Introduction To The Web & The HTTP Protocol

Course Introduction

Alexis Ahmed

Senior Penetration Tester @HackerSploit

Offensive Security Instructor @INE

Course Topic Overview

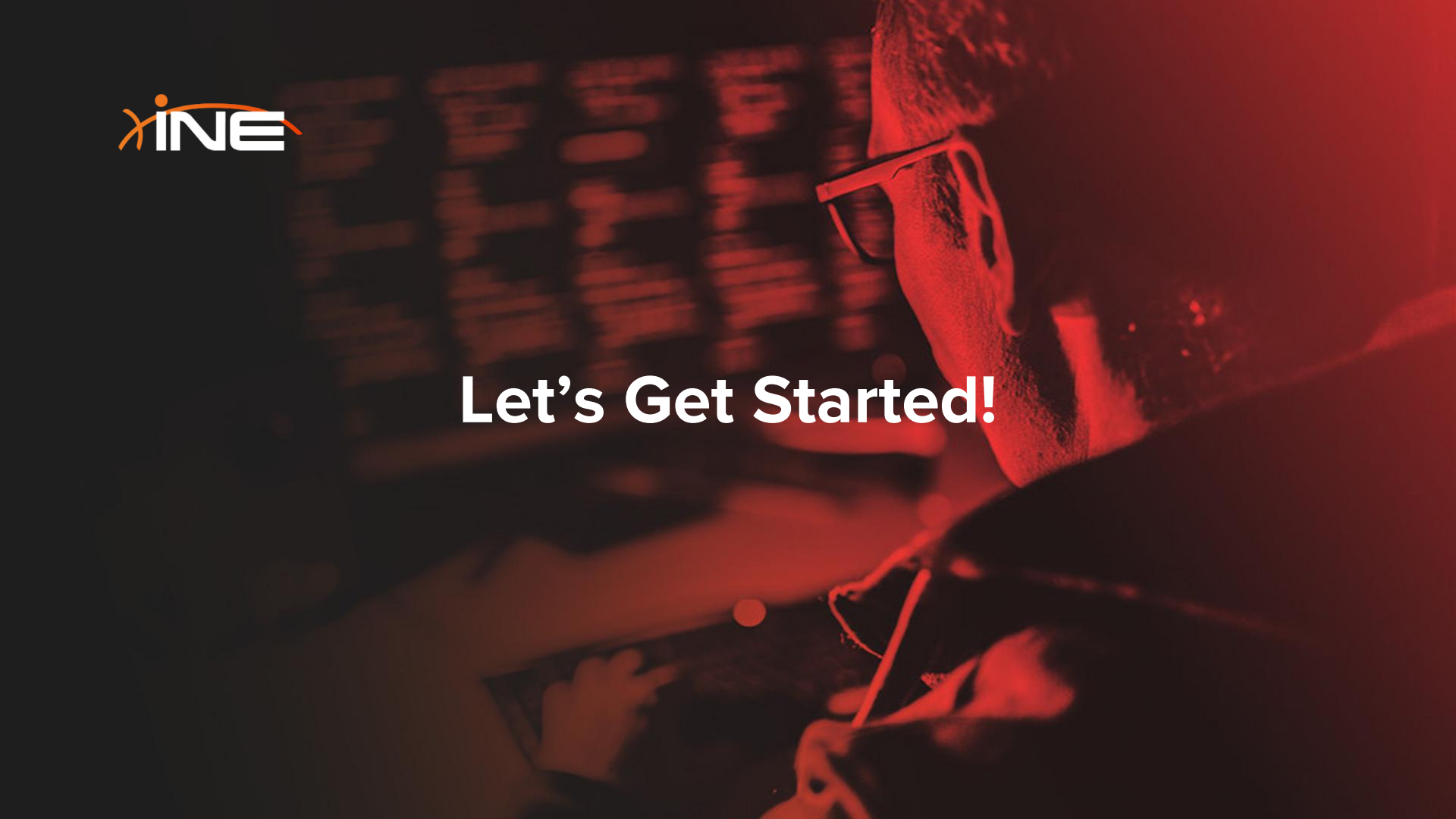
- Introduction To Web Application Security Testing
- Common Web Application Threats
- Web Application Architecture
- Web Application Technologies
- HTTP Protocol Fundamentals
- HTTP Requests & Responses
- HTTPS

Prerequisites

- Basic familiarity with Linux.
- Familiarity with Cybersecurity terms and concepts.

Learning Objectives:

- You will have a solid understanding of what web applications are, how they work and how they are deployed.
- You will have a solid understanding of importance of securing web applications.
- You will have an understanding of the various components that make up a web application's architecture.
- You will have an understanding of common web application security best practices and why they are implemented.
- You will have a good understanding of what Web Application Security Testing is, what it entails and why it is performed.
- You will be able to differentiate between the various types of web application security tests and will have an understanding of the differences between a web application security test and a web app pentest.
- You will have a functional and practical understanding of how HTTP/S works and will be able to analyze HTTP requests and responses.
- You will have an understanding of the various HTTP request and response headers, methods and status codes.


Let’s Get Started!

Introduction To Web Application Security

What Are Web Applications?

- Web applications are software programs that run on web servers and are accessible over the internet through web browsers.
- They are designed to provide interactive and dynamic functionality to users, allowing them to perform various tasks, access information, and interact with data online.
- Web applications have become an integral part of modern internet usage, and they power a wide range of online services and activities.
- Examples of web applications include:
  - Social media platforms (e.g., Facebook, Twitter)
  - Online email services (e.g., Gmail, Outlook)
  - E-commerce websites (e.g., Amazon, eBay)
  - Cloud-based productivity tools (e.g., Google Workspace, Microsoft Office 365)

How Do Web Applications Work?

- This Client-Server Architecture: Web applications follow the client-server model, where the application's logic and data are hosted on a web server, and users access it using web browsers on their devices.
- User Interface (UI): The user interface of web applications is usually presented through a combination of HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript to create dynamic and interactive interfaces.
- Internet Connectivity: Web applications require an internet connection for users to access them. Users interact with the application by sending requests to the server, which processes those requests and sends back the appropriate responses.

How Do Web Applications Work?

- Cross-Platform Compatibility: Web applications are accessible from different devices and operating systems without requiring installation or specific software, making them platform-independent.
- Statelessness: HTTP, the protocol used for communication between web browsers and servers, is stateless. Web applications must manage user sessions and state to remember user interactions and ensure continuity.

Web Application Security

- Web application security is a critical aspect of cybersecurity that focuses on protecting web applications from various security threats and vulnerabilities, and attacks.
- The primary objective of web application security is to ensure the confidentiality, integrity, and availability of data processed by web applications while mitigating the risk of unauthorized access, data breaches, and service disruptions.
- Web applications are attractive targets for attackers due to their public accessibility and the potential for gaining access to sensitive data, such as personal information, financial data, or intellectual property.

The Importance Of Web App Security

- Web application security is of paramount importance in today's digital landscape due to the increasing reliance on web applications for various purposes.
- Here are some key reasons why web application security is crucial:
  - Protection of Sensitive Data: Web applications often handle sensitive user data such as personal information, financial details, and login credentials. A security breach in a web application can lead to unauthorized access and exposure of this sensitive data, leading to severe privacy and compliance issues.
  - Safeguarding User Trust: Users expect that the web applications they interact with are secure and will protect their information. A security incident can erode user trust, resulting in a loss of customers, reputation damage, and negative publicity.

The Importance Of Web App Security

- Prevention of Financial Loss: Web application attacks can lead to financial losses for both organizations and individuals. For businesses, breaches may result in financial theft, intellectual property theft, and even legal penalties.
- Compliance and Regulatory Requirements: Many industries have strict compliance and regulatory requirements, such as GDPR, HIPAA, and PCI DSS, that mandate the implementation of strong security measures for web applications.
- Mitigation of Cyber Threats: The threat landscape is constantly evolving, with new attack techniques emerging regularly. Ensuring robust web application security helps mitigate the risk of falling victim to various cyber threats, including hacking, data breaches, and ransomware.

The Importance Of Web App Security

- Protection Against DDoS Attacks: Web applications are often targeted by Distributed Denial of Service (DDoS) attacks, which aim to overwhelm the application's infrastructure and make it unavailable to legitimate users.
- Maintaining Business Continuity: Web applications are critical for business operations, and any disruption to their availability can lead to downtime and productivity loss. Robust security measures help maintain business continuity and prevent costly disruptions.
- Preventing Defacement and Data Manipulation: Web application vulnerabilities can be exploited to deface web pages, alter content, or inject malicious code, damaging the organization's brand and credibility.

Web Application Security Practices

- Authentication and Authorization: Implementing robust authentication mechanisms to verify the identity of users and authorization controls to grant appropriate access privileges based on user roles.
- Input Validation: Ensuring that all data inputs from users are validated to prevent common attacks like SQL injection and cross-site scripting (XSS).
- Secure Communication: Using encryption protocols like HTTPS (TLS/SSL) to secure the communication between the user's browser and the web server, protecting sensitive data in transit.
- Secure Coding Practices: Adhering to secure coding standards and practices to minimize the introduction of vulnerabilities during the development phase.

Web Application Security Practices

- Regular Security Updates: Keeping the web application and its underlying software libraries up to date with the latest security patches and updates.
- Least Privilege Principle: Assigning the minimum necessary privileges to users, processes, and systems to reduce the potential impact of a security breach.
- Web Application Firewalls (WAF): Implementing a WAF to filter and monitor HTTP requests, blocking malicious traffic and protecting against known attack patterns.
- Session Management: Implementing secure session handling to prevent session hijacking and ensure the user's identity is maintained securely throughout the session.

Web Application Security Testing

Web Application Security Testing

- Web application security testing is the process of evaluating and assessing the security aspects of web applications to identify vulnerabilities, weaknesses, and potential security risks.
- It involves conducting various tests and assessments to ensure that web applications are resistant to security threats and can effectively protect sensitive data and functionalities from unauthorized access or malicious activities.
- The primary goal of web application security testing is to uncover security flaws before they are exploited by attackers.
- By identifying and addressing vulnerabilities, organizations can enhance the overall security posture of their web applications, reduce the risk of data breaches and unauthorized access, and protect their users and sensitive information.

Web Application Security Testing Types

- Web application security testing typically involves a combination of automated scanning tools and manual testing techniques.
- Some common types of security testing conducted on web applications include:
  - Vulnerability Scanning: Using automated tools to scan the web application for known vulnerabilities, such as SQL injection, Cross-Site Scripting (XSS), insecure configurations, and outdated software versions.
  - Penetration Testing: Simulating real-world attacks to assess the application's defenses and identify potential security weaknesses. This involves ethical hacking to gain insights into how an attacker might exploit vulnerabilities.
  - Code Review and Static Analysis: Manual examination of the application's source code to identify coding flaws, security misconfigurations, and potential security risks.

Web Application Security Testing Types

- Authentication and Authorization Testing: Evaluating the effectiveness of authentication mechanisms and access control features to ensure that only authorized users have appropriate access levels.
- Input Validation and Output Encoding Testing: Assessing how the application handles user inputs to prevent common security vulnerabilities like XSS and SQL injection.
- Session Management Testing: Verifying how the application manages user sessions and related tokens to prevent session-related attacks.
- API Security Testing: Assessing the security of APIs (Application Programming Interfaces) used by the web application for data exchange and integration with other systems.

Web Application Penetration Testing

- Web application pentesting, is a subset of web application security testing that specifically involves attempting to exploit identified vulnerabilities.
- It is a simulated attack on the web application conducted by skilled security professionals known as pentesters, bug bounty hunters or ethical hackers.
- The process involves a systematic and controlled approach to assess the application's security by attempting to exploit known vulnerabilities.

Web App Pentesting vs Web App Security Testing

- Key differences between web app security testing and web app pentesting:
  - **Scope:** Web application security testing covers a broader range of assessments, including static and dynamic analysis, while web application pentesting focuses on actively exploiting vulnerabilities.
  - **Objective:** The primary goal of security testing is to identify weaknesses, whereas pentesting aims to validate vulnerabilities and assess the organization's ability to detect and respond to attacks.
  - **Methodology:** Security testing includes both manual and automated techniques, while pentesting is predominantly a manual process, involving the use of various tools and techniques for exploitation.
  - **Exploitation:** Security testing does not involve exploitation of vulnerabilities, while pentesting does, albeit in a controlled and authorized manner.

Web App Pentesting vs Web App Security Testing

Aspect	Web App Security Testing	Web App Pentesting
Objective	Identify vulnerabilities and weaknesses in the web application without actively exploiting them.	Actively attempt to exploit identified vulnerabilities and assess the organization's response to attacks.
Focus	Broader in scope, includes both manual and automated testing techniques.	Specific to identifying vulnerabilities and exploiting them, mainly a manual process.
Methodology	Various types of assessments, such as SAST, DAST, IAST, SCA, etc.	Manual testing using tools and techniques to simulate real-world attacks.
Exploitation	Does not involve exploitation of vulnerabilities.	Involves controlled exploitation to validate vulnerabilities.
Impact	Non-intrusive; primarily focused on identifying issues.	Can be intrusive, may cause application disruption during testing.
Reporting	Identifies vulnerabilities and provides remediation recommendations.	Documents successful exploits, identifies weaknesses, and recommends remediation measures.
Testing Approach	May include automation for vulnerability scanning.	Primarily manual, using manual testing techniques and tools.
Goal	Enhance overall security posture of the web application.	Validate the effectiveness of existing security controls and incident response capabilities.

Common Web Application Threats & Risks

Common Web Application Threats & Risks

- Given the increased adoption of web applications, it comes as no surprise that web apps are constantly exposed to various security threats and risks due to their widespread accessibility and the valuable data they process.
- Understanding these common security threats is crucial for developers, security professionals, and organizations to implement effective measures and safeguard their web applications.
- The actual impact and severity of each threat may vary depending on the specific web application and its security measures.
- Web application security requires a proactive and comprehensive approach to mitigate these threats and protect sensitive data and user interactions effectively.

Threat vs Risk

- **Threat:**
  - A threat refers to any potential source of harm or adverse event that may exploit a vulnerability in a system or organization's security measures.
  - Threats can be human-made, such as cybercriminals, hackers, or insiders with malicious intent, or they can be natural, such as floods, earthquakes, or power outages.
  - In the context of cybersecurity, threats can include various types of attacks, like malware infections, phishing attempts, denial-of-service attacks, and data breaches.

Threat vs Risk

- **Risk:**
  - Risk is the potential for a loss or harm resulting from a threat exploiting a vulnerability in a system or organization.
  - It is a combination of the likelihood or probability of a threat occurrence and the impact or severity of the resulting adverse event.
  - Risk is often measured in terms of the likelihood of an incident happening and the potential magnitude of its impact.

- In summary, a threat is a potential danger or harmful event, while risk is the probability and potential impact of that event happening.
- Threats can exist, but they may or may not pose a significant risk depending on the vulnerabilities and security measures in place to mitigate them.

Common Web Application Threats & Risks

Threat/Risk	Description
Cross-Site Scripting (XSS)	Attackers inject malicious scripts into web pages viewed by other users, leading to unauthorized access to user data, session hijacking, and browser manipulation.
SQL Injection (SQLi)	Attackers manipulate user input to inject malicious SQL code into the application's database, leading to unauthorized data access, data manipulation, or database compromise.
Cross-Site Request Forgery (CSRF)	Attackers trick authenticated users into unknowingly performing actions on a web application, such as changing account details, by exploiting their active sessions.
Security Misconfigurations	Improperly configured servers, databases, or application frameworks can expose sensitive data or provide entry points for attackers.
Sensitive Data Exposure	Failure to adequately protect sensitive data, such as passwords or personal information, can lead to data breaches and identity theft.
Brute-Force and Credential Stuffing Attacks	Attackers use automated tools to guess usernames and passwords, attempting to gain unauthorized access to user accounts.
File Upload Vulnerabilities	Insecure file upload mechanisms can enable attackers to upload malicious files, leading to remote code execution or unauthorized access to the server.

Common Web Application Threats & Risks

Threat/Risk	Description
Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS)	DoS and DDoS attacks aim to overwhelm web application servers, causing service disruptions and denying legitimate users access.
Server-Side Request Forgery (SSRF)	Attackers use SSRF to make requests from the server to internal resources or external networks, potentially leading to data theft or unauthorized access.
Inadequate Access Controls	Weak access controls may allow unauthorized users to access restricted functionalities or sensitive data.
Using Components with Known Vulnerabilities	Integrating third-party components with known security flaws can introduce weaknesses into the web application.
Broken Access Control	Inadequate access controls can allow unauthorized users to access restricted functionalities or sensitive data.

Common Web Application Threats & Risks

- These security threats are just some of the many risks that web applications face.
- To address these challenges, organizations should adopt a multi-layered security approach, including secure coding practices, regular security testing, user education on security best practices, and the continuous monitoring and updating of web application components and infrastructure.
- Being proactive and staying informed about emerging threats is crucial for maintaining the security and integrity of web applications in an ever-evolving threat landscape.
- From the perspective of a web application penetration tester, it is important to get a firm grasp of the top/common threats faced by web applications.

Web Application Architecture

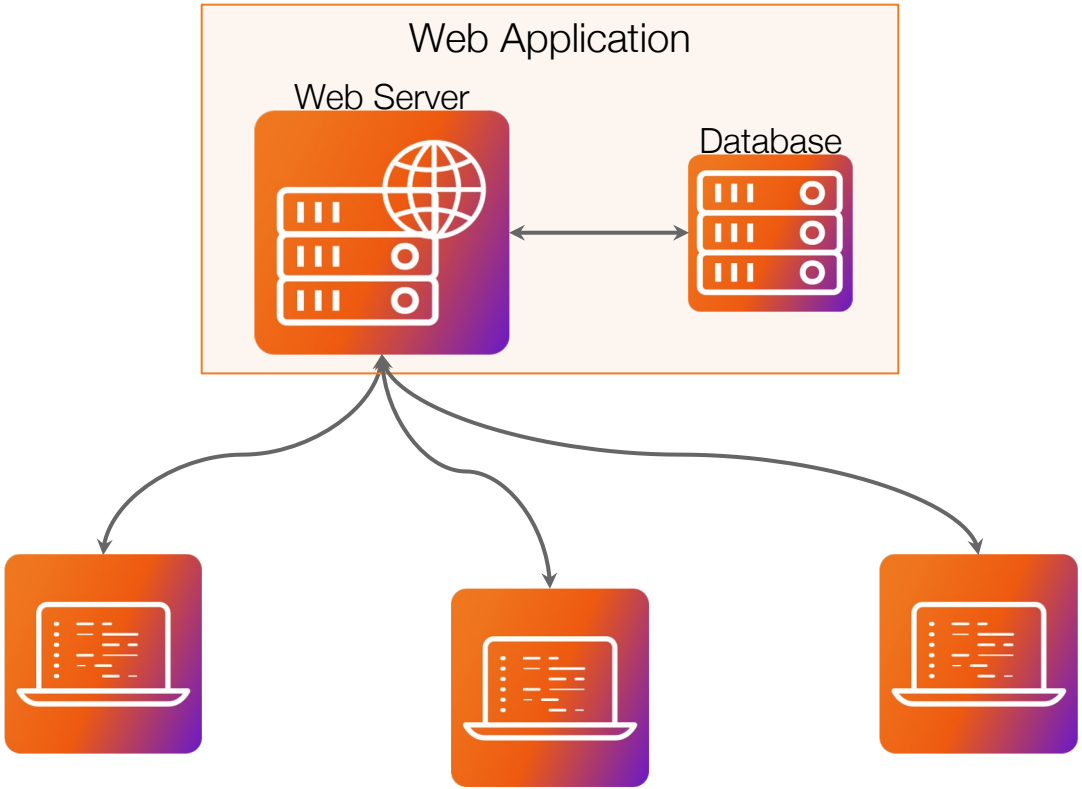
Web Application Architecture

- Web application architecture refers to the structure and organization of components and technologies used to build a web application.
- It defines how different parts of the application interact with each other to deliver its functionality, handle user requests, and manage data.
- A well-designed web application architecture is crucial for ensuring scalability, maintainability, and security.
- Before performing a security assessment on a web application, it is vitally important to know how web applications work with regards to the underlying architecture. This knowledge will provide you with a much better understanding of where and how to identify and exploit potential vulnerabilities or misconfigurations in web applications.

Client-Server Model

- Web applications are typically built on the client-server model. In this architecture, the web application is divided into two main components:
  - **Client:** The client represents the user interface and user interaction with the web application. It is the front-end of the application that users access through their web browsers. The client is responsible for displaying the web pages, handling user input, and sending requests to the server for data or actions.
  - **Server:** The server represents the back-end of the web application. It processes client requests, executes the application's business logic, communicates with databases and other services, and generates responses to be sent back to the client.

Client-Server Model



Web Application Components

Component	Function
User Interface (UI)	The user interface is the visual presentation of the web application seen and interacted with by users. It includes elements such as web pages, forms, menus, buttons, and other interactive components.
Client-Side Technologies	Client-side technologies, such as HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript, are used to create the user interface and handle interactions directly within the user's web browser.
Server-Side Technologies	Server-side technologies, such as programming languages (e.g., PHP, Python, Java, Ruby) and frameworks, are used to implement the application's business logic, process requests from clients, access databases, and generate dynamic content to be sent back to the client.
Databases	Databases are used to store and manage the web application's data. They store user information, content, configurations, and other relevant data required for the application's operation.
Application Logic	The application logic represents the rules and procedures that govern how the web application functions. It includes user authentication, data validation, security checks, and other business rules.
Web Servers	Web servers handle the initial request from clients and serve the client-side components, such as static files (HTML, CSS, JavaScript), to the users.
Application Servers	Application servers execute the server-side code and handle the dynamic processing of client requests. They communicate with databases, perform business logic, and generate dynamic content.

Client-side Processing

- Client-side processing involves executing tasks and computations on the user's device, typically within their web browser.
- The client-side refers to the user's end of the web application, where the web browser and user interface reside.
- Client-side processing has some limitations. It is not suitable for handling sensitive or critical operations, as it can be easily manipulated by users or malicious actors.

Client-side Processing

- Key characteristics of client-side processing:
  - User Interaction: Client-side processing is well-suited for tasks that require immediate user interaction and feedback, as there is no need to send data back and forth to the server.
  - Responsive User Experience: Since processing happens locally, client-side operations can provide a smoother and more responsive user experience.
  - JavaScript: JavaScript is the primary programming language used for client-side processing. It allows developers to manipulate the web page's content, handle user interactions, and perform validations without involving the server.
  - Data Validation: Client-side validation ensures that user input meets specific criteria before it is sent to the server, reducing the need to make unnecessary server requests.

Server-side Processing

- Server-side processing involves executing tasks and computations on the web server, which is the remote computer where the web application is hosted.
- The server-side refers to the backend of the web application, where the business logic and data processing take place.

Server-side Processing

- Key characteristics of server-side processing:
  - Data Processing: Server-side processing is ideal for tasks that involve sensitive data handling, complex computations, and interactions with databases or external services.
  - Security: Since server-side code is executed on a trusted server, it is more secure than client-side code, which can be manipulated by users or intercepted by attackers.
  - Server-side Languages: Programming languages like PHP, Python, Java, Ruby, and others are commonly used for server-side processing.
  - Data Storage: Server-side processing enables secure storage and management of sensitive data in databases or other storage systems.

Communication & Data Flow

- Web applications communicate over the internet using HTTP (Hypertext Transfer Protocol).
- When a user interacts with the web application by clicking on links or submitting forms, the client sends HTTP requests to the server.
- The server processes these requests, interacts with the database if necessary, performs the required actions, and generates an HTTP response.
- The response is then sent back to the client, which renders the content and presents it to the user.

Web Application Technologies

Web Application Technologies

- Understanding web technologies is essential for anyone involved in web development, web application security or web application security testing/web application penetration testing.
- As a web application penetration tester, you will be frequently interacting, assessing and exploiting the underlying technologies that make up a web application as a whole.
- As a result, you need to have a fundamental understanding of what server-side and client-side technologies make up a web application and what their functionalities are and when and why they are deployed.

Client-Side Technologies

- HTML (Hypertext Markup Language) - HTML is the markup language used to structure and define the content of web pages. It provides the foundation for creating the layout and structure of the UI.
- CSS (Cascading Style Sheets) - CSS is used to define the presentation and styling of web pages. It allows developers to control the colors, fonts, layout, and other visual aspects of the UI.
- JavaScript - JavaScript is a scripting language that enables interactivity in web applications. It is used to create dynamic and responsive UI elements, handle user interactions, and perform client-side validations.
- Cookies and Local Storage - Cookies and local storage are client-side mechanisms to store small amounts of data on the user's browser. They are often used for session management and remembering user preferences.

Server-Side Technologies

- Web Server - The web server is responsible for receiving and responding to HTTP requests from clients (web browsers). It hosts the web application's files, processes requests, and sends responses back to clients. (Apache2, Nginx, Microsoft IIS etc)
- Application Server - The application server runs the business logic of the web application. It processes user requests, accesses databases, and performs computations to generate dynamic content that the web server can serve to clients.
- Database Server - The database server stores and manages the web application's data. It stores user information, content, configurations, and other relevant data required for the application's operation. (MySQL, PostgreSQL, MSSQL, Oracle etc)

Server-Side Technologies

- Server-side Scripting Languages - Server-side scripting languages (e.g., PHP, Python, Java, Ruby) are used to handle server-side processing. They interact with databases, perform validations, and generate dynamic content before sending it to the client.

Communication & Data Flow

- Web applications communicate over the internet using HTTP (Hypertext Transfer Protocol).
- When a user interacts with the web application by clicking on links or submitting forms, the client sends HTTP requests to the server.
- The server processes these requests, interacts with the database if necessary, performs the required actions, and generates an HTTP response.
- The response is then sent back to the client, which renders the content and presents it to the user.

Data Interchange

- Data interchange refers to the process of exchanging data between different computer systems or applications, allowing them to communicate and share information.
- It is a fundamental aspect of modern computing, enabling interoperability and data sharing between diverse systems, platforms, and technologies.
- Data interchange involves the conversion of data from one format to another, making it compatible with the receiving system.
- This ensures that data can be interpreted and utilized correctly by the recipient, regardless of the differences in their data structures, programming languages, or operating systems.

Data Interchange Technologies

- APIs (Application Programming Interfaces) - APIs allow different software systems to interact and exchange data. Web applications use APIs to integrate with external services, share data, and provide functionalities to other applications.

Data Interchange Protocols

- JSON (JavaScript Object Notation) - JSON is a lightweight and widely used data interchange format that is easy for both humans and machines to read and write. It is based on JavaScript syntax and primarily used for transmitting data between a server and a web application as an alternative to XML.
- XML (eXtensible Markup Language) - XML is a versatile data interchange format that uses tags to define the structure of the data. It allows users to create their custom tags and define complex hierarchical data structures. XML is commonly used for configuration files, web services, and data exchange between different systems.

Data Interchange Protocols

- REST (Representational State Transfer) - REST is a software architectural style that uses standard HTTP methods (GET, POST, PUT, DELETE) for data interchange. It is widely used for creating web APIs that allow applications to interact and exchange data over the internet.
- SOAP (Simple Object Access Protocol) - SOAP is a protocol for exchanging structured information in the implementation of web services. It uses XML as the data interchange format and provides a standardized method for communication between different systems.

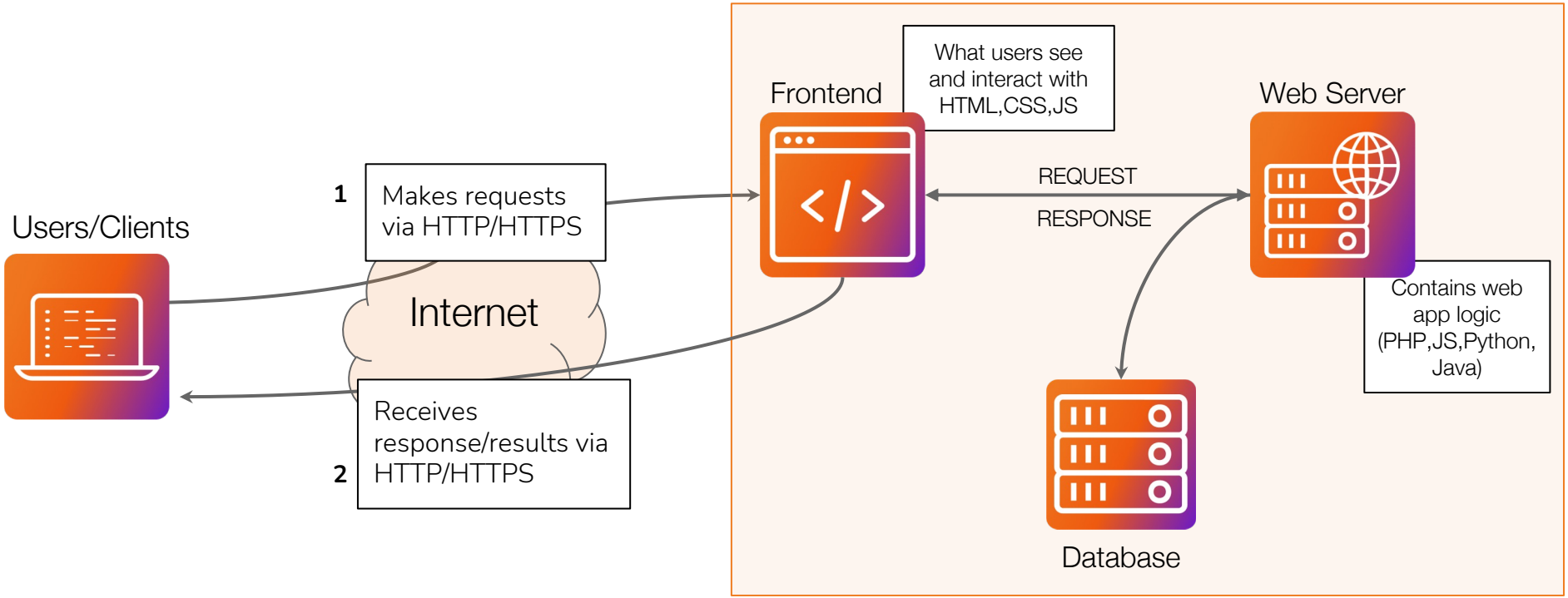
Security Technologies

- Authentication and Authorization Mechanisms - Authentication verifies the identity of users, while authorization controls access to different parts of the web application based on user roles and permissions.
- Encryption (SSL/TLS) - SSL (Secure Socket Layer) or TLS (Transport Layer Security) is used to encrypt data transmitted between the client and server, ensuring secure communication and data protection.

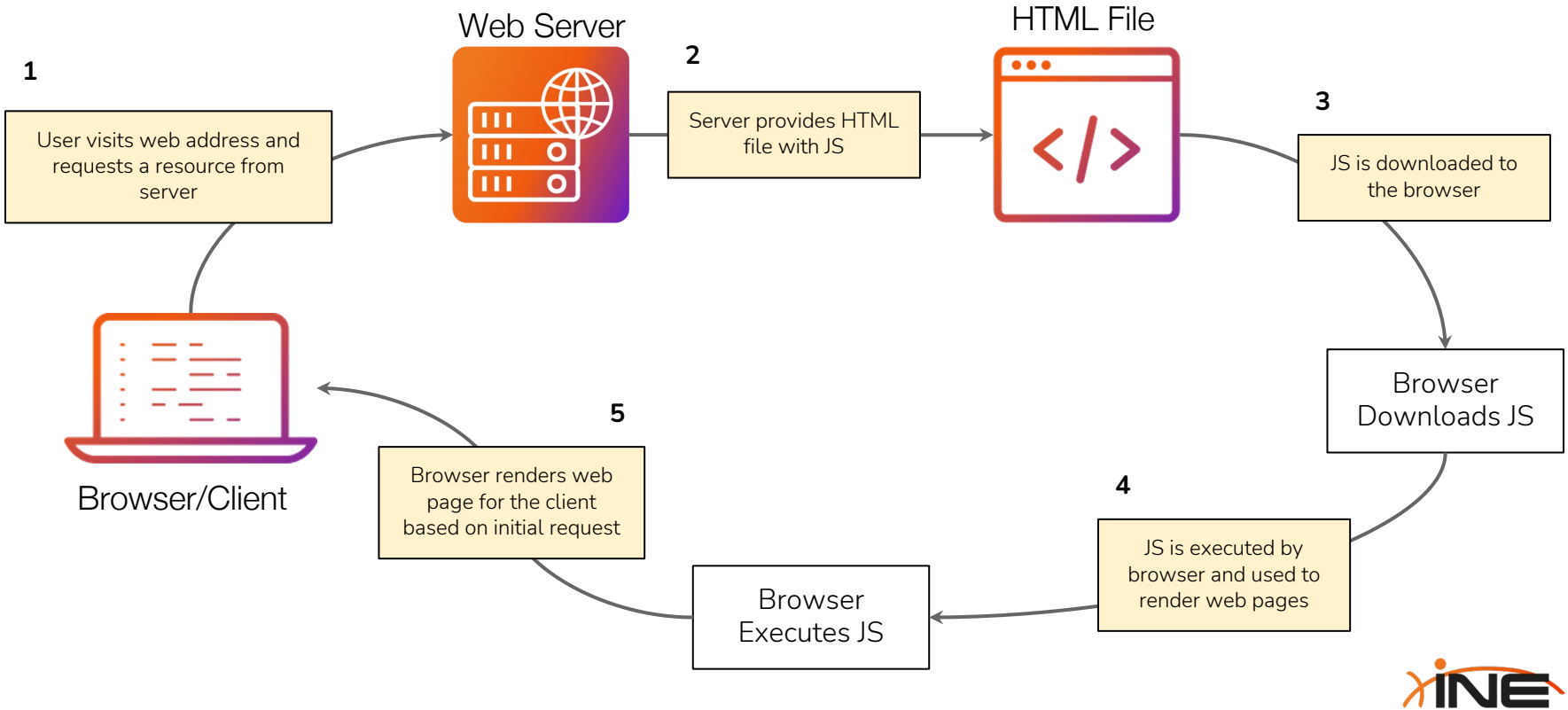
External Technologies

- Content Delivery Networks (CDNs) - CDNs are used to distribute static content (e.g., images, CSS files, JavaScript libraries) to multiple servers located worldwide, improving the web application's performance and reliability.
- Third-Party Libraries and Frameworks - Web applications often leverage third-party libraries and frameworks to speed up development and access advanced features.

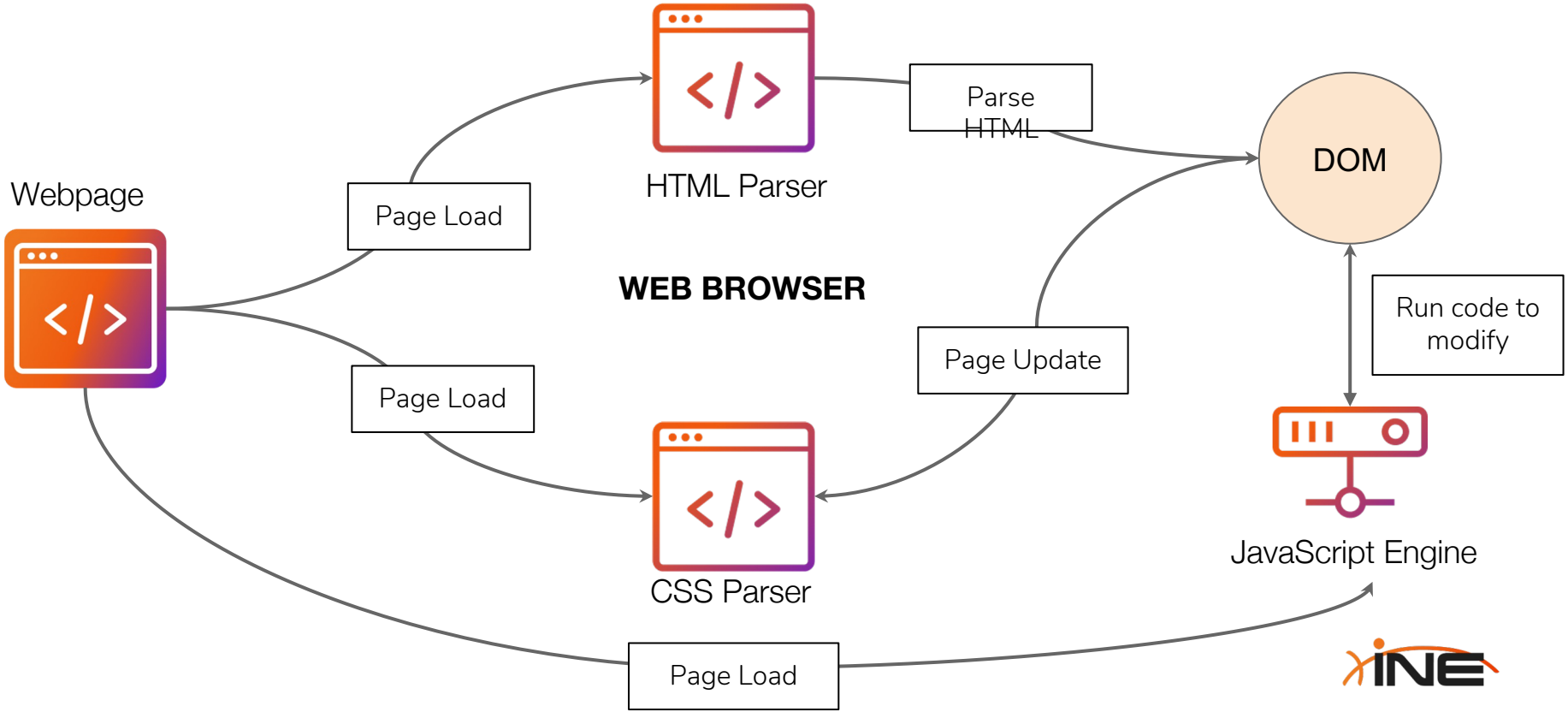
Web Application Architecture



How Web Pages Are Rendered



How Web Browsers Parse Responses

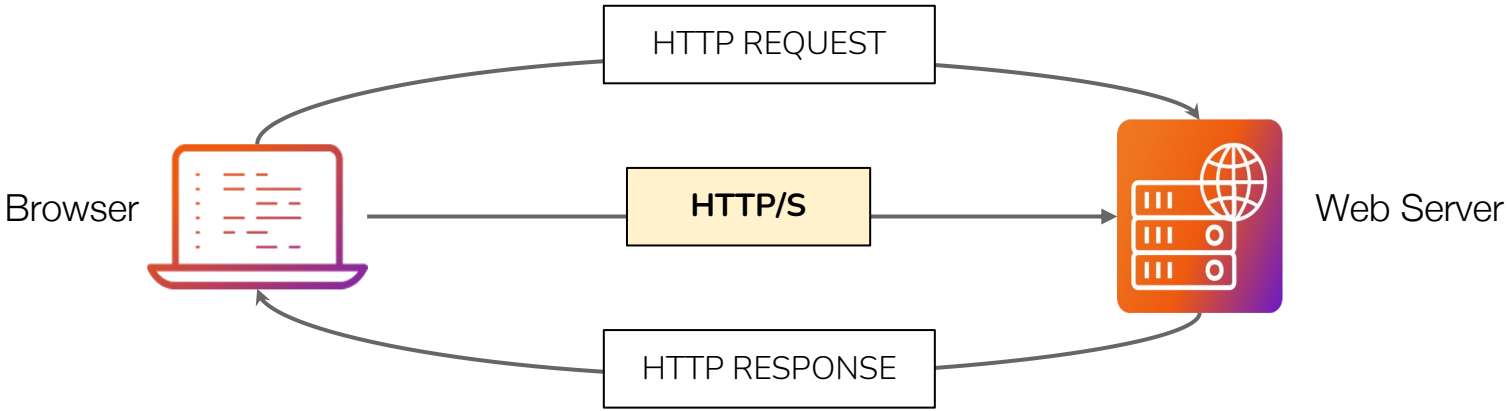


Introduction To HTTP

HTTP Protocol

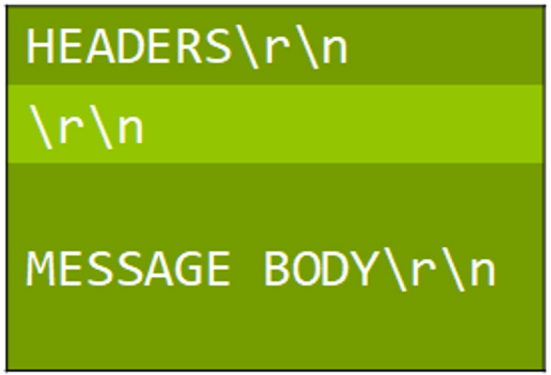
- HTTP (Hypertext Transfer Protocol) is a stateless application layer protocol used for the transmission of resources like web application data and runs on top of TCP.
- It was specifically designed for communication between web browsers and web servers.
- HTTP utilizes the typical client-server architecture for communication, whereby the browser is the client, and the web server is the server.
- Resources are uniquely identified with a URL/URI.
- HTTP has 2 versions; HTTP 1.0 & HTTP 1.1.
    - HTTP 1.1 is the most widely used version of HTTP and has several advantages over HTTP 1.0 such as the ability to re-use the same connection and can request for multiple URI's/Resources.

HTTP Protocol Basics

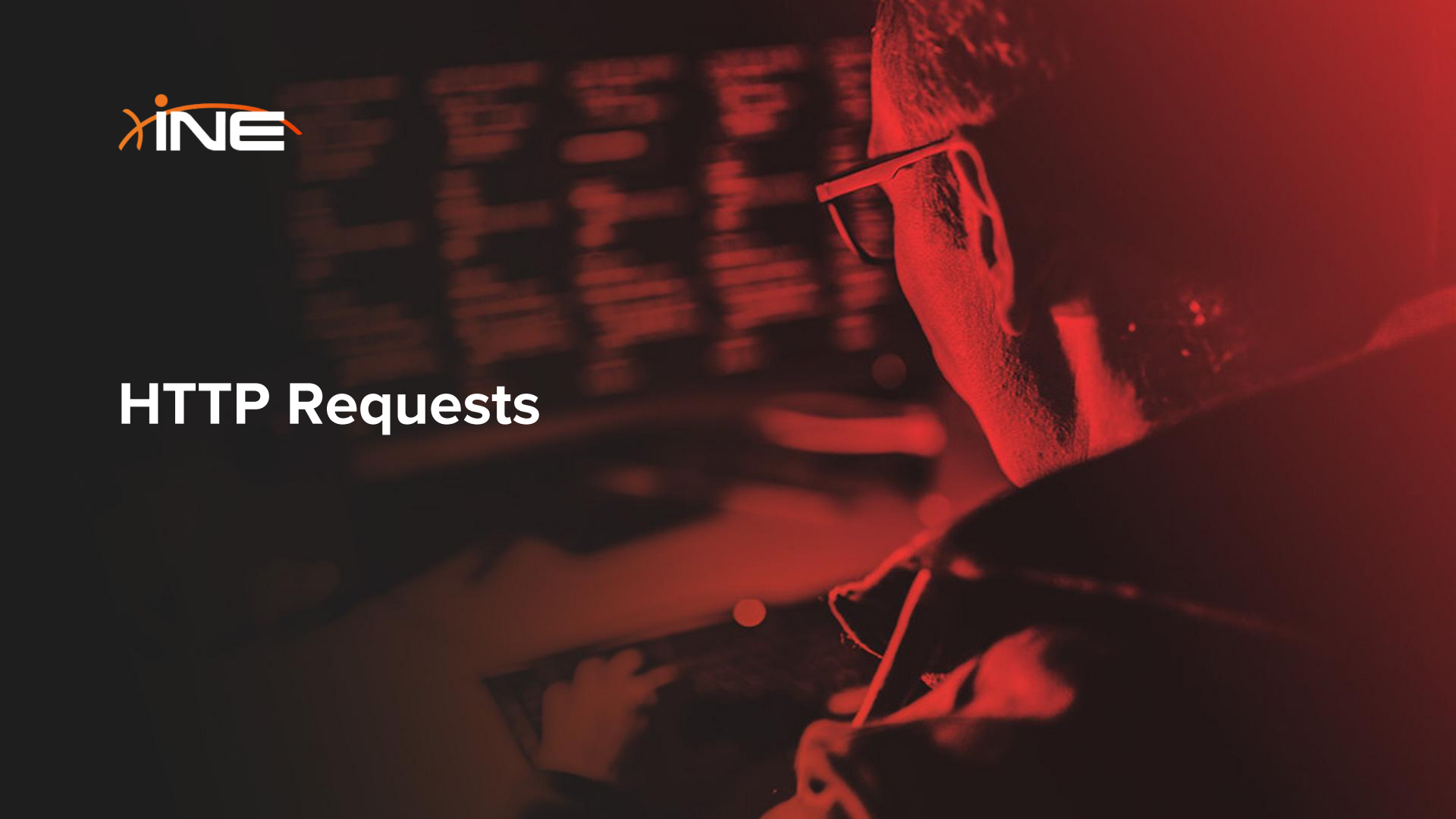


HTTP Protocol Basics

- During HTTP communication, the client and the server exchange messages, typically classified as HTTP requests and responses.
- The client sends requests to the server and gets back responses.



- **\r (Carriage Return): moves the cursor to the beginning of the line**
- **\n (Line Feed): moves the cursor down to the next line**
- **\r\n: is the same as hitting the enter key on your keyboard**



HTTP Requests

HTTP Request Components

Request Line

- The request line is the first line of an HTTP request and contains the following three components:
  - HTTP method (e.g., GET, POST, PUT, DELETE, etc.): Indicates the type of request being made.
  - URL (Uniform Resource Locator): The address of the resource the client wants to access.
  - HTTP version: The version of the HTTP protocol being used (e.g., HTTP/1.1).

HTTP Request Components

Request Headers

- Headers provide additional information about the request. Common headers include:
  - User-Agent: Information about the client making the request (e.g., browser type).
  - Host: The hostname of the server.
  - Accept: The media types the client can handle in the response (e.g., HTML, JSON).
  - Authorization: Credentials for authentication, if required.
  - Cookie: Information stored on the client-side and sent back to the server with each request.

HTTP Request Components

Request Body (Optional)

- Some HTTP methods (like POST or PUT) include a request body where data is sent to the server, typically in JSON or form data format.

HTTP Request Example

- Let's examine an HTTP request in detail. The following is the data contained in a request that we send when we navigate to www.google.com with a web browser.


```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0)
Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

HTTP Request Headers

- An HTTP request to www.google.com is initiated. What you see here are the headers (HTTP Request Headers) for this request.
- Note that a connection to www.google.com on port 80 is initiated before sending HTTP commands to the webserver.

HTTP Request Methods

- HTTP request methods (HTTP Verbs) provide a standardized way for clients and servers to communicate and interact with resources on the web. The choice of the appropriate method depends on the type of operation that needs to be performed on the resource.
- GET is the default request method used when you make a request to a web application, in this case we are trying to connect to www.google.com.

HTTP Request Methods Explained

Method	Function
GET	The GET method is used to retrieve data from the server. It requests the resource specified in the URL and does not modify the server's state. It is a safe and idempotent method, meaning that making the same GET request multiple times should not have any side effects.
POST	The POST method is used to submit data to be processed by the server. It typically includes data in the request body, and the server may perform actions based on that data. POST requests can cause changes to the server's state, and they are not idempotent.
PUT	The PUT method is used to update or create a resource on the server at the specified URL. It replaces the entire resource with the new representation provided in the request body. If the resource does not exist, PUT can create it.
DELETE	The DELETE method is used to remove the resource specified by the URL from the server. After a successful DELETE request, the resource will no longer be available at that URL.
PATCH	The PATCH method is used to apply partial modifications to a resource. It is similar to the PUT method but only updates specific parts of the resource rather than replacing the entire resource.
HEAD	The HEAD method is similar to the GET method but only retrieves the response headers and not the response body. It is often used to check the headers for things like resource existence or modification dates.
OPTIONS	The OPTIONS method is used to retrieve information about the communication options available for the target resource. It allows clients to determine the supported methods and headers for a particular resource.

HTTP Request URL/Path

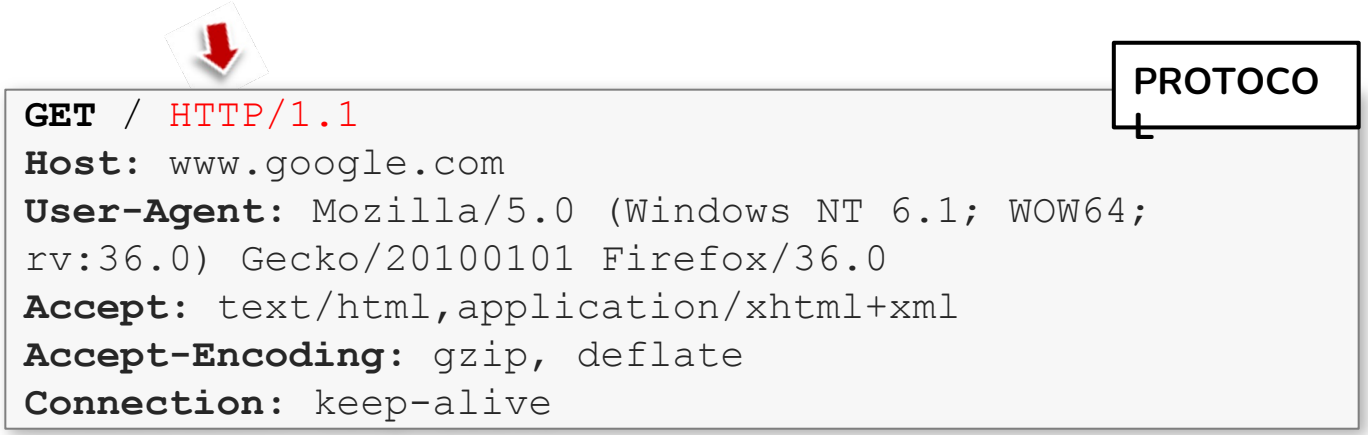
- The address of the resource/URI the client wants to access.
- The home page of a website is always "/". Other pages can be requested, of course, for example: /downloads/index.php.
- Your request always refers to the root folder to specify the requested file (hence the leading "/").

URL/PATH

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
rv:36.0) Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

HTTP Request Protocol

- This is the HTTP protocol version that your browser wants to communicate with (HTTP 1.0/HTTP 1.1).



HTTP Request Host Header

- This is the beginning of HTTP Request Headers. HTTP Headers have the following structure: `Header-name:Header-Value`.
- The Host header allows a web server to host multiple websites at a single IP address. Our browser is specifying in the Host header which website you are interested in.

HOST HEADER

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
rv:36.0) Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

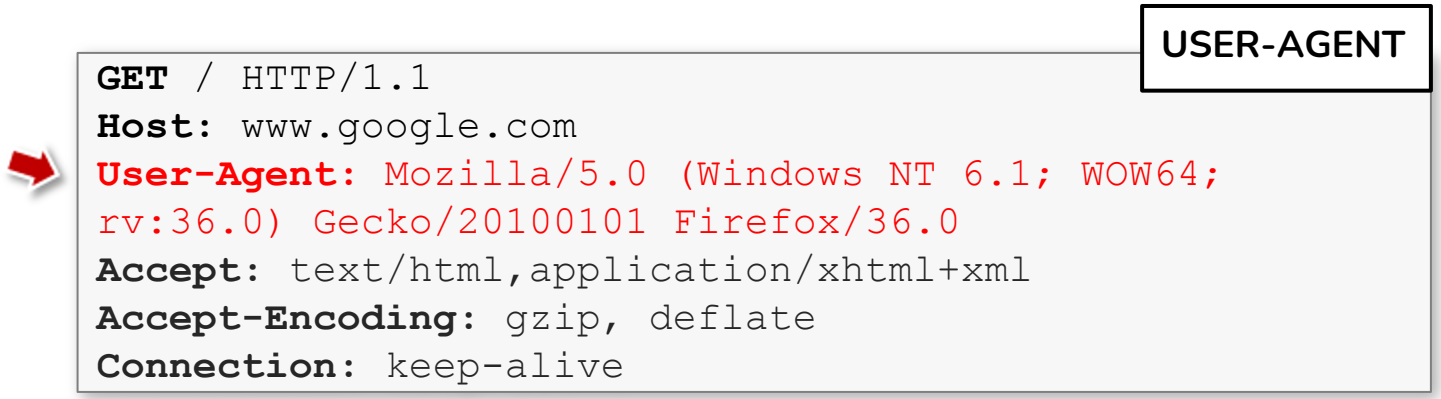
HTTP Request Host Header

- After each request header, you will find its corresponding value. In this case we want to access the Host www.google.com.

- Note: Host value + Path combine to create the full URL you are requesting: the home page of www.google.com/

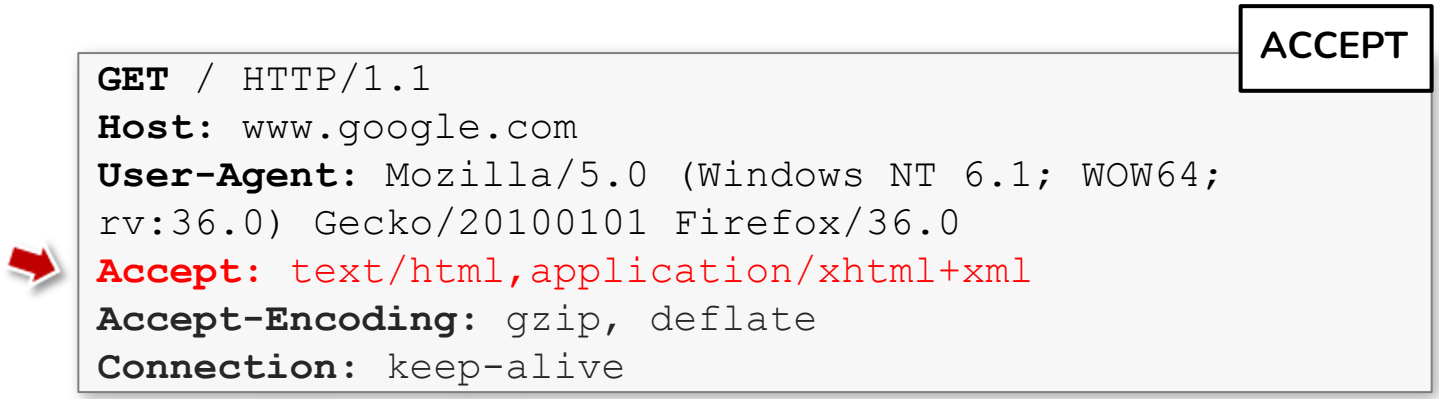
HTTP Request User-Agent Header

- The User-Agent is used to specify and send your browser, browser version, operating system and language to the remote web server.
- All web browsers have their own user-agent identification string. This is how most web sites recognize the type of browser in use.



HTTP Request Accept Header

- The Accept header is used by your browser to specify which document/file types are expected to be returned from the web server as a result of this request.



HTTP Request Accept-Encoding Header

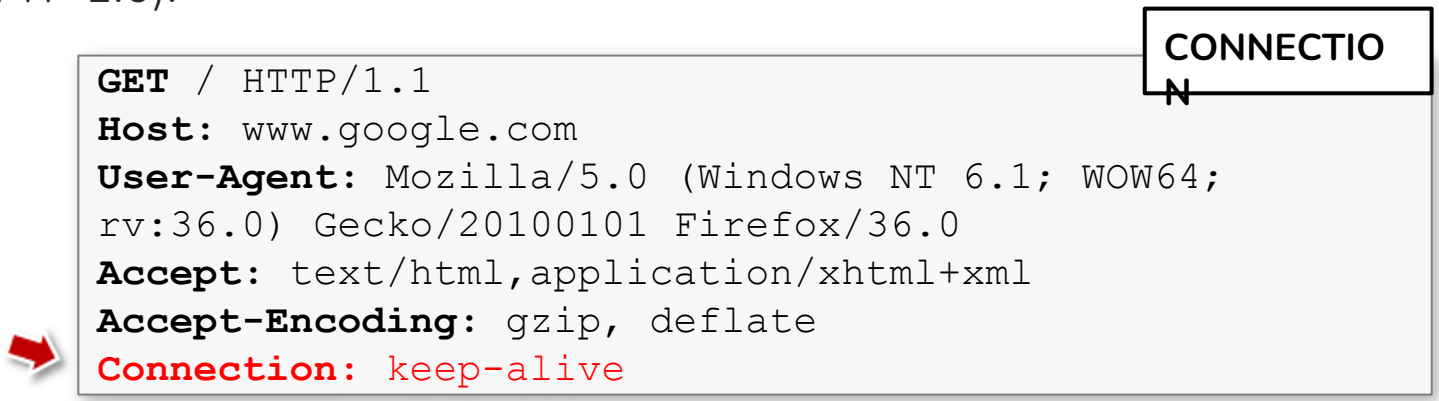
- The Accept-Encoding header is similar to Accept, and is used to restrict the content encoding that is acceptable in the response.
- Content encoding is primarily used to allow a document to be compressed or transformed without losing the original format and without the loss of information.

ACCEPT-ENCODING

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
rv:36.0) Gecko/20100101 Firefox/36.0
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

HTTP Request Connection Header

- When using HTTP 1.1, you can maintain/re-use the connection to the remote web server for an unspecified amount of time using the value "keep-alive".
- This indicates that all requests to the web server will continue to be sent through this connection without initiating a new connection every time (as in HTTP 1.0).



HTTP Responses

HTTP Response Components

Response Headers

- Similar to request headers, response headers provide additional information about the response. Common headers include:
  - Content-Type: The media type of the response content (e.g., text/html, application/json).
  - Content-Length: The size of the response body in bytes.
  - Set-Cookie: Used to set cookies on the client-side for subsequent requests.
  - Cache-Control: Directives for caching behavior.

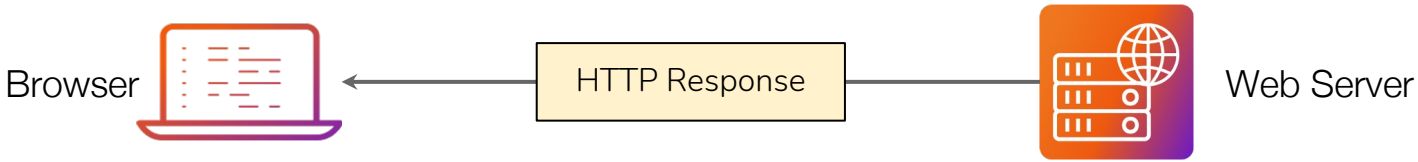
HTTP Response Components

Response Body (Optional)

- The response body contains the actual content of the response. For example, in the case of an HTML page, the response body will contain the HTML markup.

HTTP Request Response Example

- Now that we know what an HTTP request comprises of, let us inspect HTTP responses.



- In response to the HTTP Request, the web server will respond with the requested resource, preceded by a bunch of new HTTP response headers.
- These new response headers from the web server will be used by your web browser to interpret the content contained in the Response content/body of the response.

HTTP Response Example

- The code snippet below is an example of a typical web server response.
- Note: The body of the response/page content has been omitted as it is not relevant at this point in time.
- Let us inspect some of these HTTP response headers in greater detail.

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

HTTP Response Status-Line

- The first line of an HTTP Response is the Status-Line, consisting of the protocol version (HTTP 1.1) followed by the HTTP status code (200) and its relative textual meaning (OK).

STATUS-LINE

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

Common HTTP Status Codes

Status Code	Meaning
200 OK	The request was successful, and the server has returned the requested data.
301 Moved Permanently	The requested resource has been permanently moved to a new URL, and the client should use the new URL for all future requests.
302 Found	The requested resource is temporarily located at a different URL. This code is commonly used for temporary redirections, but it's often better to use 303 or 307 instead.
400 Bad Request	The server cannot process the request due to a client error (e.g., malformed request syntax).
401 Unauthorized	Authentication is required, and the client must provide valid credentials to access the requested resource.
403 Forbidden	The server understood the request, but the client does not have permission to access the requested resource.
404 Not Found	The requested resource could not be found on the server.
500 Internal Server Error	The server encountered an error while processing the request, and the specific cause is not provided.

HTTP Response Date Header

- The "Date" header in an HTTP response is used to indicate the date and time when the response was generated by the server.
- It helps clients and intermediaries to understand the freshness of the response and to synchronize the time between the server and the client.

DATE

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

HTTP Response Cache-Control Header

- The Cache headers allow the Browser and the Server to agree about caching rules. It allows web servers to instruct clients on how long they can cache the response and under what conditions they should revalidate it with the server.
- This helps in optimizing the performance and efficiency of web applications by reducing unnecessary network requests.

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

CACHE-CONTROL

Cache-Control Directives

Directive	Meaning
Public	Indicates that the response can be cached by any intermediary caches (such as proxy servers) and shared across different users.
Private	Specifies that the response is intended for a specific user and should not be cached by intermediate caches.
no-cache	Instructs the client to revalidate the response with the server before using the cached version. It does not prevent caching but requires revalidation.
no-store	Directs the client and intermediate caches not to store any version of the response. It ensures that the response is not cached in any form.
max-age=<SECONDS>	Specifies the maximum amount of time in seconds that the response can be cached by the client. After this period, the client should revalidate with the server.

HTTP Response Content-Type Header

- The "Content-Type" header in an HTTP response is used to indicate the media type of the response content.
- It tells the client what type of data the server is sending so that the client can handle it appropriately.

CONTENT-TYPE

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

HTTP Response Content-Encoding Header

- The "Content-Encoding" header in an HTTP response is used to specify the compression encoding applied to the response content.
- It tells the client how the server has encoded the response data, allowing the client to decode and decompress the data correctly.

CONTENT-ENCODING

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

HTTP Response Server Header

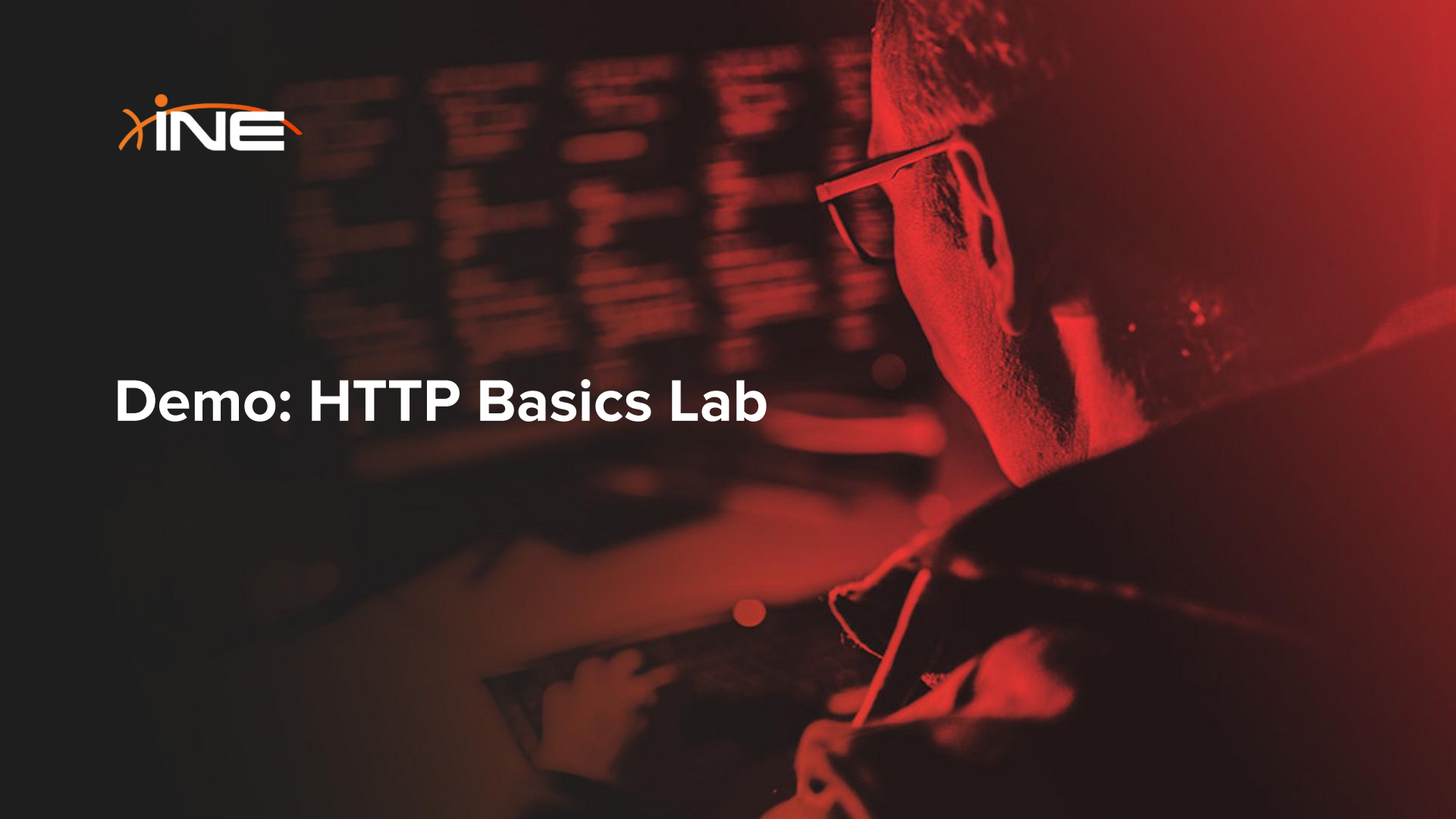
- The Server header displays the Web Server banner, for example, Apache, Nginx, IIS etc.
- Google uses a custom web server banner: gws (Google Web Server).

SERVER

```
HTTP/1.1 200 OK
Date: Fri, 13 Mar 2015 11:26:05 GMT
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 258

<PAGE CONTENT>
```

HTTP Basics Lab



Demo: HTTP Basics Lab

HTTPS

HTTPS

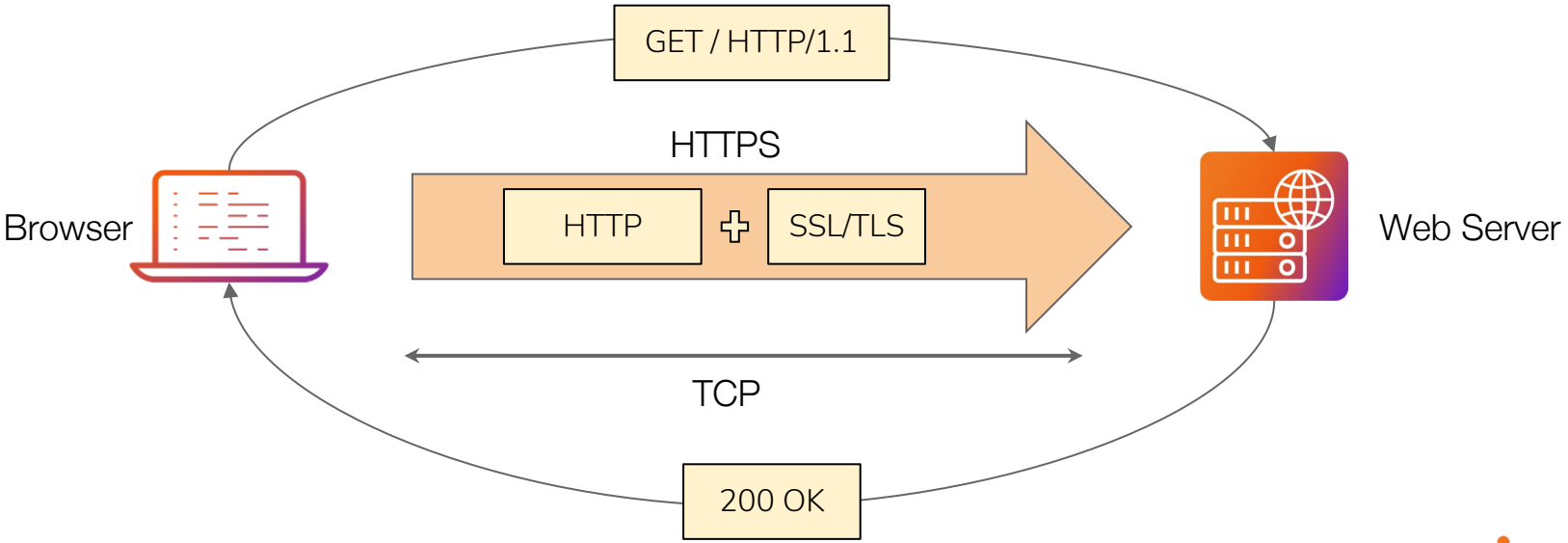
- Now that you have an understanding of how HTTP works, let us explore how it is secured/protected.
- By default, HTTP requests are sent in clear-text and can be easily intercepted or mangled by an attacker on the way to its destination.
- Moreover, HTTP does not provide strong authentication between the two communicating parties.
- HTTPS (Hypertext Transfer Protocol Secure) is a secure version of the HTTP protocol, which is used to transmit data between a user's web browser and a website or web application.
- HTTPS provides an added layer of security by encrypting the data transmitted over the internet, making it more secure and protecting it from unauthorized access and interception.

HTTPS

- HTTPS is also commonly referred to as HTTP Secure.
- HTTPS is the preferred way to use and configure HTTP and involves running HTTP over SSL/TLS.
- SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols used to provide secure communication over a computer network, most commonly the internet.
- They are essential for establishing a secure and encrypted connection between a user's web browser or application and a web server.

HTTPS

- This layering technique provides confidentiality, integrity protection and authentication to the HTTP protocol.



HTTPS Advantages

- Encryption of Data in Transit - One of the primary benefits of HTTPS is data encryption during transmission. When data is sent over an HTTPS connection, it is encrypted using strong cryptographic algorithms. This ensures that even if an attacker intercepts the data while it's in transit, they cannot decipher or read its contents.
- Protection Against Eavesdropping - HTTPS prevents unauthorized parties from eavesdropping on the data exchanged between the user's browser and the web server. This is particularly crucial when users input sensitive information, such as login credentials, credit card numbers, or personal details.

HTTPS Security Considerations

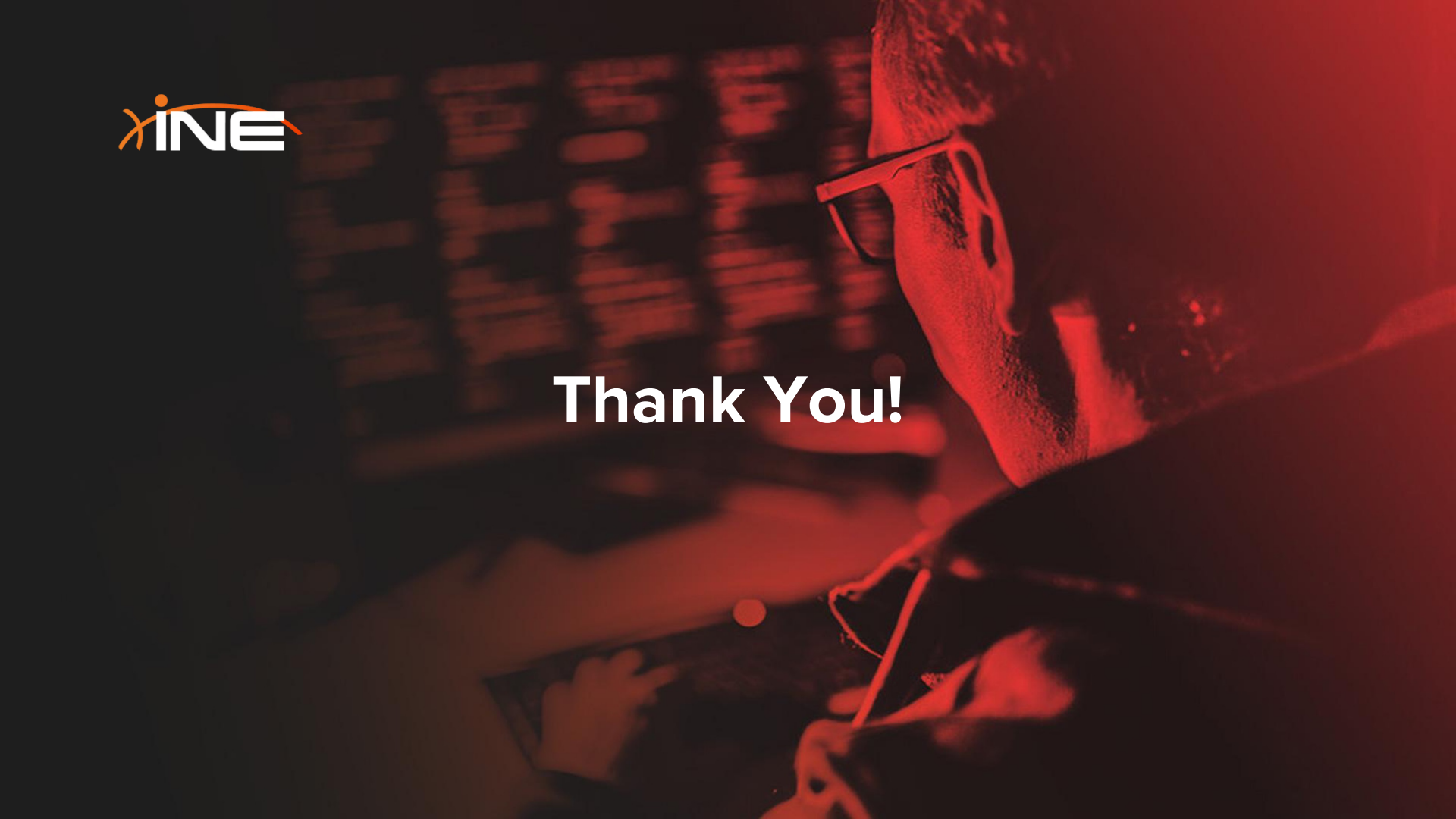
- HTTPS does not protect against web application flaws! Various web application attacks will still work regardless of the use of SSL/TLS. (Attacks like XSS and SQLi will still work)
- The added encryption layer only protects data exchanged between the client and the server and does stop attacks against the web application itself.

Introduction To The Web & The HTTP Protocol

Course Conclusion

Learning Objectives:

- You will have a solid understanding of what web applications are, how they work and how they are deployed.
- You will have a solid understanding of importance of securing web applications.
- You will have an understanding of the various components that make up a web application's architecture.
- You will have an understanding of common web application security best practices and why they are implemented.
- You will have a good understanding of what Web Application Security Testing is, what it entails and why it is performed.
- You will be able to differentiate between the various types of web application security tests and will have an understanding of the differences between a web application security test and a web app pentest.
- You will have a functional and practical understanding of how HTTP/S works and will be able to analyze HTTP requests and responses.
- You will have an understanding of the various HTTP request and response headers, methods and status codes.


Thank You!

EXPERTS AT MAKING YOU AN EXPERT