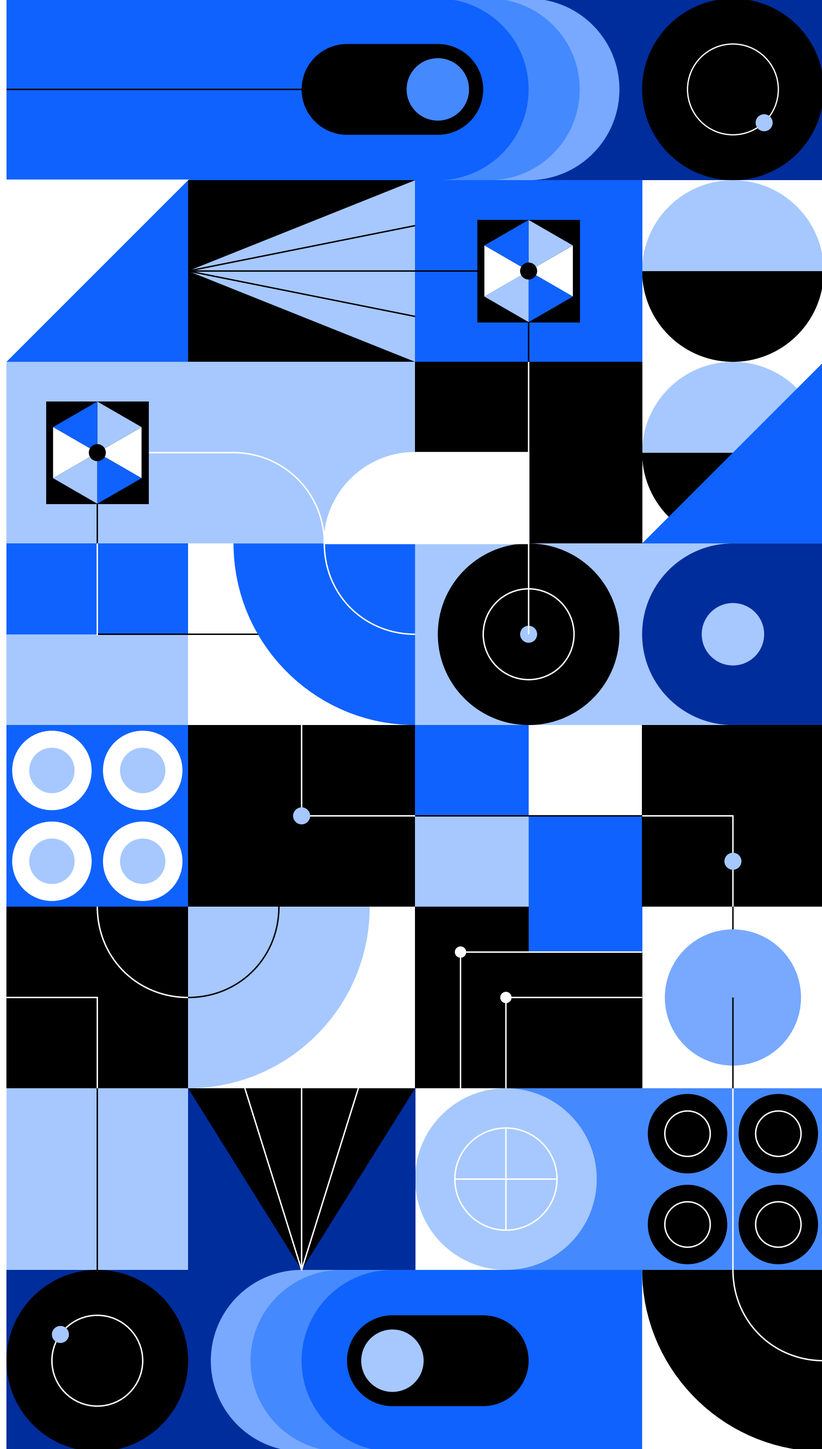
# IBM Accelerate Software Developer Track

Wednesdays June 5th – July 24th
6:00pm-8:00pm Eastern Time



# Help us identify our Rock Stars!

## How?

- Complete the nomination form: https://ibm.biz/accelerate-sw-rockstars

## Who can submit a nomination?

- Students
- Coaches
- Instructors
- Coordinators
- Everyone!

• Optional practice test

# IBM Accelerate Software Developer Track

## Wednesday, July 17, 2024

## Security

Today’s speakers:

**Shravan Kumar Raghu**
Software Architect,
IBM Cloud Storage, IBM Infrastructure

**Ope Jegede**
STSM, ServiceNow Solutions Architect,
IBM Cloud,
IBM Infrastructure


**Shidong Shan**
Software Developer,
Security Guardium,
IBM Software

# Session Agenda

| Timing | Topic | Speaker | Content |
|---|---|---|---|
| 6:00-6:05pm | Opening | Sujeily Fonseca | Welcome |
| 6:05-6:25pm | Introduction to Application Security | Shravan Raghu | Threat Landscape<br>Industry-specific drivers for Security<br>Social Engineering<br>Seeing threats in a new light<br>Data leaks<br>Security approaches for software development<br>Threat Modeling process |
| 6:25-6:40pm | Web Application Vulnerability | Ope Jegede | Top 10 Web App Security Risks<br>Authentication vs. Authorization<br>Input Validation<br>HTML Injection and Cross Site Scripting<br>Insecure Direct Object Reference<br>Role-Based Access Control (RBAC) |
| 6:40-6:55pm | Secure Software Development | Shidong Shan | Securing API Calls<br>Security Vulnerabilities in Dependencies |
| 6:55-7:00pm | Closing and Breakout | Shidong Shan | Questions and Answers<br>Lab 7 - To-do List<br>HackerRank Quiz |

# Evolving threat landscape

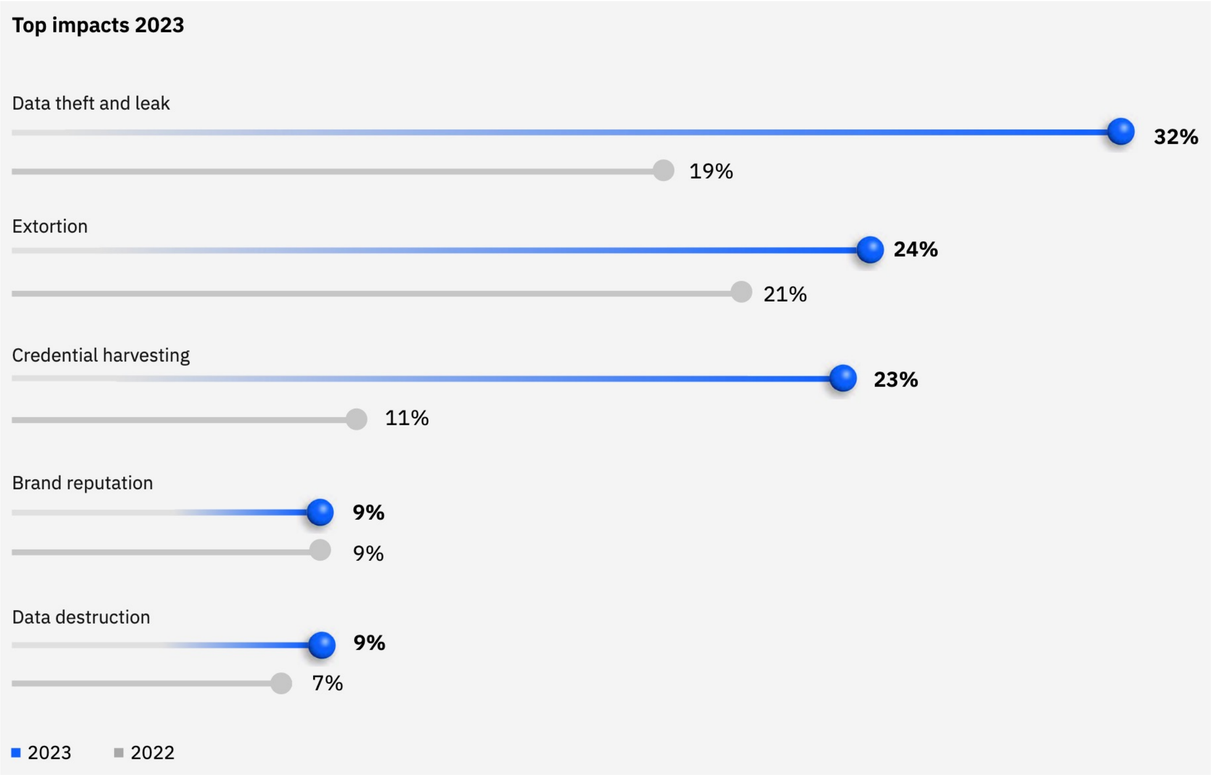
Cybercriminals remain adept at successfully infiltrating organizations across the globe

**71%**
Percentage increase year-to-year in attacks using valid credentials

**32%**
Incidents involving data theft and leak

**82%**
Share of breaches that involved data stored in cloud environments

**50%**
Market share threshold likely to trigger attacks against AI platforms




**84%**
Percentage of critical infrastructure incidents where initial access vector could have been mitigated

**$332M**
Average cost of a mega data breach (50-60M records)



Sources: 2024 IBM X-Force Threat Intelligence Index; 2023 IBM Security Cost of a Data Breach Report

# Industry-specific drivers for Security

## Financial services industry

- Focus on regulatory compliance
- Ensure cloud workloads meet new security standards
- Increase investments for “right side of the boom”
- Adopt standard security frameworks and controls
- Design integrated risk, compliance, and security analytics

## Energy and utilities

- Address risks for IT, OT, and IoT
- Harmonize the approach to IT-OT convergence
- Maintain regulatory compliance
- Increase security efficiency for detection to response
- Enable and protect digital transformation

## Government and public sector

- Lack of sufficient skills and administration
- Meet the demands of digital transformation
- Ensure data privacy and compliance
- Mitigate insider threats
- Improve real-time visibility and continuous monitoring

## Retail & consumer products

- Address increasing cyber attack vectors, including IoT
- Meet the demands of digital transformation
- Maintain regulatory compliance
- Adopt mobile tech. and leverage analytics
- Exceed client expectations

## Telecom, media, & entertainment

- Ensure client privacy
- Maintain regulatory compliance
- Meet the demands of digital transformation
- Address the rise in threats and attacks
- Adopt new technologies and IoT

## Healthcare industry

- Ensure patient privacy, safety, and security
- Secure medical devices, sensors, and IoT endpoints
- Meet the demands of digital transformation
- Maintain regulatory compliance
- Secure medical images

# Social Engineering

A manipulation technique that exploits human error to gain private information or access

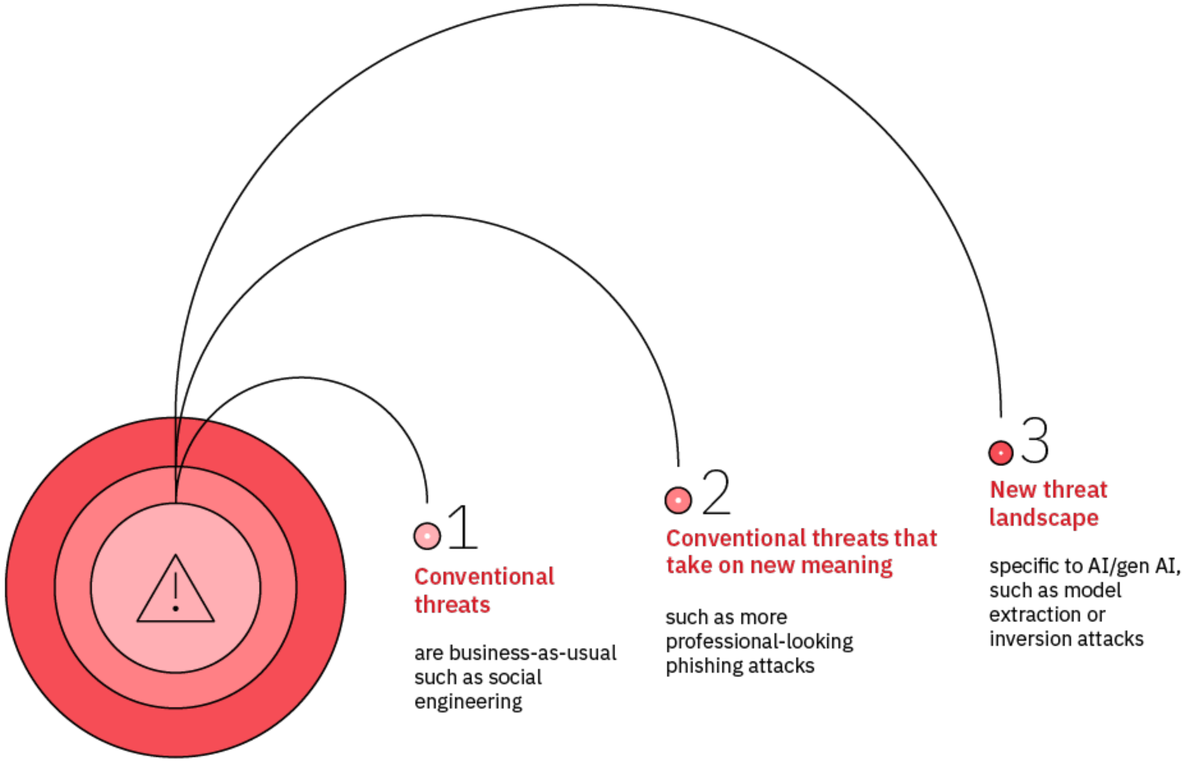
- One of the most popular social engineering attack types: **phishing** scams are email and text message campaigns aimed at creating a sense of urgency
- **Verbal scam**: another popular technique in which an attacker claims to be someone else (customer, high level employee, etc.)
  - Often uses fear or angry claims to get the victim to comply with demands.

Example: victim gets a phone call from someone claiming to be a Vice President in their company and demands the victim act – for example, turn off authentication or transfer funds into an account.

# Seeing threats in a new light

Take phishing emails as an example. With gen AI, cybercriminals can create far more effective, targeted scams—at scale.

IBM Security teams have found gen AI capabilities facilitate upwards of a 99.5% reduction in the time needed to craft an effective phishing email – from 16 hours to 5 minutes.




Source: IBM Institute for Business Value

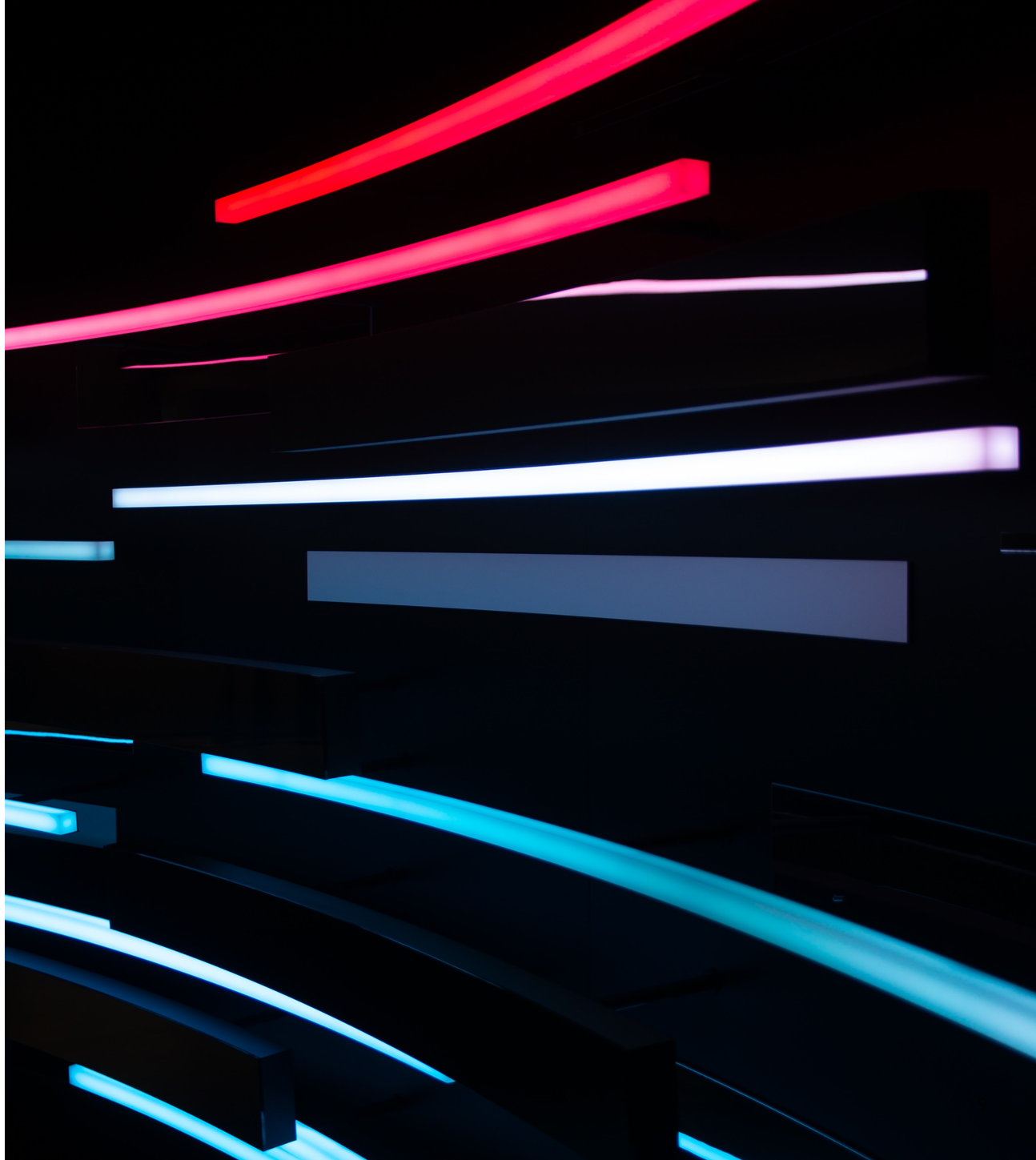
# Data Leaks!

Hackers profit at the cost of others by doing illegal activities – usually with DATA.

- Misusing Data: identity theft.
  - Loans, fake accounts, financial fraud.
- Selling Data
  - Underground marketplaces
- Giving Data away for free
  - Prove skills, gain trust of other hackers

What can be done with data from a data breach?

- Databases containing user / password information can be used in a **credential stuffing attack**
  - Credential stuffing attack: using stolen data to gain access to other accounts / services



# Software has a problem!

“75% of hacks happen at the application.”

- Theresa Lanowitz, Gartner Inc.

“92% of reported vulnerabilities are in apps, not networks.”

- NIST

“64% of developers are not confident in their ability to write secure code.”

- Bill Gates

# As a Developer, why should I care?

- Makes our jobs harder!
- Secure coding keeps our customers safe.
- Secure coding skills will make **you valuable** in the marketplace!
- **Simple steps to make the work you do more secure**

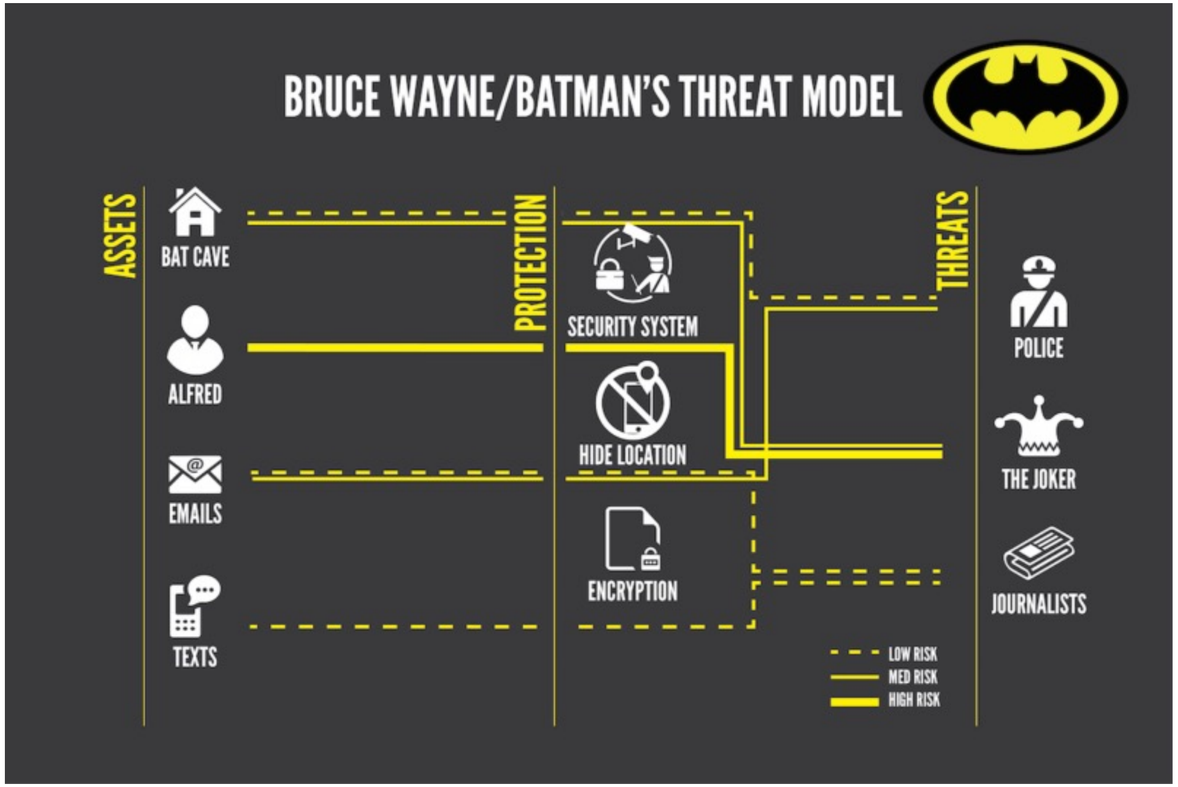
*(and save you time when your product gets tested)*

# Secure Software Development

- Threat Modeling

# What is a threat model?

- A structured representation of all the information that affects the security of an application. In essence, it is a view of the application and its environment through the lens of security. (OWASP)




*Source: https://arstechnica.com/information-technology/2017/07/how-i-learned-to-stop-worrying-mostly-and-love-my-threat-model/*

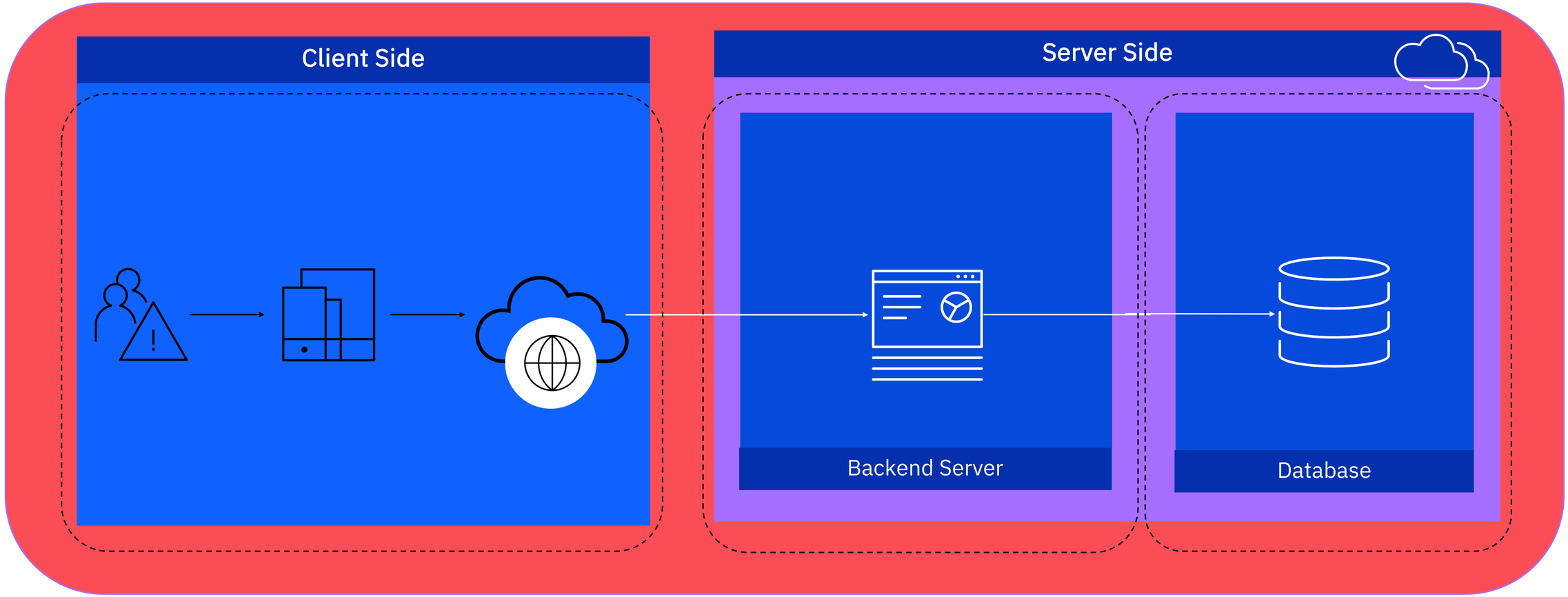
Includes:

- Description of the subject to be modeled
- Assumptions that can be checked or challenged in the future
- Potential threats to the system
- Actions that can be taken to mitigate each threat
- A way of validating the model and threats, and verification of success of actions taken

# Threat modeling process

| Visualize | Identify Threats | Manage/Mitigate Risk | Assess |
| --- | --- | --- | --- |
| What are you building? | What can go wrong? | What should you do about those things that can go wrong? | Did you do a good job? |

# Visualize

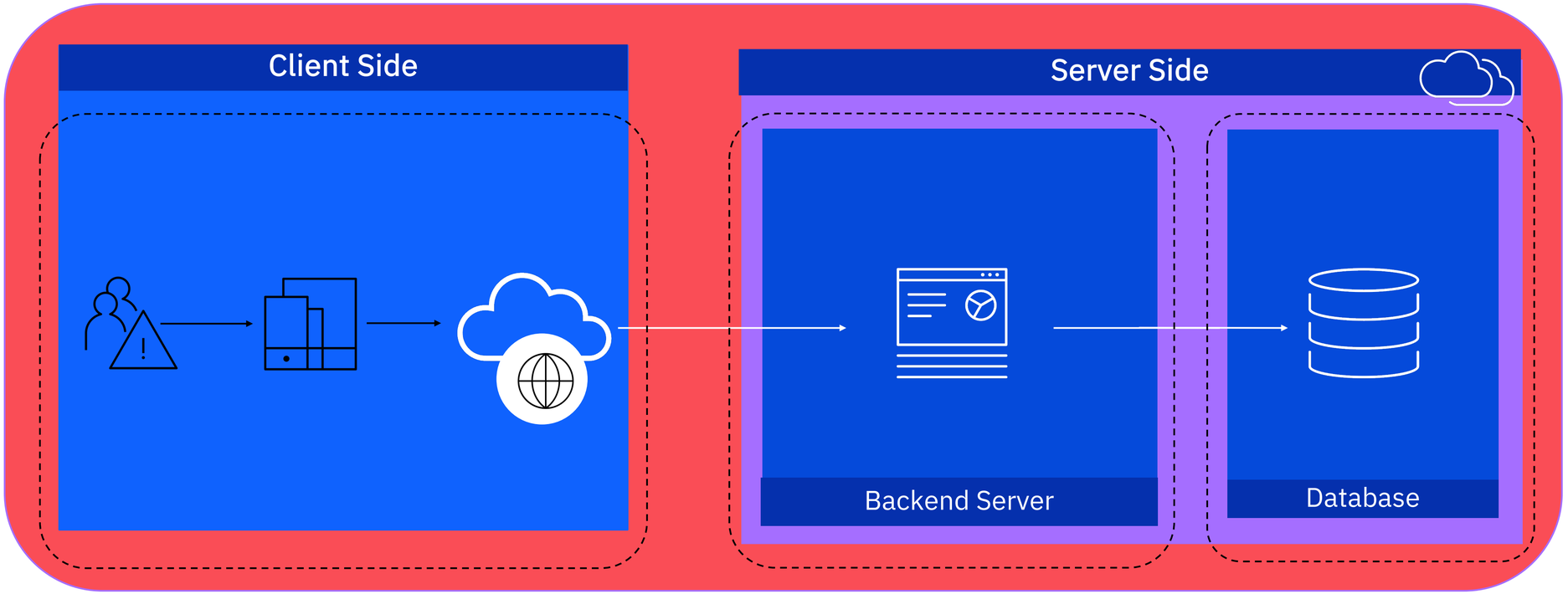


# Identify Threats

**Client Side**
1. SQL Injection
2. Unauthorized User
3. Insecure Protocol

**Backend Server**
1. Unauthorized User
2. Unprotected APIs

**Database**
1. Unauthorized Access
2. Excessive access



# Manage/Mitigate Risk

**Client Side**

1. SQL Injection → Add input validation to login.
2. Unauthorized User → Add user accounts and page-privileges.
3. Insecure Protocol → Enable communication over HTTPS instead of HTTP.

**Backend Server**

1. Unauthorized User → Secure version control and deployments.
2. Unprotected APIs → Require authentication for all endpoints.

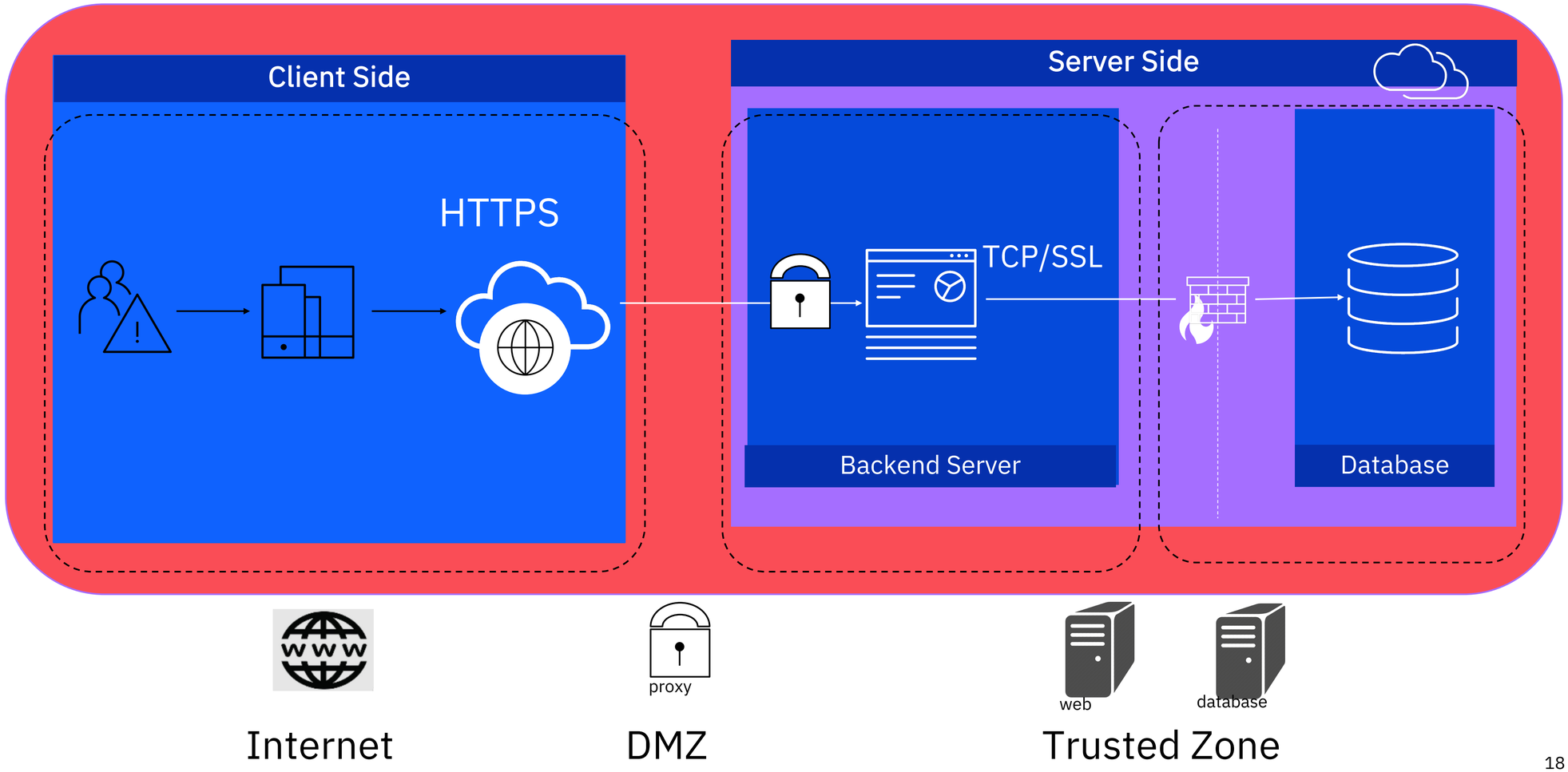
**Database**

1. Unauthorized Access → Securely store credentials within application.
2. Excessive access → Use firewalls + monitoring services.

# Assess

**Is the application:**

- More resilient to potential attacks?
- Adaptable to changes in architecture?
- Still able to perform its function?
- Adaptable to future security requirements?



# Why threat models?

## Proactivity

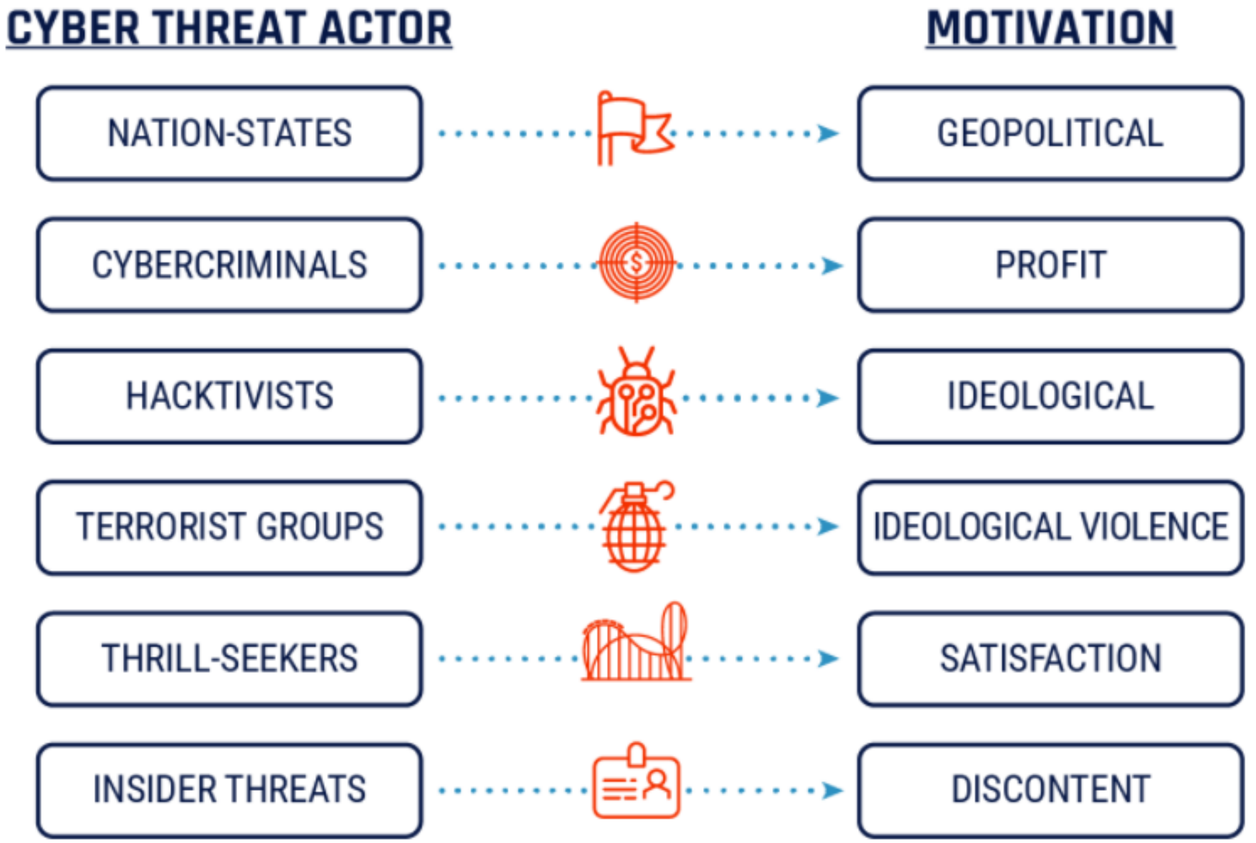
A “line-of-sight” view of a system and its potential weak points.

Make security decisions with high level of certainty by having all information available.

## Natural Assurance

Be able to explain and defend an application’s security.

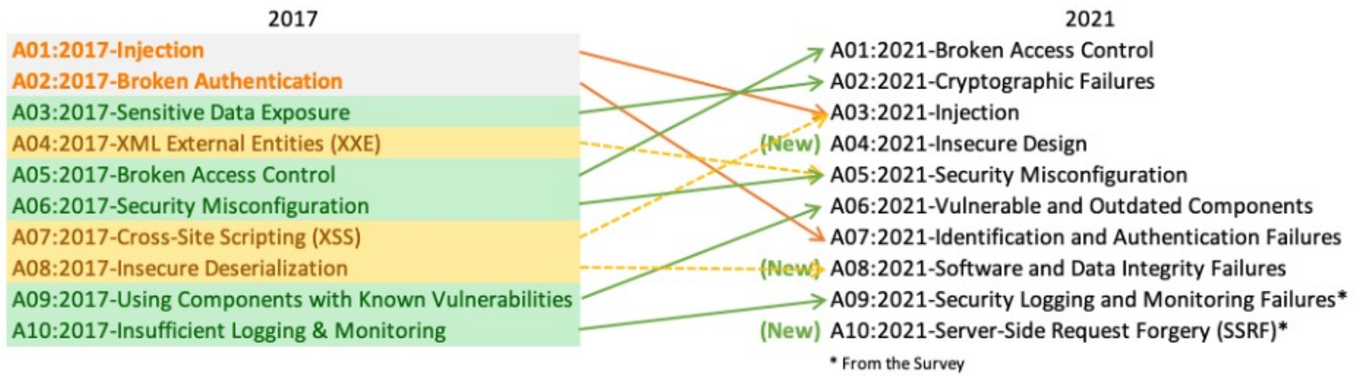
Enforces a secure-by-nature philosophy that can be shared amongst a team.



# Web Application Vulnerability

# Top 10 Web App Security Risks

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated components
7. ID and Auth Failures
8. Software/Data Integrity failures
9. Security Logging/Monitoring Failures
10. SSRF



# Authentication vs. Authorization

## Authentication

- Are you who you say you are?
- Passwords, tokens, MFA, etc.
- Username/Password is only the tip of the iceberg

## Authorization

- What do you have access to?
- Permissions to see a page, download a file, make a change to something, etc.
- Permissions assigned to a specific user, role or privilege level.


**Vulnerability Example:**

If an application is not expiring (timing out) authentication / authorization sessions and they are not limiting certain functions (e.g., failed login attempts), this results in broken authentication and access control.

# Authentication in-depth

Input Validation

- Sanitizing inputs for usage of invalid characters
- SQL injection the most common method of attack

HTTPS

- Protect credentials being sent over a network

Salt + Hash

- Credentials should NEVER be stored in plain text
- Use encryption with added salt to ensure information is secure

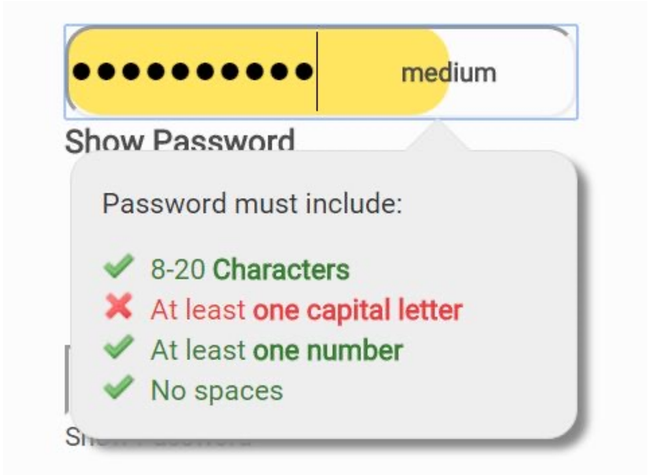
MFA and Auth Providers

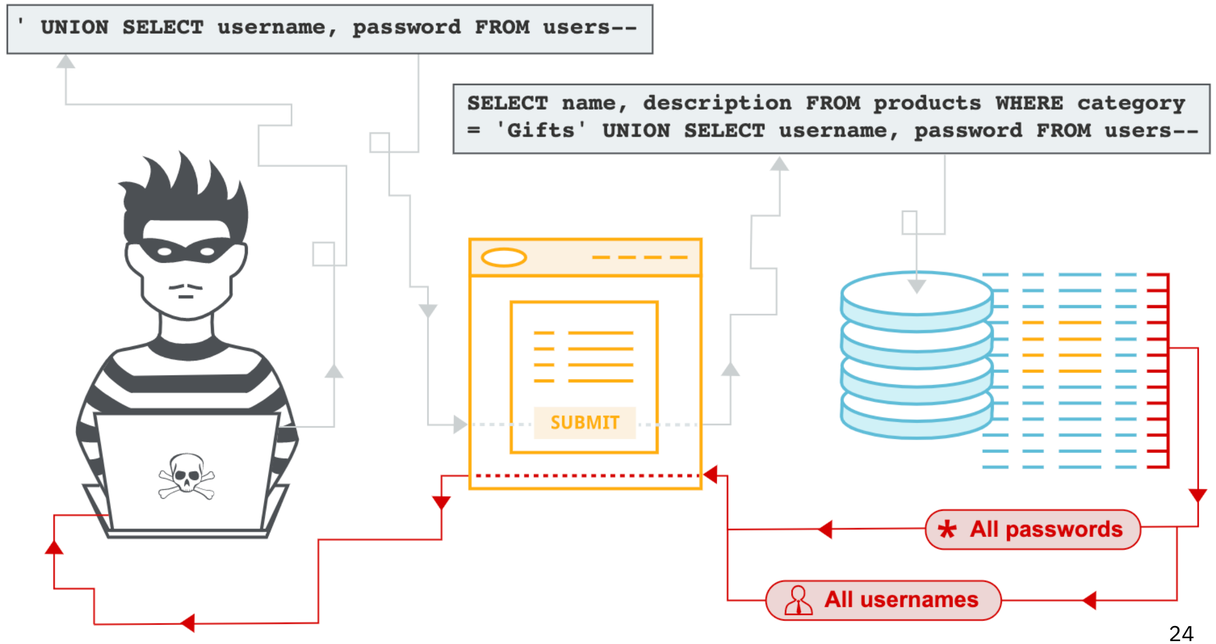
- Use more secure and trusted auth providers
- Wrap application log in and pass token through

# Input Validation

Ensuring only properly formed data  is able to enter an information system

- Filtering out special characters, malicious scripts etc. from user input.
- Provide general errors for logins, not specifically what was input wrong.
- Can be used to prevent downstream workflows from encountering areas (i.e. sending an email to a provided email adjust only if its valid).
- First line of defense for injection attacks where a user tries to execute code via their input.

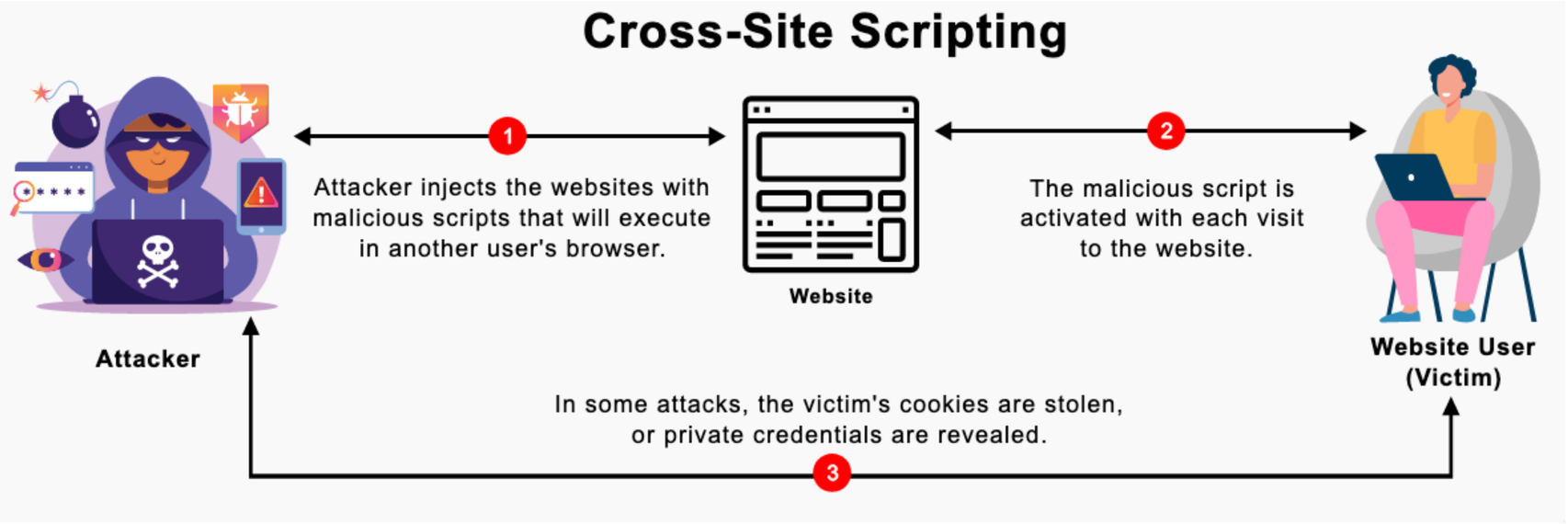


# HTML Injection and Cross Site Scripting

Below are some commonly seen real-world cross site scripting examples that attackers often use:

- User Session Hijacking
- Unauthorized Activities
- Phishing Attack
- Stealing Critical Information
- Capturing Keystrokes

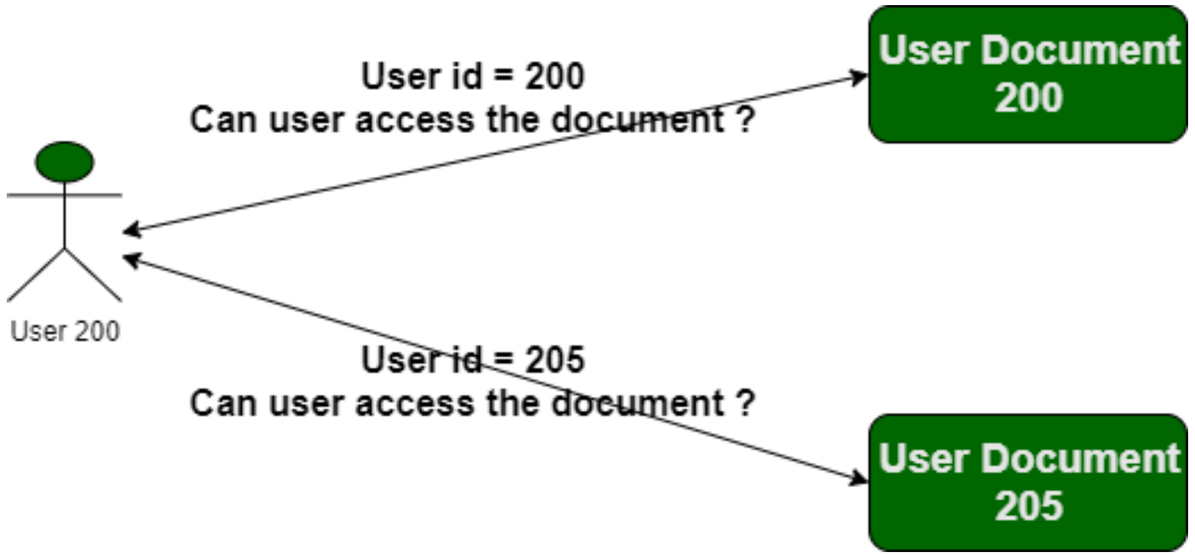



**Adversaries use it to escalate and increase the severity of an attack**

# Insecure Direct Object Reference

## Access Control Vulnerability

- Are users able to access resources without asking for permission?
- Even authenticated users need to be checked for access to certain content.
- Examples:
- Database Object:
  - https://website.com/customer_account?customer_num=132355
  - How do we know this user is this customer?
- File:
  - https://insecure-website.com/static/12144.txt
  - How do we know the user should be allowed do download this?



# Role-Based Access Control (RBAC)

## Deny first

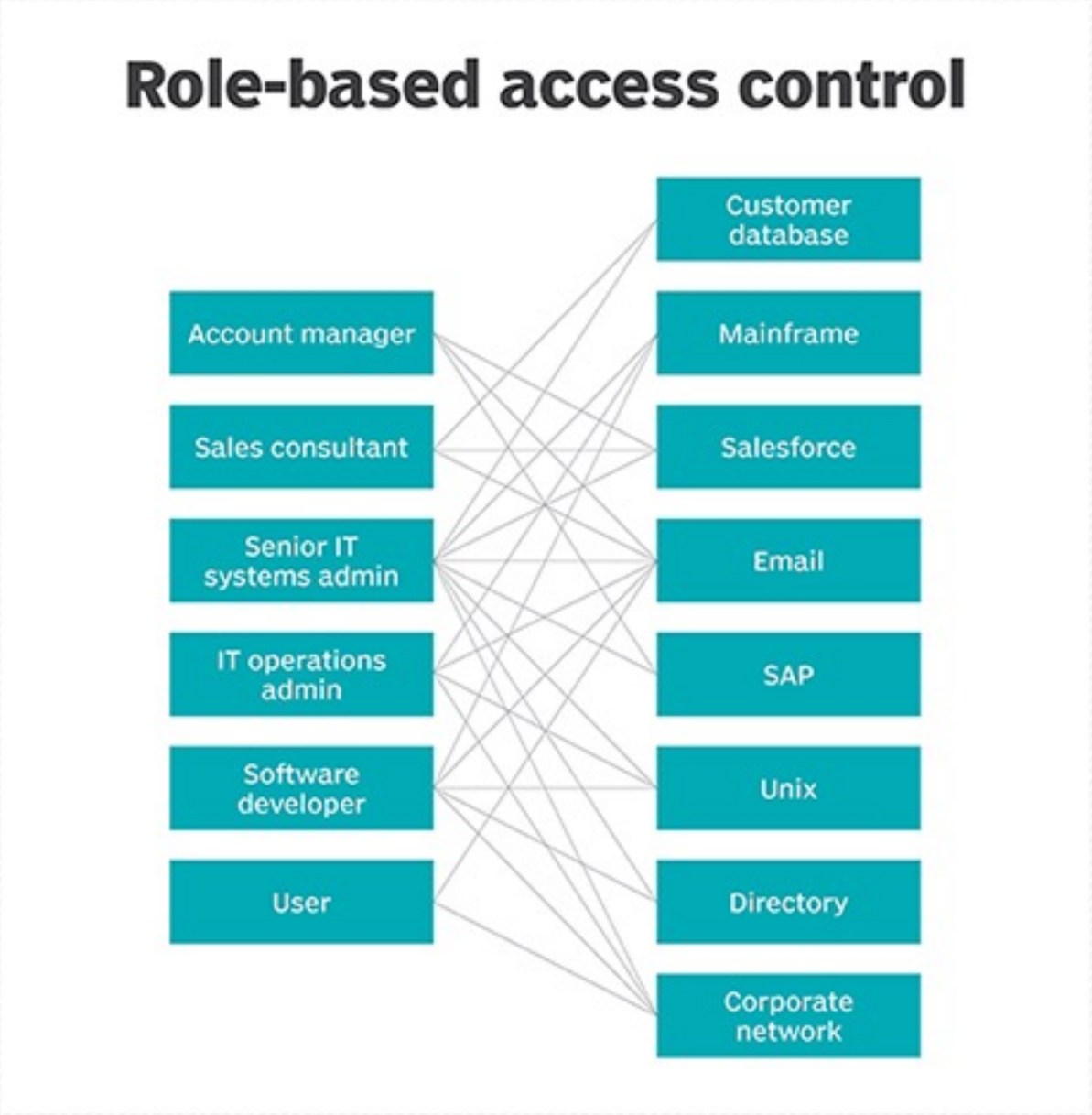
- Closed by default, users need to gain access to resources.
- Resources can be pages, databases, APIs, etc.

## Creating/Assigning roles

- Establish roles by users with similar access needs.
- Avoid creating too many roles

## Why RBAC?

- Easy to manage and modify per-role than per-user
- Makes audits/large organizational changes much simpler.



# Secure Software Development

- Securing API Calls
- Security Vulnerabilities in Dependencies

# Securing API Calls
# Authentication vs. Authorization

**Authentication** – verifying that someone is who they say they are

Example: checking username/password combination

**Authorization** – verifying that an authenticated user has access to certain resources

Example: checking that a user has the necessary permissions to view a page or request certain data

# Securing API Calls
## Authentication Types

### No authentication

Without any authentication controls in place, any user can make requests to your server and get a response.

Public web servers can **never** trust their clients without authentication.

### Basic authentication

An HTTP authentication pattern that accepts a **username** and **password**.

Most web servers like Nginx, Apache HTTP Server, and Internet Information Services (**IIS**) support this.

### API Keys

Authentication strings typically generated for repetitive use by a specific application.

For example, to create a Twitter bot, you'll need to get a Twitter API Key that is attached to all your bot's requests.

### Access Tokens

Secret strings generated on demand for a specific user.

These typically have a short (< 1 day) expiration time, and are used during a single user's login session.

# Securing API Calls

## Authorization Types

### Sessions

A web pattern where after authenticating, the server responds to the client with a secret ID in the response.

The web application will store it as a **cookie** and send it with future requests.

This secret ID can be used to maintain authentication and link server-side state to specific users.

### JSON Web Tokens

JWTs are signed tokens that can contain metadata (such as **claims** to specific data) in them, as they are encoded with a secret generated by the provider.

After authenticating with a service, the server will give a client a JWT that can be used to authorize later requests.

### OAuth

Open Authorization allows individual users of one authentication system to allow a third party to access data or take actions on behalf of the user.

For example, many sites allow you to sign in or tie your account to social media sites like Facebook by integrating with the social media's OAuth API.

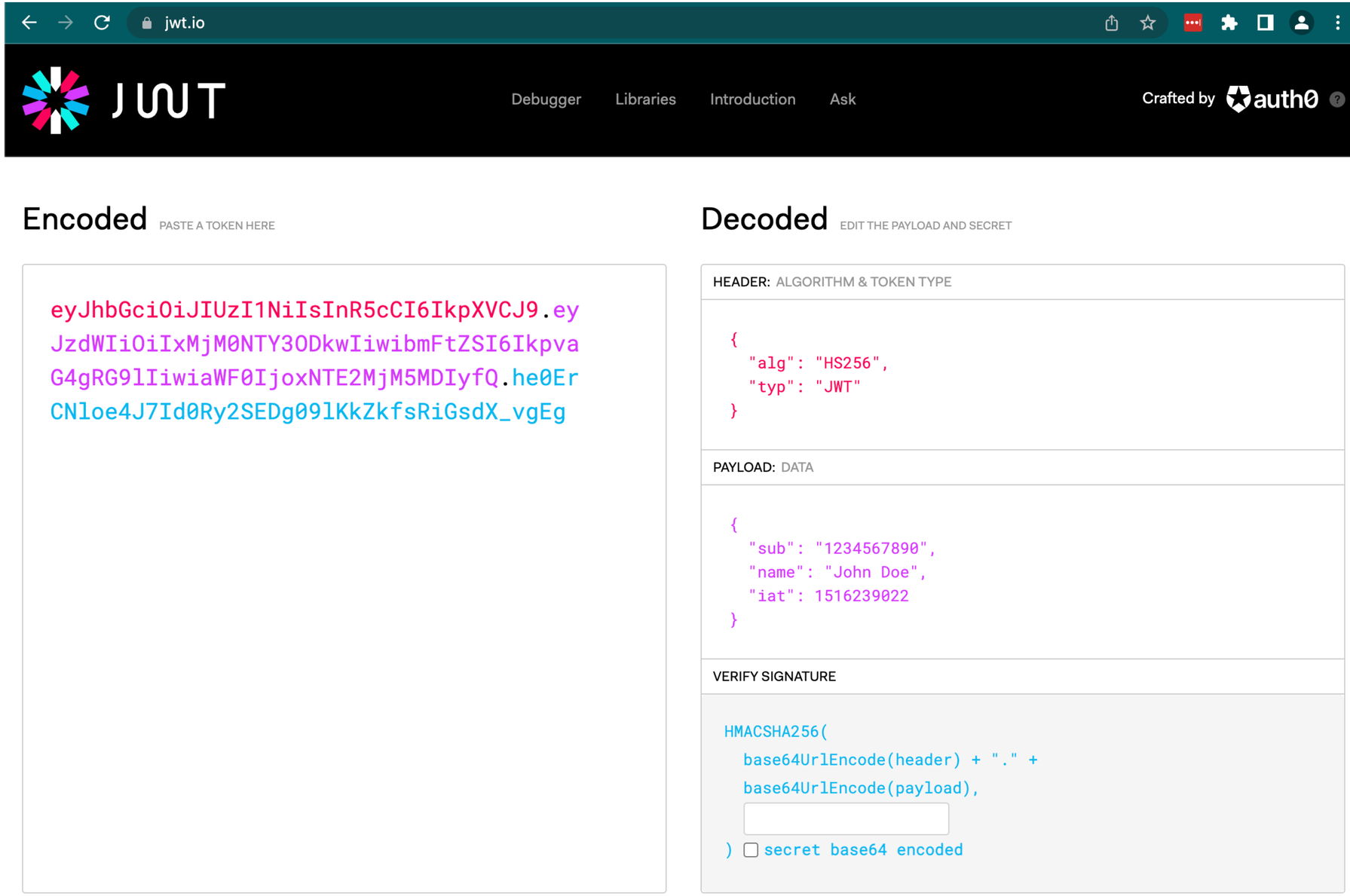
# Securing API Calls

Handling authenticated requests with JWTs

1) Authenticate user

2) Generate a JWT

3) Verify JWT on subsequent requests



# Securing API Calls
## Encryption

### Encryption in Transit

Securing data that is being transmitted over the internet.

The most common tool for securing web traffic is **HTTPS**, but its underlying mechanism, **Transport Layer Security** (TLS), can be used in many networked scenarios.

### **Encryption at Rest**

Securing data that is stored on physical drives (such as data in a database or on a file server).

Many regulated industries must **encrypt** different kinds of financial, personal, or confidential data to comply with laws and regulations, such as **GDPR, HIPAA, CCPA.**

# Vulnerabilities in Dependencies

Common Vulnerabilities and Exposures

Vulnerabilities in public libraries and tools are tracked as CVEs.

Mitre publishes CVE IDs and descriptions on https://cve.mitre.org/

**Types of threats in opensource software**

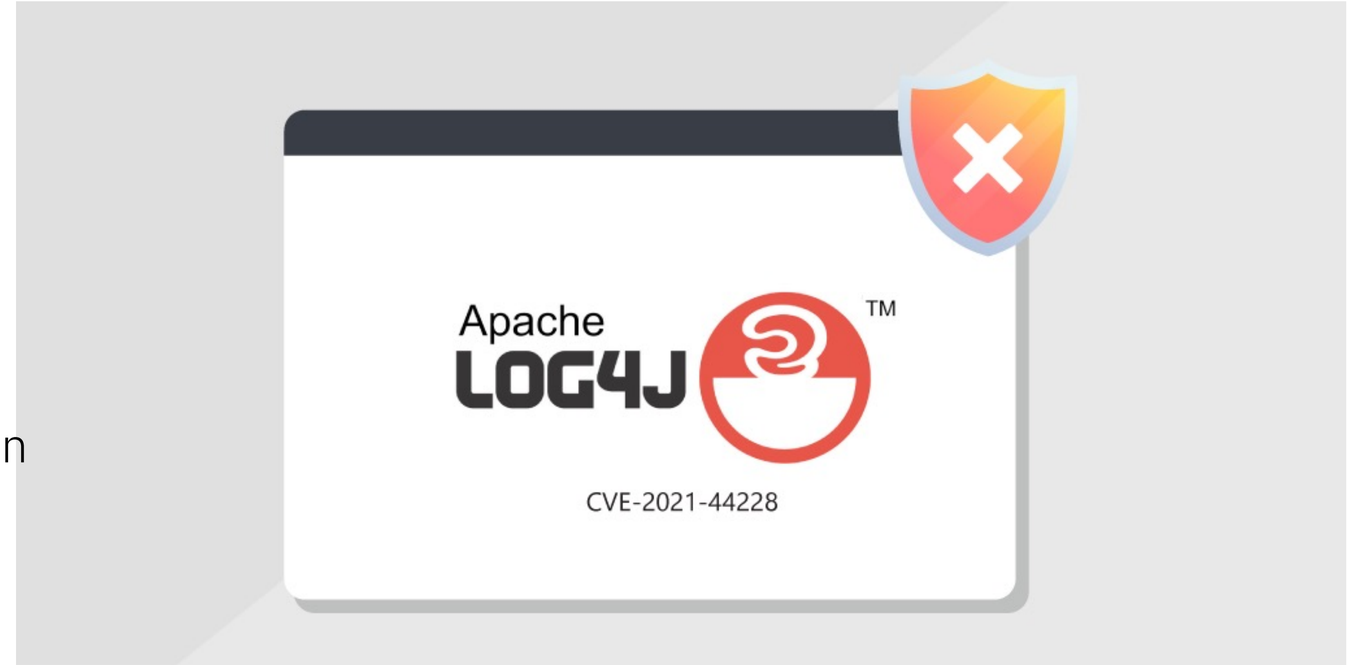
- Dependency vulnerabilities (log4j)
- Fragile dependencies (left-pad)
- Malicious repository takeover (ua-parser-js)
- Supply chain attacks (SolarWinds)

# Vulnerabilities in Dependencies

## Dependency vulnerabilities

Even popular, high quality opensource libraries often have bugs or vulnerabilities. When these are publicly detailed, they are often published as a Common Vulnerability and Exposure (CVE).

The popular Java logging library log4j recently had vulnerabilities discovered that could be abused to gain access to remote systems by making systems log specific text that could open remote shells.



# Vulnerabilities in Dependencies

## Fragile dependencies

Opensource libraries are completely controlled by their maintainers, who have their own agendas regarding features and maintenance level.

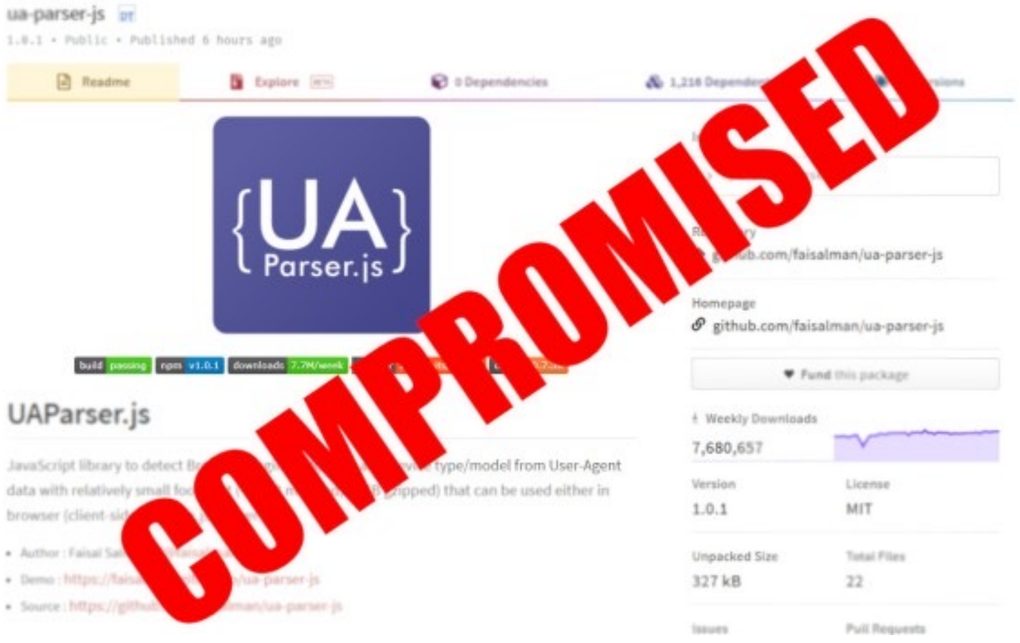
In 2016, the author of left-pad, a small but widely used npm dependency, took the package off npm after a dispute with npm about another of his repositories. Users of the library were then left scrambling to replace it. Many had no idea they even depended on the library, as it was a dependency of some other package they use.


# Vulnerabilities in Dependencies

## Malicious repo takeover

If the accounts of repository maintainers are compromised, the attacker can push new malicious updates onto unsuspecting users.

In 2021, an npm package ua-parser-js was compromised, and credential stealing malware was added to it that attempted to read end-users' data from their browser and send it off to the attacker's server.

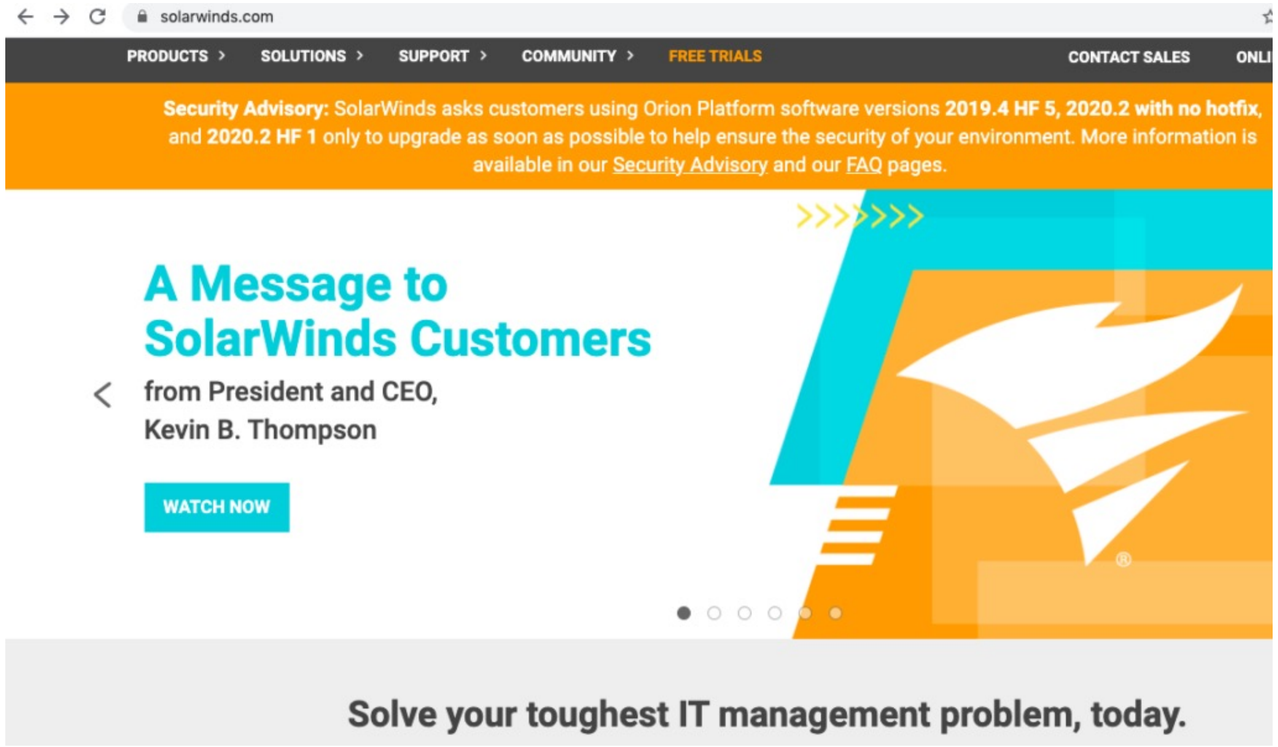


# Vulnerabilities in Dependencies

## Supply chain attacks

There is typically a “supply chain” between the source code and the built/compiled version of the code that developers or end users download. An attacker can target this payload instead of the source code to ship malware.

In 2020, hackers broke into SolarWinds servers and added malicious code to closed-source software updates that were being sent out to customers. The malware created a backdoor for the hackers to break into SolarWinds customers’ environments.



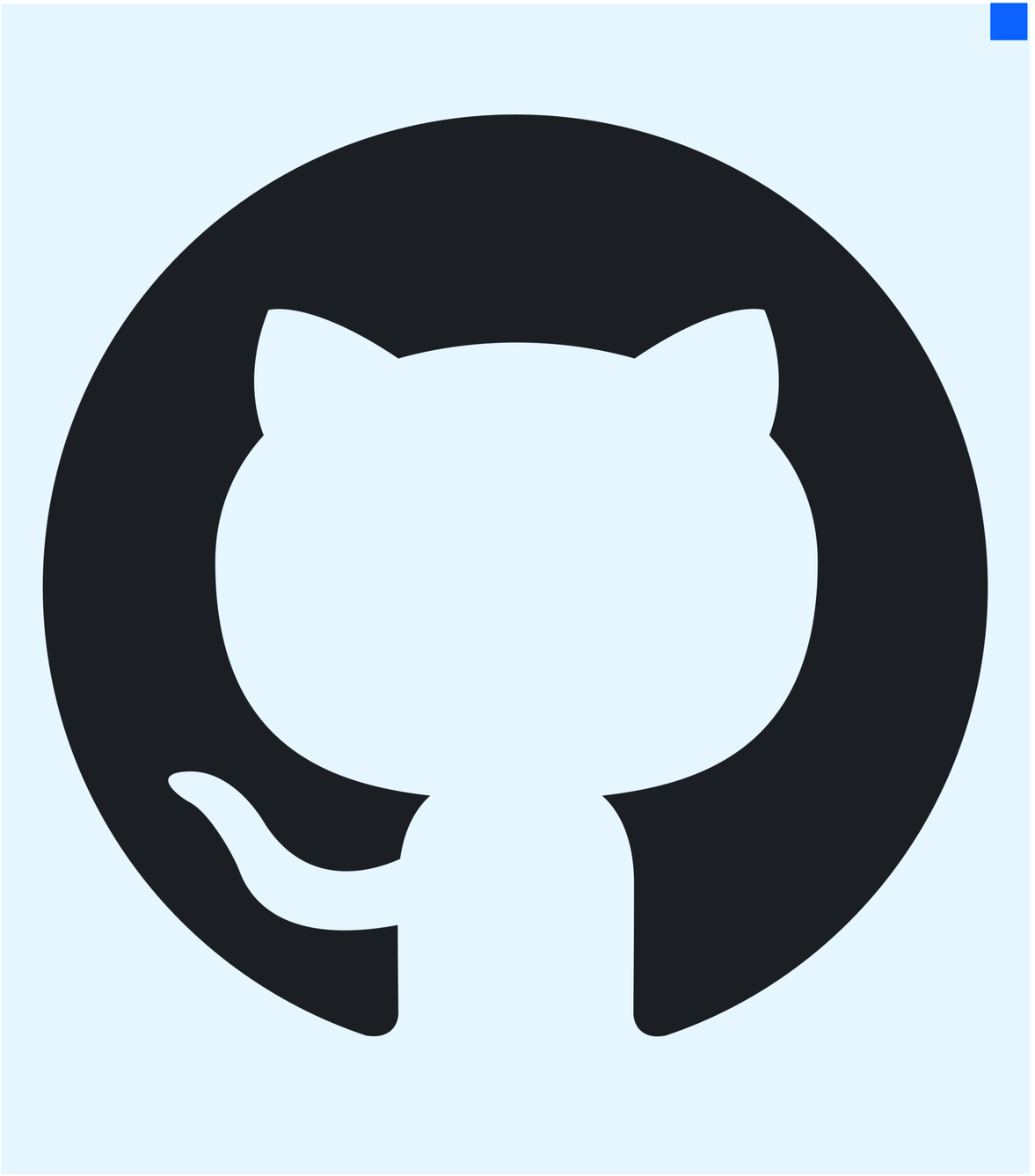
# Addressing Dependency Vulnerabilities

- Minimize dependency use, choosing high quality opensource libraries when necessary
  - Look for high usage, corporate sponsorship, and active maintenance
- Automated dependency scanning
  - Scan code repositories for all dependencies and versions
  - Look up public CVEs (known vulnerabilities) in those dependencies
  - `npm audit fix` for known npm vulnerabilities
- Verify dependency artifacts against opensource checksums

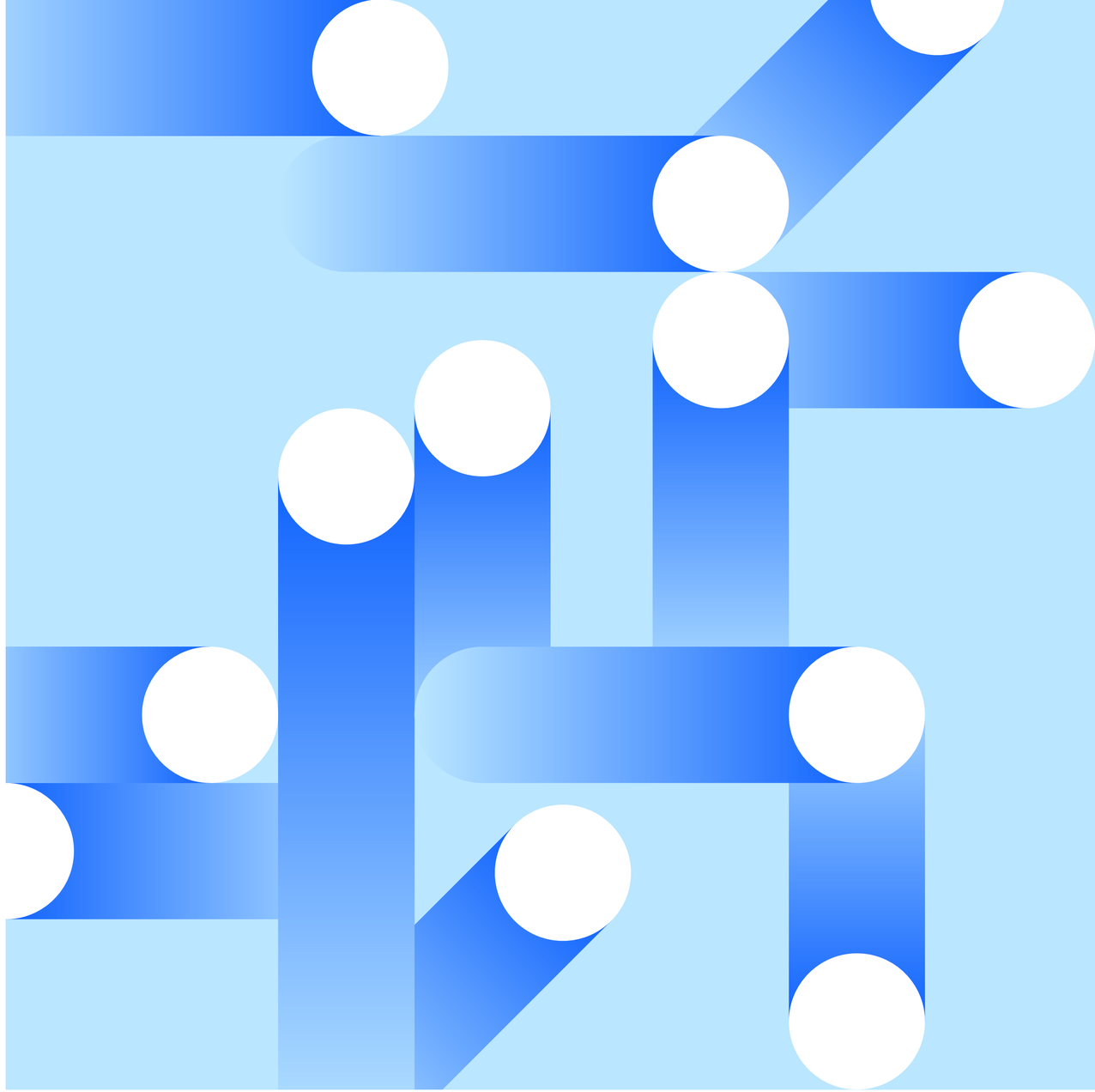
# Resources

Example CVE: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228
JWT Viewer: https://jwt.io
Identity and Access Management: https://www.ibm.com/topics/identity-access-management
OWASP Top 10: https://owasp.org/www-project-top-ten/
IBM Threat Modeling: https://www.ibm.com/garage/method/practices/code/threat-modeling/
IBM Secure Coding Practices: https://www.ibm.com/garage/method/practices/code/secure-coding-validate-inputs-and-output/

# Project/Lab and GitHub Classroom



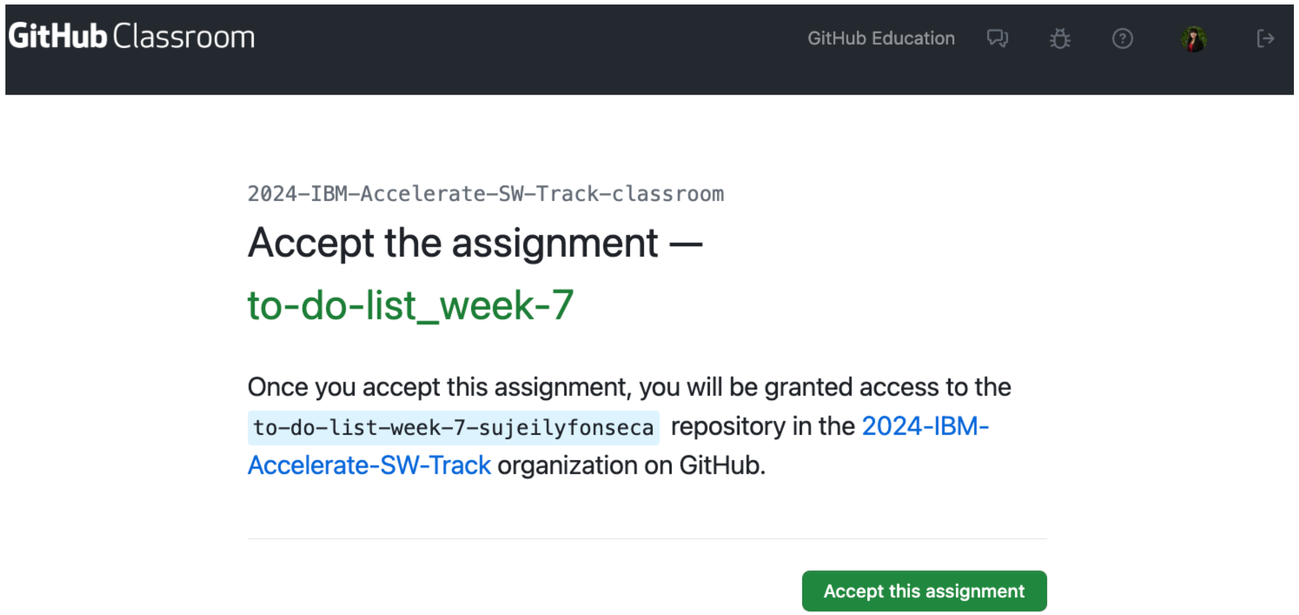
# Lab/Project 7:
# To-do-list – Add Login Authentication



# To-do-list Lab 7:
# Review

- Submissions for Lab 7 will close on Monday July 22, 2024, 11:59 PM EDT

- Use the Slack channel and office hours for any questions.




# To-do-list Lab – Add Login Authentication:
# Instructions

Feature Requirements:

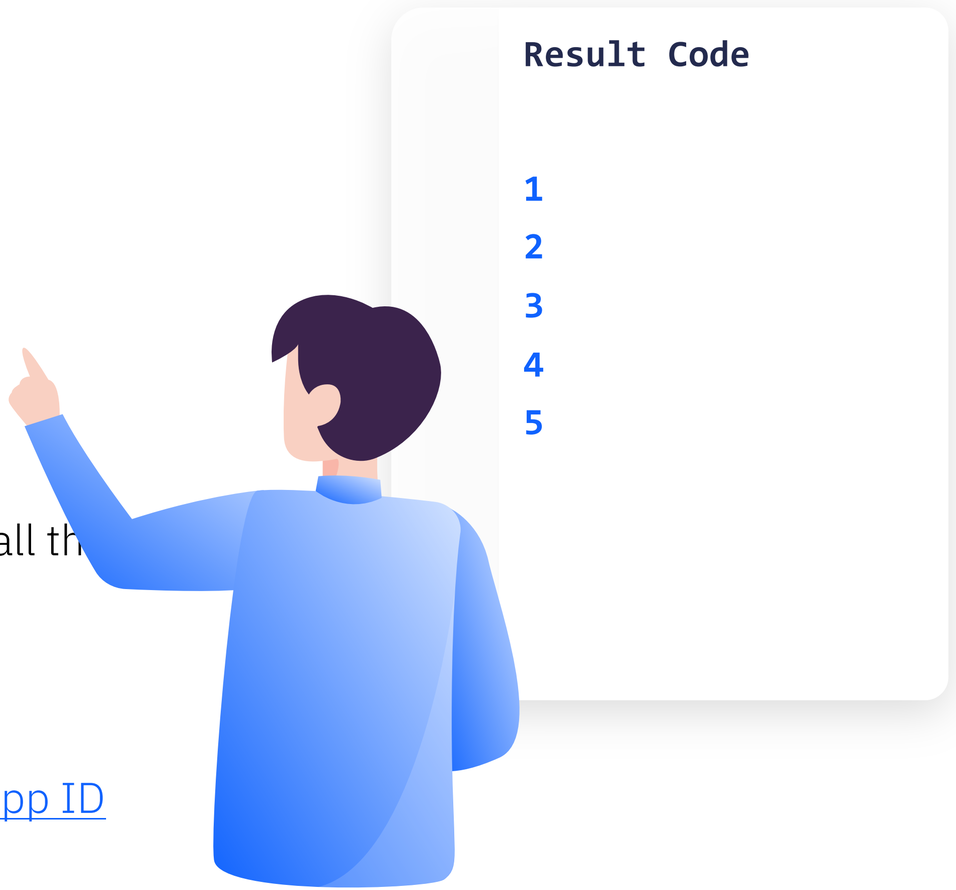
- Set up and code authentication to the Express application (Backend)
- Add a login page to the React to-do list app (Front-end)
- Create a Connection between the Front-end login page and Backend authentication
- Successfully log into your to-do list application
- Protect your other backend endpoints

Implementation Requirements:

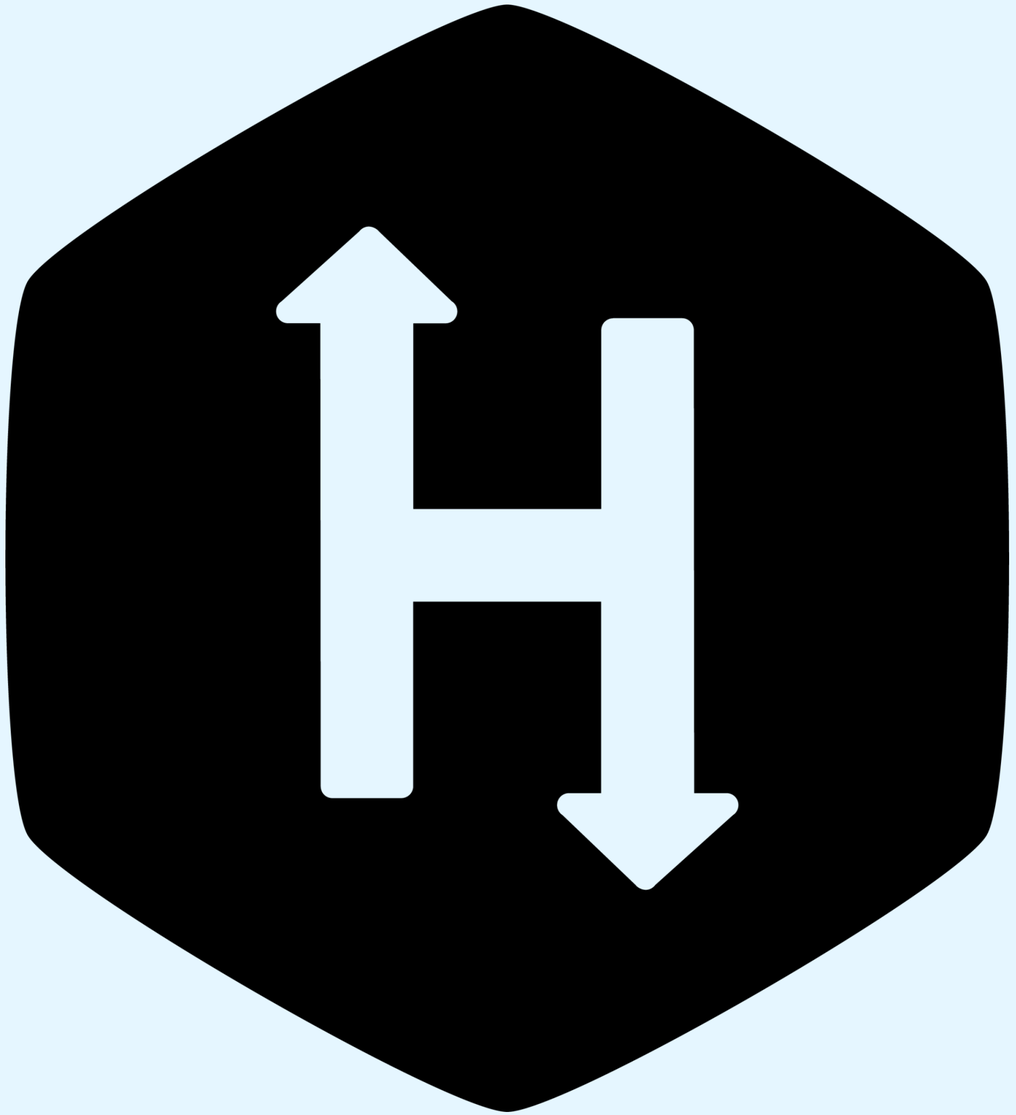
- Use express-basic-auth within the Backend component
- Use Axios within the Front-end component (To-do List Application to call th authentication)
- Modify your To-do list app to handle the session

Stretch Assignment (Optional Challenge):

- Try to set up a simple authentication for the TODO app using IBM Cloud App ID service by using the tutorial.

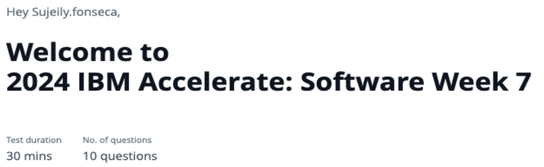


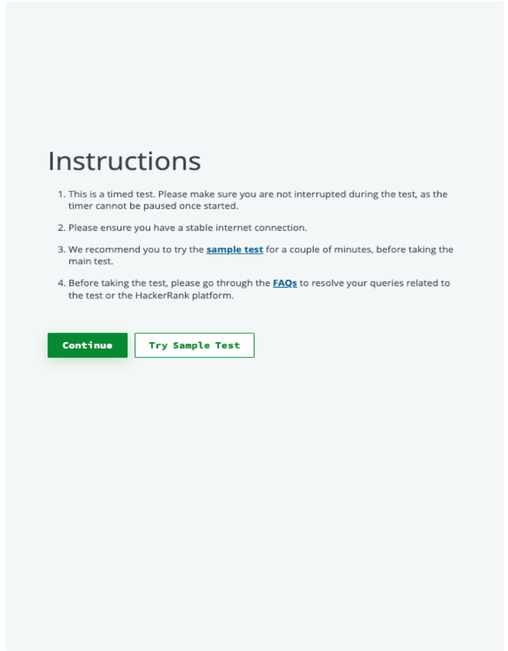
# HackerRank Test



# Required HackerRank Test for Week 6

- Tomorrow (Thursday), you should receive an email containing a link to a HackerRank quiz covering content from Week 7.
  - From IBM Corporation Hiring Team – support@hackerrankforwork.com
- Test will include **10 questions**
- Required to earn the IBM Accelerate Badge
- Passing score: 6/10
- Due by next Tuesday, July 23, 11:59 PM ET

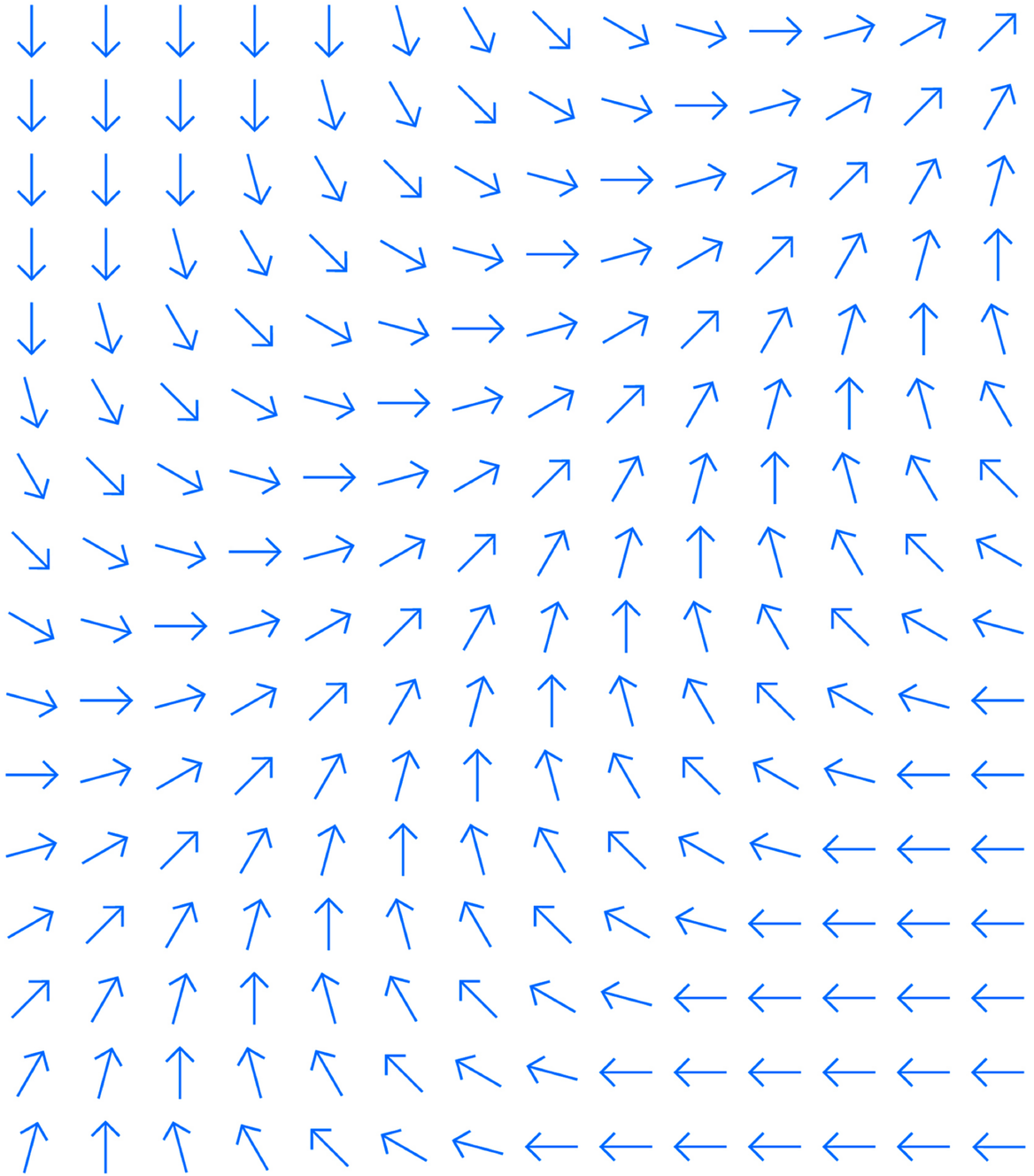


# Required HackerRank Test: Preparation

- Review labs and presentations from prior weeks
- Review optional HackerRank tests:
  - Same form and style as required test
- Use Slack and/or Office Hours to answer any questions

# Next Steps



# Next Steps

1. Finish your lab (https://ibm.biz/accelerate-sw-week7) by Monday, July 22, 11:59 PM ET.
2. If you need further help, ask in the slack channel (#ibm-accelerate-2024-software) or join the office hours (Thursday at 6:00 PM ET).
3. **Complete the required HackerRank test by Tuesday, July 23, 2024, 11:59 PM ET.**
4. Check out the solutions for the lab and HackerRank test once available.
5. Slides and video recordings of the session are available in our Box folder.
6. Watch for Week 8 materials to start to arrive by Monday.
7. Network and connect with your peers today and after today's session.

# Q&A

**Shravan Kumar Raghu**
Software Architect,
IBM Cloud Object Storage,
IBM Infrastructure

**Ope Jegede**
ServiceNow Solutions Architect,
IBM Cloud,
IBM Infrastructure

**Shidong Shan**
Senior Software Engineer,
Security Guardium,
IBM Software

Slack: ibm-accelerate-2024-software