Macaroons

Cookies with Contextual Caveats for Decentralized Authorization in the Cloud

Rihan Pereira, MSCS

COMP 524 - Security, Fall 2018
Department of Computer Science
California State University, Channel Islands

November 27, 2018

Agenda

1. Web Cookies
   - Vulnerabilities
   - CSRF attack
   - session limitations
2. Token Authentication
   - OAuth 2.0
   - JSON Web Tokens(JWT)
   - OAuth + JWT
   - OAuth + Signatures
3. Macaroons
   - Design in brief
   - applications
4. References

Web Cookies

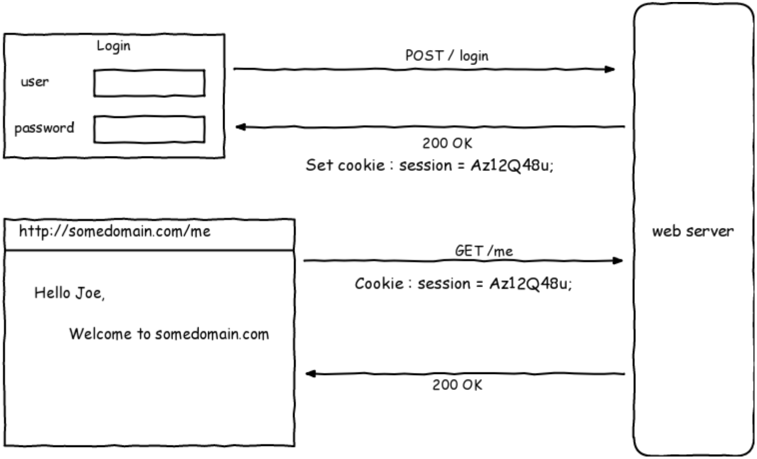
- solves user identity problem in dynamic web sites.

Web Cookies

- solves user identity problem in dynamic web sites.
- fundamentally used to store session IDs

Web Cookies

- solves user identity problem in dynamic web sites.
- fundamentally used to store session IDs
- still in use today!


Login
user
password
POST / login
200 OK
Set cookie : session = Az12Q48u;
http://somedomain.com/me
Hello Joe,
Welcome to somedomain.com
GET /me
Cookie : session = Az12Q48u;
200 OK
web server

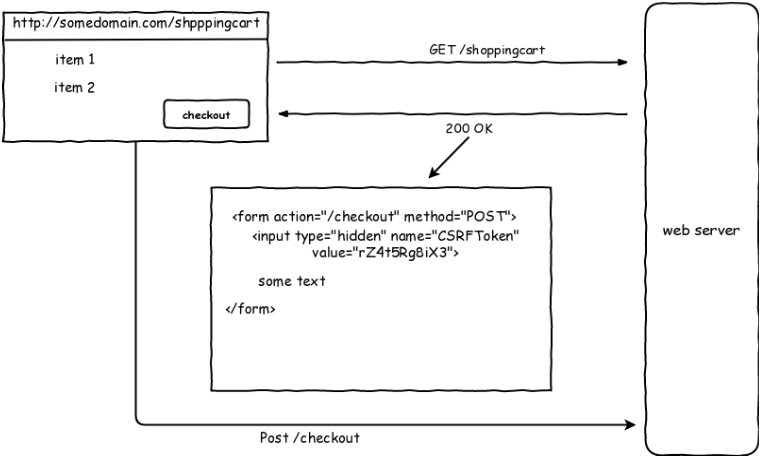
Vulnerabilities

- Man in the middle attack
- Cross site resource fogery(CSRF)

CSRF attack

Executes unwanted actions on a dynamic site in which they are currently authenticated.

fix 1 - using a csrf token

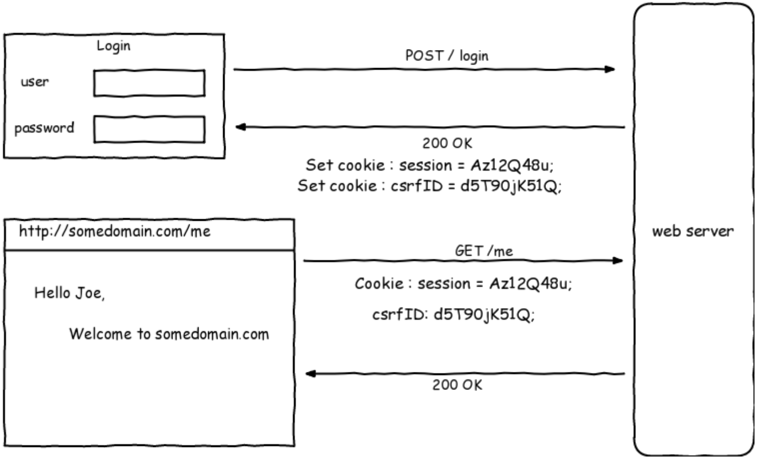


```
<form action="/checkout" method="POST">
    <input type="hidden" name="CSRFToken"
            value="rZ4t5Rg8iX3">

      some text

</form>
```

fix 2 - double submit cookie


Login
user
password
POST / login
200 OK
Set cookie : session = Az12Q48u;
Set cookie : csrfID = d5T90jK51Q;
http://somedomain.com/me
Hello Joe,
Welcome to somedomain.com
GET /me
Cookie : session = Az12Q48u;
csrfID: d5T90jK51Q;
200 OK
web server

session limitations

- web cookies are opaque

session limitations

- web cookies are opaque
- dont solve access-control problem

session limitations

- web cookies are opaque
- dont solve access-control problem
- lookup server state on every request

session limitations

- web cookies are opaque
- dont solve access-control problem
- lookup server state on every request
- really not good for distributed/clustered applications

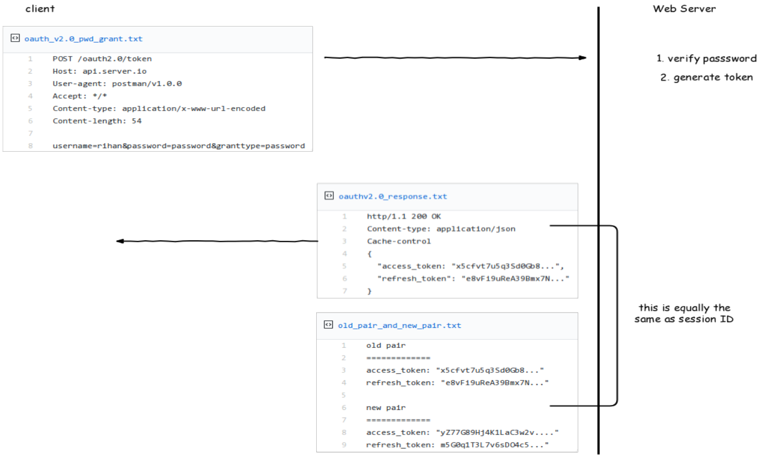
Token Authentication

- self-contained chunk of information

Token Authentication

- self-contained chunk of information
- intrinsic value in that string

OAuth 2.0



oauth_v2.0_pwd_grant.txt

```
POST /oauth2.0/token
Host: api.server.io
User-agent: postman/v1.0.0
Accept: */*
Content-type: application/x-www-url-encoded
Content-length: 54

username=rihan&password=password&granttype=password
```

oauthv2.0_response.txt

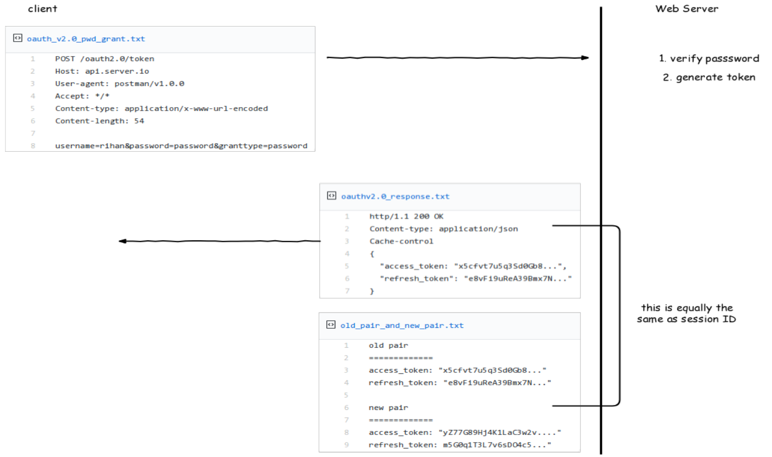
```
http/1.1 200 OK
Content-type: application/json
Cache-control
{
  "access_token: "x5cfvt7u5q3Sd0Gb8...",
  "refresh_token": "e8vFi9uReA39Bmx7N..."
}
```

old_pair_and_new_pair.txt

```
old pair
=============
access_token: "x5cfvt7u5q3Sd0Gb8..."
refresh_token: "e8vFi9uReA39Bmx7N..."

new pair
=============
access_token: "yZ77G89Hj4K1LaC3w2v...."
refresh_token: m5G0q1T3L7v6sD04c5..."
```

OAuth 2.0



oauth_v2.0_pwd_grant.txt

```
POST /oauth2.0/token
Host: api.server.io
User-agent: postman/v1.0.0
Accept: */*
Content-type: application/x-www-url-encoded
Content-length: 54

username=rihan&password=password&granttype=password
```

oauthv2.0_response.txt

```
http/1.1 200 OK
Content-type: application/json
Cache-control
{
  "access_token: "x5cfvt7u5q3Sd0Gb8...",
  "refresh_token": "e8vFi9uReA39Bmx7N..."
}
```

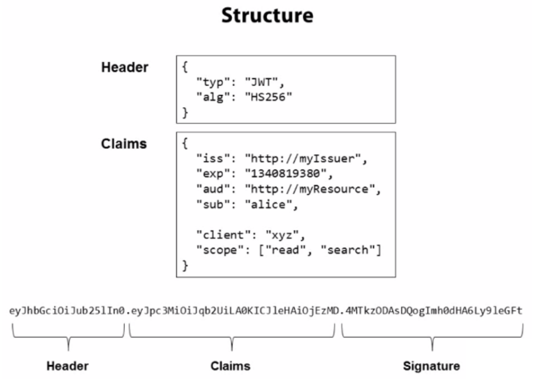
old_pair_and_new_pair.txt

```
old pair
============
access_token: "x5cfvt7u5q3Sd0Gb8..."
refresh_token: "e8vFi9uReA39Bmx7N..."

new pair
============
access_token: "yZ77G89Hj4K1LaC3w2v...."
refresh_token: m5G0q1T3L7v6sD04c5..."
```

using access token - Authorization Bearer "x5cfvt......"

JSON Web Tokens(JWT)



```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

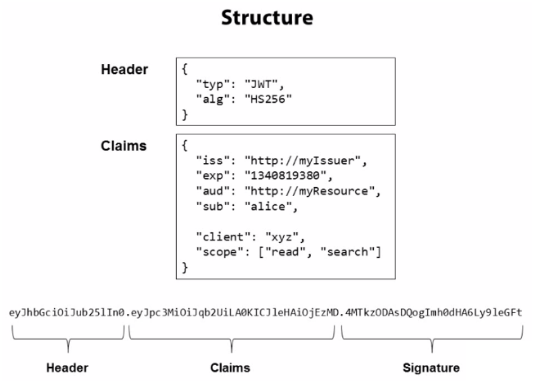
```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",
  "sub": "alice",

  "client": "xyz",
  "scope": ["read", "search"]
}
```

```
eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt
```

- URL-safe, self-contained string, digitally signed.

JSON Web Tokens(JWT)



```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

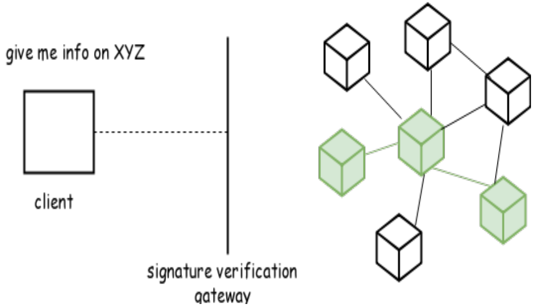
```
{
  "iss": "http://myIssuer",
  "exp": "1340819380",
  "aud": "http://myResource",
  "sub": "alice",

  "client": "xyz",
  "scope": ["read", "search"]
}
```

```
eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt
```

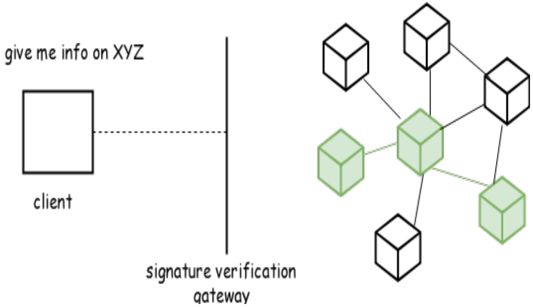
- URL-safe, self-contained string, digitally signed.
- client aware access-control.

OAuth + JWT



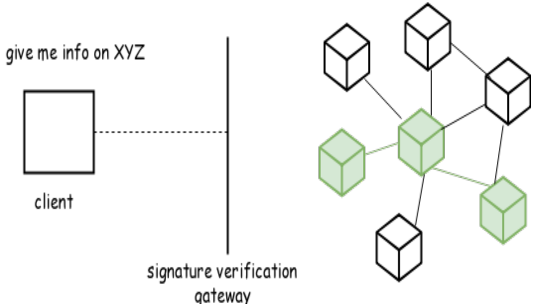
- 3 tier access token

OAuth + JWT



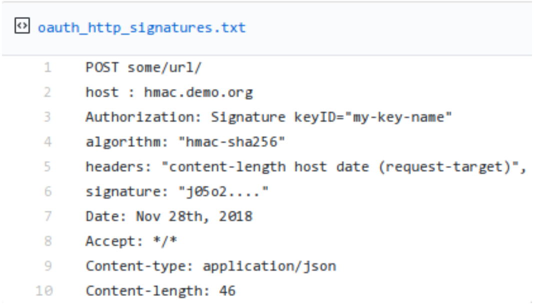
- 3 tier access token
- Instead of state on the server side, state is on the client side

OAuth + JWT



- 3 tier access token
- Instead of state on the server side, state is on the client side
- reduced data access scope

OAuth + Signatures

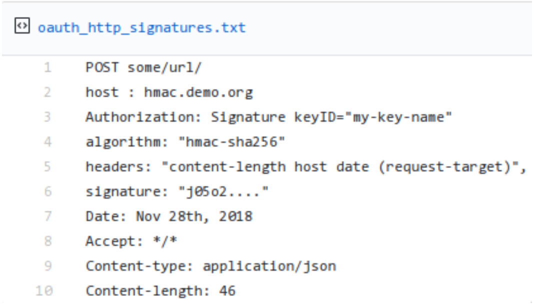


oauth_http_signatures.txt

```
POST some/url/
host : hmac.demo.org
Authorization: Signature keyID="my-key-name"
algorithm: "hmac-sha256"
headers: "content-length host date (request-target)",
signature: "j05o2...."
Date: Nov 28th, 2018
Accept: */*
Content-type: application/json
Content-length: 46
```

- no secret sent over the wire

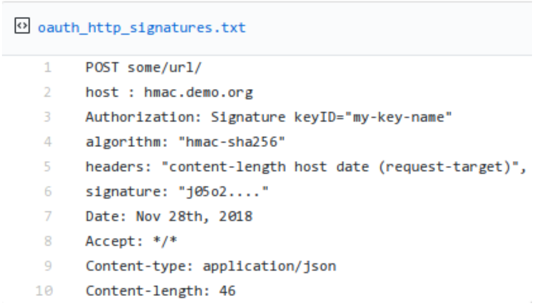
OAuth + Signatures


oauth_http_signatures.txt

```
POST some/url/
host : hmac.demo.org
Authorization: Signature keyID="my-key-name"
algorithm: "hmac-sha256"
headers: "content-length host date (request-target)",
signature: "j05o2...."
Date: Nov 28th, 2018
Accept: */*
Content-type: application/json
Content-length: 46
```

- no secret sent over the wire
- symmetric key used between trusted entities

OAuth + Signatures

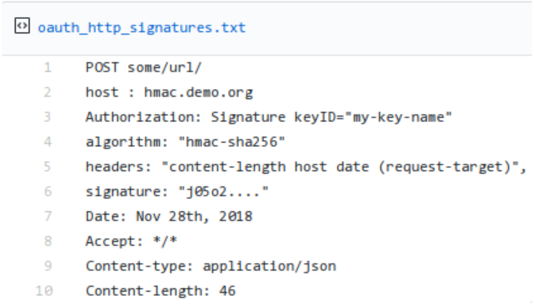


oauth_http_signatures.txt

```
POST some/url/
host : hmac.demo.org
Authorization: Signature keyID="my-key-name"
algorithm: "hmac-sha256"
headers: "content-length host date (request-target)",
signature: "j05o2...."
Date: Nov 28th, 2018
Accept: */*
Content-type: application/json
Content-length: 46
```

- no secret sent over the wire
- symmetric key used between trusted entities
- stateless

OAuth + Signatures


oauth_http_signatures.txt

```
POST some/url/
host : hmac.demo.org
Authorization: Signature keyID="my-key-name"
algorithm: "hmac-sha256"
headers: "content-length host date (request-target)",
signature: "j05o2...."
Date: Nov 28th, 2018
Accept: */*
Content-type: application/json
Content-length: 46
```

- no secret sent over the wire
- symmetric key used between trusted entities
- stateless
- driving modern REST security these days.

Introduction

- restricted bearer tokens/credentials allowing delegation

Introduction

- restricted bearer tokens/credentials allowing delegation
- embedded caveats(a.k.a claims) which attenuate the scope

Introduction

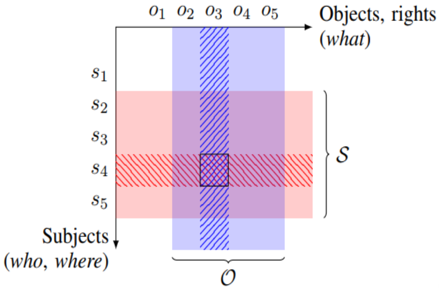
- restricted bearer tokens/credentials allowing delegation
- embedded caveats(a.k.a claims) which attenuate the scope
- used only in certain context, hence confinement

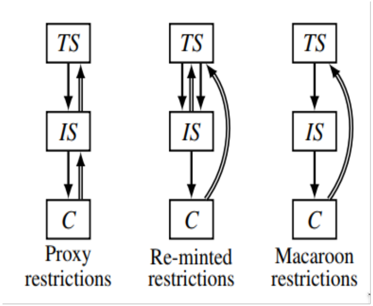
The bearer client C can access a resource at a target service(TS) as long as

- *as long as* the operation is read
- *as long as* the object is *privateImage.jpg*
- *as long as* the user at C is logged into service A
- *as long as* that logged-in user is in group G at A
- *as long as* the above proofs are recent/fresh enough
- and, *as long as* the user at C didn't just log out of A

The bearer client C can access a resource at a target service(TS) as long as

- *as long as* the operation is read
- *as long as* the object is *privateImage.jpg*
- *as long as* the user at C is logged into service A
- *as long as* that logged-in user is in group G at A
- *as long as* the above proofs are recent/fresh enough
- and, *as long as* the user at C didn't just log out of A


o_1 o_2 o_3 o_4 o_5
Objects, rights (*what*)
s_1 s_2 s_3 s_4 s_5
$\mathcal{S}$
Subjects (*who*, *where*)
$\mathcal{O}$



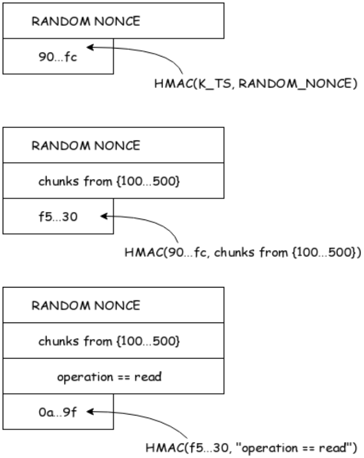
TS - Target Service

IS - Intermediary service

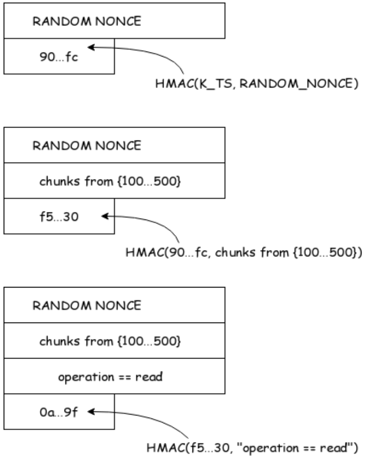
C - Client

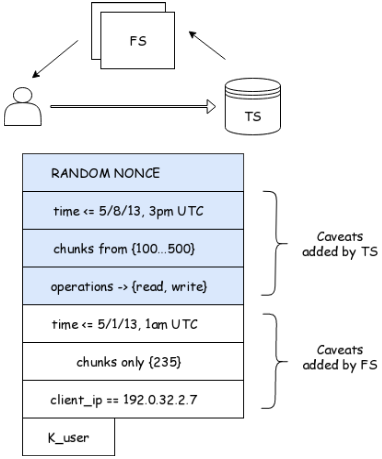
Figure: source - macaroons research paper

Design in brief



Design in brief





applications

Macaroons *vs* Web Cookies

applications

Macaroons *vs* Web Cookies

- ensure integrity even when holding state at the client side

applications

Macaroons *vs* Web Cookies

- ensure integrity even when holding state at the client side
- adding caveats to limit the usefulness of stolen cookies

applications

Macaroons *vs* Web Cookies

- ensure integrity even when holding state at the client side
- adding caveats to limit the usefulness of stolen cookies

OAuth vs Macaroons

applications

Macaroons *vs* Web Cookies

- ensure integrity even when holding state at the client side
- adding caveats to limit the usefulness of stolen cookies

OAuth vs Macaroons

- user-level access tokens: embedding caveats into the access token and mandating validation at the resource server.

applications

Macaroons *vs* Web Cookies

- ensure integrity even when holding state at the client side
- adding caveats to limit the usefulness of stolen cookies

OAuth vs Macaroons

- user-level access tokens: embedding caveats into the access token and mandating validation at the resource server.
- agents: embedding security guidelines as caveats when agents participate in OAuth flows.

- Arnar Birgisson, Joe Gibbs Politz, Ulfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner, *Macaroons: Cookies with Contextual Caveats for Decentralized Applications in the Cloud*, **2015**.
- https://www.owasp.org
- https://jwt.io/
- http://aosabook.org
- https://oauth.net/2/

Thank you! Questions ?