Spartan - A Scalable Client Authentication & Authorization System for Container Environments

USENIX UCMS15

09 Nov 2015

Aditya Mahendrakar
Binu Ramakrishnan
Yahoo Inc.

Background

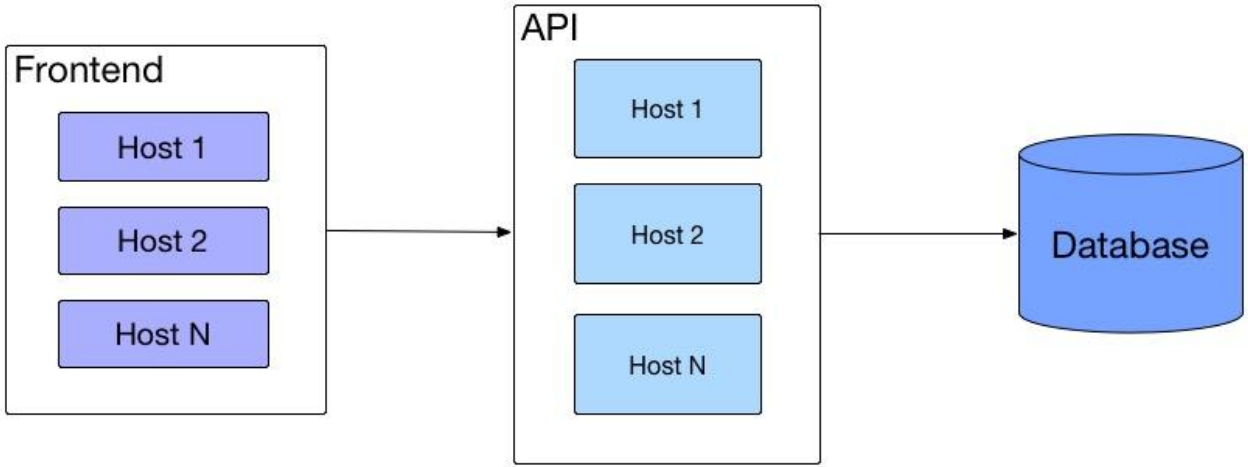
Access Control

- Authentication
- Authorization

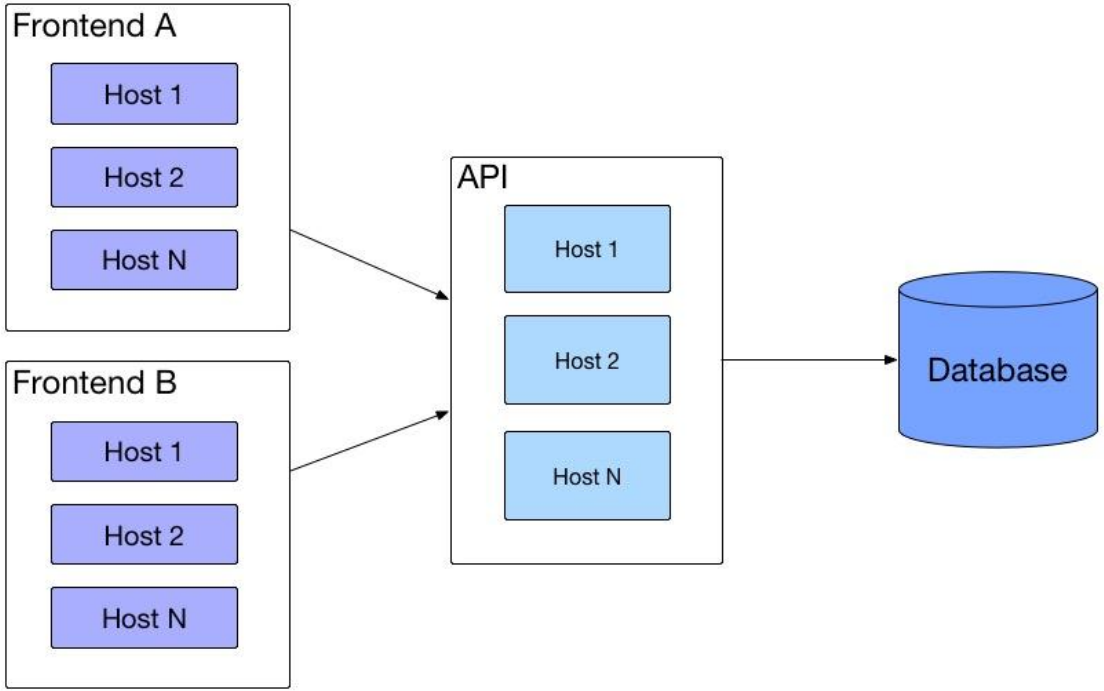


https://pixabay.com/en/policeman-officer-stop-cop-uniform-23796/

Fancy App



Fancy App v2

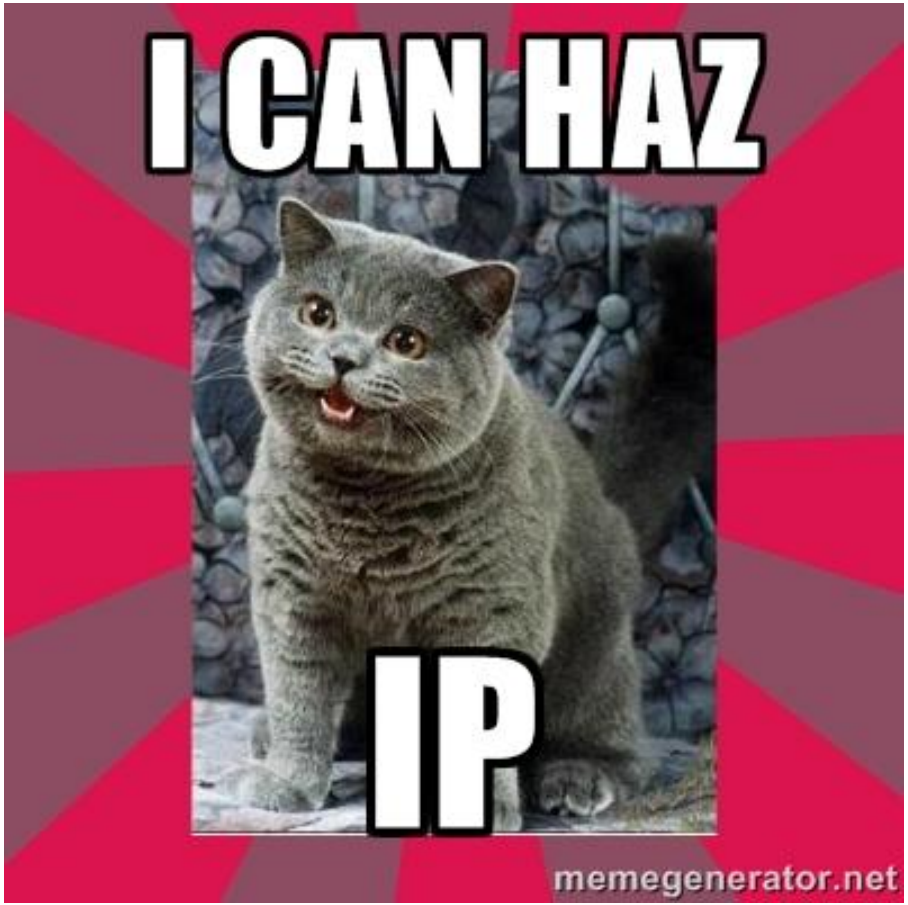

Frontend A
Host 1
Host 2
Host N
Frontend B
Host 1
Host 2
Host N
API
Host 1
Host 2
Host N
Database

IP

- Network firewall or host based ACL
- IP lists with authorization policies

Challenges

- Scale?
- proxies/NAT ?



Custom protocols

Shared secret or basic auth

Challenges

- Key distribution
- Manage policies



https://www.flickr.com/photos/mastermaq/1086323050

Client certificates anyone?



https://www.flickr.com/photos/doctorow/17591930198

Dynamic Environment

Containers are ephemeral

Inherently stateless

Complex network topologies

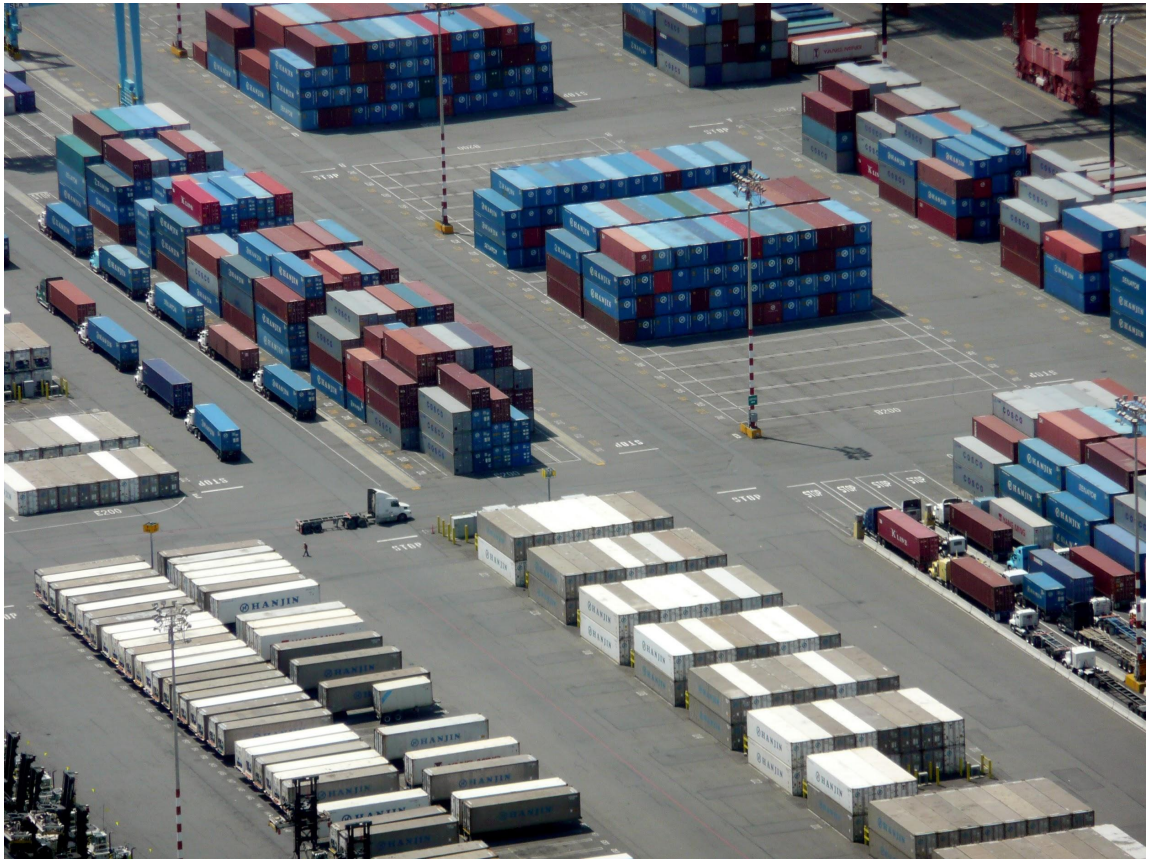


https://www.flickr.com/photos/glynlowe/10921733615/

Common Use cases

CI/CD farm

Production workloads



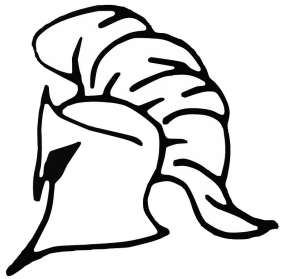
https://www.flickr.com/photos/newton/2595981931

Challenges aka Identity Crisis

- Shared IP
- Short-lived
- Scale



https://www.flickr.com/photos/chrism70/1363593259/



Spartan

Spartan

Spartan is a role based identity system that provides both authentication and authorization capabilities to clients in an automated, easy to configure, scalable fashion.

Use cases

- Enable fine-grained access control for your application
- You use HTTPS, but want to enable client authentication and authorization capabilities
- You are using client IP whitelists for access control, but find it less effective on shared IP environments like containers, NATs etc.
- Spartan as an alternative to manual client IP based access control lists (ACL) in your applications
- Replacement for shared secret based client authentication

Design Features

- Just enough abstraction
- Small code base
- All things JWT
- Security by design
- Protocol agnostic

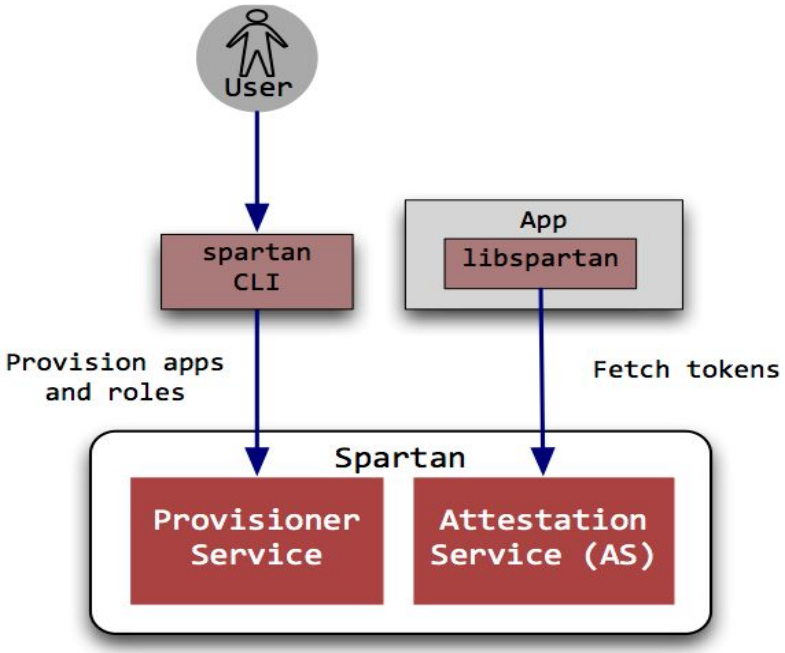
Spartan comprises of

Provisioner service - create and map applications and roles

Command line tool is an interface for the user to provision apps and roles

Attestation service - provides assertion tokens based on app's role membership

Libraries used by client and server applications to fetch and validate tokens from attestation service



Spartan Concepts

- Users
- User Group
- Apps
- Roles



Users



A human or an human agent (headless user)

An employee who operates production systems

Use credentials such as username and passwd for authentication

Users are trusted

User Group

Applications are often supported by a team.

User group represents a team



http://bit.ly/1GOh8I7

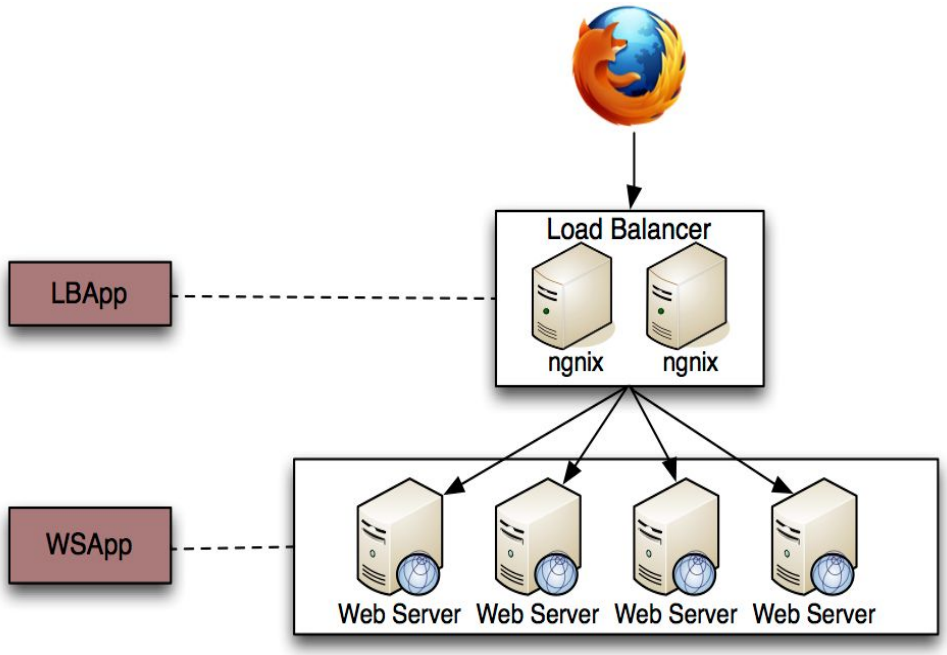
Apps

Represents an application type

Grouping of horizontally scaled application instances

E.g. Web servers, load balancing servers

Identity is based on public key fingerprints



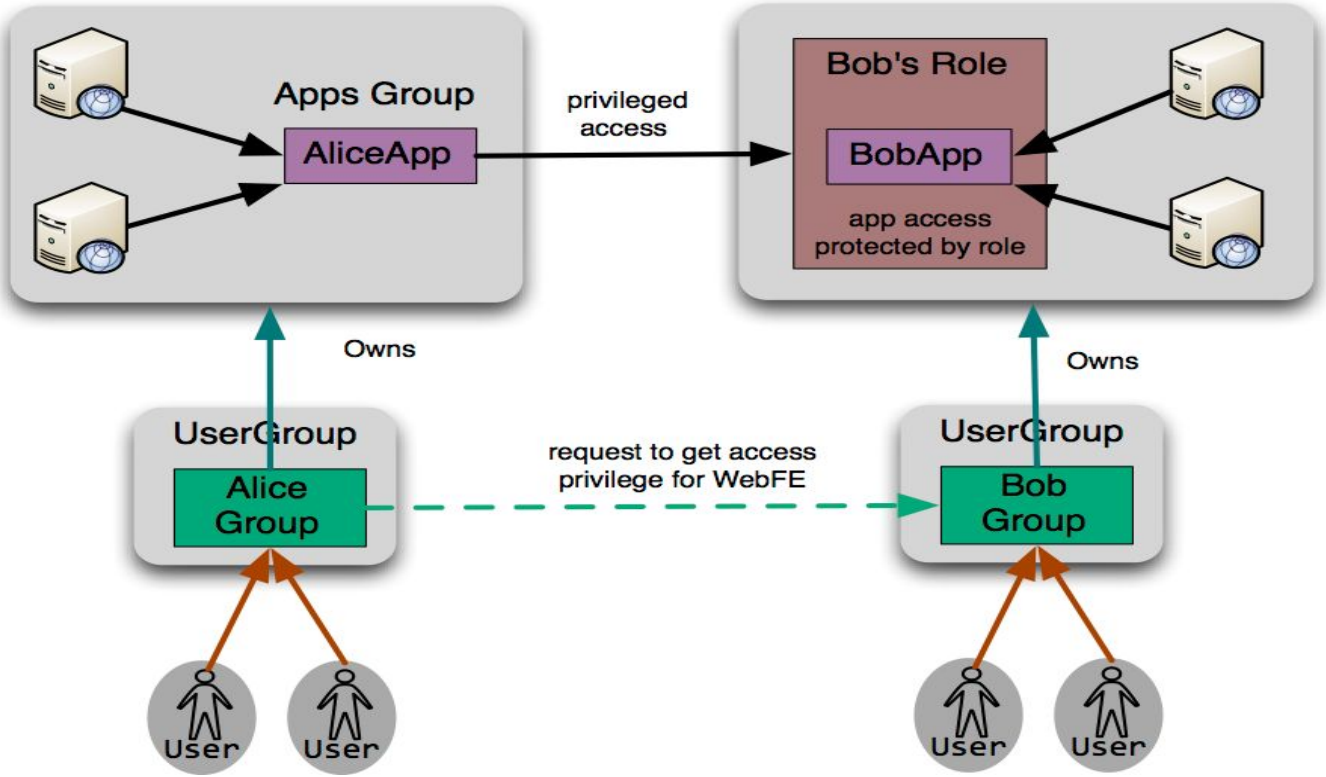
Role

A role represents a privilege to access a protected resource or an operation

A role membership allows an application to acquire permission to access protected resources

Resources: Keys, Files
Operations: API access, Host access

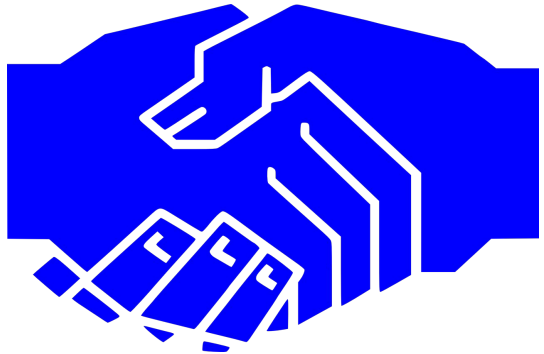
Tying concepts together



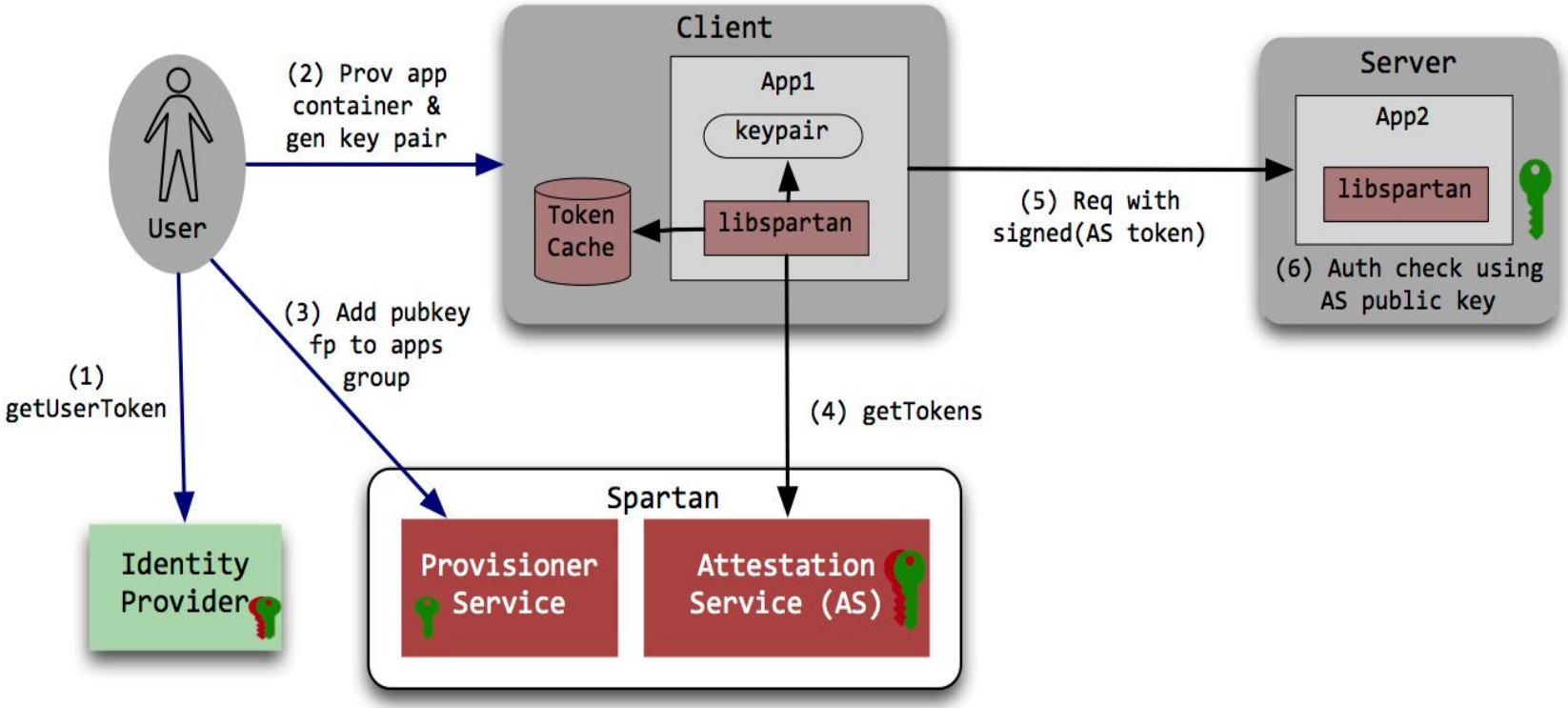
Provisioner service provides APIs to create and manage usergroup, apps, roles and their relationships

Attestation service

- Provides assertion tokens (AS tokens) based on application’s role membership
- Applications use AS public key to validate AS tokens
- Acts as a trust anchor for authorization

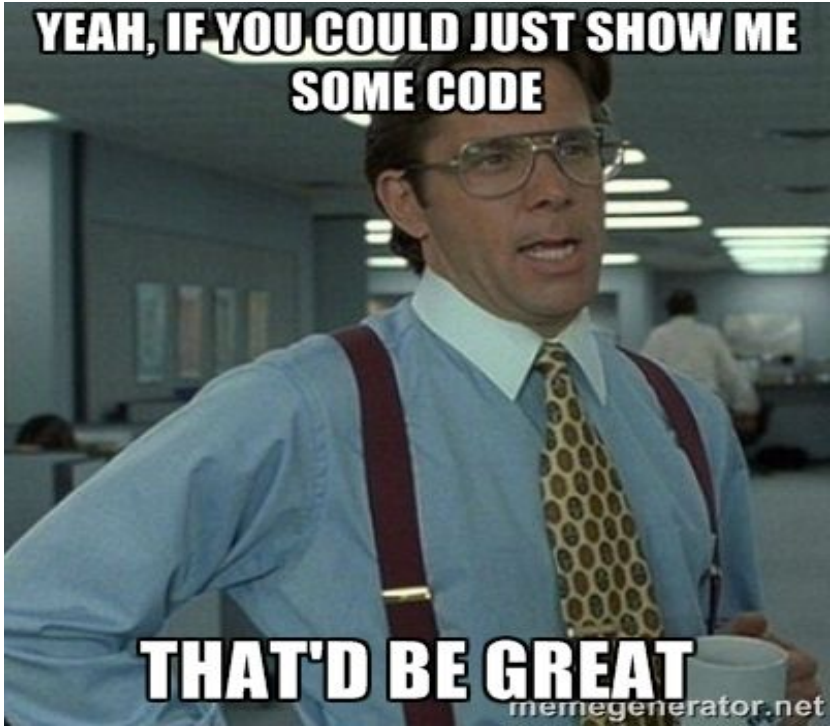


End to end flow



Threat Model

- MITM and replay attacks
  - Use TLS
  - Sign AS token and request body with client’s private key
  - Single use short-lived signed AS tokens (<1 min) with nonce
  - Scoped tokens
- AS private key compromise
  - Deployment related problem
  - Use key management best practices
  - Protect spartan servers
- Crypto related weaknesses
  - Standards based - use of modern crypto technologies such as JWT and ECDSA
  - Risk related to vulnerabilities in open source crypto libraries we use


YEAH, IF YOU COULD JUST SHOW ME SOME CODE
THAT'D BE GREAT
memegenerator.net

Spartan cmd-line interface

```
 SPARTAN_USERID=admin@example.com
 SPARTAN_URL=https://spartan-example.com/v1/

 # create usergroup 'mail-frontend-sre'
 $ spartan -u $SPARTAN_USERID -s $SPARTAN_URL create-usergroup mail-frontend-sre

 # create app 'mail-frontend-app' that is owned by 'mail-frontend-sre'
 $ spartan -u $SPARTAN_USERID -s $SPARTAN_URL create-app mail-frontend-app mail-frontend-
sre

 # create role 'SuperRole' owned by 'mail-frontend-sre',
 # with role handle(service endpoint) being https://mail-example.com and role type being
'SERVICE'
 $ spartan -u $SPARTAN_USERID -s $SPARTAN_URL create-role SuperRole mail-frontend-sre
https://mail-example.com SERVICE

 # map app to role - map app 'mail-frontend-app' to 'SuperRole'
 $ spartan -u $SPARTAN_USERID -s $SPARTAN_URL add-to-role SuperRole mail-frontend-app

 # Step (3) in the end-to-end flow (slide 23)
 # add identities to ‘mail-frontend-app’.
 $ spartan -u $SPARTAN_USERID -s $SPARTAN_URL add-to-app mail-frontend-app ./ES256-app-
pubkey.pem
```

Application client

```
//Step (4) in the end-to-end flow diagram (slide 23)


var request = require('request');
var spartan = require('spartan');

var options = {
 app_privkey: fs.readFileSync('app-privkey.pem'),
 app_pubkey: fs.readFileSync('app-pubkey.pem', 'utf8'),
 as_url: 'https://spartan-example.com/v1/as/tokens',
 cache_path: '~/.spartan'
};

spartan.getToken('SuperRole', options, getTokenCB);
```

```
//Step (5) in the end-to-end flow diagram (slide 23)

var getTokenCB = function (error, token)
{
  var options = {
    uri: 'https://mail-example.com/getmail',
    method: 'GET',
    headers: {
      'x-spartan-auth-token': token
    },
    json: {}
  };

  request(options, function (error, resp, body)
  {
    if (error) { return; }
    // got response from the server
    console.log(body);
  });
};
```

Application server

```
//Step (6) in the end-to-end flow diagram (slide 23)
var spartan = require('spartan');
var router = require('express').Router();
// Attestation server public key.
var as_pubkey = fs.readFileSync('as-pubkey.pem', 'utf8'),

// Parameters to pass for auth
var sp_options = {
      as_pubkey: as_pubkey,
      role: 'SuperRole',
    },

// Spartan express route handler
var sp_handlr = new spartan.RouteHandler(sp_options);

// This is sample on how to authenticate client using express route handler
router.post('/getmail, [sp_handlr.svcAuth.bind(sp_handlr)], function (req, res) {

   // If you reach here, that means you are authorized
   return res.status(200).json({
     msg: 'app is authenticated!'
   });

 });
```



Demo

One more thing...

https://github.com/yahoo/spartan

https://github.com/yahoo/spartan-node

https://github.com/yahoo/spartan-go

Stay tuned and put these repos in your Github watchlist !

Future

- Production ready
- More language bindings
- Explore integration with popular cluster management and deployment systems
- Key exchange support and mutual auth
- Support user auth for applications

Acknowledgements

Yahoo Paranoids

Yahoo Platform Engineering

Thank you

Aditya Mahendrakar
@adityab4u

Binu Ramakrishnan
@securitysauce