

CS 4530: Fundamentals of Software Engineering

Module 15: Software Engineering & Security

Jonathan Bell, Adeel Bhutta, Mitch Wand
Khoury College of Computer Sciences
© 2022, released under [CC BY-SA](#)

Learning Objectives for this Lesson

By the end of this lesson, you should be able to...

- Define key terms relating to software/system security
- Describe some of the tradeoffs between security and other requirements in software engineering
- Explain 5 common vulnerabilities in web applications and similar software systems, and describe some common mitigations for each of them.
- Explain why software alone isn't enough to assure security

Security: Basic Vocabulary 1 (“CIA”)

Security as a set of non-functional requirements

- Confidentiality: is information disclosed to unauthorized individuals?
- Integrity: is code or data tampered with?
- Availability: is the system accessible and usable?

Security: Basic Vocabulary 2

- Threat: potential event that could compromise a security requirement
- Security architecture: a set of mechanisms and policies that we build into our system to mitigate risks from threats
- Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise
- Attack: realization of a threat

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?



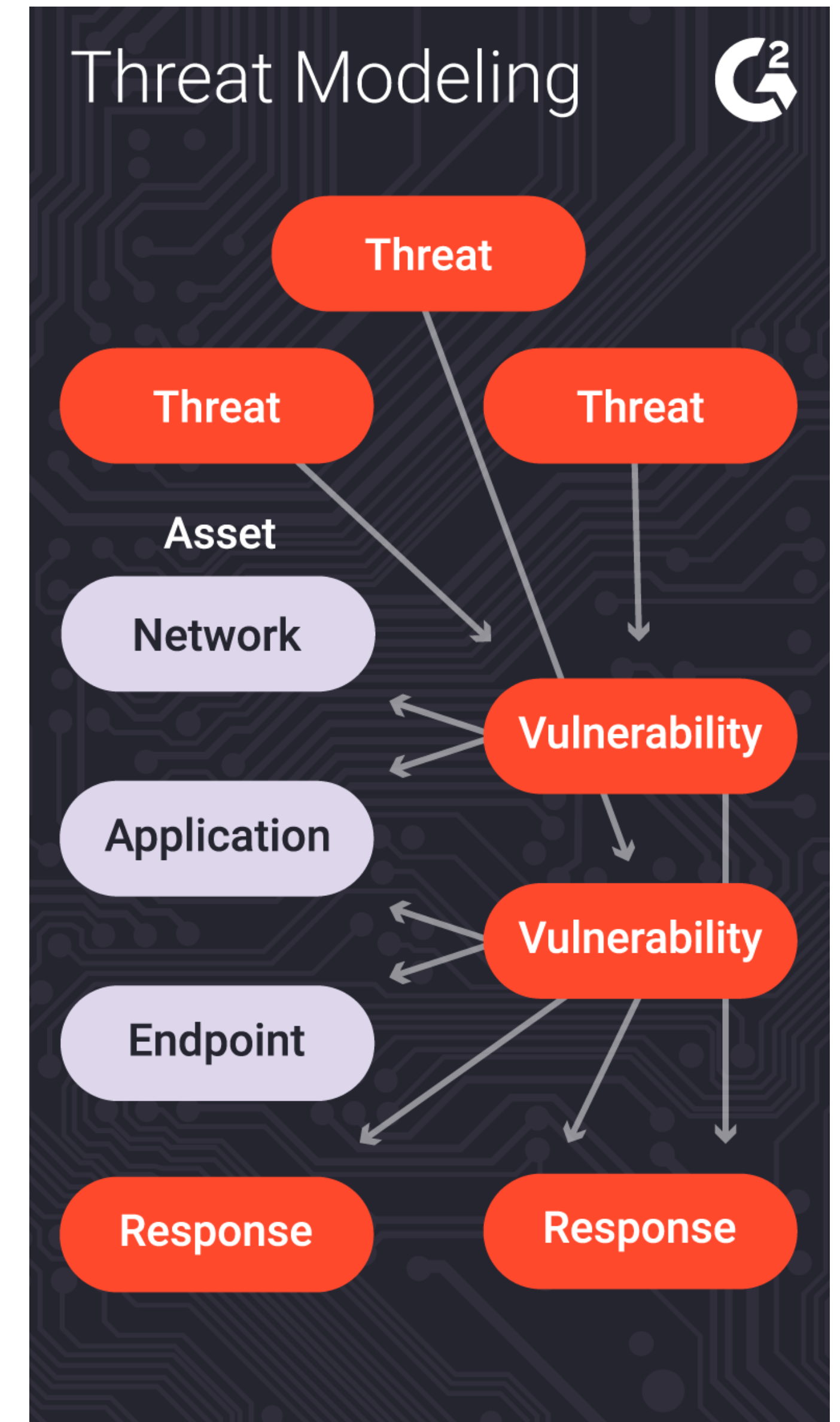
Security is about managing risk

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

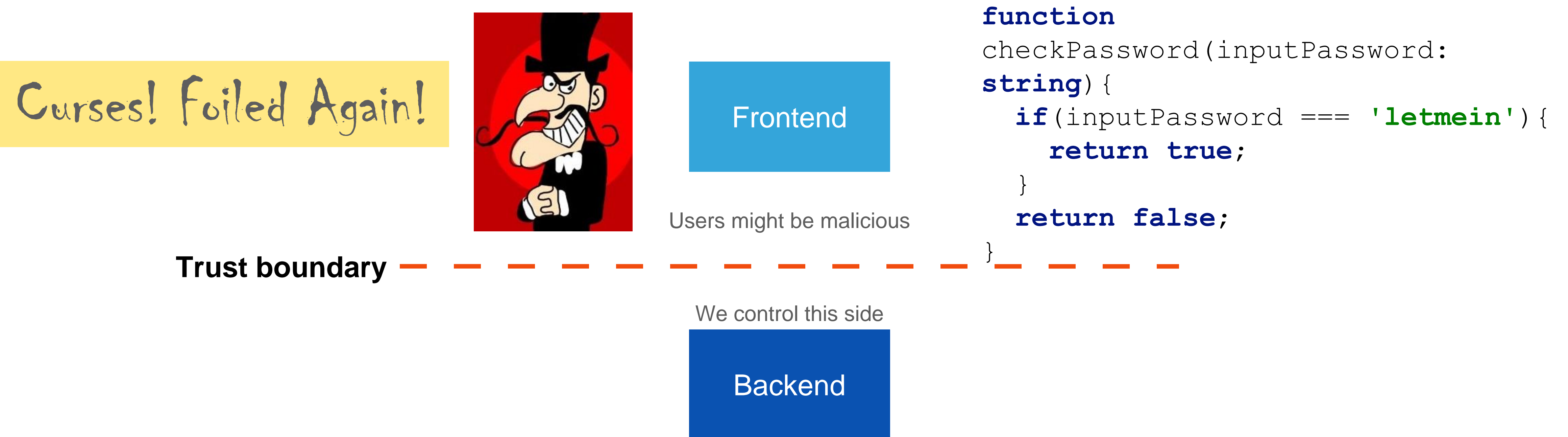
Authentication code in a web application

```
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

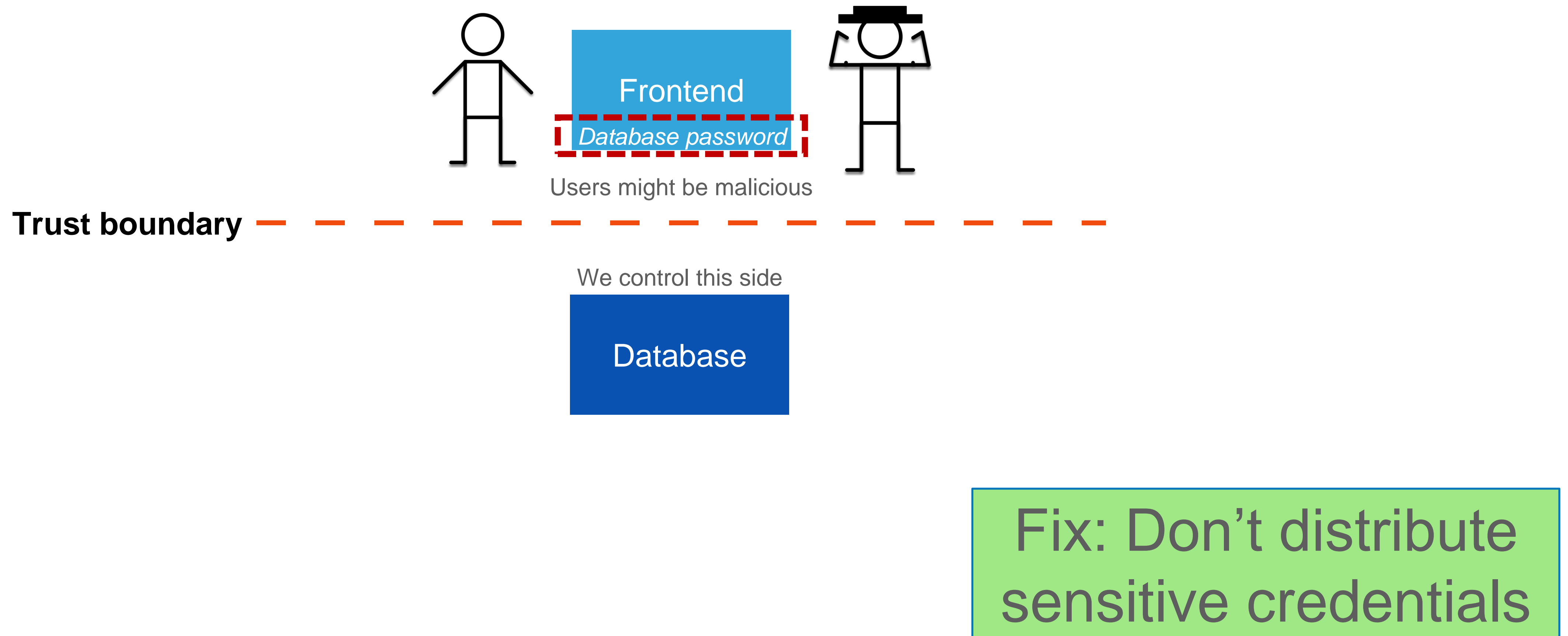
Authentication code in a web application



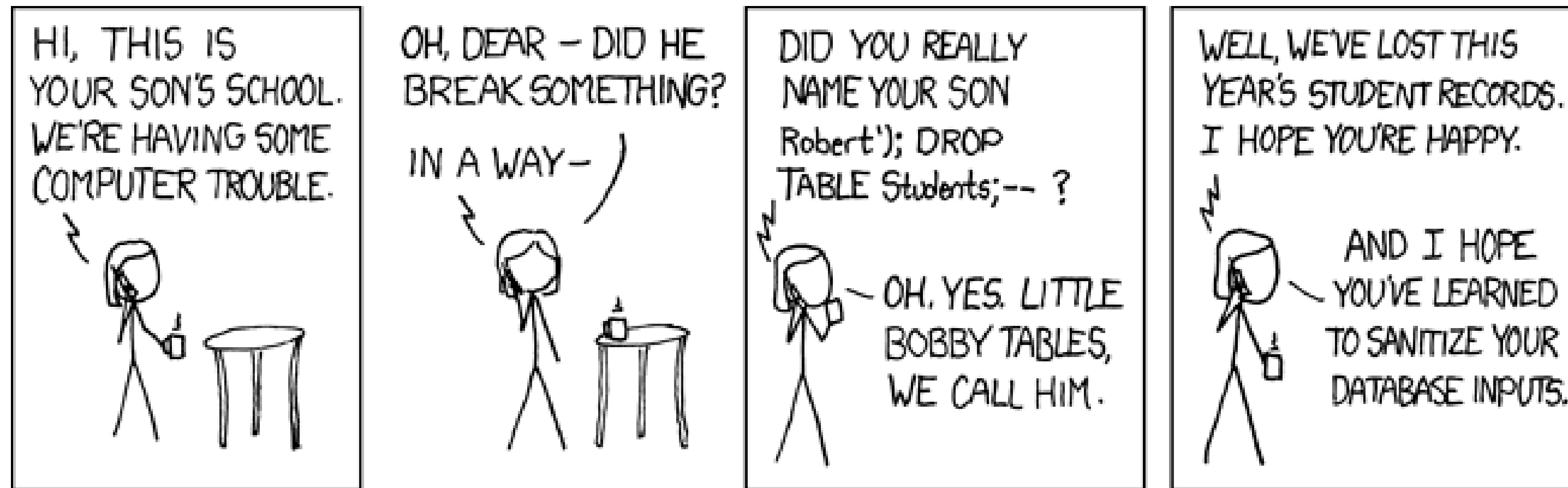
Fix: Move code to back end (duh!)

Threat Category 1: Code that runs in an untrusted environment

Access controls to database

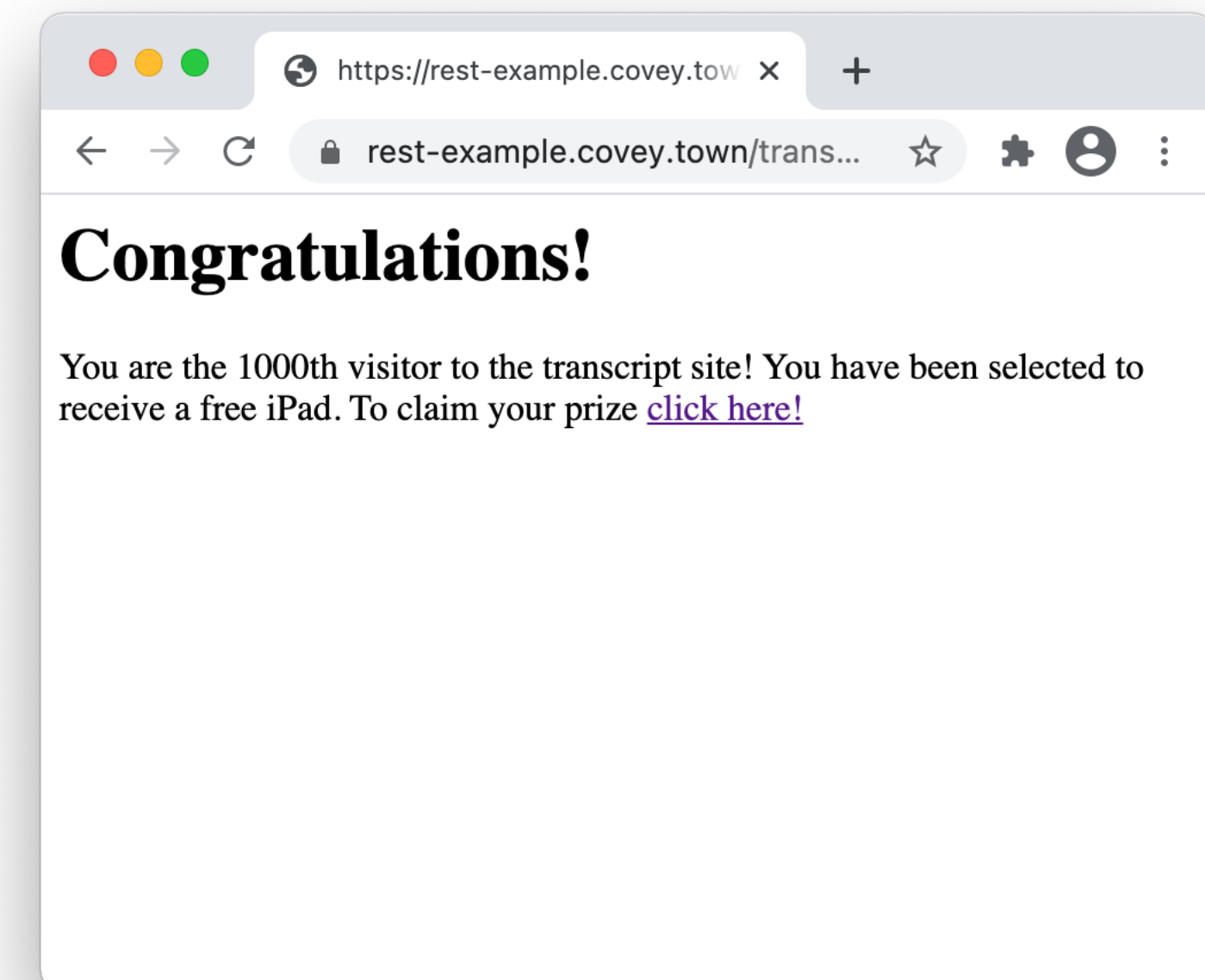
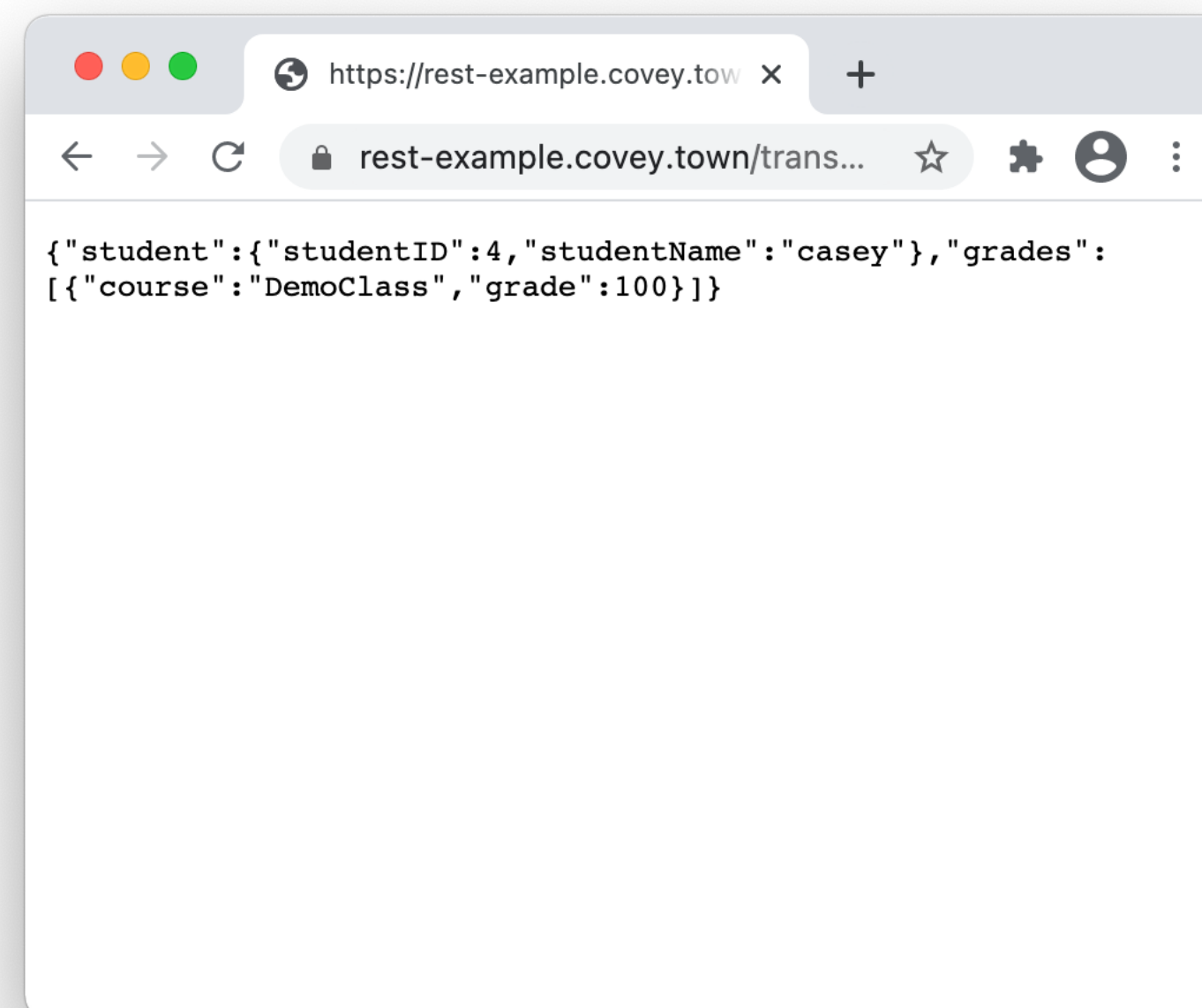


Threat 2: Data controlled by a user flowing into our trusted codebase



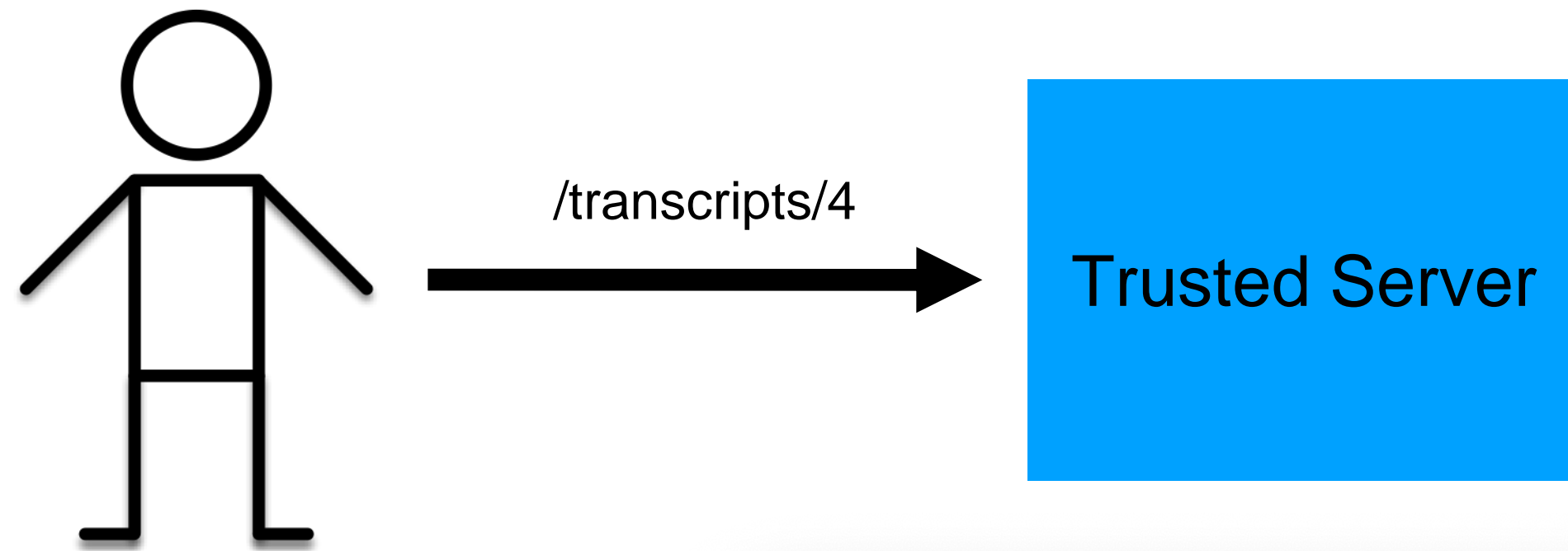
Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability

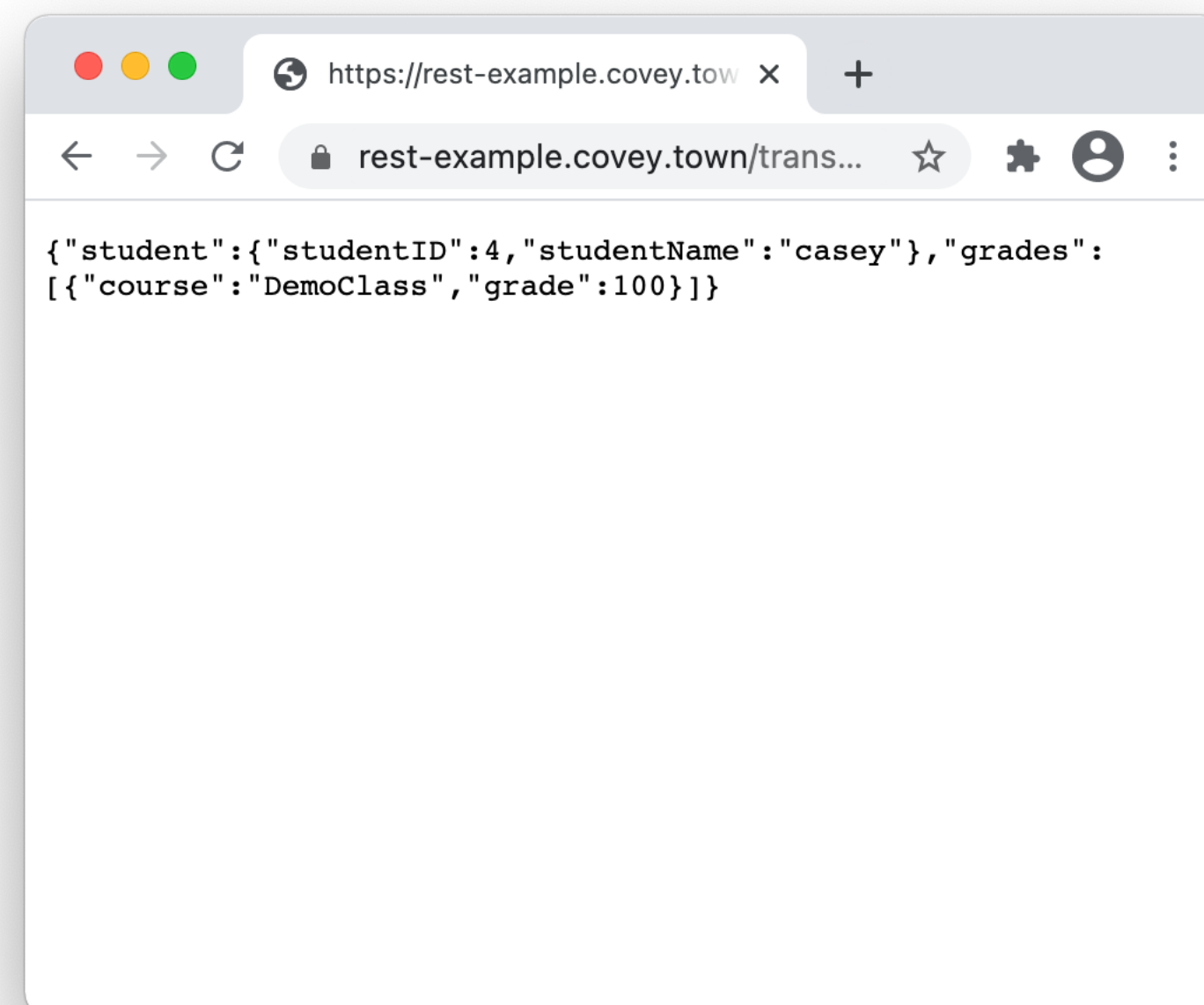


Threat 2: Data controlled by a user flowing into our trusted codebase

Cross-site scripting (XSS) vulnerability

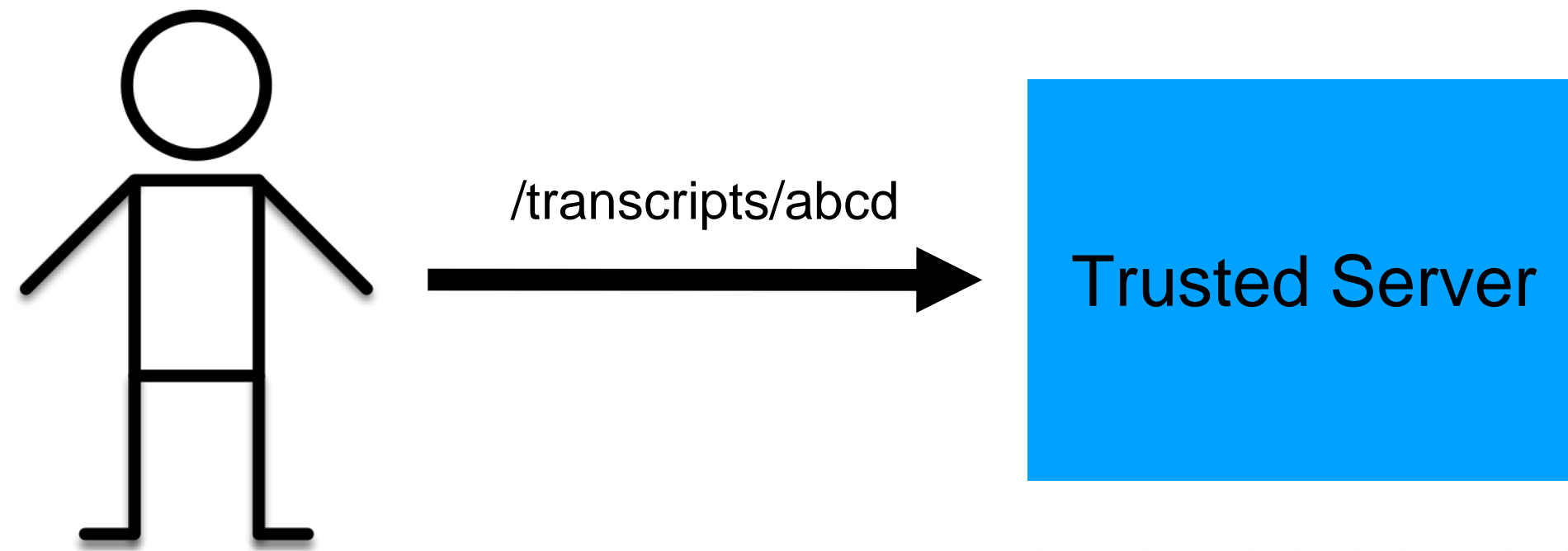


```
app.get('/transcripts/:id', (req, res) => {  
  // req.params to get components of the path  
  const {id} = req.params;  
  const theTranscript = db.getTranscript(parseInt(id));  
  if (theTranscript === undefined) {  
    res.status(404).send(`No student with id = ${id}`);  
  }  
  {  
    res.status(200).send(theTranscript);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

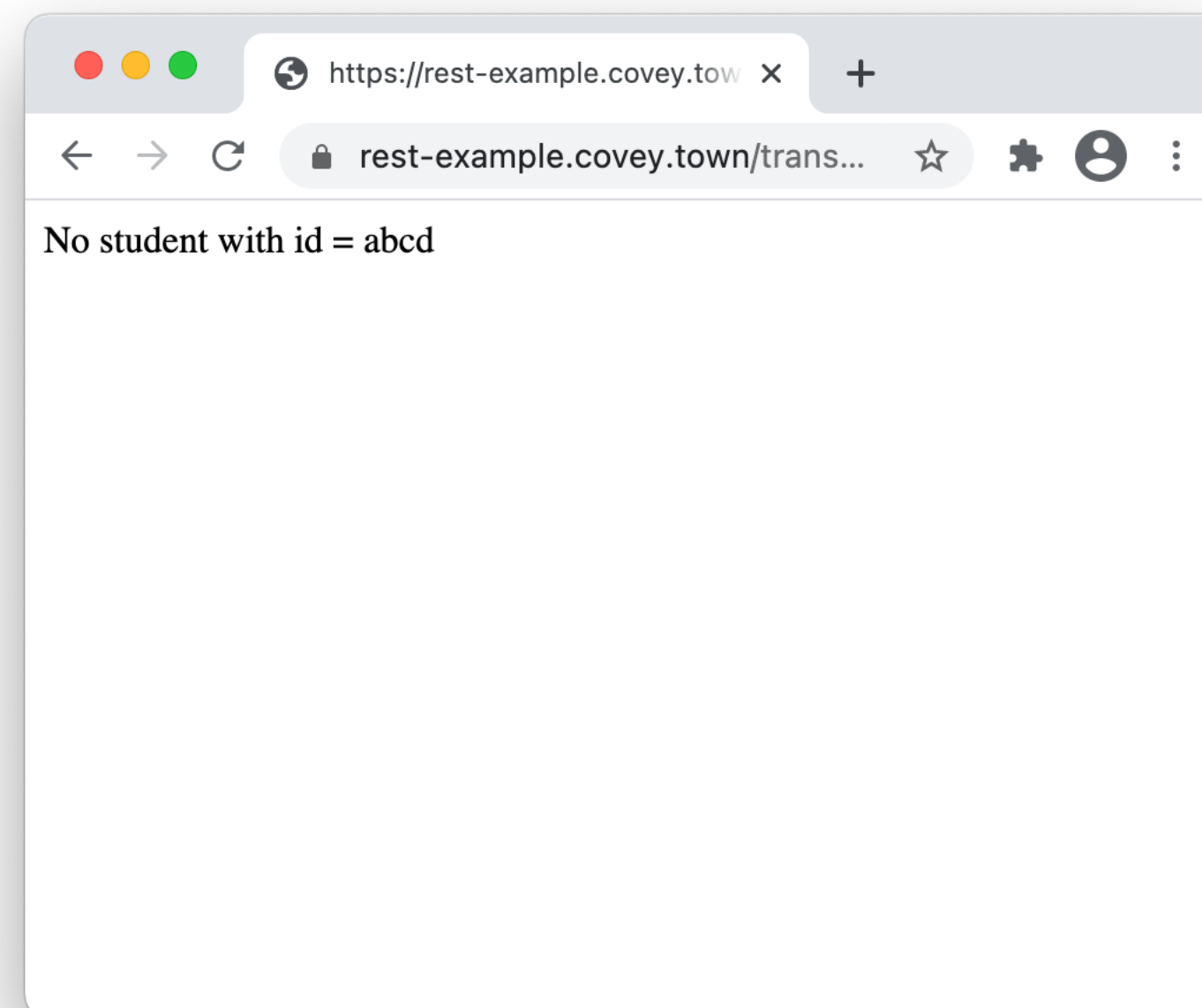


Threat 2: Data controlled by a user flowing into our trusted codebase

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);  
  }  
});
```

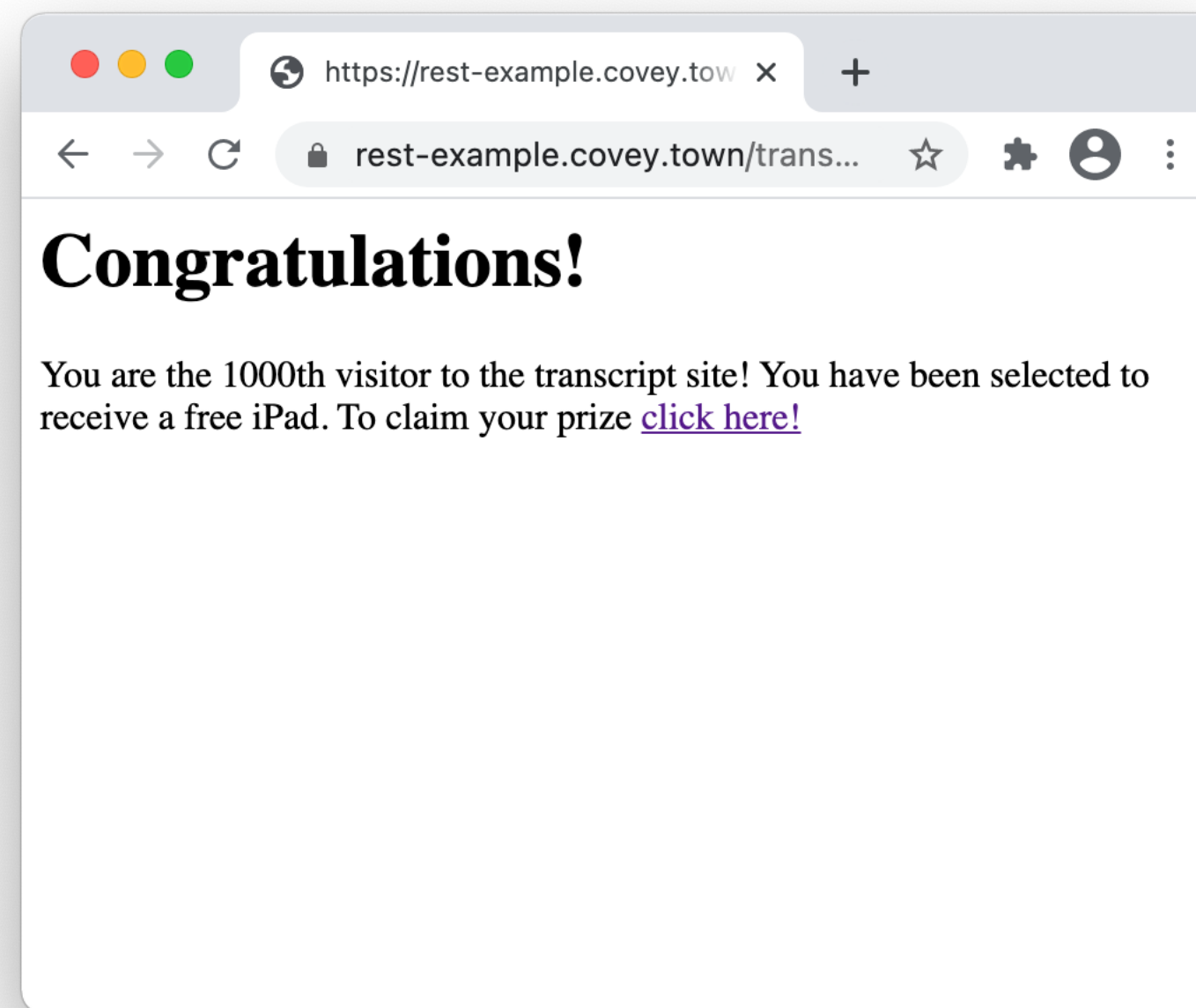
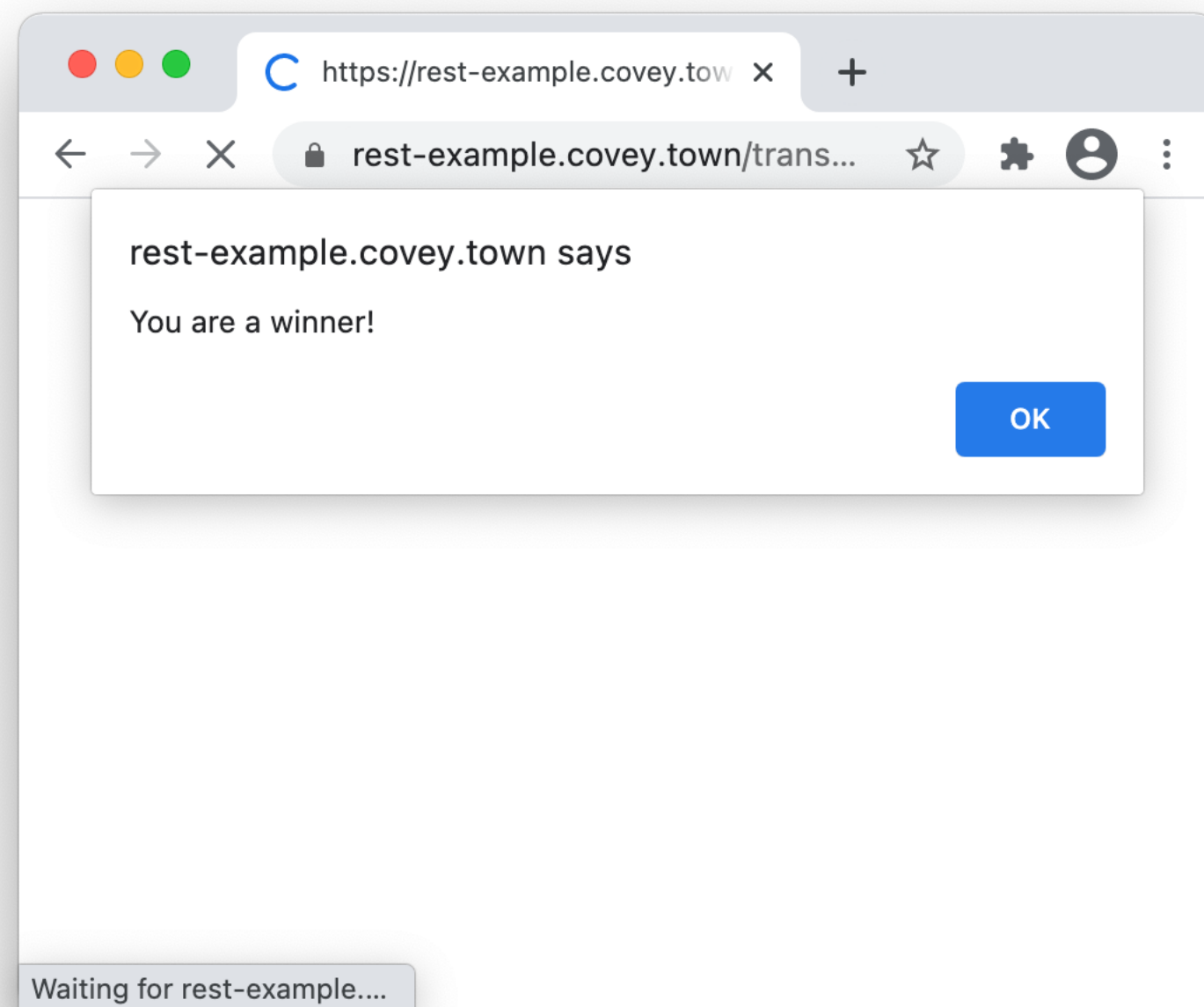


Threat 2: Data controlled by a user flowing into our trusted codebase

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=D  
LzxrzFCyOs'>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=D  
LzxrzFCyOs">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 dropdown 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 'NEWS' tag. A 'Need help? Contact Us' link is visible at the bottom of the banner. Social media icons for Facebook, Twitter, and Email are located 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

The Apache Software Foundation (ASF) announced that the vulnerability was **actively exploited** in the wild. From version 2.16.0 (along with security releases 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, Apache Log4j Java-based log4cxx, or other Apache Logging Services projects. execute malicious code a <https://nvd.nist.gov/vuln/detail/CVE-2021-44228> systems.

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

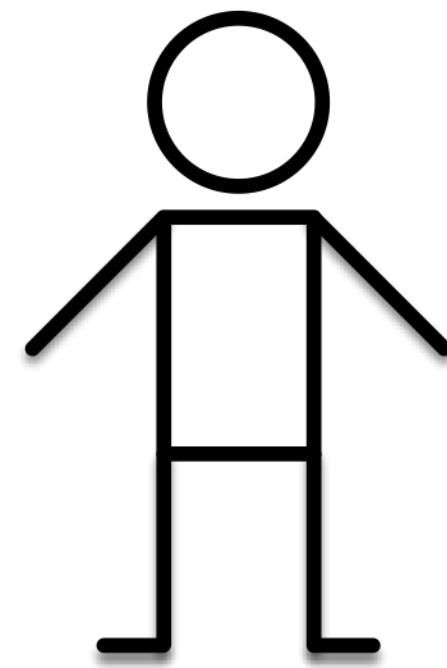
Mar 8, 2022

APT41 COMPROMISED SIX U.S. STATE GOVERNMENT NETWORKS

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”)

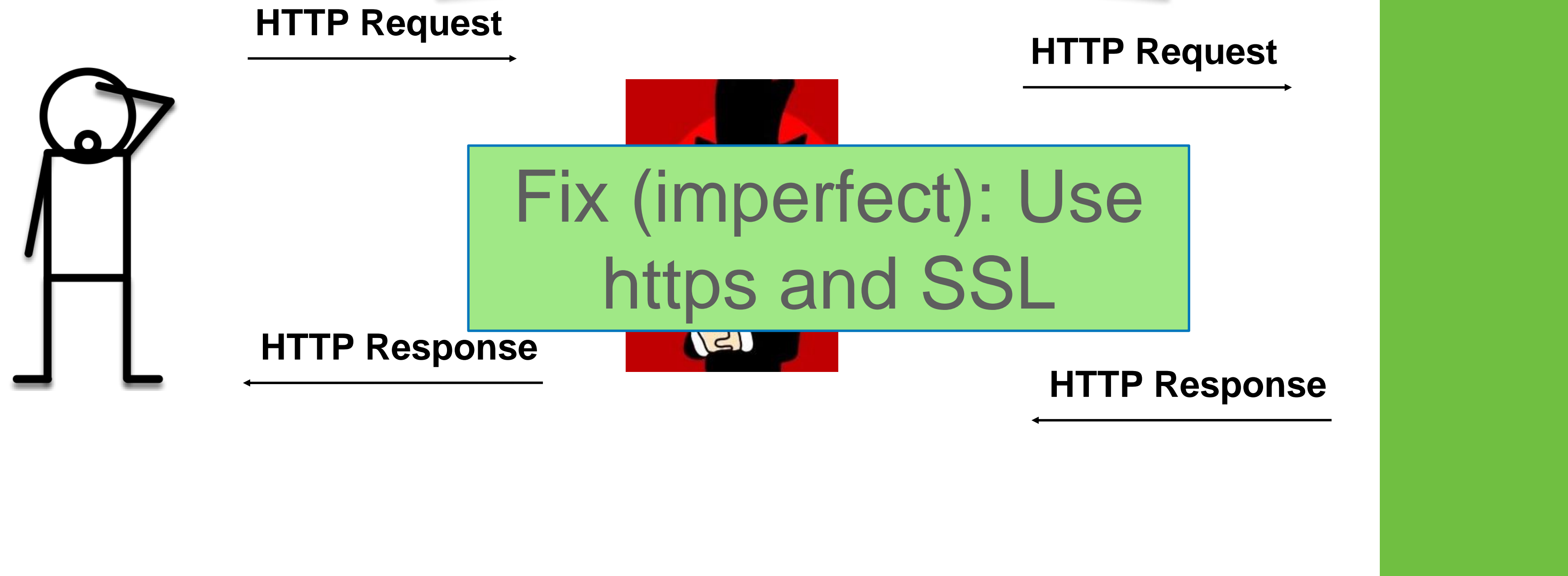
server

**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

Might be “man in the middle” that intercepts requests and impersonates user or server.



client page
(the “user”)

malicious actor
“black hat”

server

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

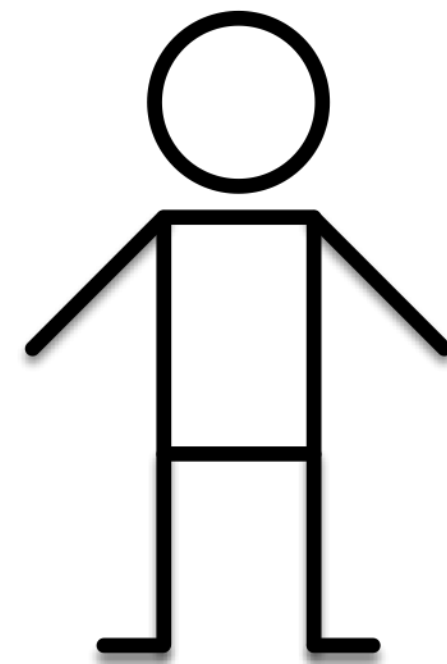
Do I trust that this request *really* came from the user?

Threat 3: Bad authentication

Preventing the man-in-the-middle with SSL



Preventing the man-in-the-middle with SSL



HTTP Request



Encrypted request

Curses! Foiled Again!

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
private key



amazon.com
public key



Some world proof that we are really amazon.com

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



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

Certificate Authority




CA private key



CA public key

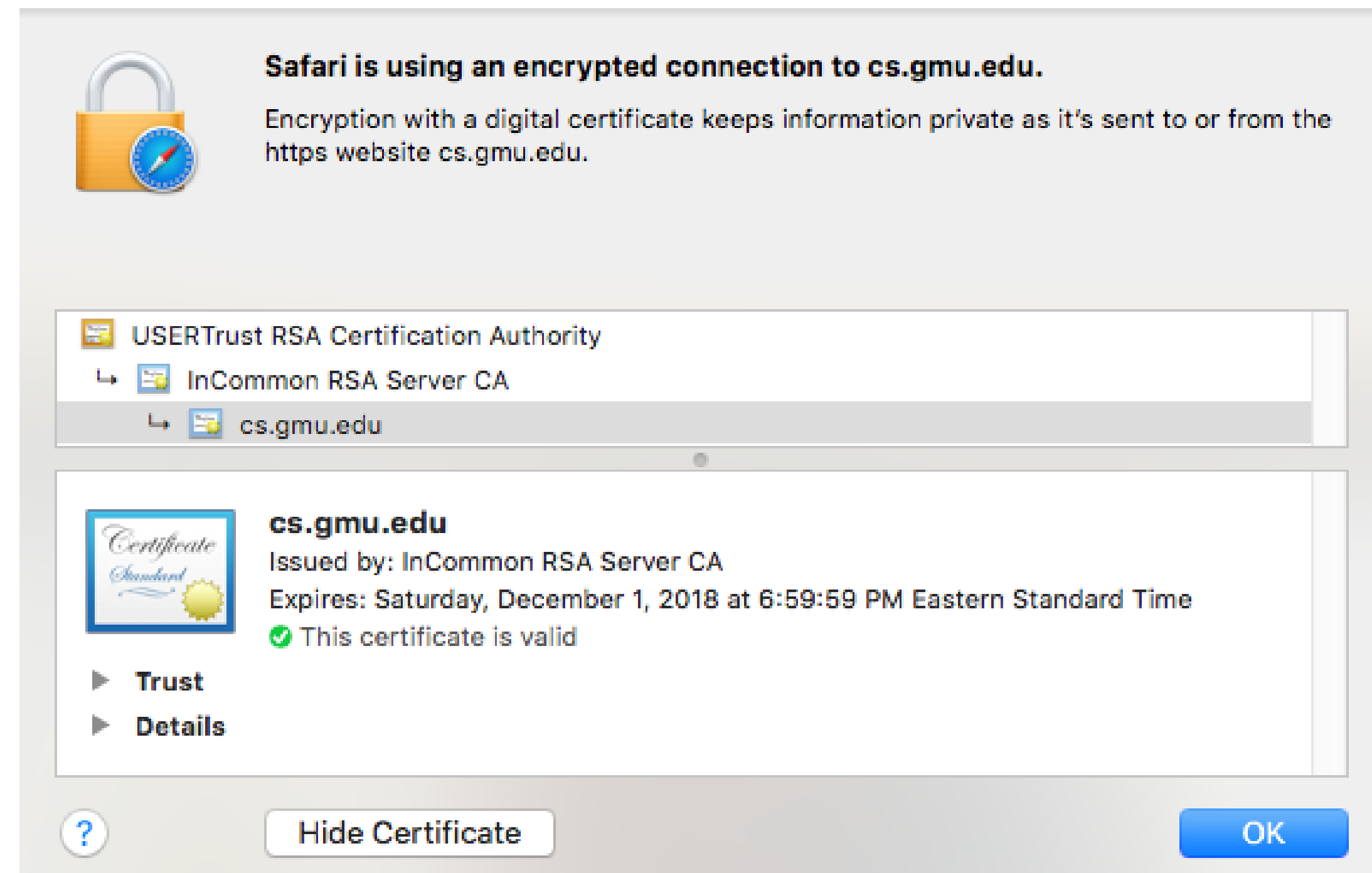
My Laptop



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, followed by the text "Let's Encrypt". To the right of the logo is a navigation menu with the following items: "Documentation", "Get Help", "Donate" (with a downward arrow), "About Us" (with a downward arrow), and "Languages" (with a flag icon and a downward arrow). Below the navigation menu is a large dark blue banner with a yellow and orange gradient background. In the center of the banner 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 buttons: "Get Started" and "Sponsor", both with blue outlines and blue text.

 **Let's Encrypt**

[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

.....

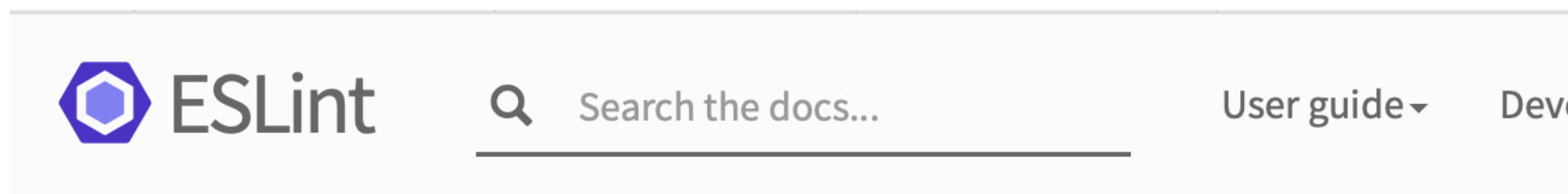
- ⚠ Your password must contain a minimum of 8 characters included with at least 1 upper case letter, 1 number, and 1 special character from !, #, \$, ^, and * (other special characters are not supported).

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

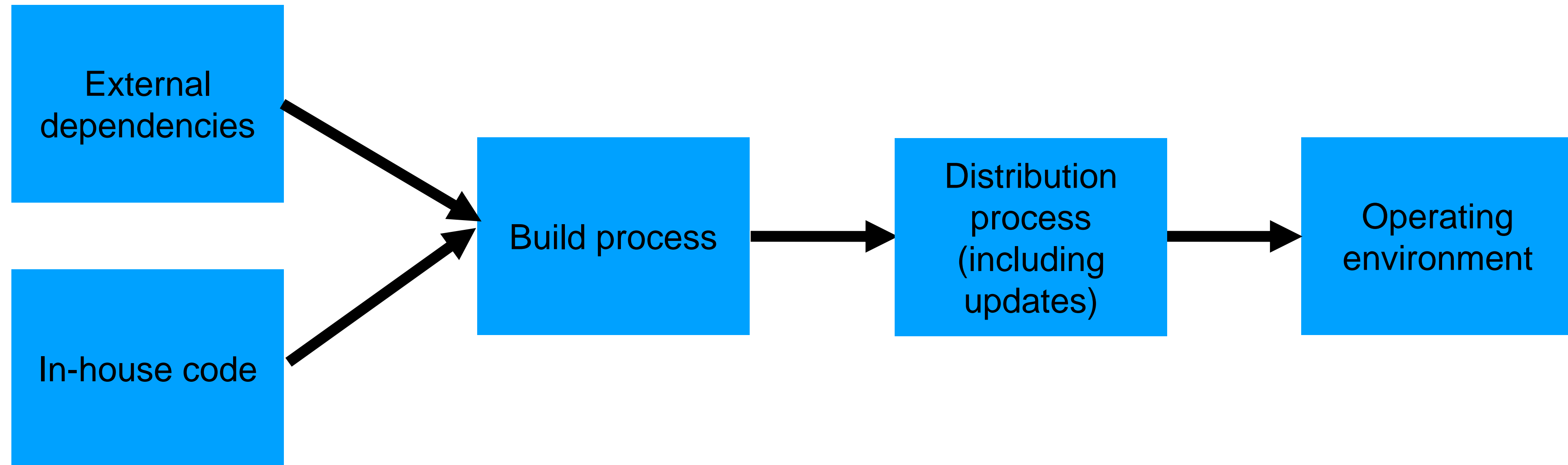
By Nilay Patel | @reckless | Jan 26, 2021, 9:13am EST



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



Common vulnerabilities in top 1% of npm packages

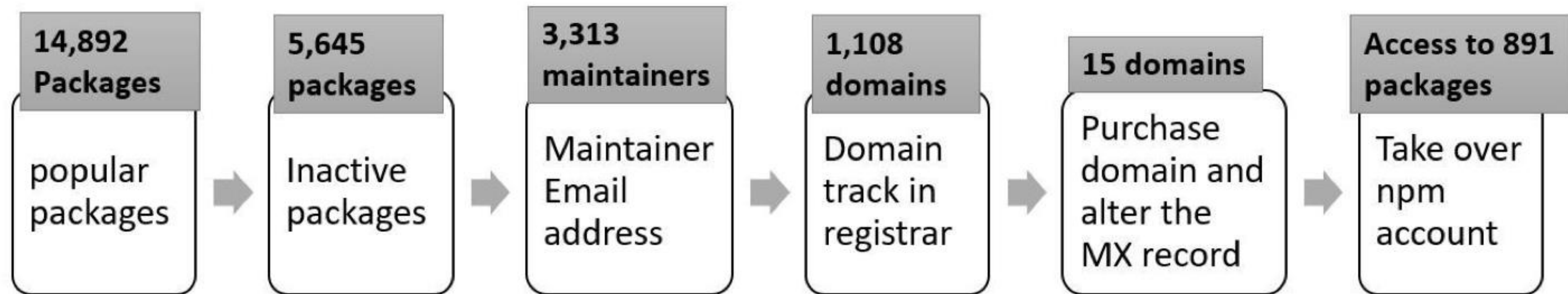
2021 NCSU/Microsoft Study

- Package inactive or deprecated, yet still in use
- No active maintainers
- At least one maintainer with an inactive (purchasable) email domain
- Too many maintainers or contributors to make effective maintenance or code control
- Maintainers are maintaining too many packages
- Many statistics/combinations: see the paper for details.

“What are Weak Links in the npm Supply Chain?” By: Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, Laurie Williams
<https://arxiv.org/abs/2112.10165>

A possible attack...

2021 NCSU/Microsoft Study



“What are Weak Links in the npm Supply Chain?” By: Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, Laurie Williams
<https://arxiv.org/abs/2112.10165>

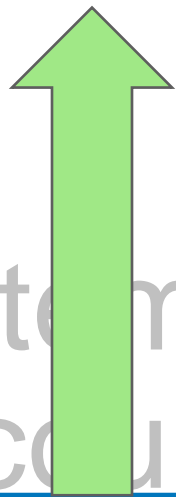
Threat Mitigation: Software Supply Chain

Process-based solutions for process-based problems

- External dependencies
 - Audit all dependencies and their updates before applying them
- In-house code
 - Require developers to sign code before committing, require 2FA for signing keys, rotate signing keys regularly
- Build process
 - Audit build software, use trusted compilers and build chains
- Distribution process
 - Sign all packages, protect signing keys
- Operating environment
 - Isolate applications in containers or VMs

Building a security architecture

- Security architecture is a set of mechanisms and policies that we build into our system to mitigate risks from threats
- Vulnerability: a characteristic or flaw in system design or implementation, or in the security procedures, that, if exploited, could result in a security compromise
- Threat: potential event that could compromise a security requirement
- Attack: realization of a threat



It's a management problem!!

Which threats to protect against, at what cost?

Consider various costs:

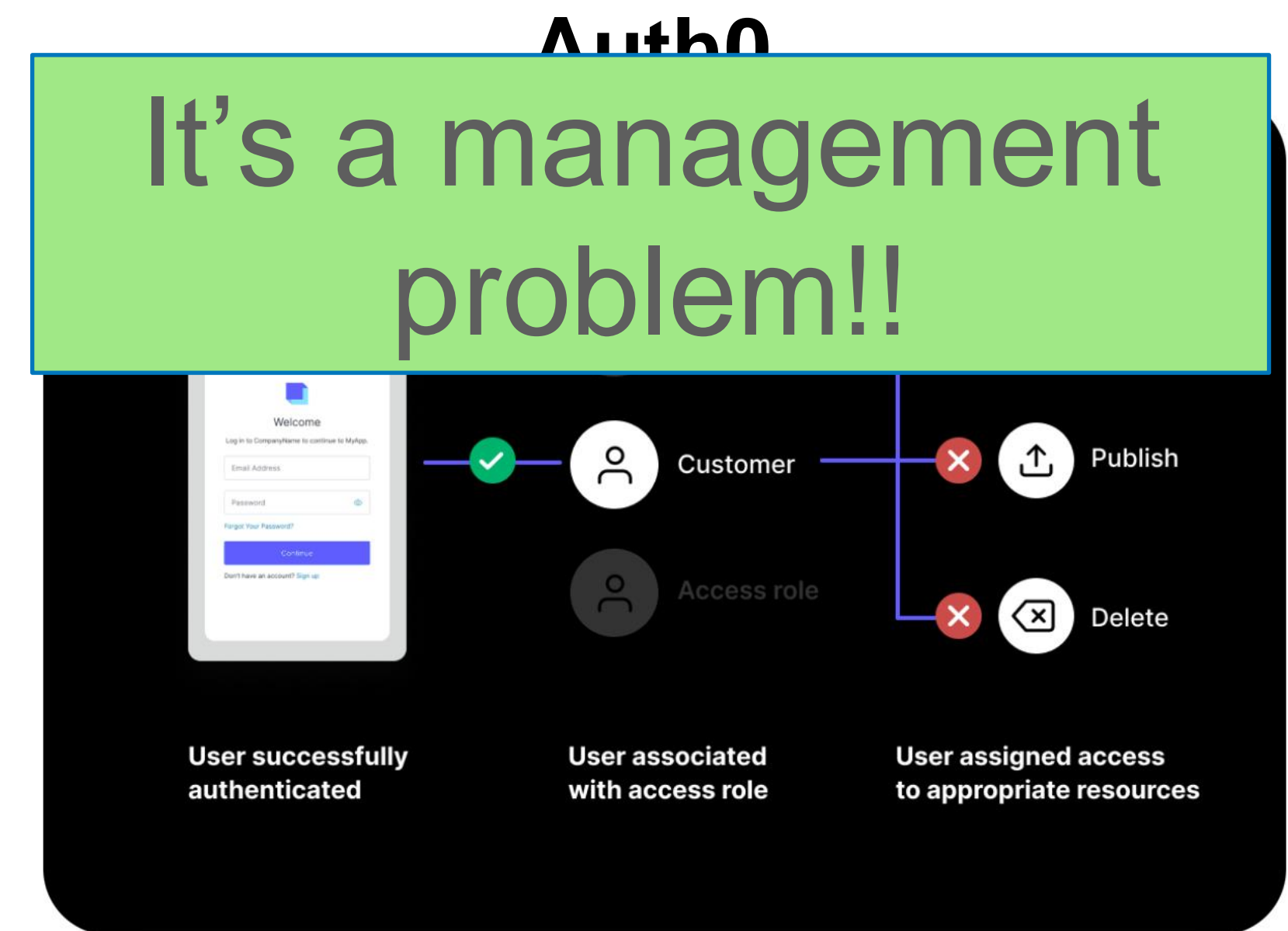
- Performance:
 - Encryption is not free;
 - C may be faster than Typescript, but is vulnerable to buffer overflows, etc.
- Expertise:
 - It is easy to try to implement these measures, it is hard to get them right
- Financial:
 - Implementing these measures takes time and resources

Broken Authentication + Access Control

OWASP #1

- Use SSL.
- Implement multi-factor authentication
- Implement weak-password checks
- Apply per-record access control
- Harden account creation, password reset pathways
- The software engineering approach: rely on a trusted component

But how to get your developers to do this?
Always.



<https://auth0.com>

Cryptographic Failures

OWASP #2

- Enforce encryption on all communication
- Validate SSL certificates; rotate certificate regularly
- Protect user-data at rest (passwords, credit card numbers, etc)
- Protect application “secrets” (e.g. signing keys)

But how to get your developers to do this.

Always.

It's a management problem!!

Table 5
token a

	kedIn	Titanium
4	1,914	
	1,783	
%	99.8%	

ial may consist of an ID

AKIA* Line filter (Ruby regex), optional 10 files per page Search

416 Files / 8.98 MB (ES took 0.131s)

Android Package	Path	Code Snippet
com.android.trigonometry	TrigonometryDefinition.java	8akIa8bJ/2m1pdLWYqTbNPFkeNN533CAvtug4dRlPdo5ZtckU/JF8RAVoi/HxG5E9jJ3skccxk75t0gUJr/sJX18nV+TxPMH81AgQ/Bk1B1FB+A4f4KyZtpkKp29+cVL8jIDAakJrkjjaKAAAAAAAAAAAAUk4u6RMY2ZQ6hoeHh5XP5zU0NKRXXnmFIwUQA4cPH9ahQ4fU3d1
com.android.ohiapp	ohiapp13.java	String str1 = work03(paramString, "", "AKIAJ1111111111111111", "ecs.amazonaws.jp", "AtxeExfJ7H1bQhDl4mc
com.amazonaws.sdk	AmazonScoreRegistry.java	protected AmazonSimpleDBClient sdbClient = new AmazonSimpleDBClient(new BasicAWSCredentials("AKIAJ1111111111111111"
com.amazonaws.sdk	signedRequestsHelper.java	private String awsAccessKeyId = "AKIAJ1111111111111111";
com.amazonaws.sdk	signedRequestsHelper.java	private String awsAccessKeyId = "AKIAJ1111111111111111";

← Previous 1 2 3 4 5 6 7 8 9 ... 41 42 Next →

Figure 9: PLAYDRONE's web interface to search decompiled sources showing Amazon Web Service tokens found in 130 ms. "A Measurement Study of Google Play," Viennot et al, SIGMETRICS '14

Do developers pay attention? Do they have good reason not to?

- Industrial study of secret detection tool in a large software services company with over 1,000 developers, operating for over 10 years
- What do developers do when they get warnings of secrets?
 - 49% remove the secrets; 51% bypass the warning
- Why do developers bypass warnings?
 - 44% report false positives, 6% are already exposed secrets, remaining are “development-related” reasons, e.g. “not a production credential” or “no significant security value”

Is it a management problem or a tool problem?

“Why secret detection tools are not enough: It’s not just about false positives - An industrial case study”

Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey & Laurie Williams

<https://link.springer.com/article/10.1007/s10664-021-10109-y>

Code Injection

OWASP #3

- **Sanitize** user-controlled inputs (remove HTML)
- Use tools like LGTM to detect vulnerable data flows (insert into commit workflow?)
- Use middleware that side-steps the problem (e.g. return data as JSON, client puts that data into React component) (how to get engineers to actually do this?)

1 path available

Reflected cross-site scripting

2 steps in server.ts

Step 1 source

Source root/src/server/server.ts

```
↑ 1-61
62 app.get('/transcripts/:id', (req, res) => {
63   // req.params to get components of the path
64   const {id} = req.params;
65   console.log(`Handling GET /transcripts/:id id = ${id}`);
66   const theTranscript = db.getTranscript(parseInt(id));
↓ 67-169
```

Step 2 sink

Source root/src/server/server.ts

```
↑ 1-65
66   const theTranscript = db.getTranscript(parseInt(id));
67   if (theTranscript === undefined) {
68     res.status(404).send(`No student with id = ${id}`);
69   } else {
70     res.status(200).send(theTranscript);
↓ 71-169
```

Cross-site scripting vulnerability due to user-provided value.



Detecting Weaknesses in Apps with Static Analysis

LGTM + CodeQL

The screenshot shows the LGTM web interface. At the top, there's a navigation bar with 'lgTM' logo, a search icon, and links for 'Help', 'Query console', 'Project lists', 'My alerts', and a user profile for 'Jonathan Bell'. Below this is a secondary navigation bar with 'Alerts 16', 'Logs', 'Files', 'History', 'Compare', 'Integrations', and 'Queries'. A tooltip explains that files are listed based on their presence in the Alerts tab. The 'Alert filters' section includes a dropdown for 'No filter selected', a save icon, and filters for 'Severity', 'Query', and 'Tag'. There's also a checkbox for 'Show excluded files' which is checked. Below the filters is a 'Source root/' field. At the bottom, a table lists files with their alert counts and lines of code.

Name	Alerts	Lines of code
public	0	0
src	16	756
package.json	0	0

But tools have both false positives and false negatives

- Text storage of sensitive information**
Sensitive information stored without encryption or hashing can expose it to an attacker.
- Text logging of sensitive information**
Sensitive information without encryption or hashing can expose it to an attacker.
- Unsafe cross-site scripting**
User input directly to the DOM allows for a cross-site scripting vulnerability.
- Unsafe URL redirect**
Client-side URL redirection based on unvalidated user input may cause redirection to malicious web sites.
- Code injection**
Interpreting unsanitized user input as code allows a malicious user arbitrary code execution.
- Download of sensitive file through insecure connection**
Downloading executables and other sensitive files over an insecure connection opens up for potential man-in-the-middle attacks.

<https://lgtm.com>

Weakly Protected Sensitive Data

OWASP #4

- Classify your data by sensitivity
- Encrypt sensitive data - in transit and at rest
- Make a plan for data controls, stick to it
- Software engineering fix: can we avoid storing sensitive data?
 - Payment processors: Stripe, Square, etc

Learning Objectives for this Lesson

By the end of this lesson, you should be able to...

- Define key terms relating to software/system security
- Describe some of the tradeoffs between security and other requirements in software engineering
- Explain 5 common vulnerabilities in web applications and similar software systems, and describe some common mitigations for each of them.
- Explain why software alone isn't enough to assure security