Web Security (Capture the Flag)

Discussion 2/3/21
Slides from CSE127, sp20
This lecture will be recorded and made available to registered students on Canvas.
Logistics:

- PA4 Login credentials are on Gradescope
- Early Turn-in is on **Tuesday** 2/9
- Final Deadline is on **Friday** 2/12
• **Topics Explored**
  - Web Security
    - HTTP/HTTPS
      - GET, POST, etc.
    - JavaScript injection
  - Base64 encoding
  - SQL injection
Communicating on the World Wide Web

Language of choice: HTTP/HTTPS

- Hypertext Transfer Protocol (OSI Application Layer)
- HTTPS = HTTP + TLS/SSL (encryption)
- Common HTTP Methods
  - GET used to request a resource (should not alter state of server)
  - POST used to send data to the server.
  - PUT request data be placed under specified URI
  - DELETE delete resource at URI

python requests

- **GET**

```python
import requests

# Optional parameters for GET request
params = {}
url = "https://www.google.com"
response = requests.get(url, params)

# Generally
# 200 -> all good
# 404 -> resource not found
# 500 -> error on server side
response.status_code

# This will have the response body
response.content
```

- **POST**

```python
import requests

# This will be the data the sender requests the server to store/update
data = {"username": "riley",
        "password": "youllneverguessit!",
        "favorite color": "blue"}

url = "https://www.google.com"
response = requests.post(url, data)
```
Disclaimer

flagrant simplifications follow
Web Page Anatomy (10,000 ft view)

- **HTML**
  - Contains main content of the page and organizes the layout

- **CSS**
  - Used for styling components of web page in consistent theme

- **JavaScript**
  - Adds dynamic features to page
  - Code run client side by web browser
Visiting a Web Page (10,000 ft view)

1) Browser requests a web page
2) Server sends back HTML, CSS, JavaScript, and other resources associated with page
3) Browser renders page according to received resources
4) Embedded JavaScript run by browser
   - responds to user clicks, mouse-movement, etc.
   - loads larger resources like pictures so that entire page isn’t held up
   - advertisement loading
Client-Side Debugging: Elements
Client-Side Debugging: Console

 Topics Explored

- Web Security
- HTTP/HTTPS
  - GET, POST, etc.
- SQL injection
- JavaScript injection
Client-Side Debugging: Sources
say we have a web page as below ... the {} is replaced by user input

malicious user input string:
`<script>document.write("\tHello, from inside the script!")</script>`
Client-Side Debugging: Network

- Web Security
- HTTP/HTTP(S)
- GET, POST, etc.
- SQL injection
- JavaScript injection
- Base64 encoding
Client-Side Debugging: Application

Topics Explored

- Web Security
- HTTP/HTTPS
- GET, POST, etc.
- SQL injection
- JavaScript injection
- Base64 encoding
Base64 encoding

What?
- Binary to text encoding

Why?
- Carry binary data reliably over channels which only support text content
  - e.g. embed images in HTML or CSS

How?
- Every 6 bits of data corresponds to a Base64 digit
  - \{A-Z, a-z, 0-9, +, /\} Most implementations agree on first 62, but last two vary
  - Padded with ‘=’

Input String: w0w
Binary: 01110111 01101111 01110111
B64 Grouping: 011101 110110 111101 110111
B64 encoded: d29
python base64

Encoding

```python
import base64
# Encoding
body = "wow"
body_ascii = [ord(x) for x in body]
body_bytes = bytes(body_ascii)
body_b64_encoded = base64.b64encode(body_bytes)
body_b64_encoded_string = "".join([chr(x) for x in body_b64_encoded])
```

Decoding

```python
base64.b64decode()
(again bytes → bytes)
```
PA4

• **General Tips**
  
  – Each level will give a clue
  – Brute force approaches will not work!
    • Server has rate limits so you will be blocked for the remainder of the day
  – Some levels can be solved with short python scripts (<10 lines)
Reminders:
- PA4 Login credentials are on Gradescope