

CS 4530 & CS 5500 **Software Engineering**

Lecture 9.4: Engineering Secure Software

Jonathan Bell, John Boyland, Mitch Wand
Khoury College of Computer Sciences
© 2021, released under [CC BY-SA](#)

Learning Objectives for this Lesson

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

- Recognize the causes of and common mitigations for common vulnerabilities in web applications
- Utilize static analysis tools to identify common weaknesses in code

OWASP Top Security Risks

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

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

Code Injection Example

OWASP A1:2017-Injection

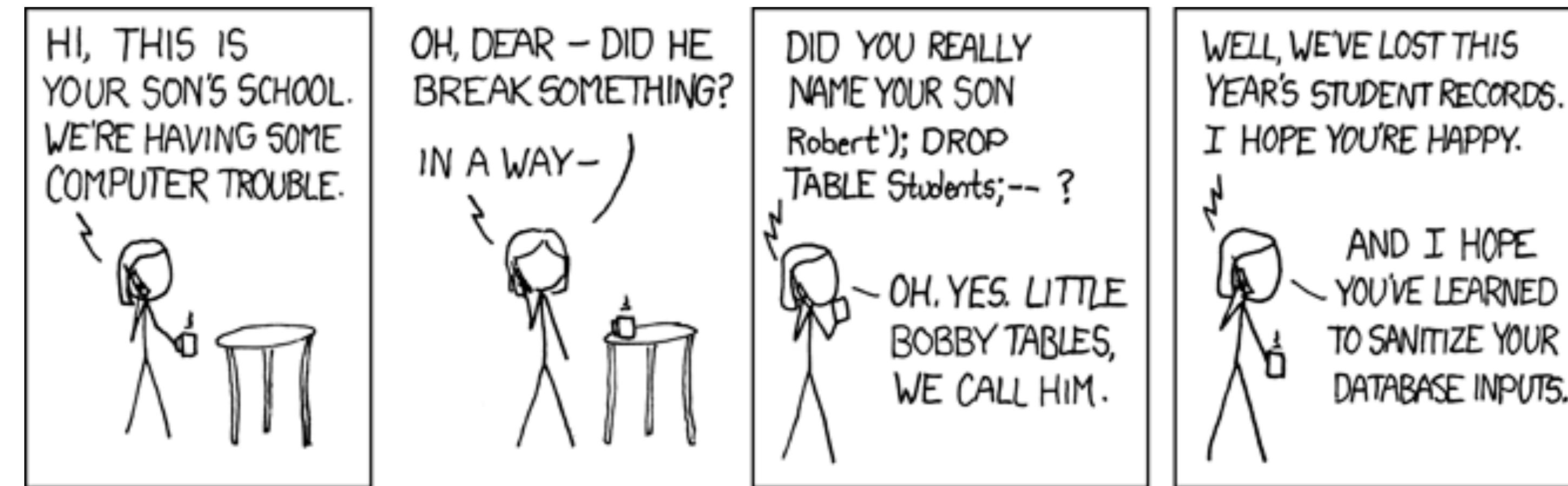
```
String query = "SELECT * FROM accounts WHERE  
    name='\" + request.getParameter("name") + '\"";
```

Parameter name	Constructed Query	Effect
Alice	SELECT * FROM accounts WHERE name='Alice';	Select a single account
Alice O'Neal	SELECT * FROM accounts WHERE name='Alice O'Neal';	SQL Error
5' OR '1'='1	SELECT * FROM accounts WHERE name='5' OR '1'='1';	Select all accounts

THIS IS AN ATTACK

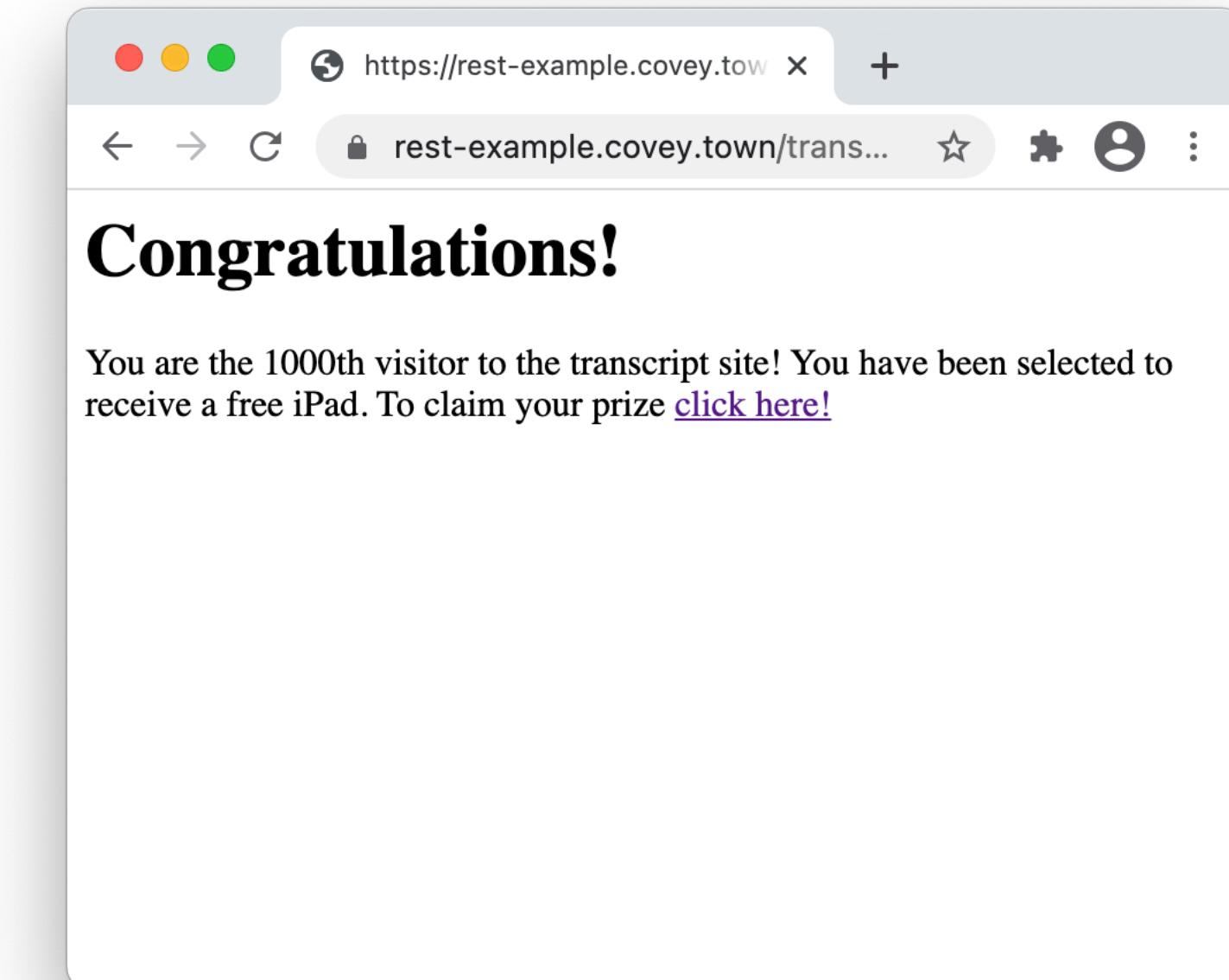
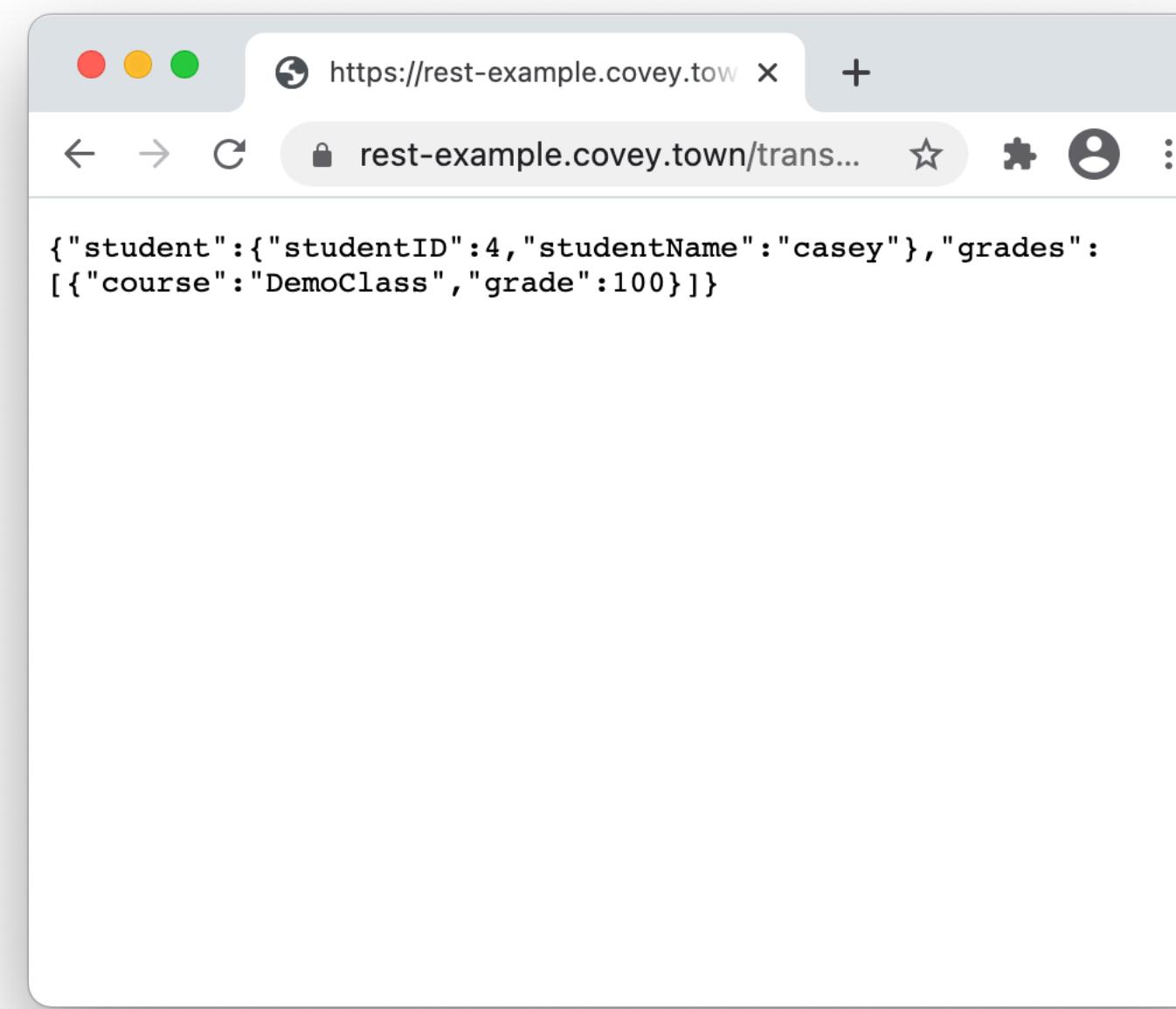
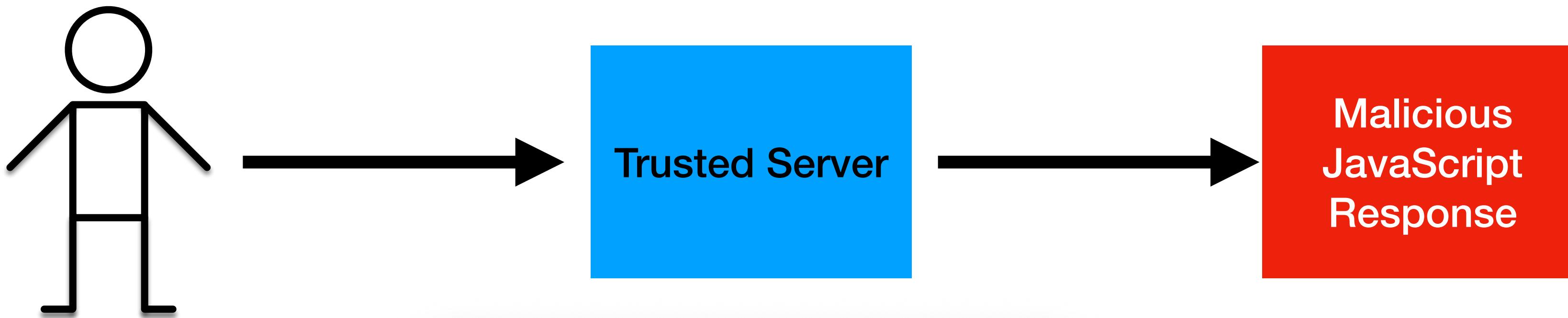
Code Injection Example

XKCD #327



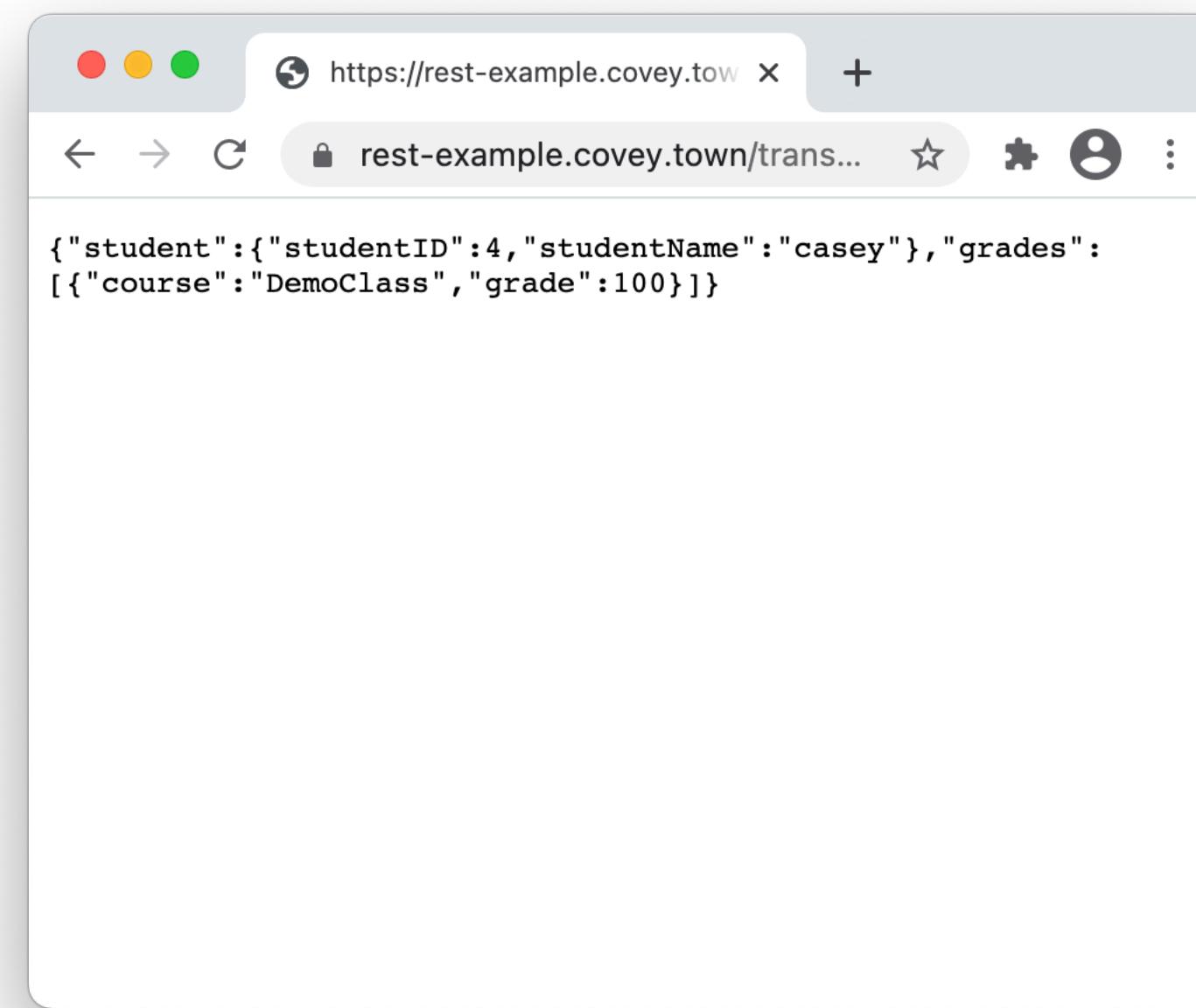
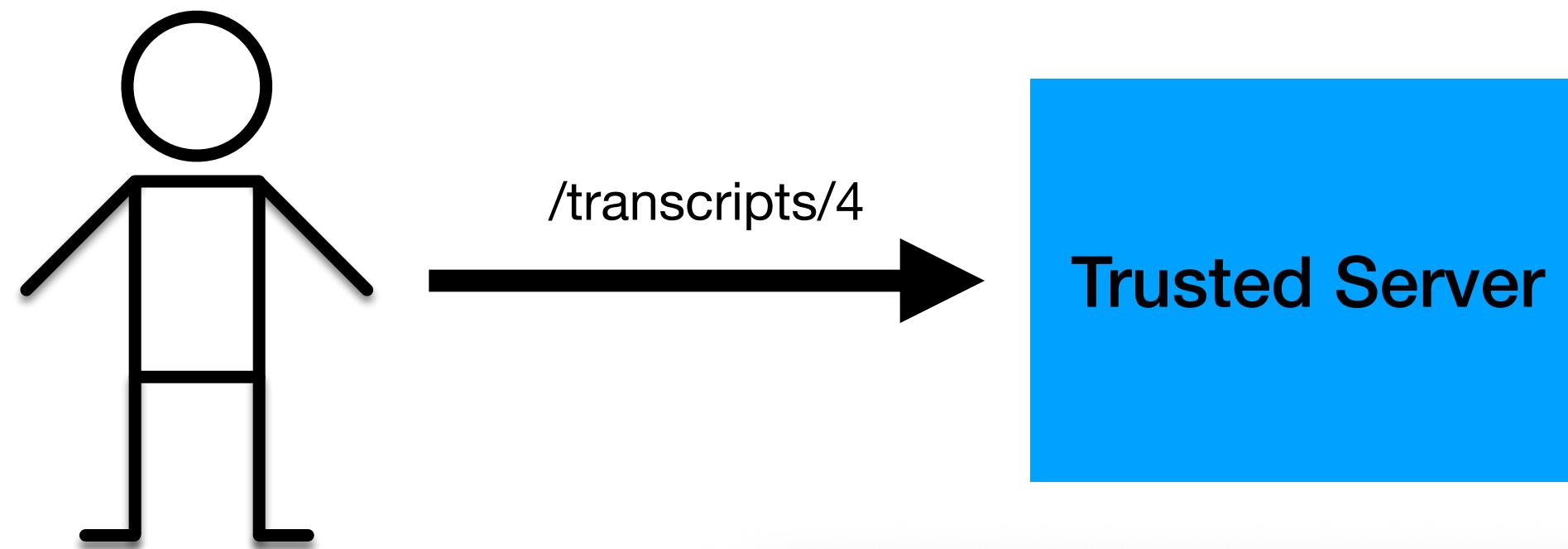
Code Injection Example

Cross-site scripting (XSS)



Code Injection Example

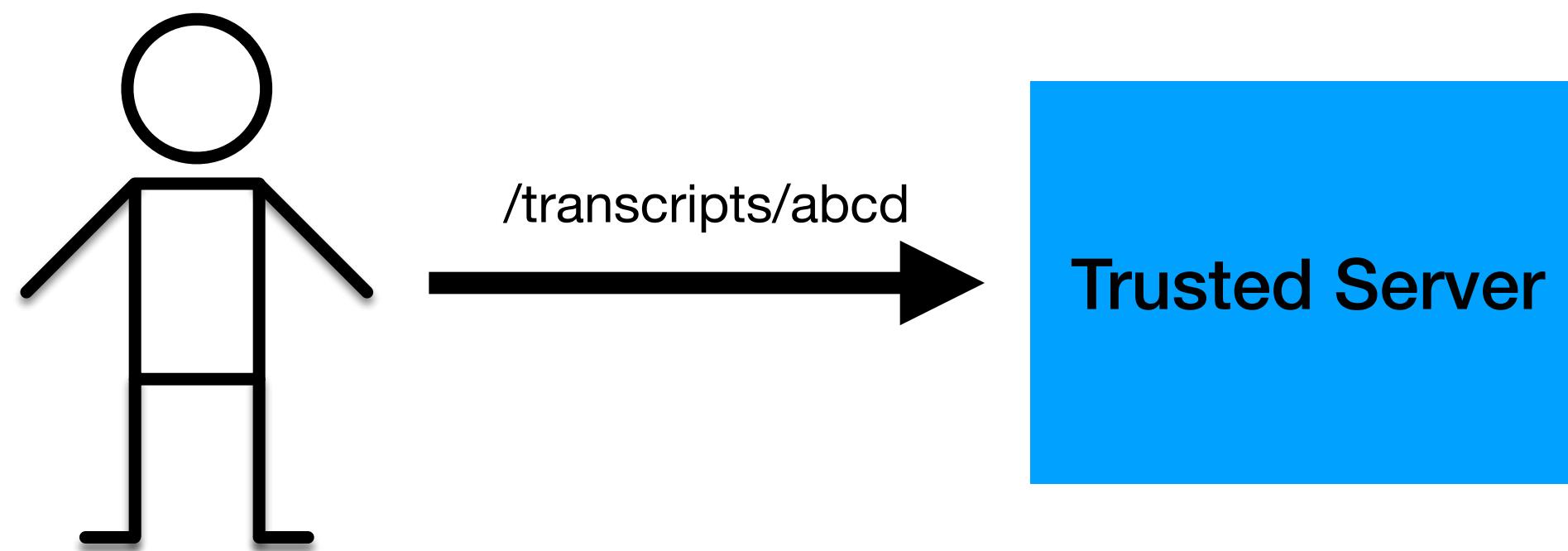
Cross-site scripting (XSS)



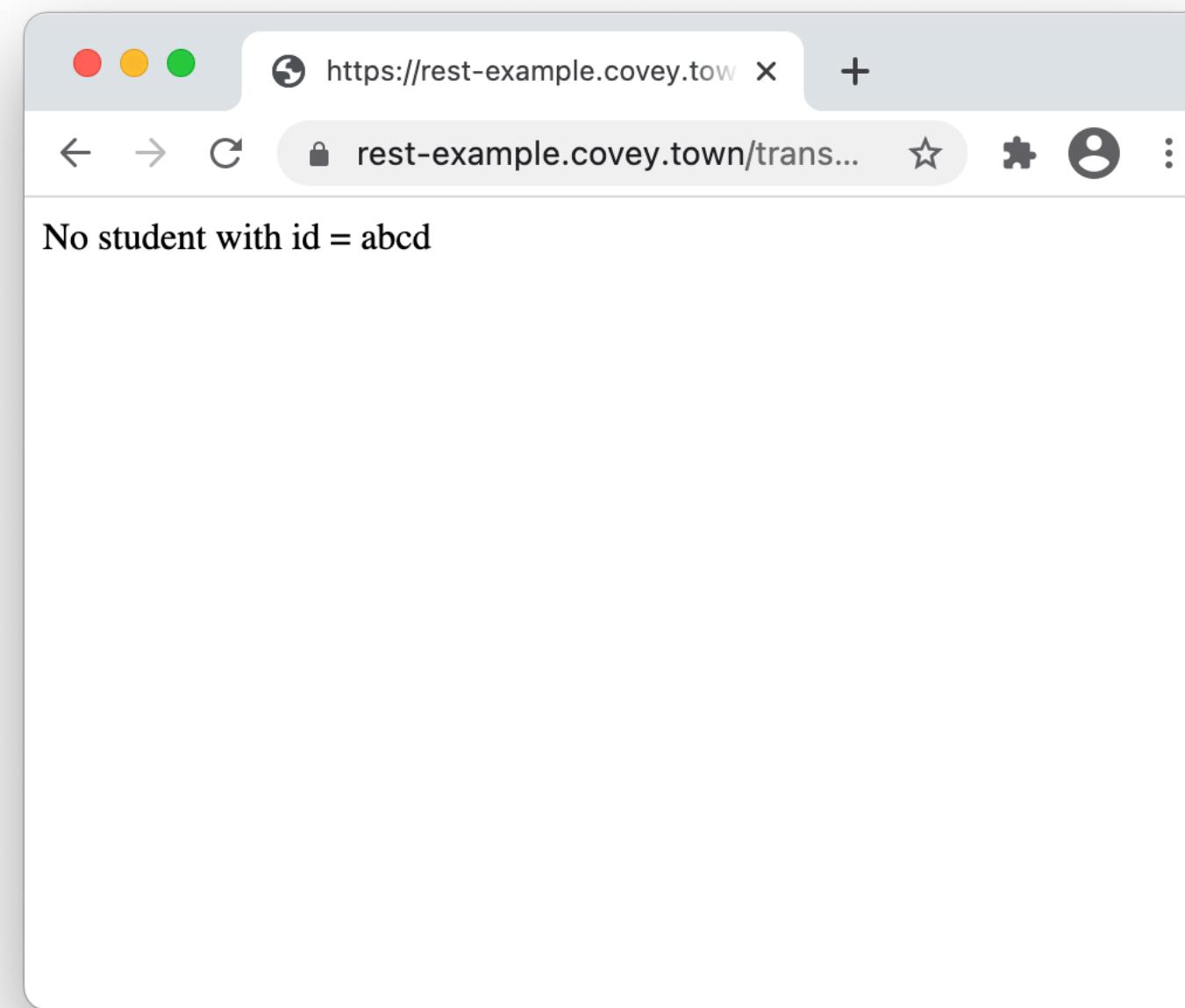
```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  {
    res.status(200).send(theTranscript);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

Code Injection Example

Cross-site scripting (XSS)

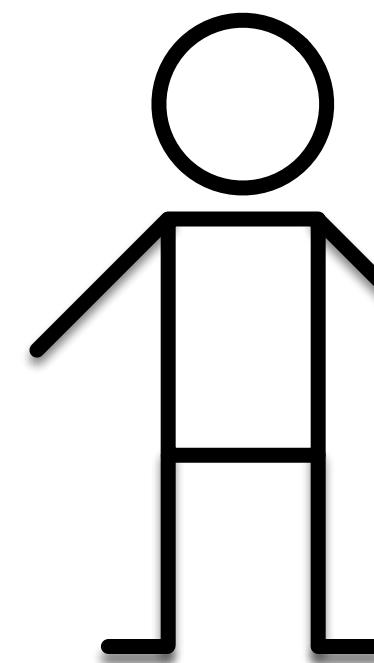


```
app.get('/transcripts/:id', (req, res) => {
  // req.params to get components of the path
  const {id} = req.params;
  const theTranscript = db.getTranscript(parseInt(id));
  if (theTranscript === undefined) {
    res.status(404).send(`No student with id = ${id}`);
  }
  {
    res.status(200).send(theTranscript);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

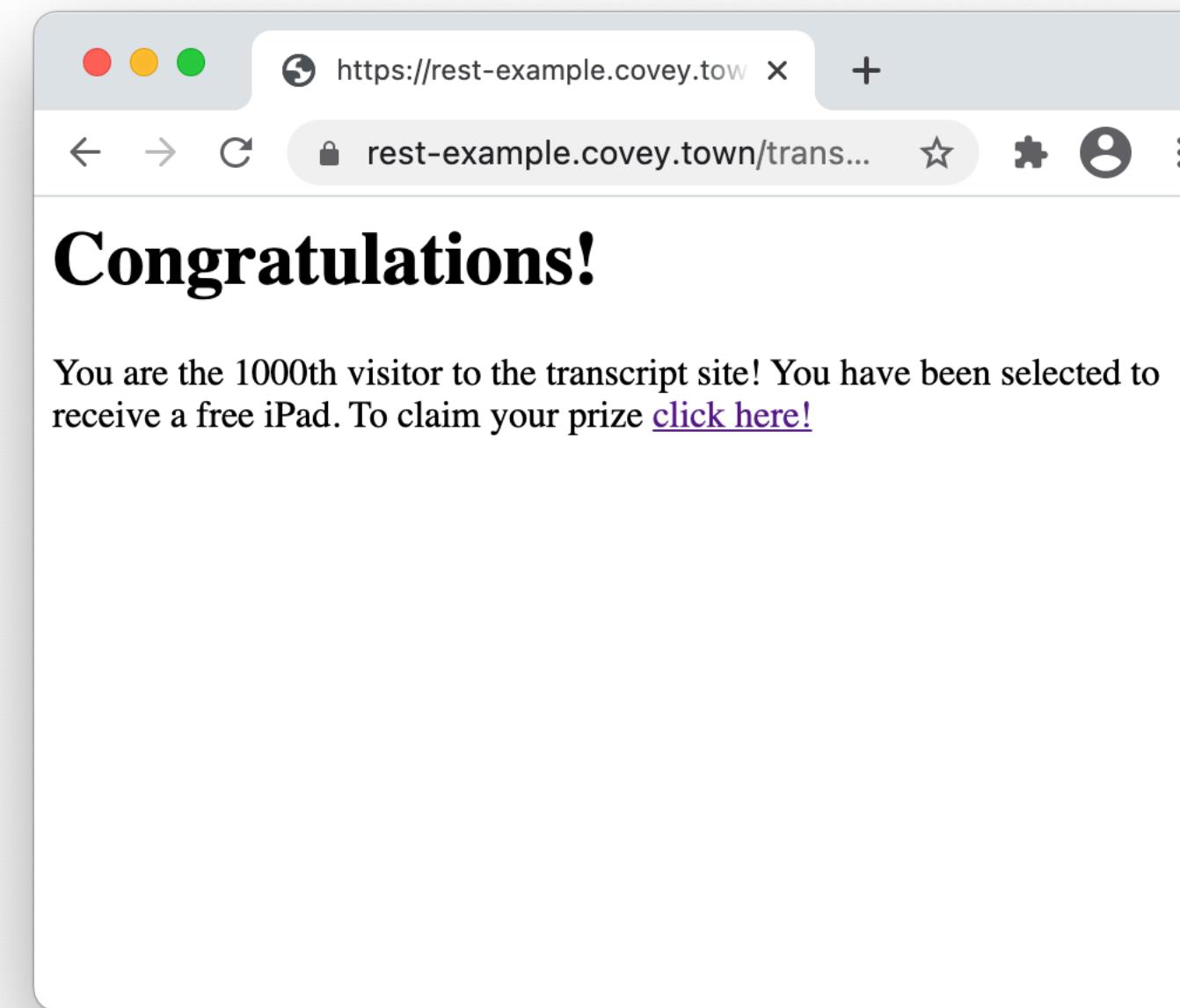
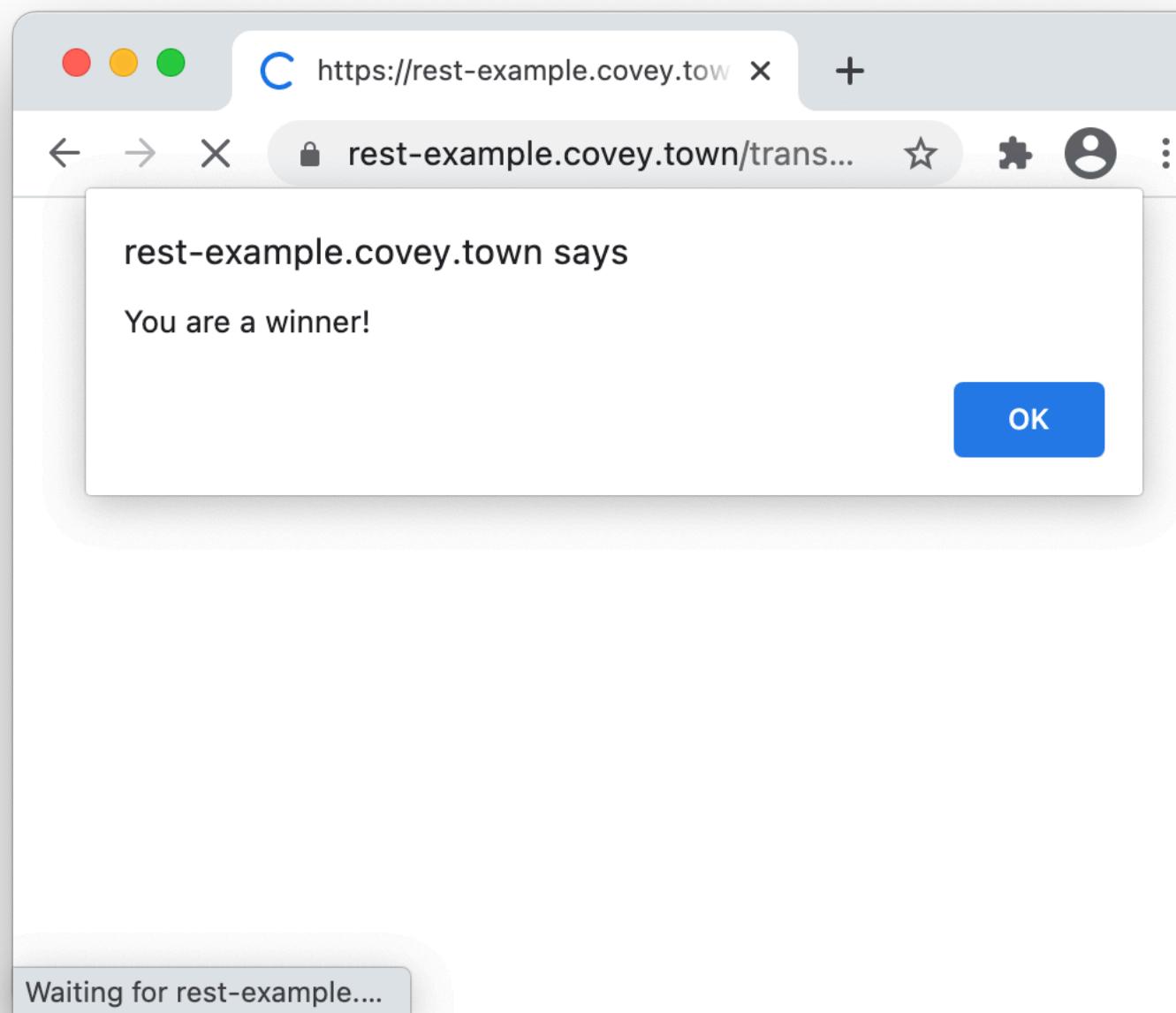
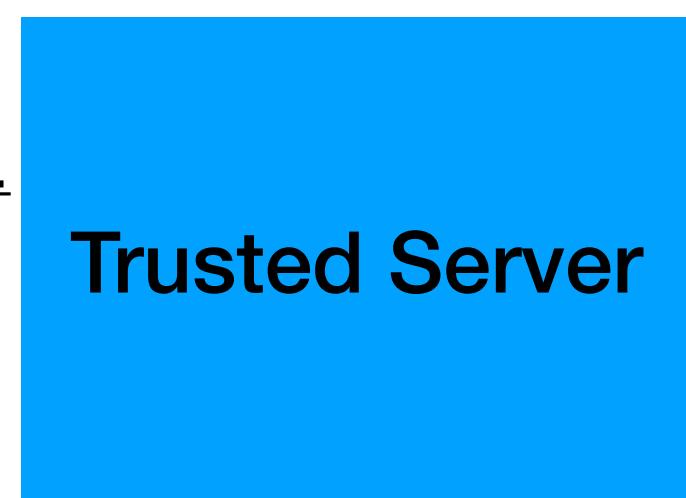


Code Injection Example

Cross-site scripting (XSS)



/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}`);
  }
  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 click here!
<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!';
alert('You are a winner!');
</script>

Code Injection Example

Java code injection 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 banner with the text "2017 Cybersecurity Incident & Important Consumer Information". On the right side of the banner, there's a news card 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 titled "CVE-2017-5638 Detail".

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.



Cross-site Scripting

How to fix it?

- **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

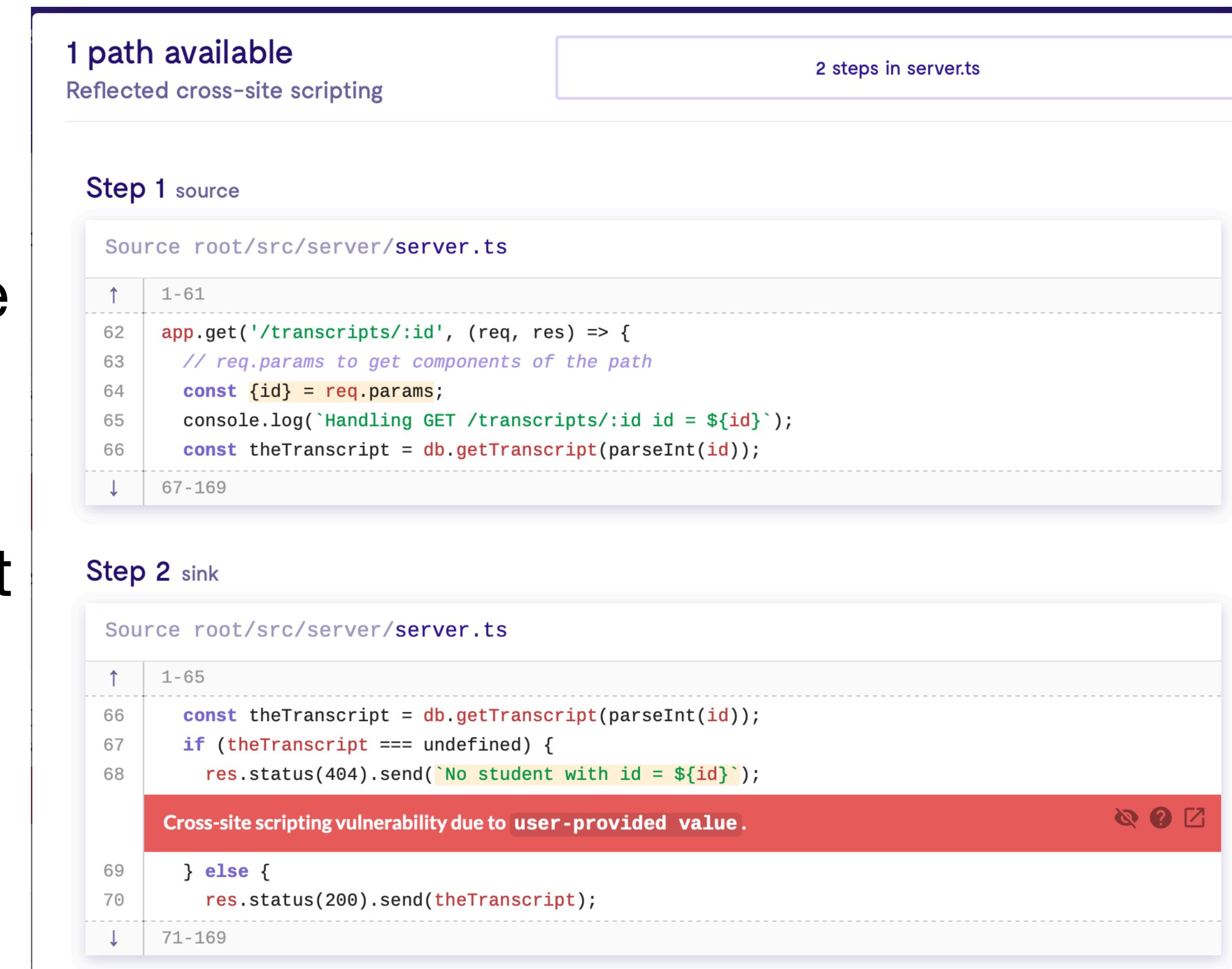
Step 1 source

```
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 notification bubble says "Sammie is joining GitHub". The main content area has a message: "By default, only the files that also appear in the [Alerts tab](#) are listed here. Files classified as non-standard, such as test code or generated files, are shown only when you check 'Show excluded files'." On the left, there's an "Alert filters" section with dropdowns for Severity, Query, Tag, and checkboxes for "Show excluded files" and "Show heatmap". Below that is a "Source root/" section with a circular progress bar. The main 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

- 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>

OWASP Top Security Risks

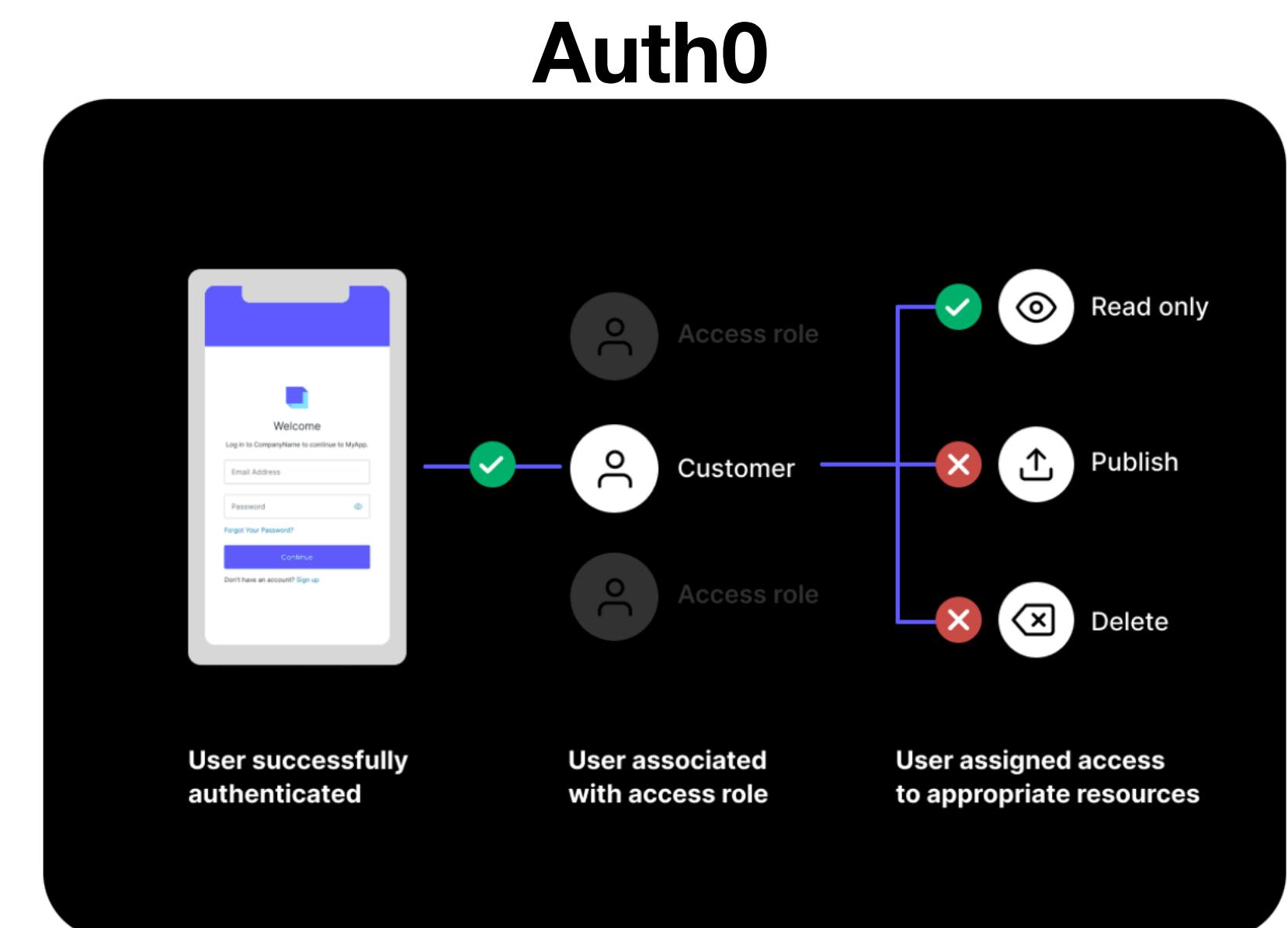
All 10: <https://owasp.org/www-project-top-ten/>

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

Broken Authentication + Access Control

How to fix it?

- 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>

Broken Authentication + Access Control

Specifically: CWE-798: Use of Hard-coded Credentials

```
<SCRIPT>
function passWord() {
    var testV = 1;
    var pass1 = prompt('Please Enter Your Password',' ');
    while (testV < 3) {
        if (!pass1)
            history.go(-1);
        if (pass1.toLowerCase() == "letmein") {
            alert('You Got it Right!');
            window.open('protectpage.html');
            break;
        }
        testV+=1;
        var pass1 =
            prompt('Access Denied – Password Incorrect, Please Try Again.','Password');
    }
    if (pass1.toLowerCase()!="password" & testV ==3)
        history.go(-1);
    return " ";
}
</SCRIPT>
<CENTER>
<FORM>
<input type="button" value="Enter Protected Area" onClick="passWord()">
</FORM>
</CENTER>
```

Broken Authentication + Access Control

CWE-798: Use of Hard-coded Credentials: Study of 1.1m Android Apps

	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 the PLAYDRONE web interface. At the top, there is a search bar with the placeholder "AKIA*", a line filter input, a dropdown for "10 files per page", and a "Search" button. Below the search bar, it says "416 Files / 8.98 MB (ES took 0.131s)". On the right, there are navigation links for "← Previous 1 2 3 4 5 6 7 8 9 ... 41 42 Next →". The main area displays a table with three columns: "Android Package", "Path", and "Line". The table lists several Java code snippets containing hardcoded AWS access keys, such as "AKIAI...", "BasicAWSCredentials localBasicAWSCredentials = new BasicAWSCredentials("AKIAI...)", and "String str1 = work03(paramString, "", "AKIAI...")". The table has 13 rows, each corresponding to a different file path and package name.

Android Package	Path	Line
com.amazonaws.Alexa	AppConst.java	public static final String AMAZON_KEY_ID = "AKIAI...";
com.amazonaws.SongCast	SongManager.java	BasicAWSCredentials localBasicAWSCredentials = new BasicAWSCredentials("AKIAI..."); "zc3/1lb...
com.amazonaws.Shoutcast	Shoutcast.java	("AWSAccessKeyId=AKIAI...").append("AssociateTag=mariuiorda-208").toString()).append("ItemPage=1&").toString()).append("Keywords=").append(str2).append("&").toSt
com.amazonaws.Shoutcast	Shoutcast.java	("AWSAccessKeyId=AKIAI...").append("AssociateTag=mariuiorda-208").toString()).append("ItemPage=1&").toString()).append("Keywords=").append(str2).append("&").toSt
com.amazonaws.SimpleDB	FluDataReaderSimpleDBImpl.java	final String accessKeyId = "AKIAI...";
com.amazonaws.SimpleDB	FluDataReaderSimpleDBImpl.java	private SimpleDB simpleDBClient = new SimpleDB("AKIAI..."); "25FJvKg5ilbLnmBr5qGw00DwgoJ0baN
com.amazonaws.Trigonometry	TrigonometryDefinition.java	8akIa8Bj/2m1pdLWyqTbNPFkeNN533CAvtug4dRLPDo5ZtckU/JF8RAVoi/HxGSE9jpJj3skccxk75t0gUJr/sJX18nV+TxPMH8lAgQ/Bk1B1FB+Af4KyZtpkKPz9+cVL8jIDAOKjRkjjaKAAAAAAAUAUk4u6RWY2ZQ6hoeHh5XP5zU0NKRXXnmFIwUQA4cPH9ahQ4fU3d1
com.amazonaws.ohiapp	ohiapp13.java	String str1 = work03(paramString, "", "AKIAI..."); "ecs.amazonaws.jp", "AtxeExfJ7HIbQhDlb4mc
com.amazonaws.sdb	AmazonScoreRegistry.java	protected AmazonSimpleDBClient sdbClient = new AmazonSimpleDBClient(new BasicAWSCredentials("AKIAI..."));
com.amazonaws.signedRequests	signedRequestsHelper.java	private String awsAccessKeyId = "AKIAI...";
com.amazonaws.signedRequests	signedRequestsHelper.java	private String awsAccessKeyId = "AKIAI...";

Figure 9: PLAYDRONE's web interface to search decompiled sources showing Amazon Web Service tokens found in 130 ms.

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 subtext reads: "Monitor public or private source code, and other data sources as well. Detect API keys, database credentials, certificates, ...". A blue button labeled "Schedule a demo" is present. On the right, a screenshot of the GitGuardian interface is displayed. The interface has a dark sidebar with icons for PERIMETER, ACTIVITY (highlighted), SECRETS, and SETTINGS. The main area is titled "Activity" and shows three charts: "Push events" (77), "Public events" (12), and "Commits" (153). Below the charts is a line graph for "Push events" from December 10 to 16. To the right is a "Table of activity" listing various events and commits with their types and actors. At the bottom, there's a decorative pattern of colored dots.

Products ▾ Pricing Resources ▾ Get a demo SIGN UP FOR FREE Internal Monitoring

Automated secrets detection & remediation

Monitor public or private source code, and other data sources as well. Detect API keys, database credentials, certificates, ...

Schedule a demo

Activity

PUSH EVENTS 77 PUBLIC EVENTS 12 COMMITS 153

Push events

Public events

Commits

Table of activity

FILTER Filter by search...

TYPE ACTOR

- Commit David Héault
- Public elacaille18
- Event genesixx
- Commit Deployment Bot (frd)
- Event elacaille18
- Commit Eric
- Event elacaille18
- Commit Eric
- Event dherault
- Commit David Héault

OWASP Top Security Risks

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

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

Weakly Protected Sensitive Data

How to fix it?

- 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

OWASP Top Security Risks

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

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

Using Components with Known Vulnerabilities

How to fix it?



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`.

Learning Objectives for this Lesson

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

- Recognize the causes of and common mitigations for common vulnerabilities in web applications
- Utilize static analysis tools to identify common weaknesses in code

This work is licensed under a Creative Commons Attribution-ShareAlike license

- This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>
- You are free to:
 - Share — copy and redistribute the material in any medium or format
 - Adapt — remix, transform, and build upon the material
 - for any purpose, even commercially.
- Under the following terms:
 - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
 - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.