Web Security & Rest APIs

And some Project 2 tips

Discussion 4

Announcements

- Project 2 DUE Friday, February 19th at 8pm EST
- Get started if you have not already!!
- Midterm is in a month (Thursday, March 11th 3-5 pm)

Network Security

- Two parties communicate over a network
  - Assume powerful adversary
    - Can read (eavesdrop on) all data transmitted
    - Can modify or delete any data
    - Can inject new data
- **Confidentiality:** Adversary should not understand message
- **Sender authenticity:** Message is really from the purported sender
- **Message integrity:** Message not modified between send and receive
- **Freshness:** Message was sent “recently”
- **Anonymity:** Attacker should not know that we are communicating

Attacks on HTTP

- Imagine that I'm communicating with a server, using HTTP. What are some attacks that take advantage of this (i.e. general idea)?
- How can these attacks be done?

Man in the middle: done by replay

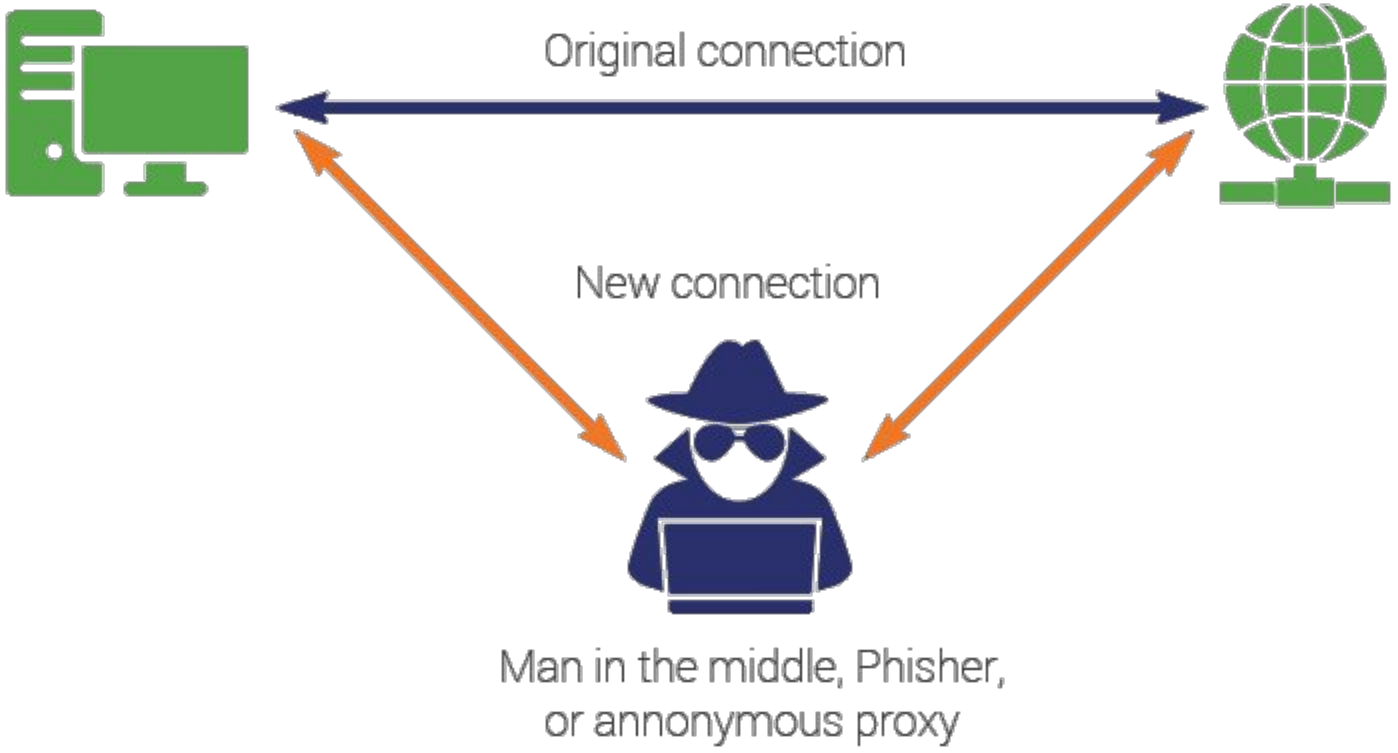
Masquerading: address spoofing

Eavesdropping: data sniffing

Attacks on HTTP

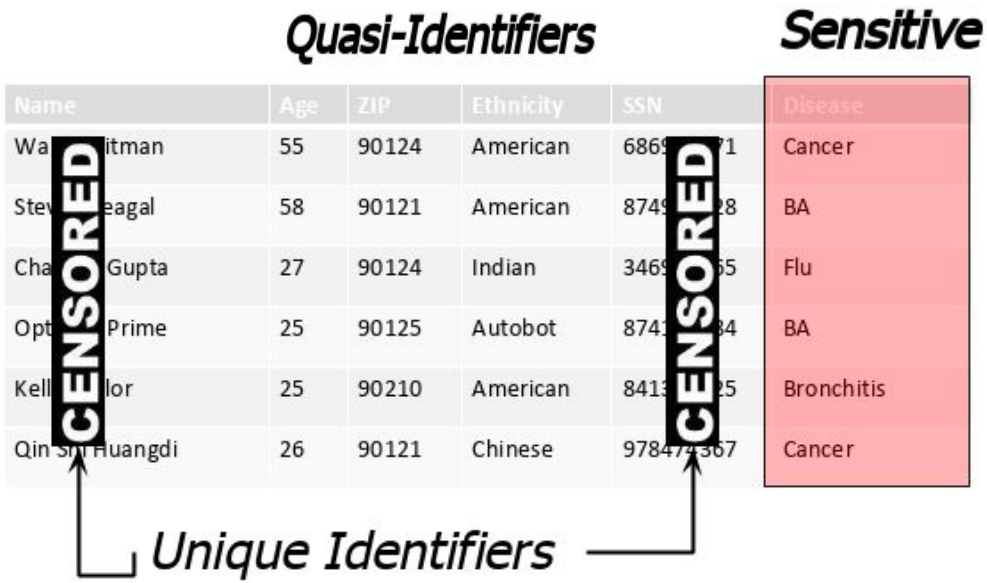
Imagine that I'm communicating with a server, using `HTTP`. What are some attacks that take advantage of this (i.e. general idea)?

- **Man-in-the-middle**: Stand in between two communicating parties and modify the conversation
  - Can be done by **Replay**: Record communication to use later
- **Masquerading**: Pretend to be someone else
  - Can be done by **Address spoofing**: Write a fake IP into packets
- **Eavesdropping**: Listen in on communication
  - Can be done by **Data sniffing**: On a local network


Original connection
New connection
Man in the middle, Phisher,
or annonymous proxy

Database Privacy

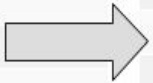
- Any release data improves an adversary's ability to identify any subset
- **Quasi-identifiers** are pieces of information that are not unique identifiers, but are well correlated with an entity that they can be combined with other quasi-identifiers to create a unique identifier.



K-Anonymity

- Manipulating data to ensure tuple cannot be distinguished from k-1 other tuples
- Uses ranges to generalize numeric data

Age	ZIP	Ethnicity	Disease
55	90124	American	Cancer
58	90121	American	BA
27	90124	Indian	Flu
25	90125	Autobot	BA
25	90210	American	Bronchitis
26	90121	Chinese	Cancer



2-anonymized view:

Age	ZIP	Ethnicity	Disease
55-58	90121-90124	*	Cancer
55-58	90121-90124	*	BA
26-27	90121-90124	*	Flu
25	90125-90210	*	BA
25	90125-90210	*	Bronchitis
26-27	90121-90124	*	Cancer

- Problems:
  - Loss of information due to generalization
  - Vulnerability #1: **Homogeneity**: everyone in a bucket has the same sensitive characteristic
  - Vulnerability #2: **Background information attacks:** what if we know that certain ethnicities are highly prone to certain diseases?

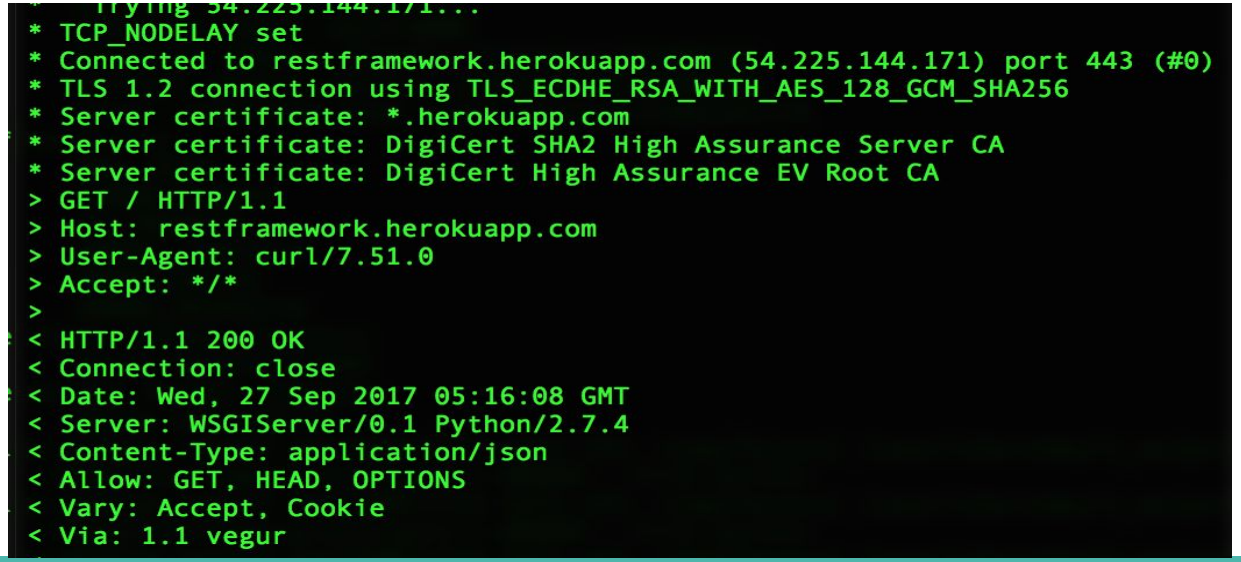
Installations!

Install some packages

- Httpie: useful and simple command line interface for reusing sessions and making HTTP requests (will be used in Project 3)
  - `brew install httpie`
  - `sudo apt-get install httpie`
  - `pip install --upgrade httpie`
- jsonlint
  - `brew install jsonlint`
  - `sudo apt-get install jsonlint`

GET Requests

- So far we've seen GET requests return HTML content
  - `curl --verbose http://cse.eecs.umich.edu/ > index.html`
- But, now we'll see how GET requests can return JSON data
  - `curl --verbose https://restframework.herokuapp.com/`



REST API

- **RE**presentational **S**tate **T**ransfer
- REST is **not** a standard nor a language
- REST is a collection of principles, usually uses HTTP and JSON
- Provides ability of computer systems to exchange and make use of information on the Internet
- Defines set of functions, where developers can perform a **request** and receive a **response** via the HTTP protocol
- This response is usually JSON (commonly used to send data from server to web client)
    - Think project 1 config.json

REST - Putting it Together

- Abstraction
  - Client and Server could both be replaced independently of each other
- Stateless
  - No client data is stored on the server between requests
  - Instead, session state is stored by the client
- Cacheable
  - Clients can cache responses, such as images
  - Responses must indicate that they are cacheable

REST - Request

- Client sends request via body contents, query-string parameters, headers, and URI
- Resource-based
  - Request for a resource that is identified by a Uniform Resource Identifier (URI)
  - URI is like a pointer
    - **`https://restframework.herokuapp.com/users/2/`**
    - https://restframework.herokuapp.com/users/2/ ← URI

- REST Verbs
  - GET - return datum
  - PUT - replace the entire datum
  - POST - create new datum
  - DELETE - delete datum
  - PATCH - update part of a datum

REST - Response

- Server delivers state to client via body content, response codes, headers
- **JSON** (JavaScript Object Notation) Response is not exclusive to JSON
- Structures:
    - Object with name/value pairs

      `{"logname": "awdeorio", "numPhotos": 4}`
    - Array: an ordered list of values

      `["awdeorio", "jflinn"]`
- The value can be of types int, string, bool, Object, null, etc.

- jsonlint - JSON validator
    - Checks for correctly formatted JSON - no trailing commas or mismatched brackets
    - https://jsonlint.com/

Let’s Try It!

- Go to `httpbin.org` if you want some more practice on HTTPs methods, Request Responses, etc
- Scroll down to the “Anything” part to try things out
- Make a curl request in terminal to see return JSON
    - `curl httpbin.org/anything`

- If you want more information, feel free to look into this: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f

HTTP Status Codes

- **200 OK**
- **201 Created**
  - Successful creation after POST
- **204 No Content**
  - Successful DELETE
- **302 URL Redirection**
  - **T**he requested resource has been temporarily moved to a different URI
- **400 Bad Request**
  - General Error
  - Domain validation errors, missing data, etc.

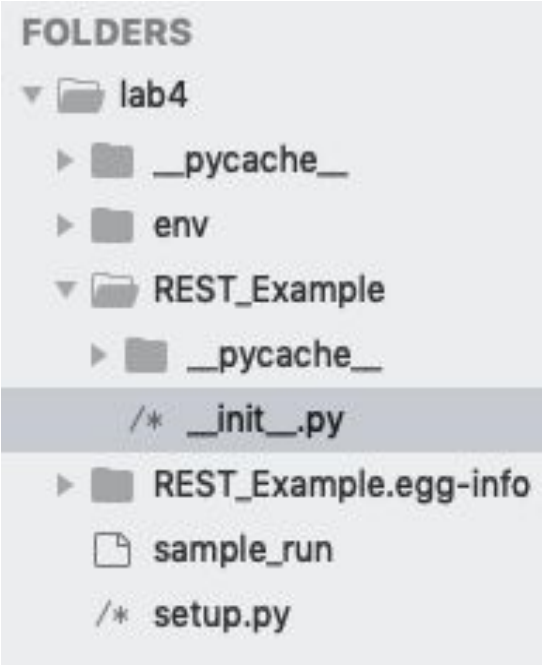
HTTP Status Codes

- **403 Forbidden**
  - Server understood, but user is not authorized
- **404 Not Found**
  - Resource could not be found
- **409 Conflict**
  - E.g., duplicate entries and deleting root objects when cascade-delete is not supported
- **500 Internal Server Error**
  - General catch-all for server-side exceptions

Making a REST API in Flask

```
$ mkdir lab4demo

$ cd lab4demo

$ wget -qO- http://bit.ly/485lab4 | tar -xzvf -

$ python3 -m venv env

$ source env/bin/activate

$ (env) pip install -e .
```

- Your file directory should look similar to the screenshot on the right



Making a REST API in Flask

- In terminal, make sure you're in **lab4demo**, run **`./sample_run`**
  - If you get "Permission Denied", run: **`chmod +x sample_run`**
  - If it still doesn't work, run these commands in terminal:

```
export FLASK_DEBUG=True
export FLASK_APP=REST_Example
flask run --host 0.0.0.0 --port 8000
```

- Application should now be running on **localhost:8000**
- Now navigate to **localhost:8000/api/v1/** then to **localhost:8000/api/v1/users/**

Project 2 DB tip

- We have model.py (in the spec)

model.py

```
"""Insta485 model (database) API."""
import sqlite3
import flask
import insta485


def dict_factory(cursor, row):
    """Convert database row objects to a dictionary."""
    output = {}
    for idx, col in enumerate(cursor.description):
        output[col[0]] = row[idx]
    return output


def get_db():
    """Open a new database connection."""
    if not hasattr(flask.g, 'sqlite_db'):
        flask.g.sqlite_db = sqlite3.connect(
            insta485.app.config['DATABASE_FILENAME'])
        flask.g.sqlite_db.row_factory = dict_factory

        # Foreign keys have to be enabled per-connection.
        # This is an sqlite3 backwards compatibility thing.
        flask.g.sqlite_db.execute("PRAGMA foreign_keys = ON")

    return flask.g.sqlite_db
```

- Dict_factory converts SQL output to a python dictionary
- Dict_factory is called in get_db

On top of python file

`From insta485.model import get_db`

To write a SQL statement and get back data from the database:

```
variable = get_db().execute(
''' SELECT * FROM <table> '''
```

) (can add `.fetchall()` or `.fetchone()` here)

Project 2 Passwords

- Everyone's password is set to `password`
- However, we **cannot** store plain text passwords in our database
  - In data.sql, you can use the hash given in the dump
- When adding a new password (such as in `/account/create/`)
  - There is code given to you to hash the password using `sha512`
- You should check for password validation (such as `login`)
  - Make sure the password is what it should be
    - Keep in mind passwords aren't stored in plain text
- *If the spec does not explicitly state what to do for certain errors, make a reasonable decision on what should happen in that case*

Project 2 Page State

- If the page is **public**, and there' a session - view the page
- If the page is **sensitive** (such as edits, deletes, etc.), check for a session
  - If there's no session, redirect to login
- From the Spec - **Access Control**
  - The server should only accept `POST` requests from the logged in user
  - A malicious party should not be able to delete someone else's post by issuing a random `POST` request. To reject a request with a permissions error, use `abort(403)`