Macaroons
More Cookie Than Cookie

Brian Sletten (@bsletten)
Speaker Qualifications

- Specialize in next-generation technologies
- Author of O’Reilly Videos on Hypermedia, Linking Data, Security and Encryption
- Author of ‘Resource-Oriented Architecture Patterns for Webs of Data’
- Teaches and speaks internationally about REST, Semantic Web, Data Science, Security, Visualization, Architecture
- Worked in Defense, Finance, Retail, Hospitality, Video Game, Health Care, Telecommunications and Publishing Industries
- International Pop Recording Artist

“Controlled sharing is fundamental to distributed systems; yet, on the Web, and in the Cloud, sharing is still based on rudimentary mechanisms.”

Birgisson et al
Solution

- Identity
- Authorization
- Contextual Confinement
- Decentralization
- Statelessness
POST /auth HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1)
  AppleWebKit/602.2.14 (KHTML) Version/10.0.1 Safari/602.2.14
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
user=scott&password=tiger

GET /auth HTTP/1.1
Host: localhost
Authorization: c2NvdHJvbGQ=

HTTP/1.0 200 OK
Set-Cookie: session=Rg3vHJZnehYLjVg7qi3bZjzg;
  Expires=Tue, 15 Jan 2017 21:47:38 GMT; Path=/; Domain=.example.com;
HttpOnly; Secure

Bearer Tokens

- OAuth Web Resource Authorization Profiles (WRAP)
- Requires TLS
- Certificate chain validation
OAuth 2.0 Roles

Credit: http://tutorials.jenkov.com/images/oauth2
hash = H(message)

Hash Functions

- Generate a fixed-size output from arbitrary input
- Computationally unfeasible to reverse a hash
- Computationally unfeasible to generate a message for a given hash
- Computationally unfeasible to find two inputs that generate the same hash
hash = H(secret | message)

https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks
HMAC(K, m) = H((K' ⊕ opad) || H((K' ⊕ ipad) || m))

**HMAC in Java**

```java
public class HMAC {
    public static String generateHMAC(String algorithm, String message, String key)
            throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException {
        String returnValue;
        String hmacAlgo = "Hmac" + algorithm;
        SecretKeySpec sks = new SecretKeySpec(key.getBytes("UTF-8"), hmacAlgo);
        Mac mac = Mac.getInstance(hmacAlgo);
        mac.init(sks);
        byte[] bytes = mac.doFinal(message.getBytes("ASCII"));
        StringBuffer hash = new StringBuffer("0x");
        for (int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(0xFF & bytes[i]);
            if(hex.length() == 1) { hash.append('0'); } else hash.append(hex);
        }
        return returnValue = hash.toString();
    }
    ...
```
HMAC in Java

```java
public class HMAC {
    ...
    public static void main(String[] args) throws Exception {
        String hmac = generateHMAC("MD5", "The quick brown fox jumps over the lazy dog", "key");
        System.out.println("MD5: " + hmac);
        hmac = generateHMAC("SHA1", "The quick brown fox jumps over the lazy dog", "key");
        System.out.println("SHA1: " + hmac);
        hmac = generateHMAC("SHA256", "The quick brown fox jumps over the lazy dog", "key");
        System.out.println("SHA256: " + hmac);
    }
}
```

MD5: 0x80070713463e7749b90c2dc24911e275
SHA1: 0xde7c9b85b878ae6bc8a7a36f70a90701c9db4d9
SHA256: 0xf7bc83f430538424b13298e60a6fb143ef4d59a14946175997479dbc2d1a3cd8

JSON Web Tokens (JWT)

- RFC 7519
- Compact JSON-based solution
- Authentication
- Claims
- Stateless
- Signed
JWT

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

```json
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

JWT

```text
JWT

base64UrlEncode(header).base64UrlEncode(payload).HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

JWT

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA950rM7E2cBab30RMHrHDCeFxj0ZgeFONFh7HgQ
```

```text
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiYWRtaW4iOnRydWV9.TJVA950rM7E2cBab30RMHrHDCeFxj0ZgeFONFh7HgQ
```
What's good about cookies and regular bearer tokens?

- Simple
- Ease of adoption
- User transparency
What's wrong with cookies and regular bearer tokens?

- Tied to a particular domain/identity
- No Audience Restrictions
- Centralized lookup
- Revocation

“Thus, today, it is practically impossible for the owner of a private, sensitive image stored at one Cloud service to email a URL link to that image, safely - given the many opportunities for impersonation and eavesdropping - such that the image can be seen only by logged-in members of a group of users that the owner maintains at another, unrelated Cloud service.”

Birgisson et al
“Currently this use case is possible only if the image, access group, and users are all at a single service, or if two Cloud services keep special, pairwise ties using custom, proprietary mechanisms (e.g., as done by Dropbox and Facebook)”

Birgisson et al
\[
\text{RANDOM\_NONCE}
\]

\[
90..fc \xrightarrow{\text{HMAC(Kr, RANDOM\_NONCE)}}
\]

\[
\text{RANDOM\_NONCE}
\]

\[
\text{chunk} \in \{100..500\}
\]

\[
f5..30 \xrightarrow{\text{HMAC(90..fc, "chunk \in \{100..500\}"}}
\]
RANDOM_NONCE
chunk $\in (100..500)$
operation == read

$0a..9f \rightarrow$ HMAC(F5..30, "operation == read")

RANDOM_NONCE

- time $\leq$ 5/8/13, 3pm GMT
- chunk $\in \{100..500\}$
- operation $\in \{\text{read, write}\}$

- time $\leq$ 5/1/13, 1am GMT
- chunk $\in \{235\}$
- client_ip == 192.0.32.7

\[ \text{KUSER} \]

- Caveats added by TS
- Caveats added by FS
Creating a Macaroon

```python
>>> import macaroons
>>> secret = 'this is our super secret key; only we should know it'
>>> public = 'we used our secret key'
>>> location = 'http://mybank/'
>>> M = macaroons.create(location, secret, public)

>>> M.identifier
'we used our secret key'

>>> M.location
'http://mybank/

>>> M.signature
'e3d9e02908526c4c0039ae15114115d97fdd68bf2ba379b342aaf0f617d0552f'
```

Printing a Macaroon

```python
>>> M.serialize(format=1)
'MDAxYzcuY2F0aW9uIGh0dHA6Ly9teWJhbmsvCjAwMjZpZGVudGlmaWVy
IHdlIHVzZWQgb3VyIHNlY3JldCBrZXkKMDAyZnNpZ25hdHVyZSd2eApC
FJstAA5rhURQRXzF91ovyujebNCqvD2F9BVLwo'

>>> print M.inspect()
location http://mybank/
identifier we used our secret key
signature e3d9e02908526c4c0039ae15114115d97fdd68bf2ba379b342aaf0f617d0552f'```
Adding Caveats

```python
>>> M = M.add_first_party_caveat('account = 3735928559')
>>> print(M.inspect())
location http://mybank/
identifier we used our secret key
cid account = 3735928559
signature 1efe4763f290dbce0c1d08477367e11f4e933cf662d79772dbb82128
```

Adding More Caveats

```python
>>> M = M.add_first_party_caveat('time < 2020-01-01T00:00')
>>> M.signature
'b5f06c8c8ef92f6c826f282cd1f8bd1849301d09a2db634ba182536a611c49'
>>> M = M.add_first_party_caveat('email = alice@example.org')
>>> M.signature
'ddf553e46083e55b8d71ab822be3d8fcf21d6bf19c40d617bb9fb438934474b6'
>>> print(M.inspect())
location http://mybank/
identifier we used our secret key
cid account = 3735928559
cid time < 2020-01-01T00:00
cid email = alice@example.org
signature ddf553e46083e55b8d71ab822be3d8fcf21d6bf19c40d617bb9fb438934474b6
```
Verifying Caveats

```python
>>> msg = M.serialize(format=1)
>>> # send msg to the bank

>>> M = macaroons.deserialize(msg)
>>> print M.inspect()
location http://mybank/
identifier we used our secret key
cid account = 3735928559
cid time < 2020-01-01T00:00
cid email = alice@example.org
signature ddf553e46083e55b8d71ab822be3d8fcf21d6bf19c40d617bb9fb438934474b6

Verifying Caveats

```
Verifying Caveats

```python
>>> import datetime
>>> def check_time(caveat):
...     if not caveat.startswith('time < '):
...         return False
...     try:
...         now = datetime.datetime.now()
...         when = datetime.datetime.strptime(caveat[7:], '%Y-%m-%dT%H:%M')
...         return now < when
...     except:
...         return False
...
>>> V.satisfy_general(check_time)
>>> V.verify(M, secret)
True
```

Verifying Additional Caveats

```python
>>> N = M.add_first_party_caveat('action = deposit')
>>> V.verify(N, secret)
True
```
Macaroon Uses

- Shared tokens across apps, domains, microservices and boundaries
- Cookie Replacement (Statelessness and Token Theft)
- OAuth2 Bearer Tokens (Centralization and Token Theft)
- HyperDex

Questions?

✉️ brian@bosatsu.net
🐦 @bsletten
🔗 http://tinyurl.com/bjs-gplus
䠴 bsletten