Security Issues in Web Programming (Part 1)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - Some important web pgmming security risks
  - Some mitigation techniques

- Thanks to **Joseph Eichenhofer** (COS 333 TA during the Spring 2020 semester) for major contributions to this lecture

Web Programming Security Risks

[see slide]

From Wikipedia:

> “The **Open Web Application Security Project** (**OWASP**) is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.[4][5]The Open Web Application Security Project (OWASP) provides free and open resources. It is led by a non-profit called The OWASP Foundation.”

From OWASP website:

> “The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.”
>
> “Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.”

Web Pgmming Security Risks

OWASP 2021 List

OWASP Order	Security Risk
1	Broken access control
2	Cryptographic failures
3	Injection
4	Insecure design
5	Security misconfiguration
6	Vulnerable and outdated components
7	Identification and authentication failures
8	Software and data integrity failures
9	Security logging and monitoring failures
10	Server-side request forgery (SSRF)

Web Pgmming Security Risks

[see slide]

For reference

Web Pgmming Security Risks

OWASP 2017 List

OWASP Order	Security Risk	Coverage Order
1	**Injection**	6
2	**Broken authentication**	10
3	**Sensitive data exposure**	8
4	XML external entities (XXE)	1
5	Broken access control	5
6	Security misconfiguration	3
7	**Cross-site scripting (XSS)**	9
8	Insecure deserialization	7
9	Using components with known vulnerabilities	4
10	Insufficient logging and monitoring	2

5

Web Pgmming Security Risks

[see slide]

For reference; I used to organize this sequence of lectures

Let's consider those security risks one at a time

- Approximately in order by increasing relevance to COS 333
  - Little to say for risks at beginning of list
  - More to say for risks at end of list

For each risk:

- Describe example risks & action items (that you might think about for your project)

Agenda

- **XML external entities (XXE)**
- Insufficient logging & monitoring
- Security misconfiguration
- Using components with known vulnerabilities
- Broken access control
- Injection
- Insecure deserialization
- Sensitive data exposure

XML External Entities (XXE)

XML External Entities (XXE)

“Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.”

-- https://owasp.org/www-project-top-ten/

Mitigation is beyond our scope

Agenda

- XML external entities (XXE)
- **Insufficient logging & monitoring**
- Security misconfiguration
- Using components with known vulnerabilities
- Broken access control
- Injection
- Insecure deserialization
- Sensitive data exposure

Insufficient Logging & Monitoring

Insufficient Logging & Monitoring

“Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.”

-- https://owasp.org/www-project-top-ten/

Insufficient Logging & Monitoring

- **Example risk:**
  - Attacker, having attacked your website by exploiting some vulnerability, might do so again
- **Example risk:**
  - Attacker, having attacked your website by exploiting some vulnerability, might do so to some other website

Insufficient Logging & Monitoring

- **Action item:**
  - Detect attacks quickly
- **Action item:**
  - Design your website to write log messages
    - (Non-Heroku) Write messages to a file
    - (Heroku) Write messages to stdout or stderr
- **Action item:**
  - Check log messages frequently
    - (Heroku) `heroku logs` command

Insufficient Logging & Monitoring

- **Message attributes**
  - When
    - Log date and time; event date and time, …
  - Where
    - App identifier, app address, code location. …
  - Who
    - IP address, mobile tel num, …
  - What
    - Event type, event severity, event description, …

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Logging_Cheat_Sheet.md#event-attributes

Insufficient Logging & Monitoring

- **Which events to log:**
  - Application errors
  - Input and output validation failures
  - Authentication successes and failures
  - Authorization failures
  - …

https://devcenter.heroku.com/articles/writing-best-practices-for-application-logs

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- **Security misconfiguration**
- Using components with known vulnerabilities
- Broken access control
- Injection
- Insecure deserialization
- Sensitive data exposure

Security Misconfiguration

Security Misconfiguration

“Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. ”

-- https://owasp.org/www-project-top-ten/

Security Misconfiguration

- **Example risk:**
  - Attacker might exploit sensitive information contained in error messages that your website sends to browser
- **Action item:**
  - Design your website to generate **descriptive private** error messages (in server log), and **vague public** error messages (in browser)

16

Security Misconfiguration

As applied to COS 333:

In reg applications…
Website detects “no such table: classes” error
Website should write descriptive “no such table: classes” message to server log
Website should send vague “Server error. Contact system administrator.” message to browser

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- Security misconfiguration
- **Using components with known vulnerabilities**
- Broken access control
- Injection
- Insecure deserialization
- Sensitive data exposure

Using Components with Known Vulnerabilities

Using Components with Known Vulnerabilities

"Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts."

-- https://owasp.org/www-project-top-ten/

Using Components with Known Vulnerabilities

- **Example risk:**
  - Attacker might take advantage of your website if it uses components with known vulnerabilities
- **Action item:**
  - Stay informed!!!

Using Components with Known Vulnerabilities

- **Example risk:**
  - Attacker might exploit vulnerability in old version of operating system or other software product that your website uses
- **Action item:**
  - Install software updates/patches ASAP!

20

Security Misconfiguration

[see slide]

Every time a software vendor releases an update of a product to correct a security flaw, the vendor thereby (indirectly) informs attackers of the security flaw

Either explicitly or implicitly (via diffs)

It's essential to install newest versions of software products ASAP

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- Security misconfiguration
- Using components with known vulnerabilities
- **Broken access control**
- Injection
- Insecure deserialization
- Sensitive data exposure

Broken Access Control

Broken Access Control

“Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users’ accounts, view sensitive files, modify other users’ data, change access rights, etc.”

-- https://owasp.org/www-project-top-ten/

Broken Access Control

- **Example risk:**
  - Attacker might exploit incorrect file/directory permissions to access website files
- **Action item:**
  - Make sure permissions are correct!
  - Heroku: irrelevant
  - Unix systems: use `chmod` command to set file/directory permissions
  - Compose/run scripts that set permissions

Broken Access Control

[see slide]

Personal experience maintaining website to accompany my book

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- Security misconfiguration
- Using components with known vulnerabilities
- Broken access control
- **Injection**
- Insecure deserialization
- Sensitive data exposure

Injection

Injection

“Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.”

-- https://owasp.org/www-project-top-ten/

Injection

- **Example risk:**
  - Attacker might attack your website via SQL injection
- **Action item:**
  - Use SQL prepared statements
    - As described in *Database Programming* lecture
    - As used in your assignment solutions
  - or…
  - Use an ORM (e.g., SQLAlchemy)

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- Security misconfiguration
- Using components with known vulnerabilities
- Broken access control
- Injection
- **Insecure deserialization**
- Sensitive data exposure

Insecure Deserialization

Insecure Deserialization

“Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.”

-- https://owasp.org/www-project-top-ten/

Insecure Deserialization

- **Example risk (client/server apps)**:
  - Attacker might attack your client/server app by changing the picked Python objects that it communicates

Insecure Deserialization

- Python `pickle` module

```
from pickle import dump
from pickle import load
…
flo = sock.makefile(mode='wb')
dump(some_object, flo)
flo.flush()
…
flo = sock.makefile(mode = 'rb')
some_object = load(flo)
…
```

Insecure Deserialization

- `dump(some_object, flo)`
  - Generates **statements** that, when executed, create `some_object`
  - Writes those **statements** to `flo`
- `some_object = load(flo)`
  - Reads **statements** from `flo`
  - Executes those **statements** to generate `some_object`

Insecure Deserialization

- There is a known security risk associated with using Python pickled objects
- Example
  - See **unsuspectingserver.py**
  - See **maliciousclient.py**
    - Thanks to Joseph Eichenhofer

Insecure Deserialization

- **Example risk (cont.):**

```
$ python unsuspectingserver.py 55555
Opened server socket
Bound server socket to port
Listening
```

```
$ python maliciousclient.py localhost 55555
$
```

```
$ python unsuspectingserver.py 55555
Opened server socket
Bound server socket to port
Listening
Accepted connection for ('127.0.0.1', 54408)
Opened socket for ('127.0.0.1', 54408)
$ cat badfile.txt
hello there
$
```

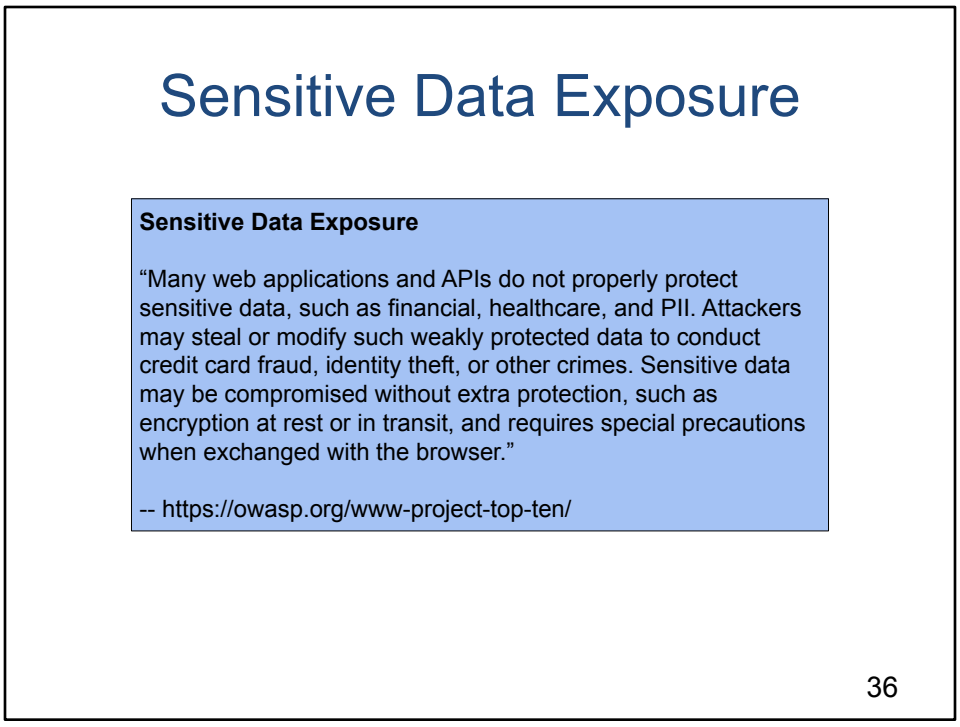
Insecure Deserialization

- **Action item:**
  - In client/server apps, when security matters:
    - Don't use pickled objects for comm; instead:
      - Object → char string
      - Object → JSON/XML doc
    - Encrypt the comm

- Note:
  - Java *serialization* is secure

Agenda

- XML external entities (XXE)
- Insufficient logging & monitoring
- Security misconfiguration
- Using components with known vulnerabilities
- Broken access control
- Injection
- Insecure deserialization
- **Sensitive data exposure**



Sensitive Data Exposure

[see slide]

“PII” is personally identifiable information

Sensitive Data Exposure: Storage

- **Example risk (data at rest):**
  - Attacker might attempt to access sensitive data (e.g., passwords) stored in your website
  - How to **store** sensitive data securely?

Sensitive Data Exposure: Storage

- **Insight:**
  - No need to store passwords!
- **Action item:** Store ***hashed*** passwords instead of passwords
  - `hashed_password = hash(password)`
  - Store `hashed_password` instead of `password`

38

Sensitive Data Exposure: Storage

Insight:

No need to store the passwords!

It's sufficient to know whether given password is correct!

Mitigation: Store hashed passwords

hashed_password = hash(password)

Given user and password, app can determine if password is correct for that user

App can use hash() function to compute hashed_password, and see if it agrees with hashed_password previously stored for that user in DB

hash() function might return same hashed_password for multiple passwords, but (with a good hash function) that would be very improbable

If attacker discovers hashed_password, attacker can't infer user's password

Works because hash functions are one-way functions

Sensitive Data Exposure: Storage

- **Action item (more specific):** Store ***md5 hashed*** passwords instead of passwords
  - `hashed_password = md5(password)`
  - No! See https://en.wikipedia.org/wiki/MD5
- **Action item (more specific):** Store ***sha256 hashed*** passwords instead of passwords
  - `hashed_password = sha256(password)`
  - Yes! See https://en.wikipedia.org/wiki/SHA-2

Sensitive Data Exposure: Storage

- **Example risk (data at rest):**
  - Attacker might perform brute-force attack on `hashed_password`
  - Having discovered `hashed_password`, attacker could search (malevolent?) DB of common `hashed_password`s (i.e., sha256 hash codes for common passwords), and thereby (maybe) determine `password`

40

Sensitive Data Exposure: Storage

https://crackstation.net/ is an example of a website that accepts a hash code and writes the string that has that hashcode

Sensitive Data Exposure: Storage

- **Action item:** ***salt*** the passwords before hashing
  - "Salt" the password with some extra application-specific text, and then hash it
- Example:
  - `salted_hashed_password = hash('xxx' + password + 'yyy')`
  - `salted_hashed_password` will not be found in DB of common `hashed_password`s

Sensitive Data Exposure: Storage

- Using both hashing & salting
  - App can determine to high degree of probability if given `password` is correct
  - Attacker discovers `salted_hashed_password` => attacker doesn't know `password`, and is unlikely to be able to infer it
- Note:
  - Same techniques can be applied to other stored data

Sensitive Data Exposure: Storage

sha256 hashing & salting in Python

```
$ python -m pip install werkzeug
$ python
>>> from werkzeug.security import generate_password_hash
>>> from werkzeug.security import check_password_hash
>>> h = generate_password_hash('somepassword')
>>> h
'pbkdf2:sha256:260000$Mv6BjV9UcZDQdZTu$db46d71d2730119feb5e
0977f0d5729c2664da197713878b05263e87fab9c3a2'
>>> check_password_hash(h, 'somepassword')
True
>>> check_password_hash(h, 'someotherpassword')
False
>>> quit()
$
```

Sensitive Data Exposure: Comm

- **Example risk (data in transit):**
  - Attacker might attempt to access sensitive data during comm between website and browser
- **Action item:**
  - Use ***Hypertext Transfer Protocol Secure (HTTPS)*** instead of HTTP

Sensitive Data Exposure: Comm

- **Technical** advantages of using HTTPS
  - Prohibits ***message eavesdropping attacks***
    - When Alice sends message to Bob, nobody except Alice and Bob can understand it
  - Prohibits ***forgery attacks***
    - When Alice sends message to Bob, Bob can be sure that it's from Alice

Sensitive Data Exposure: Comm

- **Other** advantages of using HTTPS
  - Increases user confidence/trust in website
  - Increases Google search rank of website
    - Google's search engine optimization (SEO) favors HTTPS websites over HTTP websites

Sensitive Data Exposure: Comm

- All popular **browsers** support HTTPS
  - Chrome, Firefox, …
- All popular **HTTP servers** support HTTPS
  - IIS, Apache, Nginx, …
  - Sys admins must configure HTTP server to support HTTPS (create public/private keys, certificates)

Sensitive Data Exposure: Comm

- **Question**:
  - How to use HTTPS in your web app?
- **Answer:**
  - Through examples…

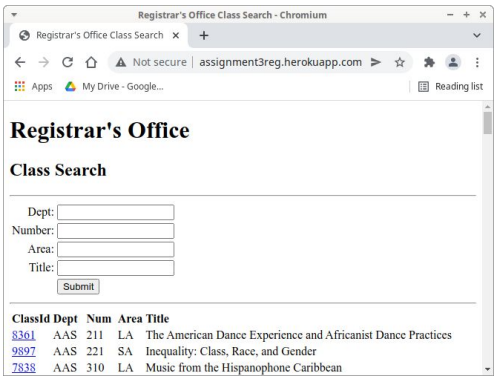
Sensitive Data Exposure: Comm

- **Example 1: Asgt 3 ref app**
  - No requirement that browser & server use HTTPS
  - Code in the normal way
  - Then…

Sensitive Data Exposure: Comm

- **Example 1: Asgt 3 ref app**
  - Heroku + HTTP

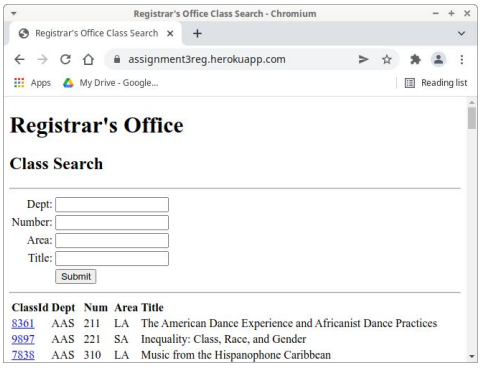
After entering **http**://assignment3reg.herokuapp.com



Sensitive Data Exposure: Comm

- **Example 1: Asgt 3 ref app**
  - Heroku + HTTPS

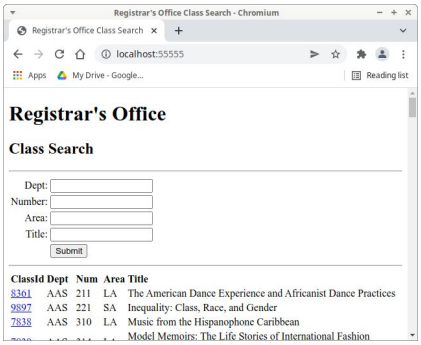
After entering **https**://assignment3reg.herokuapp.com



Sensitive Data Exposure: Comm

- **Example 1: Asgt 3 ref app**
  - Flask test server + HTTP

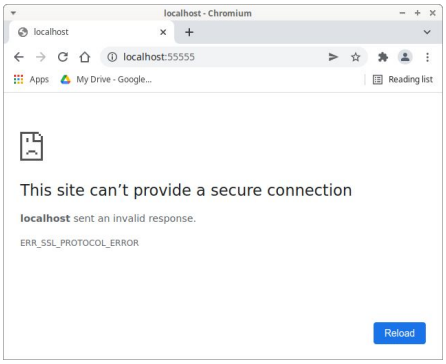
After entering
http://localhost:55555



Sensitive Data Exposure: Comm

- **Example 1: Asgt 3 ref app**
  - Flask test server + HTTPS

After entering
https://localhost:55555



Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Not required by asgt, but…
  - Suppose you want browser & server to use HTTPS
  - Code with these changes…

For more info:
https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https

Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**

  In reg.py:

```
…
app = Flask(__name__, template_folder='.')
…
@app.before_request
def before_request():
    if not request.is_secure:
        url = request.url.replace('http://', 'https://', 1)
        return redirect(url, code=301)
```

Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**

In runserver.py

```
…
app.run(host='0.0.0.0', port=port,
   debug=True, ssl_context='adhoc')
```

Commands Flask to create a ***self-signed certificate***

56

Sensitive Data Exposure: Comm

[see slide]

Alternative to self-signed certificate:

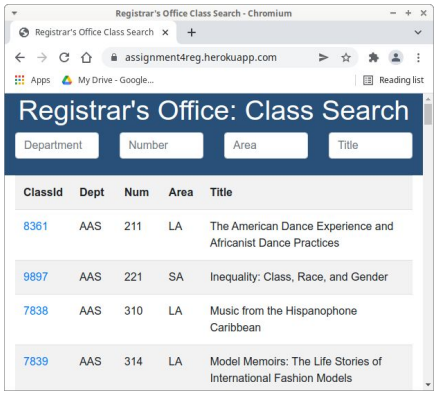
Get a real one from some certificate agency

Unnecessary for development

Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Heroku + HTTPS

After entering **https**://assignment4reg.herokuapp.com

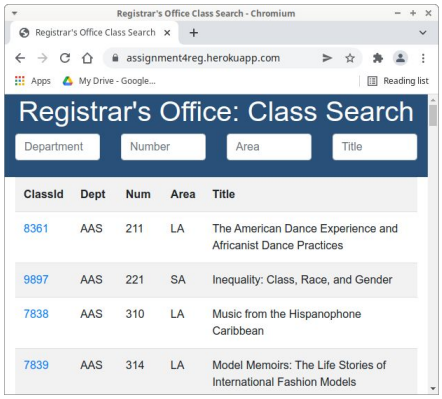


Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Heroku + HTTP

After entering **http**://assignment4reg.herokuapp.com…

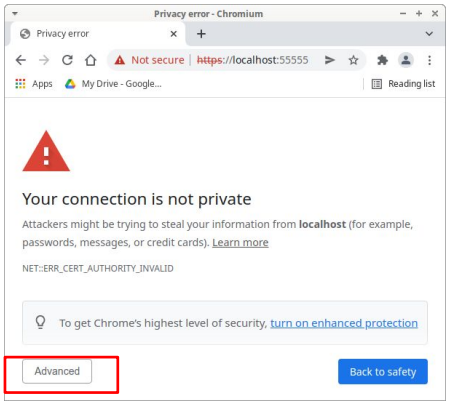
Server redirects to **https**://assignment4reg.herokuapp.com



Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Flask test server + HTTPS

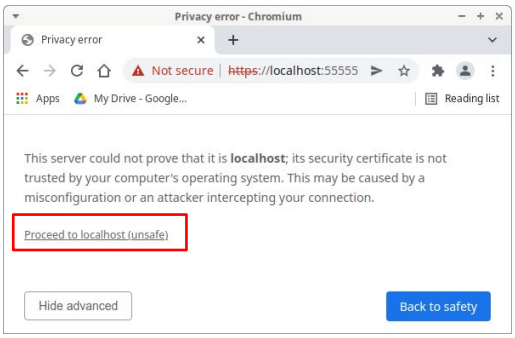
After entering
https://localhost:55555



Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Flask test server + HTTPS (cont.)

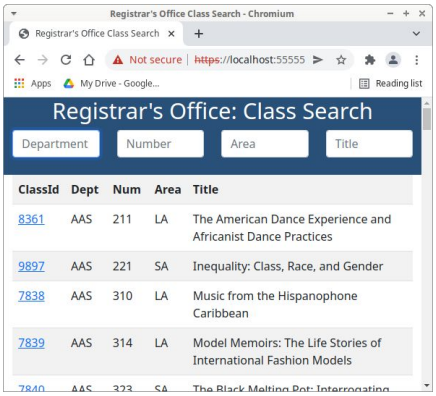
After entering
https://localhost:55555



Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Flask test server + HTTPS (cont.)

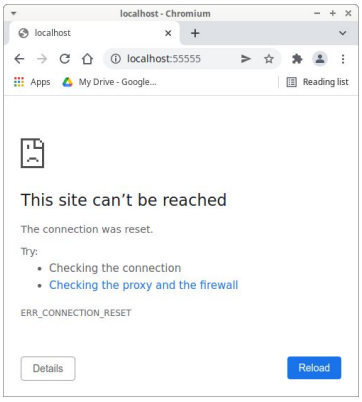
After entering
https://localhost:55555



Sensitive Data Exposure: Comm

- **Example 2: Asgt 4 ref app**
  - Flask test server + HTTP

After entering
http://localhost:55555



Sensitive Data Exposure: Comm

- **Question:**
  - How does HTTPS work?
- **Answer:**
  - See Wikipedia *HTTPS* article
  - See *COS 432: Information Security*
    - Commentary: A very important course!

Summary

- We have covered:
  - XML external entities (XXE)
  - Insufficient logging & monitoring
  - Security misconfiguration
  - Using components with known vulnerabilities
  - Broken access control
  - Injection
  - Insecure deserialization
  - Sensitive data exposure