

SWE 432 -Web Application Development

Fall 2022



George Mason
University

Dr. Kevin Moran

Week 7:
Security





Administrivia

- HW Assignment 2 - Due today Before Class
- Midterm Exam - In class on Thursday
 - Review Video Posted today



Midterm Exam

- 3 Parts, In-class exam, closed book, 200 points total
 - **Part 1:** Multiple Choice Questions
 - **Part 2:** Short Answer
 - Either provide program output, or answer in a few short sentences
 - **Part 3:** Multi-Part Code Question (*implementing a simple microservice*)
- Covers material from weeks 1-7, from both lectures and readings
- You will have the **entire** class period to complete



Class Overview

- Today - *Security*: What is it, authentication, and important types of attacks

Quiz #5



Go to:

<https://bit.ly/3RDofBQ>

Reminder: Quiz can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

SWE 432 - Web Application Development



George Mason
University

Instructor:
Dr. Kevin Moran

Class will start in:

09:00

Quiz 5



Go to:

<https://bit.ly/3RDofBQ>



Quiz 5 Review

- Question 1: An error occurred in your server, and your server will be unable to satisfy the client's request. The server should indicate this to the client by:

General Answer: Sending the appropriate error code (e.g. a 50X code), and returning a meaningful message.



Quiz 5 Review

- Question 2: You're working to build a microservice that offers clients data about weather data. Your teammate proposes the following endpoint: `/requestCurrentWeatherFor.cfm` Describe at least two different ways in which this endpoint violates REST principles and what impact these issues might have.

General Answer: Using a verb within the URI and appending the web framework to the URI.



Quiz 5 Review

- Question 3: What are two ways in which a NoSQL database differs from a SQL database?

General Answer: schema-less, non relational, and weaker consistency



Quiz 5 Review

- Question 4: In Firebase, what's one difference between storing data with set and with add?

General Answer: Modifying an existing record (if it exists) with set vs. creating an entirely new entry with add. Creating a new unique id with add.

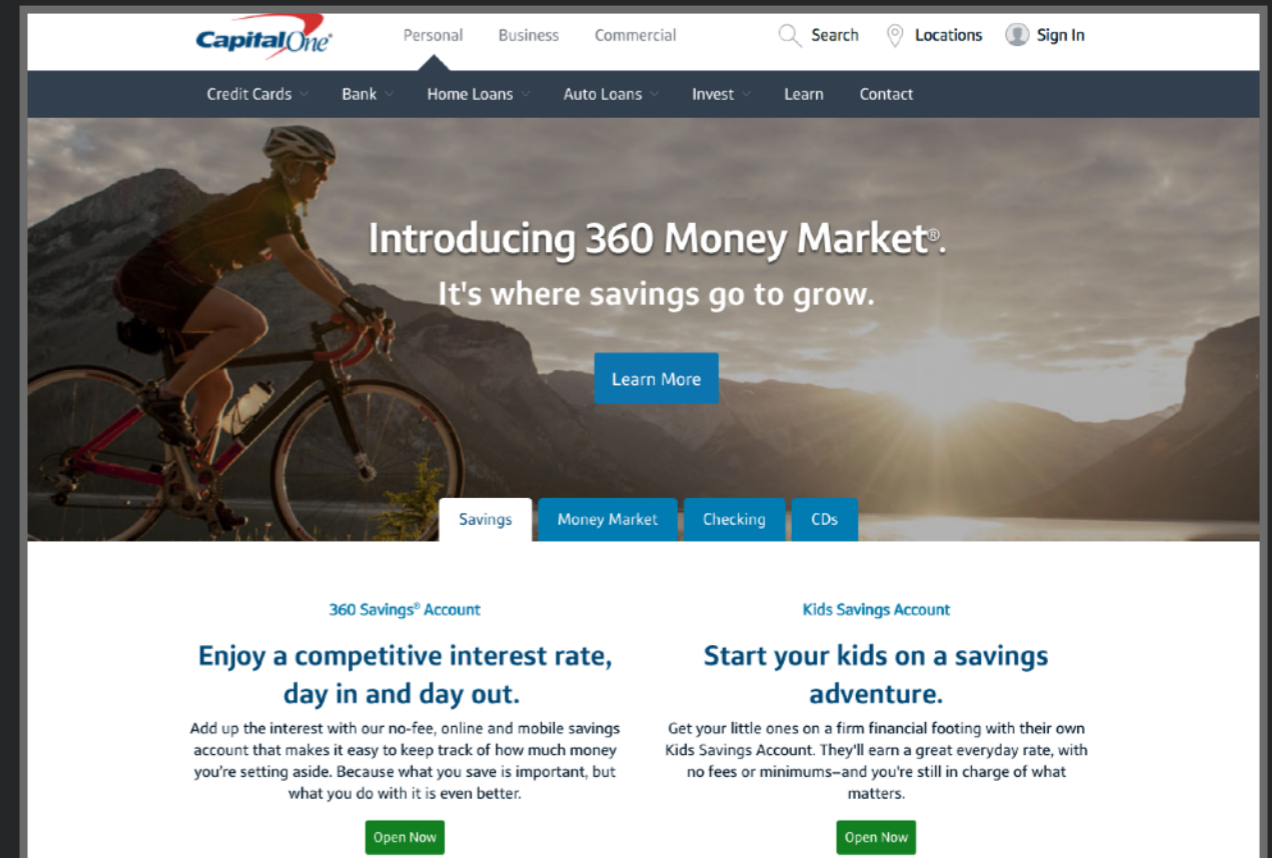
Web Security



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 *identity* 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?

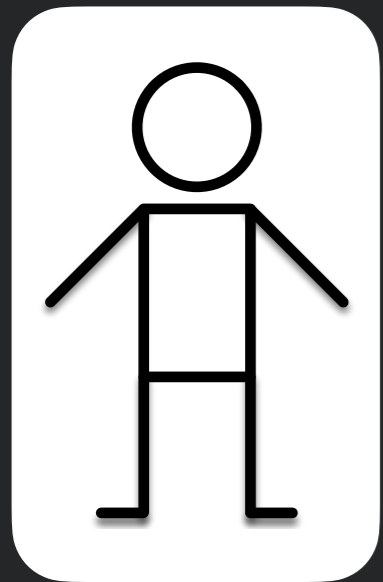


Threat Models

- 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

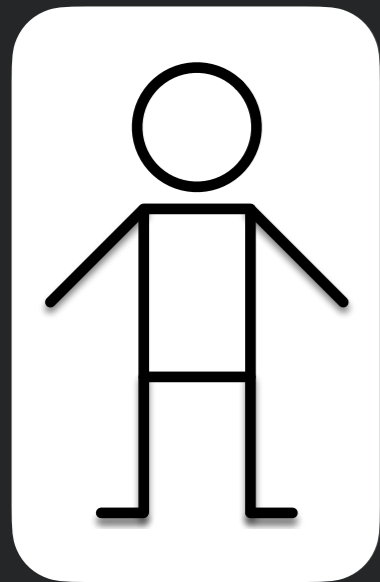


HTTP Response



server

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



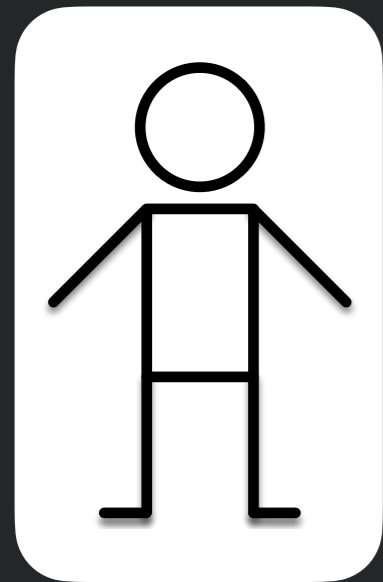
HTTP Response



server

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

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



HTTP Response

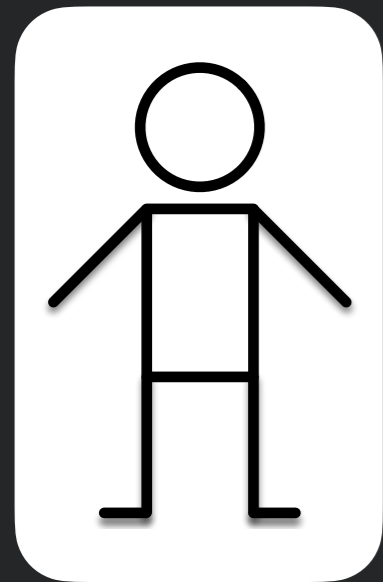


server

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

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

Web Threat Models: Big Picture



client page
(the “user”)

HTTP Request



HTTP Response

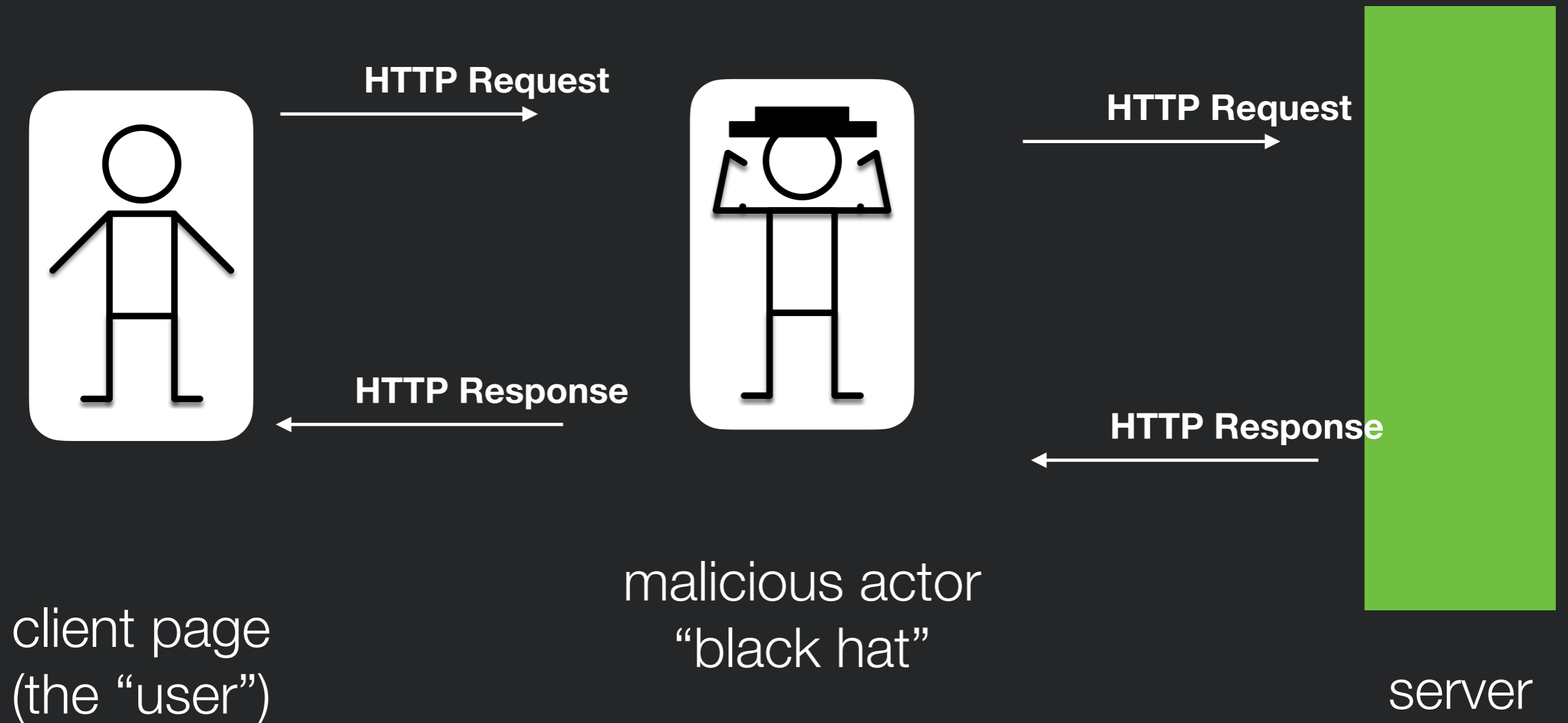


server

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

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

Web Threat Models: Big Picture

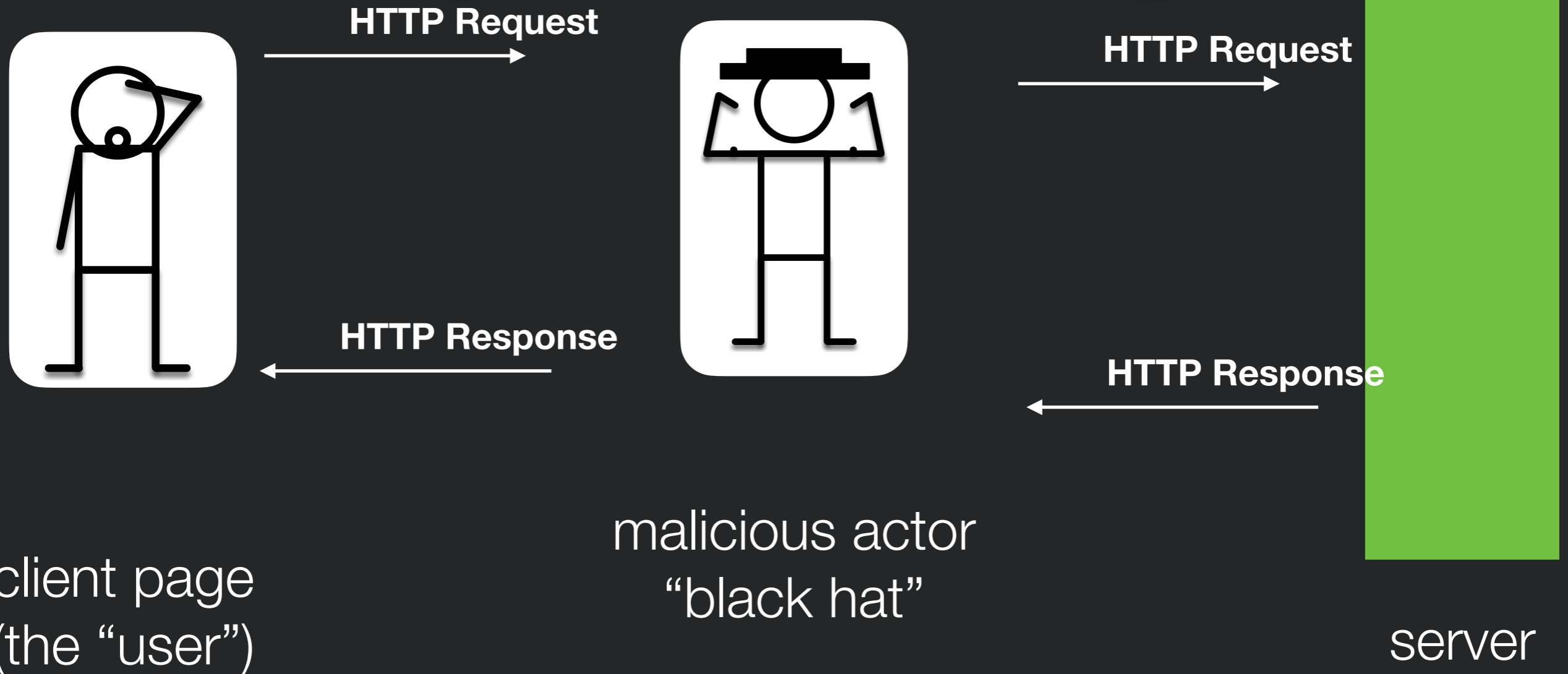


Do I trust that this response *really* came from the server?

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

Web Threat Mitigation Strategy

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 *identity* of the parties involved
- Threat: Impersonation. A person pretends to be someone they are not.

2. Authorization

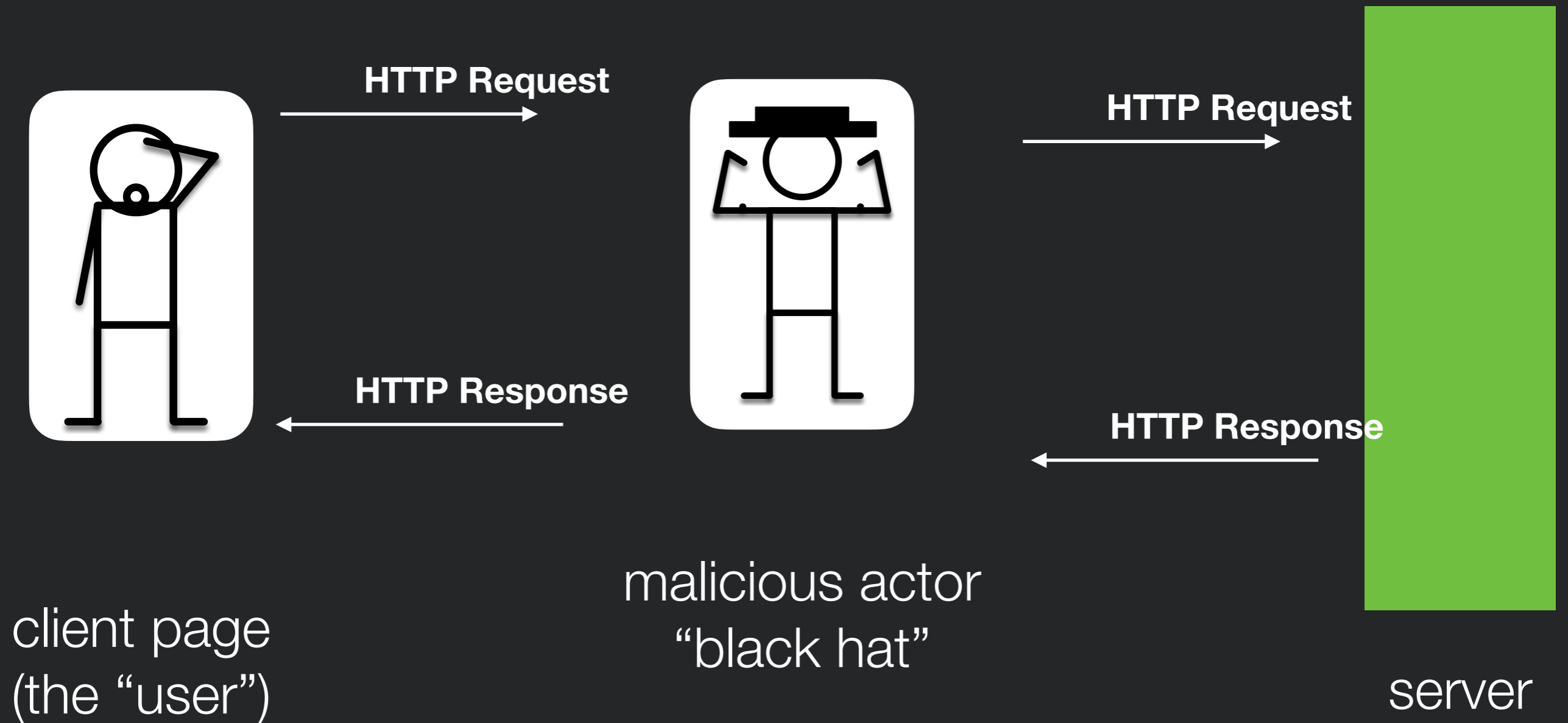
3. Confidentiality

- 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*.

Web Threat Models: Big Picture



What if malicious actor impersonates server?



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 Bootstrap through HTTP rather than HTTPS
 - Attacker intercepts request for Bootstrap script, replaces with malicious script that steals user data or executes malicious action



Authentication

- How can we know the identify 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, ...



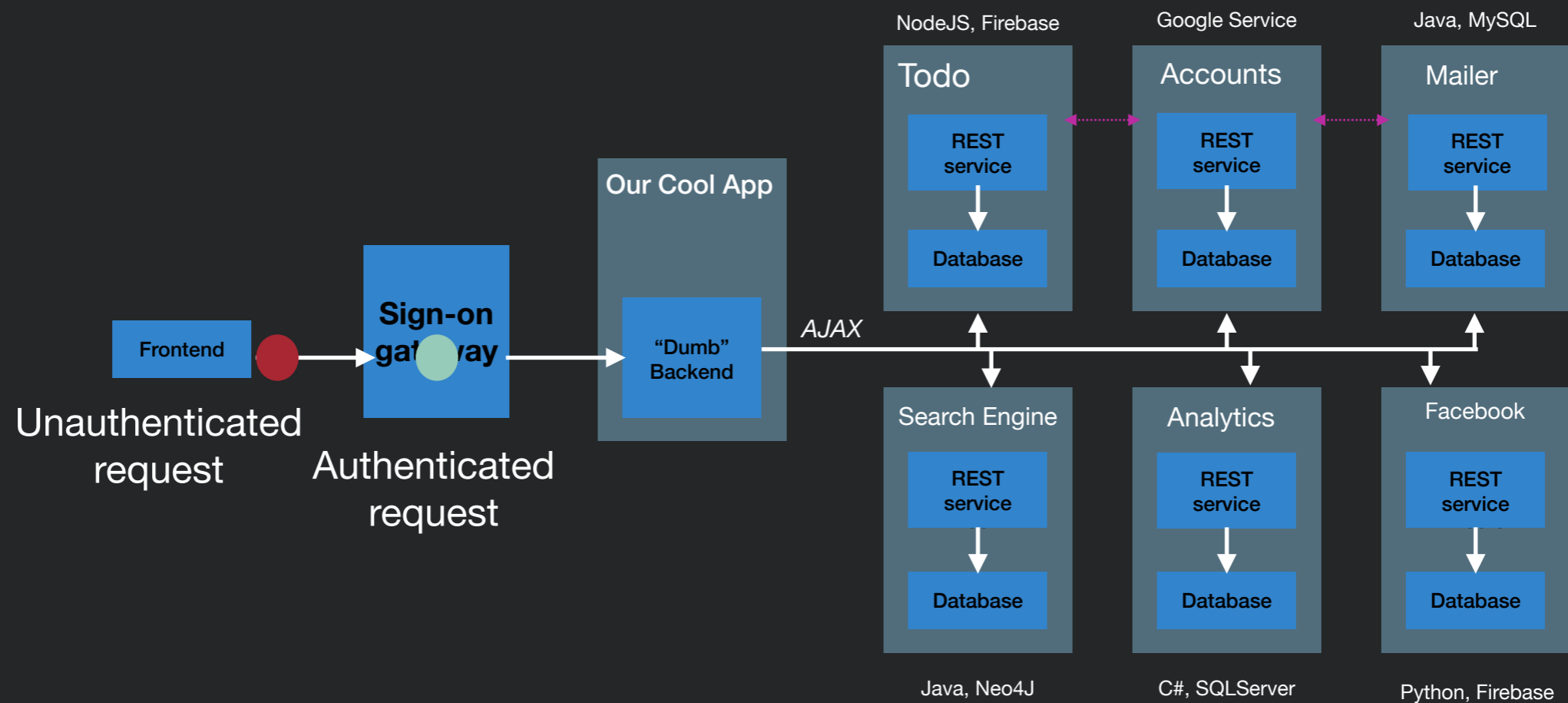
Authentication Provider

- 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



Sign-on Gateway

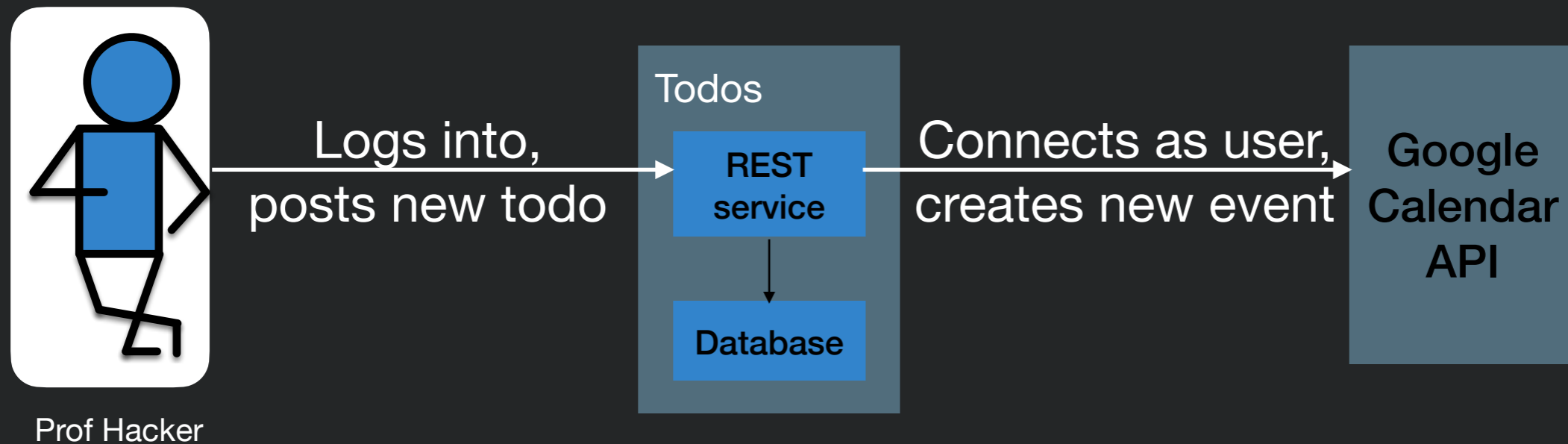
- Can place some magic “sign-on gateway” before our app - whether it’s got multiple services or just one



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



How does Todos tell Google that it's posting something for Prof Hacker?
Should Prof Hacker tell the Todos app her Google password?



We've Got Something for That...

The screenshot shows a Google account permissions dialog box. At the top left is the Google logo, and at the top right is the user's email address, "user@gmail.com", with a dropdown arrow. Below this is a header area with two circular icons: a blue person icon and a blue globe icon. Underneath the icons is the text "Example App would like to:" followed by a dropdown arrow. Below this are two permission items, each with a blue person icon on the left, a text description in the middle, and a blue information icon on the right. The first item is "View your basic profile info" and the second is "View your email address". At the bottom of the dialog is a paragraph of text: "By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other [Account Permissions](#) at any time." Below the text are two buttons: a grey "Deny" button and a blue "Allow" button.

Google user@gmail.com

Example App would like to:

- View your basic profile info
- View your email address

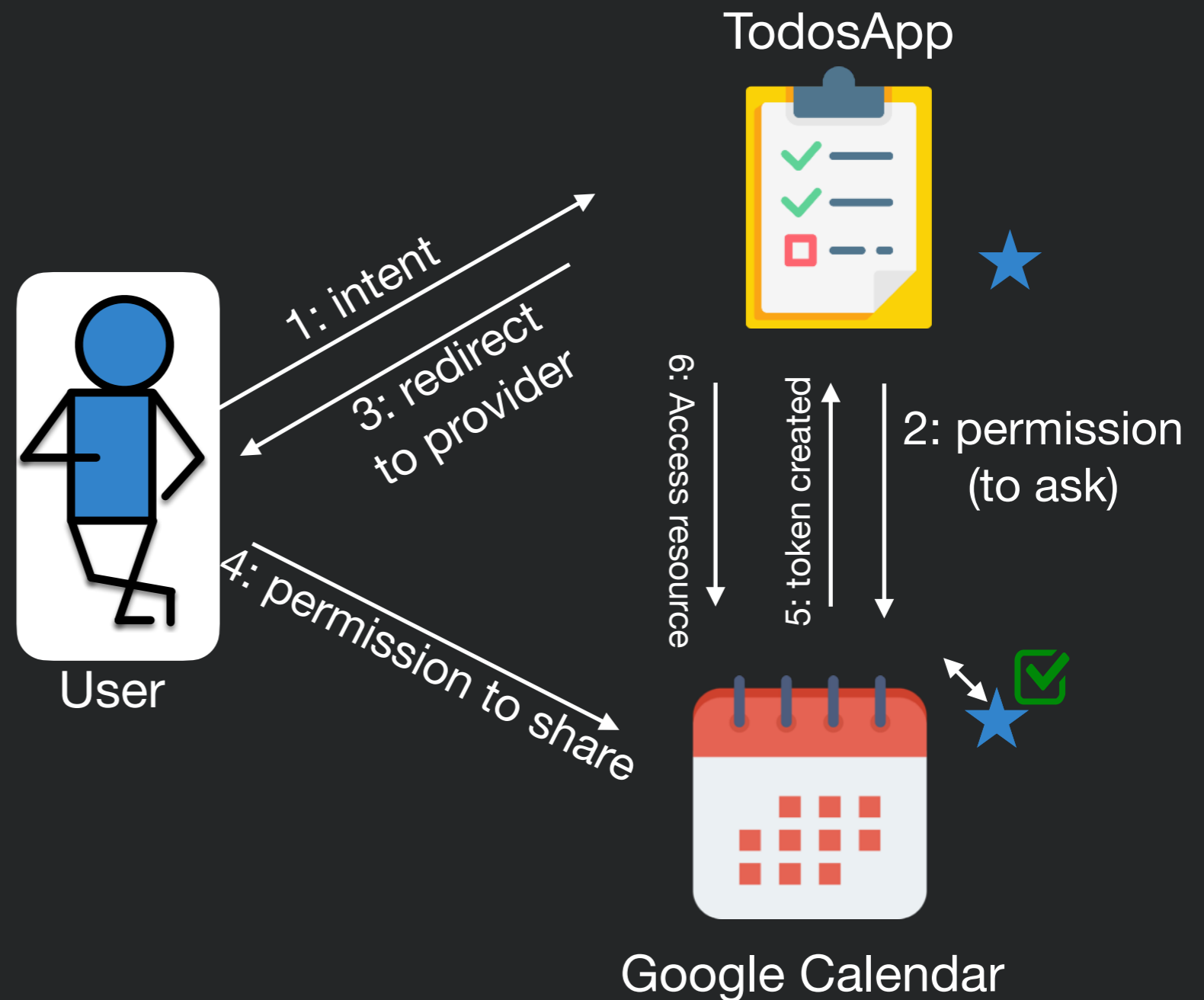
By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other [Account Permissions](#) at any time.

Deny Allow

- 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 Hacker,” “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





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:

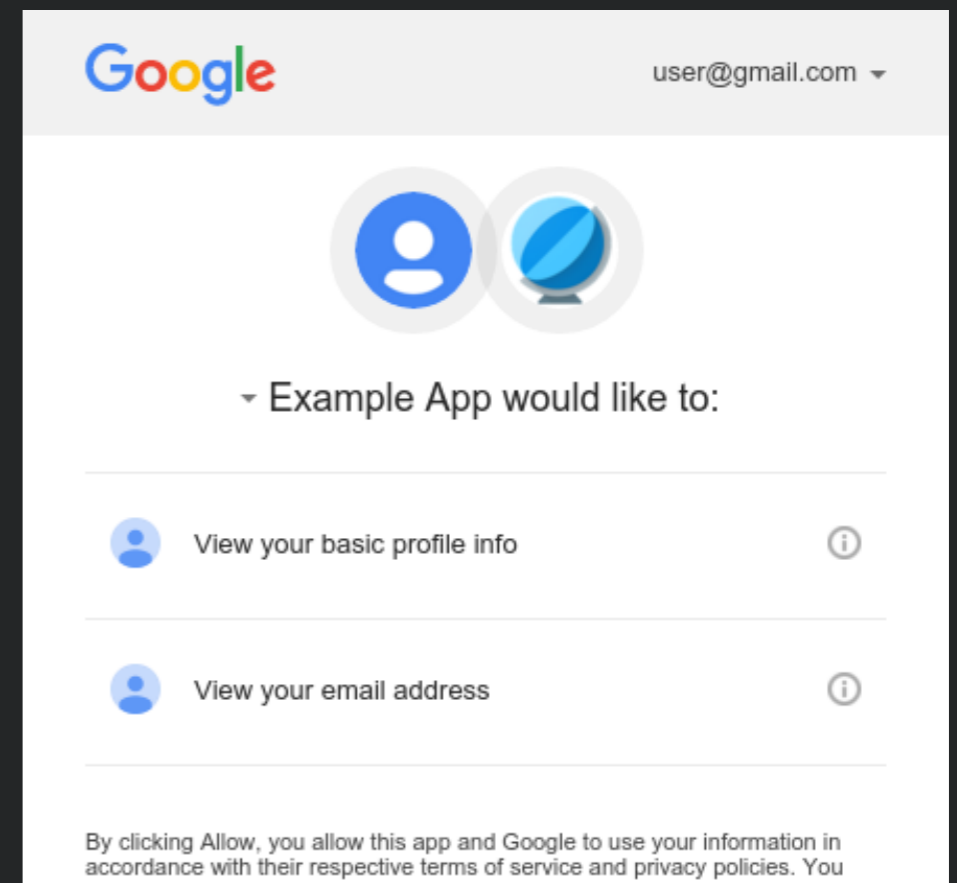
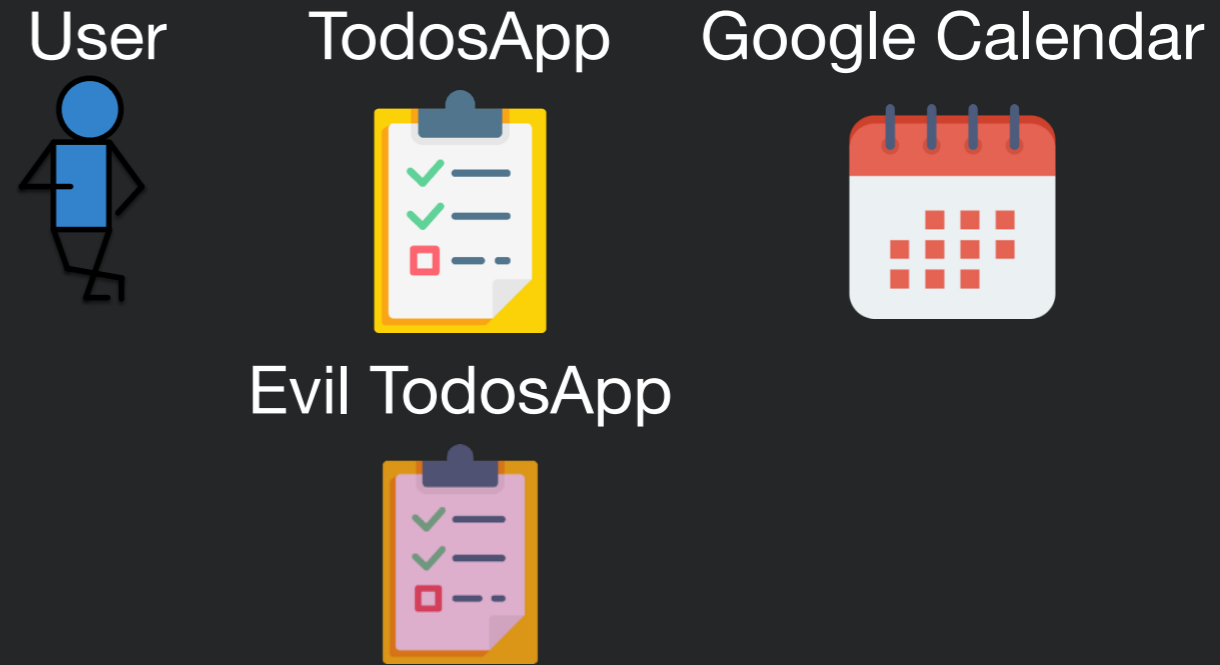
eyJhbGciOiJSUzI1NiIsImtpZCI6ImU3Yjg2NjFjMGUwM2Y3ZTk3NjQyNGUxZWFiMzI5OWI5NzRhNGVlNWUifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vYXV0aGRlbW8tNzJhNDIiLCJuYV11IjoiaSm9uYXRoYW4gQmVsbCIiInBpY3R1cmUiOiJodHRwczovL2xoNS5nb29nbGV1c2VyY29udGVudC5jb20vLW0tT29jRlU1R0x3L0FBQUFBQUFBQUFJL0FBQUFBQUFBQUUwL0JVV2t0NkRtTVJrL3Bob3RvLmpwZyIsImF1ZCI6ImF1dGhkZW1vLTcyYTQyIiwiaXV0aF90aW11IjoixNDc3NTI5MzcwLCJ1c2VyX2lkIjoiaSk1RcLFpdTlTUlRkeDY0YlR5Z0EzeHhEY3VIMiIsInN1YiI6IkpNUXJRaXU5U1JUZHg2NGJueWdBM3h4RGN1SDIiLCJpYXQiOiJE0Nzc1MzA4ODUsImV4cCI6MTQ3NzUzNDQ4NSwiZW1haWwiOiJqb25iZWxsd2l0aG5vaEBnbWFpbC5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmlyZWJhc2UiOi0nsiaWRlbnRpdGlscyI6eyJnb29nbGUuY29tIjpbIjEwOTA0MDM1MjU3NDMxMjE1NDIxNiJdLCJlbWFpbCI6WyJqb25iZWxsd2l0aG5vaEBnbWFpbC5jb20iXX0sInNpZ25faW5fcHJvdmlkZXIiOiJnb29nbGUuY29tIn19.rw1pPK377hDGmSaX31uKRphKt4i79ahjceepnA8A2MppBQnPJlCqmgSapxs-Pwmp-1Jk382VooRwc8TfL6E1UQUl65yi2aYYzSx3mWMTwtPTHtkMN4E-GNprp7hX-pqD3PncBh1bq1dThPNyjHLp3CUlPP0_QwaAeSuG5xALhzfYkvLSINTy4FguD9vLHydpVHWscBNCDHAC0qSeV5MzUs6ZYMnBIitFhbkak6z50ClvxGTGMhvI8m11hIHdWgNGnDQNNosifzlwMqDHiF5t3K0L-mxtcnq33TvMAc43JElxnyB4g7qV2hJI0y4MLtLxphAfCeQZA3sxGf7vDXBQ

Decoded:

```
{
  "iss": "https://securetoken.google.com/authdemo-72a42",
  "name": "Alyssa P Hacker",
  "picture": "https://lh5.googleusercontent.com/-m-0ocFU5GLw/AAAAAAAAAAI/AAAAAAAAAH0/BUWkN6DmMRk/photo.jpg",
  "aud": "authdemo-72a42",
  "auth_time": 1477529371,
  "user_id": "JMQRQiu9SRTdx64bTygA3xxDcuH2",
  "sub": "JMQRQiu9SRTdx64bTygA3xxDcuH2",
  "iat": 1477530885,
  "exp": 1477534485,
  "email": "alyssaphacker@gmail.com",
  "email_verified": true,
  "firebase": {
    "identities": {
      "google.com": ["109040352574312154216"],
      "email": ["alyssaphacker@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

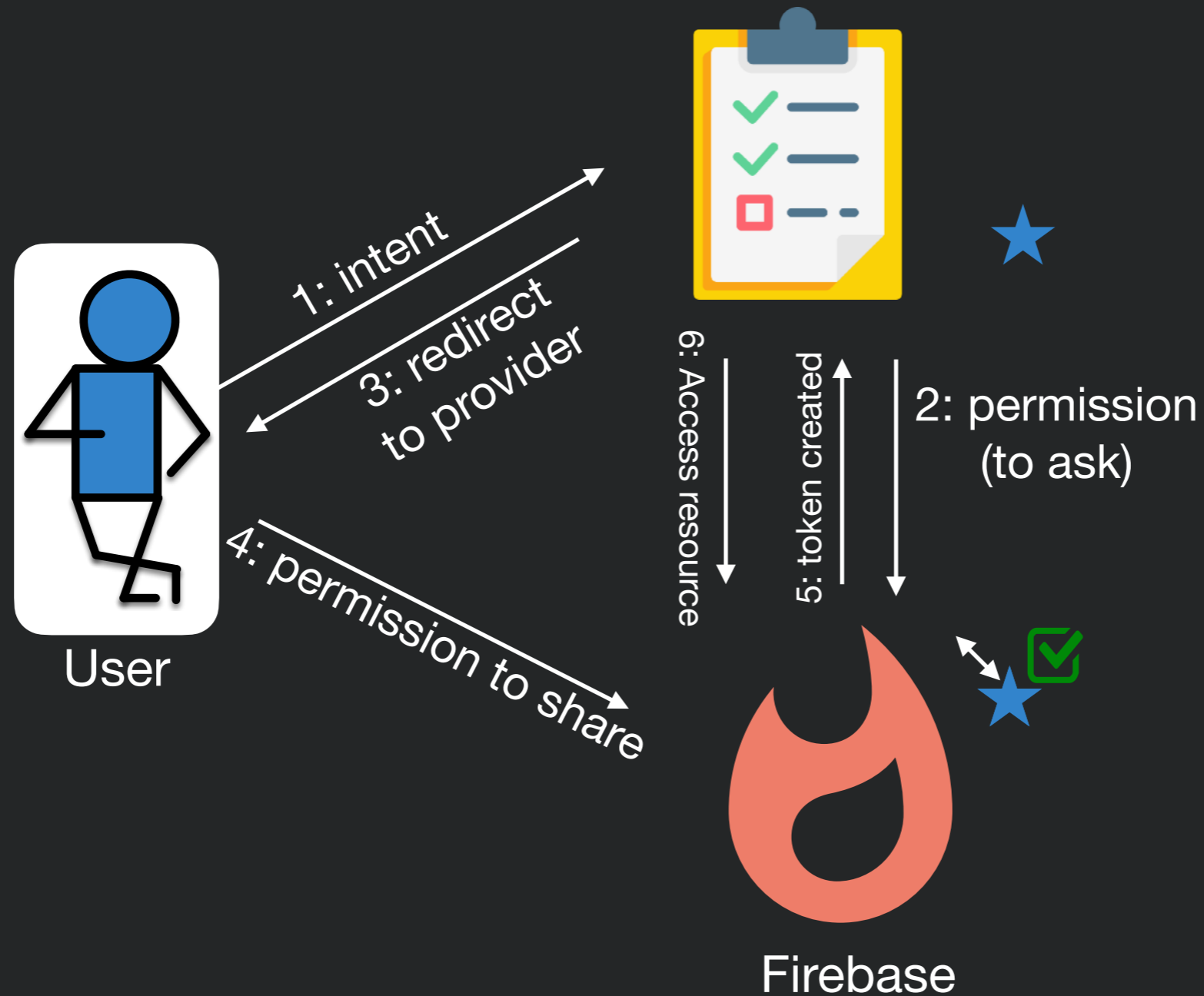




Authentication as a Service

- 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!

Using an Authentication Service





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>
- Firebase handles browser local storage to track that the user is logged in across pages (woo)



Top 3 Web Vulnerabilities

- OWASP collected data on vulnerabilities
 - Surveyed 7 firms specializing in web app security
 - Collected 500,000 vulnerabilities across hundreds of apps and thousands of firms
 - Prioritized by prevalence as well as exploitability, detectability, impact

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project



#3 - XSS: Cross Site Scripting

- User input that contains a *client-side* script that does not belong
 - A todo item:

```
/><script>alert("LASAGNA FOR PRESIDENT");</script>
```

- Works when user input is used to render DOM elements without being escaped properly
- User input saved to server may be served to other users
 - Enables malicious user to execute code on other's users browser
 - e.g., click 'Buy' button to buy a stock, send password data to third party, ...



#2 - Broken Authentication and Session Management

- Building authentication is hard
 - Logout, password management, timeouts, secrete questions, account updates, ...
- Vulnerability may exist if
 - User authentication credentials aren't protected when stored using hashing or encryption.
 - Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs).
 - Session IDs are exposed in the URL (e.g., URL rewriting).
 - Session IDs don't timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren't properly invalidated during logout.
 - Session IDs aren't rotated after successful login.
 - Passwords, session IDs, and other credentials are sent over unencrypted connections.



#1 - Injection

- User input that contains *server-side* code that does not belong
- Usually comes up in context of SQL (which we aren't using)
 - e.g.,
 - `String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";`
- Might come up in JS in context of eval
 - `eval(request.getParameter("code"));`
 - Obvious injection attack - don't do this!



Validating User Input

- Escape Strings that originate from user
- Type of escaping depends on where data will be used
 - HTML - HTML entity encoding
 - URL - URL Escape
 - JSON - Javascript Escape
- Done automatically by some frameworks such as React
- More details: [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)



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
- Attack: malicious page
 - User visits a malicious page in a second tab
 - Malicious page steals data from page or its data, 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>

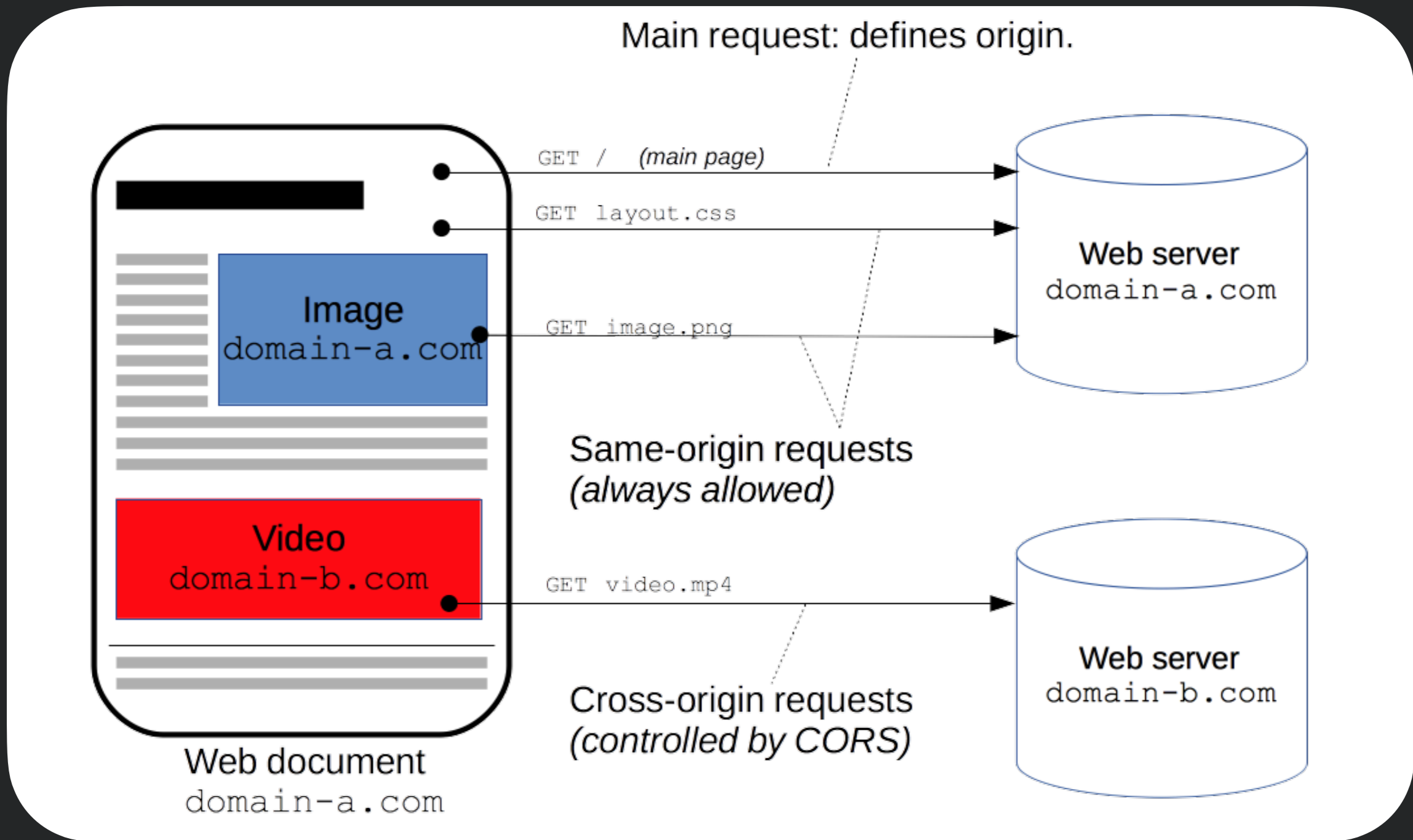
https://en.wikipedia.org/wiki/Same-origin_policy



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:

```
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