

CS 4530: Fundamentals of Software Engineering

Module 15: Software Engineering & Security

Adeel Bhutta, Jan Vitek and Mitch Wand
Khoury College of Computer Sciences

© 2023, released under [CC BY-SA](#)

Learning objectives

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

- Define key terms relating to security
- Describe tradeoffs between security and other SE requirements
- Explain common vulnerabilities in web apps, and mitigations
- Explain why software alone isn't enough to assure security

Vocabulary (CIA)

Security as non-functional requirements:

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

Vocabulary

Threat: potential event that could compromise a security requirement

Security architecture: mechanisms and policies we build in to mitigate threats

Vulnerability: flaw in a system that could be exploited by an adversary

Exploit: use of a vulnerability to mount an attack

Security isn't free

In software, as in the real world...

- *You moved to a new house, someone moved out. What do you do?*
- 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 risk management

Cost of attack v. cost of defense?

Increasing security might

- increase development & maintenance cost
- increase infrastructure requirements
- degrade performance

But, if attacked, increasing security might also

- decrease financial and intangible losses

So, how likely is any exploit and how bad is the threat?

Threat models help analyze tradeoffs

What is being defended?

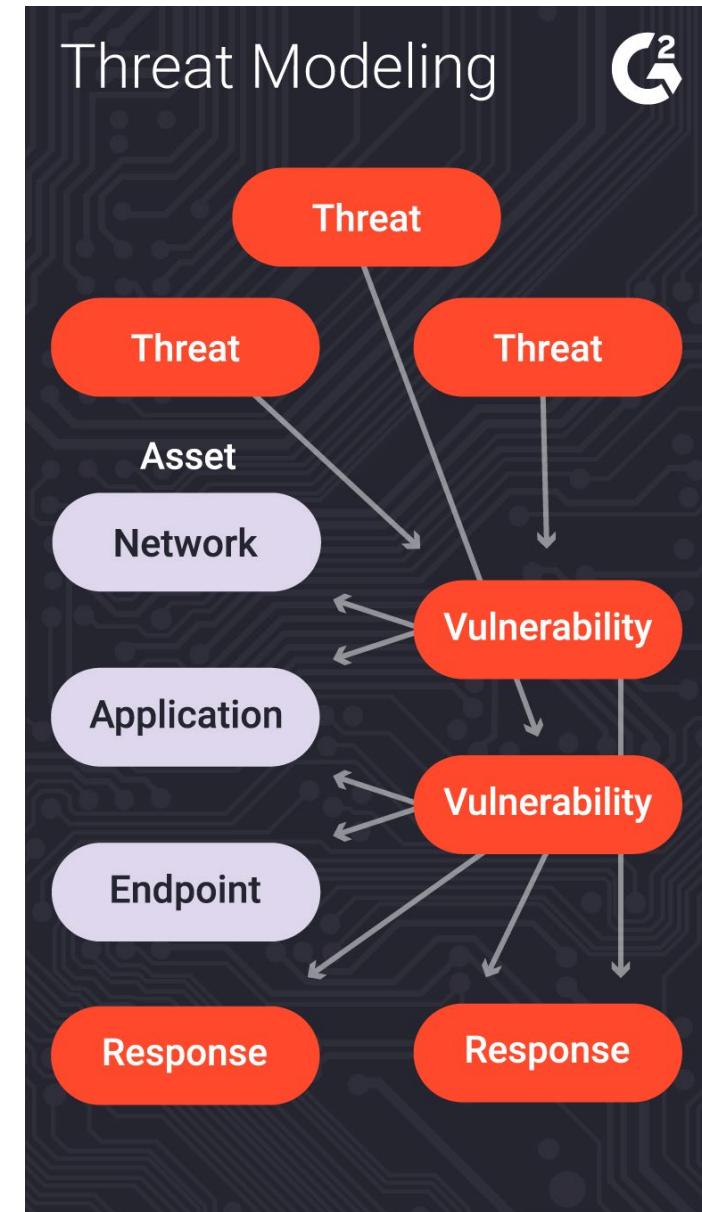
- what resources are important?
- who are the malicious actors
- what vulnerabilities may be exploited?
- what is the value of an exploit?

Who do we trust?

- what can be considered secure and trusted?

Plan responses to possible attacks

- can we prioritize?



A baseline security architecture

Best practices

Trust:

- Our developers
- Server running our code
- Libraries that we have vetted

Don't trust:

- Code downloaded from a web site
- Unfiltered inputs
- Anyone else

A baseline security architecture

Security best practices

- Encryption (data in transit, sensitive data at rest)
- Code signing, multi-factor authentication
- Encapsulated zones of security (different people access different resources)
- Log everything (accesses/modifications)

Bring in security experts early for riskier situations

Top vulnerabilities

From OWASP (more at <https://owasp.org/www-project-top-ten/>)

- Broken authentication + access control
- Cryptographic failures
- Code injection
- Weakly protected sensitive data
- Using components with known vulnerabilities

Sample vulnerabilities

- Untrusted environments
- Untrusted Inputs
- Bad authentication
- Malicious software

Recurring theme: No silver bullet

Untrusted environment

Should authentication code in a web application run in the browser?

```
function checkPassword(pwd: string) {  
  return pwd === 'letmein'  
}
```

Untrusted environment

Curses! Foiled Again!



Frontend

Users might be malicious

Trust boundary

```
function checkPassword(pwd: string) {  
    return pwd === 'letmein'  
}
```

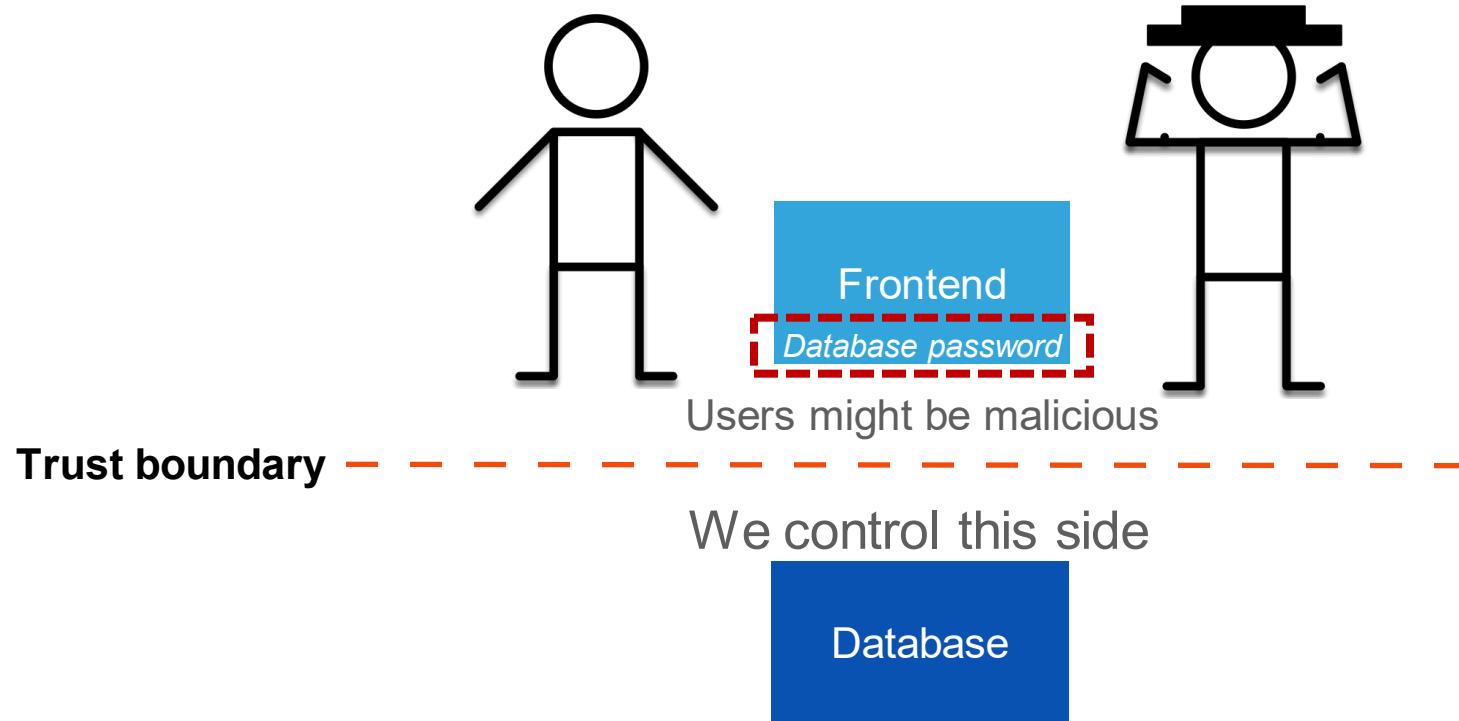
We control this side

Backend

Fix: Move code to
back end (duh!)

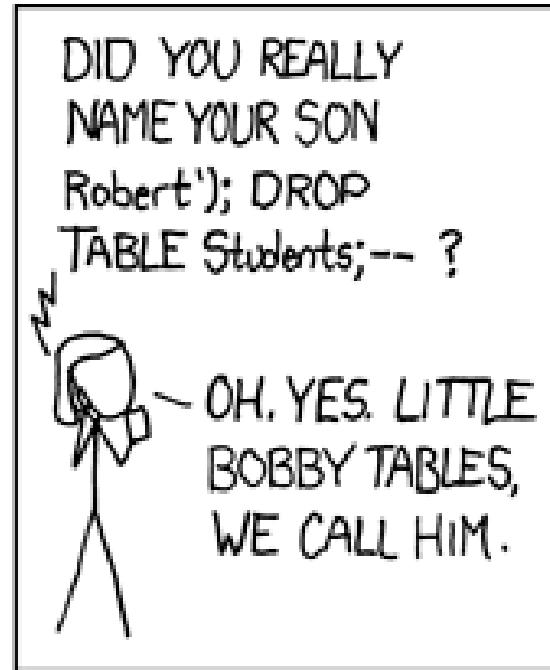
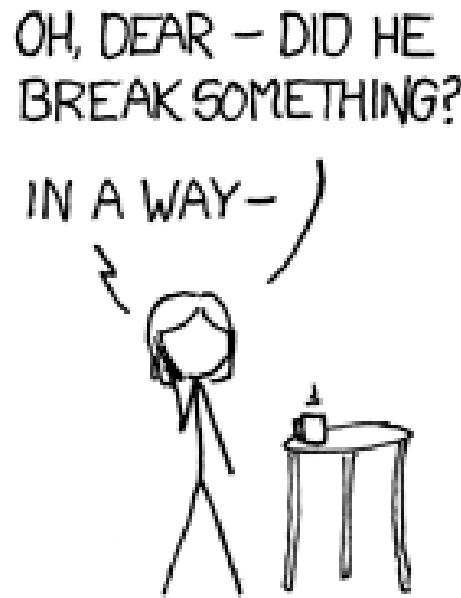
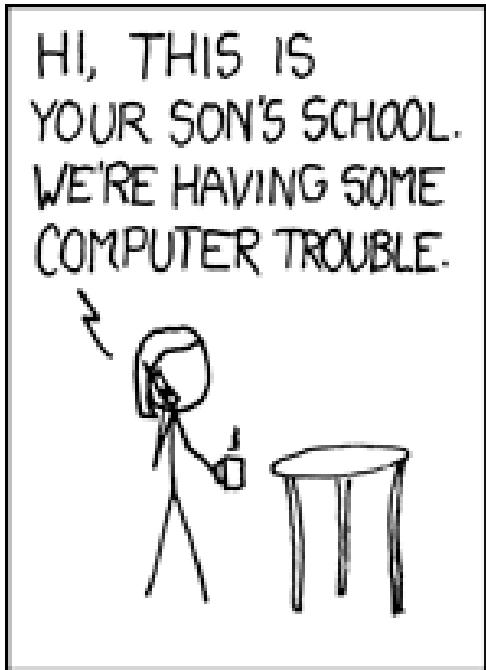
Untrusted environment

Access controls to database



Fix: Don't distribute sensitive credentials

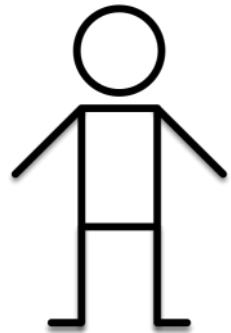
Untrusted inputs



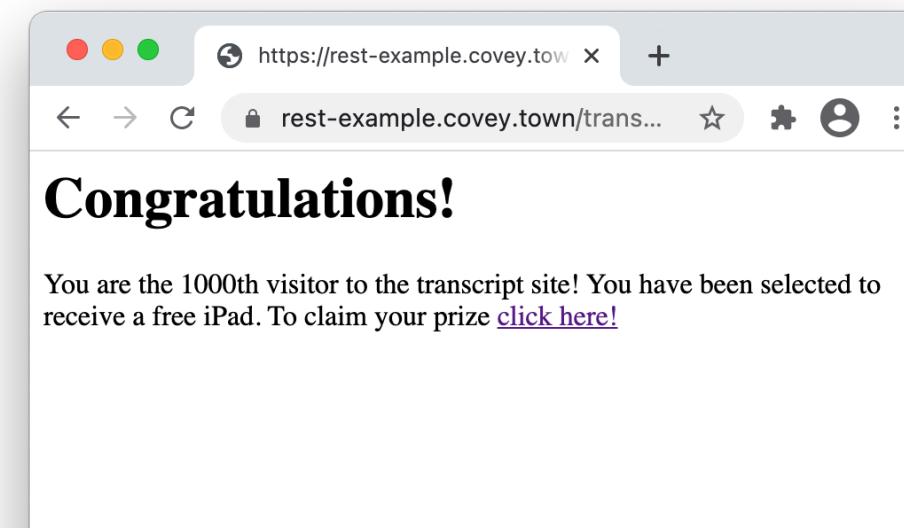
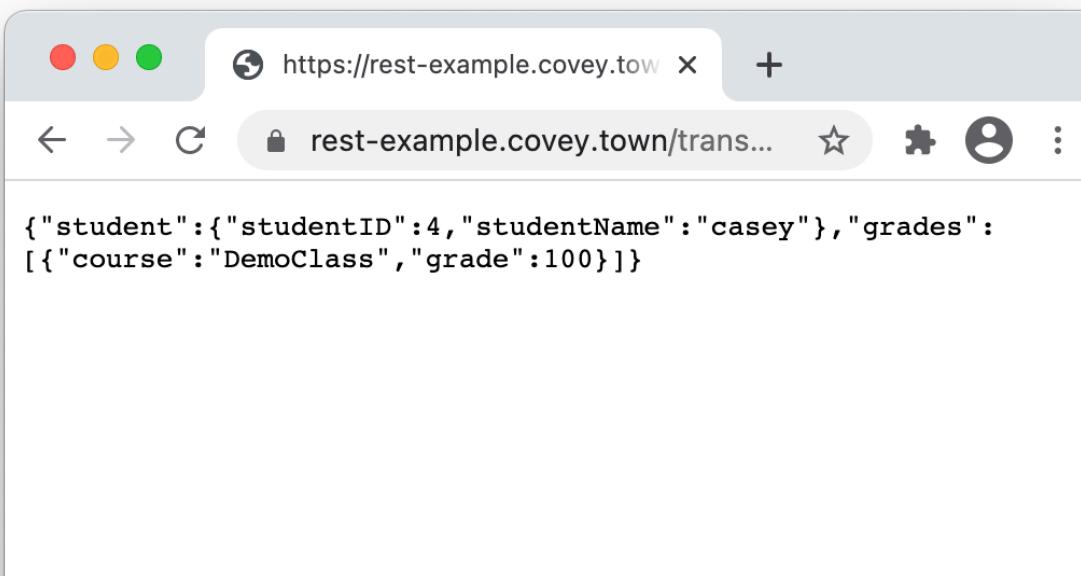
<https://xkcd.com/327>

Untrusted inputs

Cross-site scripting (XSS) vulnerability



Malicious
JavaScript
Response



Untrusted inputs

Cross-site scripting (XSS) vulnerability



/transcripts/4

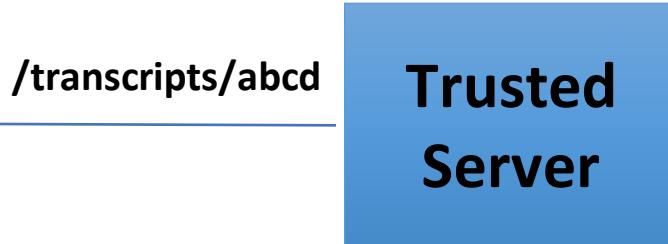


```
app.get('/transcripts/:id', (req, res) => {
  const {id} = req.params;
  const transcript = db.getTranscript(parseInt(id));
  if (transcript === undefined) {
    res.status(404).send(` ${id} is invalid student id`);
  } else {
    res.status(200).send(transcript);
  }
});
```

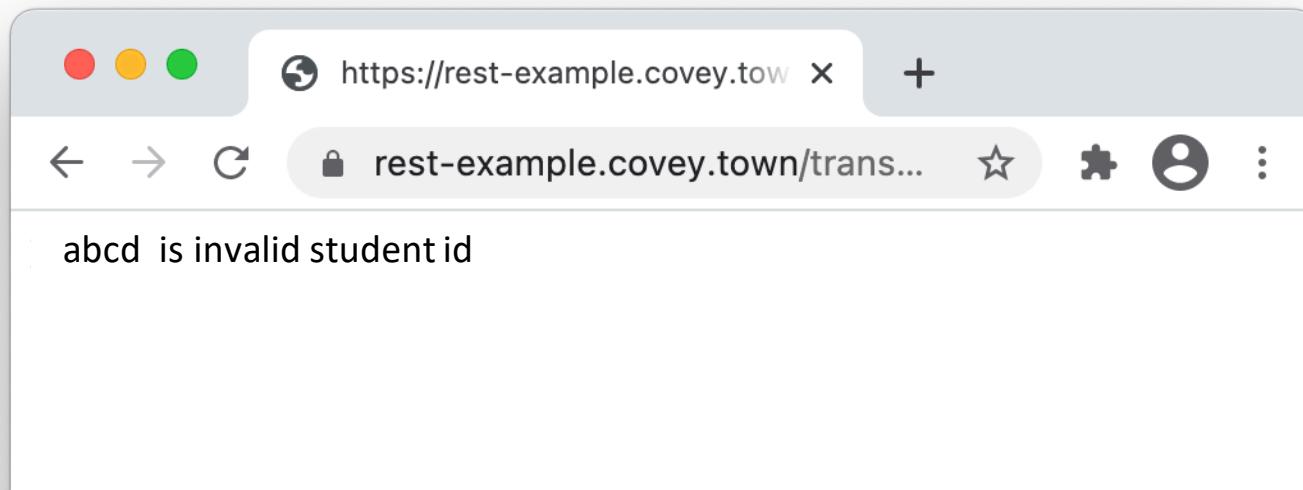


Untrusted inputs

Cross-site scripting (XSS) vulnerability



```
app.get('/transcripts/:id', (req, res) => {
  const {id} = req.params;
  const transcript = db.getTranscript(parseInt(id));
  if (transcript === undefined) {
    res.status(404).send(` ${id} is invalid student id`);
  } else {
    res.status(200).send(transcript);
  }
});
```



Untrusted inputs

Cross-site scripting (XSS) vulnerability



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!](#)

```
app.get('/transcripts/:id', (req, res) => {
  const {id} = req.params;
  const transcript = db.getTranscript(parseInt(id));
  if (transcript === undefined) {
    res.status(404).send(`#${id} is invalid student id`);
  } else {
    res.status(200).send(transcript);
  }
});
```

<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!</a>';
alert('You are a winner!');
```

Untrusted inputs

Java code injection vulnerability in Apache Struts (@Equifax)



English

[Return to equifax.com ▶](#)

2017 Cybersecurity Incident & Important Consumer Information

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.

NEWS

Equifax Says Cybersecurity Breach Has Cost \$1.4 Billion

EMMA HURT • MAY 10, 2019



Untrusted inputs

Java code injection vulnerability in Log4J

Extremely Critical Log4J Vulnerability

Leaves Much of the Internet at Risk

December 10, 2021 Ravie Lakshmanan

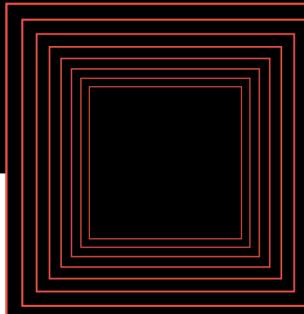


Mar 8, 2022

APT41 COMPROMISED SIX U.S. STATE GOVERNMENT NETWORKS

By Lindsey O'Donnell-Welch

Share



CVE-2021-44228 Detail

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

Untrusted inputs

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

Characters like
<, >, ‘, “ and `
often involved in
untrusted input
exploits

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).

Fix: Always use input validation

Untrusted inputs

Sanitize inputs – prevent them from being executable

Avoid languages 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 like OGNL in Java)
- Languages that can construct arbitrary pointers or write beyond end of array

Untrusted input

Mitigate code injection with static analysis

- 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)
- (how to get engineers to actually do this?)

1 path available
Reflected cross-site scripting

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 vulnerabilities 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 main content area with tabs for Alerts (16), Logs, Files (selected), History, Compare, Integrations, and Queries. A note says "Sonic is joining GitHub". The "Files" tab displays 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'." Below this is an "Alert filters" section with dropdowns for Severity, Query, Tag, and checkboxes for "Show excluded files" and "Show heatmap". An "Export alerts" button is also present. The main content area shows a "Source root /" folder structure with three items: "public" (0 alerts, 0 lines of code), "src" (16 alerts, 756 lines of code), and "package.json" (0 alerts, 0 lines of code). To the left of the dashboard is a large orange donut chart.

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.

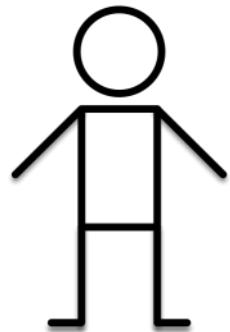
Code injection

Interpreting unsanitized user input as code allows a malicious user arbitrary code execution.

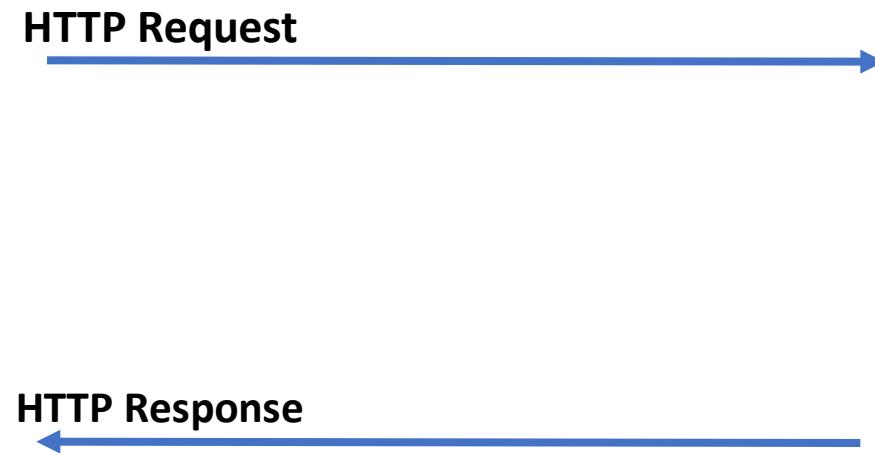
Download of sensitive file through insecure connection

Downloading executables and other sensitive files over an insecure connection opens up for potential man-in-the-middle attacks.

Bad authentication



client page
(the “user”)



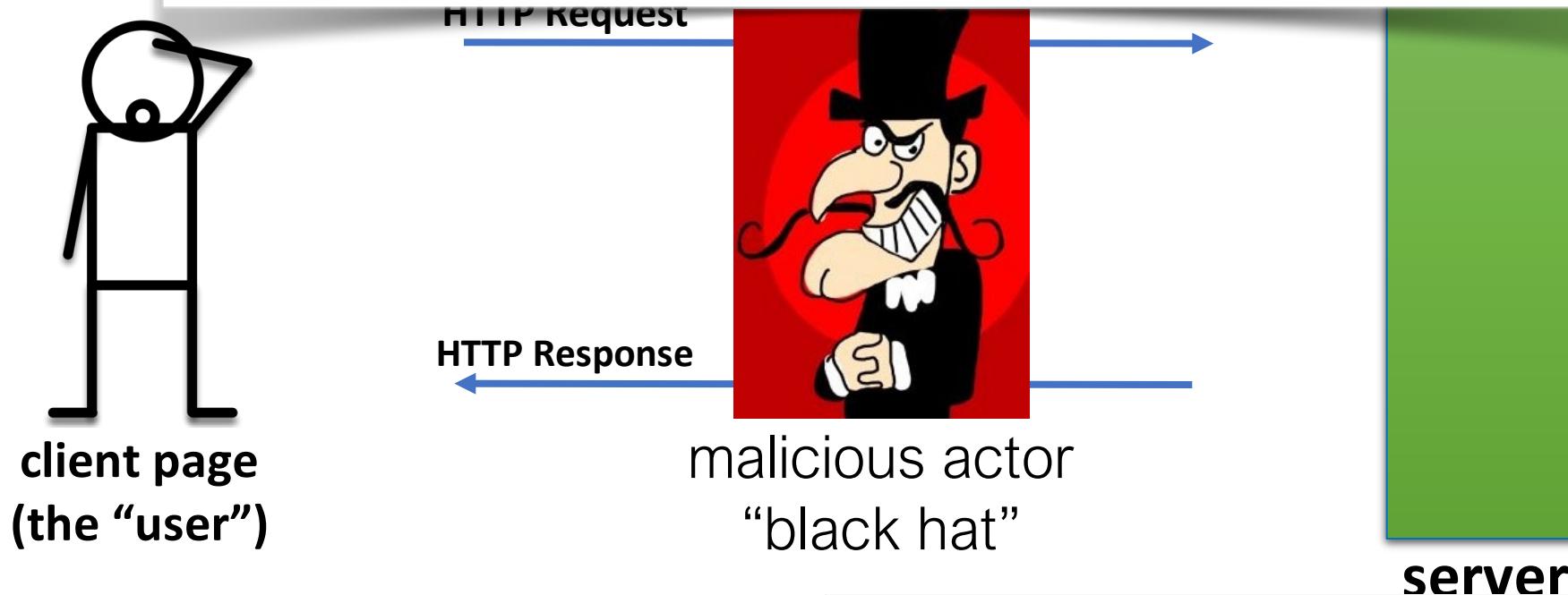
server

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

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

Bad auth

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

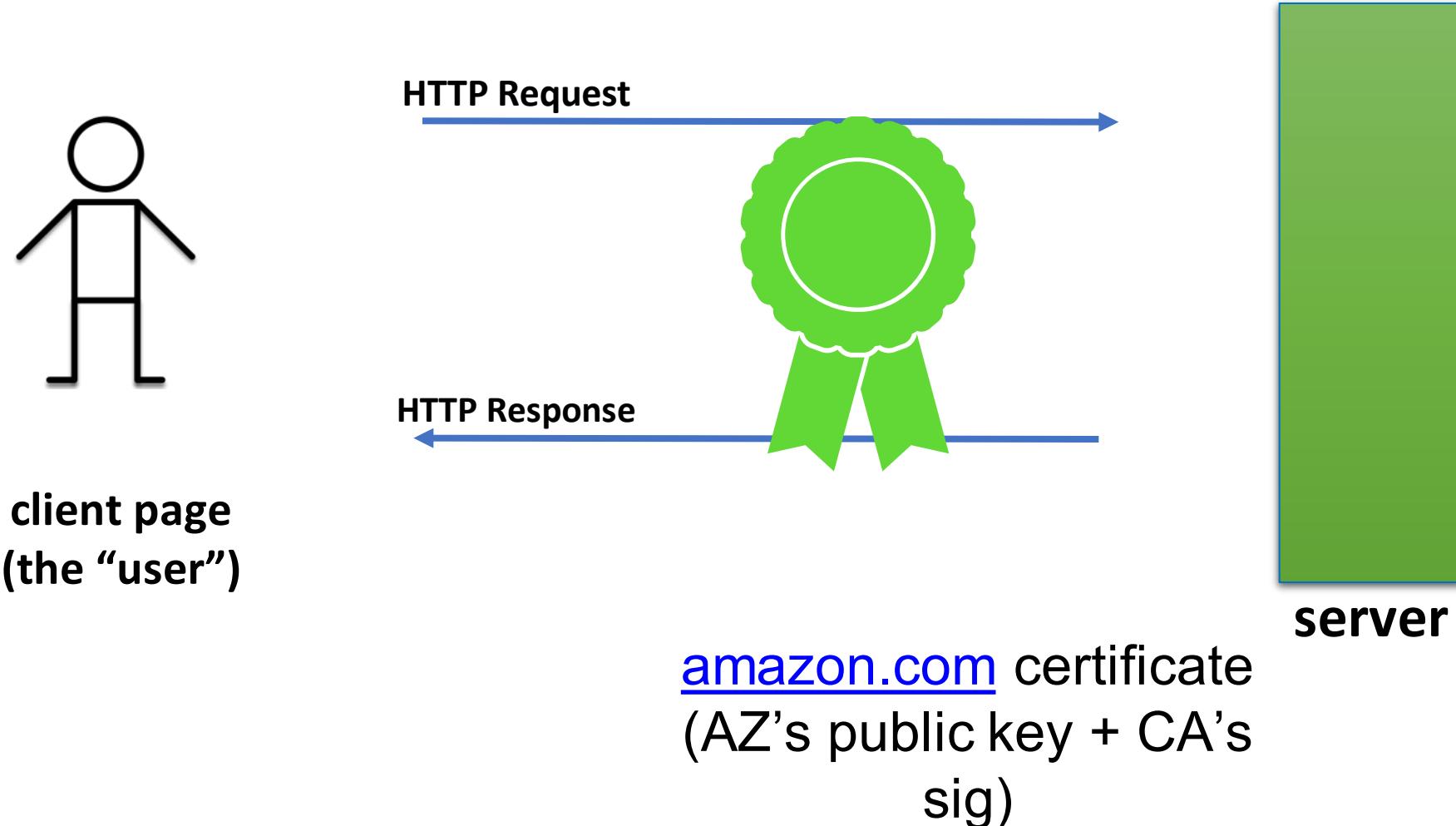


Do I trust that this response
really came from the server?

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

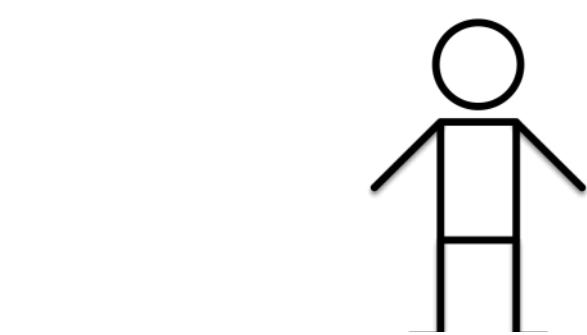
Bad authentication

Preventing the man-in-the-middle with SSL



Bad authentication

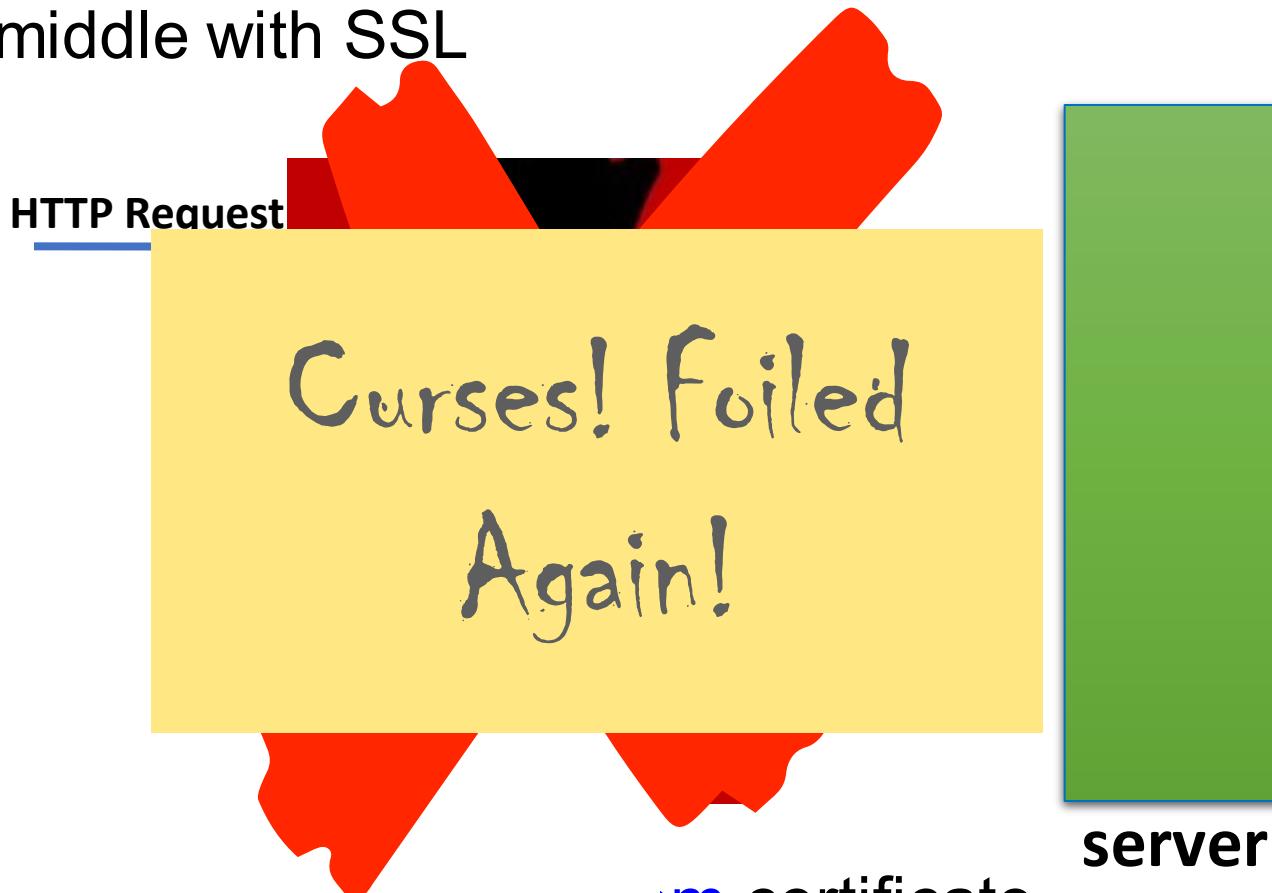
Preventing the man-in-the-middle with SSL



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



im certificate
c key + CA's
sig)

server

Bad authentication

Certificate authorities

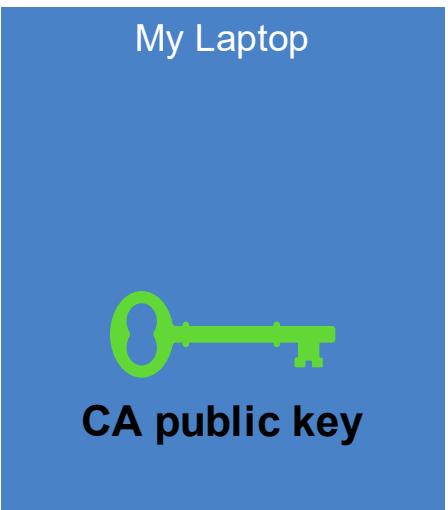
- A certificate authority (or CA) binds some public key to a real-world entity that we might be familiar with
- CA is the clearinghouse that verifies that amazon.com is truly amazon
- CA creates a certificate that binds amazon.com's public key to the CA's public key (signing it using CA's private key)

Bad authentication

Certificate Authorities issue SSL Certificates



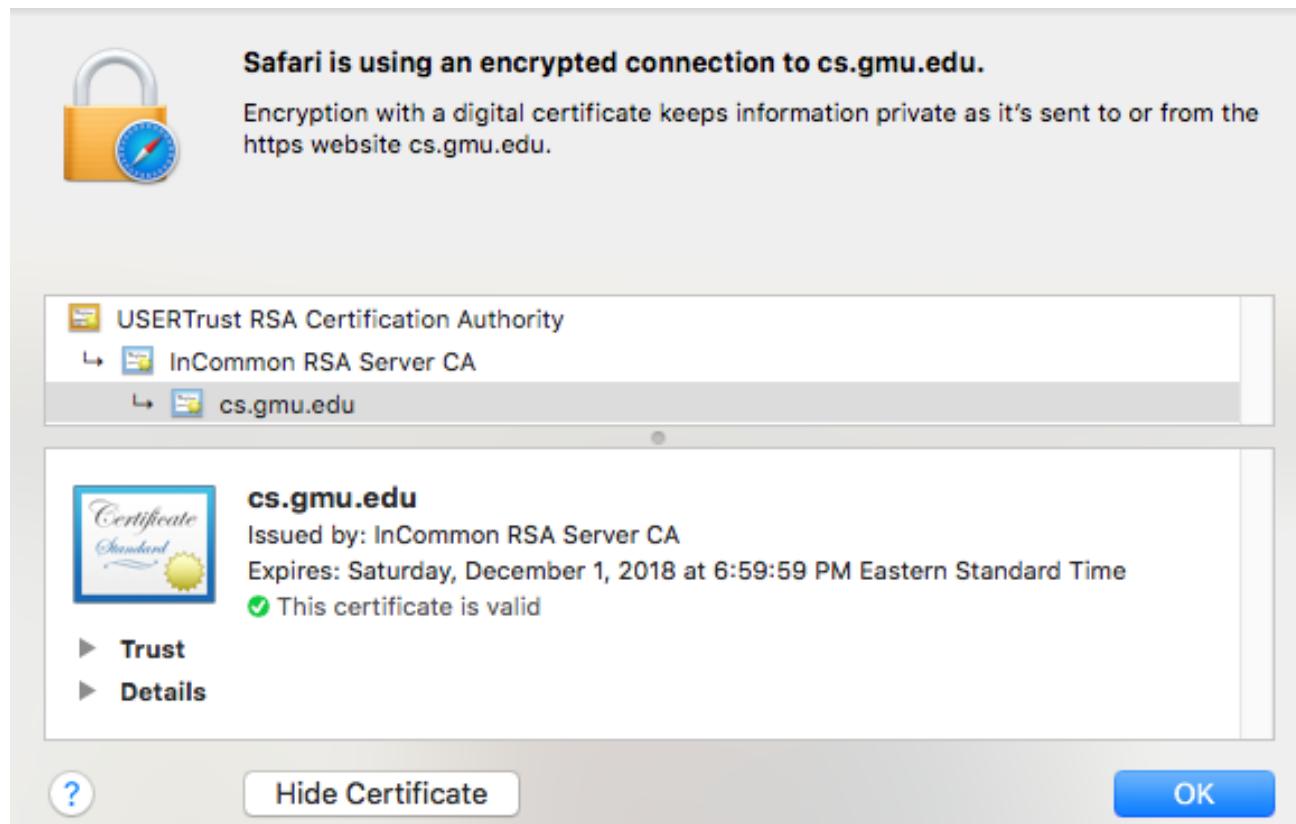
amazon.com certificate
(AZ's public key + CA's sig)



Bad authentication

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



Bad authentication

Should CA 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



Gooale has read the riot act to Svmantec. scolding the security biz for its

Bad authentication

You can do this for your website for free with letsencrypt.com



The screenshot shows the official website for Let's Encrypt. At the top left is the logo, which consists of a yellow lock icon with three orange radiating lines above it, followed by the text "Let's Encrypt". To the right of the logo are five navigation links: "Documentation", "Get Help", "Donate", "About Us", and "Languages" with a small gear icon. Below the header is a large, semi-transparent dark overlay containing white text. The text reads: "A nonprofit Certificate Authority providing TLS certificates to **300 million** websites." Below this, in smaller text, is the sentence: "We were awarded the Levchin Prize for Real-World Cryptography! Learn more". At the bottom of the overlay are two blue rectangular buttons with rounded corners, both containing white text: "Get Started" on the left and "Sponsor" on the right.

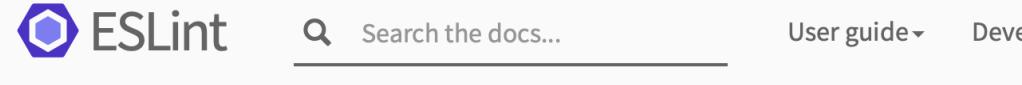
A nonprofit Certificate Authority providing TLS certificates to **300 million** websites.

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

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

Malicious software

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.

<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

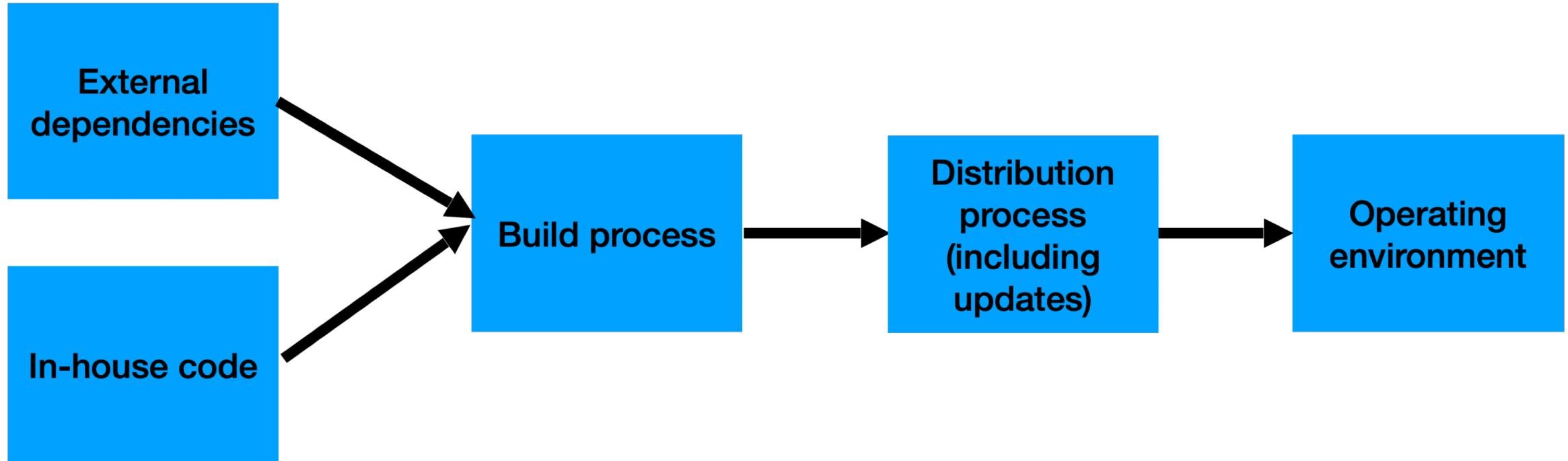
By Nilay Patel | @reckless | Jan 26, 2021, 9:13am EST

SHARE

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

Malicious software

The software supply chain has many points of weakness



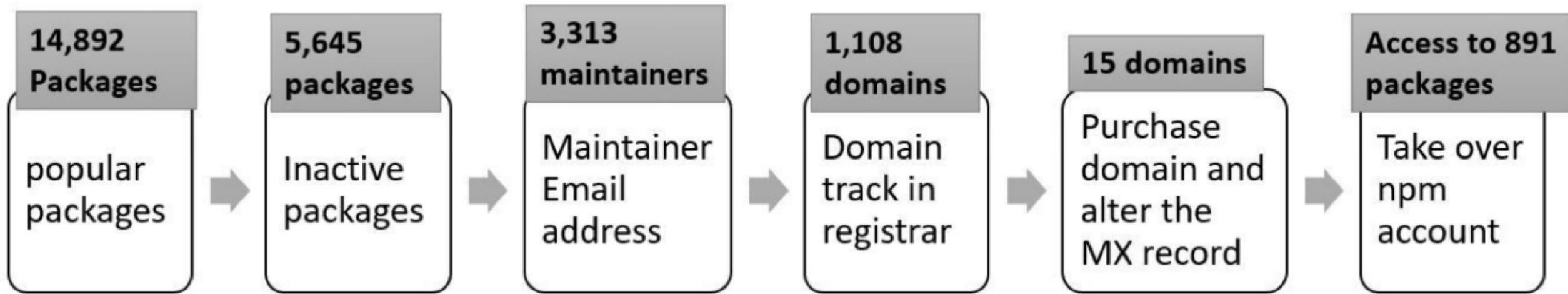
Malicious software

Common vulnerabilities in top 1% of npm packages from a 2021 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 contributors to make effective code control
- Maintainers are maintaining too many packages
- Many statistics/combinations: see the paper for details.

Malicious software

A possible attack...



Malicious software

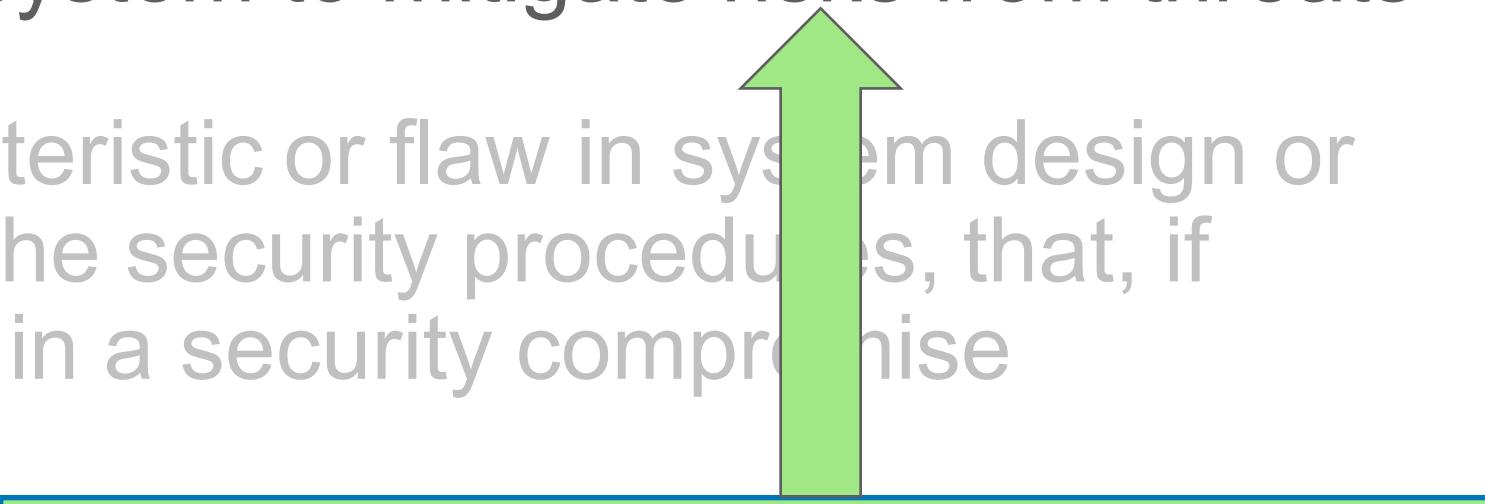
How can we mitigate threats from the software supply chain?

Process-based solutions for process-based problems

- External dependencies
 - Audit all dependencies and their updates before applying them
- In-house code
 - Require devs to sign code at commit; 2FA for signing keys, rotate 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 or requirement
- Attack: realization of a threat



It's a management problem!!

Which to protect against, at what cost?

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
- Rely on a trusted component

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

Auth0

The Auth0 interface shows a mobile application icon and a user profile icon. A blue line connects them to a tree diagram representing access roles. The tree has 'Access role' at the root, which branches into 'Customer' (marked with a green checkmark) and 'Read only' (marked with a red X). The 'Customer' node further branches into 'Publish' (marked with a green checkmark) and 'Read only' (marked with a red X).

It's a management problem!!

<https://auth0.com>

Cryptographic Failures

OWASP #1

- 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)

Ama
Total candidates 1,241
Valid candidates 308
Unique % valid 93.5%

Table 5: Credentials statistics from June 22, 2013 and validated on November 11, 2013. A credential may consist of an AWS access key and secret access key, or a session token and temporary credentials.

The screenshot shows a web-based search interface for decompiled Android source code. At the top, there's a search bar with placeholder text "AKIA*" and a "Search" button. Below the search bar are filters for "Line filter (Ruby regex, optional)" and "10 files per page". The main area displays a table with three columns: "Android Package", "Path", and "Line". The table contains several rows of code snippets, such as:

Android Package	Path	Line
com.google.android.gms	AppConst.java	public static final String AMAZON_KEY_ID = "AKIA...";
com.google.android.gms	SongManager.java	BasicAWSCredentials localBasicAWSCredentials = new BasicAWSCredentials("AKIA...");
com.google.android.gms	Shoutcast.java	("AWSAccessKeyId=AKIA...").append("AssociateTag=mariuiorda-

Below the table, there are navigation links: "← Previous 1 2 3 4 5 6 7 8 9 ... 41 42 Next →". To the right of the table, there's a large green box containing the text "It's a management problem!!".

Figure 9: PLAYDRONE’s web interface to search decompiled sources showing Amazon Web Service tokens found in 130 m

“A Measurement Study of Google Play,” Viennot et al, SIGMETRICS ‘14

Do we pay attention? Should we?

Industrial study of secret detection tool in a company with >1K 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”

Is it a management problem or a tool problem?

Learning objectives

By now, you should be able to...

- Define key terms relating to security
- Describe tradeoffs between security and other SE requirements
- Explain common vulnerabilities in web apps, and mitigations
- Explain why software alone isn't enough to assure security