

# **CS 4530**

# **Software Engineering**

## **Lecture 10: Software Engineering & Security**

**Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, 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...**

- Describe that security is a spectrum, and be able to define a realistic threat model for a given system
- Evaluate the tradeoffs between security and costs in software engineering
- Recognize the causes of and common mitigations for common vulnerabilities in web applications
- Utilize static analysis tools to identify common weaknesses in code

# **Security as non-functional requirements**

## **CIA: An overview of security properties**

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

# 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

## Vocabulary

- Security architecture is a set of mechanisms and policies that we build into our system to mitigate risks from threats
- Threat: potential event that could compromise a security requirement
- Attack: realization of a threat
- 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: Code that runs in an untrusted environment

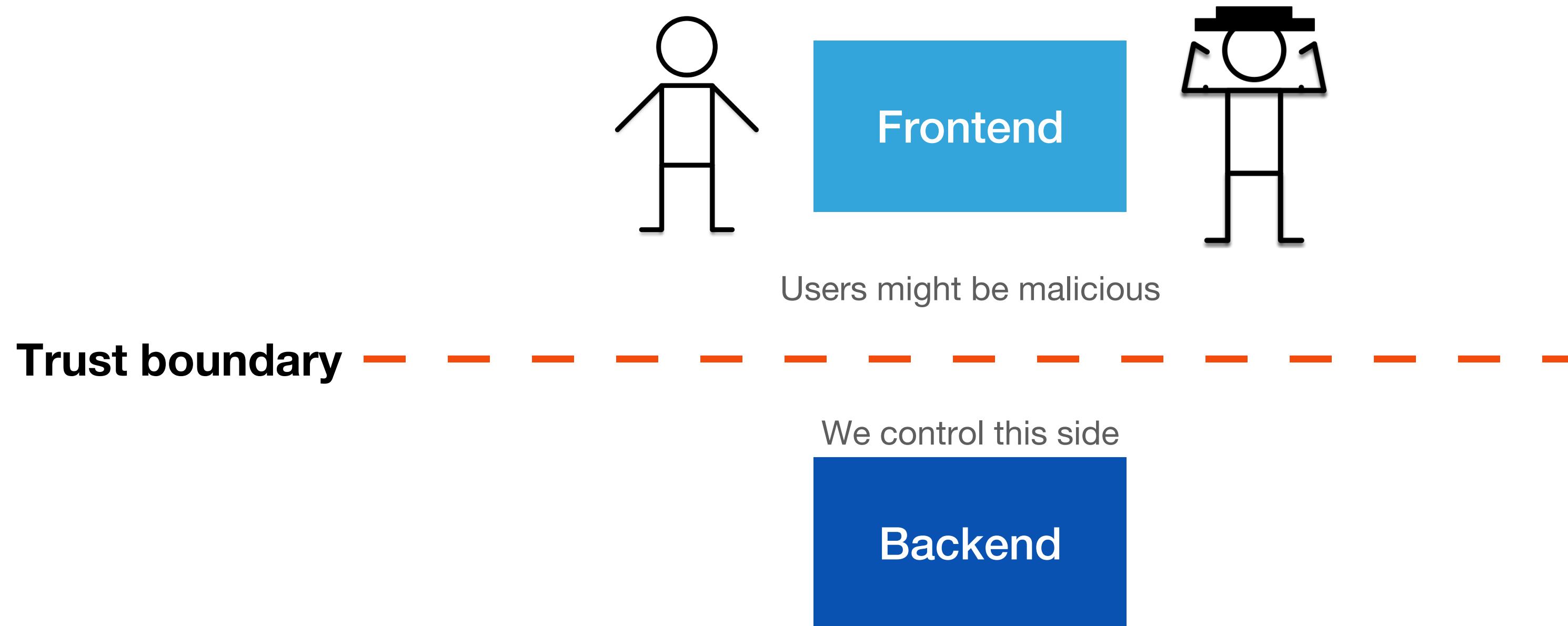
## Authentication code in a web application

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

Should this go in our frontend code?

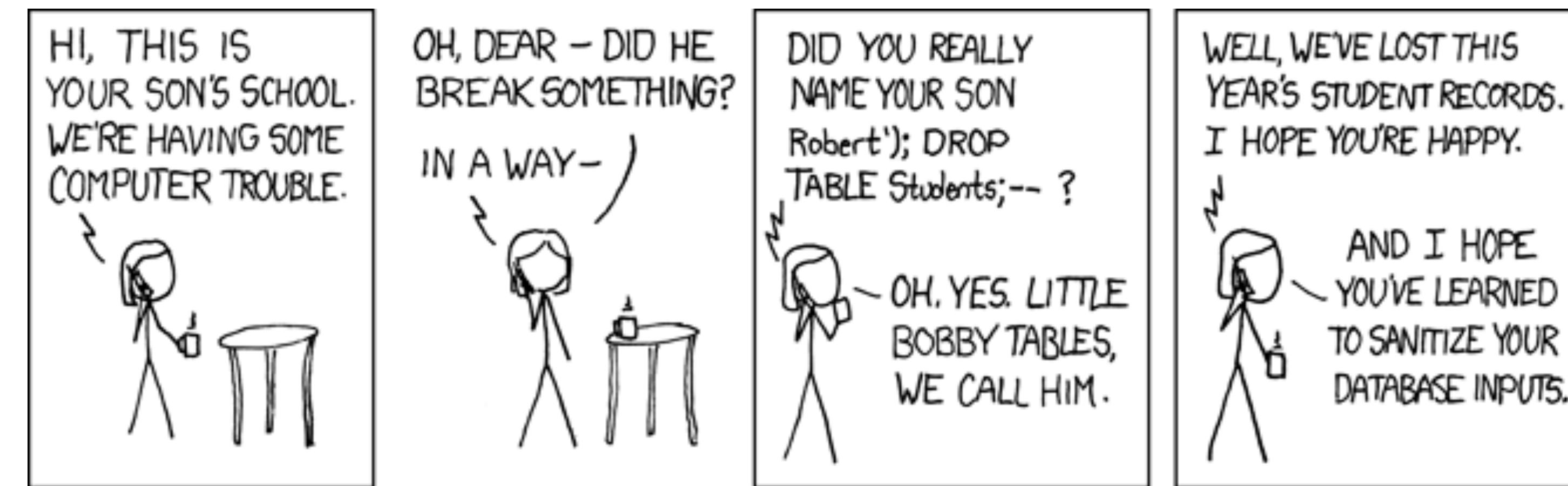
# Threat: Code that runs in an untrusted environment

## Authentication code in a web application



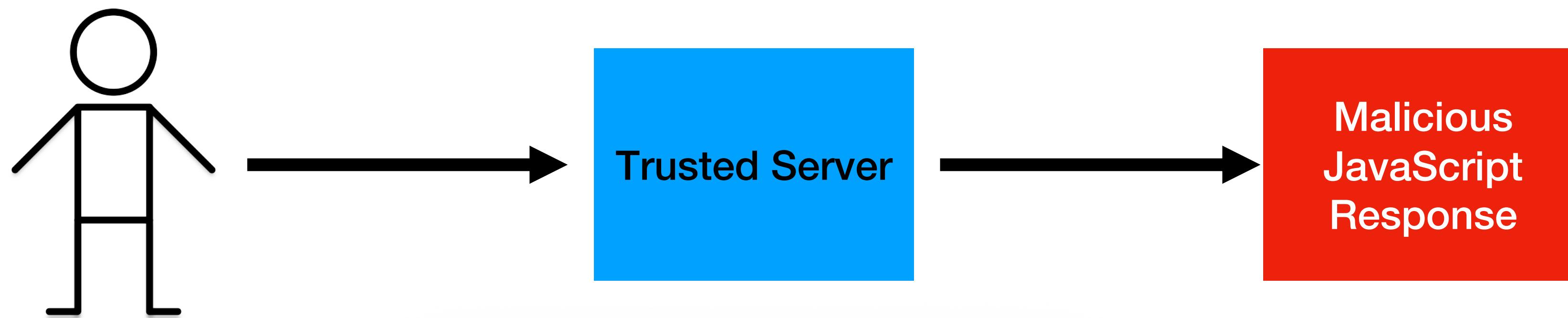
```
function checkPassword(inputPassword: string){  
  if(inputPassword === 'letmein') {  
    return true;  
  }  
  return false;  
}
```

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



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

## Cross-site scripting (XSS) vulnerability



The diagram shows two side-by-side browser windows. The left window displays a JSON response from a REST API endpoint:

```
{"student": {"studentID": 4, "studentName": "casey"}, "grades": [{"course": "DemoClass", "grade": 100}]}
```

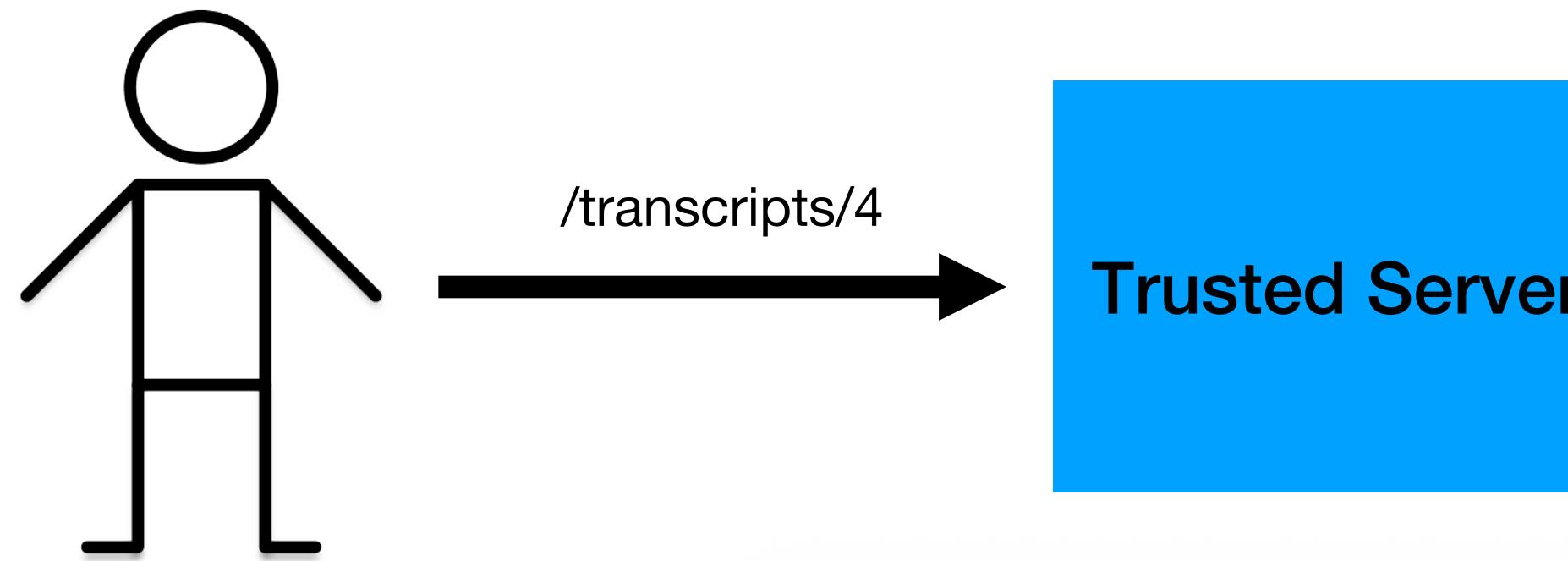
The right window shows a malicious JavaScript response:

**Congratulations!**

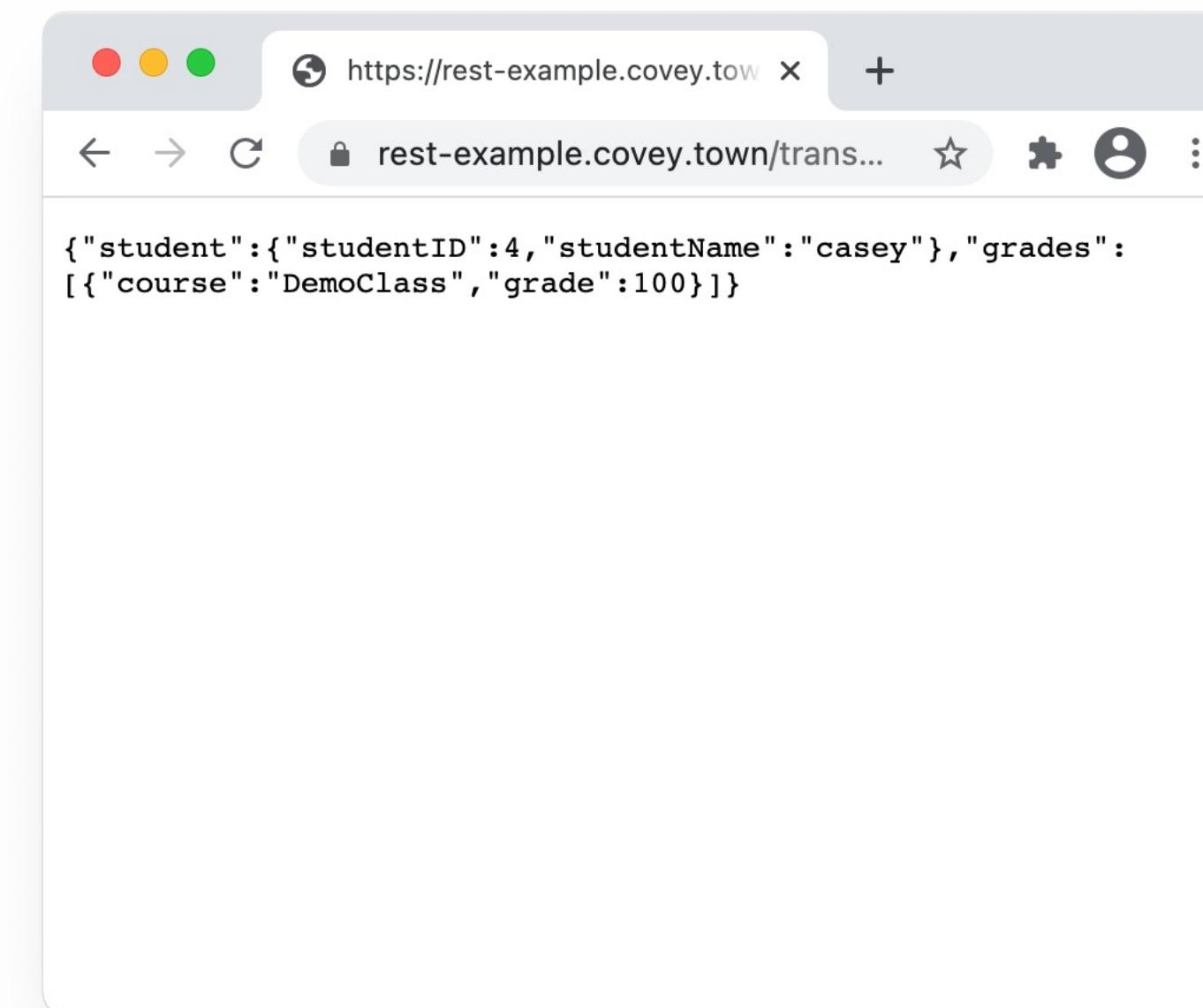
You are the 1000th visitor to the transcript site! You have been selected to receive a free iPad. To claim your prize [click here!](#)

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

## Cross-site scripting (XSS) vulnerability

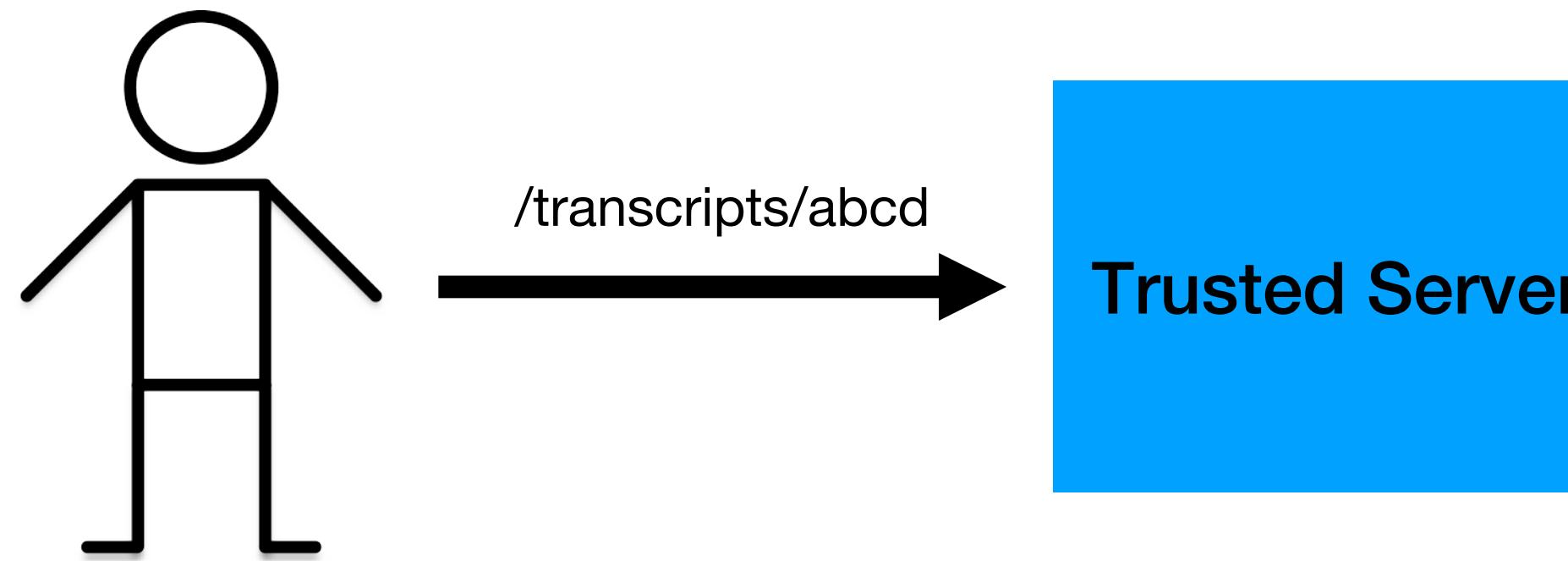


```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  res.status(200).send(theTranscript);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

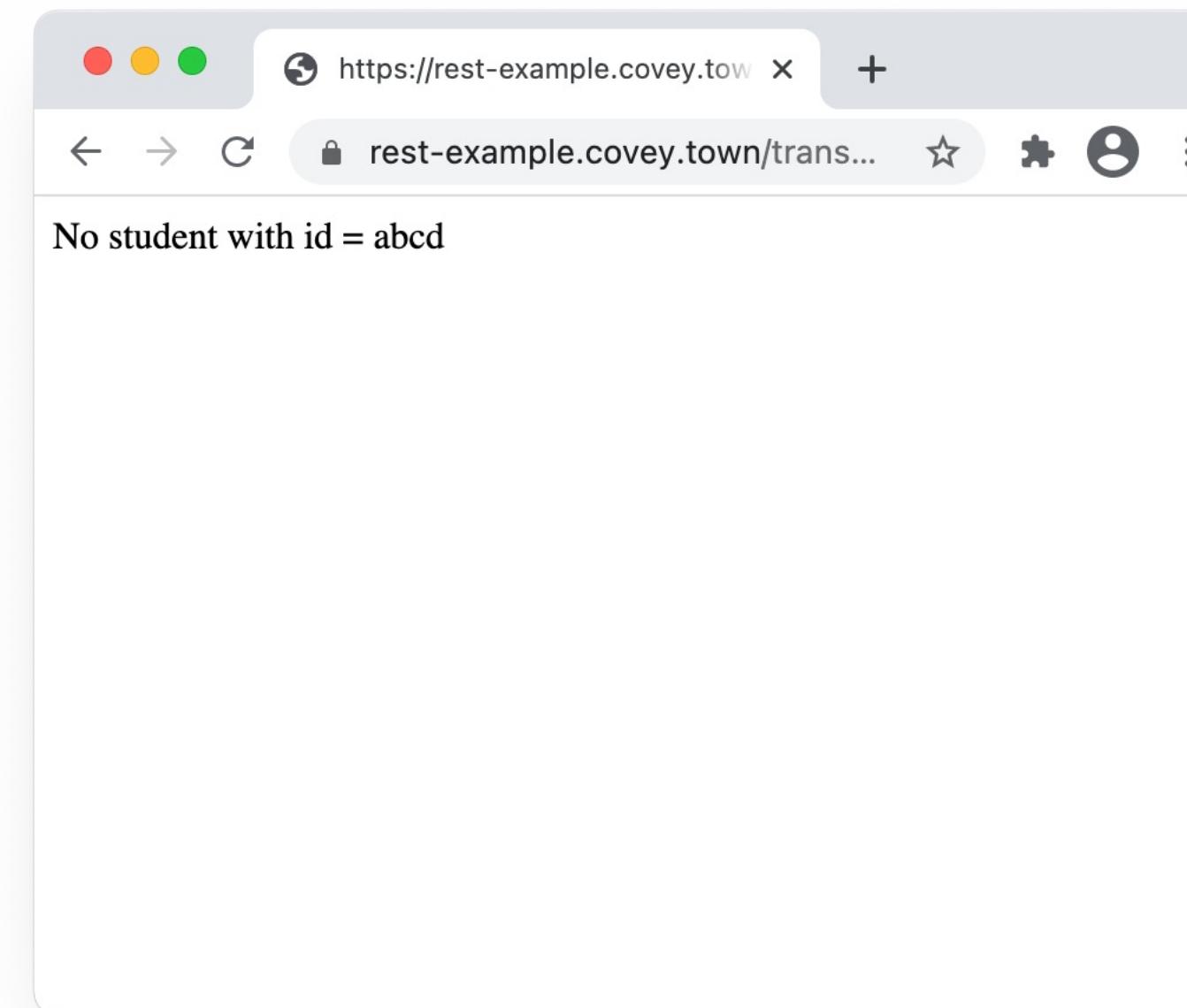


# Threat: Data controlled by a user flowing into our trusted codebase

## Cross-site scripting (XSS) vulnerability

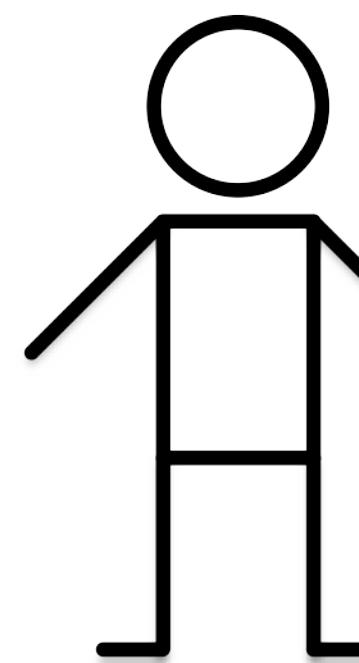


```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  res.status(200).send(theTranscript);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

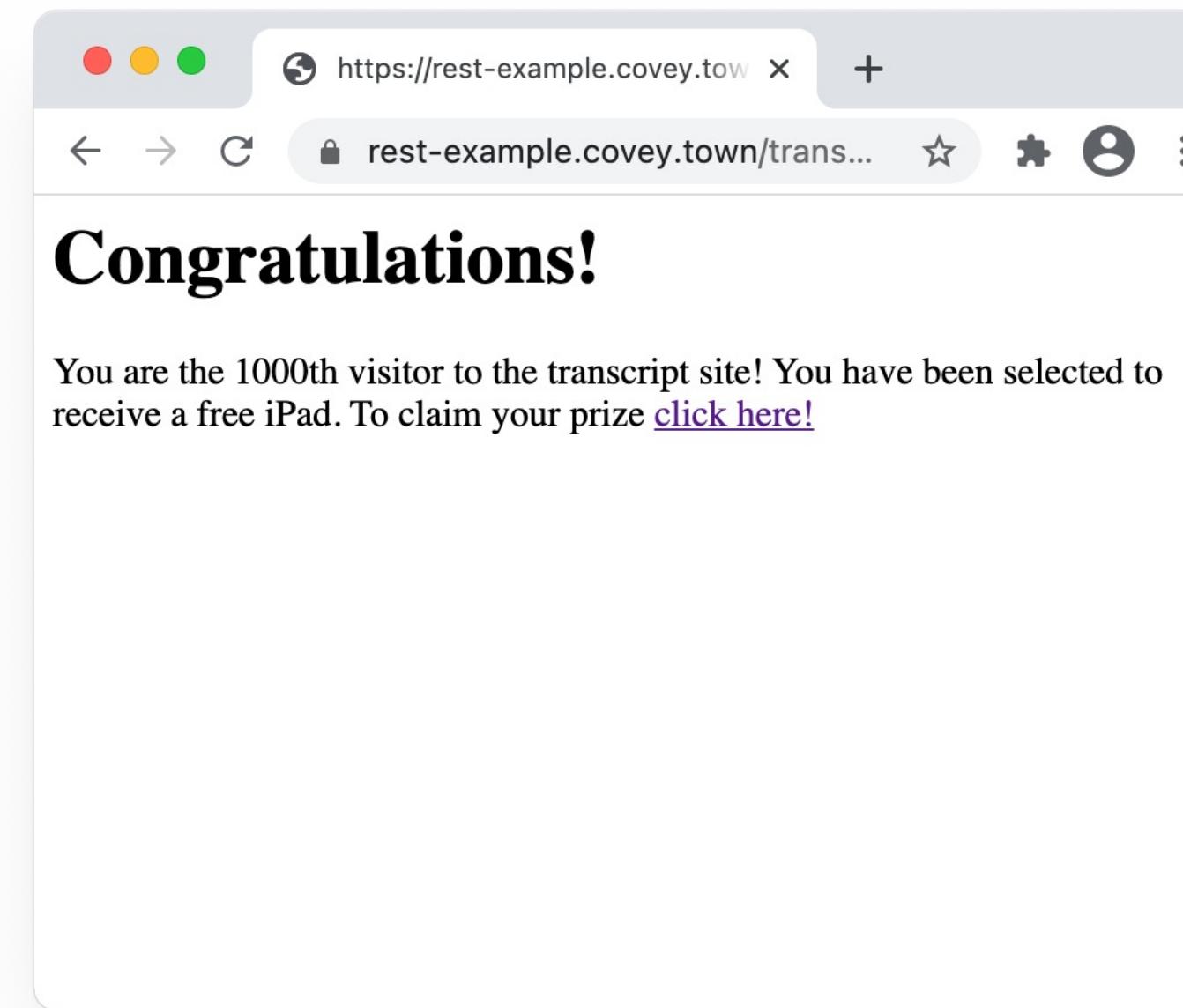
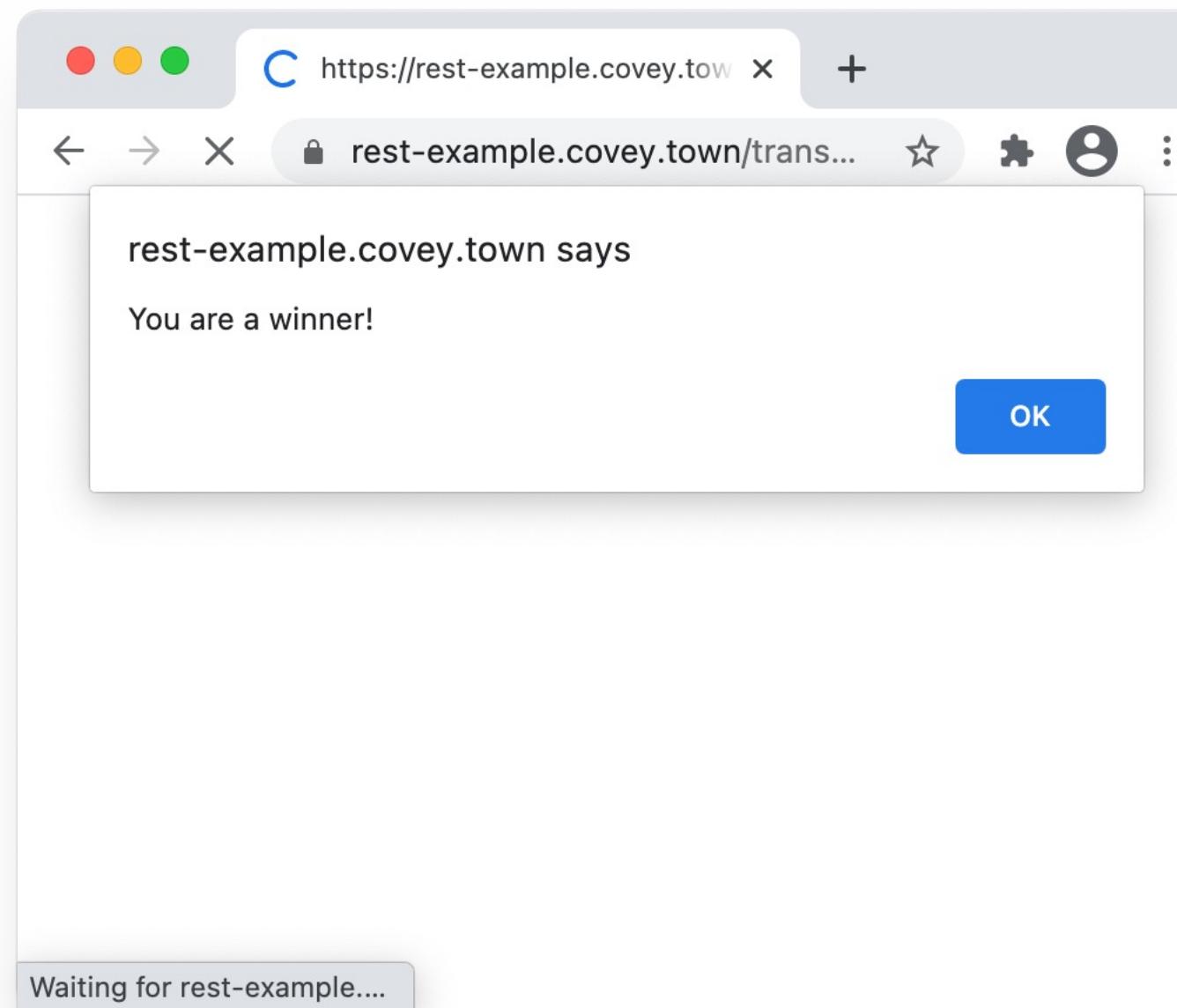
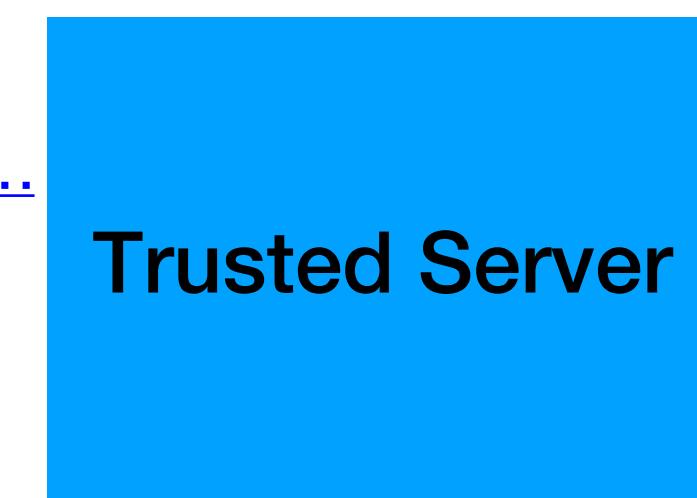


# Threat: Data controlled by a user flowing into our trusted codebase

## Cross-site scripting (XSS) vulnerability



/transcripts/%3Ch1%3e...



```
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}`);
  } else {
    res.status(200).send(`

<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: Data controlled by a user flowing into our trusted codebase

## Java code injection vulnerability in Apache Struts (@Equifax)

The screenshot shows the Equifax website with a red header. The header includes the Equifax logo, language selection (English), and a link to "Return to equifax.com". Below the header, there's a large white banner with the text "2017 Cybersecurity Incident & Important Consumer Information". On the right side of the banner, there's a news article snippet with the title "Equifax Says Cybersecurity Breach Has Cost \$1.4 Billion". At the bottom of the banner, there's a call-to-action button with the text "Need help? [Contact Us](#)". The main content area below the banner is white and features the heading "CVE-2017-5638 Detail" and a section titled "Current Description".

**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.

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: 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.

The Apache Software Foundation actively exploited zero-day vulnerabilities in Log4j2. 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. To execute malicious code a system, visit <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>.

Mar 8, 2022

APT41 COMPROMISED  
SIX U.S. STATE  
GOVERNMENT

# Threat: 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](https://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](https://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>

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

f SHARE

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 systems to breach even

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

# 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?

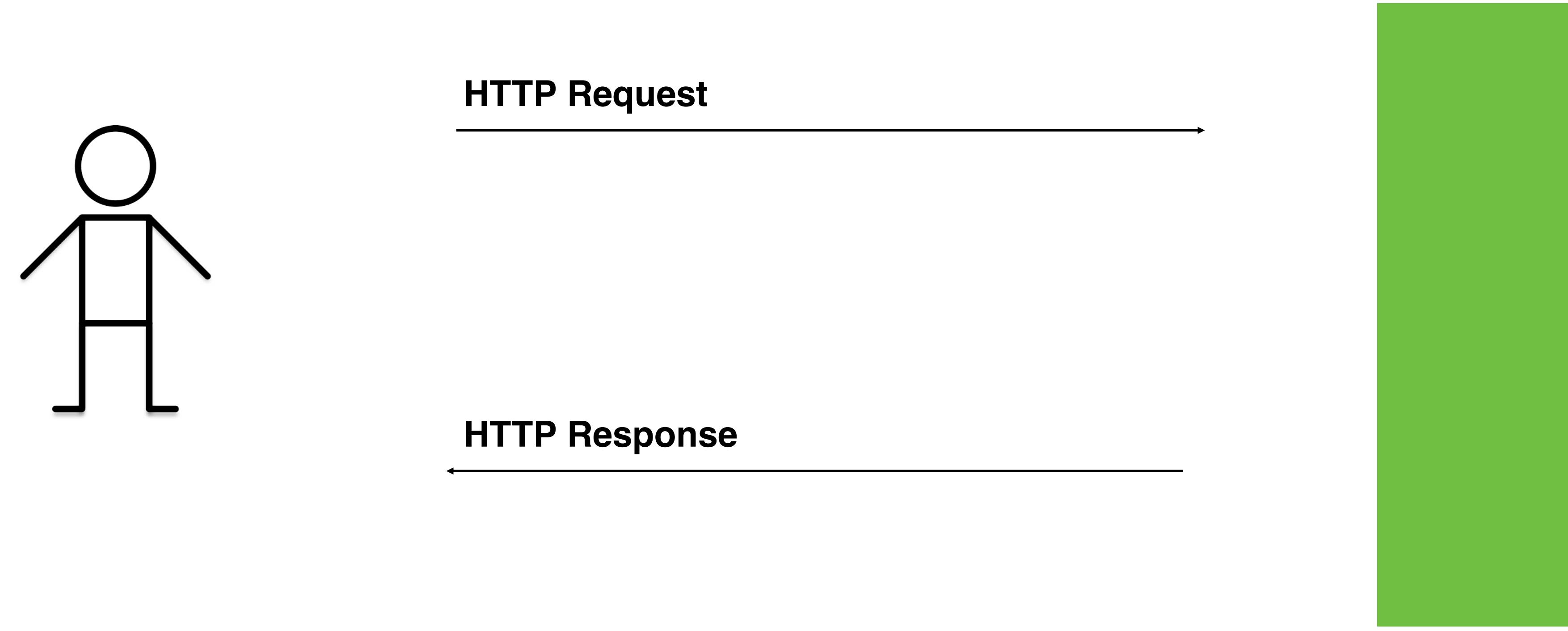
# Threat Models capture these tradeoffs

- What is being defended?
  - What resources are important to defend?
  - What malicious actors exist and what attacks might they employ?
- Who do we trust?
  - What entities or parts of system can be considered secure and trusted
  - Have to trust **something!**
  - Never trust remote users (especially remote users!)

# Mitigating security threats in software engineering

- For these threats:
  - Threat category: Code that runs in an untrusted environment
  - Threat category: Inputs that are controlled by an untrusted user
  - Threat category: Software supply chain
- Recurring theme: No silver bullet

# Threat Mitigation: Trusted Code



client page  
(the “user”)

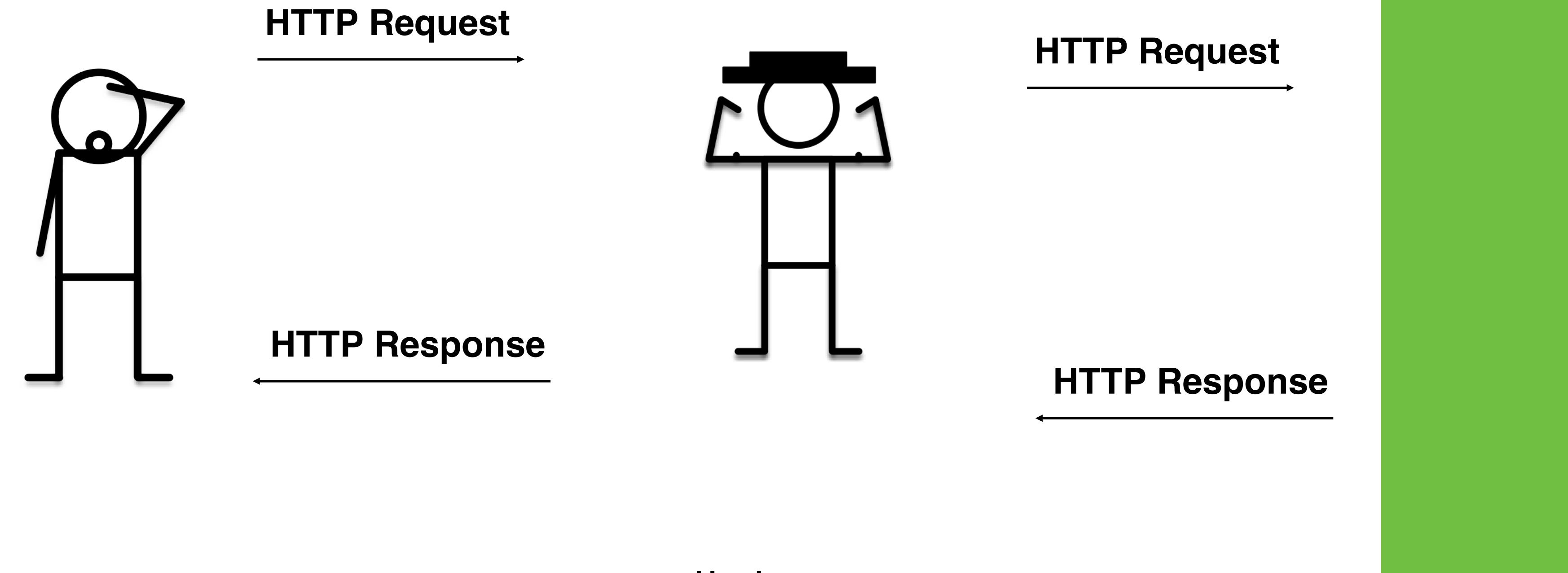
**Do I trust that this response  
really came from the server?**

server

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

# Threat Mitigation

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



client page  
(the “user”)

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

malicious actor  
“black hat”

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

server

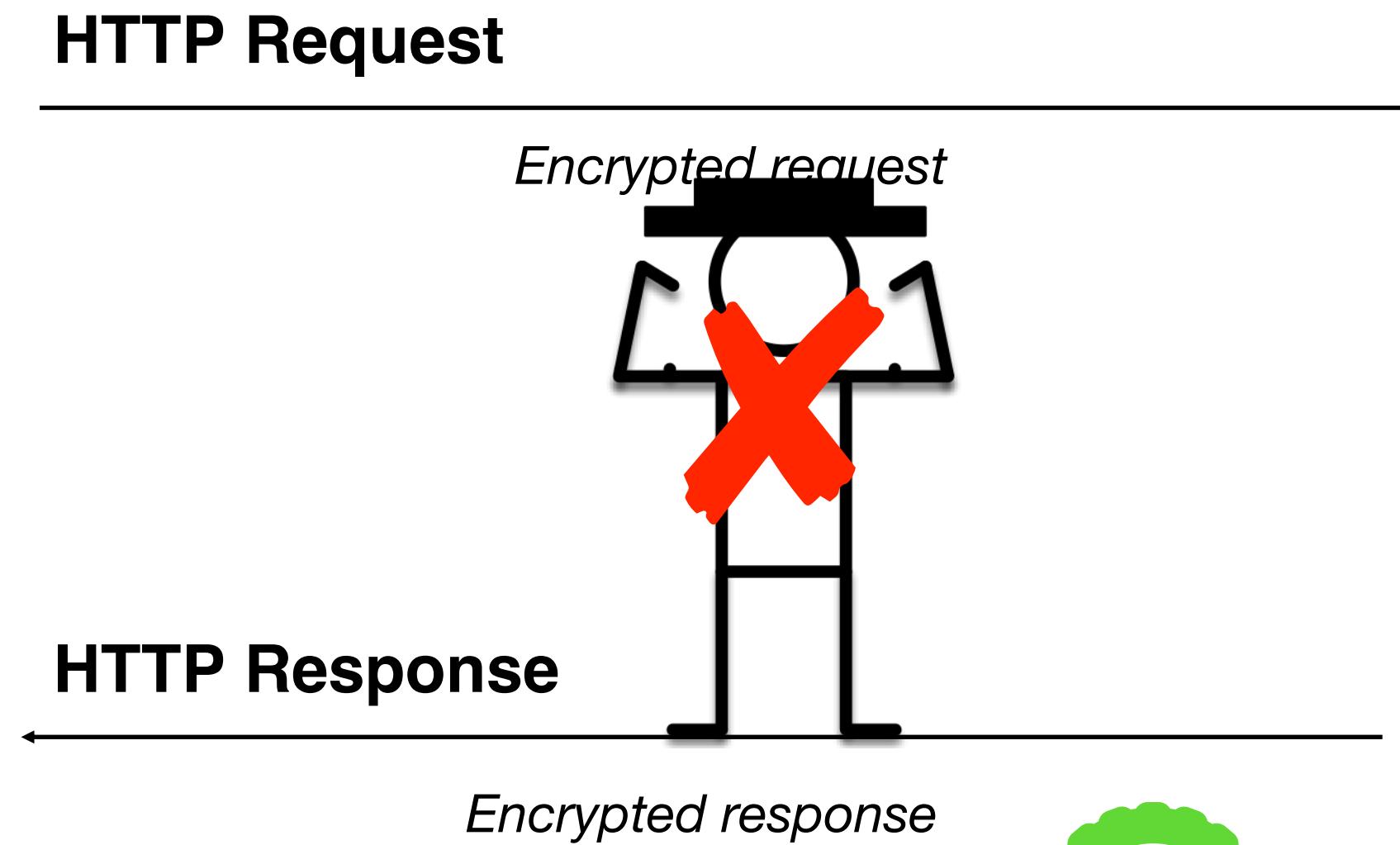
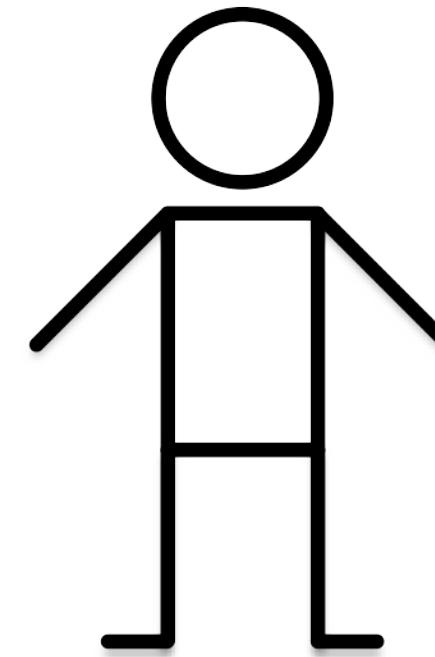
# Threat Mitigation: Trusted Code

## Preventing the man-in-the-middle with SSL



# Threat Mitigation: Trusted Code

## Preventing the man-in-the-middle with SSL



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)**



server

# 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](#) is truly [amazon.com](#)
- CA creates a certificate that binds [amazon.com](#)'s public key to the CA's public key (signing it using the CA's private key)

# Certificate Authorities issue SSL Certificates

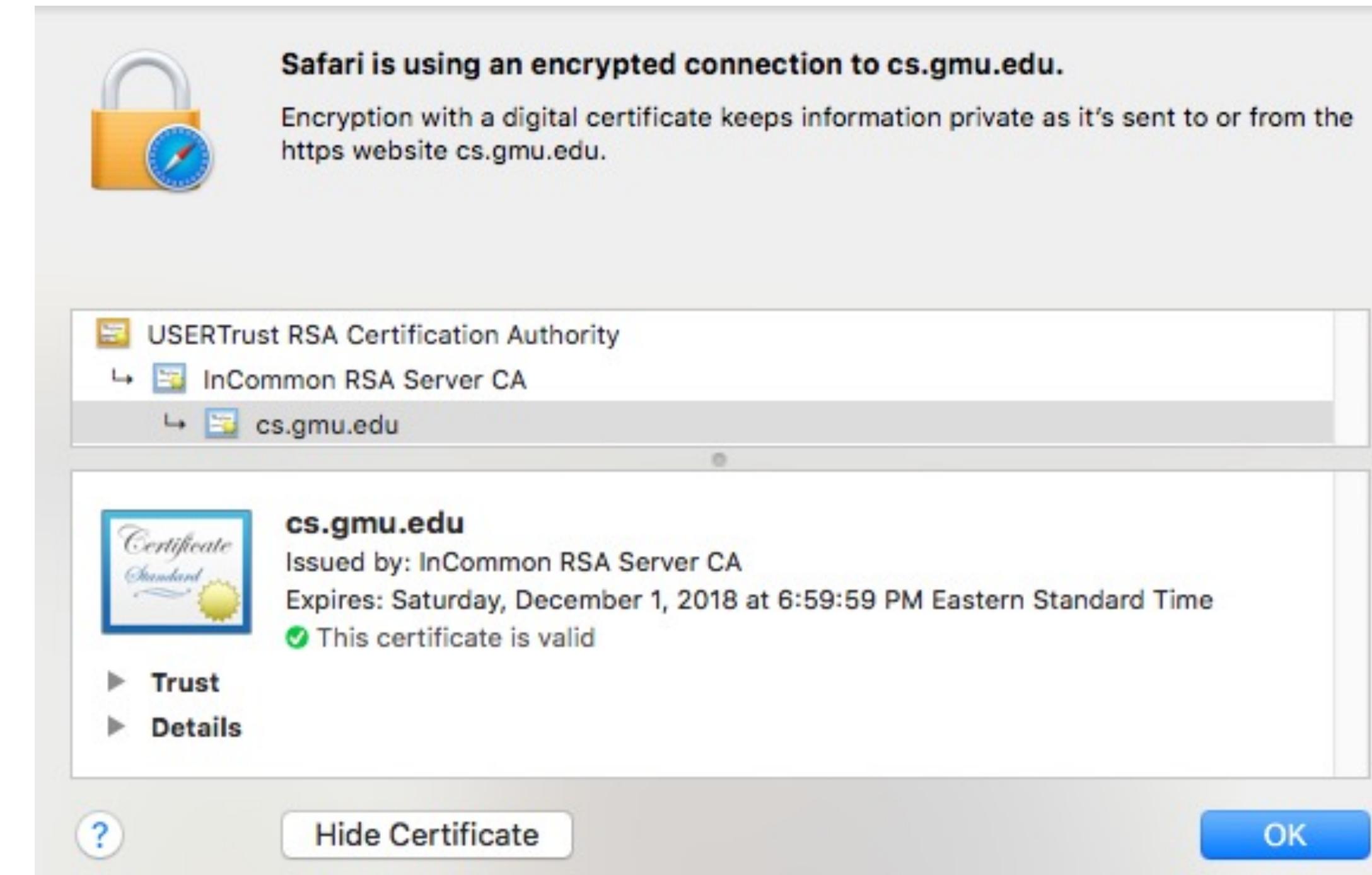


[amazon.com certificate](#)  
(AZ's public key + CA's sig)



# 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

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 ▾

Security

Comodo-gate hacker brags about forged certificate exploit

Tiger-blooded Persian cracker boasts of mighty exploits



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

# Threat Mitigation: Untrusted Inputs

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

- Special characters like <, >, ‘, “ and ` are often involved in exploits involving untrusted inputs
- Simple fix: Prohibit such inputs using 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

.....

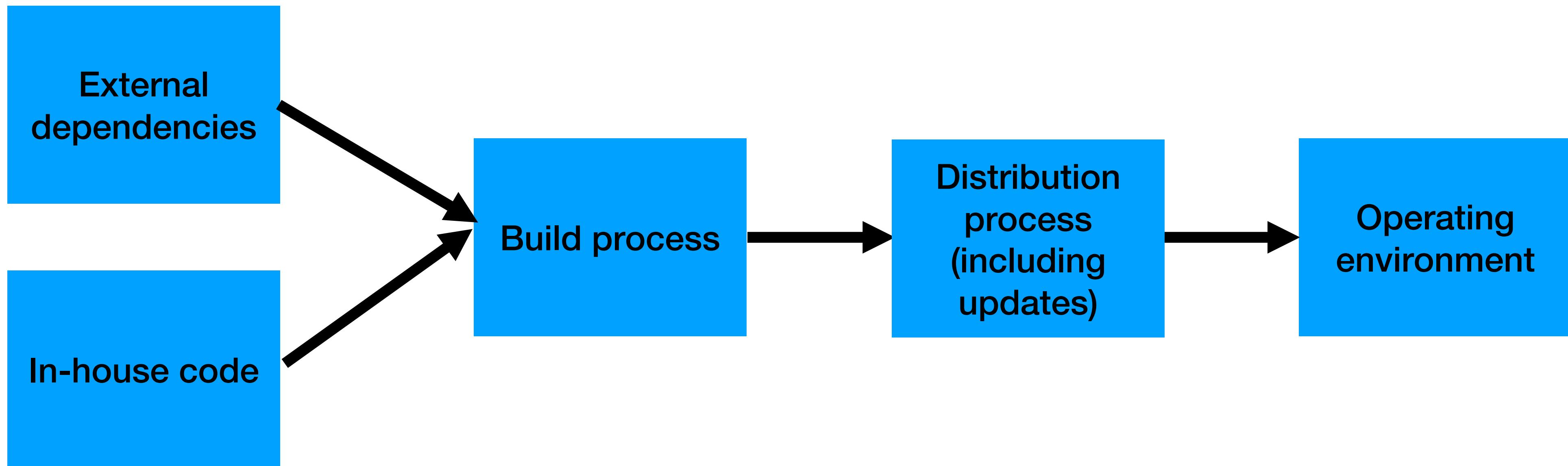
⚠ Your password must contain a minimum of 8 characters included with at least 1 upper case letter, 1 number, and 1 special character from !, #, \$, ^, and \* (other special characters are not supported).

# Threat Mitigation: Untrusted 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 Mitigation: Software Supply Chain

Consider threats at each phase



# Threat Mitigation: Software Supply Chain

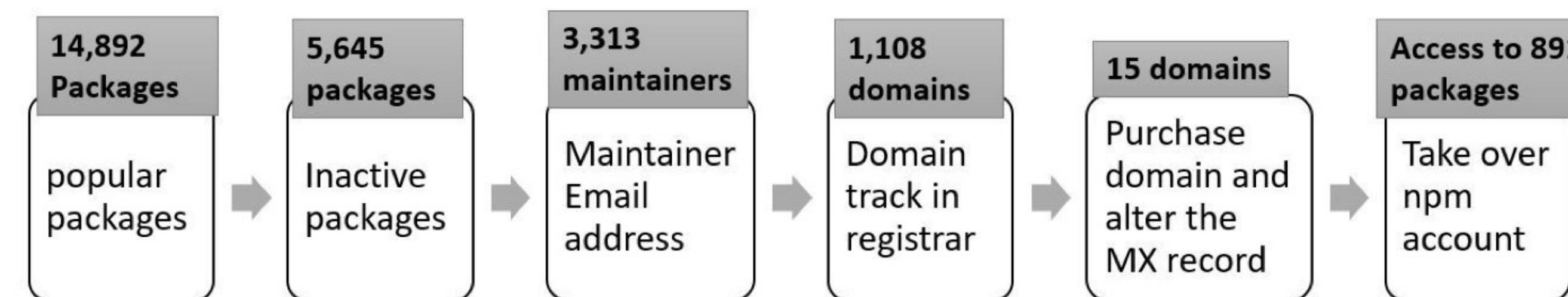
## Process-based solutions for process-based problems

- 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

# Weak Links in Software Supply Chain

## 2021 NCSU/Microsoft Study

- 8,498 NPM packages are maintained by at least one maintainer whose email address is inactive and could be purchased
- 33,249 NPM packages include installation scripts that can be exploited to run arbitrary code on developers' machines at installation-time
- 5,645 NPM packages are not actively maintained



"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>

# Which threats to protect against, at what cost?

## Consider various costs:

- Performance:
  - Encryption is not free; optimized code in C will be faster than code in TypeScript
- 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

# OWASP Top Security Risks

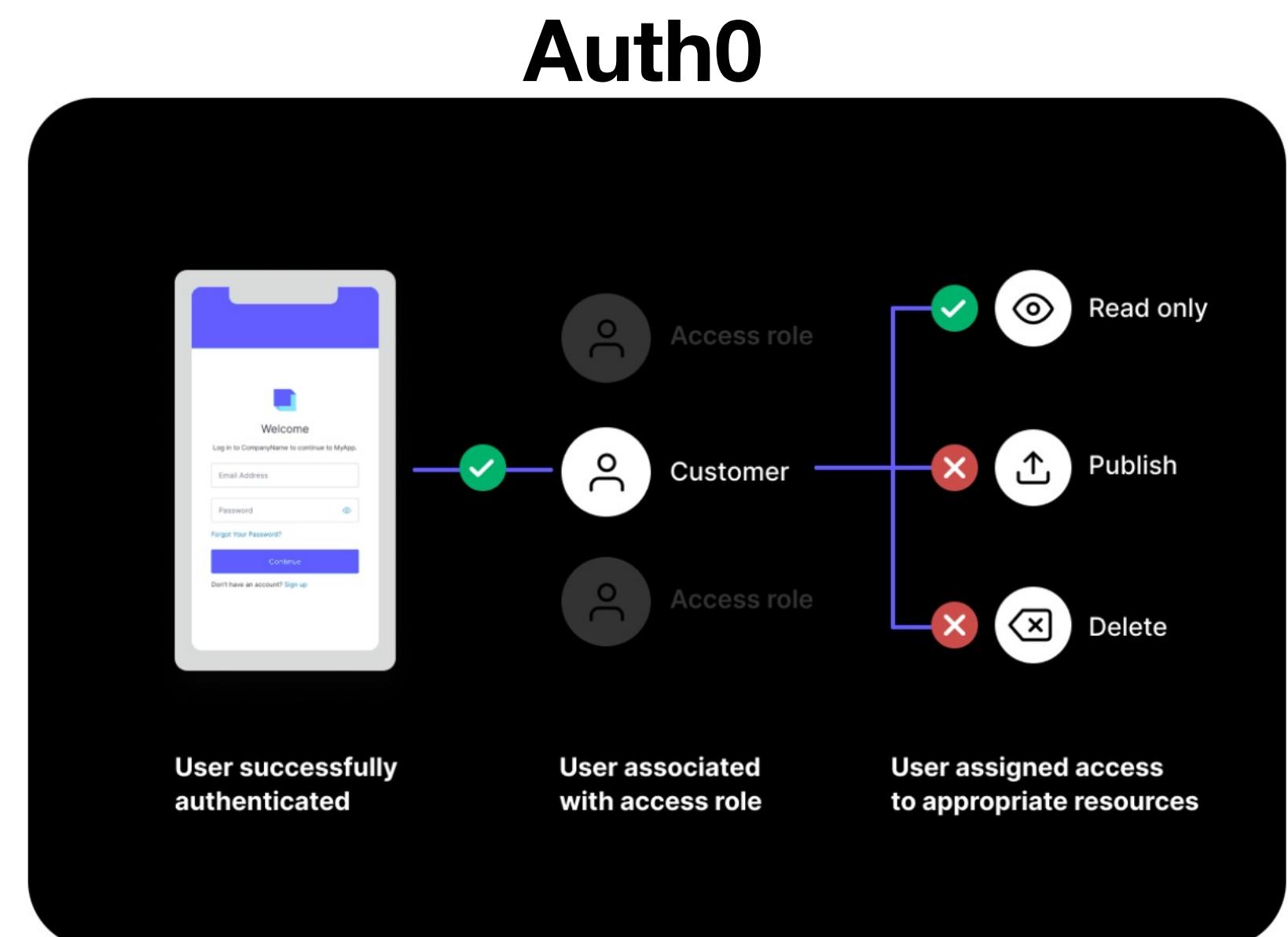
All 10: <https://owasp.org/www-project-top-ten/>

- 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

# Broken Authentication + Access Control

## OWASP #1

- 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



<https://auth0.com>

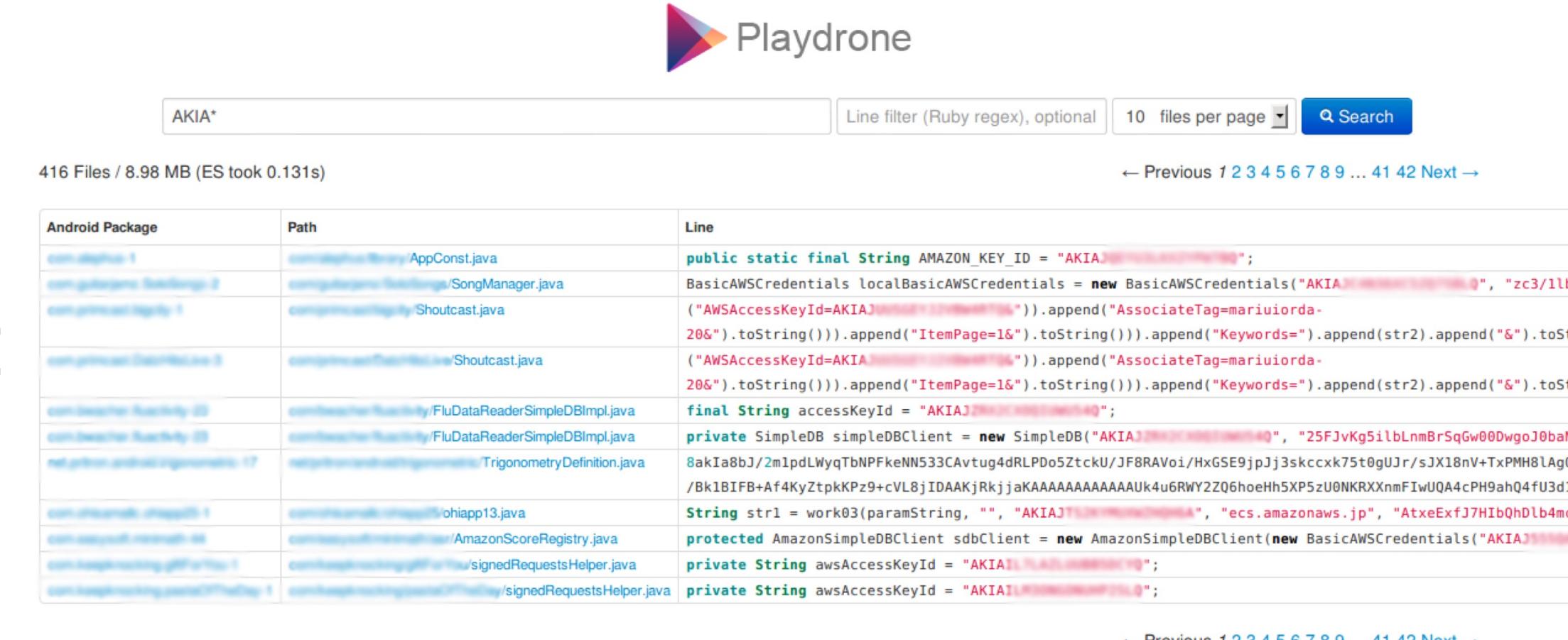
# Cryptographic Failures

## OWASP #2

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

	Amazon	Facebook	Twitter	Bitly	Flickr	Foursquare	Google	LinkedIn	Titanium
Total candidates	1,241	1,477	28,235	3,132	159	326	414	1,434	1,914
Unique candidates	308	460	6,228	616	89	177	225	181	1,783
Unique % valid	93.5%	71.7%	95.2%	88.8%	100%	97.7%	96.0%	97.2%	99.8%

Table 5: Credentials statistics from June 22, 2013 and validated on November 11, 2013. A credential may consist of an ID token and secret authentication token.



The screenshot shows a search interface for decompiled Java code. The search bar contains "AKIA\*". Below it, a table lists search results with columns for "Android Package", "Path", and "Line". The "Line" column displays code snippets containing the AWS access key "AKIA". The interface includes navigation links for "← Previous 1 2 3 4 5 6 7 8 9 ... 41 42 Next →".

Android Package	Path	Line
com.playdrone	com/playdrone/AppConst.java	public static final String AMAZON_KEY_ID = "AKIA";
com.playdrone.buildings	com/playdrone/buildings/SongManager.java	BasicAWS Credentials localBasicAWS Credentials = new BasicAWS Credentials("AKIA", "zc3/1lb");
com.primecast.library	com/primecast/library/Shoutcast.java	("AWS AccessKeyId=AKIA").append("AssociateTag=mariuorda-206").toString()).append("ItemPage=1&").toString()).append("Keywords=").append(str2).append("&").toString());
com.primecast.library	com/primecast/library/Shoutcast.java	("AWS AccessKeyId=AKIA").append("AssociateTag=mariuorda-206").toString()).append("ItemPage=1&").toString()).append("Keywords=").append(str2).append("&").toString());
com.dreamer.knuckley	com/dreamer/knuckley/FluDataReaderSimpleDBImpl.java	final String accessKeyId = "AKIA";
com.dreamer.knuckley	com/dreamer/knuckley/FluDataReaderSimpleDBImpl.java	private SimpleDB simpleDBClient = new SimpleDB("AKIA", "25FJvKg51bLnmBrSqGw00DwgoJ0baN8akIa8bJ/2m1pdLWyqTbNPfKeNN533CAvtug4dRLPD05ZtckU/JF8RAVoi/HxGSE9jpJj3skccxk75t0gUJr/sJX18nV+TxPMH8lAgQ/Bk1BIFB+A4KyZtpkKPz9+cVL8jIDAAKjRkjjaKAAAAAAAUAk4u6RWY2ZQ6hoeHh5XP5zU0NKRXXnmFiwUQA4cPH9ahQ4fU3d1String str1 = work03(paramString, "", "AKIA", "ecs.amazonaws.jp", "AtxeExfJ7HibQhDlbamc");
com.universalimagekit	com/universalimagekit/ohapp13.java	protected AmazonSimpleDBClient sdbClient = new AmazonSimpleDBClient(new BasicAWS Credentials("AKIA", "25FJvKg51bLnmBrSqGw00DwgoJ0baN8akIa8bJ/2m1pdLWyqTbNPfKeNN533CAvtug4dRLPD05ZtckU/JF8RAVoi/HxGSE9jpJj3skccxk75t0gUJr/sJX18nV+TxPMH8lAgQ/Bk1BIFB+A4KyZtpkKPz9+cVL8jIDAAKjRkjjaKAAAAAAAUAk4u6RWY2ZQ6hoeHh5XP5zU0NKRXXnmFiwUQA4cPH9ahQ4fU3d1String str1 = work03(paramString, "", "AKIA", "ecs.amazonaws.jp", "AtxeExfJ7HibQhDlbamc");
com.universalimagekit	com/universalimagekit/AmazonScoreRegistry.java	private String awsAccessKeyId = "AKIA";
com.universalimagekit	com/universalimagekit/signedRequestsHelper.java	private String awsAccessKeyId = "AKIA";
com.universalimagekit	com/universalimagekit/signedRequestsHelper.java	private String awsAccessKeyId = "AKIA";

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

# Hardcoded Credentials: Automated Checker

## GitGuardian (Launched in 2017)

The image shows the GitGuardian homepage. On the left, there's a large call-to-action section with the heading "Automated secrets detection & remediation". Below it, a sub-section says "Monitor public or private source code, and other data sources as well. Detect API keys, database credentials, certificates, ...". A blue button at the bottom of this section says "Schedule a demo". Above this, the main navigation bar includes links for "Products", "Pricing", "Resources", "Get a demo", and a "SIGN UP FOR FREE Internal Monitoring" button. To the right of the navigation is a large screenshot of the GitGuardian interface. The interface has a dark sidebar with icons for "PERIMETER", "ACTIVITY" (which is highlighted), "SECRETS", and "SETTINGS". The main area is titled "Activity" and shows three charts: "Push events" (77), "Public events" (12), and "Commits" (153). Below these charts is a table titled "Table of activity" with columns for "TYPE" and "ACTOR". The table lists various commits and events made by users like David Héault, elacaille18, genesixx, Deployment Bot (fro, Eric, elacaille18, dherault, and David Héault again).

TYPE	ACTOR
Commit	David Héault
Public	elacaille18
Event	genesixx
Commit	Deployment Bot (fro
Event	elacaille18
Commit	Eric
Event	elacaille18
Commit	Eric
Event	dherault
Commit	David Héault

# Cryptographic Failures

## Secret detection tools are not enough

- 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 in repository?
  - 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”

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

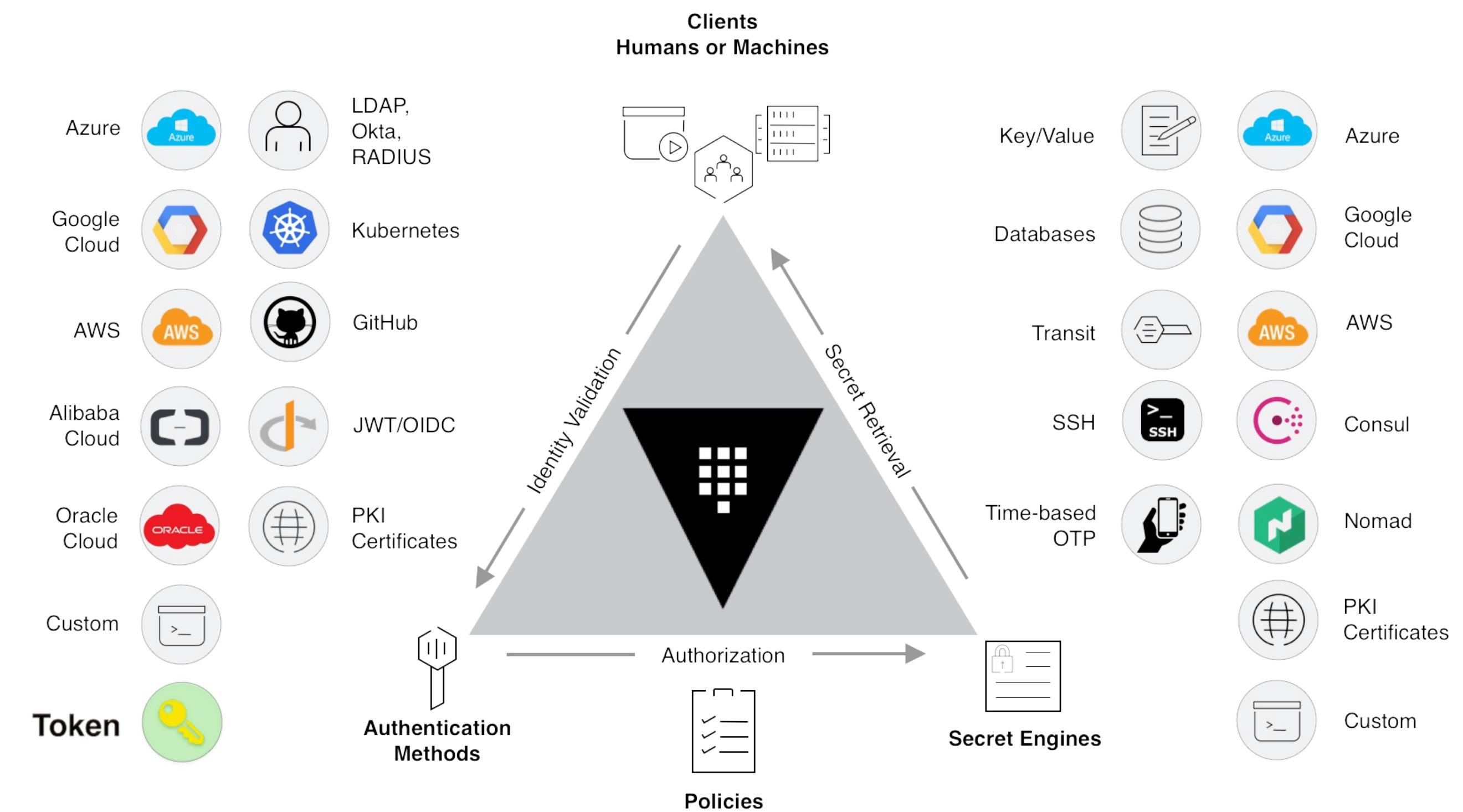
Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey & Laurie Williams

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

# Cryptographic Failures

Secret management tools (“Vaults”) centralize points of failure, and automates:

- Authorizing access to secrets
- Providing time-limited secrets
- Audit secret access



Example platform: HashiCorp Vault (open source, or cloud-hosted)  
<https://learn.hashicorp.com/tutorials/vault/getting-started-intro?in=vault/getting-started>

# Code Injection

## OWASP #3

- **Sanitize user-controlled inputs (remove HTML)**
- Use tools like LGTM to detect vulnerable data flows
- Use middleware that side-steps the problem (e.g. return data as JSON, client puts that data into React component)

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 links for Help, Query console, Project lists, My alerts, and a user profile for Jonathan Bell. Below the navigation is a purple header bar with tabs for Alerts (16), Logs, Files (selected), History, Compare, Integrations, and Queries. A green banner at the top of the main content area says "Sammie is joining GitHub". The main content area has a sidebar on the left with "Alert filters" (No filter selected, Severity, Query, Tag, Show excluded files, Show heatmap) and an "Export alerts" button. Below the sidebar is a circular progress bar. To the right is a table with columns for Name, Alerts, and Lines of code. The table shows three rows: "public" (0 alerts, 0 lines of code), "src" (16 alerts, 756 lines of code), and "package.json" (0 alerts, 0 lines of code). A large orange circle is overlaid on the bottom left of the main content area.

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

- Clear text storage of sensitive information**  
Sensitive information stored without encryption or hashing can expose it to an attacker.
- Clear-text logging of sensitive information**  
Logging sensitive information without encryption or hashing can expose it to an attacker.
- Client-side cross-site scripting**  
Writing user input directly to the DOM allows for a cross-site scripting vulnerability.
- Client-side 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

# Using Components with Known Vulnerabilities

## OWASP #5



Bump junit from 4.12 to 4.13.1 #155

Merged jon-bell merged 1 commit into master from dependabot/maven/junit-junit-4.13.1 22 days ago

This automated pull request fixes a security vulnerability  
Only users with access to Dependabot alerts can see this message. [Learn more about Dependabot security updates, opt out, or give us feedback.](#)

Conversation 0 Commits 1 Checks 2 Files changed 1

**dependabot** bot commented on behalf of github on Oct 13

Bumps junit from 4.12 to 4.13.1.

▶ Release notes

▶ Commits

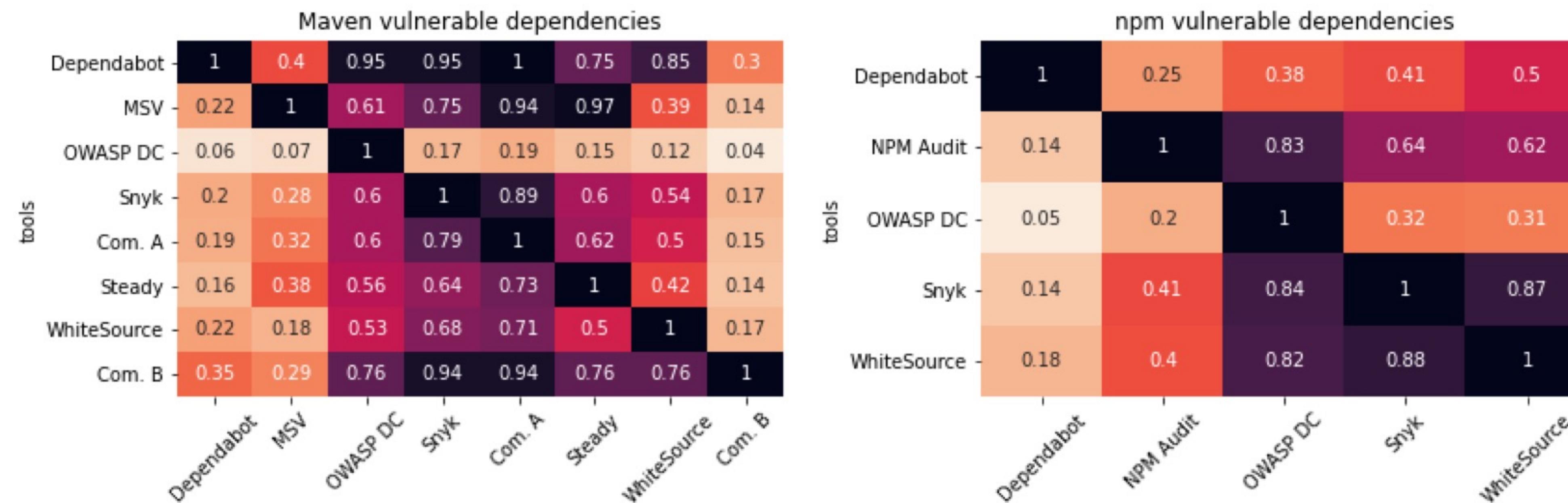
compatibility 93%

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting @dependabot rebase .

# Using Components with Known Vulnerabilities

## Static analyses are imperfect

- Study: Vulnerable dependencies reported on a large, open source project, OpenMRS. Compare results across tools.



“A comparative study of vulnerability reporting by software composition analysis tools ”

Nasif Imtiaz, Seaver Thorn and Laurie Williams

<https://dl.acm.org/doi/10.1145/3475716.3475769>

# Creating a Reasonable Threat Model

**What value can an attacker extract from a vulnerability?**

- Does our code contain any sensitive data?
- What is the cost if that data is breached or tampered with?
  - Even if your code is not “sensitive”: does it expose other routes of attack?



# Creating a Reasonable Threat Model

## Best practices applicable in most situations

- Trust:
  - Developers writing our code
  - Server running our code
  - Popular dependencies that we use and update
- Don't trust:
  - Code running in browser
  - Inputs from users
- Practice good security practices:
  - Encryption (all data in transit, sensitive data at rest)
  - Code signing, multi-factor authentication
- Bring in security experts early for riskier situations

# Learning Objectives for this Lesson

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

- Describe that security is a spectrum, and be able to define a realistic threat model for a given system
- Evaluate the tradeoffs between security and performance in software engineering
- Recognize the causes of and common mitigations for common vulnerabilities in web applications
- Utilize static analysis tools to identify common weaknesses in code