Node.js web application security

CYDNdWeb3d | 3 days | On-site or online | Hands-on

Your Web application written in JS/TS works as intended, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -1 or -2^31? Because that's what the bad guys will do – and the list is far from complete.

Handling security needs a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hands-on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

The curriculum goes through the common Web application security issues following the OWASP Top Ten but goes far beyond it both in coverage and the details.

All this is put in the context of Node, and extended by core programming issues, discussing security pitfalls of the JS and TS language.

So that you are prepared for the forces of the dark side.

So that nothing unexpected happens.

Nothing.

Cyber security skills and drills

26 LABS   **14 CASE STUDIES**

Audience

Node developers working on Web applications

Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Wrap up

Group size

12 participants

Preparedness

General Node.js and Web development

Standards and references

OWASP, SEI CERT, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in JS/TS
- Understanding Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of JS/TS
- Going beyond the low hanging fruits
- Managing vulnerabilities in third party components

Table of contents

Day 1

› ## Cyber security basics

What is security?

Threat and risk

Cyber security threat types – the CIA triad

Consequences of insecure software

› ## The OWASP Top Ten 2021

A01 - Broken Access Control

- Access control basics
- Confused deputy
  - Insecure direct object reference (IDOR)
  - Path traversal
  - 🔬 *Lab – Insecure Direct Object Reference*
  - Path traversal best practices
  - Authorization bypass through user-controlled keys
  - 🔖 *Case study – Authorization bypass on Facebook*
  - 🔬 *Lab – Horizontal authorization*
- File upload
  - Unrestricted file upload
  - Good practices
  - 🔬 *Lab – Unrestricted file upload*
- Open redirects and forwards
  - 🔖 *Case study – Hacking Fortnite accounts*
  - 🔖 *Case study – Unvalidated redirect at Epic Games*
  - Open redirects and forwards – best practices

A02 - Cryptographic Failures

- Information exposure
  - Exposure through extracted data and aggregation
  - 🔖 *Case study – Strava data exposure*
- Cryptography for developers
  - Cryptography basics

- Cryptography in Node.js
- Elementary algorithms
  - Random number generation
  - Pseudo random number generators (PRNGs)
  - Cryptographically secure PRNGs
  - Weak and strong PRNGs in Node.js
  - *Lab – Using random numbers in Node.js*
    - *Case study – Equifax credit account freeze*
  - Hashing
  - Hashing basics
  - Hashing in Node.js
  - *Lab – Hashing in Node.js*
- Confidentiality protection
  - Symmetric encryption
  - Block ciphers
  - Modes of operation
  - Modes of operation and IV – best practices
  - Symmetric encryption in Node.js via Crypto
  - Symmetric encryption and decryption of streams in Node.js
  - *Lab – Symmetric encryption in Node.js*
  - Asymmetric encryption
  - The RSA algorithm
  - Using RSA – best practices
  - RSA in Node.js
  - Combining symmetric and asymmetric algorithms

Day 2

› The OWASP Top Ten 2021

A03 – Injection
- Injection principles
- Injection attacks
- SQL injection
  - SQL injection basics
  - *Lab – SQL injection*
  - Attack techniques
  - Content-based blind SQL injection
  - Time-based blind SQL injection
  - SQL injection best practices
    - Input validation
    - Parameterized queries

- *Lab – Using prepared statements*
  - *Case study – Hacking Fortnite accounts*
- Code injection
  - OS command injection
    - *Lab – Command injection*
    - OS command injection best practices
    - Command injection best practices
    - *Lab – Command injection best practices*
      - *Case study – Shellshock*
    - *Lab - Shellshock*
- HTML injection - Cross-site scripting (XSS)
  - Cross-site scripting basics
  - Cross-site scripting types
    - Persistent cross-site scripting
    - Reflected cross-site scripting
    - Client-side (DOM-based) cross-site scripting
  - *Lab – Stored XSS*
  - *Lab – Reflected XSS*
  - *Case study – XSS in Fortnite accounts*
  - XSS protection best practices
    - Protection principles - escaping
    - Anti-XSS protection in Node.js
    - *Lab – XSS fix / stored*
    - *Lab – XSS fix / reflected*
    - Additional protection layers – defense in depth
    - XSS protection in Angular
    - XSS protection in React
    - XSS protection in Vue.js

A04 – Insecure Design

- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder
  - Economy of mechanism
  - Fail-safe defaults
  - Complete mediation
  - Open design
  - Separation of privilege
  - Least privilege
  - Least common mechanism
  - Psychological acceptability
- Client-side security
  - Frame sandboxing
    - Cross-Frame Scripting (XFS) attacks

🔬 *Lab - Clickjacking*
- Clickjacking protection best practices

🔬 *Lab – Using CSP to prevent clickjacking*

A05 – Security Misconfiguration

- Configuration principles
- Node.js configuration best practices
  - General Node.js security considerations
  - Express.js security
  - Limiting resource use in Node.js apps
  - Security headers
  - Package manager considerations
- XML entities
  - DTD and the entities
  - Entity expansion
  - External Entity Attack (XXE)
    - File inclusion with external entities
    - Server-Side Request Forgery with external entities
    - 🔬 *Lab – External entity attack*
      - 📑 *Case study – XXE vulnerability in SAP Store*
    - Preventing XXE
  - 🔬 *Lab – Prohibiting DTD*

Day 3

› **The OWASP Top Ten 2021**

A06 – Vulnerable and Outdated Components

- Using vulnerable components
- Assessing the environment
- Hardening
- Untrusted functionality import
- Malicious packages in NPM
- Vulnerability management
  - Patch management
  - Vulnerability management
  - Vulnerability databases
  - 🔬 *Lab – Finding vulnerabilities in third-party components*
  - DevOps, the build process and CI / CD
  - Dependency checking in NodeJS

🔬 *Lab – Detecting vulnerable components*

A07 – Identification and Authentication Failures

- Authentication
  - Authentication basics
  - Multi-factor authentication (MFA)
  - Authentication weaknesses
  - 🗐 *Case study – PayPal 2FA bypass*
- Password management
  - Inbound password management
    - Storing account passwords
    - Password in transit
    - 🔬 *Lab – Is just hashing passwords enough?*
    - [Dictionary attacks and brute forcing](#)
    - Salting
    - Adaptive hash functions for password storage
    - 🔬 *Lab – Using adaptive hash functions in Node.js*
    - Password policy
    - [NIST authenticator requirements for memorized secrets](#)
      - 🗐 *Case study – The Ashley Madison data breach*
      - 🗐 *The ultimate crack*
      - 🗐 *Exploitation and the lessons learned*

A08 – Software and Data Integrity Failures

- Integrity protection
  - Message Authentication Code (MAC)
    - MAC in Node.js
    - 🔬 *Lab – Calculating MAC*
  - Digital signature
    - Digital signature with RSA
    - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
    - Digital signature in Node.js
    - 🔬 *Lab – Digital signature with ECDSA in Node.js*
  - Authenticated encryption
    - Authenticated encryption modes of operation
    - Authenticated encryption modes of operation: CCM
    - Authenticated encryption modes of operation: GCM
- Subresource integrity
  - Importing JavaScript
  - 🔬 *Lab – Importing JavaScript*
  - Subresource integrity in Angular
  - Subresource integrity in Vue.js

📑 *Case study – The British Airways data breach*
- Insecure deserialization
  - Serialization and deserialization challenges
  - Integrity – deserializing untrusted streams
  - Integrity – deserialization best practices

A09 - Security Logging and Monitoring Failures
- Logging and monitoring principles
- Log forging
- Log forging – best practices
- Logging best practices

A10 - Server-side Request Forgery (SSRF)
- Server-side Request Forgery (SSRF)

📑 *Case study – SSRF and the Capital One breach*

› Wrap up

Secure coding principles
- Principles of robust programming by Matt Bishop

And now what?
- Software security sources and further reading
- NodeJS and JavaScript resources