WA3358 Security Fundamentals for Software Engineers

Web Age Solutions Inc.
USA: 1-877-517-6540
Canada: 1-877-812-8887
Web: http://www.webagesolutions.com

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, INc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries or both.

UNIX is a registered trademark of the Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

IBM WebSphere, DB2 and Tivoli are trademarks of the International Business Machines Corporation in the Untied States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

For customizations of this book or other sales inquiries, please contact us at:

USA: 1-877-517-6540, email: getinfousa@webagesolutions.com
Canada: 1-877-812-8887 toll free, email: getinfo@webagesolutions.com

Web Age Solutions Inc.
821A Bloor Street West
Toronto
Ontario, M6G 1M1

Table of Contents

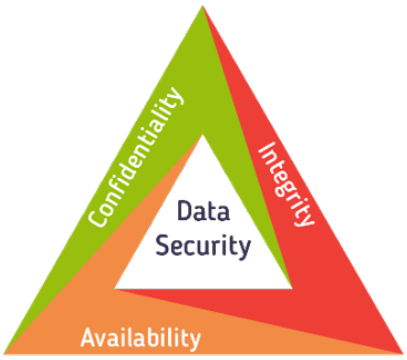
Chapter 1. Security Concepts

1.1. Introduction

- Developing secure software is as much a philosophy as a skill
- As developers we're often asked to "Make it work!"
- Later we work on "Make it safe!"
- This course aims to get you thinking earlier about security in the design and development process

notes: DevOps and DevSecOps are big buzzwords. In general the idea is find bugs or vulnerabilities earlier in the design and development process and provide open communication about the issues, risks, and corrections of those issues to everyone in the DevOps cycle.

1.2. CIA Triad



- Confidentiality
  - Determines the secrecy of the information asset.
- Integrity
  - Provides the assurance that the data is accurate and reliable.
- Availability
  - The ability of the users to access an information asset.

1.3. Confidentiality

- Protection of disclosure to unauthorized parties
- Any information leak can be catastrophic

- Includes protection from disclosure of:
  - stored data
  - restricted data
  - communication traffic
  - communication content

1.4. Integrity

- Protection of information from alteration, creation, or destruction by unauthorized parties
- Any alteration or threat of alteration may be catastrophic
- Includes protection from:
  - data file modification
  - program file modification
  - replication, altering, deleting, creating new or false data

1.5. Availability

- Assurance that systems and system resources will be accessible when needed
- Attacks often called DoS (Denial of Service)
- A temporary restriction of access
  - Permanent restriction is failure of integrity
- Includes protection from:
  - Excessive requests impeding ability to serve clients

- System rendered inoperable by removal of files/components/applications
- Overloading parts of the system leading to crashes

1.6. Authentication, Authorization, and Accounting (AAA)

- A three-process framework
  - Used to manage user access
  - Enforce user policies and privileges
  - Measure the consumption of network resources

1.7. Identification

- The unproven assertion of who you are
- For example:
  - A username
  - An ID card
  - A bank card

1.8. Authentication

- A user providing information about who they are and proving it
- Users or systems present login credentials that affirm they are who they claim to be

1.9. Authorization

- Authorization follows authentication
- A user can be granted privileges to access certain areas of a network, application, or system
- Areas and sets of permissions granted a user are often stored in a database along with the user's identity
- The user's privileges can be changed by an administrator or developer with access to the database

Notes

Authorization is different from authentication in that authentication only checks a user's identity, whereas authorization dictates what the user is allowed to do.

1.10. Accounting

- Tracking user activity while users interact with the system
- Often logging what data accessed or services used
- May be used to:
  - Analyze user trends

- Audit user activity
- Provide early warning of misuse

1.11. Principle of Least Privilege

- AKA principle of minimal privilege (PoMP) or the principle of least authority (PoLA)
- Every entity (such as a process, a user, or a program) must be able to access **only** the information and resources that are necessary for its legitimate purpose

Notes

In others words, watch for permissions creep!

1.12. Separation of Duties

- Ensures one individual cannot complete a critical task by themselves
- Reduces the possibility for fraud, sabotage, theft or general abuse
- Requires collusion of two or more parties to circumvent

1.13. Non-repudiation

- Provides proof of the origin, authenticity, and integrity of data.
- Ensures that no party can deny sending or receiving a message
- Ensures no party can contest the authenticity of their signature on a document

Notes

Achieved through encryption, digital signatures, and other cryptographic techniques.

It is a crucial aspect of cybersecurity and information assurance, which involves managing information-related risks and protecting information systems. Non-repudiation helps establish trust and accountability in digital transactions and communications. MERN stack developers can implement non-repudiation measures, such as using secure communication protocols, implementing strong authentication mechanisms, and logging and auditing system activities.

1.14. Defense in Depth

- A cybersecurity approach that involves layering multiple defensive mechanisms to protect valuable data and information
- If one mechanism fails, there are other layers of defense in place to prevent or mitigate an attack
- Addresses different attack vectors and provides a more robust and resilient security posture

Notes

Defense in depth works like a medieval castle with multiple layers of defense, where each layer provides an additional barrier to protect the castle from invaders. In the digital world,

defense in depth works by implementing various security measures such as firewalls, intrusion detection systems, encryption, access controls, regular software updates, and employee training on cybersecurity best practices.

1.15. Defense in Depth as a Developer

- Follow secure coding practices
- Stay updated on the latest security vulnerabilities
- Learn best practices in web application security

Notes

By implementing this approach, you can enhance the security of your applications and contribute to a more secure digital environment.

1.16. Software Security Terminology

- Asset
  - The object which requires protection
- Attack path
  - The mechanism taken by an attacker to access an asset
- Attack surface
  - The sum of all the possible attack paths

1.17. Software Security Terminology

- Defects/weaknesses
  - Errors that could result in unexpected or undesired behavior
- Vulnerability
  - A defect that can be exploited to compromise a system
- Threats
  - A harm that might be caused by an attacker
- Exploit
  - When an attacker uses a vulnerability to cause harm

1.18. Common Threats

- Hackers
- Users
- Insiders

Notes

Threats can come from anywhere, intentionally or unintentionally. As software developers,

finding bugs and vulnerabilities earlier is much less expensive than waiting until the product is deployed. It’s practically free to fix a vulnerability in development.

1.19. Common Attacks

- Buffer overflow
- Command injection
- SQL injection
- Cross site scripting

Notes

These are just some examples of attacks. We’ll explore more in other sections of the course.

Chapter 2. Zero Trust

Objectives
Key objectives of this chapter ▪ Learn about the Zero Trust model ▪ Discuss how Zero Trust applies to application development

2.1. Zero Trust

- An evolving set of cybersecurity paradigms
- A security framework that requires all users to be authenticated, authorized, and continuously validated before being granted access to resources
- Assumes no implicit trust granted to assets or user accounts based solely on their physical or network location
- The main mantra: Never Trust! Always Verify!

2.2. Zero Trust Core Principles

- Never trust, always verify
- Principle of least privilege
- Move security away from the perimeter to the application itself
- Continuous monitoring and analytics
- Embrace adaptability and flexibility

Notes

Developers should design applications to have built-in security measures and the ability to validate and enforce security policies

2.3. Never Trust, Always Verify

- Systems operating under a Zero Trust framework do not initially trust access or transactions from anyone, including internal users
- Always authenticate and authorize based on all available data points, including
  - User identity
  - Location
  - Service or workload
  - Data classification
  - Any anomalies

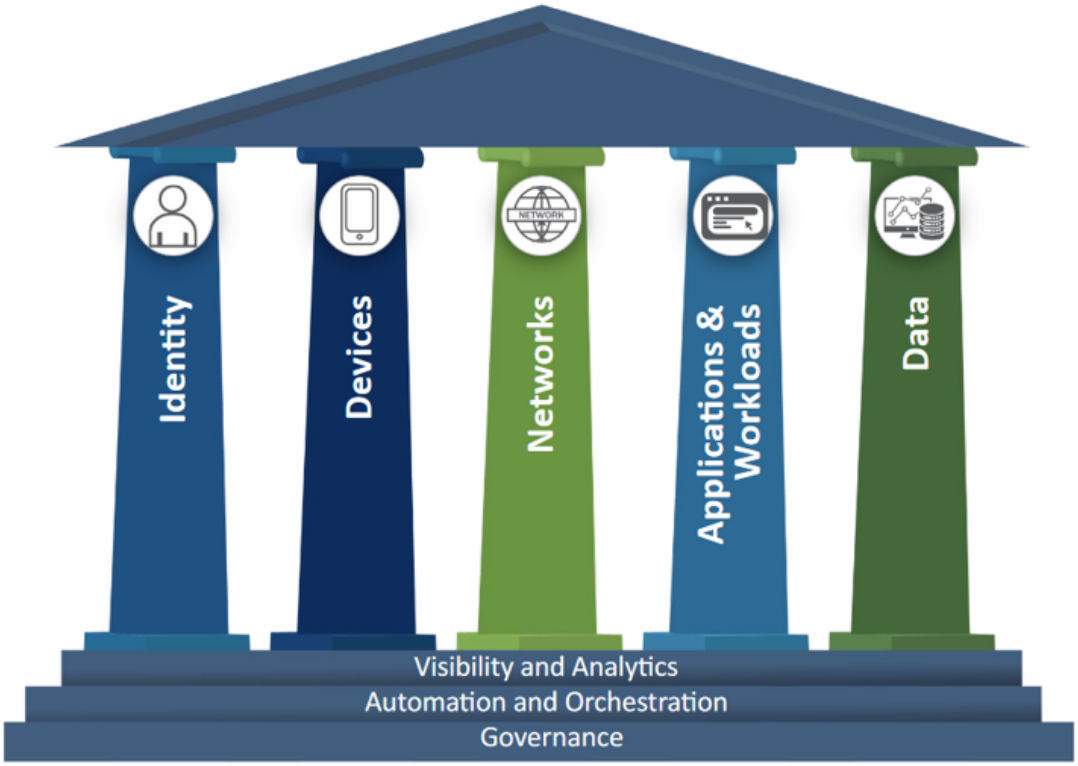
2.4. Assume Breach

- Minimize blast radius and segment access to modules or components of your application
- Verify end-to-end encryption
- Use analytics to get visibility, drive threat detection, and improve defenses

2.5. Apply Least-privilege

- Limit user access with
  - Just-in-time
  - Just-enough access
  - Risk-based adaptive polices
  - Data protection to help secure both data and productivity
- This applies to insiders as well!

2.6. The Zero Trust Maturity Model



Notes

From https://www.cisa.gov/zero-trust-maturity-model

The maturity model, which includes five pillars and three cross-cutting capabilities, is based on the foundations of zero trust. Within each pillar, the maturity model provides specific examples of traditional, initial, advanced, and optimal zero trust architectures.

2.7. Identities

Verify and secure each identity with strong authentication across your entire digital estate.

2.8. Devices

- Gain visibility into devices accessing the network.
- Ensure compliance and health status before granting access.

2.9. Network

- Ensure that devices and users aren’t trusted just because they’re on an internal network.
- Encrypt all internal communications, limit access by policy, and employ micro-segmentation and real-time threat detection.

2.10. Applications and Workloads

- Discover shadow IT, ensure appropriate in-app permissions, gate access based on real-time analytics, and monitor and control user actions.
- Use telemetry to detect attacks and anomalies, automatically block and flag risky behavior, and employ least-privilege access principles.

2.11. Data

- Move from perimeter-based data protection to data-driven protection.
- Use intelligence to classify and label data.
- Encrypt and restrict access based on organizational policies.

2.12. Zero Trust and Developers

- Incorporate Zero Trust principles throughout the development cycle
- Not a replacement for security fundamentals, developers should still follow best practices
- Manage and secure data where no actual perimeter exists such as cloud services
- Developers should verify the security of open source and third-party components used in their projects and apply updates and fixes as necessary

Notes

Developers should ensure that access controls are implemented to restrict users' access to only the resources they need to perform their tasks Developers should design applications to have built-in security measures and the ability to validate and enforce security policies Developers should incorporate logging, monitoring, and analytics capabilities into their applications to enable real-time threat detection and response

Chapter 3. OWASP Top Ten

Objectives
Key objectives of this chapter ▪ Learn about the Open Web Application Security Project ▪ Discuss tools and approaches to mitigate OWASP vulnerabilities

3.1. The OWASP Top Ten

The Open Web Application Security Project formed in 2003, a foundation that focuses on software security. It supports the security industry with the OWASP Top 10.

Notes

- A report that identifies current software security vulnerability concerns
- Represents a consensus from the OWASP core team, security analysts, security organizations, and other security experts
- Used globally as a de facto standard check for web application security

3.2. OWASP Top 10

Vulnerability	Description
1. Broken access control	failures can jeopardize information disclosures and data integrity
2. Cryptographic failures	concerns data exposure and compromises confidentiality
3. Injection	involves hostile data use, attacks, and unsafe queries
4. Insecure design	covers weaknesses and flaws in control designs
5. Security misconfiguration	involves features that are incorrectly enabled or have other configuration

3.3. OWASP Top 10

Vulnerability	Description
6. Vulnerable and outdated components	involve version control and other compatibility issues

Vulnerability	Description
7. Identification and authentication failures	covers password issues, automated attacks like credential stuffing, and session
8. Software and data integrity failures	involve integrity violations, which are often from untrusted sources
9. Security and logging and monitoring failures	involve detecting and responding to breaches
10. Server-side request forgery	results in URL validation failures

3.4. OWASP Testing Framework

- Activities that should take place:
  - Before development begins
  - During definition and design
  - During development
  - During deployment and
  - During maintenance and operations

3.5. 2021 OWASP Top 10 Details

3.6. Injection

- Injection attacks occur when untrusted data is sent to an interpreter
- Can lead to data loss, corruption, or unauthorized data disclosure
- Includes SQL injection where user input constructs a SQL query without proper validation or escaping

3.7. Broken Authentication

- Incorrect implementation of session management and authentication functions
- May compromise passwords, keys, or session tokens
- Weak password enforcement can lead to unauthorized access

3.8. Sensitive Data Exposure

- Occurs when sensitive data is inadequately protected
- Transmitting sensitive data over HTTP instead of HTTPS can lead to exposure

3.9. XML External Entity (XXE)

- XXE attacks happen with XML input containing external entity references
- Can result in disclosure of internal files, denial of service, or remote code execution

3.10. Broken Access Control

- Flaws in enforcing restrictions on authenticated users
- Allows attackers to access unauthorized functionality or data

3.11. Security Misconfigurations

- Using default or incomplete configurations can lead to unauthorized access
- Misconfigurations can expose private information, such as directory listing vulnerabilities

3.12. Cross-Site Scripting (XSS)

- Flaws in adding untrusted data to web pages
- Attackers can execute malicious scripts, hijack sessions, deface websites, or redirect users

3.13. Insecure Deserialization

- Insecure deserialization allows remote code execution
- Can lead to replay attacks, injection attacks, and privilege escalation

Using Components with Known Vulnerabilities

- Components have the same privileges as the application using them
- Exploiting vulnerable components can lead to data loss or server takeover

3.14. Insufficient Logging & Monitoring

- Inadequate logging and monitoring give attackers opportunities to persist and pivot

- Proper logging and monitoring are crucial for timely detection and response

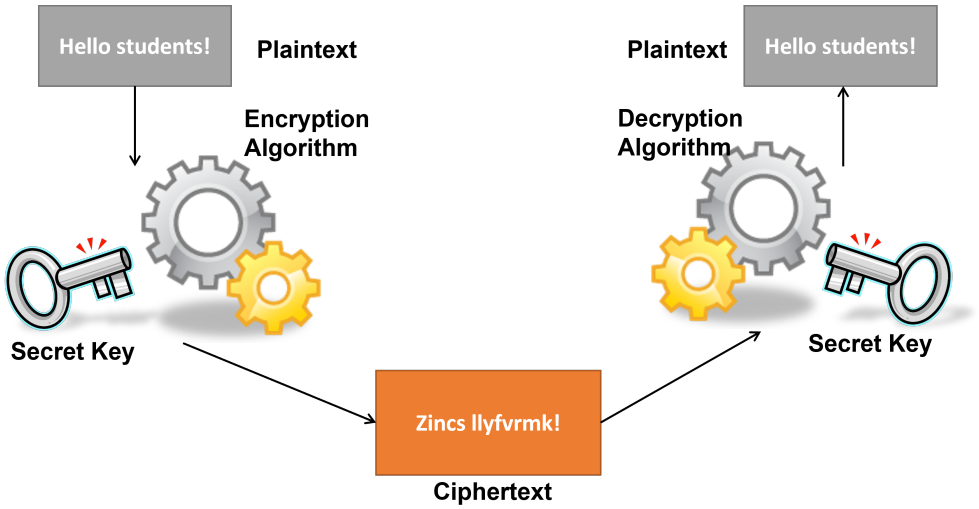
Chapter 4. Encryption and Decryption

Objectives

Key objectives of this chapter

- Learn some key principles of cryptography
- Understand hashing
- Understand digital certificates and their use

4.1. Encryption and Decryption



Notes

Algorithm is a fancy word for recipe. Encryption algorithms are recipes to scramble data and make it difficult, if not impossible, for someone to unscramble without the secret key. Observers may be able to see that the data exists but not the contents of the data itself. In other words, encryption is about hiding information in plain sight. When we deal with encryption, we often talk about the three states of data, at rest, in transit, or at work (in memory). Different algorithms are used for one of more of these purposes. For example, the Advanced Encryption Standard (AES) is often used for data at rest and in transit. RSA is often used for data in transit, in fact it is often used to encrypt the shared secret key for AES.

4.2. Three Key Ideas

- Confusion (substitution)
- Diffusion (transposition)
- Secrecy only in the key

4.3. Terminology

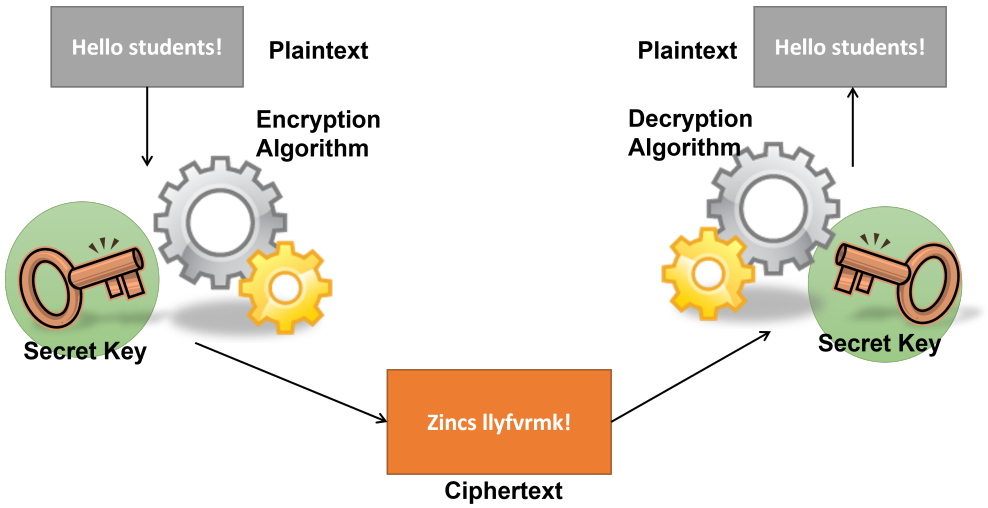
- Encryption - the method of transforming data (plaintext) into an unreadable format
- Plaintext - the format (usually readable) of data before being encrypted

- Ciphertext - the "scrambled" format of data after being encrypted

4.4. Cryptographic Terminology

- Decryption - the method of turning cipher text back into plaintext
- Encryption algorithm a set or rules or procedures that dictates how to encrypt and decrypt data, also called an encryption cipher
- Key - (cryptovariable) a value used in the encryption process to encrypt and decrypt

4.5. Symmetric Encryption



Notes

With symmetric encryption algorithms the same key that is used to encrypt the data is also used to decrypt the data. This raises the question, if the two parties that wish to communicate securely both need a copy of the secret key, then how do we transport the key securely between the two?

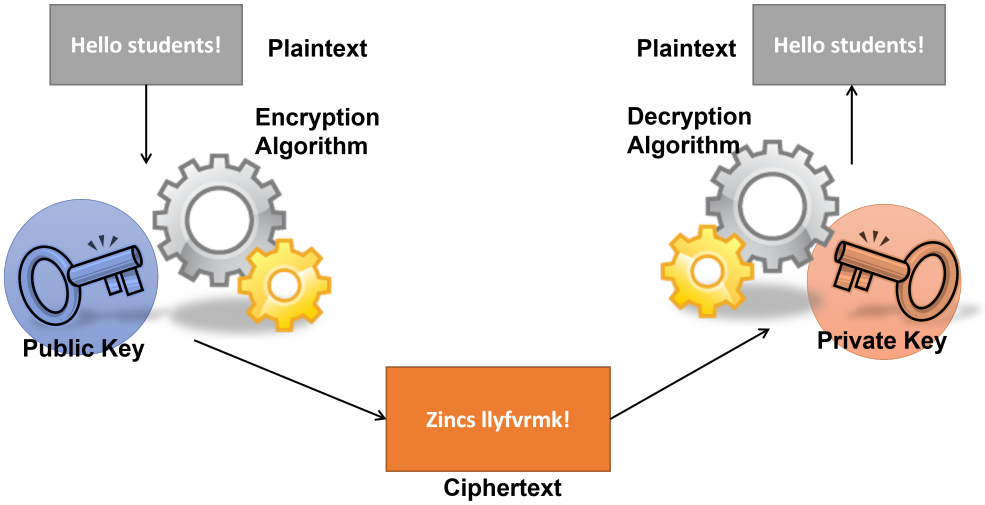
4.6. Some Common Symmetric Algorithms

- AES (Advanced Encryption Standard): AES is widely used and considered secure. It supports key sizes of 128, 192, and 256 bits.
- ChaCha20: ChaCha20 is a stream cipher that is gaining popularity due to its speed and security.

Notes

Today AES is the most commonly used symmetric algorithm. It has different configurations. You've probably heard a web site advertise they use "military grade" AES-256 bit encryption.

4.7. Asymmetric Encryption



Notes

With asymmetric algorithms it takes two keys to successfully share data. One key is used to encrypt the data, the other to decrypt. We call one key Public because we generally don't worry about if anyone has a copy of this key, in fact we often share this key publicly so that someone can send us encrypted data. The other key is Private and should **never be shared with anyone**.

Think of the keys as a matched set where one is the antidote to the other. It doesn't matter which key is used to encrypt or decrypt the data, the way to decrypt is always to use the other key. What matters is **when** a key is used. For example, a Private key is used to encrypt a small piece of data, usually a hash (summary) of a document. This is a signature. The signature can be decrypted by **anyone with the Public key**. This means it doesn't protect the data but since only the owner has the Private key, it proves that only the owner could have sent the original encrypted data (signature).

The Public key is used to encrypt message.

Private key is used to decrypt message.

Private key is used to sign messages.

Public key is used to validate signed messages.

4.8. Common Asymmetric Algorithms

- RSA (Rivest-Shamir-Adleman)
- Elliptic Curve Cryptography (ECC)
- Diffie-Hellman Key Exchange (DH
- DSA (Digital Signature Algorithm)
- ECDSA (Elliptic Curve Digital Signature Algorithm)
- ElGamal

Notes

RSA is one of the most widely used asymmetric encryption algorithms. It is based on the

mathematical properties of large prime numbers and is commonly used for secure communication, digital signatures, and key exchange.

ECC is a family of asymmetric encryption algorithms that are based on the mathematics of elliptic curves. ECC offers strong security with shorter key lengths compared to other algorithms like RSA, making it more efficient for resource-constrained devices.

DSA is an asymmetric encryption algorithm used for digital signatures. It is based on the mathematical properties of modular exponentiation and is commonly used for authentication and integrity verification of digital documents.

ECDSA is a variant of DSA that uses elliptic curve cryptography. It provides the same functionality as DSA but with shorter key lengths, making it more efficient for resource-constrained devices.

ElGamal is an asymmetric encryption algorithm that is based on the Diffie-Hellman key exchange. It is primarily used for encryption and key exchange.

4.9. Hashing

- Hashing is a one-way operation
- Converts a large amount of data into a fixed length value (hash digest)

Notes

You can see examples of different hashing algorithms by entering some sample text at this web site:

Hash: online hash value calculator: https://www.fileformat.info/tool/hash.htm

Change just one character in the sample text and see the giant change in the hash.

4.10. Secure Hashing Algorithm

- Family of cryptographic functions published by NIST
- SHA-3 (formerly Keccak) is the most recent
- Used to establish integrity
- Combined with encryption algorithms can be used as part of a digital signature

4.11. Public Key Infrastructure

- A set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates
- Enables C and I of the CIA triad
- Enables non-repudiation

4.12. Digital Certificates

- Each person or server can have a *digital certificate*

- Has information about a person, including a person's *public* key
- Typically, not attached to a person but to a server or system

4.13. Certificate Authorities

- Certificates are signed by a Certificate Authority
- The Certificate Authority vouches for each organization's certificate
- Some certificates can be self-signed
- Typically, organizations purchase **wildcard** certificates from authorities such as Let's Encrypt, Verisign, or Thawte

4.14. X.509 Certificates

- A standard format for public key certificates
- Applications:
  - SSL/TLS
  - Signed and encrypted email
  - Code signing
  - Document signing
  - Client authentication
  - Government-issued electronic IDs

4.15. Parts of a certificate

- A public key
  - Part of a key pair that also includes a private key
  - Private key is kept secure
  - Only the public key is included in the certificate
- A digital signature
  - Information about both the identity associated with the certificate and its issuing certificate authority (CA)
  - An encoded hash (fixed-length digest) of a document that has been encrypted with a private key
- Fields specifying the subject, issuing CA, validity and other and other required information

4.16. Secure Sockets Layer/Transport Security Layer

- Encrypt data in transit
- Its successor is TLS (Transport Layer Security)
- Both are protocols for establishing authenticated and encrypted links between networked computers

- Encrypts plaintext, like passwords, and credit card numbers
- Only the user and the website can decrypt
- Ensures the data remain unchanged
- Authenticates websites
- Modern browsers show warnings when the connection is NOT entirely secure

4.17. SSL (TLS) Certificate

- Identify an individual, server, or organization
- A digital document that binds the identity of a website to a cryptographic key pair
- The key pair consists of a public key and a private key
- The public key, included in the certificate, allows a web browser to initiate an encrypted communication session with a web server via the TLS and HTTPS protocols
- The private key is kept secure on the server, and is used to digitally sign web pages and other documents (such as images and JavaScript files)

4.18. HTTPS

- HTTPS (Hypertext Transfer Protocol Secure) is a secure version of the HTTP protocol
- Uses the SSL/TLS protocol for encryption and authentication
- HTTPS is specified by RFC 2818 (May 2000) and uses port 443 by default instead of HTTP's port 80

4.19. HTTPS Provides

- Encryption
- Authentication
- Integrity
- Validation
  - Domain validation (DV)
  - Organization/Individual validation (OV/IV)
  - Extended validation (EV)

4.20. How does HTTPS work?

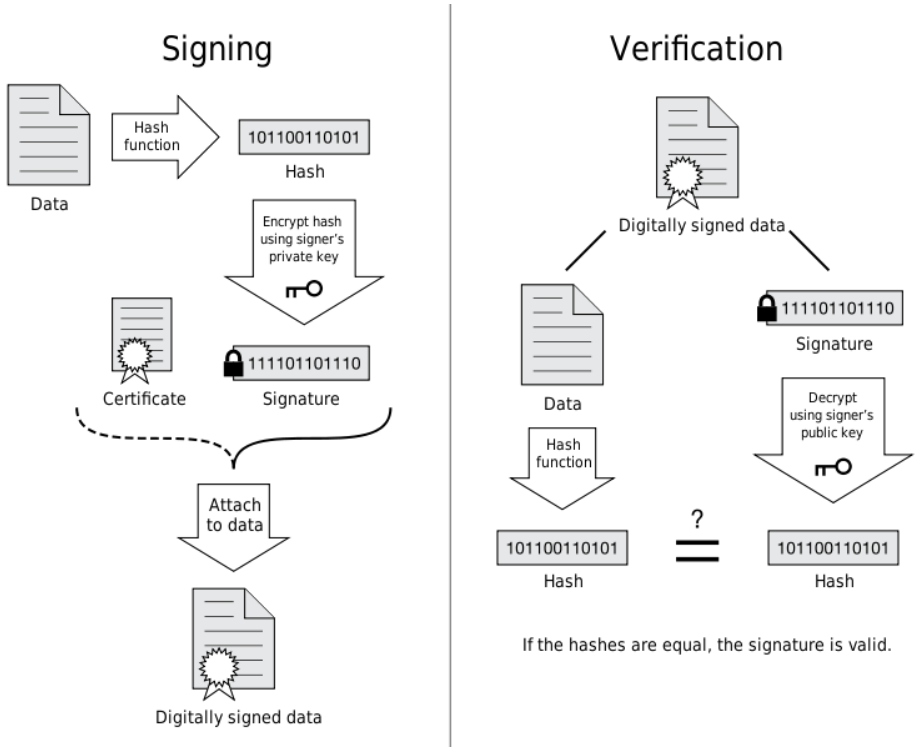
- Adds encryption to the HTTP protocol by wrapping HTTP inside the SSL/TLS protocol
  - Which is why SSL is called a tunneling protocol)
- All messages are encrypted in both directions between two networked computers including:
  - Request URL (which web page was requested by the client)
  - Website content

- Query parameters
- Headers
- Cookies

4.21. HTTPS and Certificates

- Uses digital documents known as X.509 certificates
- Binds cryptographic key pairs to the identities of entities such as websites, individuals, and companies
- Each key pair includes
  - a private key, which is kept secure
  - a public key, which can be widely distributed
- Anyone with the public key can use it to:
  - send data that only the private key possessor can decrypt
  - confirm that data has been digitally signed by the corresponding private key

4.22. Digital Signatures



4.23. Code-signing Certificates

Property	Code-signing	SSL/TLS
Purpose	Digitally sign software code, ensuring its integrity and authenticity	Used to establish secure encrypted connections between a client (such as a web browser) and a server

Property	Code-signing	SSL/TLS
Protection	Protect end-users by guaranteeing the integrity of the code they download	Protect end-users by securing communication between their browsers and websites
Outcome	Verifies that the code has not been tampered with and comes from a trusted source	Ensures that the communication between the client and server is encrypted and secure, protecting sensitive data during transmission

Notes

Code-signing certificates are used by software developers and publishers to sign their code, including applications, scripts, and executables. This allows users to verify the authenticity and integrity of the code before installation or execution.

SSL/TLS certificates are used by website owners and operators to secure their websites and enable HTTPS connections. They are essential for encrypting sensitive information, such as login credentials, credit card details, and personal data, transmitted between the user's browser and the website.

Technical Differences: Code-signing certificates typically use digital signatures to verify the integrity and authenticity of the code, while SSL/TLS certificates use asymmetric encryption to establish secure connections. Code-signing certificates and SSL/TLS certificates have different certificate chains and key usages.

Chapter 5. Authentication and Authorization

Objectives
Key objectives of this chapter ▪ Learn authentication factors ▪ Explain the difference between Authentication and Authorization ▪ Understand Single Sign-On ▪ Introduce OAuth

5.1. What is Authentication?

- The process of verifying the identity of a given user or client
- Making sure that they really are who they claim to be
- Most websites are exposed to anyone who is connected to the internet by design
- Robust authentication mechanisms are vital to effective web security

5.2. Authentication Factors

- something you know
- something you have
- something you are or do

5.3. Authentication vs. Authorization

- Authentication
  - ◇ Verifying that a user really is **who they claim to be**
- Authorization
  - ◇ Verifying whether a user is **allowed to do something**

5.4. Introduction to Single Sign-On (SSO)

- Allows users to log in to one application and be automatically signed in to other applications, regardless of platform or domain
- Provides a seamless user experience
- Eliminates the need to remember multiple credentials for different applications.
- Example: Google’s SSO - logging in to Gmail automatically authenticates the user to YouTube, AdSense, Google Analytics, and more

5.5. SSO Workflow

- Users log in to one domain, which serves as the authentication domain.

- **Authentication Domain**: The domain responsible for verifying user credentials.
- **Redirection**: When accessing a domain requiring authentication, users are redirected to the authentication domain.
- **Already Logged-In Users**: If already logged in at the authentication domain, users are immediately redirected back to the original domain without signing in again.

5.6. Single Logout (SLO)

- Logging out of one application automatically logs the user out of all associated applications.
- Example: Logging out of Gmail also logs the user out of other Google apps.
- Benefit of SLO: Ensures a secure logout process across all applications.

5.7. Advantages of SSO

- Advantages of SSO:
  - Enhanced User Experience: Users only need to log in once for access to all applications.
  - Simplified Credential Management: No need to remember separate login details for each app.
  - Increased Security: Centralized authentication and logout process.

5.8. Implementing SSO for MERN Stack

- Implementing SSO for MERN Stack:
  1. Choose a reliable SSO solution.
  2. Integrate the SSO solution into your MERN application.
  3. Configure authentication and redirection domains.
  4. Test the SSO functionality thoroughly.

5.9. Choose an SSO provider

- Select an SSO provider that fits your project requirements
- Popular options include Auth0 and Okta
- Follow the provider’s documentation to create an account, set up an application, and configure the necessary settings

Notes

Resource: Build a SSO integration Build a Single Sign-On (SSO) integration | Okta Developer [https://developer.okta.com/docs/guides/build-sso-integration/-/main/]

5.10. Authentication and Authorization in MERN

- Build user authentication functionality in your MERN stack application using a popular

library like Passport.js or JSON Web Tokens (JWT)

- Establish user roles and permissions to control access to different parts of your application

Notes

How to use JSON Web Tokens in Express.js How To Use JSON Web Tokens (JWTs) in Express.js | DigitalOcean [https://www.digitalocean.com/community/tutorials/nodejs-jwt-expressjs]

5.11. Integrate SSO into your application

- Utilize the SDKs or libraries provided by your chosen SSO provider to integrate SSO functionality into your application
- Implement the necessary code and configurations to enable SSO login and logout flows

Notes

React Authentication By Example: Using React Router 6 [https://developer.auth0.com/resources/guides/spa/react/basic-authentication]

5.12. Test and secure your SSO implementation:

- Thoroughly test your SSO implementation to ensure seamless authentication and authorization across your application
- Implement appropriate security measures like using HTTPS, securely storing and handling user data, and applying necessary security best practices.

Notes

Resource: OWASP Top Ten Project https://owasp.org/www-project-top-ten/

5.13. OAuth 2

- An authorization framework that enables applications — such as Facebook, GitHub, and DigitalOcean — to obtain limited access to user accounts on an HTTP service
- Works by delegating user authentication to the service that hosts a user account and authorizing third-party applications to access that user account
- OAuth 2 provides authorization flows for web and desktop applications, as well as mobile devices

Notes

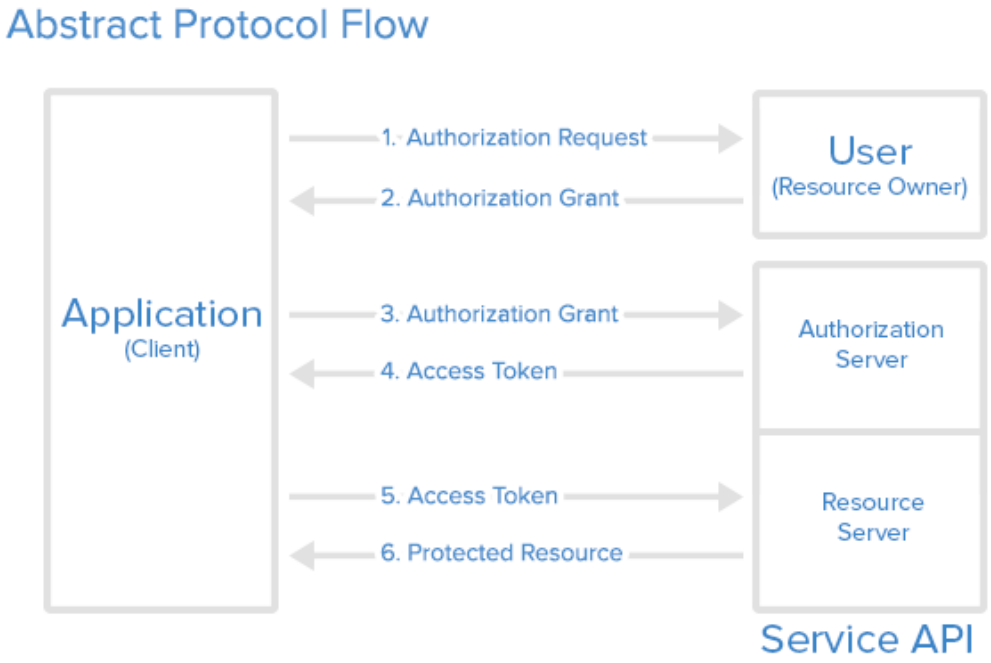
Resources: OAuth 2.0 — OAuth [https://oauth.net/2/] An Introduction to OAuth 2 | DigitalOcean [https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2]

5.14. OAuth Roles

- **Resource Owner**: the *user* who authorizes an *application* to access their account. The application's access to the user's account is limited to the scope of the authorization granted (e.g. read or write access)

- **Client**: the *application* that wants to access the *user*'s account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.
- **Resource Server**: The resource server hosts the protected user accounts.
- **Authorization Server**: The authorization server verifies the identity of the *user* then issues access tokens to the *application*.

5.15. OAuth Protocol Flow



Notes

An organization can build their own OAuth service or use an implementation from a provider such as Amazon or Azure.

1. The *application* requests authorization to access service resources from the *user*
2. If the *user* authorized the request, the *application* receives an authorization grant
3. The *application* requests an access token from the *authorization server* (API) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the *authorization server* (API) issues an access token to the application. Authorization is complete.
5. The *application* requests the resource from the *resource server* (API) and presents the access token for authentication
6. If the access token is valid, the *resource server* (API) serves the resource to the *application*

resource: What is OAuth (The Modern Guide) [https://fusionauth.io/articles/oauth/modern-guide-to-oauth]

5.16. OpenID Connect

- An interoperable authentication protocol based on the OAuth 2.0 framework of specifications (IETF RFC 6749 and 6750)
- Simplifies the way to verify the identity of users based on the authentication performed by an Authorization Server
- Obtains user profile information in an interoperable and REST-like manner

5.17. OpenID Connect

- Enables application and website developers to launch sign-in flows and receive verifiable assertions about users across Web-based, mobile, and JavaScript clients
- Is extensible to support a range of optional features such as encryption of identity data, discovery of OpenID Providers, and session logout

Notes

For developers, it provides a secure and verifiable answer to the question “What is the identity of the person currently using the browser or mobile app that is connected?” Best of all, it removes the responsibility of setting, storing, and managing passwords which is frequently associated with credential-based data breaches.

Chapter 6. IAM and PAM

Objectives

Key objectives of this chapter

- Understand the importance of Identity and Access Management
- Learn about Privileged Access Management
- Discuss tools for IAM and PAM

6.1. Identity and Access Management (IAM)

- A security discipline that focuses on controlling and managing user access to resources within an organization
- For MERN developers, plays a crucial role in ensuring that the right individuals or job roles have the appropriate access to the tools and functionalities they need to perform their tasks

6.2. Identity Management:

- Confirming and storing information about a user's identity, such as their job title and permissions
- Ensures that the user is who they claim to be and authenticates their identity
- An identity management database holds information about the user's identity and attributes

6.3. Access Management:

- Uses the information from identity management to determine what resources and functionalities a user is allowed to access
- Controls and monitors network access, ensuring that users have the necessary permissions to perform their tasks
- Access management systems manage a range of identities, including people, software, and hardware devices

6.4. Authorization:

- The process of determining what actions a user can perform within an application based on their identity, roles, and permissions
- Granting or denying access to specific resources or functionalities based on the user's authorization level

6.5. Privileged Account Management (PAM)

- Privileged accounts: accounts with elevated permissions that have access to critical systems, sensitive data, and administrative functions

- For these accounts, PAM
  - Manages
  - Secures
  - Controls
  - Monitors

6.6. PAM User Audience

- Designed for managing access for
  - privileged users
  - system administrators
  - IT staff
  - other users with elevated privileges

6.7. PAM Access Control

- focuses on securing and controlling access to privileged accounts
- enforces the principle of least privilege
- implements additional security measures:
  - session monitoring
  - password management
  - multi-factor authentication for privileged users

6.8. PAM and IAM Compared

- IAM manages user identities, authentication, and authorization for all users within an organization
- IAM controls access to resources such as applications, databases, and network resources based on user roles, permissions, and policies
- PAM is a subset of IAM that specifically focuses on managing access to privileged accounts and critical resources

6.9. PAM and IAM Compared

- PAM systems enforce stricter controls and monitoring for privileged accounts to prevent unauthorized access and mitigate the risk of insider threats
- PAM helps organizations implement the principle of least privilege, granting privileged access only when necessary and for a limited time
- IAM and PAM are both important for maintaining security, compliance, and effective access management strategies within an organization

6.10. IAM Tools and Libraries

- Okta: A popular IAM platform that provides authentication, authorization, and user management capabilities
- Auth0: A flexible IAM solution that offers features like single sign-on (SSO), social login, and user management
- Keycloak: An open-source IAM solution that provides features like SSO, user federation, and identity brokering
- AWS IAM: Amazon Web Services Identity and Access Management (IAM) service for managing access to AWS resources

6.11. PAM Tools and Libraries

- CyberArk: A leading PAM solution that offers features like password management, session monitoring, and privileged session recording.
- BeyondTrust Privilege Management: A PAM solution that focuses on managing and securing privileged access to critical resources.
- Thycotic Secret Server: A PAM solution that provides secure storage and management of privileged account credentials.
- HashiCorp Vault: A popular open-source tool for managing secrets and protecting privileged access

Chapter 7. Secrets Management

Objectives

Key objectives of this chapter

- Learn about secrets
- Discuss techniques to manage secrets
- Explore tools to protect and use secrets

7.1. Secrets Management Guidelines

- Apply a strong management policy to passwords, tokens, keys, and other privileged information
- Separate secrets from source code
- Use a secure configuration approach
- Employ encryption for secrets
- Use secrets management tools
- Follow other security best practices

7.2. Separate Secrets from Source Code

- Never hardcode sensitive information (such as API keys, database credentials, or passwords) directly into your source code
- Use environment variables or a secrets management solution to store and access these secrets securely
- Consider using a tool like dotenv to load environment variables from a `.env` file during development

7.3. Use a Secure Configuration Approach

- Implement a configuration module or file that holds your secrets and configuration settings
- Ensure that this configuration file is properly protected and only accessible to authorized users
- Avoid committing the configuration file to your version control system (e.g., Git) to prevent accidental exposure
- Use techniques like .gitignore to exclude sensitive files

7.4. Employ Encryption for Secrets

- Encrypt sensitive data at rest, such as credentials stored in databases or files
- Consider using encryption libraries or frameworks provided by your chosen MERN stack components (e.g., bcrypt for password hashing in Node.js, or built-in encryption features of MongoDB)

- Encrypt sensitive data in transit using secure communication protocols (e.g., HTTPS) and encryption libraries (e.g., SSL/TLS)

7.5. Utilize Secrets Management Tools

- Take advantage of secrets management tools specifically designed for developers, such as HashiCorp Vault or AWS Secrets Manager
- These tools offer secure storage, retrieval, and rotation of secrets, as well as access control and audit logs
- Integrate these tools into your MERN stack application by utilizing appropriate client libraries or SDKs

7.6. Follow Security Best Practices

- Regularly update and patch dependencies and libraries used in your MERN stack application to mitigate security vulnerabilities
- Implement proper authentication and authorization mechanisms to ensure only authorized users can access sensitive data and perform privileged actions
- Implement secure coding practices, such as input validation, output encoding, and protection against common web application vulnerabilities like cross-site scripting (XSS) and SQL injection

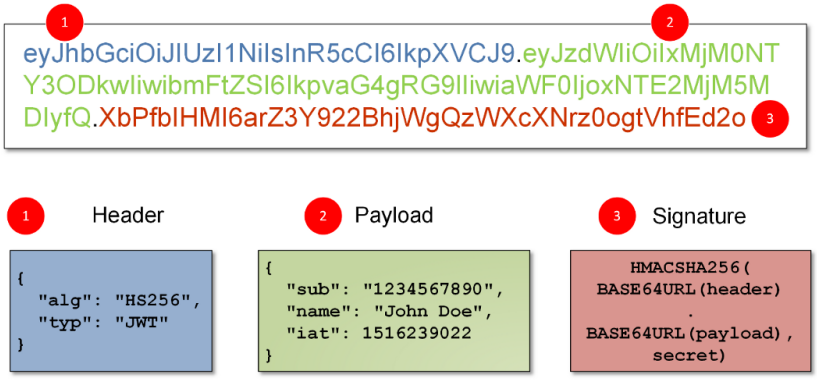
Chapter 8. JSON Web Tokens

Objectives

Key objectives of this chapter

- Examine JSON Web Tokens (JWT)
- Discuss the correct use of JWTs

8.1. JSON Web Tokens (JWT)



Notes

- Open standard for securely transmitting information between parties as a JSON object
- Three parts:
  - Header (1)
  - Payload (2)
  - Signature (3)
- Header and payload are Base64 encoded JSON objects to create a string representation for easier transmission
- A **unique** signature is created by taking the Base64 encoded header and payload, along with a secret key, and running them through a hashing algorithm

8.2. JWT Guidelines

- Use JWT for user authentication and authorization
- Implement JWT in the backend
- Store JWT securely on the client-side
- Refresh and revoke JWTs

8.3. JSON Web Tokens Best Practices

- Cryptography and hashing algorithms should not be implemented from scratch
  - Best to use battle-tested libraries like CryptoJS.
- Server should handle the signing and verification of JWTs, it is the only place that can securely handle secrets and properly verify tokens
- If any part of the payload is altered, the signature will change, which helps ensure the integrity of the JWT and prevents tampering
- JWTs can be transmitted in the URL bar but should be handled carefully to ensure URL safety and prevent potential issues with certain characters

Notes

Sometimes you'll get some extra padding, you can see with:

`jwt.io Window.btoa()` - binary to ASCII (base64)

crypto-js - npm:

https://www.npmjs.com/package/crypto-js

8.4. Signing JWTs

- Use a library like JSON Web Token from npm rather than writing the signing code manually
- Install using npm
- Import into the project using `import * as jwt from 'jsonwebtoken'`
- Use a strong, long, and complex secret key
- Token payload includes claims such as "sub" for subjects and "iss" for issuer

8.5. Signing Tokens

- Use the `jwt.sign()` method
- Pass payload, secret key, and any additional options as arguments to this method
- Set an expiration time for the token using the "exp" claim
- After signing the token, optionally log to the console or process further

8.6. JWT Dos

- Do use long, strong, and unguessable secrets (e.g., 256-bit) for signing tokens
- Do keep the token payload small to reduce the token size
- Do ensure that HTTPS is used to secure the connection

Notes

- Keep the token payload small to reduce the token size, larger tokens result in larger requests to the API, impacting performance
- Do ensure that HTTPS is used to secure the connection. Only send traffic over secure SSL connections to prevent interception of requests and token theft. Valid certificates are crucial for maintaining a secure connection.

8.7. JWT Don’ts

- Don’t store tokens in local storage
- Don’t store secret keys used for signing tokens in the browser
- Don’t decode tokens on the client side, especially access tokens
- Don’t put anything in the body of a JWT that the user/client should not be able to see

Notes

Don’t store tokens in local storage; it’s easily scriptable, and malicious JavaScript can steal tokens, leading to potential impersonation of users. Consider using HttpOnly cookies or keeping tokens in React state, in browser memory.

Don’t store secret keys used for signing tokens in the browser. Keep secret keys on the backend, where they can be properly secured.

Don’t decode tokens on the client side, especially access tokens. Access tokens are meant for APIs and should only be read at the API level. To retrieve payload information, consider using specific API endpoints or user info obtained during sign-in or sign-up.

Don’t put anything in the body of a JWT that the user/client should not be able to see. no key is necessary to decode the token and read the contents - treat any data stored in a JWT as plain text.

Chapter 9. Risk Management, Governance, and Compliance

Objectives
Key objectives of this chapter ▪ Learn general secure architecture design principles ▪ Learn about the NIST Cybersecurity Framework ▪ Explore open source vulnerability management tools

9.1. Secure Architecture and Design General Principles

- Use enterprise tools and patterns
- Protect sensitive data
- Think Zero Trust!
- Provide Resiliency (backup/restore)
- Server-side validation
- Isolate security functions
- Application partitioning
- Re-authorize high-value transactions
- Provide logging and monitoring

9.2. Secure Architecture and Design for MERN Principles

- Implement user authentication and authorization
- Protect against cross-site scripting (XSS) attacks
- Prevent cross-site request forgery (CSRF) attacks
- Securely handle and store sensitive data
- Regularly update and patch dependencies

9.3. Supply Chain Security

- Understand and vet open-source resources and third-party tools
- Regularly update dependencies
- Verify package authenticity
- Implement secure coding practices
- Continuously monitor for vulnerabilities

9.4. NIST Cybersecurity Framework

- Provides a set of guidelines, best practices, and standards for managing and improving

cybersecurity risk

- Helps organizations assess and enhance their cybersecurity posture by focusing on five core functions
  - Identify
  - Protect
  - Detect
  - Respond
  - Recover

Notes

Resource: Cybersecurity Framework, NIST: https://www.nist.gov/cyberframework

9.5. Identify

- Identify critical enterprise processes and assets
- Document information flows
- Maintain hardware and software inventory
- Establish policies for cybersecurity that include roles and responsibilities
- Identify threats, vulnerabilities, and risk to assets

Notes

Identify critical enterprise processes and assets

What are your enterprise's activities that absolutely must continue in order to be viable? For example, this could be maintaining a website to retrieve payments, protecting customer/patient information securely, or ensuring that the information your enterprise collects remains accessible and accurate.

Document information flows

It's important to not only understand what type of information your enterprise collects and uses, but also to understand where the data is located and how it is used, especially where contracts and external partners are engaged.

Maintain hardware and software inventory

It’s important to have an understanding of the computers and software in your enterprise because these are frequently the entry points of malicious actors. This inventory could be as simple as a spreadsheet.

Establish policies for cybersecurity that include roles and responsibilities

These policies and procedures should clearly describe your expectations for how cybersecurity activities will protect your information and systems, and how they support critical enterprise processes. Cybersecurity policies should be integrated with other enterprise risk considerations (e.g.,financial, reputational).

Identify threats, vulnerabilities, and risk to assets

Ensure risk management processes are established and managed to ensure internal and external threats are identified, assessed, and documented in risk registers. Ensure risk responses are identified and prioritized, executed, and results monitored.

9.6. Protect

- Manage access to assets and information
- Protect sensitive data
- Conduct regular backups
- Protect your devices
- Manage device vulnerabilities
- Train users

Notes

Manage access to assets and information

Create unique accounts for each employee and ensure that users only have access to information, computers, and applications that are needed for their jobs. Authenticate users (e.g., passwords, multi-factor techniques) before they are granted access to information, computers, and applications. Tightly manage and track physical access to devices.

Protect sensitive data

If your enterprise stores or transmits sensitive data, make sure that this data is protected by encryption both while it’s stored on computers as well as when it’s transmitted to other parties. Consider utilizing integrity checking to ensure only approved changes to the data have been

made. Securely delete and/or destroy data when it's no longer needed or required for compliance purposes.

Conduct regular backups

Many operating systems have built-in backup capabilities; software and cloud solutions are also available that can automate the backup process. A good practice is to keep one frequently backed up set of data offline to protect it against ransomware.

Protect your devices

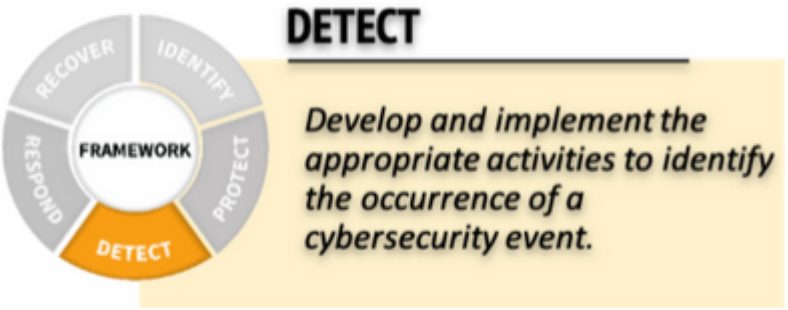
Consider installing host-based firewalls and other protections such as endpoint security products. Apply uniform configurations to devices and control changes to device configurations. Disable device services or features that are not necessary to support mission functions. Ensure that there is a policy and that devices are disposed of securely.

Manage device vulnerabilities

Regularly update both the operating system and applications that are installed on your computers and other devices to protect them from attack. If possible, enable automatic updates. Consider using software tools to scan devices for additional vulnerabilities; remediate vulnerabilities with high likelihood and/or impact.

Train users Regularly train and retrain all users to be sure that they are aware of enterprise cybersecurity policies and procedures and their specific roles and responsibilities as a condition of employment.

9.7. Detect



- Test and update detection processes
- Maintain and monitor logs
- Know the expected data flows for your enterprise
- Understand the impact of cybersecurity events

Notes

Test and update detection processes

Develop and test processes and procedures for detecting unauthorized entities and actions on the networks and in the physical environment, including personnel activity. Staff should be aware of their roles and responsibilities for detection and related reporting both within your

organization and to external governance and legal authorities.

Maintain and monitor logs

Logs are crucial in order to identify anomalies in your enterprise's computers and applications. These logs record events such as changes to systems or accounts as well as the initiation of communication channels. Consider using software tools that can aggregate these logs and look for patterns or anomalies from expected network behavior

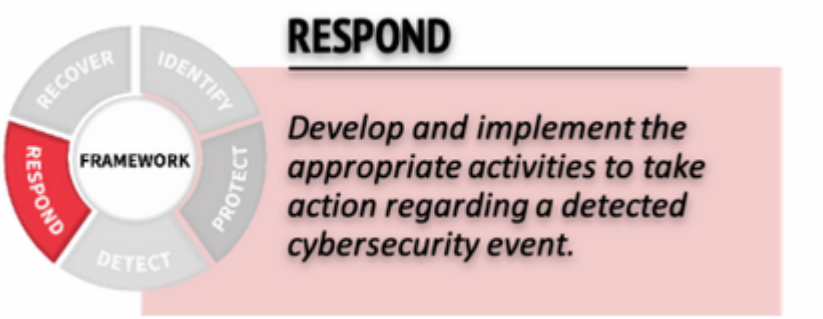
Know the expected data flows for your enterprise

If you know what and how data is expected to be used for your enterprise, you are much more likely to notice when the unexpected happens and unexpected is never a good thing when it comes to cybersecurity. Unexpected data flows might include customer information being exported from an internal database and exiting the network. If you have contracted work to a cloud or managed service provider, discuss with them how they track data flows and report, including unexpected events.

Understand the impact of cybersecurity events

If a cybersecurity event is detected, your enterprise should work quickly and thoroughly to understand the breadth and depth of the impact. Seek help. Communicating information on the event with appropriate stakeholders will help keep you in good stead in terms of partners, oversight bodies, and others (potentially including investors) and improve policies and processes.

9.8. Respond



- Ensure response plans are tested
- Ensure response plans are updated
- Coordinate with internal and external stakeholders

Notes

Ensure response plans are tested

It's even more important to test response plans to make sure each person knows their responsibilities in executing the plan. The better prepared your organization is, the more effective the response is likely to be. This includes knowing any legal reporting requirements or required information sharing.

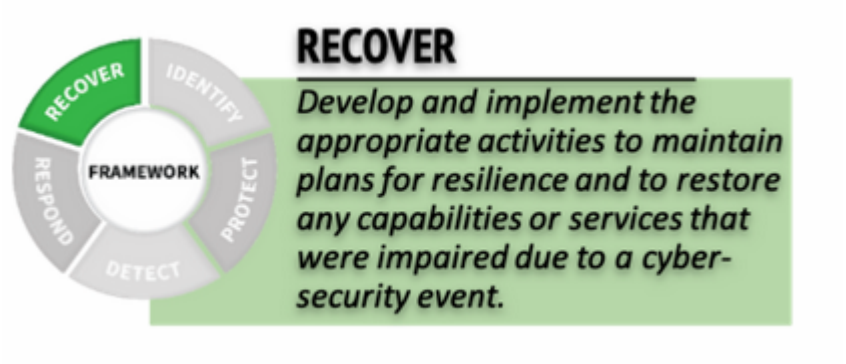
Ensure response plans are updated

Testing the plan (and execution during an incident) inevitably will reveal needed improvements. Be sure to update response plans with lessons learned.

Coordinate with internal and external stakeholders

It's important to make sure that your enterprise's response plans and updates include all key stakeholders and external service providers. They can contribute to improvements in planning and execution.

9.9. Recover



- Communicate with internal and external stakeholders
- Ensure recovery plans are updated
- Manage public relations and company reputation

Notes

Communicate with internal and external stakeholders

Part of recovery depends upon effective communication. Your recovery plans need to carefully account for what, how, and when information will be shared with various stakeholders so that all interested parties receive the information they need but no inappropriate information is shared.

Ensure recovery plans are updated

As with response plans, testing execution will improve employee and partner awareness and highlight areas for improvement. Be sure to update Recovery plans with lessons learned.

Manage public relations and company reputation

One of the key aspects of recovery is managing the enterprise's reputation. When developing a recovery plan, consider how you will manage public relations so that your information sharing is accurate, complete, and timely and not reactionary.

9.10. Open-source Vulnerability Management

- Identifies vulnerabilities in software and systems
- Guides risk assessment
- Provides steps for patching and mitigation

- Useful for continuous monitoring
- Provides a collaborative and engaged community

Chapter 10. Data Classification and Protection

Objectives
Key objectives of this chapter ▪ Discuss the importance of labeling and classifying data ▪ Learn Travelers data classifications

10.1. Classification Controls

- Add access controls
- Encrypt data in transit and at rest
- Log and audit data access
- Periodically review classifications

10.2. Define and Document

- Backup and restoration procedures
- Change control procedures
- Proper data disposal

10.3. Travelers Data Classifications

10.4. Restricted information

- Highly sensitive information
- Unauthorized use or disclosure could seriously and adversely impact Travelers, its business or its competitive advantage.

10.5. Regulated Personal Information

- Warrants special protection and is identified by Travelers Policy

10.6. Confidential information

- Sensitive information whose unauthorized use or disclosure could negatively impact Travelers, its business, or its competitive advantage
- Includes identifiable personal information that does not meet the definition of Regulated Personal Information

10.7. Internal information

- Commonly shared within Travelers
- Does not meet the standard for Restricted information, Regulated Personal Information, or Confidential information but is nevertheless **not intended for distribution outside Travelers**

10.8. Public information

- Is or has been made freely available outside of Travelers or is intended for public use

Chapter 11. Threat Modeling

Objectives
Key objectives of this chapter ▪ Discuss steps to create a threat model for an application ▪ Learn about various threat model techniques such as STRIDE or DREAD

11.1. Basic Steps

1. Decompose the application
2. Determine and rank threats
3. Determine countermeasures and mitigations

11.2. 1. Decompose the Application

- Create use cases to understand how the application is used
- Identifying entry points to see where a potential attacker could interact with the application
- Identifying assets (items or areas that the attacker would be interested in)
- Identifying trust levels that represent the access rights that the application will grant to external entities
- Data flow diagrams can help illustrate

11.3. 2. Determine and Rank Threats

- Use a threat categorization methodology
  - STRIDE
  - Application Security FRAME
  - PASTA
  - DREAD
- Identify threats from the attacker
- Identify the defensive perspective
- Use threats to build threat trees
  - One tree for each threat goal

11.4. STRIDE Overview

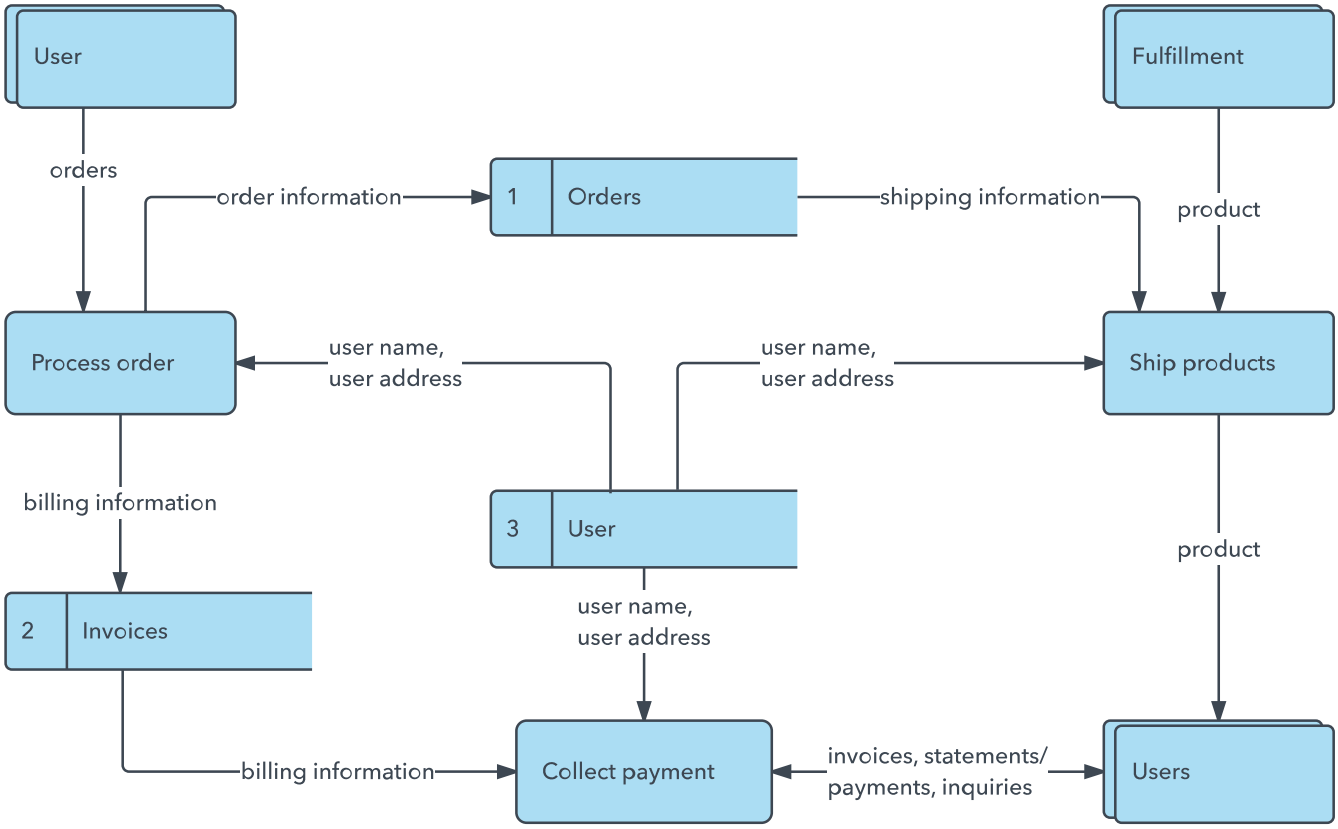
Threat	Desired Property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiation

Threat	Desired Property
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

11.5. DREAD Risk Assessment

Threat Category	Description
Damage	How bad would an attack be?
Reproducibility	How easy is it to reproduce the attack?
Exploitability	How much work is it to launch the attack?
Affected users	How many people will be impacted?
Discoverability	How easy is it to discover the threat?

11.6. Data flow



Notes

Lucidchart is a good tool to create data flow diagrams. The data flow diagram shows you data inputs, outputs, where data is stored, and how it gets from one point to another.

These are points where you need to be concerned about security of your data. You can look at each point, using STRIDE or DREAD as your guide, and simply ask "What could go wrong?" or "What is at risk?".

A quick overview of data flow diagrams at lucidchart: What is a Data Flow Diagram | Lucidchart [https://www.lucidchart.com/pages/data-flow-diagram]

Chapter 12. MongoDB Security Best Practices

|== |

a|Key objectives of this chapter

- Discuss MongoDB security best practices
- Learn about MongoDB database encryption |==

12.1. Use strong authentication mechanisms:

- MongoDB provides several authentication mechanisms
  - SCRAM
  - x.509
  - LDAP
- Create separate users with different roles to limit access

12.2. Limit remote access

- Reduces the risk of data leaks
- Recommended to whitelist specific IP addresses for remote connections

12.3. Implement role-based access control

- Use role-based access control to define and enforce access permissions for different users or processes accessing MongoDB
- Helps ensure that only authorized users have the necessary privileges

12.4. Encrypt your data

- Encrypting your data adds an extra layer of security
- MongoDB provides features for data encryption, including encryption at rest and encryption in transit
- Implementing encryption helps protect sensitive data from unauthorized access

12.5. Keep MongoDB up to date

- Regularly update MongoDB to the latest version to ensure that you have the latest security patches and bug fixes
- MongoDB releases updates and security patches to address vulnerabilities and improve security

12.6. Implement proper input validation

- Ensure that you properly validate and sanitize user input to prevent common security

vulnerabilities such as

- NoSQL injection
- Cross-site scripting (XSS) attacks

12.7. Enable auditing and monitoring

- Enable auditing and monitoring features provided by MongoDB to track and log activities in the database
- Can help detect and investigate any suspicious or unauthorized activities

Notes

Resources:

https://www.linkedin.com/pulse/strengthening-security-essential-authentication-practices-rimu

https://www.elluminatiinc.com/mern-stack-security/

https://www.linkedin.com/pulse/securing-mern-stack-applications-protecting-against-common-shawon

https://www.mongodb.com/features/security/best-practices

https://www.freecodecamp.org/news/how-to-secure-your-mern-stack-application/

https://www.21twelveinteractive.com/develop-web-app-with-mern-stack-development/

12.8. Encrypting Data in MongoDB

- To encrypt data in MongoDB, you can use the Client-Side Field Level Encryption (CSFLE) feature.
- IMPORTANT: MongoDB cannot encrypt **existing** data (If you have existing data, you'll need to follow additional steps to encrypt it.)
- MongoDB also provides encryption at rest, which encrypts the data files stored on disk, ensures that the data remains encrypted even if the physical storage is compromised

=HTTP Security Headers

Objectives
Key objectives of this chapter ▪ Learn about HTTP security headers ▪ Discuss helmet.js

12.9. What are HTTP Security Headers?

- HTTP Security Headers are a set of response headers that instruct the browser on how to behave when handling your web application.

- These headers add an extra layer of security, preventing various attacks and vulnerabilities.
- Some common security headers include Content Security Policy (CSP), X-XSS-Protection, X-Frame-Options, etc.

12.10. helmet.js

- A crucial security package that helps secure your HTTP headers and protect your MERN applications from common web vulnerabilities.

Notes

Resources:

The helmet.js web site: https://helmetjs.github.io/

Scan your site with SecurityHeaders.io: https://securityheaders.com/

12.11. Why Use helmet.js?

- Writing these headers manually can be cumbersome and error-prone.
- helmet.js simplifies the process by providing an easy-to-use middleware to set these headers securely.
- It is widely used, well-maintained, and compatible with Express.js, a popular framework in MERN stack development.

12.12. Installing helmet.js

- Before using helmet.js, we need to install it in our MERN application.
- Open your terminal and run: `npm install helmet`

12.13. Implementing helmet.js

- Import helmet in your Express.js server file:

```
const express = require('express');
const helmet = require('helmet');
const app = express();

// Use helmet middleware
app.use(helmet());
```

12.14. Content Security Policy (CSP)

- A critical security header is Content Security Policy (CSP)
- CSP restricts the sources from which your application can load content, mitigating XSS and data injection attacks

- Implement CSP using helmet.js

```
const express = require('express');
const helmet = require('helmet');
const app = express();

// Use helmet middleware with Content Security Policy
app.use(helmet.contentSecurityPolicy({
  directives: {
    defaultSrc: ["'self'"],
    scriptSrc: ["'self'", "'unsafe-inline'"],
    styleSrc: ["'self'", "'unsafe-inline'"]
  }
}));
```

12.15. X-XSS-Protection and X-Frame-Options

- X-XSS-Protection header enables the browser's Cross-Site Scripting (XSS) filter
- X-Frame-Options prevents your application from being embedded in a frame on another site, mitigating clickjacking attacks

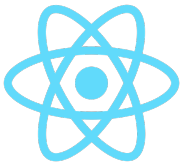
```
const express = require('express');
const helmet = require('helmet');
const app = express();

// Use helmet middleware with X-XSS-Protection and X-Frame-Options
app.use(helmet.xssFilter());
app.use(helmet.frameguard({ action: 'sameorigin' }));
```

Chapter 13. React Security Best Practices

Objectives
Key objectives of this chapter ▪ Discuss security best practices particular to React ▪ Explore resources for React security

13.1. The React Virtual DOM is not enough



- The Virtual DOM provides some security benefits
- Does not address all security concerns and should be part of a broader security strategy
- Consider:
  - Security Vulnerabilities in Dependencies
  - Server-Side Rendering (SSR) Vulnerabilities
  - Proper Use of Props and State

Notes

React applications often depend on numerous third-party libraries and packages. Vulnerabilities in these dependencies can pose security risks. Regularly updating dependencies and using tools to monitor for known vulnerabilities can help mitigate this risk.

For applications using server-side rendering with React, there are specific security considerations. For example, improperly handled user input in SSR can lead to XSS vulnerabilities. Developers need to ensure that user input is sanitized and validated, even when using SSR.

React's security model relies on the proper use of props and state. Developers should avoid dangerous patterns, such as directly inserting user input into the DOM without sanitization or using dangerouslySetInnerHTML without proper precautions.

13.2. Regularly update third-party packages and dependencies

- Keeping your packages and dependencies up to date helps reduce vulnerabilities
- Make sure to regularly check for updates and apply them to your project

13.3. Encrypt sensitive data

- Ensure that sensitive data is encrypted both when stored and transmitted over the network
- Helps protect the confidentiality of the data
- Use encryption libraries or frameworks to implement encryption in your React application

13.4. Implement authentication and authorization

- Properly authenticate and authorize users to access your application
- Use secure authentication mechanisms such as JSON Web Tokens (JWT)
- Implement authorization checks to control access to different parts of your application

13.5. Validate and sanitize user input

- Validate and sanitize user input to prevent common security vulnerabilities such as cross-site scripting (XSS) and SQL injection attacks
- Use libraries or frameworks that provide input validation and sanitization features

13.6. Protect against cross-site scripting (XSS) attacks

- XSS attacks can be prevented by properly validating and sanitizing user input
- Use security headers like Content Security Policy (CSP) to mitigate the risk of XSS attacks

13.7. Securely handle session management

- If your application uses sessions, make sure to handle session management securely
- Use secure session storage mechanisms and implement measures to prevent session hijacking or session fixation attacks

13.8. Implement secure communication

- Use secure communication protocols such as HTTPS to encrypt the communication between the client and server
- Helps protect the integrity and confidentiality of the data transmitted over the network

13.9. Follow React security best practices

- Familiarize yourself with React-specific security best practices
- This includes disabling any instruction-containing markup, using snippet libraries, conducting code audits, and following React security checklists

13.10. Keep your code modular and readable

- Writing modular and readable code helps improve maintainability and reduces the risk of introducing security vulnerabilities

- Follow best practices for code organization, naming conventions, and documentation for your team

13.11. Perform security testing

- Regularly perform security testing on your application to identify and address any security vulnerabilities
- This can include manual code reviews, automated security scanning, and penetration testing

Notes

Resources:

https://www.elluminatiinc.com/mern-stack-security/

https://www.linkedin.com/pulse/securing-mern-stack-applications-protecting-against-common-shawon

https://aglowiditsolutions.com/blog/reactjs-security-best-practices/

https://www.freecodecamp.org/news/how-to-secure-your-mern-stack-application/

https://www.cmarix.com/blog/react-best-practices/

https://gaper.io/mern-stack-tips-tricks/

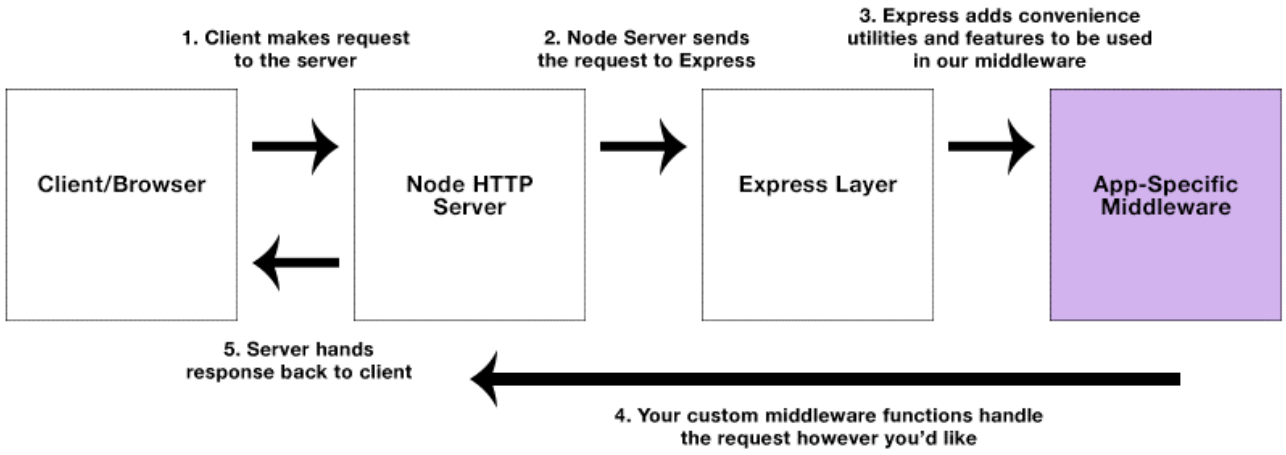
Chapter 14. Node Security Best Practices

|== |

a|Key objectives of this chapter

- Discuss best practices to keep Node secure
- Explore resources to secure Node |==

14.1. Node in the application flow



Notes

As a reminder, let's look at where node fits into a typical web application. As you develop your threat models, node is a boundary between the client and the Express Layer.

14.2. Regularly update packages and dependencies

- Keep your codebase up to date by regularly updating third-party packages and dependencies
- Helps to reduce vulnerabilities and ensures that you are using the latest security patches

14.3. Encrypt sensitive data

- Ensure that sensitive data is encrypted both when stored and transmitted over the network
- Helps to protect the confidentiality of the data and prevent unauthorized access

14.4. Implement secure authentication

- Follow best practices for secure authentication in MERN stack applications
- This includes using:
  - Strong password hashing algorithms
  - Implementing multi-factor authentication

- Protecting against common vulnerabilities such as SQL injections and cross-site scripting (XSS) attacks

14.5. Use security linters and scanners

- Take advantage of security linters, such as ESLint with security rules or Snyk, to identify and fix potential security issues in your code
- These linters can help catch common security vulnerabilities and enforce best practices

Notes

Snyk can be incorporated directly into Visual Studio Code.

14.6. Secure transmission of data

- Ensure that data transmitted over the network is done securely using HTTPS
- Helps to protect the integrity and confidentiality of the data during transit

14.7. Control access and permissions

- Implement proper access controls and permissions to restrict unauthorized access to sensitive resources
- Use role-based access control (RBAC) or other access control mechanisms to enforce least privilege principles

14.8. Validate and sanitize user input

- Always validate and sanitize user input to prevent common security vulnerabilities such as SQL injections and cross-site scripting (XSS) attacks
- Use input validation techniques and sanitize user input before using it in database queries or rendering it in HTML templates

14.9. Implement logging and monitoring

- Set up logging and monitoring mechanisms to track and detect any suspicious activities or security incidents
- Monitor logs and system metrics to identify potential security breaches and respond to them in a timely manner

14.10. Perform security testing

- Conduct regular security testing, including vulnerability scanning and penetration testing, to identify and address any security weaknesses in your application
- Helps to identify and fix vulnerabilities before they can be exploited by attackers

14.11. Follow security best practices for the entire MERN stack

- Remember that security is not limited to just Node.js
- Ensure that you follow security best practices for all components of the MERN stack

14.12. Security is a PROCESS

- Security is a continuous process
- Stay informed about the latest security vulnerabilities
- Seek out groups and mentors to support you

Notes

Resources https://www.elluminatiinc.com/mern-stack-security/ https://www.linkedin.com/pulse/securing-mern-stack-applications-protecting-against-common-shawon https://www.linkedin.com/pulse/strengthening-security-essential-authentication-practices-rimu https://www.decipherzone.com/blog-detail/best-practices-node-js-security https://www.cmarix.com/blog/node-js-best-practices/ https://snyk.io/learn/nodejs-security-best-practice/