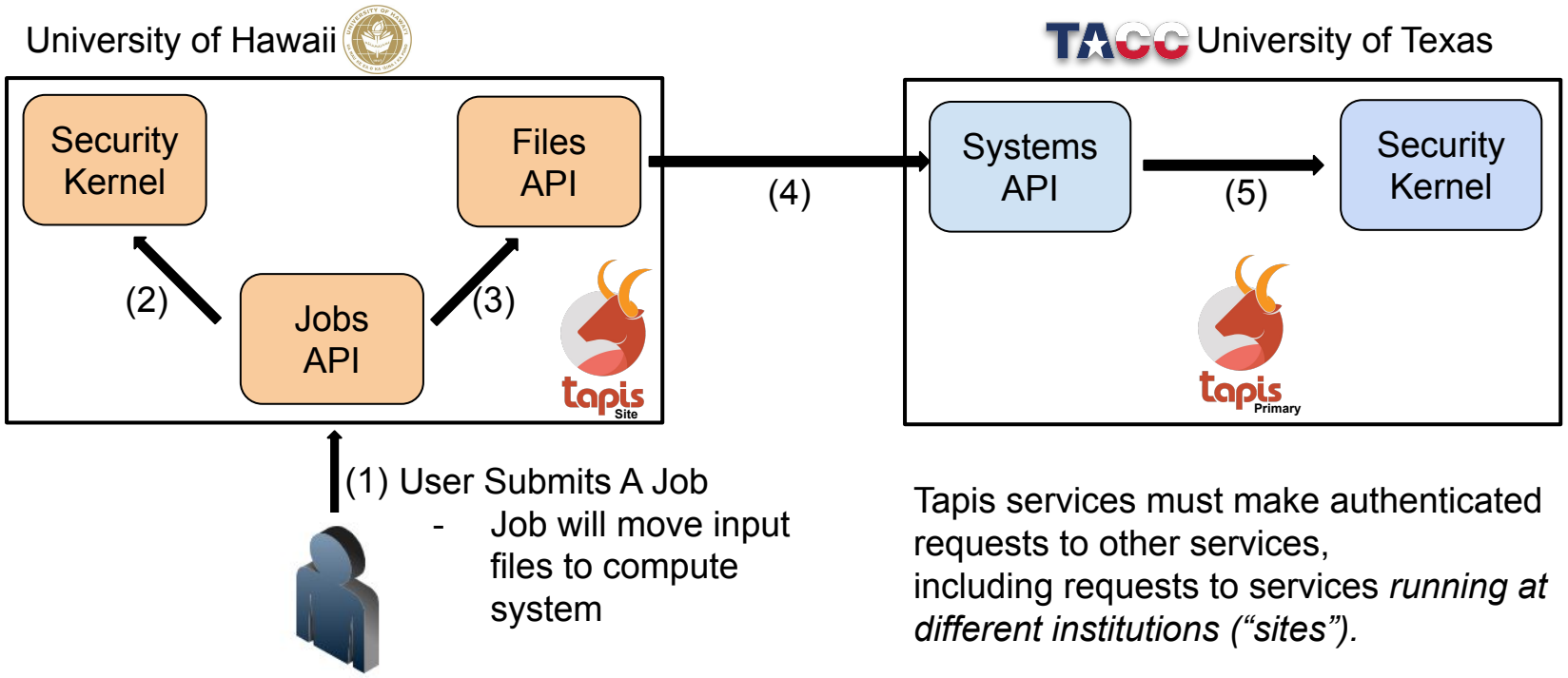
Tapis services must make authenticated requests to other services on behalf of the users.

2) Check user permissions

3) Schedule Data Transfer

4) Retrieve Storage System Definition

Use Case: Cross-Site Service Requests



Tapis services must make authenticated requests to other services, including requests to services *running at different institutions ("sites").*

Components of Tapis Security Architecture

- Tenancy
- Multi-Site Support
- Security Microservices
  - Security Kernel (SK)
  - Tokens
  - Authenticators
- Tapis JWT

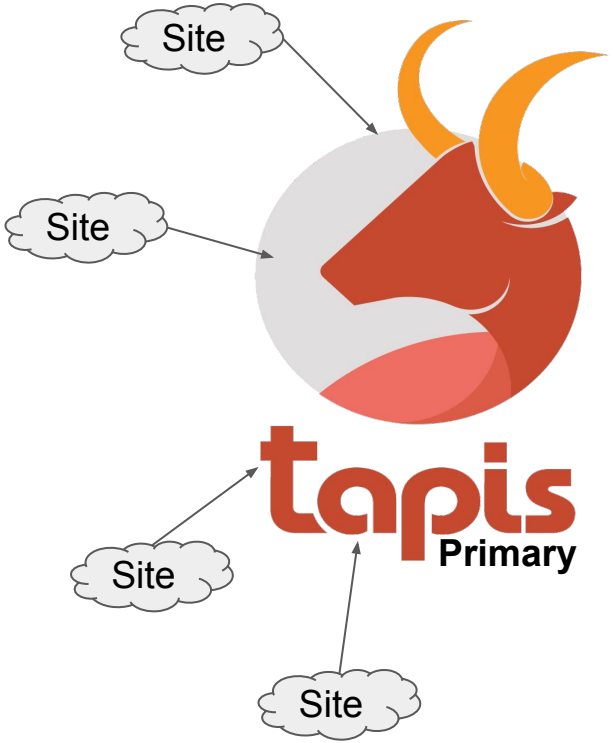
Tenancy in Tapis

- Tenants are logically isolated views of the platform
  - Partition groups of users and their resources
- Every tenant is “owned” by a site
- A site can have 1 or more tenants
- The *Tenants service* provides a registry of all tenants and sites
- A single Tenants instance runs only at the primary site
- Every tenant has its own JWT signing key pair
  - Tenant public keys accessible via Tenants API w/o authentication

Multi-Site Support

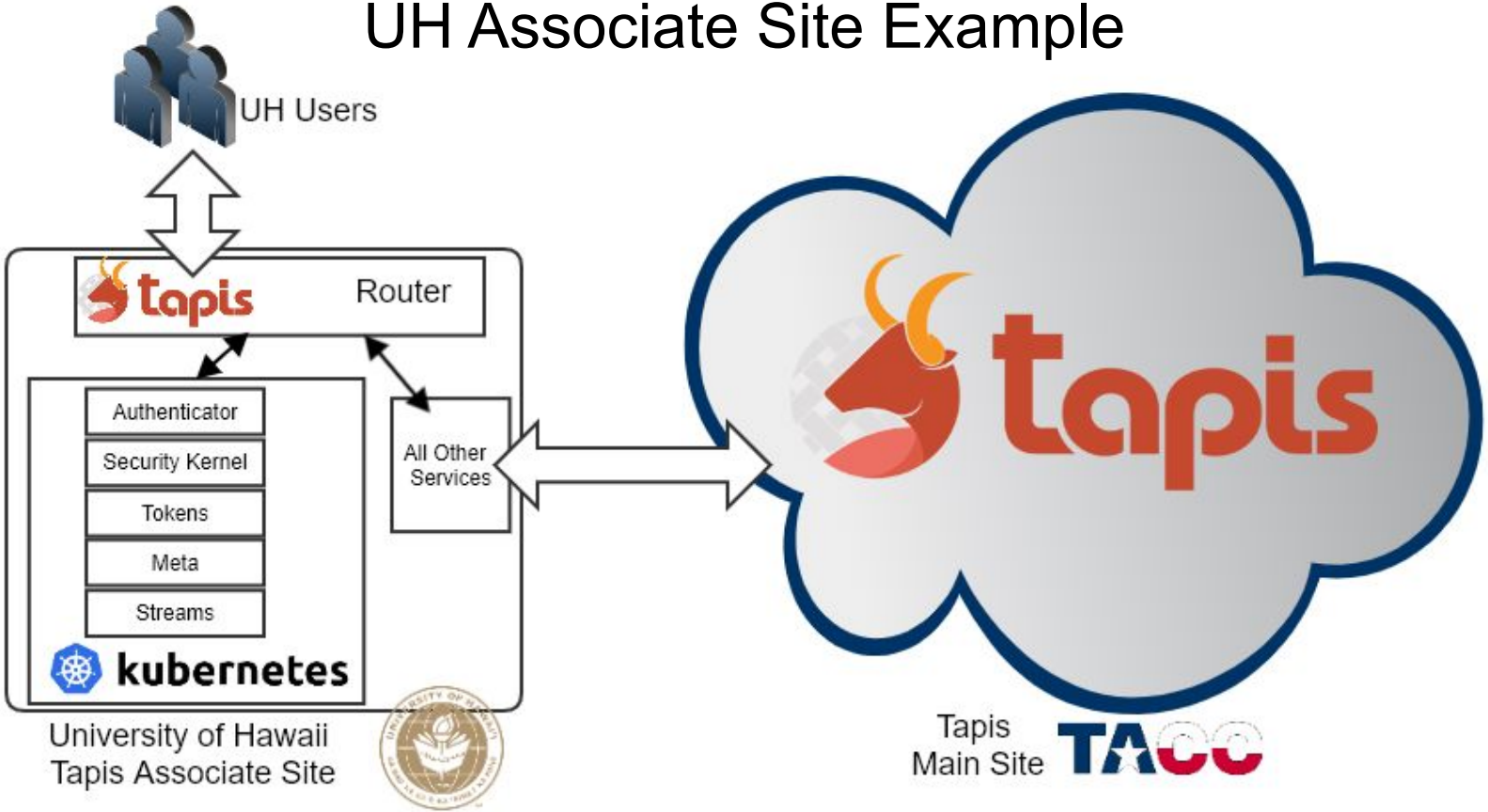
Hub and Spoke Communication Model

- Associate sites don't make requests to each other
- Local Control of Identity and Access Management
  - Can use own LDAP or user stores
  - Can use the default authenticator or own
- Local Control of Secrets
  - All secrets and keys stored at local site
- Local Control of Deployed Services
  - Improve data locality
  - Accommodate large databases
  - Extend Tapis by adding/integrating custom services



Requests to services not running at an associate site route to the primary site

UH Associate Site Example



Authenticators

- Implement user “login” function by interacting with Identity Provider (IDP)
    - IDPs are typically external to Tapis, such as an institution’s LDAP server
- Each tenant can have its own Authenticator and IDP
- Authenticators interact with Tokens to acquire JWTs for users
    - Authenticator calls IDP to validate user credentials
    - Authenticator calls Tokens to create a new Tapis JWT for user

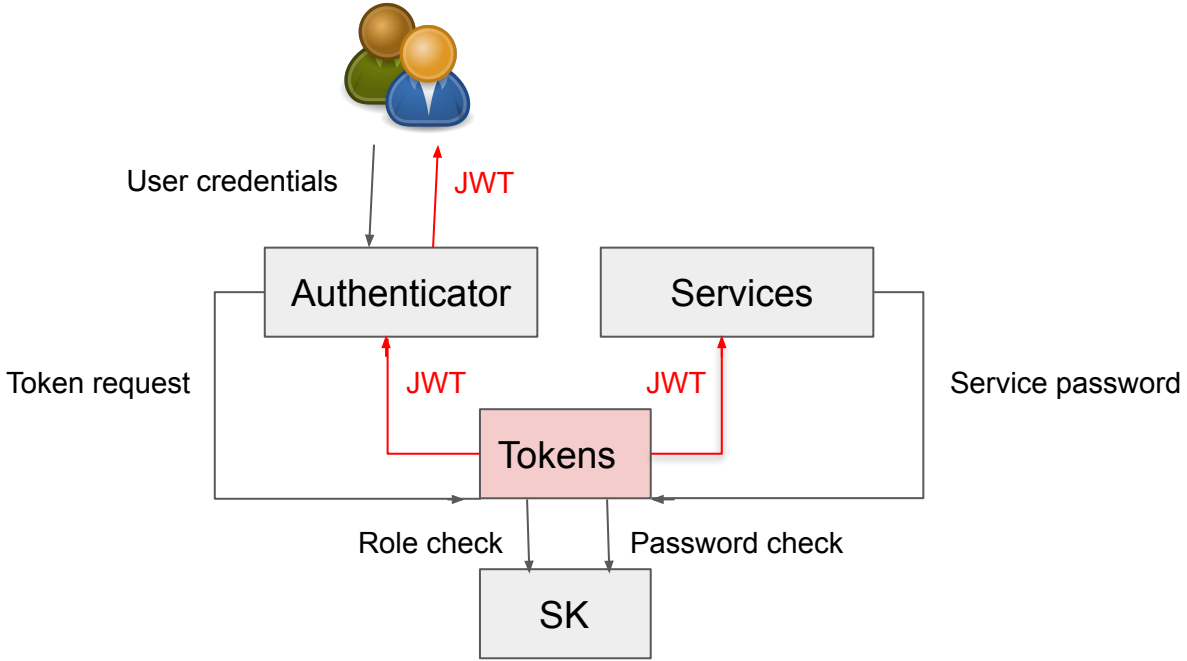
Tapis Tokens Service

- Tokens creates and signs JWTs
  - Uses tenant-specific signing keys
- Loads signing keys from SK at startup
  - Only Tokens service can access keys
- Authenticators validate user credentials
  - Request user JWTs from Tokens
- Services authenticate with a service password
  - Passwords injected into services at startup
  - Tokens calls SK to validate password
  - Tokens creates refreshable service JWTs

Tapis Security Kernel

- Security Kernel (SK) - Manages secrets and authorization data
  - Hashicorp Vault for secrets management
  - Apache Shiro based roles and permissions
- Every site runs a Security Kernel
- Only local services can access local SK
- Maintains the public/private key pairs used for signing and verifying tokens
  - Only the keys for tenants owned by that site
- Every site runs its own Authenticator(s), Tokens and SK services
  - ***Services at a site only interact with their local Tokens and SK***

Tapis JWT Creation Flows



The Tapis JWT

- Specified on API calls in the ***X-Tapis-Token*** header
- Contains standard (iss, exp, sub) and custom Tapis claims

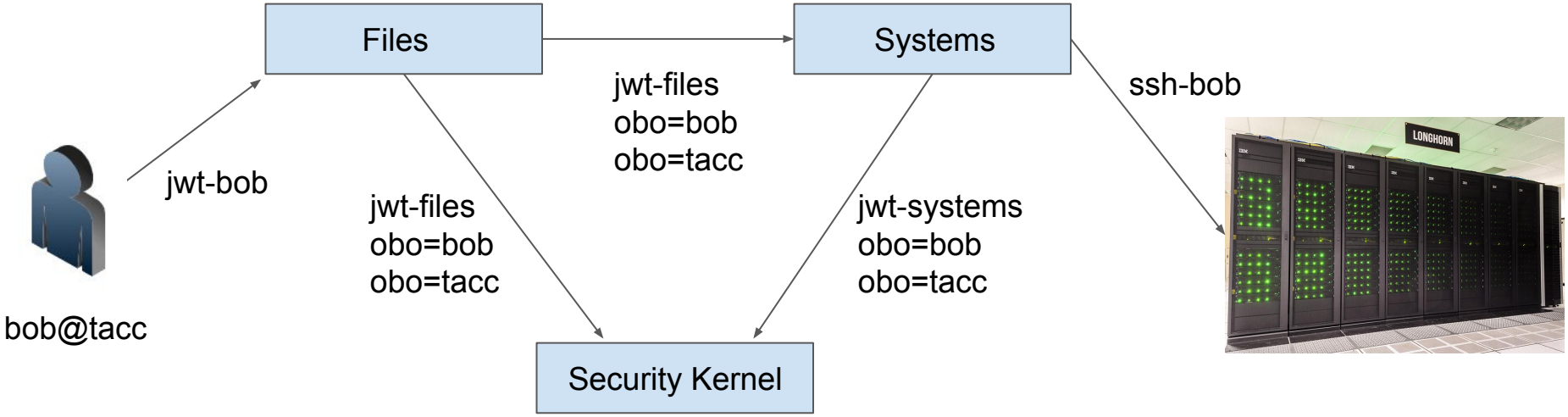
"sub": standard subject in <username>@<tenant> format
"**tapis/token_type**": access | refresh
"**tapis/account_type**": service | user
"**tapis/site_id**": originator's site
"**tapis/target_site_id**": site where JWT is valid
"**tapis/tenant_id**": tenant_id of the subject of the JWT
"**tapis/username**": username of the subject of the JWT
"**tapis/delegation**": true | false
"**tapis/delegation_sub**": the authorized *delegator* who created JWT (<username>@<tenant>)

Challenges Using Tokens in Tapis

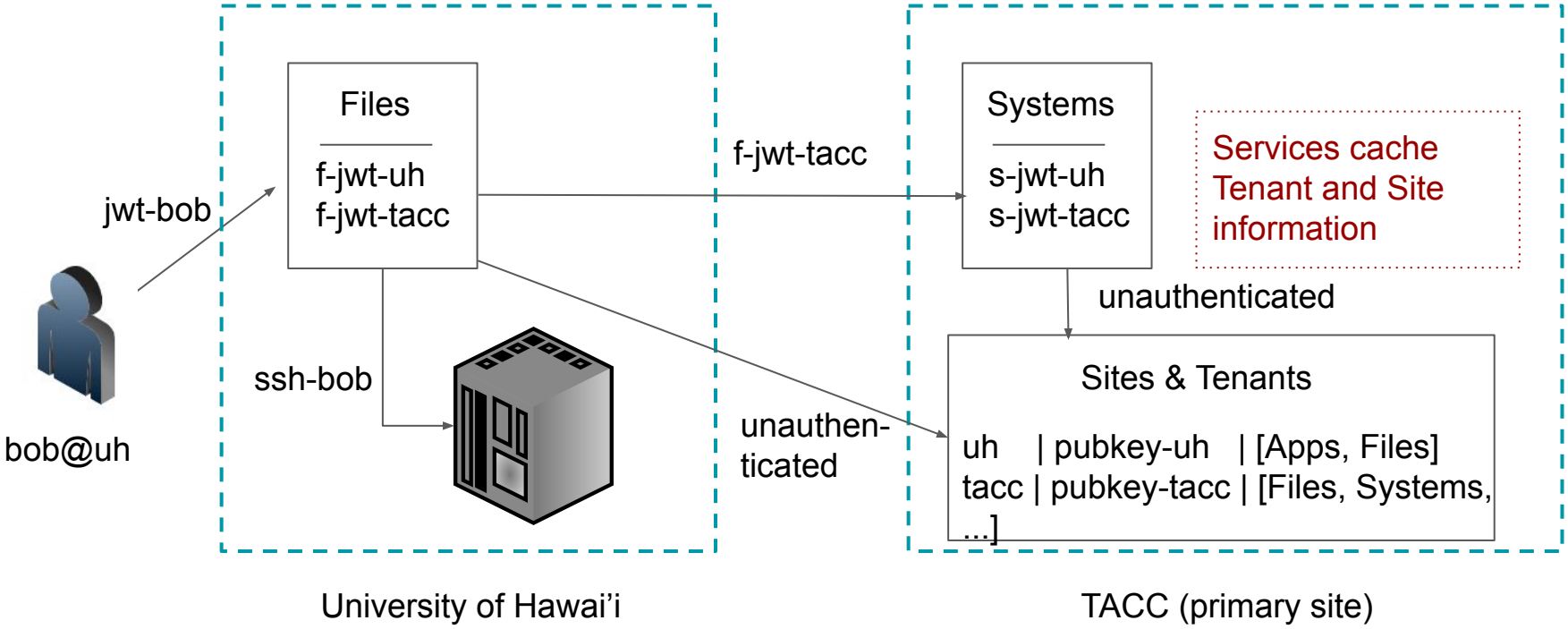
- On-Behalf-Of (OBO) data transmission
- Sending Service-to-service requests (routing, JWT selection)
- Receiving Service Requests (validation)
- *Cross-Site resource access*
- *Dynamic authentication*

On Behalf Of (OBO) Request Data

Context: When a service makes a request to another service, it uses its own service JWT to authenticate.
Challenge: Preserve the identity of the original (user) requester in the service HTTPS request.
Solution: Use specific headers, X-Tapis-Tenant and X-Tapis-User, to transmit the original requester's identity.



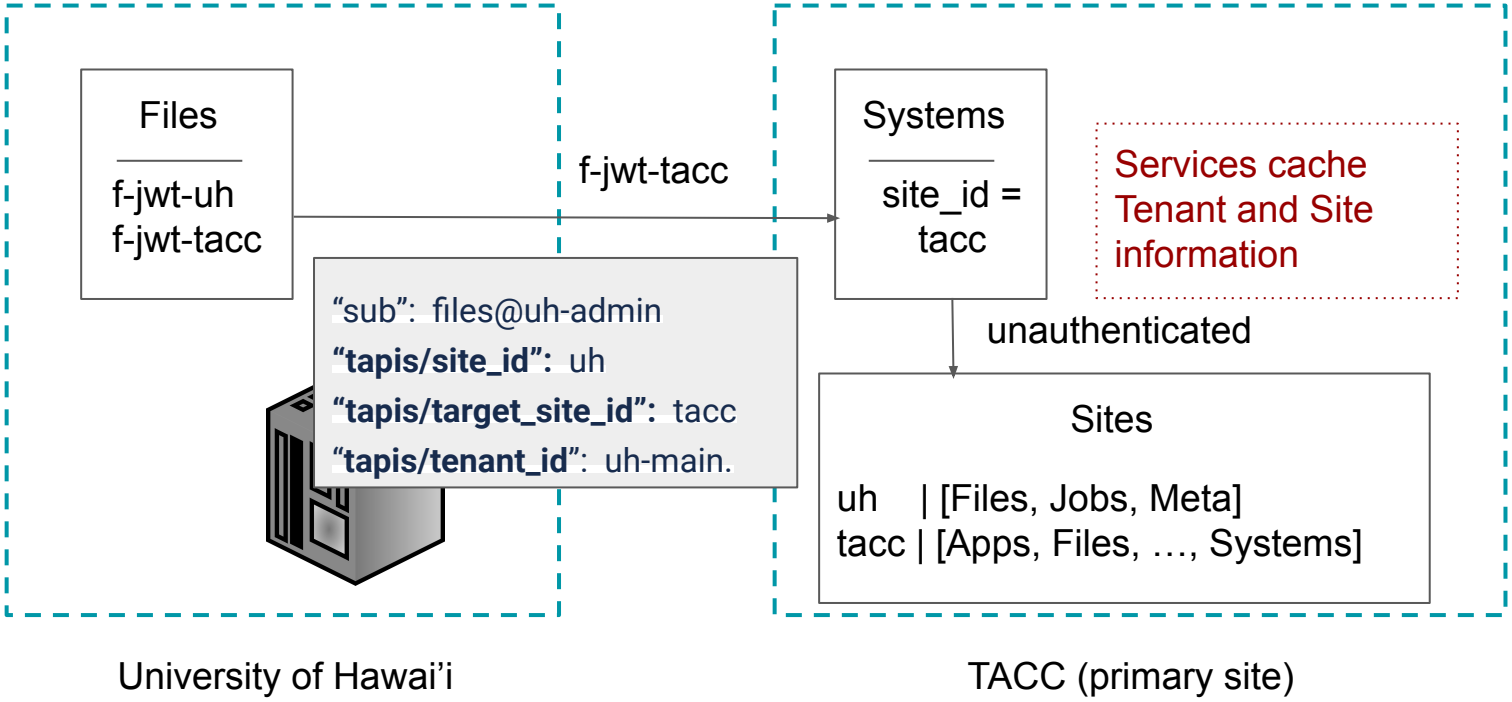
Sending Service-to-Service Requests - Example Case



Sending Service-to-Service Requests (Full Algorithm)

1. Determine the site for the request:
    a. If target service == Tenants -> primary site; If target service == SK, Tokens -> local site.
    b. Determine the tenant of the request: this is the tenant in which the objects of the request (the system(s), app(s), job(s), …) belong.
    c. Determine the site owning the tenant. Each tenant is owned by exactly one site, and this site is available from the Tenants API.
    d. Two case:
        i. If the service being requested is listed as a service run by the site, this is the site.
        ii. Otherwise, the site is the primary site.
2. Send a service JWT with target_site_id claim equal to the site computed in 1.
3. Determine the base URL for the request
    a. If 1ci), use the tenant's base URL.
    b. If 1cii), use the base URL for the tenant at the primary site.

Receiving Service Requests (Validation) - Example Case



Receiving Service Requests (Validation, Full Algorithm)

When a service receives a service request, it performs the following validation

Service Token Validation:

1. Decode the JWT, ignoring the signature, to get all claims.
2. IF tapis/target_site_id ≠ service_config.site_id THEN REJECT
3. Additional checks that this service should be fulfilling this request:
    a. IF service == Tenants then OK IF service_config. running_at_primary_site
    b. ELIF service IN [Tokens, SK] then OK IF request.tenant_id.owning_site == service_config.site_id
    c. ELIF request.tenant_id <=primary_site then OK IF service_config.running_at_primary_site
    d. ELSE (request.tenant_id.site_id == AssociateSite) then OK IF either:
        i. service_config.site_id == AssociateSite AND service IN AssociateSite.services OR
        ii. service_config.running_at_primary_site AND service NOT IN AssociateSite.services
4. *# Validate signature using public key associated with tenant...*
5. *# Check authorizations with the SK at the site…*
    a. *No authz in the JWT (different from scitokens)*

Cross-site Resource Utilization (Future Work)

Goal: Within a single tenant, access systems at both the primary and associate site without sharing secrets beyond the site where system is physically located.

Today:

A. Exclusively use systems at their site (for example, as an associate site)

OR

B. Use systems at TACC and local institution but must share secrets and SSH access with TACC (as a tenant within the TACC site)

Cross-site Resource Utilization (Future Work)

Challenges:

1. Restricted SSH access, including MFA policies, at the local institution.
2. Multiple identities at different institutions

Approaches:

1. System access routed to Tapis agents running at the target site
2. Authorization and secrets data available at the target site
    a. Authorization data mirrored from owning site
    b. Secrets data stored exclusively at target site
3. Identity Mapping and Reconciliation
    a. InCommon
4. Globus Auth and File Transfers

Dynamic Authentication

Goal: Allow SSH access to systems without manual key distribution

Today:

A. Assign user credentials to each system that a user will access

B. Users share a service account (discouraged)

Dynamic Authentication

Challenges: Extending trust relationships to include dynamically generated credentials

- Allow SSH access using credentials created on the fly by trusted components

Approaches:

A. Use Vault CA to create short-lived certificates upon request from Tapis
   a. Systems have to trust Vault CA and Tapis authentication/authorization

B. Use SciTokens to create short-lived tokens
   a. Systems have to trust SciToken issuer and its authentication/authorization
   b. Requires a SciToken PAM module

Thank You

https://tapis-project.org

Contact Us:

Joe Stubbs (jstubbs@tacc.utexas.edu)

Sean Cleveland (seanbc@hawaii.edu)

Richard Cardone (rcardone@tacc.utexas.edu)

Funding

•The Tapis Framework: NSF Office of Advanced CyberInfrastructure #1931439 and #1931575

Backup Slides

Cross-site Resource Utilization (Future Work)

Within a single tenant, access systems at both the primary and associate site without sharing secrets beyond the site where system is physically located.

Challenges:

- We must honor local security requirements, including MFA requirements, when accessing the physical system.
- The API identity may be different from the identity used to access the physical system.
  - For example, a University of Hawaii tenant user authenticates to Tapis as its UH identity but wants to access a TACC system using its TACC identity.
- Tapis must be able to route requests to the site where a system physically resides (not based on the site owning the tenant in which the system is defined).

Solution ingredients:

- Dynamic authentication based on pre-established trust relationship between Tapis and physical systems.
  - OAuthSSH + SciTokens
- MFA integrated into the access token and/or SciToken.
- Identity mapping across identity providers.
- More sophistication in the Tapis API Router.

Using JWTs for Cross-site Resource Utilization (previous)

Future Work

- Within a single tenant, access systems at both the primary and associate site without sharing secrets beyond the site where system is physically located.
  a. MFA exemption is restricted to SSH sessions within the network.
  b. Programmatic MFA
- Dynamic authentication based on pre-established trust relationship between Tapis and systems.
  a. Implement using SciTokens?
- Today, associates site can either:
  a. Exclusively use systems at their site
  b. Use systems at TACC and their site but they have to share secrets and SSH access with TACC
-
- Using JWTs to access resources (systems) across different associate sites.
  a. For example, enable a UH tenant to run jobs on both UH and TACC systems.

Tapis JWTs for Cross-site Service Requests: Challenge

When service A makes a request to service B, it includes its service token in a header in the request.

- This means that service B obtains a token representing service A as part of processing the request. (Service A's token "leaks" to service B).
- In theory, service B could use the token to impersonate service A.
- When the two services are at different sites, this seems especially troublesome.

Our solution: Use a special claim in the JWT, `tapis/target_site_id`, to specify a site where a service JWT is valid.

- Requires services to maintain a JWT for each site they plan to communicate with.
- Requires services to be able to "compute" which site will receive their request.
- Impersonation within a site is still possible.

Tapis JWTs for Cross-site Service Requests: Use Cases

Cross-site service requests are needed to support use cases where sites only run a subset of Tapis services.

Use Case 1 (“Performance Site”). The site runs the Files service but not the Jobs or Systems services.

- This use case support running a minimum number of services while still providing good performance when transferring data to and from storage and execution system within the site.
- The Jobs API running at TACC calls the Files API running at the site.
- The Files API running at the site calls the Systems API running at TACC.

Use Case 2 (“Local Security Site”). The site runs the Systems, Files and Jobs services.

- This use case supports keeping all secrets within the site.
- In this case, cross-site requests can still arise from other services (Actors, Apps, Meta, Streams, etc.)