Security

SWE 432, Fall 2017
Design and Implementation of Software for the Web
Today

• Security
  • What is it?
  • Most important types of attacks
• Authorization
  • oAuth
Security

- Why is it important?
  - Users’ data is on the web
    - Blog comments, FB, Email, Banking, …
  - Can others steal it?
    - or who already has access?
  - Can others impersonate the user?
    - e.g., post on FB on the user’s behalf
Security Requirements for Web Apps

1. Authentication
   • Verify the **identify** of the parties involved
   • Who is it?

2. Authorization
   • Grant **access** to resources only to allowed users
   • Are you allowed?

3. Confidentiality
   • Ensure that **information** is given only to authenticated parties
   • Can you see it?

4. Integrity
   • Ensure that information is **not changed** or tampered with
   • Can you change it?
Threat Models

• What is being defended?
  • What resources are important to defend?
  • What malicious actors exist and what attacks might they employ?

• Who do we trust?
  • What entities or parts of system can be considered secure and trusted
  • Have to trust something!
Web Threat Models: Big Picture

client page (the “user”) —> HTTP Request —> server

HTTP Response —> server

GMU SWE 432 Fall 2017
Web Threat Models: Big Picture

client page (the “user”)

HTTP Request

server

HTTP Response

Do I trust that this request *really* came from the user?
Web Threat Models: Big Picture

HTTP Request

client page (the “user”)

Do I trust that this request really came from the user?

server

HTTP Response

Do I trust that this response really came from the server?
Web Threat Models: Big Picture

client page (the “user”)

Do I trust that this response really came from the server?

malicious actor “black hat”

Do I trust that this request really came from the user?

server
Web Threat Models: Big Picture

**HTTP Request**

*client page (the “user”)*

**HTTP Response**

*malicious actor “black hat”*

**HTTP Request**

*server*

Might be “man in the middle” that intercepts requests and impersonates user or server.

Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?
Security Requirements for Web Apps

1. Authentication
   • Verify the **identify** of the parties involved
   • Threat: Impersonation. A person pretends to be someone they are not.

2. Authorization

3. Confidentiality
   • Ensure that **information** is given only to authenticated parties
   • Threat: Eavesdropping. Information leaks to someone that should not have it.

4. Integrity
   • Ensure that information is **not changed** or tampered with
   • Threat: **Tampering**.
Integrity and Confidentiality

What if malicious actor impersonates server?

HTTP Request

client page (the “user”)

HTTP Response

malicious actor “black hat”

HTTP Request

server

HTTP Response
Man in the middle

• Requests to server intercepted by man in the middle
  • Requests forwarded
  • But… response containing code edited, inserting malicious code
• Or could
  • Intercept and steal sensitive user data
HTTPS: HTTP over SSL

- Establishes secure connection from client to server
  - Uses SSL to encrypt traffic
- Ensures that others can’t impersonate server by establishing certificate authorities that vouch for server.
- Server trusts an HTTPS connection iff
  - The user trusts that the browser software correctly implements HTTPS with correctly pre-installed certificate authorities.
  - The user trusts the certificate authority to vouch only for legitimate websites.
  - The website provides a valid certificate, which means it was signed by a trusted authority.
  - The certificate correctly identifies the website (e.g., certificate received for “https://example.com" is for "example.com" and not other entity).
Using HTTPS

• If using HTTPS, important that all scripts are loaded through HTTPS
  • If mixed script from untrusted source served through HTTP, attacker could still modify this script, defeating benefits of HTTPS

• Example attack:
  • Banking website loads jQuery through HTTP from a CDN rather than HTTPS
  • Attacker intercepts request for jQuery script, replaces with malicious script that steals user data or executes malicious action
Authentication

• How can we know the identity of the parties involved
• Want to customize experience based on identity
  • But need to determine identity first!
• Options
  • Ask user to create a new username and password
    • Lots of work to manage (password resets, storing passwords securely, …)
    • Hard to get right (#2 on the OWASP Top 10 Vulnerability List)
  • User does not really want another password…
  • Use an authentication provider to authenticate user
    • Google, FB, Twitter, Github, …
Authentication Provider

• Creates and tracks the identity of the user

• Instead of signing in directly to website, user signs in to authentication provider
  • Authentication provider issues token that uniquely proves identity of user
  • Talk more next lecture about how these tokens work
Microservices & Authentication

- If using microservices, how do we decide who is logged in?
- Typical solution: Sign-on gateway
Microservices & Authentication

- If using microservices, how do we decide who is logged in?
- Typical solution: Sign-on gateway
Authentication & Authorization

• Putting this sign on gateway will ensure that people are signed in
• But how do we ensure that someone is **authorized** to view some given data or make some request?
• Role of individual services to check back (either with authorization service, or some other service)
Bigger picture - authentication with multiple service providers

• Let’s consider updating a Todos app so that it can automatically put calendar events on a Google Calendar

Prof LaToza

Logs into, posts new todo

Todos

REST service

Connects as user, creates new event

Google Calendar API

How does Todos tell Google that it’s posting something for Prof LaToza? Should Prof LaToza tell the Todos app his Google password?
We’ve got something for that...
OAuth

- OAuth is a standard protocol for sharing information about users from a "service provider" to a "consumer app" without them disclosing their password to the consumer app

- 3 key actors:
  - User, consumer app, service provider app
  - E.x. "Prof LaToza," "Todos App," "Google Calendar"

- Service provider issues a token on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user
- Protocol could be considered a conversation…
An OAuth Conversation

Goal: **TodosApp** can post events to **User’s** calendar. **TodosApp** never finds out **User’s** email or password

1: intent
2: permission (to ask)
3: redirect to provider
4: permission to share
5: token created
6: Access resource

**TodosApp**

**Google Calendar**

**User**
Tokens?

A token is a **secret value**. Holding it gives us access to some privileged data. The token identifies our users and app.

Example token:

eyjhbGciOiJSUzI1NiIsImtpZCI6ImU3Yjg2NjJfMjMGUwM2Y3ZTk3NjQyNGUzWFiMzI5OIXzNzRhNGVlNWUifQ.ejpc3MioiJodHRwczovL3N1Y3VyZXRva2VuLm9vb2dsZS5jb20vXYVoAGRlW8tNzJhNDIiLCJwYWNrYW5zZXRva2VuLm9vb2dsZS5jb20vYXNzd29yZS5jb20vSm9uY2Fwcy5jb20vSm9uY2Fwcy5jb20vMTIwMjA0MjE1NzYyMDE1Ny8yMjU4NjE1MjczOTkxMzY2OjRfcmFjY2Vzc2FnZS5kYXRhLm5ldC50eXBlL3RhdGFzZS9QcmVzaXM/JmQrQiu995Tdx64bTygA3xxDcuH2

Decoded:

```
"iss": "https://securetoken.google.com/authdemo-72a42",
"name": "Thomas LaToza",
"picture": "https://lh5.googleusercontent.com/-m-OocFU5GLw/AAAAAAAAAAI/AAAAAAAAAH0/BUWkN6DmMRk/photo.jpg",
"aud": "authdemo-72a42",
"auth_time": 1477529371,
"user_id": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
"sub": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
"iat": 1477530885,
"exp": 1477534485,
"email": "latoza@gmail.com",
"email_verified": true,
"firebase": {"identities": {"google.com": ["109040352574312154216"], "email": ["latoza@gmail.com"]}, "sign_in_provider": "google.com"},
"uid": "JMQrQiu9SRTdx64bTygA3xxDcuH2"
```
Trust in OAuth

• How does the Service provider (Google calendar) know what the TodosApp is?

• Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider

• Let the user decide
  • … they were the one who clicked the link after all
Authentication as a Service

• Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service

• Why?
  • Security
  • Reliability
  • Convenience

• We can use OAuth for this!
• We’re going to use Firebase’s authentication API in our homework this week
Using an Authentication Service

1: intent
2: permission (to ask)
3: redirect to provider
4: permission to share
5: token created
6: Access resource

User

Firebase
Firebase Authentication

- Firebase provides an entire suite of authentication services you can use to build into your app
- Can either use “federated” logins (e.g. login with google, facebook, GitHub credentials) or simple email/password logins. Use whichever you want.
- Getting started guide: https://github.com/firebase/FirebaseUI-Web
- For backend: https://firebase.google.com/docs/auth/server/verify-id-tokens
- Firebase handles browser local storage to track that the user is logged in across pages (woo)
Authentication: Sharing data between pages

- Browser loads many pages at the same time.
- Might want to share data between pages
  - Popup that wants to show details for data on main page
  - Cookies that let user login once for a page and still be logged in when visiting page in separate tab
- Attack: malicious page
  - User visits a malicious page in a second tab
  - Malicious page steals data from page or its cookies, modifies data, or impersonates user
Solution: Same-Origin Policy

• Browser needs to differentiate pages that are part of same application from unrelated pages
• What makes a page similar to another page?
  • Origin: the protocol, host, and port
    http://www.example.com/dir/page.html

• Different origins:
  https://www.example.com/dir/page.html
  http://www.example.com:80/dir/page.html
  http://en.example.com:80/dir/page.html

https://en.wikipedia.org/wiki/Same-origin_policy
Same-Origin Policy

• “Origin” refers to the *page that is executing it*, NOT where the data comes from
  • Example:
    • In one HTML file, I directly include 3 JS scripts, each loaded from a different server
    • -> All have same “origin”
  • Example:
    • One of those scripts makes an AJAX call to yet another server
    • -> AJAX call not allowed
• Scripts contained in a page may access data in a second web page (e.g., its DOM) if they come from the same origin
Cross Origin Requests

CORS: Cross Origin Resource Sharing

- Same-Origin might be safer, but not really usable:
  - How do we make AJAX calls to other servers?
- Solution: Cross Origin Resource Sharing (CORS)
- HTTP header:

  ```
  Access-Control-Allow-Origin: <server or wildcard>
  ```

- In Express:

  ```javascript
  res.header("Access-Control-Allow-Origin", "*");
  ```
Takeaways

• Think about all potential threat models
  • Which do you care about
  • Which do you not care about

• What user data are you retaining
  • Who are you sharing it with, and what might they do with it
Readings for next time

• Intro to microservices
  • https://www.martinfowler.com/articles/microservices.html