

Software Engineering & Security

Advanced Software Engineering
Spring 2023

Security isn't (always) free

In software, as in the real world...

- You just moved to a new house, someone just moved out of it. What do you do to protect your belongings/property?
- Do you change the locks?
- Do you buy security cameras?
- Do you hire a security guard?
- Do you even bother locking the door?



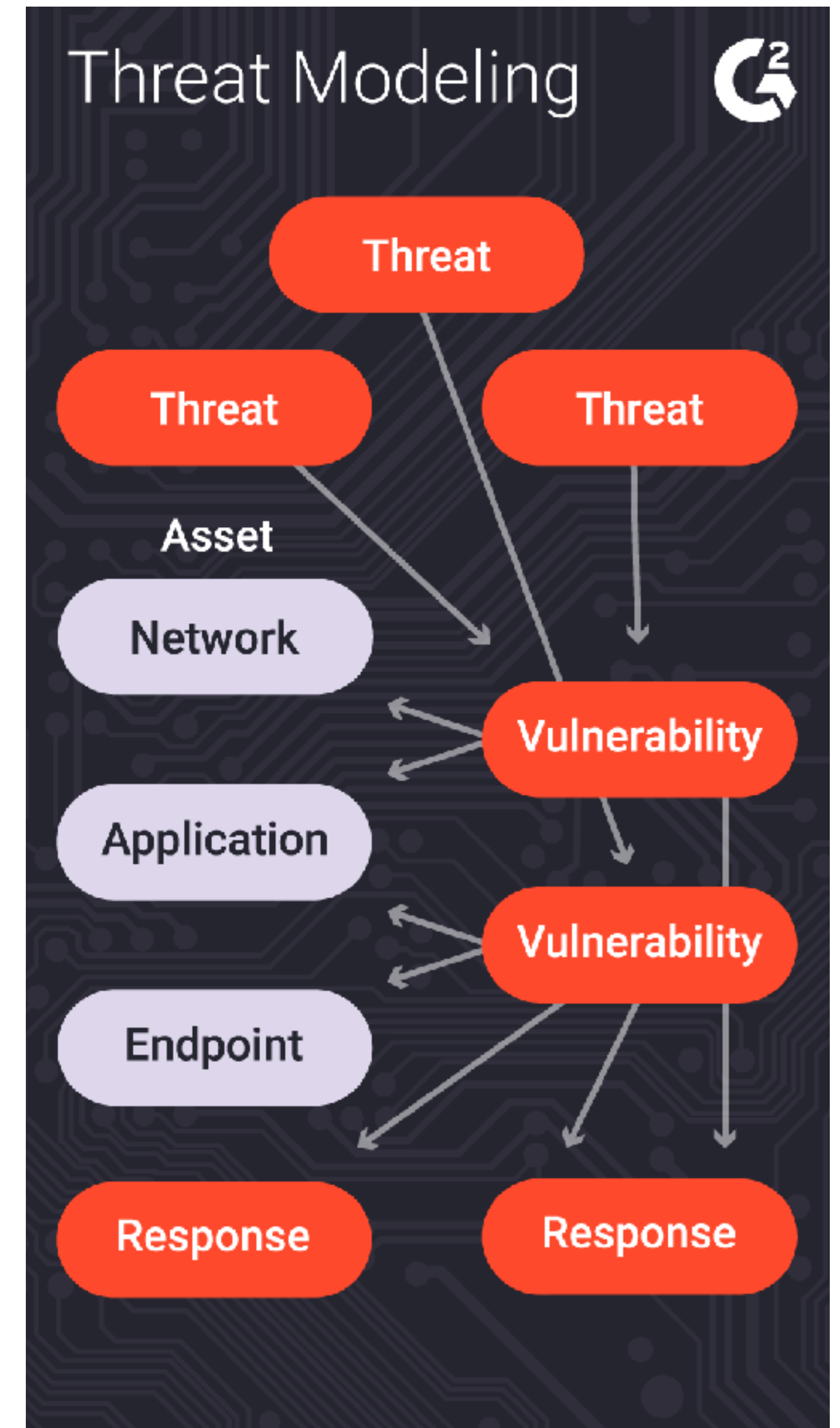
Security is about managing risk

Cost of attack vs cost of defense?

- Increasing security might:
 - Increase development & maintenance cost
 - Increase infrastructure requirements
 - Degrade performance
- But, if we are attacked, increasing security might also:
 - Decrease financial and intangible losses
- So: How likely do we think we are to be attacked in way **X**?

Threat Models help analyze these tradeoffs

- What is being defended?
 - What resources are important to defend?
 - What malicious actors exist and what attacks might they employ?
 - What value can an attacker extract from a vulnerability?
- Who do we trust?
 - What entities or parts of system can be considered secure and trusted
- Plan responses to possible attacks
 - Prioritize?



A Baseline Security Architecture (1)

Best practices applicable in most situations

- Trust:
 - Developers writing our code (at least for the code they touch)
 - Server running our code
 - Popular dependencies that we use and update
- Don't trust:
 - Code running in browser
 - Inputs from users
 - Other employees (different employees should have access to different resources)

A Baseline Security Architecture (2)

Best practices applicable in most situations

- Practice good security practices:
 - Encryption (all data in transit, sensitive data at rest)
 - Code signing, multi-factor authentication
 - Encapsulated zones/layers of security (different people have access to different resources)
 - Log everything! (employee data accesses/modifications) (maybe)
- Bring in security experts early for riskier situations

OWASP Top Security Risks

All 10: <https://owasp.org/www-project-top-ten/>

- Broken authentication + access control
- Cryptographic failures
- Code injection (various forms - SQL/command line/XSS/XML/deserialization)
- Weakly protected sensitive data
- Using components with known vulnerabilities

Threats discussed in this lesson:

- Threat 1: Code that runs in an untrusted environment
- Threat 2: Inputs that are controlled by an untrusted user
- Threat 3: Bad authentication (of both sender and receiver!)
- Threat 4: Untrusted Inputs
- Threat 5: Software supply chain delivers malicious software
- Recurring theme: No silver bullet

Threat 1: Code that runs in an untrusted environment

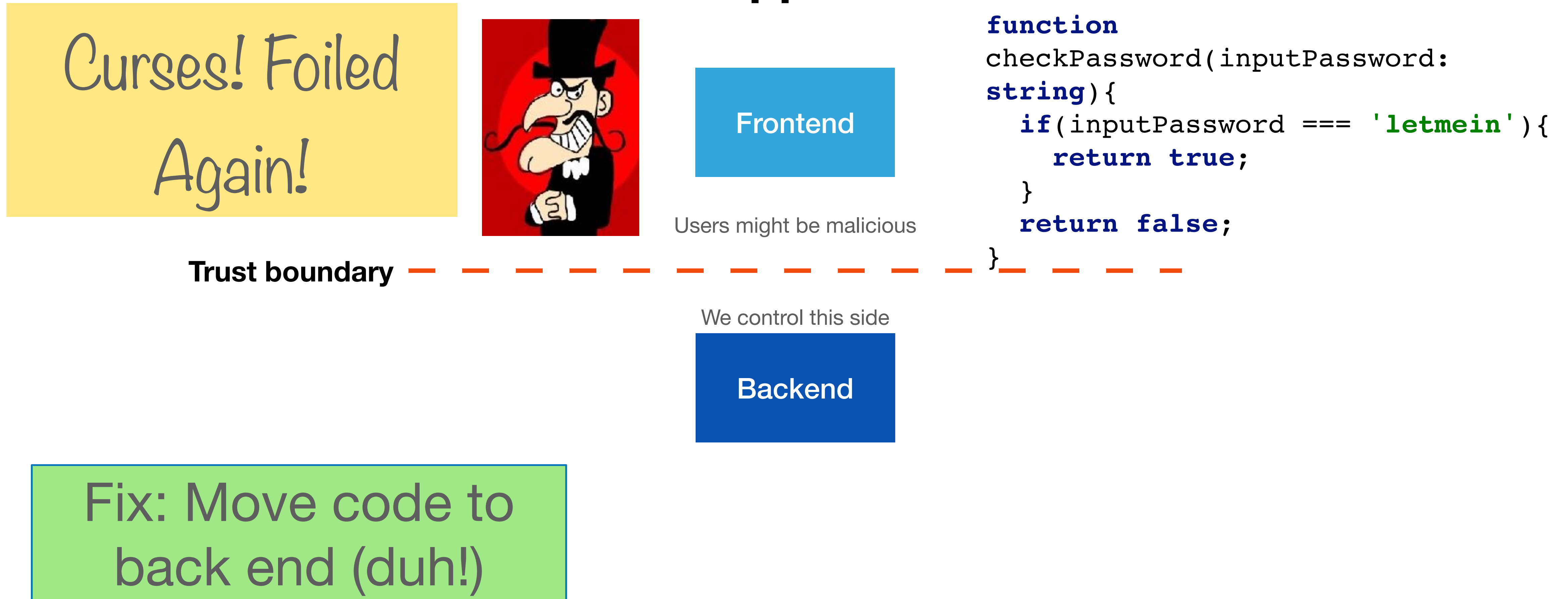
Authentication code in a web application

```
function checkPassword(inputPassword: string) {  
  if(inputPassword === 'letmein') {  
    return true;  
  }  
  return false;  
}
```

Should this go in our frontend code?

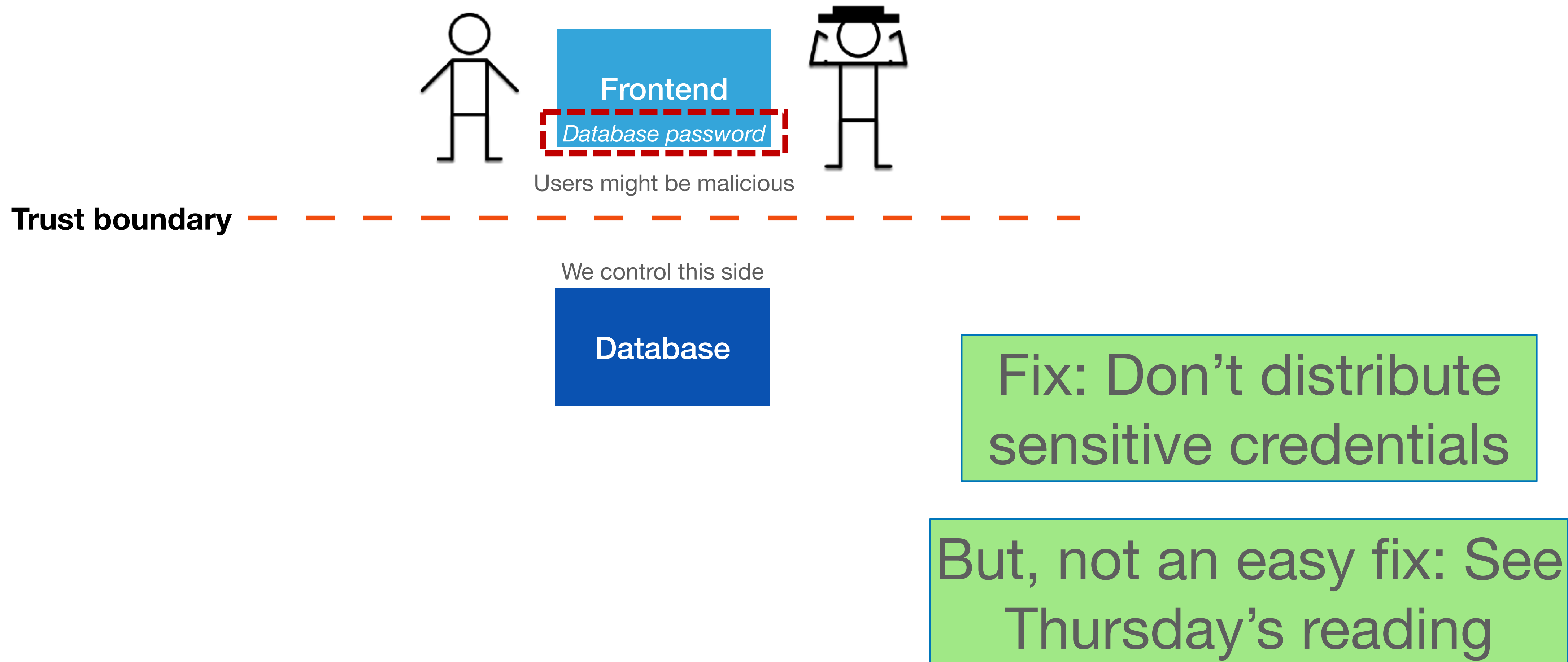
Threat 1: Code that runs in an untrusted environment

Authentication code in a web application

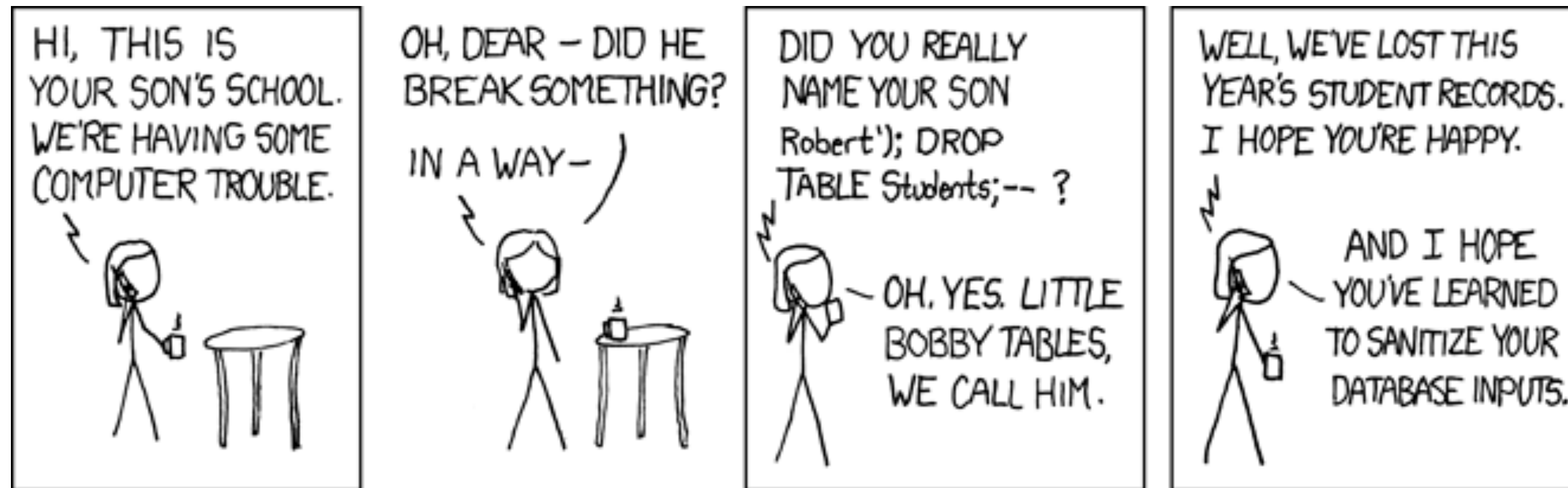


Threat Category 1: Code that runs in an untrusted environment

Access controls to database

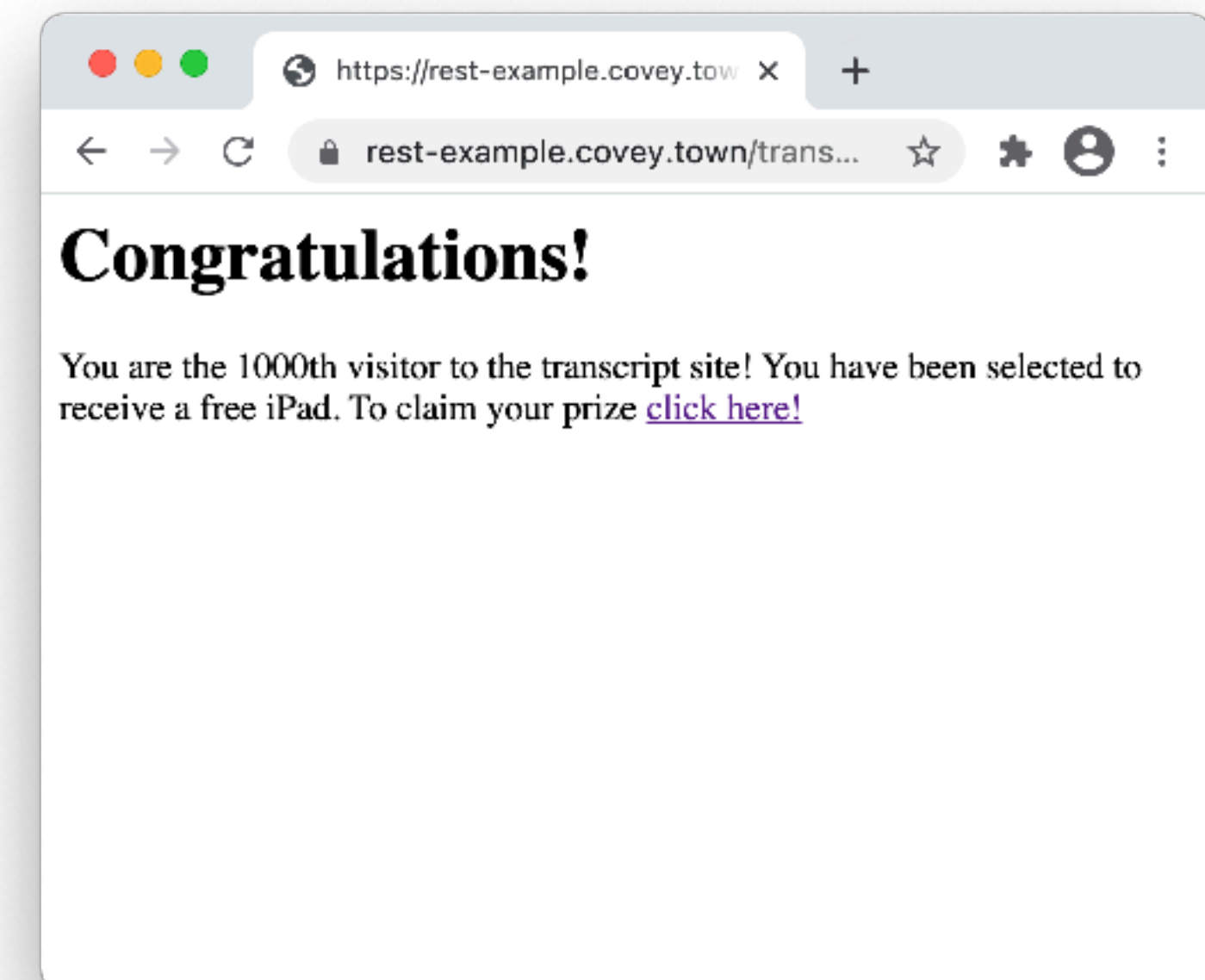
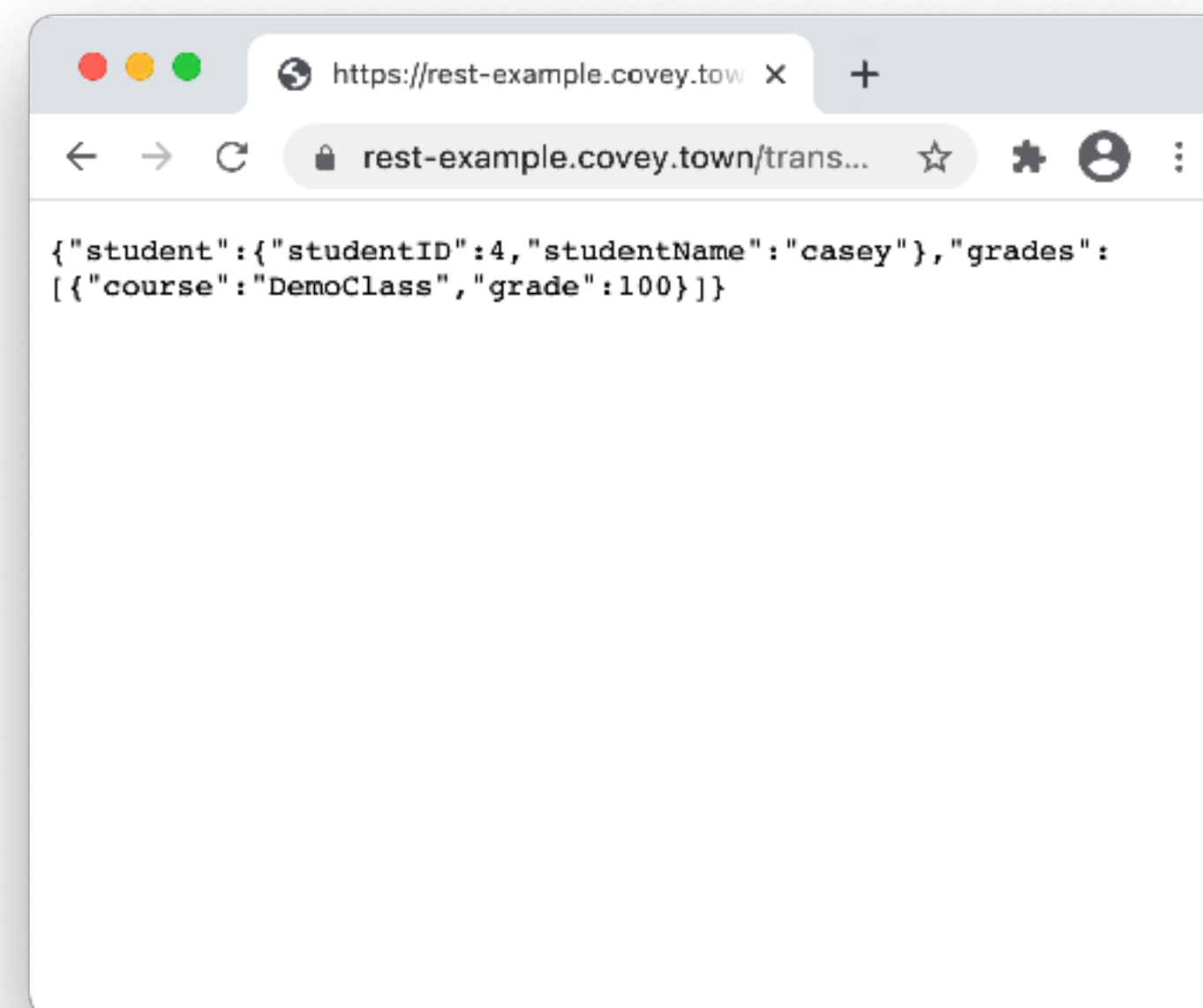
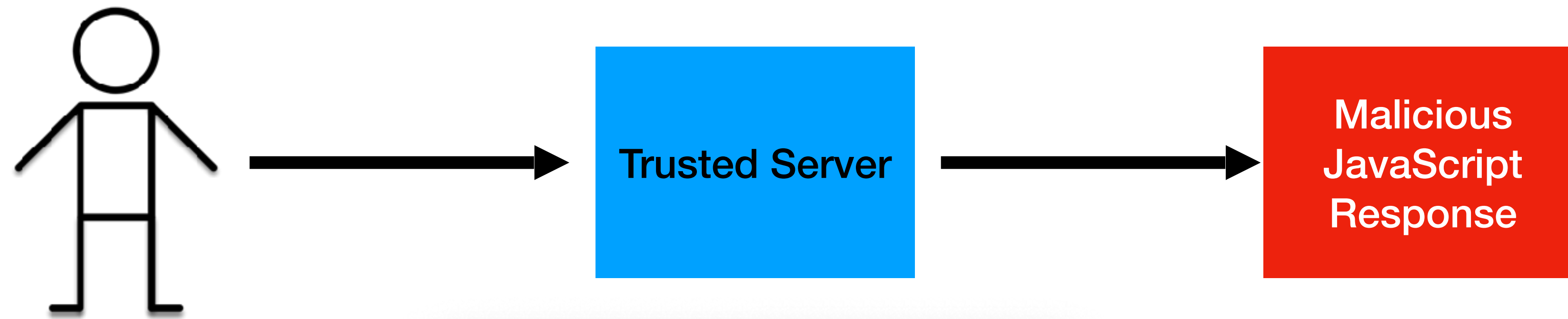


Threat 2: Data controlled by a user flowing into our trusted codebase



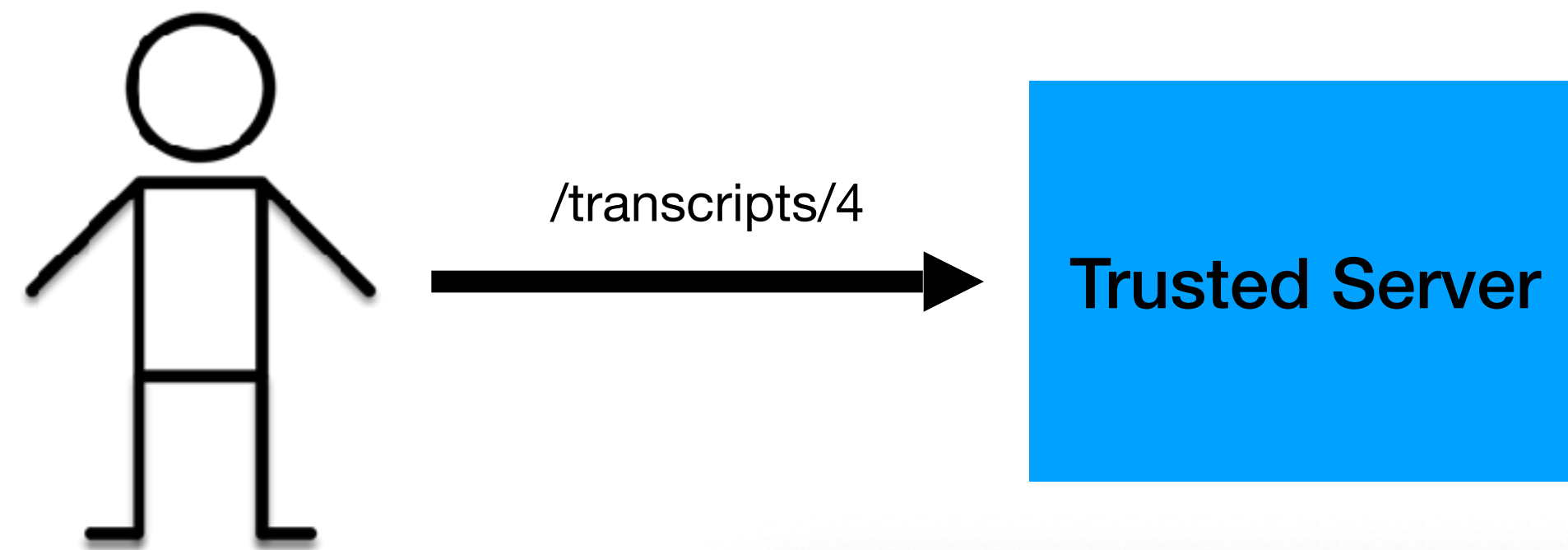
Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability

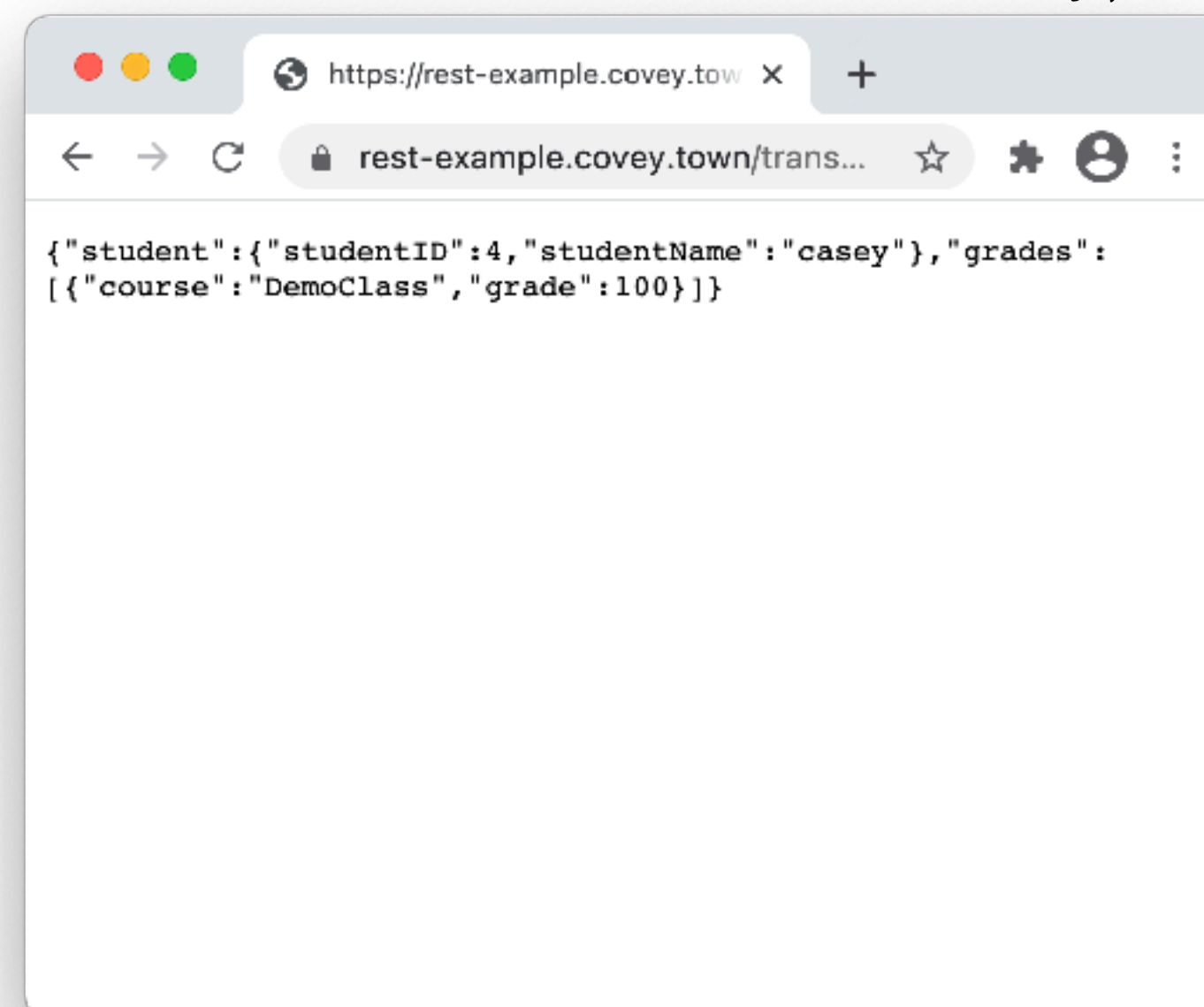


Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability

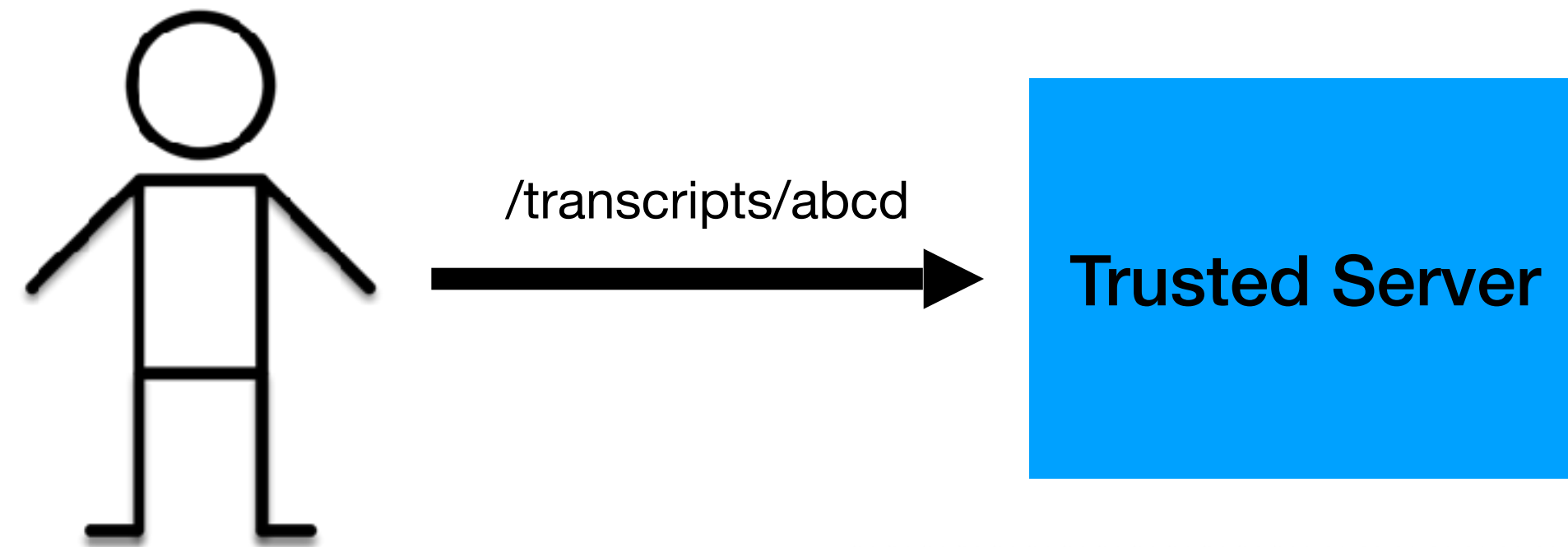


```
app.get('/transcripts/:id', (req, res) => {  
  // req.params to get components of the path  
  const {id} = req.params;  
  const theTranscript = db.getTranscript(parseInt(id));  
  if (theTranscript === undefined) {  
    res.status(404).send(`No student with id = ${id}`);  
  }  
  {  
    res.status(200).send(theTranscript);  
  }  
});
```

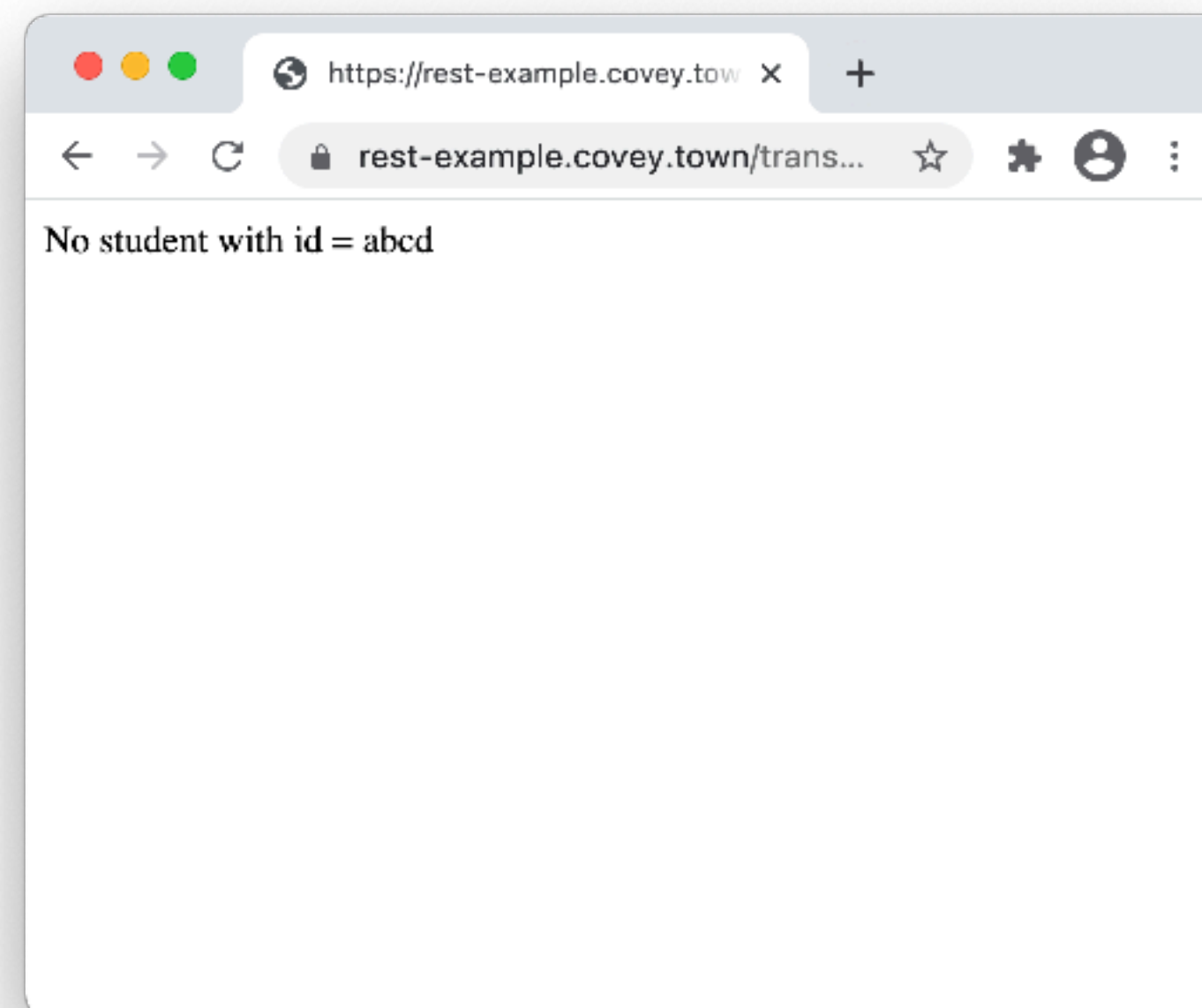


Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability

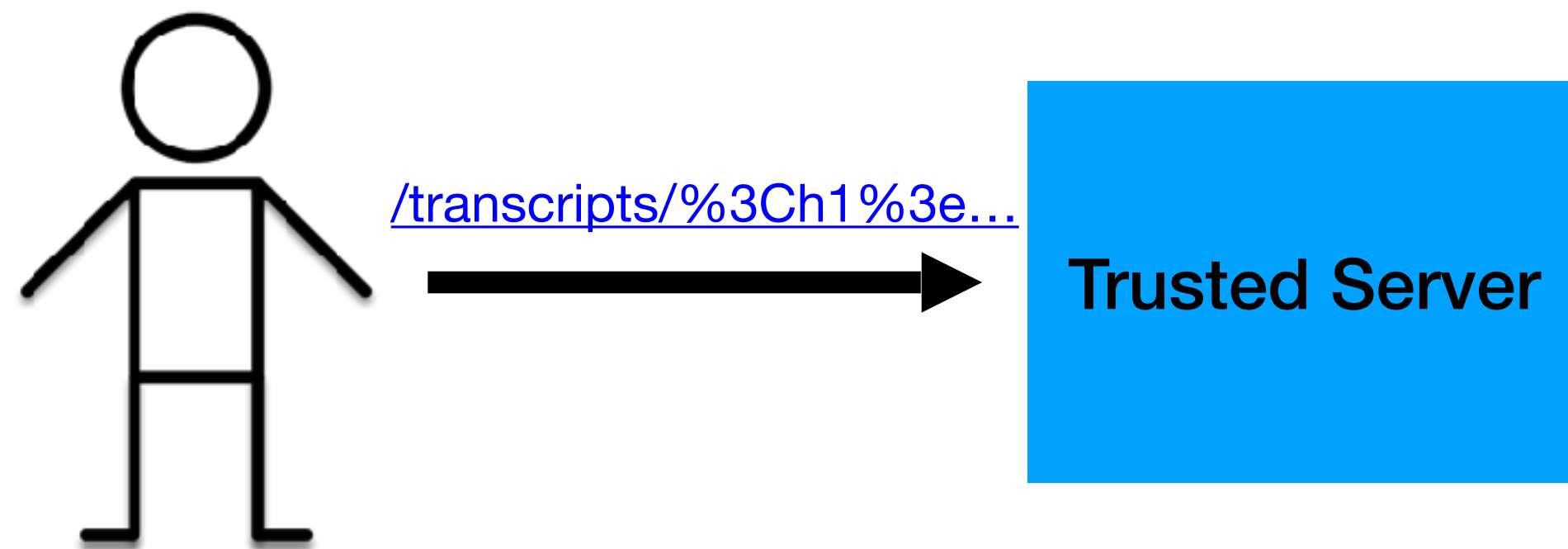


```
app.get('/transcripts/:id', (req, res) => {  
  // req.params to get components of the path  
  const {id} = req.params;  
  const theTranscript = db.getTranscript(parseInt(id));  
  if (theTranscript === undefined) {  
    res.status(404).send(`No student with id = ${id}`);  
  }  
  {  
    res.status(200).send(theTranscript);  
  }  
});
```

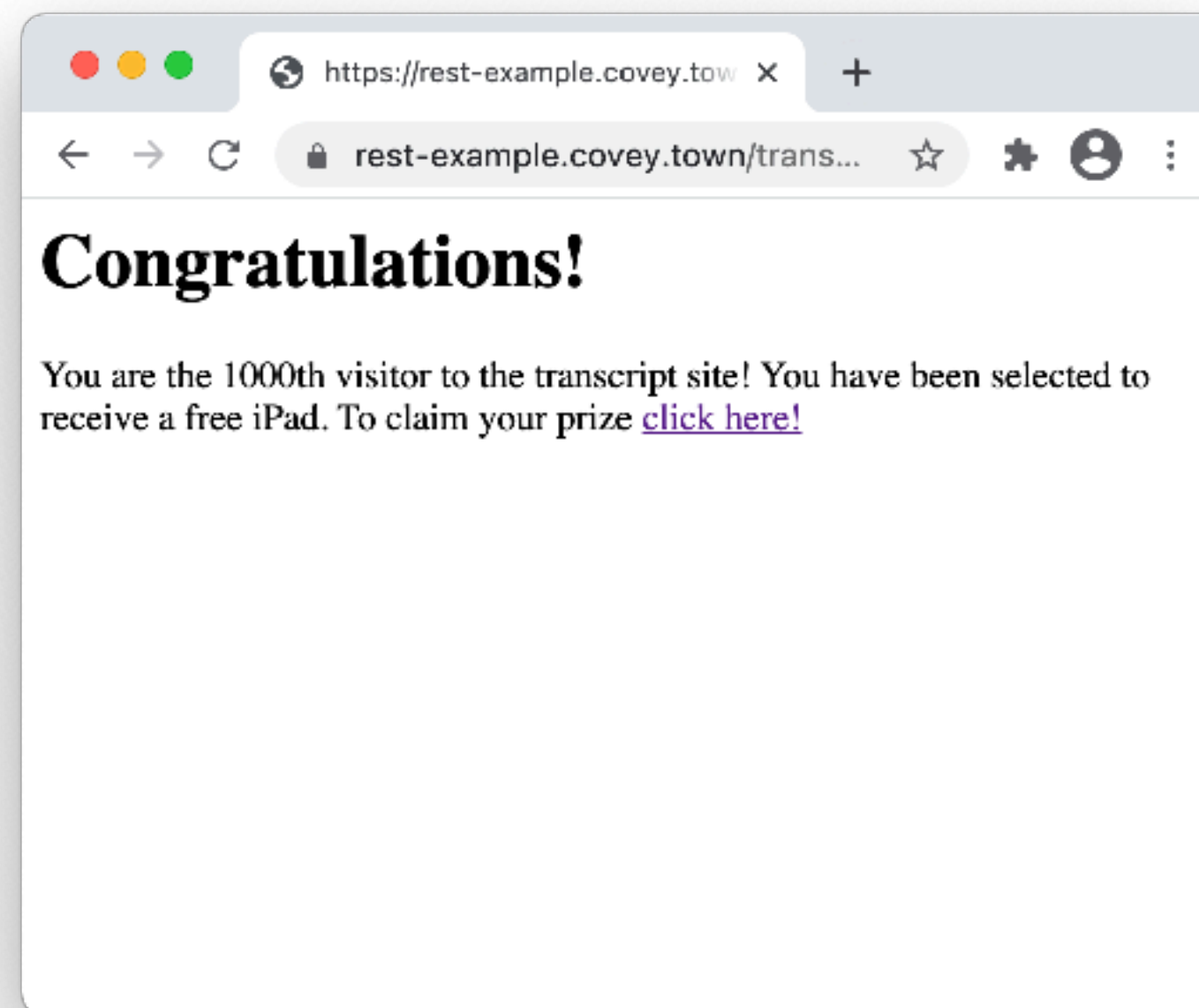
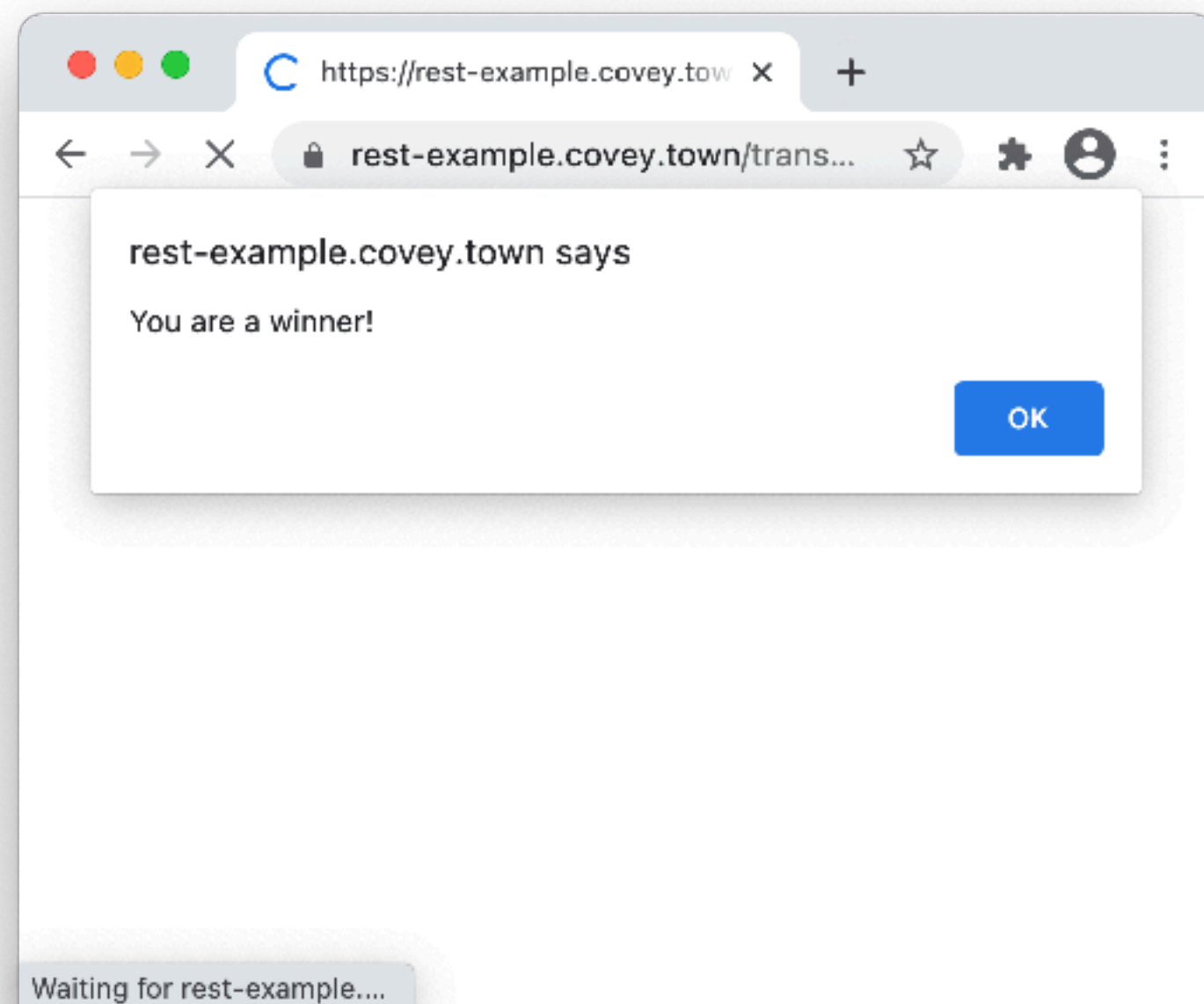


Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability



```
app.get('/transcripts/:id', (req, res) => {  
  // req.params to get components of the path  
  const {id} = req.params;  
  const theTranscript = db.getTranscript(parseInt(id));  
  if (theTranscript === undefined) {  
    res.status(404).send(`No student with id = ${id}`);  
  }  
  res.status(200).send(theTranscript);  
});
```



```
<h1>Congratulations!</h1>  
You are the 1000th visitor to the  
transcript site! You have been selected  
to receive a free iPad. To claim your  
prize <a href='\"https://www.youtube.com/  
watch?v=DLzxrzFCyOs'>click here!</a>  
<script language="javascript">  
document.getRootNode().body.innerHTML=  
'<h1>Congratulations!</h1>You are the  
1000th visitor to the transcript site!  
You have been selected to receive a  
free iPad. To claim your prize <a  
href="\"https://www.youtube.com/watch?  
v=DLzxrzFCyOs\">click here!</a>';  
alert('You are a winner!');  
</script>
```


Threat 2: Data controlled by a user flowing into our trusted codebase

Java code injection vulnerability in Apache Struts (@Equifax)



The screenshot shows the Equifax website header with the logo, a language selector set to 'English', and a link to 'Return to equifax.com'. The main content area features a large red banner with the text '2017 Cybersecurity Incident & Important Consumer Information'. Below the banner, there is a news article titled 'Equifax Says Cybersecurity Breach Has Cost \$1.4 Billion' with a 'Contact Us' link. Social media icons for Facebook, Twitter, and Email are visible in the bottom right corner of the banner area.

CVE-2017-5638 Detail

Current Description

The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to **execute arbitrary commands via a crafted Content-Type, Content-Disposition, or Content-Length HTTP header**, as exploited in the wild in March 2017 with a Content-Type header containing a #cmd= string.

Threat 2: Data controlled by a user flowing into our trusted codebase

Java code injection vulnerability in Log4J

Extremely Critical Log4J Vulnerability Leaves Much of the Internet at Risk

December 10, 2021 Ravie Lakshmanan



CVE-2021-44228 Detail Current Description

Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related **endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled.** From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

The Apache Software Foundation has **actively exploited** zero-day vulnerabilities in Apache Log4j Java-based logging systems to execute malicious code on systems.

<https://nvd.nist.gov/vuln/detail/CVE-2021-44228>

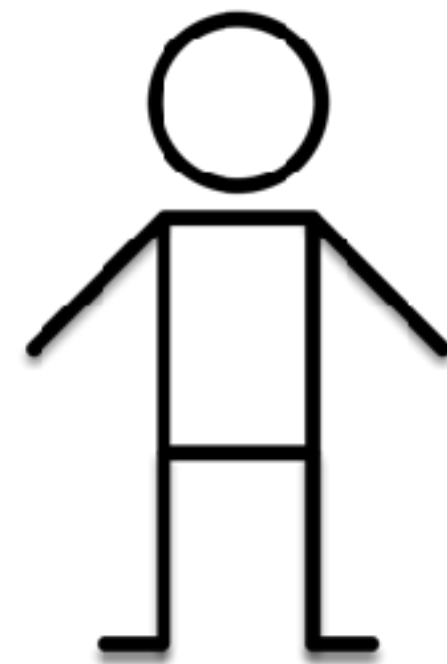
<https://thehackernews.com/2021/12/extremely-critical-log4j-vulnerability.html>



The APT41 group compromised at least six U.S. state government networks between May and February in a "deliberate campaign" that reflects new attack vectors and retooling by the prolific Chinese state-sponsored group.

<https://duo.com/decipher/apt41-compromised-six-state-government-networks>

Threat 3: Bad authentication



HTTP Request



HTTP Response



client page
(the “user”)

server

Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

Threat 3: Bad authentication



Do I trust that this response really came from the server?

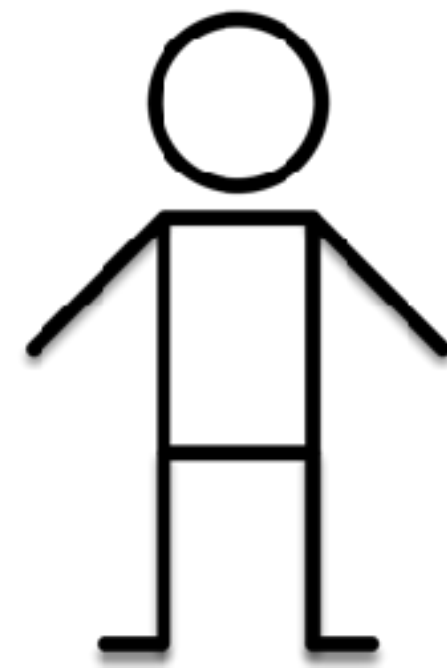
Do I trust that this request *really* came from the user?

Threat 3: Bad authentication

Preventing the man-in-the-middle with SSL



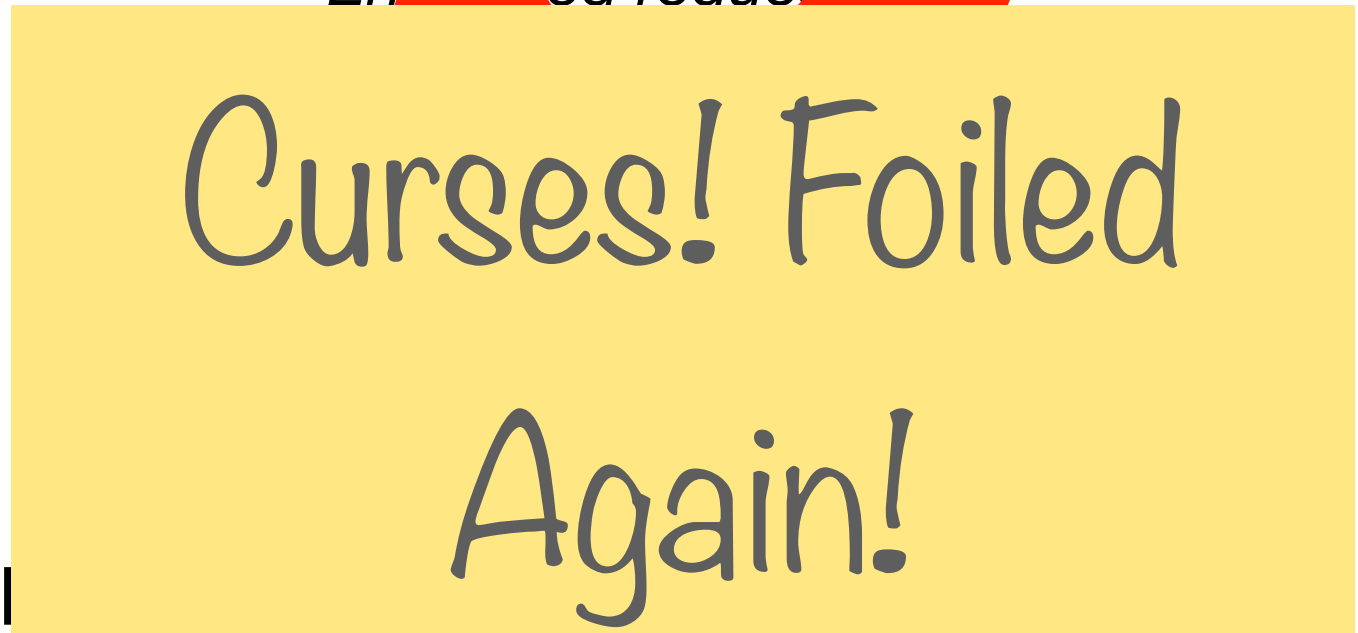
Preventing the man-in-the-middle with SSL



HTTP Request



Encrypted request



HTTP Response



Encrypted response



server



Your connection is not private

Attackers might be trying to steal your information from **192.168.18.4** (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID



[amazon.com](#) certificate
(AZ's public key + CA's sig)


SSL: A perfect solution?

Certificate authorities


- A certificate authority (or CA) binds some public key to a real-world entity that we might be familiar with
- The CA is the clearinghouse that verifies that [amazon.com](https://www.amazon.com) is truly [amazon.com](https://www.amazon.com)
- CA creates a certificate that binds [amazon.com](https://www.amazon.com)'s public key to the CA's public key (signing it using the CA's private key)

Certificate Authorities issue SSL Certificates

Amazon



[amazon.com](https://www.amazon.com)
private key



[amazon.com](https://www.amazon.com)
public key

Some world proof that we are
really
amazon.com

[amazon.com](https://www.amazon.com) certificate



[amazon.com](https://www.amazon.com) certificate
(AZ's public key + CA's sig)

Certificate Authority




CA private key



CA public key

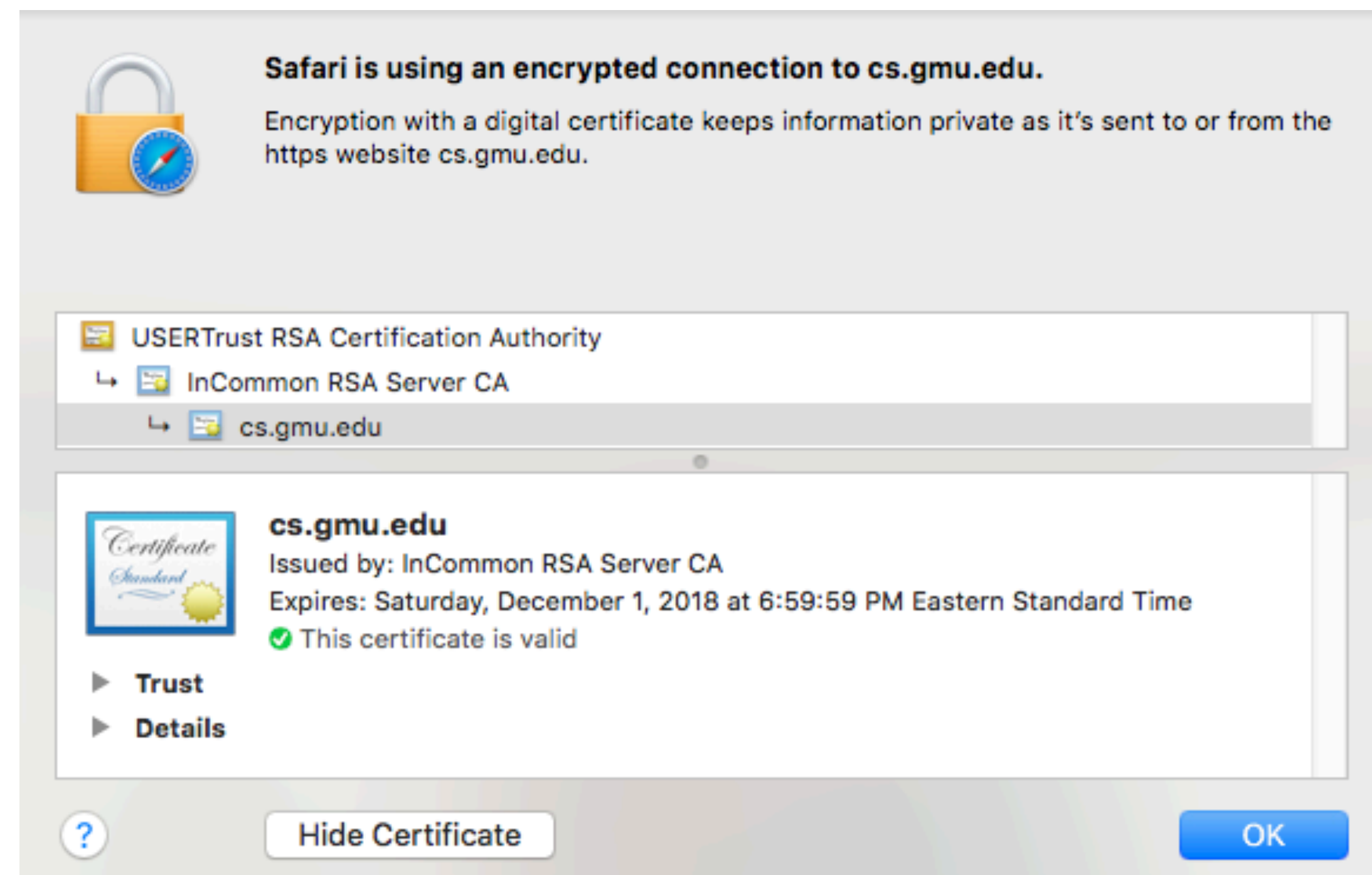
My Laptop



CA public key

Certificate Authorities are Implicitly Trusted

- Note: We had to already know the CA's public key
- There are a small set of “root” CA’s (think: root DNS servers)
- Every computer/browser is shipped with these root CA public keys



Should Certificate Authorities be Implicitly Trusted?

Signatures only endorse trust if you trust the signer!

- What happens if a CA is compromised, and issues invalid certificates?
- Not good times.

Security

Comodo-gate hacker brags about forged certificate exploit


Tiger-blooded Persian cracker boasts of mighty exploits

Security

Fuming Google tears Symantec a new one over rogue SSL certs

We've got just the thing for you, Symantec ...

By Iain Thomson in San Francisco 29 Oct 2015 at 21:32

36  SHARE ▼




Google has read the riot act to Symantec, scolding the security biz for its

You can do this for your website for free

letsencrypt.com



The screenshot shows the homepage of Let's Encrypt. At the top left is the Let's Encrypt logo, which consists of a yellow sun icon above a blue padlock icon with the text "Let's Encrypt" in blue. To the right of the logo is a navigation menu with the following items: "Documentation", "Get Help", "Donate" (with a small blue dot), "About Us" (with a small blue dot), and "Languages" (with a small flag icon and a dropdown arrow). Below the navigation menu is a large hero section with a dark blue background and a yellow-to-blue gradient. In the center of the hero section is a white rounded rectangle containing the following text: "A nonprofit Certificate Authority providing TLS certificates to **300 million** websites." Below this text is a line of smaller text: "We were awarded the Levchin Prize for Real-World Cryptography! [Learn more](#)". At the bottom of the white rectangle are two blue-outlined buttons: "Get Started" and "Sponsor".

 Documentation Get Help Donate • About Us • Languages 

A nonprofit Certificate Authority providing TLS certificates to **300 million** websites.

We were awarded the Levchin Prize for Real-World Cryptography! [Learn more](#)

[Get Started](#) [Sponsor](#)

Threat 4: Untrusted Inputs

Restrict inputs to only “valid” or “safe” characters

- Special characters like <, >, ‘, “ and ` are often involved in exploits involving untrusted inputs

Fix: Always use input validation

Create password

Please create your password. Click [here](#) to read our password security policy.

Your password needs to have:

- ✓ At least 8 characters with no space
- ✓ At least 1 upper case letter
- ✓ At least 1 number
- ✓ At least 1 of the following special characters from ! # \$ ^ * (other special characters are not supported)

Password
.....

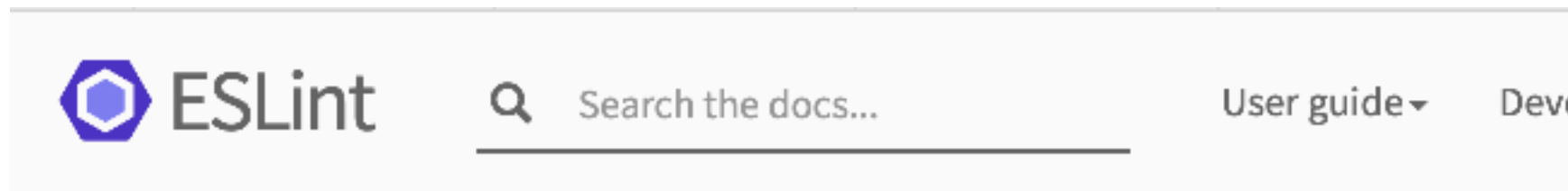
- ⚠ Your password must contain a minimum of 8 characters included with at least 1 upper case letter, 1 number, and 1 special character from !, #, \$, ^, and * (other special characters are not supported).

Other ways to sanitize your inputs:

- Sanitize inputs – prevent them from being executable
- Avoid use of languages or features that can allow for remote code execution, such as:
 - `eval()` in JS – executes a string as JS code
 - Query languages (e.g. SQL, LDAP, language-specific languages like OGNL in java)
 - Languages that allow code to construct arbitrary pointers or write beyond a valid array index

Threat 5: Software Supply Chain

Do we trust our own code? Third-party code provides an attack vector



Postmortem for Malicious Packages Published on July 12th, 2018

Summary

On July 12th, 2018, an attacker compromised the npm account of an ESLint maintainer and published malicious versions of the `eslint-scope` and `eslint-config-eslint` packages to the npm registry. On installation, the malicious packages downloaded and executed code from `pastebin.com` which sent the contents of the user's `.npmrc` file to the attacker. An `.npmrc` file typically contains access tokens for publishing to npm.

The malicious package versions are `eslint-scope@3.7.2` and `eslint-config-eslint@5.0.2`, both of which have been unpublished from npm. The `pastebin.com` paste linked in these packages has also been taken down.

npm has revoked all access tokens issued before 2018-07-12 12:30 UTC. As a result, all access tokens compromised by this attack should no longer be usable.

The maintainer whose account was compromised had reused their npm password on several other sites and did not have two-factor authentication enabled on their npm account.

We, the ESLint team, are sorry for allowing this to happen. We

<https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes>

THE VERGE

Photo Illustration by Grayson Blackmon / The Verge

PODCASTS

HARD LESSONS OF THE SOLARWINDS HACK

Cybersecurity reporter Joseph Menn on the massive breach the US didn't see coming

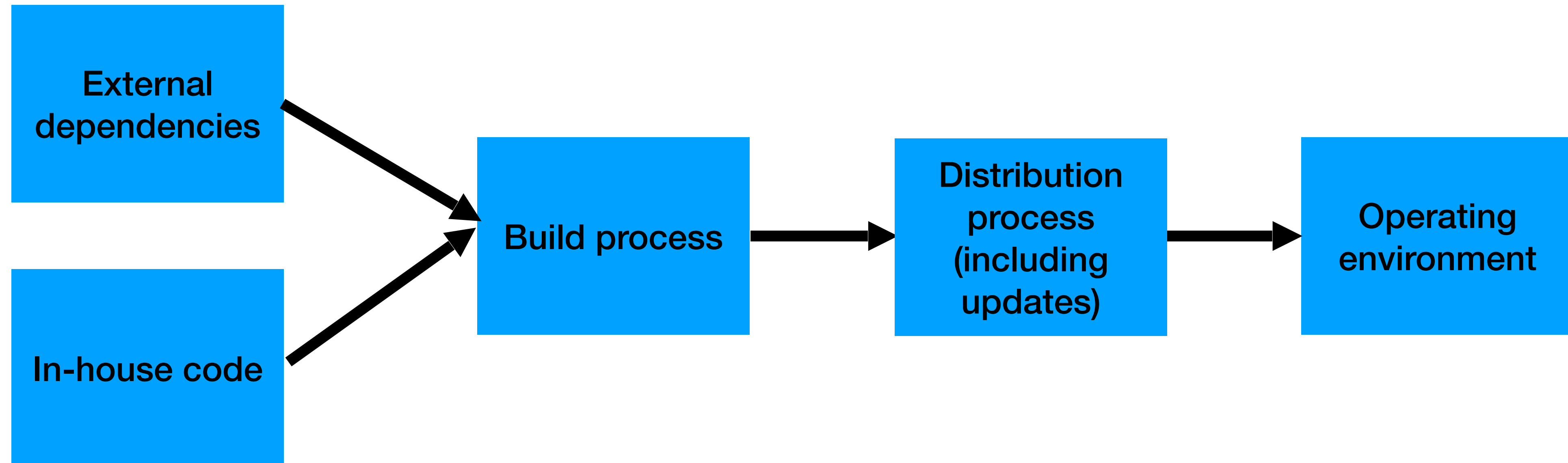
By Nilay Patel | @reckless | Jan 26, 2021, 9:13am EST



In December, details came out on one of the most massive breaches of US cybersecurity in recent history. A group of hackers, likely from the Russian government, had gotten into a network management company called SolarWinds and infiltrated its customer base. The breach allowed hackers to breach even the most secure systems, including the US Treasury and departments of

<https://www.theverge.com/2021/1/26/22248631/solarwinds-hack-cybersecurity-us-menn-decoder-podcast>

Threat 5: The software supply chain has many points of weakness



What are Weak Links in the npm Supply Chain?

Discussion: Zahan et al, ICSE 22

- What is a “weak link” signal in a software supply chain?
 - An indicator of a potential vulnerability (weakness)
- Some signals are direct vectors (with processes to mitigate) vs part of systemic issues
- Weak link signals:
 - Expired maintainer domain - Someone else can buy the domain and take over the email address
 - Installation scripts - Presence of installation scripts is troublesome
 - Unmaintained packages - Might invite malicious contributors, indicate that vulnerabilities won't be patched
 - Too many maintainers + too many contributors
- Other weak link signals not in this paper but should be considered in future?
 - Rapid updates - first update might be a test

What are Weak Links in the npm Supply Chain?

Discussion: Zahan et al, ICSE 22

- Data-driven attack example:
 - Scary. Did they actually confirm it was feasible? Have the domains been secured?
 - How “important” are those 899 packages?
 - Counter-measure: circle of trust between maintainers

Practical automated detection of malicious npm packages

Discussion: Adriana Sejfia, Max Schäfer at ICSE 22

- Definition: what is a “malicious package on NPM?”
 - Any package that performs an operation that compromises security requirements
 - Does intention matter?
 - Any violation of the NPM terms of service - including “malware” that is things like “OFFICE 2010 TOOLKIT BEST.RAR”
- What is a reasonable baseline for detecting malicious packages?
 - Whatever we have been doing already, maybe manually
- What kinds of features might we use to detect malware?
 - Obfuscation (why would OSS be obfuscated?)
 - Similarity to existing packages (malware or not)
 - Typos in the package name (eslint: esInt aslant)
- What is different about detecting “malicious packages” vs “malicious updates”?
 - Additional features to include: Geo location of the author, other anomalies in access. Have access to source code diff. What new behavior categorized by its use of other APIs/code constructs. Time since last update.

Practical automated detection of malicious npm packages

Discussion: Adriana Sejfia, Max Schäfer at ICSE 22

- What are the tradeoffs to prevent malicious packages on NPM?
 - Adding speed bumps to new accounts and new packages [DoS target]
 - Claiming that you are trying hard at this sets you up to fail in the court of public opinion when an adversary wins
 - Other barriers to entry for newcomers. Potential to delay patches.
 - Things cost money, how the heck do these people expect to make money?
 - Potential implications for privacy - anything requiring a real-world identity
- What is NPM's interest in detecting and preventing malicious packages?
- What is GitHub's interest detecting and preventing malicious packages?
 - As of 2020 Microsoft owns NPM :)
 - Solving this problem for NPM may help solve problems for other languages too