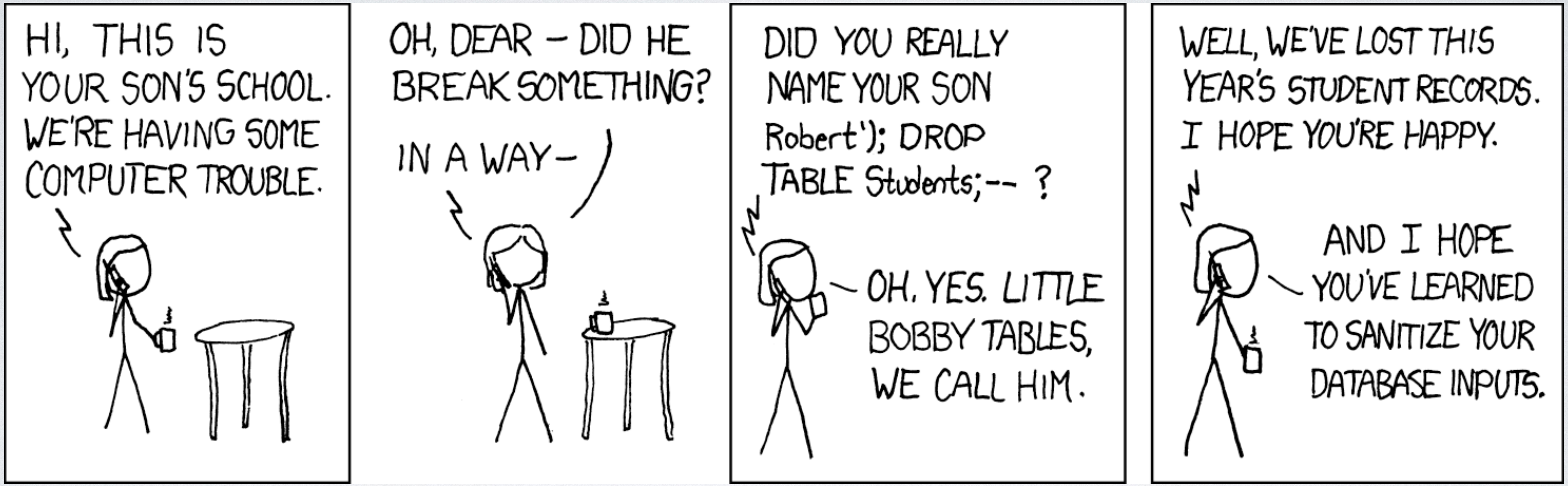

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

Security and DevSecOps

Integrating Security into the Software Development Process

Why Is Security Different?

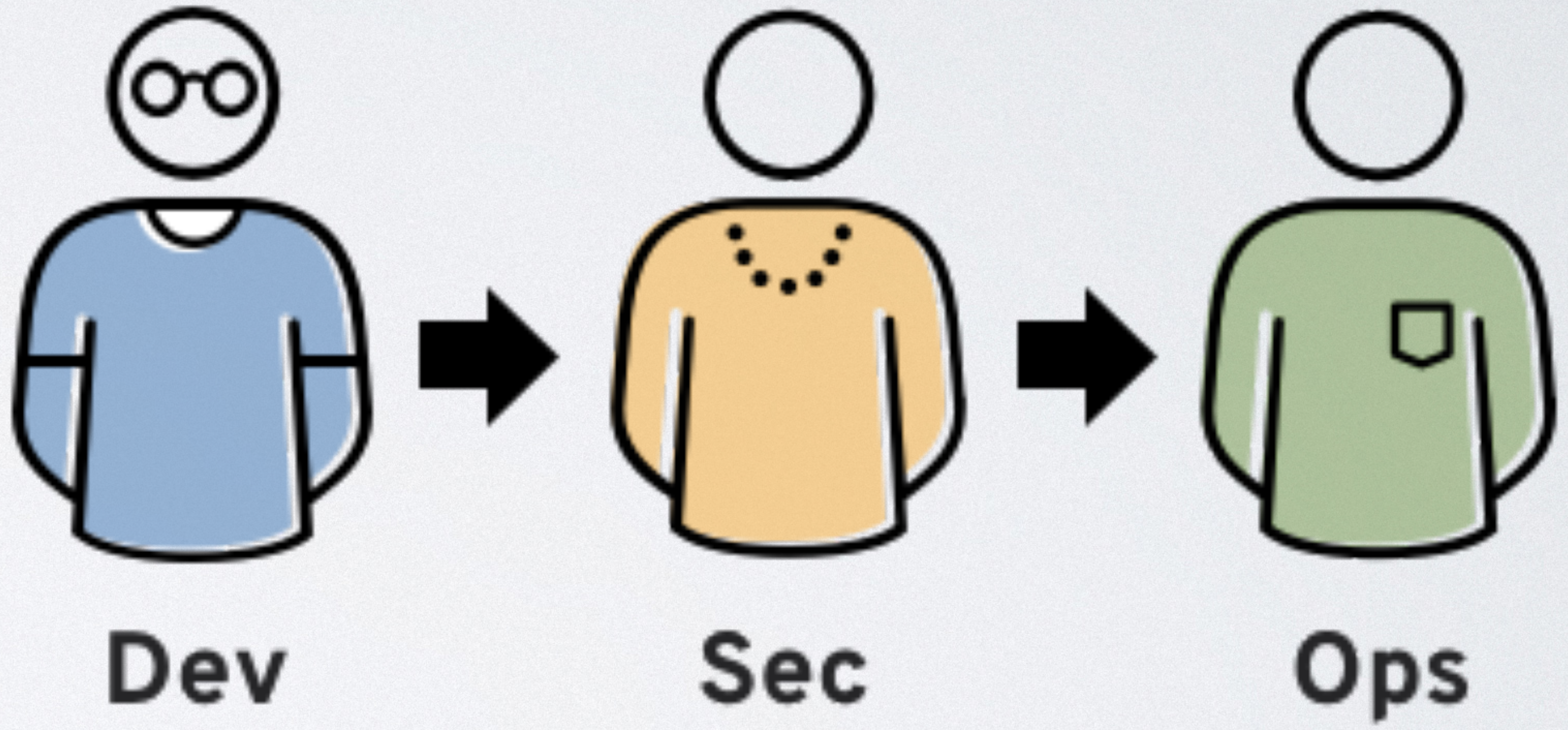
- It only takes one weakness to take this castle down
- Attackers are *trying* to find weaknesses
  - Need an *adversarial mindset* to defend the castle

Credit: https://upload.wikimedia.org/wikipedia/commons/8/84/Sand_castle,_Cannon_Beach.jpg

Have You Been the Victim of a Security Vulnerability? (Have You Been Hacked?)

The Old Way

- First, write the code
- Then, have the security people do their thing
- Then, let the operations people host it
- But doing security too late is bad…



https://www.redhat.com/en/topics/devops/what-is-devsecops

Security Has Architectural Implications

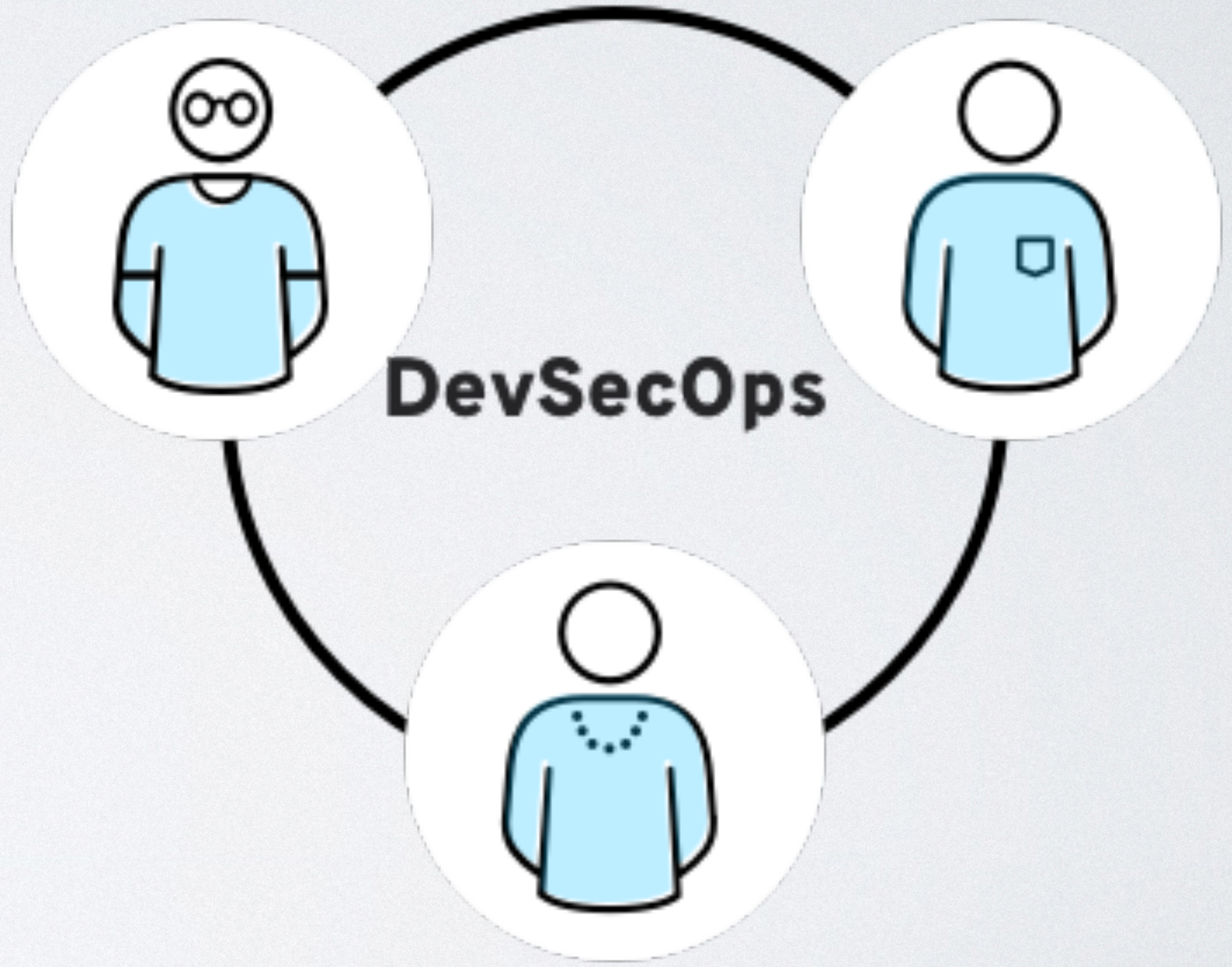
- Where is access control?
- Where is authentication?
- How are credentials passed?
- What are the attack vectors?

More Design Implications

- Tooling: you aren't going to use C/C++, are you?
- Testing processes
  - Penetration tests?
- How will you mitigate social engineering attacks?

DevSecOps

- Integrate security into the development process
- The rest of today: how to include security concerns



https://www.redhat.com/en/topics/devops/what-is-devsecops

Kinds of Security Challenges

Challenge	Approach
Undefined behavior	Don't use unsafe languages (when possible)
Incorrect security-related code	Review, test, control changes
Higher-level design mistakes	Architectural review, penetration testing
Users (e.g., social engineering attacks)	HCI techniques; training; compromise procedures

Key Terminology

- Integrity: protect against invalid or untrusted code or data being treated as trusted or valid
  - Example failure: student sets their own grade in the gradebook to "A"
- Confidentiality: protect against untrusted extraction of sensitive data
  - Example failure: student A sees student B's grade ($A \neq B$)

Microsoft DevSecOps Advice

- Train
- Define security requirements
- Define metrics and compliance reporting
- Use Software Composition Analysis and Governance
- Perform threat modeling
- Use tools and automation
- Keep credentials safe
- Use continuous learning and monitoring

https://www.microsoft.com/en-us/securityengineering/devsecops#Metrics

Train

- Glad you're here.

Define Security Requirements

- Legal and industry requirements
- Internal standards and coding practices
- Review of previous incidents, and known threats.
- Traditional requirements analysis, with security focus

Define Metrics and Compliance Reporting

- How will you know whether you've succeeded?
- Does one breach mean you've failed?
  - Better to focus on progress than success/failure

Threat Modeling

- Goal: enumerate all possible threats
- STRIDE model helps you remember possible threats:
  - **S**poofing identity
  - **T**ampering with data
  - **R**epudiation
  - **I**nformation disclosure
  - **D**enial of service
  - **E**levation of privilege

If I were writing a final exam, I might ask you to explain one of the letters in STRIDE!

Exercise

- In groups: enumerate possible threats for your project
  - In a real meeting: spend 2 hours, identify 20-40 issues.

https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee798544(v=cs.20)

Use Software Composition Analysis and Governance

- Vulnerabilities can come via third-party tools and components

Use Tools and Automation

- Tools must be integrated into the CI/CD pipeline.
- Tools must not require security expertise.
- Tools must avoid a high false-positive rate of reporting issues.
- Static analysis
- Dynamic analysis

Keep Credentials Safe

- Scan for keys in source code
- Put keys in a .env file (not in your source code)
  - Put .env in your .gitignore

Use Continuous Learning and Monitoring

- Continuous integration / continuous delivery
  - Should run analyses automatically
- Mean time to identify (MTTI)
- Mean time to contain (MTTC)

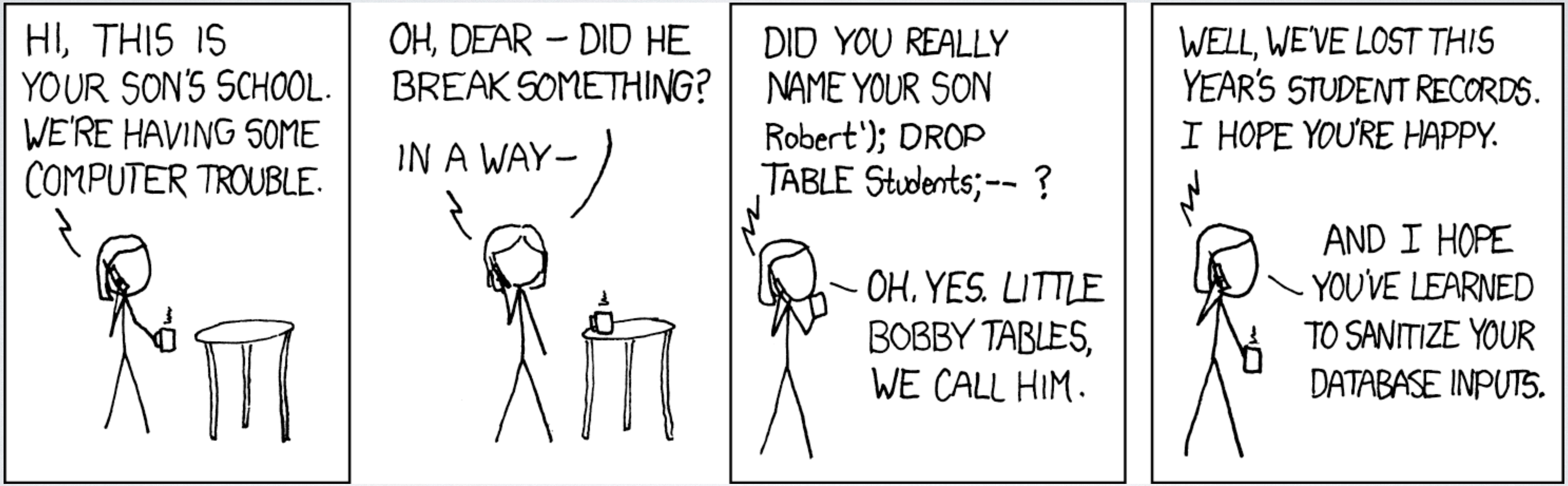
https://owasp.org/www-project-top-ten/

Top 10 Threats (OWASP 2025)

- Broken Access Control
- Security Misconfiguration
- Software Supply Chain Failures
- Cryptographic Failures
- Injection
- Insecure Design
- Authentication Failures
- Software or Data Integrity Failures
- Security Logging and Alerting Failures
- Mishandling of Exceptional Conditions

Mitigating Key Threats

Threat 1: Untrusted Data


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Avoiding Injection Attacks

- Validate input
- Avoid eval()
- Sanitize input when constructing SQL queries

Cross-Site Scripting (XSS) Attacks

1. Untrusted data enters web app
2. Data is included in content sent to a user (victim)

Example source: https://owasp.org/www-community/attacks/xss/

XSS Example

User clicks a link sent in email: `http://www.ucsd.edu/<script>alert("TEST");</script>`

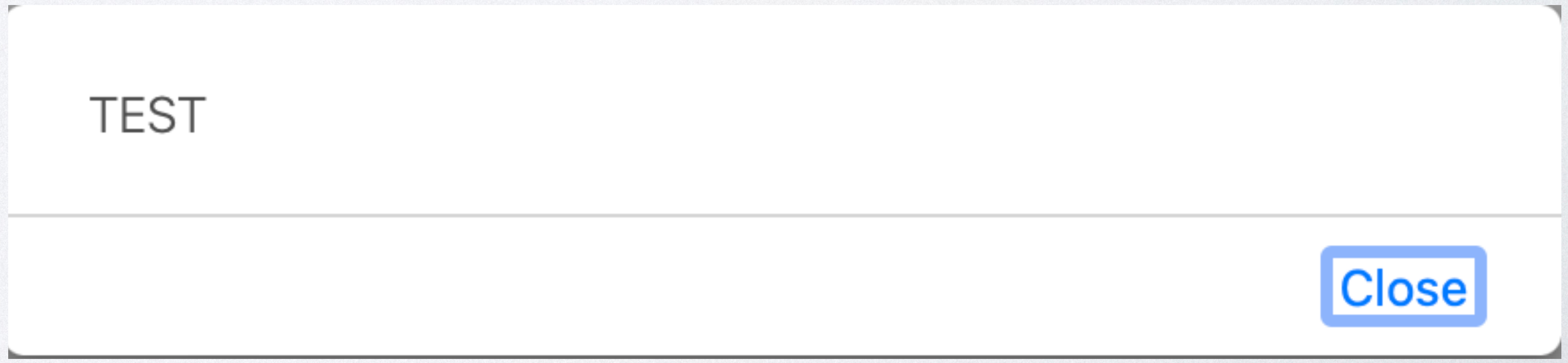
(user thinks this is OK because ucsd.edu is trusted)

Suppose ucsd.edu is vulnerable:

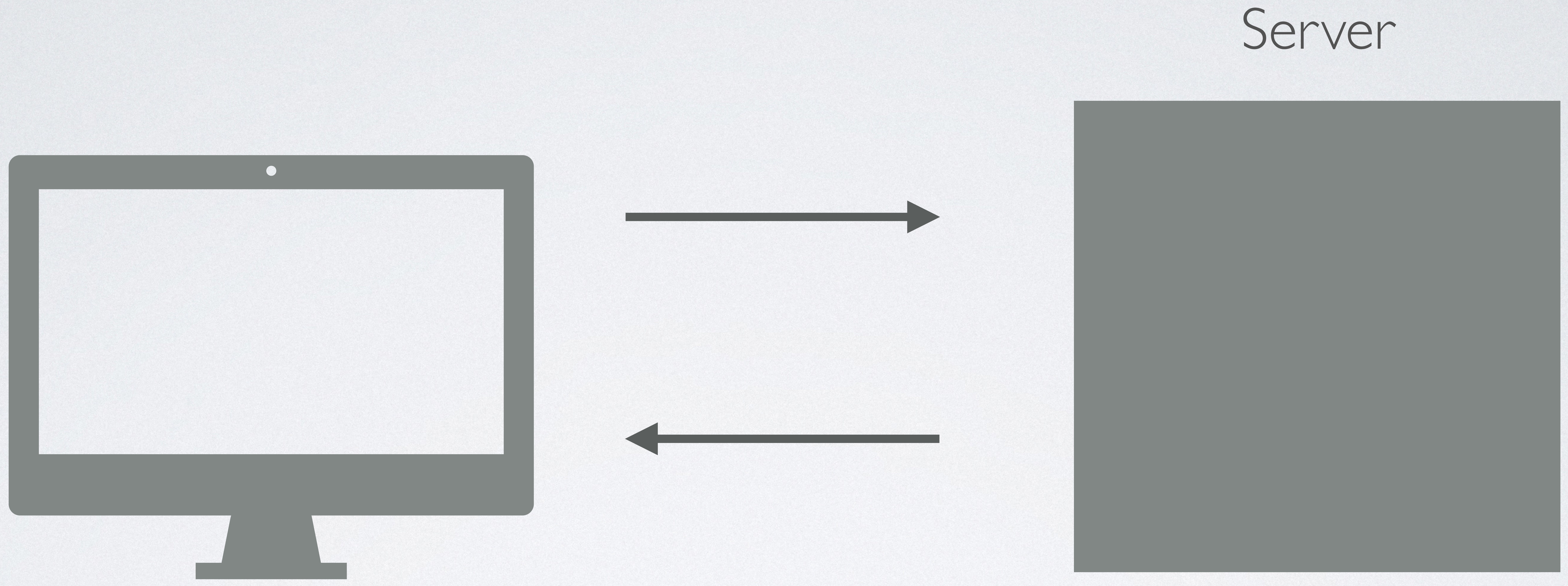
```
<html>
<body>
<?php
print "Not found: " . urldecode($_SERVER["REQUEST_URI"]);
?>

</body>
</html>
```

User is surprised to see an alert:


TEST

Threat 2: Bad Authentication



"Am I talking to a legitimate server?"

"Is the client who they say they are?"

Use TLS to check server's certificate

Check user credentials

Authentication vs. Authorization

- Authentication: are you who you say you are?
- Authorization: Given who you are, what can you do?
  - Policies enforced with access control

https://www.icann.org/en/blogs/details/what-is-authorization-and-access-control-2-12-2015-en

Use letsencrypt.com for Free Certificates

- Without a certificate, your users can be victims of a man-in-the-middle attack

Password Basics

- Someone might steal the password file!
- If it has plaintext passwords, all users are compromised.
- Solution: store only *cryptographic hashes* of passwords
- Assumption: inverting a cryptographic hash function is infeasible
  - $h^{-1}(h(p)) = p$

Password Cracking

- Brute force: try all strings
    - Mitigation: large space of passwords
    - Mitigation: avoid commonly-used passwords ("password")
- *Rainbow table*: pre-compute hashes of common passwords
    - Search hashes in stolen password table for known passwords
    - Mitigation: salts

What if Two Users Have the Same Password?

- username: harry; password: ucsd4life
- username: bovik; password: ucsd4life
- sha256sum("ucsd4life") = 5a321b082a1e8c97f1af3314c374780d44bb7f8dce4107231660ba0a6b852d43
- Both users' passwords hash to the same value!
- An attacker who compromises harry's account and gets a copy of the password database also gets access to bovik's account.

Salts

- Solution: each user gets a random "salt"

username	salt	password
harry	y893r2e	sha256sum("harryy893r2e")
bovik	asdffdsjlkfs	sha256sum("bovikasdffdsjlkfs")

Secrets

- Secrets do not go in your repository!
  - Secrets go in config files (store these somewhere safe)
- Passwords do not go in your database!
  - Salted, hashed passwords go in your database

Principle of Least Privilege

- Only authorize access that is actually needed
- Does Chancellor Khosla need admin access to the course web site?
  - We trust him, but he doesn't need access.
  - If the chancellor's account were compromised, the web site would be vulnerable too.

Defense in Depth

- Not enough to just have one security check
- Individual checks can be imperfect
- Example: encrypt traffic *and* restrict access to the VPN (Virtual Private Network)

Conclusion

- Security is different from "regular" software engineering:
  - Need to have an adversarial mindset
- Security requires up-front work
  - Waiting until the last minute will result in vulnerabilities, risk, and expensive fixes