

# WEB APP SECURITY



```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => {
                  console.log(value)
                }}
              </ProductConsumer>
            </div>
          </div>
        </div>
      <React.Fragment>
```

## Course Outcomes

**CO1** Apply client-side web development to design interactive front-end web user interfaces.

**CO2** Use server-side web application concepts to develop back-end web server application

**CO3** Identify and mitigate various client-side web application security vulnerabilities

**CO4** Identify and mitigate various server-side web application security vulnerabilities

# Syllabus

- Web application development – Introduction - Architecture – Client-side technologies and frameworks – HTML – CSS – Javascript - Ajax/Fetch - Data interchange formats – XML, JSON. Server-side scripting and technologies - development – technologies - Handling client requests – Database connectivity – Sessions – Cookies.
- Web application vulnerabilities – Client-side Vulnerabilities - Cross Site Scripting (XSS) - Cross Site Request Forgery (CSRF) - Cross-origin resource sharing (CORS) - Clickjacking. Server-side Vulnerabilities - SQL injection - OS command injection - Directory traversal - Authentication - Server-side request forgery (SSRF)
- **Textbook(s)**

# Textbook and Reference books

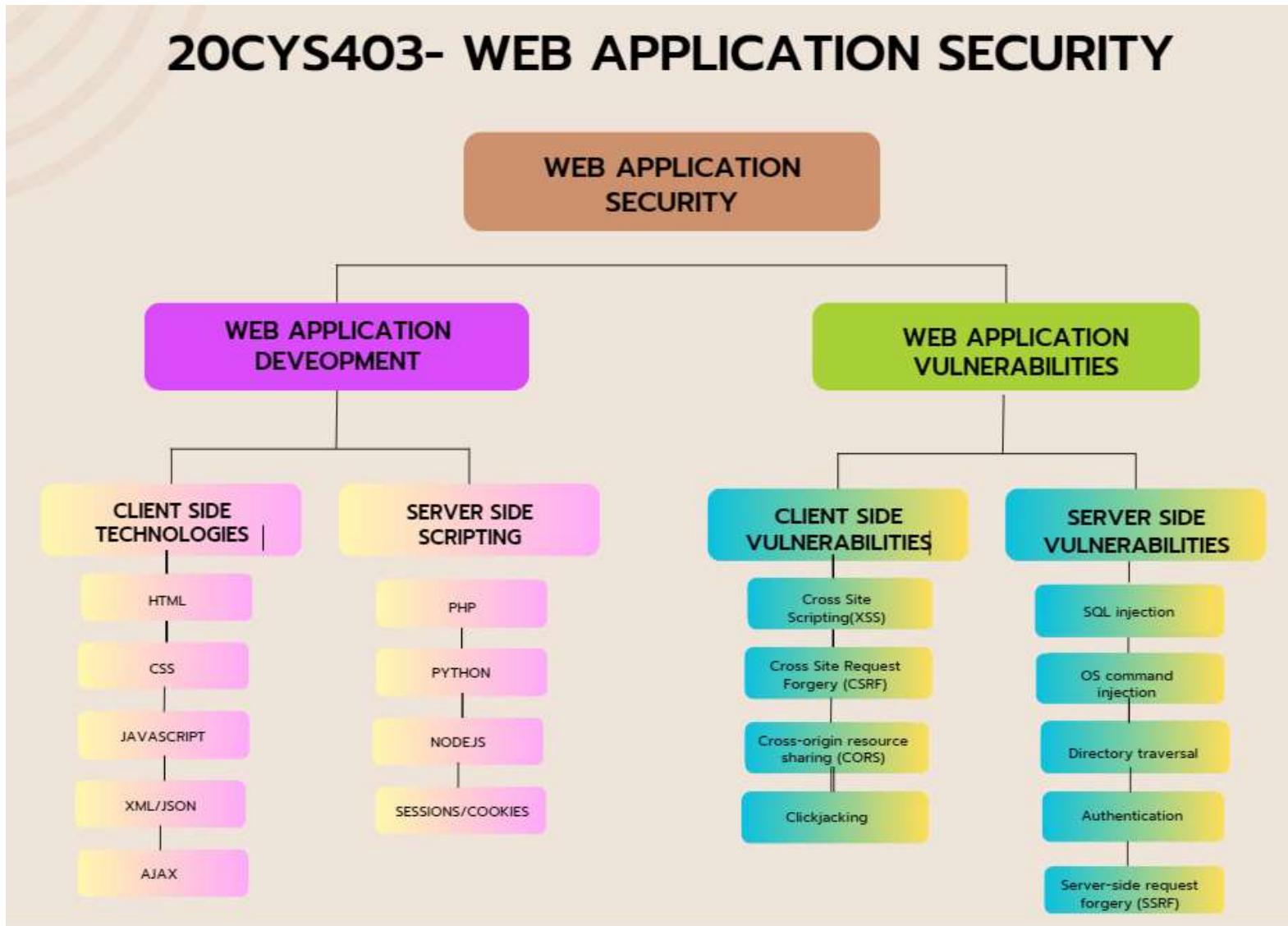
- **Textbook(s)**

1. Robin Nixon, Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites, Fifth Edition, O'Reilly Media, Inc.; 2018.
2. Dafydd Stuttard, and Marcus Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition, John Wiley & Sons; 2011

- **Reference(s)**

1. John Paul Mueller, Security for Web Developers: Using Javascript, Html, and CSS, O'Reilly, 2015
2. Andrew Hoffman. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Shroff Publishers & Distributors Pvt. Ltd, 2020.
3. Richa Gupta, Hands-on Penetration Testing for Web Applications, BPB Publications (29 March 2021)
4. Ivan Ristic, Bulletproof TLS and PKI, Second Edition: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Feisty Duck Ltd; 2nd edition, 2022
5. <https://web.stanford.edu/class/cs253/>
6. <https://nptel.ac.in/courses/128106006>
7. <https://www.rapid7.com/fundamentals/web-application-security/>
8. <https://owasp.org/>
9. <https://www.udemy.com/course/web-application-security/>

# Concept Map



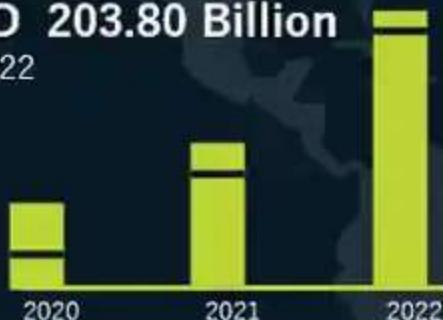


# Introduction to Web Application Security

Market is expected to  
REGISTER a CAGR of **12.2%**



The market was valued at  
**USD 203.80 Billion**  
in 2022



**31.0%**

of the global  
market revenue  
was accounted  
for by North  
America in 2022



Based on component, the  
hardware segment is expected  
to register a CAGR of  
**12.8%**

The market is **FAIRLY FRAGMENTED** with  
many players accounting for majority of  
market revenue share



One of the **KEY DRIVERS** for  
market growth is rapid adoption  
of Cybersecurity Mesh  
Architecture (CSMA) by  
organizations



CYBERSECURITY  
MARKET 2019–2032

10,000+ reports  
covering niche topics



**EMERGEN**  
RESEARCH

<https://blog.cloudflare.com/application-security-2023/>

# What are common web application security risks?

Web applications may face a number of attack types depending on the attacker's goals, the nature of the targeted organization's work, and the application's particular security gaps. Common attack types include:

1. **Zero-day vulnerabilities:** These are vulnerabilities unknown to an application's makers, and which thus do not have a fix available. We now see more than 20,000 every year. Attacks look to exploit these vulnerabilities quickly, and often follow up by seeking to evade protections put in place by security vendors.
2. Cross site scripting (XSS): XSS is a vulnerability that allows an attacker to inject client-side scripts into a webpage in order to access important information directly, impersonate the user, or trick the user into revealing important information.
3. SQL injection (SQi): SQi is a method by which an attacker exploits vulnerabilities in the way a database executes search queries. Attackers use SQi to gain access to unauthorized information, modify or create new user permissions, or otherwise manipulate or destroy sensitive data.

# What are common web application security risks?

4. **Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks:** Through a variety of vectors, attackers are able to overload a targeted server or its surrounding infrastructure with different types of attack traffic. When a server is no longer able to effectively process incoming requests, it begins to behave sluggishly and eventually deny service to incoming requests from legitimate users.
5. **Memory corruption:** Memory corruption occurs when a location in memory is unintentionally modified, resulting in the potential for unexpected behavior in the software. Bad actors will attempt to sniff out and exploit memory corruption through exploits such as code injections or buffer overflow attacks.
6. **Buffer overflow:** Buffer overflow is an anomaly that occurs when software writing data to a defined space in memory known as a buffer. Overflowing the buffer's capacity results in adjacent memory locations being overwritten with data. This behavior can be exploited to inject malicious code into memory, potentially creating a vulnerability in the targeted machine.

# What are common web application security risks?

7. **Cross-site request forgery (CSRF)**: Cross site request forgery involves tricking a victim into making a request that utilizes their authentication or authorization. By leveraging the account privileges of a user, an attacker is able to send a request masquerading as the user. Once a user's account has been compromised, the attacker can exfiltrate, destroy or modify important information. Highly privileged accounts such as administrators or executives are commonly targeted.
8. **Credential stuffing**: Attackers may use bots to quickly input large numbers of stolen username and password combinations into a web application's login portal. If this practice gives the attacker access to a real user's account, they may steal the user's data or make fraudulent purchases in the user's name.
9. **Page scraping**: Attackers may also use bots to steal content from webpages on a large scale. They may use this content to gain a pricing advantage over a competitor, imitate the page owner for malicious purposes, or other reasons.

# What are common web application security risks?

10. **API abuse:** APIs, or Application Programming Interfaces, are software that allow two applications to communicate with each other. Like any type of software, they may have vulnerabilities that allow attackers to send malicious code into one of the applications or intercept sensitive data as it moves from one application to another. This is an increasingly common attack type as API use increases. The OWASP API Top ten list succinctly summarized key API security risks organizations face today.
11. **Shadow APIs:** Development teams work quickly to meet business objectives, frequently building and publishing APIs without informing security teams. These unknown APIs may expose sensitive company data, operating in the “shadows” as security teams tasked with protecting APIs are unaware of their existence.
12. **Third-party code abuse:** Many modern web applications use a variety of third-party tools — for example, an ecommerce site using a third-party payment processing tool. If attackers find a vulnerability in one of these tools, they may be able to compromise the tool, and steal the data it processes, prevent it from functioning, or use it to inject malicious code elsewhere in the application. Magecart attacks, which skim credit card data from payment processors, are an example of this attack type. These attacks are also considered to be browser supply chain attacks.
13. **Attack surface misconfigurations:** An organization’s attack surface is its entire IT footprint that could be susceptible to cyberattacks: servers, devices, SaaS, and cloud assets that are accessible from the Internet. This attack surface can remain vulnerable to attack due to certain elements being overlooked or misconfigured.

# What are important web application security strategies?

- **DDoS mitigation:** DDoS mitigation services sit between a server and the public Internet, using specialized filtration and extremely high bandwidth capacity to prevent surges of malicious traffic from overwhelming the server. These services are important because many modern DDoS attacks deliver enough malicious traffic to overwhelm even the most resilient servers.
- **Web Application Firewall (WAF):** Which filter out traffic known or suspected to be taking advantage of web application vulnerabilities. WAFs are important because new vulnerabilities emerge too quickly and quietly for nearly all organizations to catch on their own.
- **API gateways:** Which help identify overlooked ‘shadow APIs,’ and block traffic known or suspected to target API vulnerabilities. They also help manage and monitor API traffic. ([Learn more about API security.](#))
- **DNSSEC:** A protocol which guarantees a web application’s DNS traffic is safely routed to the correct servers, so users are not intercepted by an on-path attacker.

- **Encryption certificate management:** In which a third party manages key elements of the SSL/TLS encryption process, such as generating private keys, renewing certificates, and revoking certificates due to vulnerabilities. This removes the risk of those elements going overlooked and exposing private traffic.
- **Bot management:** Which uses machine learning and other specialized detection methods to distinguish automated traffic from human users, and prevent the former from accessing a web application.
- **Client-side security:** Which checks for new third-party JavaScript dependencies and third-party code changes, helping organizations catch malicious activity sooner.
- **Attack surface management:** actionable attack surface management tools should provide a single place to map your attack surface, identify potential security risks, and mitigate risks with a few clicks.

# What application security best practices should organizations expect from their vendors?

- Web developers can design and build applications in ways that prevent attackers from accessing private data, fraudulently accessing user accounts, and performing other malicious actions. The [OWASP Top 10 list](#) captures the most common application security risks developers should be aware of. Practices to prevent these risks include:
- **Requiring input validation:** Blocking improperly formatted data from passing through the application's workflows helps prevent malicious code from entering the application via an injection attack.
- **Using up-to-date encryption:** Storing user data in an encrypted fashion, along with using [HTTPS](#) to encrypt transmission of inbound and outbound traffic, helps prevent attackers from stealing data.
- **Offering strong authentication and authorization:** Building in and enforcing controls for strong passwords, offering multi-factor authentication options including hard keys, offering access control options, and other practices make it harder for attackers to fraudulently access user accounts and move laterally within your application.
- **Keeping track of APIs:** Tools exist to identify overlooked 'shadow APIs' that could constitute an attack surface, but API security becomes easier when APIs never get overlooked in the first place.
- **Documenting code changes:** Which helps security and developer teams fix newly introduced vulnerabilities sooner.
- [DDoS mitigation](#), a [Web Application Firewall](#), [API protection](#), [DNSSEC](#), [Managed SSL/TLS](#), [Bot management](#), [client-side security](#), and more.



# Topic 1:

## Web application development – Introduction & Architecture

# WEB APPLICATION DEVELOPMENT PROCESS

1. Find a Genuine App Idea



9. Deploy Your App



2. Market Research



3. Define  
Functionality



8. Host Your  
Application  
on the Web



7. Choose Your  
Technology



6. Time to Start  
Validating



5. Wireframes &  
Prototypes



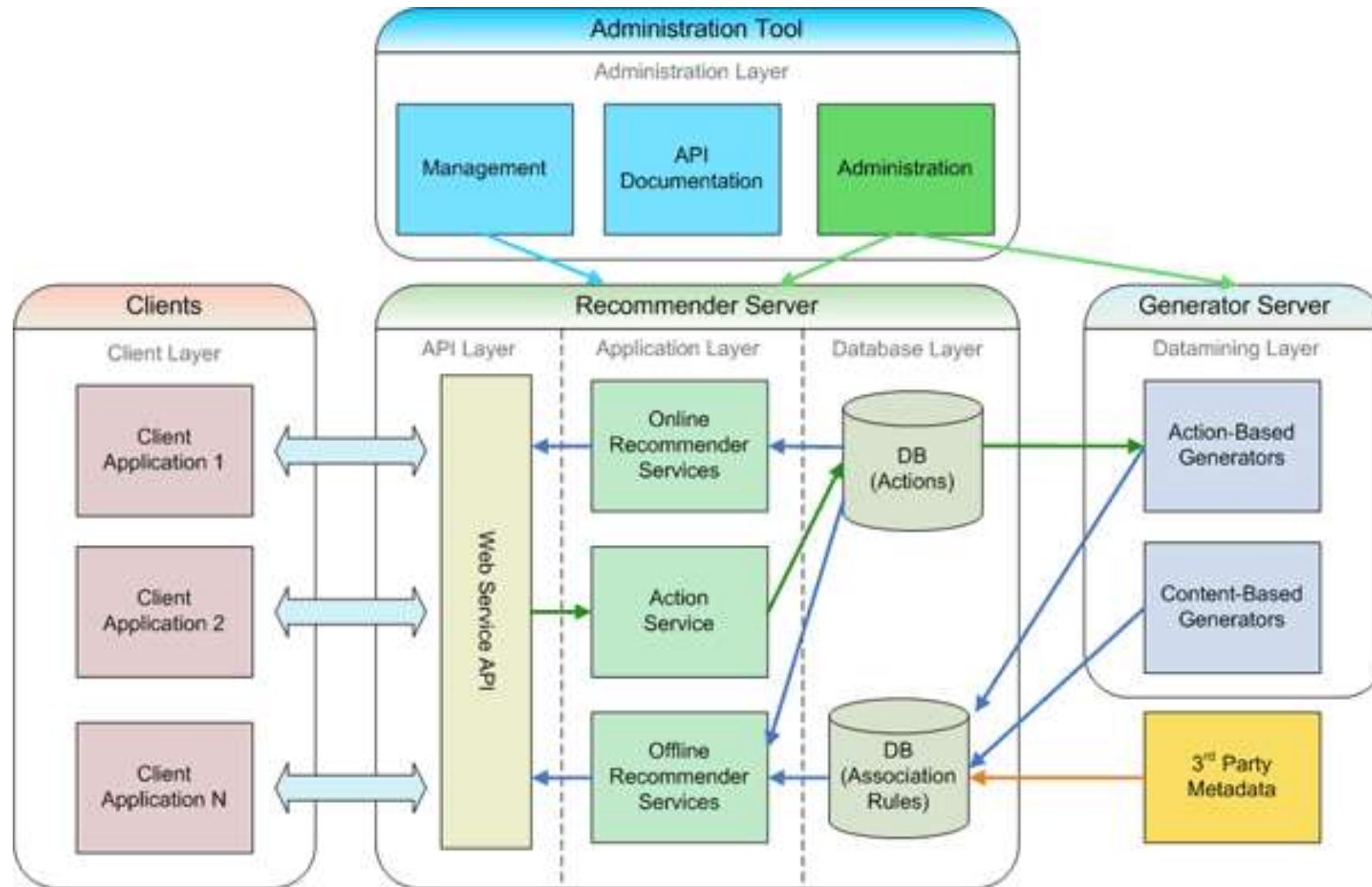
# Web application development – Introduction

- Web Application security is a branch of information security that deals with the security of web applications, web services, and websites. It is a kind of application security that is applied on to web or internet level specifically.
- Web security is important as web applications get attacked due to bad coding or improper sanitizing of application inputs and outputs. Common web security attacks are Cross-site scripting(XSS) and **SQL Injections**.
- Apart from XSS, SQL Injections, the other types of web security attacks are Arbitrary code execution, Path Disclosure, Memory corruption, Remote file inclusion, **Buffer overflow**, local file inclusion, etc.
- Web security is completely based on people and processes. Hence it is extremely important that the developers **use proper coding** standards and sanity checks for any such web security threats before making websites go live.

# Web application development – Definition

- Web application security is the practice of protecting websites, applications, and APIs from attacks.
- It is a broad discipline, but its ultimate aims are keeping web applications functioning smoothly and protecting business from cyber vandalism, data theft, unethical competition, and other negative consequences.

# Web application development – Architecture



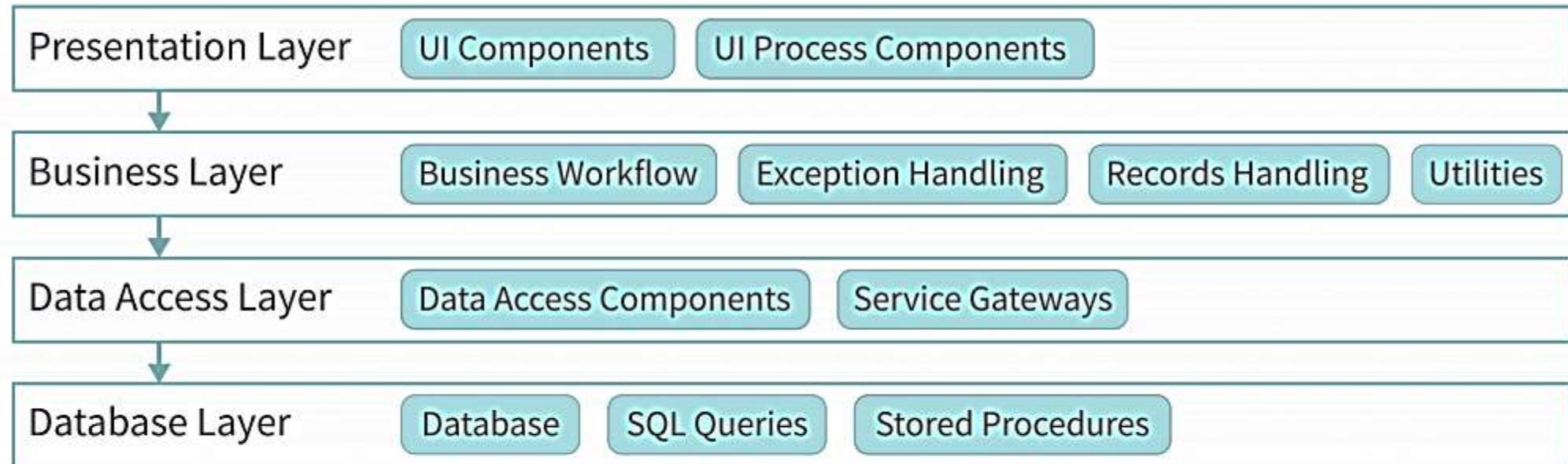
# How Does Web App Architecture Work?

- The two basic parts of every application are:
  - 1.The code is located on the user's machine and is composed of HTML, CSS, and JavaScript, which is known as client-side code. On the client, user behavior is what happens.
  - 2.The server side contains the logic to handle transactions and respond to HTTP requests. Java, PHP, Ruby, Python, and so on are examples of server-side code.

# Web application development – Architecture

- There are two programs running concurrently:
  - The code which lives in the browser and responds to user input
  - The code which lives on the server and responds to [HTTP requests](#)
- Web application architecture defines precisely how an application will function. Some features include:
  - Delivering persistent data through HTTP, which can be understood by client-side code and vice-versa
  - Making sure requests contain valid data
  - Offers authentication for users
  - Limits what users can see based on permissions
  - Creates, updates and deletes records

# Layers of Web App Architecture



# Layers of Web App Architecture

1. The **presentation layer** consists of HTML, CSS, JavaScript, and its frameworks, which provide an HTML template for the presentation layer and enable interaction between the interface and browser.
2. The **business logic** and rules are located in the business layer. A browser request is processed here, followed by the execution of the business instructions specified in that request, and then passed on to the presentation layer.
3. The **data persistence layer** is part of the persistence layer and is also known as the **Data Access Layer**. It connects to the business layer and, in addition to data retrieval, also manages data storage. Data is retrieved from the database servers by the data persistence layer.
4. The business logic is connected to the client-side via the **database layer**, which protects data integrity by separating it from the business.

# Web Application Components

- Web app components consist of two parts –
- The architecture of an application does not impact the appearance of its user interface. For example, activity logs, configuration settings, dashboards, statistics, widgets, notifications, and the like are simply displayed on a web page and have no effect on it.
- They provide information, widgets, and notifications in addition to enhancing the user experience.
- Structural web components are client-side and server-side components that work with web applications. HTML, CSS, and JavaScript are frequently used to create these components.
- Business logic is handled by the web app server, which is composed of PHP, Java, Python, Node.js, .NET, and Ruby on Rails. PHP, Java, Python, Node.js, .NET, and Ruby on Rails are used to build server components.

# Models of Web Application



# Types of Web Application Architecture : 1. SOA

## How SOA Works

A Web services SOA involves three primary entities that communicate with one another: a producer or provider of services, a consumer or requester of services, and a directory, registry or broker of available services.

**PUBLISH:** The service provider (which may be internal or external to an organization) publishes to a broker or registry a list of services it can provide.



3

**FIND:** The consumer application queries the registry to find the service it needs.

The registry provides the service and access information to the consumer application.

2

1  
**SERVICE PROVIDER**



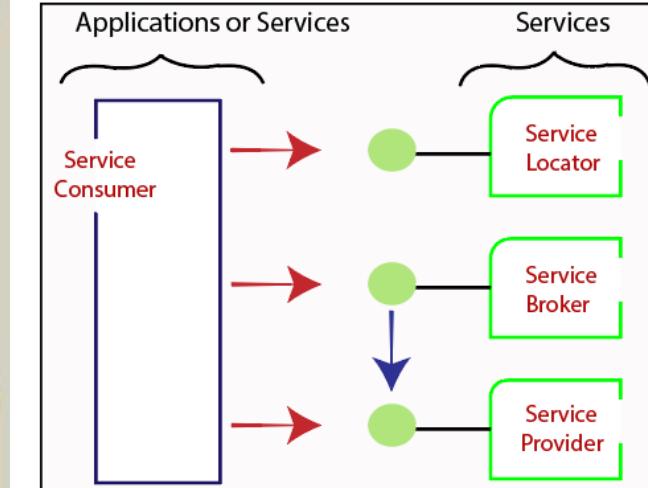
5

The consumer then requests the service directly from the service provider.

4  
**SERVICE CONSUMER**

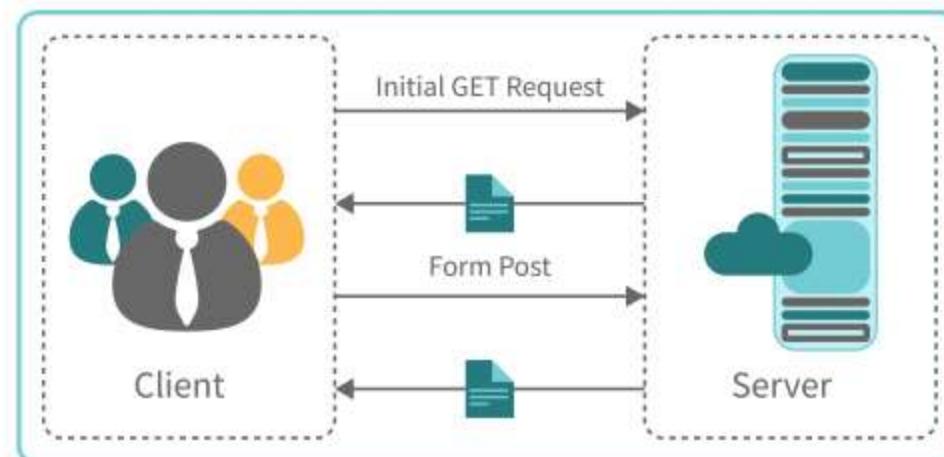


**BIND:** The service provider authorizes the consumer to use the specific service.

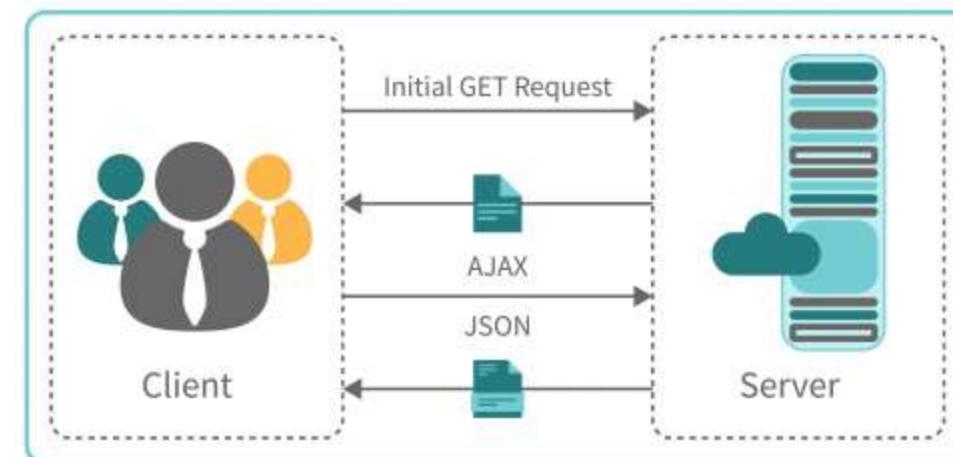


# Types of Web App Architecture – 2. SPA (Single Page Applications)

- seeks to overcome the classic difficulty of constructing smooth apps, in order to provide an easy-to-use, intuitive user experience.
- A SPAs loads a single web page and updates the data on that page with dynamically generated content rather than loading a new page. Instead of loading a new page, SPAs load a single page and refresh the data on that page with dynamically updated content.
- The front end receives the same logic as the back end. JavaScript frameworks are used to develop SPAs on the client side using client-side **JavaScript frameworks**.



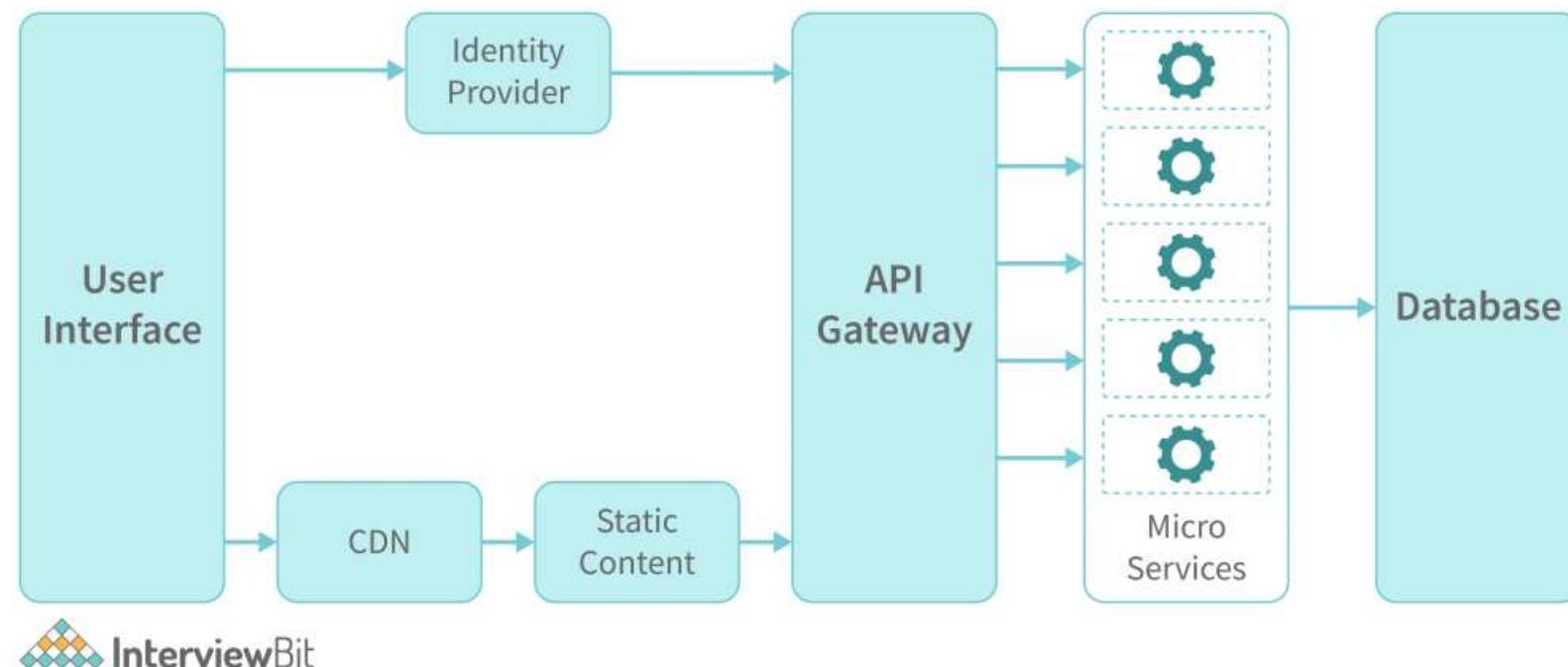
Traditional Page Life Cycle



SPA Life Cycle

# Types of Web App Architecture – 3. Microservice Architecture

- Monolithic architectures are now outcompeted by microservice architectures, which are built around a multitude of services that can function asynchronously to solve complicated problems. Because APIs are used to communicate between services, they are loosely associated.
- Microservices have eliminated the difficulty of deploying service components in a monolithic fashion.

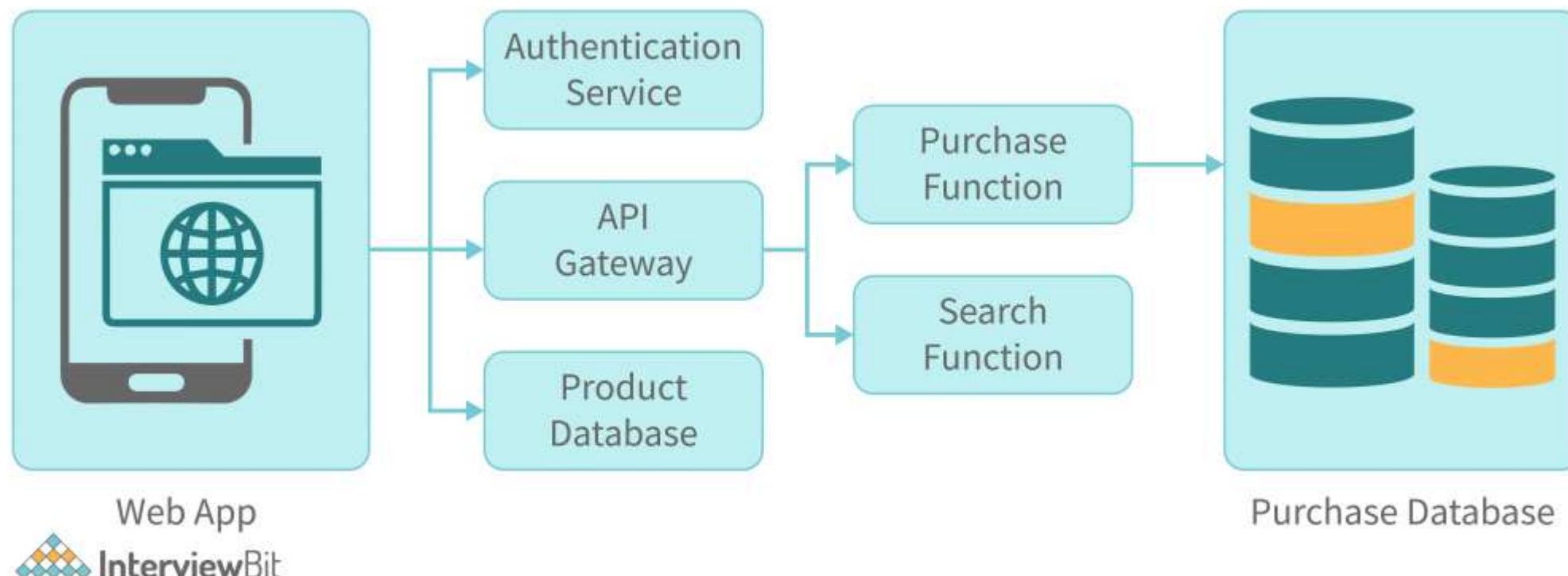


# Types of Web App Architecture – 4. Serverless Architecture

- Cloud service providers such as Amazon and Microsoft manage the servers that are used by serverless architectures, so no manual intervention is required on the server.
- No matter how complex your code is, cloud service providers can handle everything for you— you don't need to deploy them manually on your server.
- Serverless architecture is a design pattern in which applications are built and run without any human intervention on servers managed by third-party cloud service providers like Amazon and Microsoft.

# Serverless Architecture.....

- It allows you to focus on the quality of the product and the difficulty of making them highly scalable and reliable. **Backend-As-A-Service (BaaS) and Function-As-A-Service (FaaS)**



# BaaS and FaaS

- Developers who use BaaS to eliminate the back-end operations can focus on building the front-end aspects of their applications without worrying about operating the back-end.
- Amazon Amplify, for example, is a popular BaaS product.
- FaaS, on the other hand, is an event-driven model that allows developers to concentrate on coding and event triggers. The remaining work will be handled by FaaS service providers such as **Amazon Lambda**.

# Types of Web App Architecture – 5. Progressive Web Applications

- In 2015, Google created Progressive Web Apps (PWAs). To develop apps that offer rich and native functionality with enhanced capabilities, reliability, and ease of installation, PWAs were created.
- A PWA is capable of functioning on any browser and on any device.

# Tools and choices that can help deliver the best web app experience:

1. There are IDEs for productivity enhancement in **Webstorm**, **Github's Atom**, **NetBeans**, and **AWS Cloud9**.
2. Design and improve user experience with UX Builder tools: **Figma**, **Sketch**, and **Invision** are among the most popular today.
3. Integration tools such as **MuliSoft**, **Cleo**, **JitterBit**, and **Automate.io** offer a seamless, entertaining, and unified user experience.
4. There are a lot of popular frameworks such as **React**, **Angular**, **Python**, **Vu**, **Express**, **Django**, and so on, that can be utilised to create quality products.

## Topic 2: Client Side Technologies and Framework

## Basics .....Client Vs Server Side Scripting

### CLIENT SIDE SCRIPTING

HTML, CSS, and javascript are used.

The source code is visible to the user.

It does not provide security for data.

Its main function is to provide the requested output to the end user.

It usually depends on the browser and its version.

It runs on the user's computer.

### SERVER SIDE SCRIPTING

PHP, Python, Java, Ruby are used.

The source code is not visible to the user because its output of server-side is an HTML page.

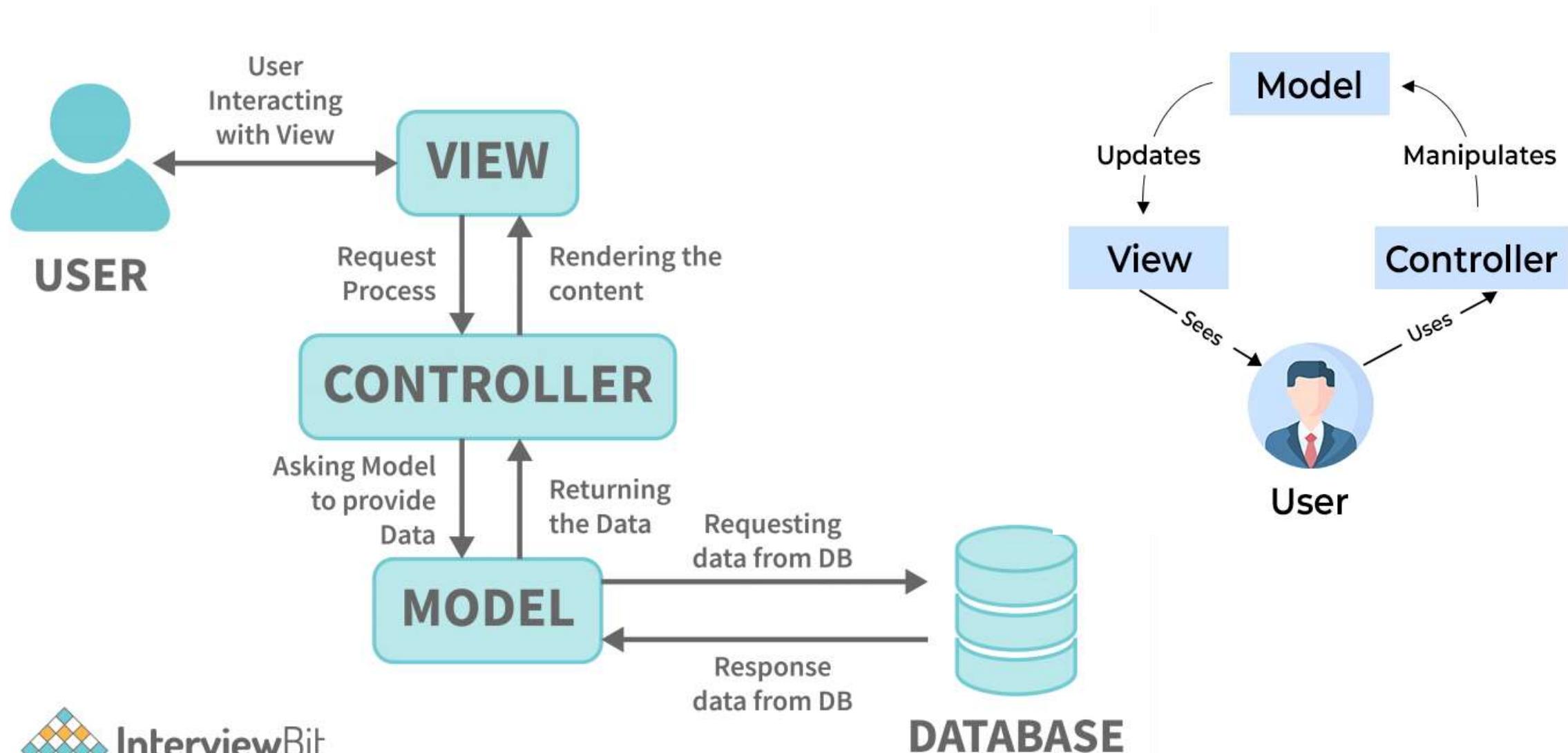
It provides more security for data.

Its primary function is to manipulate and provide access to the respective database as per the request.

In this any server-side technology can be used and it does not depend on the client.

It runs on the webserver.

# Basics .....MVC architecture



## Client Side Rendering



1. User requests a website



2. Server sends HTML file with Javascript links



3. Browser downloads HTML



4. Browser downloads CSS and Javascript



5. Browser executes framework or library



6. Browser loads the website



# Common client-side web technologies

1. [HTML](#)

2. [CSS](#)

3. [JavaScript](#)

- By separating the content of the page (the HTML) from its layout and styling (the CSS), and its behavior (via JavaScript), complex web apps can leverage the Separation of Concerns principle.
- **CSS preprocessors**
- CSS preprocessors can help your stylesheets follow the [DRY principle](#) by adding support for variables and logic. (The Don't Repeat Yourself (DRY) principle states that duplication in logic should be eliminated via abstraction; duplication in process should be eliminated via automation.)
- **Legacy web apps with jQuery**
- Although ancient by JavaScript framework standards, jQuery continues to be a commonly used library for working with HTML/CSS and building applications that make AJAX calls to web APIs. However, jQuery operates at the level of the browser document object model (DOM), and by default offers only an imperative, rather than declarative, model.

# Client Side Technology : 10 year before

## Client-Side Programming

- Aims to enhance the appearance of web sites
- JavaScript, ActiveX, Java, Flash, Silverlight, etc.
- Doesn't involve any server processing
- The complete application is downloaded to the client browser and browser executes it locally
- Provides immediate feedback to the user
- Reduces the load on a server
- Reduces network traffic
- Client-side technologies are not supported equally by all browsers and operating systems

# Client Side Technology : 2023

TOP

## 10 client side technology considerations for web development companies



*Knockout.*



MERCURY



# Client-Side Frameworks

- Many web developers find it useful to build their web applications on top of a client-side framework library.
- These libraries are “frameworks” in the sense that they build a new higher-level API for client-side programming on top of the standard and proprietary APIs offered by web browsers: once you adopt a framework, your code needs to be written to use the APIs defined by that framework.
- The obvious benefit of using a framework is that it is a higher-level API that allows you to do more with less code.
- A well-written framework will also address many of the compatibility, security, and accessibility issues described above.

# Client Side Frameworks - SPA

- **Angular**
- Originally developed and referred to as AngularJS, Angular is an open source, front-end, [JavaScript](#) framework that can be used with any platform. It was designed by **Google** to allow developers to create dynamic and interesting web applications.
- With Angular, developers can build front-end-based apps without using any other plugins or frameworks. Websites such as Netflix have been built using Angular.
- **React**
- Originally developed and maintained by **Facebook**, React is an declarative front-end JavaScript library aimed at helping web developers build user interfaces through a "learn once, write anywhere" approach. This framework is best used with single-page or mobile apps as it is simple to use, scalable and fast.
- **Backbone.js**
- This JavaScript framework uses a model-view-presenter design model that allows developers to create single-page web applications. **Airbnb, Hulu, Netflix and LinkedIn Mobile** all use Backbone.js, a widely used and supported framework.
- Backbone's main benefit is that it separates business logic and user interface. This means developers can easily make changes to the application logic without affecting the user interface and vice versa. Since Backbone is lightweight and flexible, it can be used with any back-end language or front-end template.
- **Bootstrap**
- Created by **Twitter**, Bootstrap is a HTML, [CSS](#) and JavaScript open source framework that developers can use to build responsive and native mobile websites. [Bootstrap](#) is used by companies like **Lyft, Vogue and Newsweek**.

# Client Side Frameworks - SPA

- **Cordova**
- Cordova is an open source mobile application development framework backed by **Apache**. It allows standard web languages to be wrapped and presented as native mobile applications. Cordova is widely used by companies like Oracle, SAP and IBM. it provides web applications with [Node-API](#) that maps standard web languages to native functionality, such as a phone's GPS, camera and accelerometer.
- Cordova allows developers to build their applications using standard web languages while still being able to access functionality using native code, as well as market their apps on native app stores.
- **Vue**
- Vue is a progressive framework developers can use to build user interfaces. The core library is focused on the view layer only and can be easily picked up and integrated with other libraries or existing projects. When used with [modern tooling](#) and [supporting libraries](#), Vue can power sophisticated single-page applications.
- **Blazor WebAssembly**
- Unlike other JavaScript frameworks, Blazor WebAssembly is a single-page app (SPA) framework for building interactive client-side web apps with .NET. Blazor WebAssembly uses open web standards without plugins or recompiling code into other languages. Blazor WebAssembly works in all modern web browsers, including mobile browsers.

# SPA framework

## jQuery vs a SPA Framework

Factor	jQuery	Angular
Abstracts the DOM	Yes	Yes
AJAX Support	Yes	Yes
Declarative Data Binding	No	Yes
MVC-style Routing	No	Yes
Templating	No	Yes
Deep-Link Routing	No	Yes



Created by [Johannes Hoppe](#)

# Implementing HTML 5 in a secure fashion



## 1 Web Messaging

- Web Messaging (also known as Cross Domain Messaging) provides a means of messaging between documents from different origins.
- However, some recommendations to keep in mind:

1. When posting a message, explicitly state the expected origin as the second argument to `postMessage` rather than \* in order to prevent sending the message to an unknown origin after a redirect.

The receiving page should always:

1. Check the `origin` attribute of the sender to verify the data is originating from the expected location.
2. Perform input validation on the `data` attribute of the event to ensure that it's in the desired format.

# Implementing HTML 5 in a secure fashion

2. Both pages should only interpret the exchanged messages as data. Never evaluate passed messages as code (e.g. via `eval()`) or insert it to a page DOM (e.g. via `innerHTML`), as that would create a DOM-based XSS vulnerability.
3. To assign the data value to an element, instead of using a insecure method like `element.innerHTML=data;`, use the safer option: `element.textContent=data;`
4. Process the messages (`event.data`) as data and never evaluate the content as HTML or script code.
5. Always check the origin attribute of the message (`event.origin`) to ensure the message is coming from a trusted domain. Use an allow-list approach.

# Credential and Personally Identifiable Information (PII) Input hints

## 6. Protect the input values from being cached by the browser.

- Access a financial account from a public computer. Even though one is logged-off, the next person who uses the machine can log-in because the browser autocomplete functionality.
- To mitigate this, we tell the input fields not to assist in any way.
- `<input type="text" spellcheck="false" autocomplete="off" autocorrect="off" autocapitalize="off"></input>`
- Text areas and input fields for PII (name, email, address, phone number) and login credentials (username, password) should be prevented from being stored in the browser. Use these HTML5 attributes to prevent the browser from storing PII from your form:
  - `spellcheck="false"`
  - `autocomplete="off"`
  - `autocorrect="off"`
  - `autocapitalize="off"`

# Implementing HTML 5 in a secure fashion

## 7. Offline Applications¶

- Whether the user agent requests permission from the user to store data for offline browsing and when this cache is deleted, varies from one browser to the next.
- **Cache poisoning** is an issue if a user connects through insecure networks, so for privacy reasons it is encouraged to require user input before sending any manifest file.
- Users should only cache trusted websites and clean the cache after browsing through open or insecure networks.

## 8. Geolocation¶

The Geolocation API requires that user agents ask for the user's permission before calculating location. Whether or how this decision is remembered varies from browser to browser.

Some user agents require the user to visit the page again in order to turn off the ability to get the user's location without asking, so for privacy reasons, it's recommended to require user input before calling `getCurrentPosition` or `watchPosition`.

# Implementing HTML 5 in a secure fashion

## • 9. Tabnabbing

- Tabnabbing is a computer exploit and phishing attack, which persuades users to submit their login details and passwords to popular websites by impersonating those sites and convincing the user that the site is genuine.
- The attack's name was coined in early 2010 by Aza Raskin, a security researcher and design expert.
- [1][2] The attack takes advantage of user trust and inattention to detail in regard to tabs, and the ability of browsers to navigate across a page's origin in inactive tabs a long time after the page is loaded.
- **Tabnabbing is different from most phishing attacks in** that the user no longer remembers that a certain tab was the result of a link unrelated to the login page, because the fake login page is loaded in one of the long-lived open tabs in their browser.[3]
- **To prevent this issue, the following actions are available:**

# 9. Tabnabbing

- Cut the back link between the parent and the child pages:
- For **HTML links**:
  - To cut this back link, add the attribute ***rel="noopener"*** on the tag used to create the link from the parent page to the child page. This attribute value cuts the link, but depending on the browser, lets referrer information be present in the request to the child page.
  - To also remove the referrer information use this attribute value:  
***rel="noopener noreferrer"***.
- For the **JavaScript** `window.open` function, add the values ***noopener,noreferrer*** in the `windowFeatures` parameter of the `window.open` function.

# 10. Reverse tabnabbing

- Reverse tabnabbing is an attack where a page linked from the target page is able to rewrite that page, for example to replace it with a phishing site.
- As the user was originally on the correct page they are less likely to notice that it has been changed to a phishing site, especially if the site looks the same as the target. If the user authenticates to this new page then their credentials (or other sensitive data) are sent to the phishing site rather than the legitimate one.



# Best practices for Securing thru HTML page

1. 1. Setup Captcha for Forms(reCAPTCHA, hCaptcha)
2. 2. Double opt-in form
3. 3. Add a test question
4. 4. Add Honeypots
5. 5. Implement time-analysis.
6. 6. Hide target request
7. 7. Form validation after geolocation of the IP address.
8. 8. Blacklist IPs and limited IP addresses
9. 9. Email address validation API
10. 10. Validation
11. 11. Block copy and paste in your forms

# Securing Cascading Style Sheets

- [Self Study Topic](#)
- [Securing Cascading Style Sheets - OWASP Cheat Sheet Series](#)

# **JavaScript security**

# JavaScript vulnerabilities

- Like nearly any programming language, JavaScript is not without its share of potential security exposures.
- Exploiting JavaScript vulnerabilities can
  - manipulate data,
  - redirect sessions,
  - modify and steal data, and much more.
- Although JavaScript is typically thought of as a client-side application, JavaScript security issues can create problems on server-side environments as well.
- The best defense against common JavaScript security vulnerabilities is to be aware of them and implement the proper controls to reduce exposure.

# 8 JavaScript Security Vulnerabilities

---



- 01** Source Code Vulnerabilities
- 02** Unintended Script Execution
- 03** Escaping/Encoding User Input
- 04** Filtering Input
- 05** Input Validation
- 06** Reliance on Client-Side Validation Alone
- 07** Stealing Session Data
- 08** Inducing Users to Perform Unintended Actions

# What Is JavaScript Security?

- JavaScript security is related to investigating, preventing, protecting, and resolving security issues in applications where JavaScript is used.
- JavaScript itself is a fundamental technology for building web applications and is also very popular for building server-side, desktop, and even mobile applications.
- Its widespread popularity, however, also makes it a prime target for hackers, looking to target it through various attack vectors. Because JavaScript is used mostly in the front-end, it makes sense to focus first on JavaScript security issues in browsers.

# JavaScript Language Security

1. Improper validation
2. Lack of sanitization
3. Evaluating untrusted code (e.g., JS allows eval)
4. Writing overly complex code with multiple nested loops and functions doing too much
5. Not making data private, or hidden, by default. (Why are you not using property descriptors? 😐 )
6. Not understanding mutability or forgetting to make objects immutable
7. Missing declarations
8. Improper initialization
9. Misusing null and undefined
10. Not using locales
11. Misusing operators
12. Using the wrong numeric types
13. Not understanding characters and character encoding
14. Not understanding scope rules
15. Dealing with prototypes wrong
16. Getting asynchronous ordering wrong (e.g., messing up promise chains)
17. Blocking the event loop
18. Not using strict mode



# Securing JavaScript

## Bare Minimal Steps

<https://www.synopsys.com/blogs/software-security/javascript-security-best-practices/>

# 1. JavaScript Integrity Checks

As a frontend developer, you may have used `<script>` tags to import third-party libraries. Have you thought about the security risks of doing so?

**What if the third-party resource has been tampered with?**

Yes, these are things that can happen when you render external resources on your site. As a result, your site may face a security vulnerability.

As a safety measure for this, you can add an **integrity** (also known as [Subresource integrity — SRI Check](#)) code to your script as follows.

```
<script  
src="https://code.jquery.com/jquery-3.3.1.slim.min.js"  
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzbzo5smXKp4YfRvH+8abTE1Pi6jizo"  
crossorigin="anonymous">  
</script>
```

When an external resource with the subresource integrity check is downloaded, a base64-encode hash is generated based on the received data. If this generated hash value differs from the one provided by the developer, the resource is not loaded.

## 2. Frequent Tests for NPM Vulnerabilities

- Widespread use of public packages and libraries. NPM, a lead player in the JavaScript ecosystem, offers more than a million packages in its registry.
- While the sheer variety offered is certainly an advantage, this also means there are potentially a huge number of hidden vulnerabilities in these packages that are installed in web application projects.
- GitHub introduced a bot name **Dependabot**, to scan the NPM dependencies automatically and notify you by email stating the risks.

### 3. Do not trust user input: Have Validations in Place to Avoid Injections

- Most web applications allow users to insert data through text input. After the data is inserted, it is reflected somewhere in the web application. But accepting and displaying inputs from users opens the door to cross-site scripting attacks in which cybercriminals use special characters to trick browsers into interpreting text as HTML markup or JavaScript code.
- For Example, if you type in the comment field anything with quotes <script><script/>, those quotes will be replaced with double — <<script>><</script>>.
- Output encoding transforms potentially dangerous characters into a safe form. The encoding mechanism to use depends on where the untrusted input data is placed. Some of the encoding that the Open Web Application Security Project (OWASP) recognizes are HTML entity encoding, HTML attribute encoding, URL encoding, JavaScript string encoding, and CSS Hex
- OWASP recommends using a **security-focused encoding library** to make sure that encodings are implemented properly.
- When a web application needs to accept HTML inserted by the user, sanitize the input before displaying it on a page or sending it to other systems.
- **HTML sanitization** involves validating the input and cleaning up unexpected characters. The outcome should be a safe HTML version of the HTML input. An excellent tool to sanitize HTML is [DOMPurify](#).

# 4. Content Security Policies (CSP)

- Content Security Policies (CSP) are another way to avoid malicious injections. Using CSP is quite straightforward as follows.
  - Content-Security-Policy: trusted-types;
  - Content-Security-Policy: trusted-types 'none';
  - Content-Security-Policy: trusted-types <policyName>;
  - Content-Security-Policy: trusted-types <policyName> <policyName> 'allow-duplicates';

# 5. Always Keep Strict Mode On

- Having Strict mode “on” will limit you from writing unsafe code. Besides, it’s straightforward to enable this mode. It’s as simple as adding the below line as the first in your JavaScript files.
  - `use strict`
- When the strict mode is on;
  - It throws errors for some errors that were previously kept silent.
  - Fixes mistakes that make it difficult for JavaScript engines to perform optimizations.
  - Prohibits the use of reserved words likely to be defined in future versions of ECMAScript.
  - Throws errors when ‘unsafe’ actions are taken (such as gaining access to the global object).
- Every modern browser has supported strict mode for years. If the browser does not support strict mode, the expression is simply ignored.

# 6. Lint Your Code

- Linters perform static analysis on your codebase. It helps to establish quality and avoid common pitfalls. Since quality goes hand in hand with security, Linting helps to reduce the security risks. Few popular tools that we use for JavaScript as follows.
  - JSLint
  - JSHint
  - ESLint
- Further, tools like **SonarCloud** can also be used to identify code smells and known security vulnerabilities.

# 7. Write quality code

- The following JavaScript security best practices can reduce this risk.
- **Avoid `**eval()**`:** Don't utilize this command in code, since it simply executes passed argument if it is a JavaScript expression. This means if the hacker succeeds in manipulating input value, he or she will be able to run any script she wants. Instead, opt for alternative options that are more secure.
- **Encrypt:** Use `HTTPS/SSL` to encrypt data exchanged between the client and the server.
- **Set secure cookies:** To ensure SSL/HTTPS is in use, set your cookies as "secure," which limits the use of your application's cookies to only secure web pages.
- **Set API access keys:** Assign individual tokens for each end user. If these tokens don't match up, access can be denied or revoked.
- **Use safe methods of DOM manipulation:** Methods such as `innerHTML` are powerful and potentially dangerous, as they don't limit or escape/encode the values that are passed to them. Using a method like `innerText` instead provides inherent escaping of potentially hazardous content. This is particularly useful in preventing DOM-based XSS attacks.

# 8. Protect your cookies

- HTTP cookies are used to preserve information such as authentication or session tokens, user preferences, or anything that the server should remember between requests. A HTTP cookie is a small string of data that the web server sends to the browser using the Set-Cookie HTTP header in the response. After this, the browser will automatically send the cookie back on almost every request (to the same domain) using the Cookie HTTP header.
- If you do not set any flags, the cookie content is accessible programmatically using document.cookie. This is not always desirable. If an attacker is able to inject JavaScript within a web application, the script could read the content of document.cookie, and that could enable the bad actor to access any sensitive information in the cookie.
- There are a variety of ways to protect cookies. If the cookie is used only by the webserver, you can use the httpOnly flag to restrict programmatical access to the cookie's content.

# 8. Protect your cookies .....

- HTTP does not encrypt messages, so man-in-the-middle attacks could succeed, and messages could be intercepted. If the cookie holds sensitive information, restrict the browser from sending it over unencrypted HTTP connections. Use the `secure` flag to instruct the browser to send cookies only through HTTPS, a protocol extension of HTTP. If you do not use this flag, the browser will send the cookie using both secure (<https://my-secure-site.com>) and insecure (<http://my-secure-site.com>) connections to a site.
- In the case of a cross-site request forgery attacks, attackers can succeed because web applications can't differentiate between valid requests and forged requests. For example, say you have a form for updating a user's password, and the website uses cookies for handling sessions.
- An attacker could create a page that automatically sends a request to update your password.
- If cookies aren't protected and you have a valid session cookie, when you visit this site, a request to update your password is sent from the site.
- Since your session cookie is valid, the browser will automatically include the cookie in the request. With this information, the webserver receives a request with a valid session cookie and updates the password. Thus, an attacker can reset your password to the value of their choice.
- **To protect your cookies against cross-site request forgery attacks, use the cookie flag `samesite=strict` to ensure that the cookie is not sent when the request comes from another domain than the site that sets the cookie.**

# 9. Defend against prototype pollution

- JavaScript is a prototype-based language. When an object is created, it inherits all properties and methods following the so-called prototype chain. As it is a chain, prototypes have references to other prototypes. The chain is followed until reaching `null`. The `Object` prototype sits just below `null` in this prototype chain. Almost all objects in JavaScript are instances of `Object`.
- When someone targets `Object` and alters the methods that most objects in JavaScript inherit through the prototype chain, this is called prototype pollution attack. It can happen in the server side or client side, and its consequences can include remote code execution, cross-site scripting, and denial-of-service attacks.
- On the client side, an attacker injects code that can modify the `Object.prototype` directly. On the server side, the application is if it recursively clones object properties or if it sets object properties through a given path. If we have an HTTP server accepting HTTP requests, an attacker could send an HTTP request payload that contains a JSON object that tries to exploit the `_proto_`, `constructor`, or `prototype` properties.

# 9. mitigate prototype pollution attacks.

- Freeze Object.prototype to prevent the default prototype from getting polluted.
  - ...
    - // This should be executed only once
    - Object.freeze(Object.prototype);
    - ...
    - clone(source, target);
    - ...
- Create prototypeless objects using Object.create(null). These types of objects do not have a prototype. As such, they do not inherit methods from Object.prototype.
  - ...
  - const target = Object.create(null);
  - clone(source, target);
  - ...
- Use Map instead of Object.
- Avoid unsafe recursive merge functions.

# 10. Evaluate the need of third-party libraries

- Code reuse is seen as a good practice, and JavaScript developers have pushed this idea to the extreme, creating packages even for the simplest tasks. But reusing packages in a noncontrolled way exposes JavaScript applications to issues and security threats. Consider treating dependencies as code needed for the project to run. The more that is needed, the more points of failures there can be.
- By using well-known secure libraries and frameworks, you are protecting yourself. When using less-known third-party packages, inspect who has developed it; whether the package is maintained, inactive, and well-tested; and if it has unpatched vulnerabilities. And make sure you're installing the right package
- **typosquatting attacks**, where malicious packages with similar names to well-known packages make their way into real applications, are common.
- You should also use **software composition analysis (SCA)** tools to help you to detect open source license violations, vulnerabilities, and out-of-date dependencies in open source software.

# 11. Secure browser communication between window objects



- Often, web pages present information to users by opening other windows or iframes. Sometimes these windows and iframes require communication between each other.
- But enabling communication between pages from different subdomains by setting the same value for `document.domain` on both pages should be avoided as it weakens the security protections provided by the same-origin policy. Instead, the `window.postMessage()` method should be used.
- The `window.postMessage()` method enables safe communication between window objects such as a page and a pop-up or with an iframe.
- It allows two windows to share messages without one having any direct control over the other one. Although it's a safer method, you must be careful with how you use it. For instance, `window.postMessage()` accepts a `targetOrigin` parameter, but you should avoid using \* as the `targetOrigin` and make sure that the URL you provide uses HTTPS.
- The window receiving the message should always verify the sender's identity (by checking `message.origin`), as well as validate the received message. Do not execute strings coming through the `window.postMessage()` method to run as code, and avoid listening on message events if you are not expecting them.

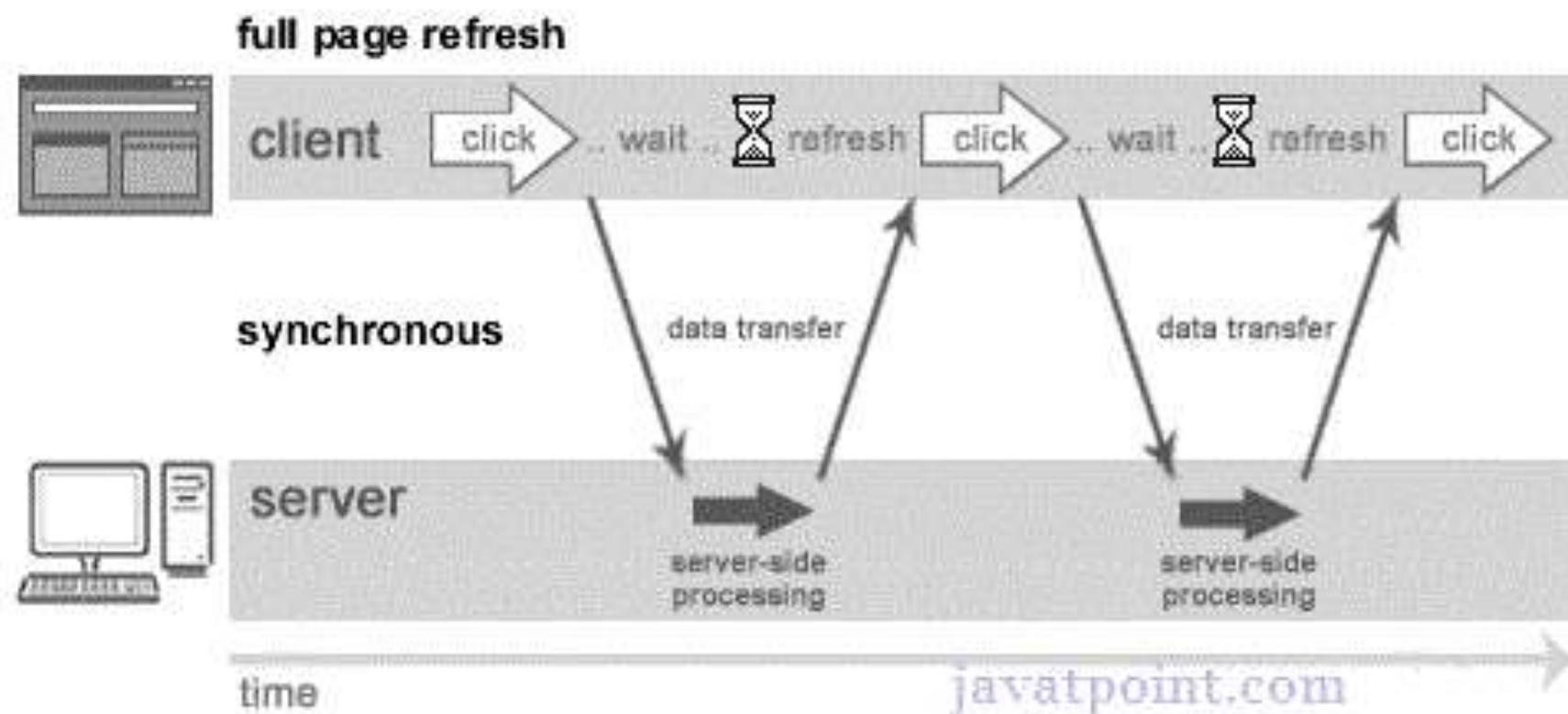
# ajax

# Asynchronous JavaScript and XML

- It is the use of the XMLHttpRequest object to communicate with servers. The Ajax was publicly used on 18 February 2005 by Jesse James Garrett.
- AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like [JavaScript](#), [DOM](#), [XML](#), [HTML/XHTML](#), [CSS](#), [XMLHttpRequest](#) etc.
- AJAX allows you to send and receive data asynchronously without reloading the web page.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

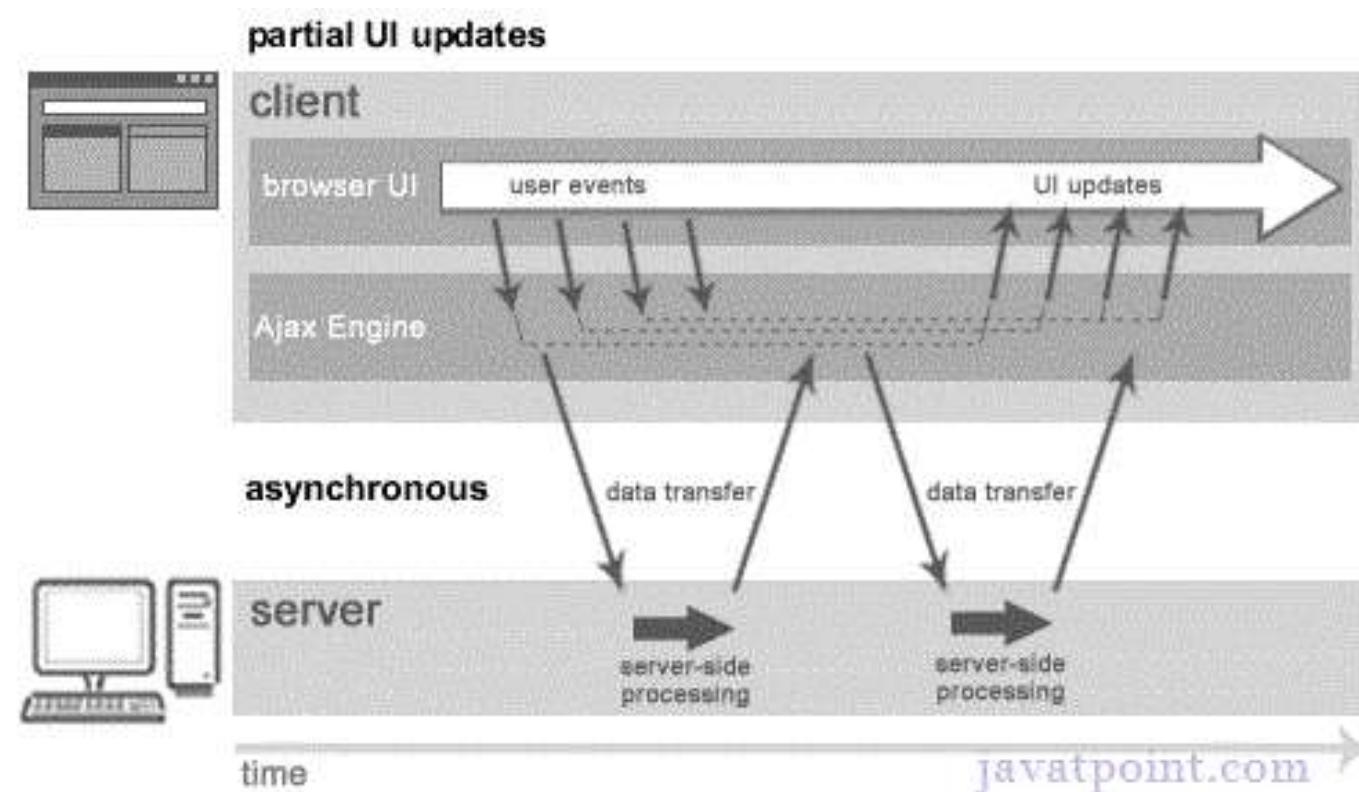
# Synchronous (Classic Web-Application Model)

- A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.
- full page is refreshed at request time and user is blocked until request completes.
- 



# Asynchronous (AJAX Web-Application Model)

- An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.
- Full page is not refreshed at request time and user gets response from the ajax engine.



# AJAX Technologies

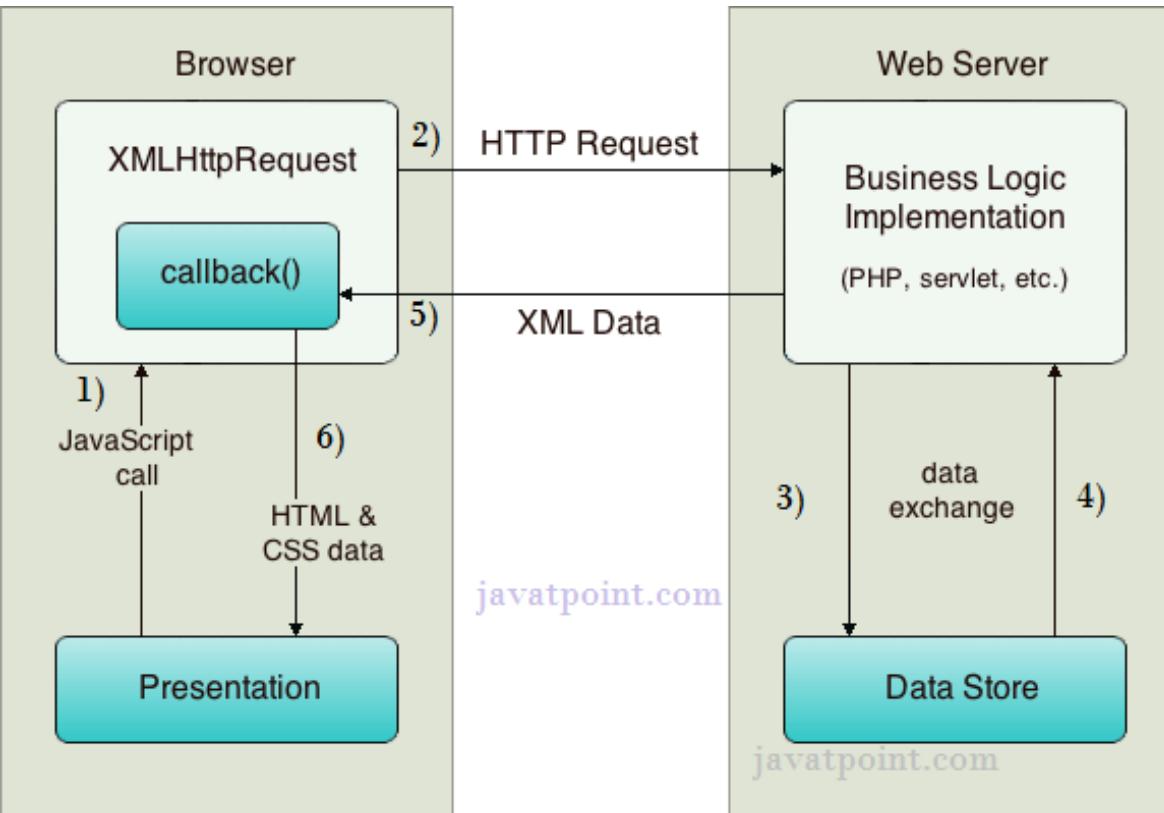
- Ajax is not a technology but group of inter-related technologies. [AJAX](#) technologies includes:
- [HTML/XHTML](#) and [CSS](#)
- DOM
- [XML](#) or [JSON](#)
- [XMLHttpRequest](#)
- [JavaScript](#)
- An object of XMLHttpRequest is used for asynchronous communication between client and server.
- It performs following operations:
  1. Sends data from the client in the background
  2. Receives the data from the server
  3. Updates the webpage without reloading it.

# Properties of XMLHttpRequest object

Property	Description
onreadystatechange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4. <b>0 UNOPENED</b> open() is not called. <b>1 OPENED</b> open is called but send() is not called. <b>2 HEADERS_RECEIVED</b> send() is called, and headers and status are available. <b>3 LOADING</b> Downloading data; responseText holds the data. <b>4 DONE</b> The operation is completed fully.
responseText	returns response as text.
responseXML	returns response as XML

# How AJAX works?

- AJAX communicates with the server using XMLHttpRequest object.



1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

# Important Features of AJAX

- AJAX is a user-friendly approach.
- Does not depend on server technology.
- Makes web pages faster.
- Increases the performance of the web page.
- Support for client-side template rendering.
- Assists in the data view control.
- Support for live data binding.
- Reduces consumption of server resources.
- Responsive and rich user interfaces.
- Needs no traditional form to submit and the whole page refresh.
- Only some part of the page is refreshed/tweaked.
- Processing is similar for all browser types.
- Faster interaction and development of web applications.
- The server uses a reduced amount of bandwidth.
- Improves user's interactivity.
- Offers better usability.

# The advantages of Ajax

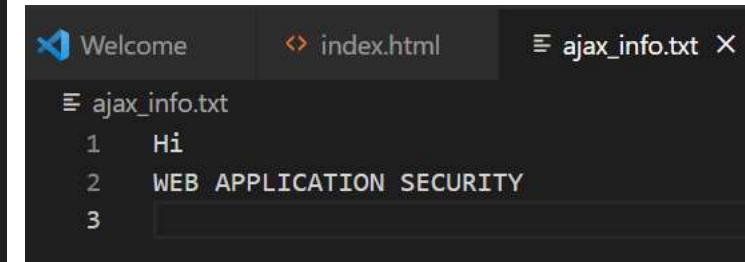
- **Improved user experience** – Ajax's richer user experience is its main advantage. Ajax allows web pages to be continuously updated, but requires little data to interact with the server. In this way, a part of the web page can be updated without having to update the whole web page.
- **Enhance user productivity** – Ajax library provides object-oriented auxiliary functions, which can reduce trouble and enhance productivity for users.
- **Reduce bandwidth usage and increase speed** – Ajax uses client scripts to communicate with web servers and JavaScript to interact with data. Using Ajax can reduce network load and bandwidth usage and only get the data you need. This gives you a faster interface and a lower response time. Faster response, resulting in increased performance and speed
- **Support asynchronous processing** – use XMLHttpRequest for asynchronous data acquisition, which is the backbone of Ajax applications. Therefore, requests can be processed effectively, dynamic content loading is promoted to a higher level, and performance is enhanced

<> index.html > html > body > script > loadDoc > onreadystatechange

```

1   <!DOCTYPE html>
2   <html>
3   <body>
4
5   <div id="demo">
6     <h1>The XMLHttpRequest Object</h1>
7     <button type="button" onclick="loadDoc()">Change Content</button>
8   </div>
9
10  <script>
11    function loadDoc() {
12      var xhttp = new XMLHttpRequest();
13      xhttp.onreadystatechange = function() {
14        if (this.readyState == 4 && this.status == 200) {
15          document.getElementById("demo").innerHTML =
16            this.responseText;
17        }
18      };
19      xhttp.open("GET", "ajax_info.txt", true);
20      xhttp.send();
21    }
22  </script>
23
24  </body>
25  </html>
26

```



The screenshot shows a code editor interface with the following details:

- File structure: Welcome, index.html, ajax\_info.txt
- Content of ajax\_info.txt:
 

```

1 Hi
2 WEB APPLICATION SECURITY
3 
```

## The XMLHttpRequest Object

[Change Content](#)

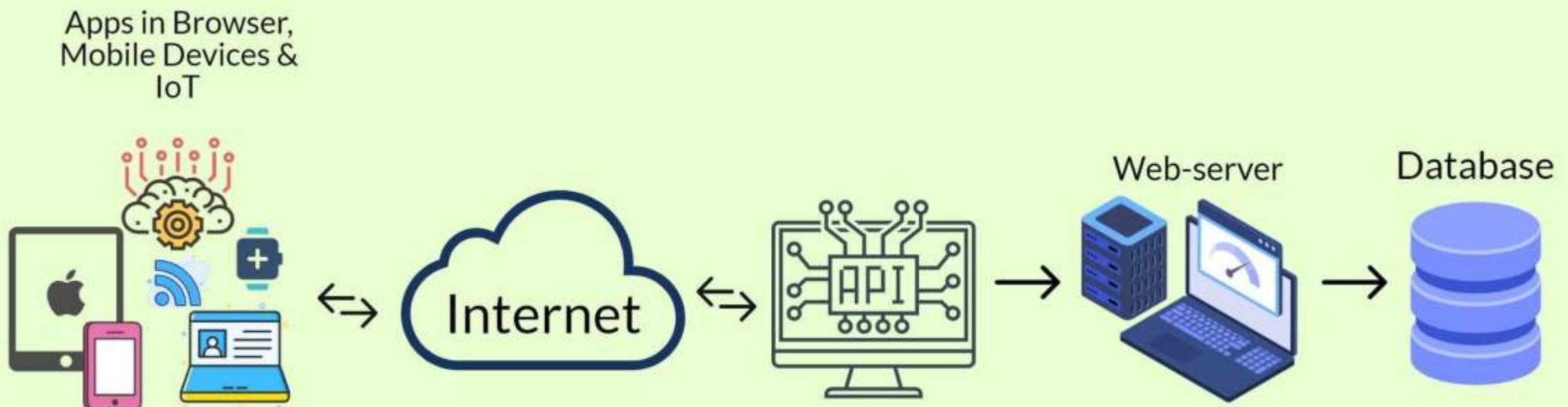
Hi WEB APPLICATION SECURITY



# Application Programming Interface

- API stands for Application Programming Interface. It is a specification intended to be used as an interface by software components to communicate with each other, including routines, data structure, object classes, and variables.
- In simple terms, it is a set of functions and procedures that allow applications to access data and features of other applications, services, and operating systems.
- An API specification including International Standard such as POSIX or Microsoft Windows API (vendor documentation), or the libraries of a programming language (Java API or Standard Template Library in C++).
- The API isn't the database or the server or even not a web-server, it's the bridge that governs the access point for the server to connect with the database.

# API



# Types of API

- There are four main types of APIs:

## **1. Public APIs**

Also known as Open API, which is publicly available for everyone to utilize and there is no restriction to access these types of API.

## **2. Partner APIs**

It is custom-designed by enterprises to allow business partners to reserve or buy specific items such as tickets. Need specific licenses to access these types of API.

## **3. Private APIs**

Also known as Internal APIs, used only for internal purposes, which means less known and often meant to be used inside the company. Many companies use this type of API for internal purposes among teams to improve their products and services.

## **4. Composite APIs**

This API contains different data and service APIs. It is used to improve the speed of the execution process and performance of the listeners in the web interfaces.

# Web- Browser- Server API

- API stands for **A**pplication **P**rogramming **I**nterface.
- A Web API is an application programming interface for the Web.
- A Browser API can extend the functionality of a web browser.
- A Server API can extend the functionality of a web server.

Javascript  
**Fetch**

# History of Fetch API and AJAX

- The **Fetch API** is a relatively new web browser feature but builds on older technologies such as AJAX.
- AJAX stands for Asynchronous JavaScript and XML. It was the first widely adopted technology to allow websites to send requests without needing to reload the entire page. Before AJAX, if you wanted to update something on your web page, you would need to reload the entire page - which was clunky and inefficient.
- AJAX allowed developers to make HTTP requests without needing a full-page refresh. This technology revolutionized web development, but it had its limitations.
- AJAX requests were limited to retrieving data from the same origin (domain/ subdomain) as the page requested.
- Fetch API had all of the power of AJAX but with no cross-origin security issues and added support for more HTTP methods like PUT and DELETE.

# Fetch API

- The Fetch API is a modern interface that allows you to make HTTP requests to servers from web browsers.
- If you have worked with XMLHttpRequest (XHR) object, the Fetch API can perform all the tasks as the XHR object does.
- The **fetch()** method is available in the global scope that instructs the web browsers to send a request to a URL.
- **Sending a Request**
  - The fetch() requires only one parameter which is the URL of the resource that you want to fetch:
    - `let response = fetch(url);`
    - The fetch() method returns a Promise so you can use the **then()** and **catch()** methods to handle it:

- `fetch(url)`
- `.then(function(response) {`
- `return response.json();`
- `})`
  
- `.catch(function(error) {`
- `console.log(error)`
- `});`

Here are some other useful methods that you can use with the Fetch API:

- *clone()*: creates a clone of the response
- *redirect()*: creates a new response but with a different URL
- *arrayBuffer()*: returns a promise that resolves with an array buffer
- *formData()*: returns a promise that resolves with a form data object
- *blob()*: resolves with a blob
- *text()*: resolves with a string
- *json()*: resolves the promise with JSON

# JS FETCH program1

## Indexfetch.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JSON Fetch and Display</title>
</head>
<body>
    <div id="data-container"></div>
    <script src="script.js"></script>
</body>
</html>
```

```
indexfetch.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>JSON Fetch and Display</title>
7  </head>
8  <body>
9      <div id="data-container"></div>
10     <script src="script.js"></script>
11 </body>
12 </html>
13
```

Script.js



## Script.js

```
const dataContainer = document.getElementById('data-container');

// Function to fetch JSON data using the Fetch API
function fetchData() {
    fetch('https://jsonplaceholder.typicode.com/users') // Replace with the actual URL of your JSON data
        .then(response => response.json())
        .then(data => displayData(data))
        .catch(error => console.error('Error fetching data:', error));
}

// Function to display the fetched JSON data
function displayData(data) {
    // Clear previous data
    dataContainer.innerHTML = '';

    // Display each item in the data
    data.forEach(item => {
        const itemDiv = document.createElement('div');
        itemDiv.innerHTML = `
            <p>ID: ${item.id}</p>
            <p>Name: ${item.name}</p>
            <p>Email: ${item.email}</p>
            <hr>
        `;
        dataContainer.appendChild(itemDiv);
    });
}

// Fetch data when the page loads
fetchData();
```

Source of json file

ID: 1	Name: Leanne Graham	Email: Leanne.Graham@reqres.in
ID: 2	Name: Ervin Howell	Email: Ervin.Howell@reqres.in
ID: 3	Name: Clemence Boothe	Email: Clemence.Boothe@reqres.in
ID: 4	Name: Patricia Lebsack	Email: Patricia.Lebsack@reqres.in
ID: 5	Name: Charley Dietrich	Email: Charley.Dietrich@reqres.in



# **XML**

## **Extensible Markup Language**

# History of XML

- The World Wide Web Consortium (W3C) is an international consortium where Member organizations, a full-time staff, and the public work together to develop Web standards
- Tim Berners-Lee and others created W3C (1994)
- Berners-Lee, who invented the World Wide Web in 1989.
- In 1970 IBM Introduced SGML
- SGML: Standard Generalized Markup Language
- SGML is a semantic and structural language for text documents.
- SGML is complicated.
- XML Working Group is formed under W3C in 1996.
- In 1998 W3C introduced XML 1.0
- Extensible Markup Language (XML) is a subset of SGML

- XML stands for **Extensible Markup Language**.
  - It is a text-based markup language derived from Standard Generalized Markup Language (SGML).
- XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.
- XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

# Characteristics of XML

- There are three important characteristics of XML that make it useful in a variety of systems and solutions –
  - **XML is extensible** – XML allows you to create your own self-descriptive tags, or language, that suits your application.
  - **XML carries the data, does not present it** – XML allows you to store the data irrespective of how it will be presented.
  - **XML is a public standard** – XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

# XML Usage

- A short list of XML usage says it all –
  - XML can work behind the scene to simplify the creation of HTML documents for large web sites.
  - XML can be used to exchange the information between organizations and systems.
  - XML can be used for offloading and reloading of databases.
  - XML can be used to store and arrange the data, which can customize your data handling needs.
  - XML can easily be merged with style sheets to create almost any desired output.
  - Virtually, any type of data can be expressed as an XML document.

# Is XML a Programming Language?

- A programming language consists of grammar rules and its own vocabulary which is used to create computer programs.
- These programs instruct the computer to perform specific tasks.
- XML does not qualify to be a programming language as it does not perform any computation or algorithms.
- It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

# The Difference Between XML and HTML

- XML and HTML were designed with different goals:
  - XML was designed to carry data - with focus on what data is
  - HTML was designed to display data - with focus on how data looks
  - XML tags are not predefined like HTML tags are
- **XML Does Not Use Predefined Tags**
  - The XML language has no predefined tags.
  - The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
  - HTML works with predefined tags like <p>, <h1>, <table>, etc.
  - With XML, the author must define both the tags and the document structure.
- **XML Separates Data from Presentation**
- **XML Separates Data from HTML**

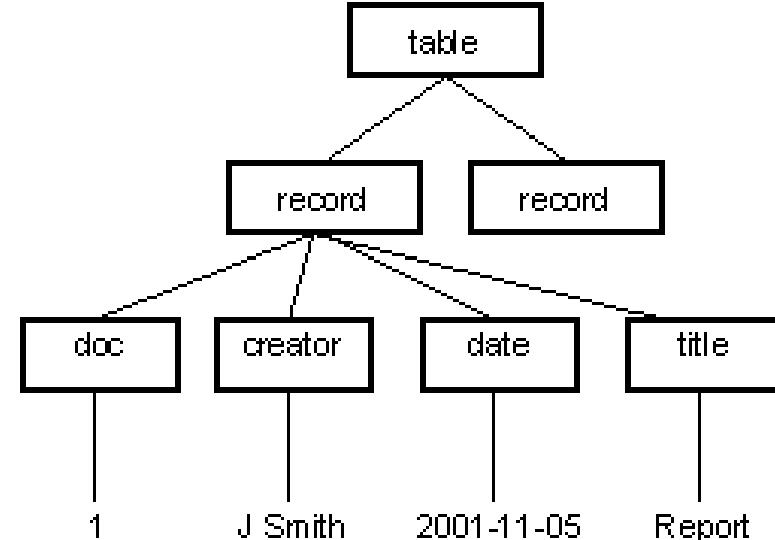
# XML Tree structure

Doc	Creator	Date	Title
1	J Smith	2001-11-05	Report

```

<table>
  <record>
    <doc>1</doc>
    <creator>J Smith</text>
    <date>2001-11-05</date>
    <title>Report</title>
  </record>
</table>

```



# Simple XML example : Bookstore

```

<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>

    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J. K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>

    <book category="web">
        <title lang="en">XQuery Kick Start</title>
        <author>James McGovern</author>
        <author>Per Bothner</author>
        <author>Kurt Cagle</author>
        <author>James Linn</author>
        <author>Vaidyanathan Nagarajan</author>
        <year>2003</year>
        <price>49.99</price>
    </book>

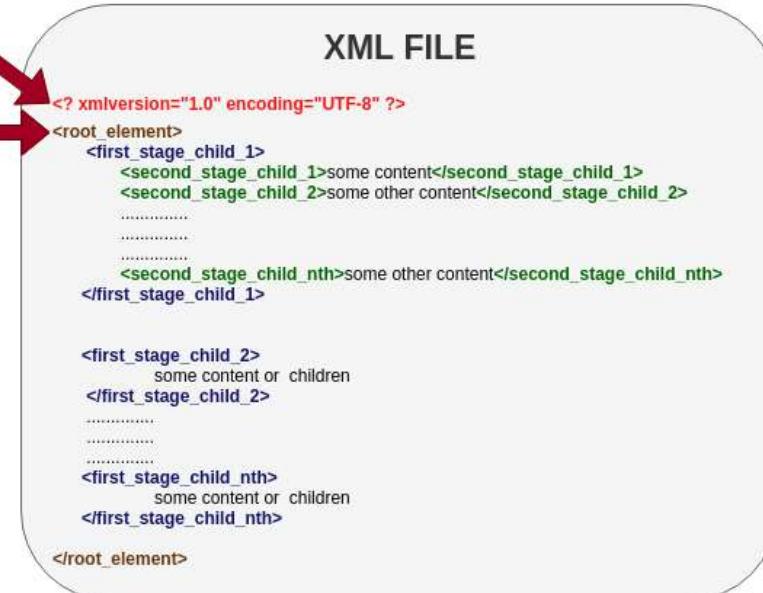
    <book category="web" cover="paperback">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>

</bookstore>

```

Prolog is mandatory

root element must  
be defined



XML documents must contain one **root element** that is the **parent** of all other elements:  
In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:  
XML tags are case sensitive. The tag `<Letter>` is different from the tag `<letter>`.

# Syntax

- XML Attribute Values Must Always be Quoted
  - XML elements can have attributes in name/value pairs just like in HTML.
  - In XML, the attribute values must always be quoted:
    - <note date="12/11/2007">
    - <to>Tove</to>
    - <from>Jani</from>
    - </note>

# Entity References

- Some characters have a special meaning in XML.
- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
- This will generate an XML error:
- <message>salary < 1000</message> -----→ this is an error

- To avoid this error, replace the "<" character
- <message>salary &lt; 1000</message>

There are 5 pre-defined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

# Well Formed XML

- White-space is Preserved in XML
  - XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):
- Well Formed XML
- XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

An XML document is said to be **well-formed** if it adheres to the following rules –

- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

# Example of a well-formed XML document –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address
[
    <!ELEMENT address (name,company,phone)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
]>

<address>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</address>
```

# XML Attributes

- XML elements can have attributes, just like HTML.
- Attributes are designed to contain data related to a specific element
- XML Elements vs. Attributes
  - <person gender="female">  
    <firstname>Anna</firstname>  
    <lastname>Smith</lastname>  
  </person>
  - ..... BOTH provide same info
  - <person>  
    <gender>female</gender>  
    <firstname>Anna</firstname>  
    <lastname>Smith</lastname>  
  </person>

# XML Namespaces

- Name Conflicts
- In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

```
<furniture>
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>

<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
</furniture>
```

If these XML fragments were added together, there would be a name conflict. Both contain a `<table>` element, but the elements have different content and meaning.  
A user or an XML application will not know how to handle these differences.

# Solving the Name Conflict Using a Prefix

- In the example , there will be no conflict because the two <table> elements have different names.

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

# XML Namespaces - The xmlns Attribute

- When using prefixes in XML, a **namespace** for the prefix must be defined.
- The namespace can be defined by an **xmlns** attribute in the start tag of an element.
- The namespace declaration has the following syntax.  
**xmlns:prefix="URI"**.

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

# XML Parser

- All major browsers have a built-in XML parser to access and manipulate XML.
- The [XML DOM \(Document Object Model\)](#) defines the properties and methods for accessing and editing XML.
- However, before an XML document can be accessed, it must be loaded into an XML DOM object.
- All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

# Parsing a Text String

```
<html>
<body>

<p id="demo"></p>

<script>
var text, parser, xmlDoc;

text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text, "text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>

</body>
</html>
```

Text string is defined

An XML DOM parser is created

The parser creates a new XML DOM object using the text string:

# XSLT

## eXtensible Stylesheet Language

[XSLT Introduction \(w3schools.com\)](https://www.w3schools.com/xsl/)

# XQuery

[XQuery Tutorial \(w3schools.com\)](https://www.w3schools.com/xquery/)

# XML Schema Definition (XSD)

[XML Schema Tutorial \(w3schools.com\)](https://www.w3schools.com/xml/xml_schema.asp)

# XML SECURITY

# XML Security

- **Malformed XML Documents:** vulnerabilities using not well formed documents.
- **Invalid XML Documents:** vulnerabilities using documents that do not have the expected structure.

# Malformed XML Documents

- The W3C XML specification defines a set of principles that XML documents must follow to be considered well formed.
- When a document violates any of these principles, it must be considered a fatal error and the data it contains is considered malformed.
- Multiple tactics will cause a malformed document: removing an ending tag, rearranging the order of elements into a nonsensical structure, introducing forbidden characters, and so on.
- The XML parser should stop execution once detecting a fatal error.
- The document should not undergo any additional processing, and the application should display an error message.

# Attacks on malformed Document

- A malformed document may affect the consumption of Central Processing Unit (CPU) resources.
- In certain scenarios, the amount of time required to process malformed documents may be greater than that required for well-formed documents.
- When this happens, an attacker may exploit an **asymmetric resource consumption attack** to take advantage of the greater processing time to cause a Denial of Service (DoS).
- A malformed document may affect the consumption of Central Processing Unit (CPU) resources. In certain scenarios, the amount of time required to process malformed documents may be greater than that required for well-formed documents.
- When this happens, an attacker may exploit an asymmetric resource consumption attack to take advantage of the greater processing time to cause a Denial of Service (DoS).

# Billion Laughs attack

- The [Billion Laughs](#) attack – also known as exponential entity expansion – uses multiple levels of nested entities.
- Each entity refers to another entity several times, and the final entity definition contains a small string.
- The exponential expansion results in several gigabytes of text and consumes lots of memory and CPU time.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

# XInclude

- XML Inclusion is another way to load and include external files:
- ```
<root xmlns:xi="http://www.w3.org/2001/XInclude">
```
- ```
  <xi:include href="filename.txt" parse="text" />
```
- ```
</root>
```
- This feature should be disabled when XML files from an untrusted source are processed. Some Python XML libraries and libxml2 support XInclude but don't have an option to sandbox inclusion and limit it to allowed directories.

# Coercive Parsing

- A coercive attack in XML involves parsing deeply nested XML documents without their corresponding ending tags.
- The idea is to make the victim use up -and eventually deplete- the machine's resources and cause a denial of service on the target. Reports of a DoS attack in Firefox 3.67 included the use of 30,000 open XML elements without their corresponding ending tags.
- The number of tags being processed eventually caused a stack overflow. A simplified version of such a document would look like this:
  - <A1>
  - <A2>
  - <A3>
  - ...
  - <A30000>

# Document without Schema

- If there is no control on the document's structure, the application could also process different well-formed messages with unintended consequences.

# Jumbo Payloads

- Sending an XML document of 1GB requires only a second of server processing and might not be worth consideration as an attack.
- Instead, an attacker would look for a way to minimize the CPU and traffic used to generate this type of attack, compared to the overall amount of server CPU or traffic used to handle the requests.
- **Traditional Jumbo Payloads**
  - There are two primary methods to make a document larger than normal:
    - Depth attack: using a huge number of elements, element names, and/or element values.
    - Width attack: using a huge number of attributes, attribute names, and/or attribute values.
  - In most cases, the overall result will be a huge document. This is a short example of what this looks like:
    - <SOAPENV:ENVELOPE XMLNS:SOAPENV="HTTP://SCHEMAS.XMLSOAP.ORG/SOAP/ENVELOPE/"  
  XMLNS:EXT="HTTP://COM/IBM/WAS/WSSAMPLE/SEI/ECHO/B2B/EXTERNAL">
    - <SOAPENV:HEADER LARGENAME1="LARGEVALUE"  
  LARGENAME2="LARGEVALUE2"
    - LARGENAME3="LARGEVALUE3" ...>

# Schema Poisoning

- When an attacker is capable of introducing modifications to a schema, there could be multiple high-risk consequences.
- In particular, the effect of these consequences will be more dangerous if the schemas are using DTD (e.g., file retrieval, denial of service).
- An attacker could exploit this type of vulnerability in numerous scenarios, always depending on the location of the schema.

# XML Security standards

# XML Security standards

- XML Digital Signature for integrity and signing solutions,
- XML Encryption for confidentiality,
- XML Key Management (XKMS) for public key registration, location and validation,
- Security Assertion Markup Language (SAML) for conveying authentication, authorization and attribute assertions,
- XML Access Control Markup Language (XACML) for defining access control rules, and
- Platform for Privacy Preferences (P3P) for defining privacy policies and preferences.
- Major use cases include securing
- Web Services (WS-Security) and
- Digital Rights Management (eXtensible Rights Markup Language 2.0 – XrML).

- Many XML languages have already been defined, including
- XHTML for creating web pages,
- DocBook for creating technical documentation,
- RSS for content distribution (syndication),
- RDF for representing information,
- MathML for mathematics markup,
- BRML for business reports, and many others.



{JSON}

# JSON

- In the early 2000s, JSON was initially specified by Douglas Crockford. In 2013, JSON was standardized as ECMA-404, and RCF 8259 was published in 2017.
- JSON is an open standard for exchanging data on the web
- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight data-interchange format
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers
- JSON is language independent \*
- JSON supports array, object, string, number and values
- The JSON file must be save with **.json** extension.

# first.json

```
1.{"employees":  
2.  {"name":"Sonoo", "email":"sonoojaiswal1987@gmail.com"},  
3.  {"name":"Rahul", "email":"rahul32@gmail.com"},  
4.  {"name":"John", "email":"john32bob@gmail.com"}  
5.]}
```

- In JSON, data is represented in key-value pairs, and curly braces hold objects, where a colon is followed after each name.
- The comma is used to separate key-value pairs.
- Square brackets are used to hold arrays, where each value is comma-separated.

# Why do we use JSON?

- Since JSON is an easy-to-use, lightweight language data interchange format in comparison to other available options, it can be used for API integration. Following are the advantages of JSON:
  - **Less Verbose:** In contrast to XML, JSON follows a compact style to improve its users' readability. While working with a complex system, JSON tends to make substantial enhancements.
  - **Faster:** The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files. However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.
  - **Readable:** The JSON structure is easily readable and straightforward. Regardless of the programming language that you are using, you can easily map the domain objects.
  - **Structured Data:** In JSON, a map data structure is used, whereas XML follows a tree structure. The key-value pairs limit the task but facilitate the predictive and easily understandable model.

# JSON Data Types

Data Type	Description	Example
String	A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters.	"student", "name", "1234", "Ver_1"
Number	Number represents the numeric characters.	121, 899
Boolean	It can be either True or False.	true
Null	It is an empty value.	

# JSON Objects

- In JSON, objects refer to dictionaries, which are enclosed in curly brackets, i.e., { }.
- These objects are written in key/value pairs, where the key has to be a string and values have to be a valid JSON data type such as string, number, object, Boolean or null.
- Here the key and values are separated by a colon, and a comma separates each key/value pair.
- For example:

```
{"name": "Jack", "employeeid": 001, "present": false}
```

# JSON Arrays

- In JSON, arrays can be understood as a list of objects, which are mainly enclosed in square brackets [ ].
- An array value can be a string, number, object, array, boolean or null.

```
[  
  {"PizzaName" : "Country Feast",  
   "Base" : "Cheese burst",  
   "Toppings" : ["Jalepenos", "Black Olives", "Extra cheese", "Sausages", "Cherry tomatoes"],  
   "Spicy" : "yes",  
   "Veg" : "yes"  
  },  
  
  {  
    "PizzaName" : "Veggie Paradise",  
    "Base" : "Thin crust",  
    "Toppings" : ["Jalepenos", "Black Olives", "Grilled Mushrooms", "Onions", "Cherry tomatoes"],  
    "Spicy" : "yes",  
    "Veg" : "yes"  
  }  
]
```

In the above example, the object "Pizza" is an array. It contains five objects, i.e., PizzaName, Base, Toppings, Spicy, and Veg.

# JSON Vs XML

JSON is easy to learn.	XML is quite more complex to learn than JSON.
It is simple to read and write.	It is more complex to read and write than JSON.
It is data-oriented.	It is document-oriented.
JSON is less secure in comparison to XML.	XML is highly secured.
It doesn't provide display capabilities.	It provides the display capability because it is a markup language.
It supports the array.	It doesn't support the array
Example :[ { "name" : "Peter", "employed id" : "E231", "present" : true, "numberofdayspresent" : 29 }, { "name" : "Jhon", "employed id" : "E331", "present" : true, "numberofdayspresent" : 27 } ]	Example :<name> <name>Peter</name></name>

```
{"employees": [
    {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
    {"name":"Bob", "email":"bob32@gmail.com"},
    {"name":"Jai", "email":"jai87@gmail.com"}
]}
```

```
<employees>
  <employee>
    <name>Shyam</name>
    <email>shyamjaiswal@gmail.com</email>
  </employee>
  <employee>
    <name>Bob</name>
    <email>bob32@gmail.com</email>
  </employee>
  <employee>
    <name>Jai</name>
    <email>jai87@gmail.com</email>
  </employee>
</employees>
```

# What is JSON injection?

- JSON injection attacks happen when unsanitized JSON data containing a malicious payload is accepted and parsed by a web application or browser.
- Server-side JSON injection attacks are possible if input data is not sanitized by the server and is written directly to a JSON stream. Client-side JSON injection attacks are possible if incoming JSON data is not sanitized and is parsed directly using the JavaScript eval function.
- **How to mitigate JSON injection attacks?**
- You can completely eliminate the risk of client-side JSON injections by enforcing [Content Security Policy](#), which [by default prevents the use of eval](#). This forces the developers to use the safer `JSON.parse` method instead.

# What is JSON hijacking?

- JSON hijacking happens when a malicious web page causes the victim's web browser to retrieve JSON data from a targeted web application. The web browser then passes the JSON data to a web server controlled by the attacker.
- While JSON hijacking (a subset of cross-site script inclusion – XSS) also involves the JSON format, it is a slightly different attack, in some ways similar to [cross-site request forgery \(CSRF\)](#). Attackers can use JSON hijacking to intercept JSON data sent from a web server to a web application. A typical JSON hijacking attack might look like this:
  - 1.The attacker creates a malicious website containing a *script* tag that references a JSON data URL of the web application under attack and includes code to hijack the JSON data.
  - 2.A user logged into the targeted web application is tricked into visiting the malicious website (usually using social engineering).
  - 3.Since the [same-origin policy \(SOP\)](#) allows JavaScript from any website to be included and executed in the context of any other site, the user's web browser loads the JSON data in the context of the malicious site.
  - 4.The malicious website hijacks the JSON data.

# WEB APP SECURITY



```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => {
                  console.log(value)
                }}
              </ProductConsumer>
            </div>
          </div>
        </div>
      <React.Fragment>
```

## Course Outcomes

**CO1** Apply client-side web development to design interactive front-end web user interfaces.

**CO2** Use server-side web application concepts to develop back-end web server application

**CO3** Identify and mitigate various client-side web application security vulnerabilities

**CO4** Identify and mitigate various server-side web application security vulnerabilities

# Syllabus

- Web application development – Introduction - Architecture – Client-side technologies and frameworks – HTML – CSS – Javascript - Ajax/Fetch - Data interchange formats – XML, JSON. Server-side scripting and technologies - development – technologies - Handling client requests – Database connectivity – Sessions – Cookies.
- Web application vulnerabilities – Client-side Vulnerabilities - Cross Site Scripting (XSS) - Cross Site Request Forgery (CSRF) - Cross-origin resource sharing (CORS) - Clickjacking. Server-side Vulnerabilities - SQL injection - OS command injection - Directory traversal - Authentication - Server-side request forgery (SSRF)
- **Textbook(s)**

# Textbook and Reference books

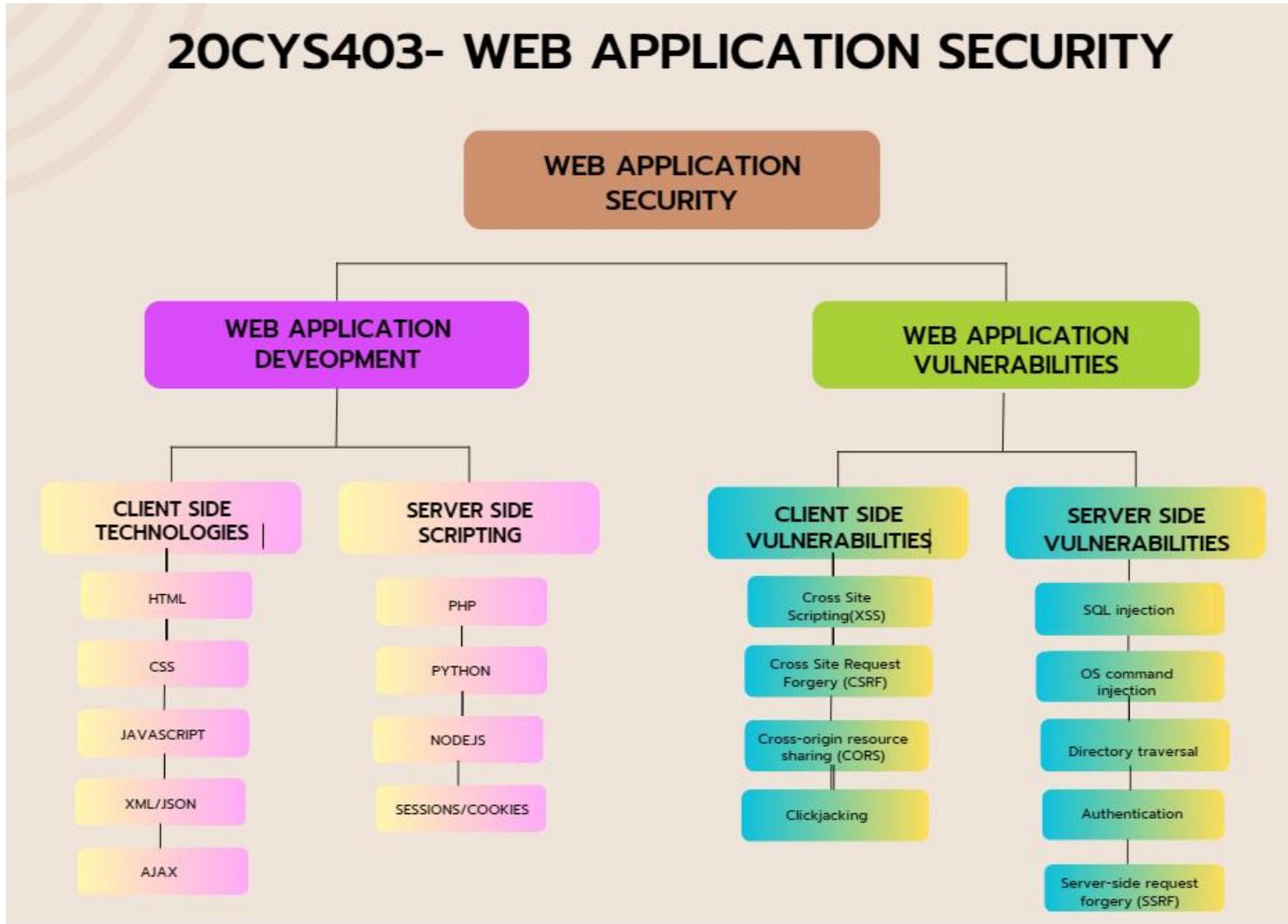
- **Textbook(s)**

1. Robin Nixon, Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites, Fifth Edition, O'Reilly Media, Inc.; 2018.
2. Dafydd Stuttard, and Marcus Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition, John Wiley & Sons; 2011

- **Reference(s)**

1. John Paul Mueller, Security for Web Developers: Using Javascript, Html, and CSS, O'Reilly, 2015
2. Andrew Hoffman. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Shroff Publishers & Distributors Pvt. Ltd, 2020.
3. Richa Gupta, Hands-on Penetration Testing for Web Applications, BPB Publications (29 March 2021)
4. Ivan Ristic, Bulletproof TLS and PKI, Second Edition: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Feisty Duck Ltd; 2nd edition, 2022
5. <https://web.stanford.edu/class/cs253/>
6. <https://nptel.ac.in/courses/128106006>
7. <https://www.rapid7.com/fundamentals/web-application-security/>
8. <https://owasp.org/>
9. <https://www.udemy.com/course/web-application-security/>

# Concept Map

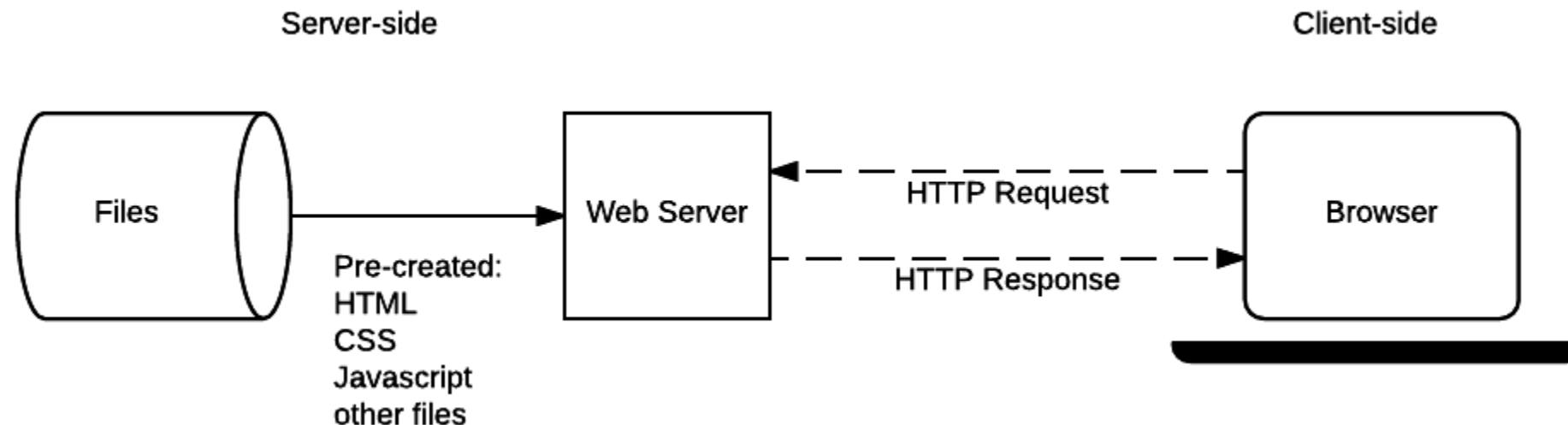


# What is server-side website programming?

- Web browsers communicate with [web servers](#) using the **HyperText Transfer Protocol (HTTP)**. When you click a link on a web page, submit a form, or run a search, an **HTTP request** is sent from your browser to the target server.
- The request includes a URL identifying the affected resource, a method that defines the required action (for example to get, delete, or post the resource), and may include additional information encoded in URL parameters (the field-value pairs sent via a [query string](#)), as POST data (data sent by the [HTTP POST method](#)), or in associated [cookies](#).
- Web servers wait for client request messages, process them when they arrive, and reply to the web browser with an **HTTP response** message. The response contains a status line indicating whether or not the request succeeded (e.g. "HTTP/1.1 200 OK" for success).
- The body of a successful response to a request would contain the requested resource (e.g. a new HTML page, or an image), which could then be displayed by the web browser.

# Static sites

- a static site is one that returns the same hard-coded content from the server whenever a particular resource is requested).
- When a user wants to navigate to a page, the browser sends an HTTP "GET" request specifying its URL.

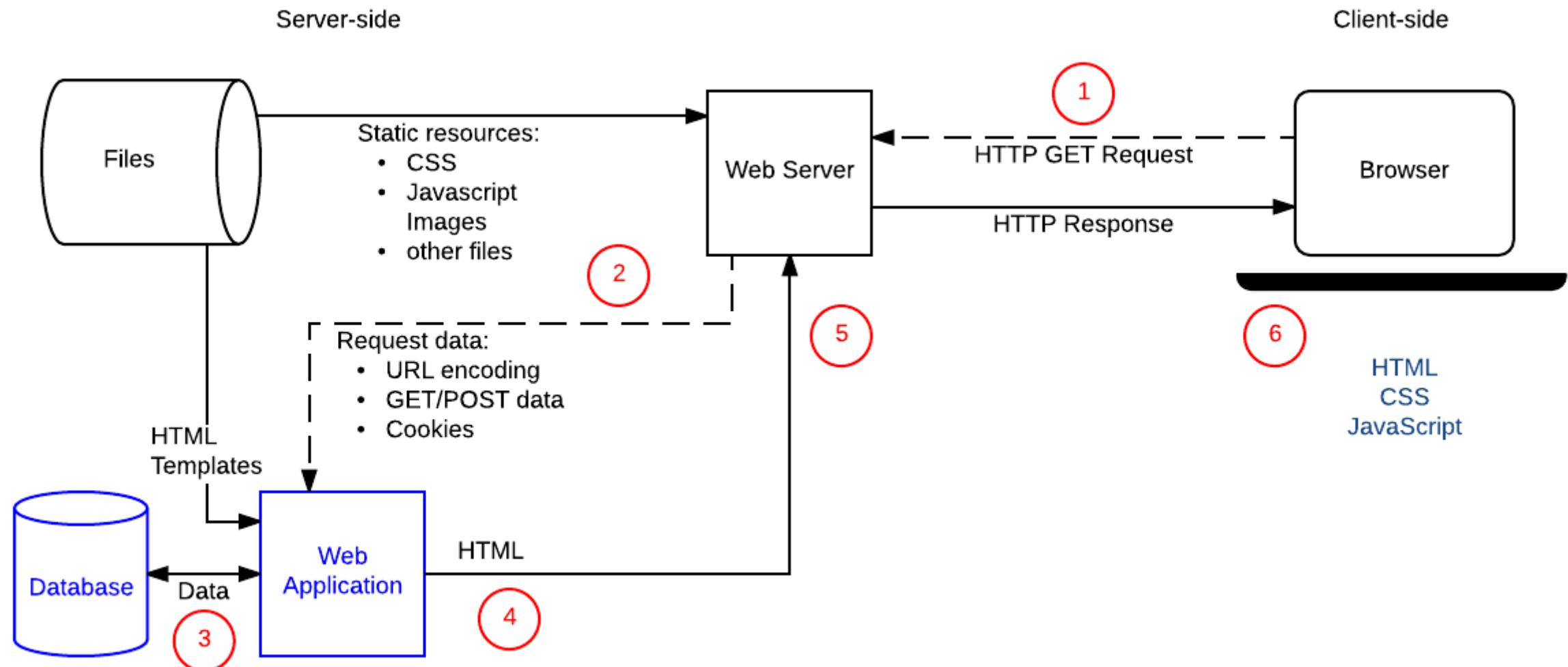


- The server retrieves the requested document from its file system and returns an HTTP response containing the document and a [success status](#) (usually 200 OK). If the file cannot be retrieved for some reason, an error status is returned (see [client error responses](#) and [server error responses](#)).

# Dynamic sites

- A dynamic website is one where some of the response content is generated *dynamically*, only when needed.
- On a dynamic website HTML pages are normally created by inserting data from a database into placeholders in HTML templates (this is a much more efficient way of storing large amounts of content than using static websites).
- A dynamic site can return different data for a URL based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g. sending notifications).
- Most of the code to support a dynamic website must run on the server. Creating this code is known as "**server-side programming**" (or sometimes "**back-end scripting**").

# Dynamic website



# Client-side code

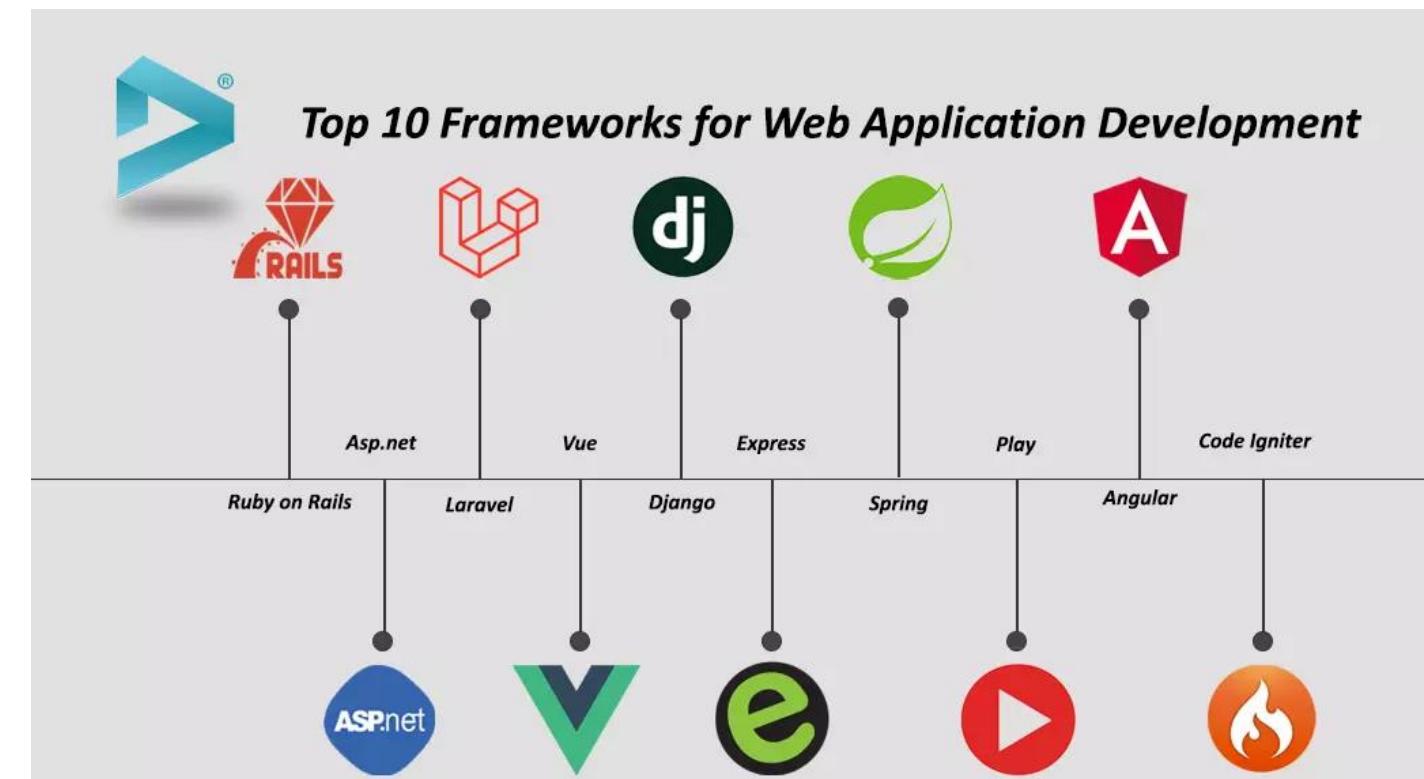
- is primarily concerned with improving the appearance and behavior of a rendered web page.
- This includes selecting and styling UI components, creating layouts, navigation, form validation, etc.
- Client-side code is written using [HTML](#), [CSS](#), and [JavaScript](#) — it is run inside a web browser and has little or no access to the underlying operating system

# Server-side code

- By contrast, server-side website programming mostly involves choosing *which content* is returned to the browser in response to requests.
- The server-side code handles tasks like validating submitted data and requests, using databases to store and retrieve data and sending the correct data to the client as required.
- By contrast, server-side website programming mostly involves choosing *which content* is returned to the browser in response to requests.
- The server-side code handles tasks like validating submitted data and requests, using databases to store and retrieve data and sending the correct data to the client as required.

# Web frameworks

- Web frameworks are collections of functions, objects, rules and other code constructs designed to solve common problems, speed up development, and simplify the different types of tasks faced in a particular domain.
- Build web services, web APIs, and other web resources



# Benefits of server-side programming

- Efficient storage and delivery of information
- Customized user experience
- Controlled access to content
- Store session/state information
- Notifications and communication
- Data analysis

# Models of how a program can generate a webpage

1. HTML with Server-Side Code Added
2. Server-Side Code Generating HTML
3. Same HTML using Client-Side JavaScript to Modify Webpage

# HTML with Server-Side Code Added

- With this approach a webpage on the server-side looks almost like a tradition HTML file, except in a few places, we ask the server to insert new information based on code executed on the server. ▪
- Here is a sample using the server-side language PHP.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Time of Day</title>
</head>
<body>
<h1>Time of Day</h1>
<?php
$now = new DateTime();
echo $now->format("h:i:sA"); ?>
?
</body>
</html>
```

When a user visits our webpage, the server will send the HTML in our file, but before sending it, it will execute the PHP code within the section.

- This approach to server-side programming works well if your webpage is mostly static, but you have a few items you want to add to it using server-side programming.

# Server-Side Code Generating HTML

- In this approach we generate a new HTML file from scratch. ▪ Here's an example using JavaScript for Node.js

```
var http = require('http');
var server = http.createServer(function(request, response)
{
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("Simple HTML Example");
  response.end("Hello World! ");
});

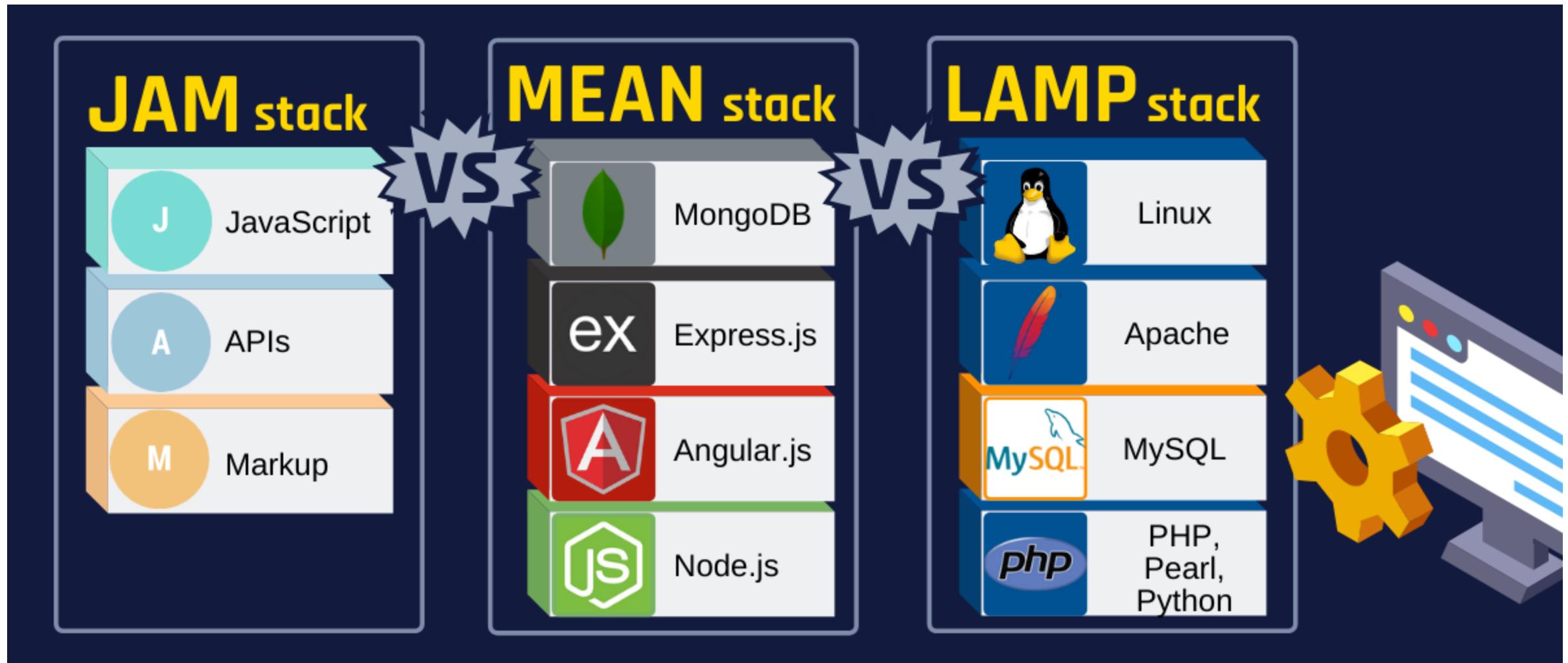
server.listen(8080);
```

This approach can be used to generate new HTML files, new Image files, or any other type of file.

# Same HTML using Client-Side JavaScript to Modify Webpage



- In this approach we always serve up a traditional HTML file. However, that file includes client-side JavaScript code, which pulls data from the server and modifies the webpage on the fly.
- This approach is particularly useful when trying to create a Single Page Application (SPA)
- An SPA is a website consisting of only a single webpage, where that webpage is dynamically modified in response to user interactions. • Google Mail is an example of an SPA.



 LAMP	 WAMP	 JAM	 MEAN	 MERN	 MEVN	 SERVERLESS	 Flutter
Linux	Window	Java Script	MangoDb	MangoDb	MangoDb	Dev-Ops Using	Flutter
Apatche	Apache	API	ExpressJs	ExpressJs	Express	Google Cloud	Dart Platform
MySQL	MySQL	Markup	AngularJs	ReactJs	Vue.js	AWS	
PHP/Python	PHP/Python		Node.js	Node.js	Node.js	Azure	

## **CLIENT SIDE SCRIPTING**

**HTML, CSS, and javascript are used.**

**The source code is visible to the user.**

**It does not provide security for data.**

**Its main function is to provide the requested output to the end user.**

**It usually depends on the browser and its version.**

**It runs on the user's computer.**

## **SERVER SIDE SCRIPTING**

**PHP, Python, Java, Ruby are used.**

**The source code is not visible to the user because its output of server-side is an HTML page.**

**It provides more security for data.**

**Its primary function is to manipulate and provide access to the respective database as per the request.**

**In this any server-side technology can be used and it does not depend on the client.**

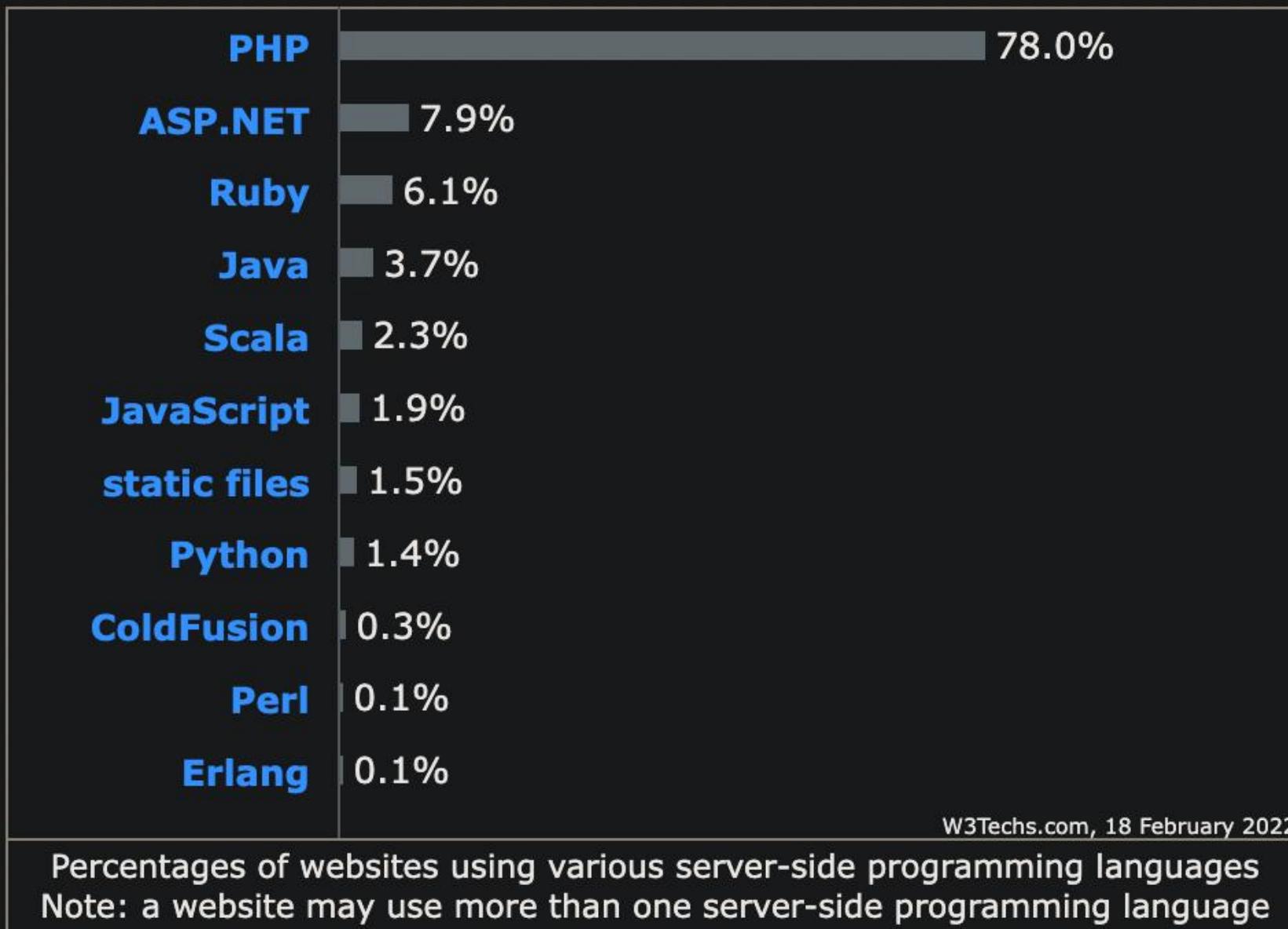
**It runs on the webserver.**

## Difference between client-side scripting vs. Server side scripting

Client Side Scripting	Server Side Scripting
The client-side environment used to run scripts is usually a browser.	The server-side environment that runs a scripting language is a web server.
The source code is transferred from the web server to the user's computer over the internet and run directly in the browser.	A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser.
Advantages to client-side scripting including faster response times, a more interactive application, and less overhead on the web server.	The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.
The Disadvantages of client-side scripting are that scripting languages require more time and effort, while the client's browser must support that scripting language.	The disadvantage of server-side processing is the page <u>postback</u> : it can introduce processing overhead that can decrease performance and force the user to wait for the page to be processed and recreated. Once the page is posted back to the server, the client must wait for the server to process the request and send the page back to the client.
Example <pre>&lt;script&gt; document.getElementById('hello').innerHTML = 'Hello'; &lt;/script&gt;</pre>	Example: <pre>&lt;h1 id="hello"&gt;&lt;?php echo 'Hello'; ?&gt;&lt;/h1&gt;</pre>

How to read the diagram:

PHP is used by 78% of all the websites whose server-side programming language we know.



# An overview of HTTP

# An overview of HTTP

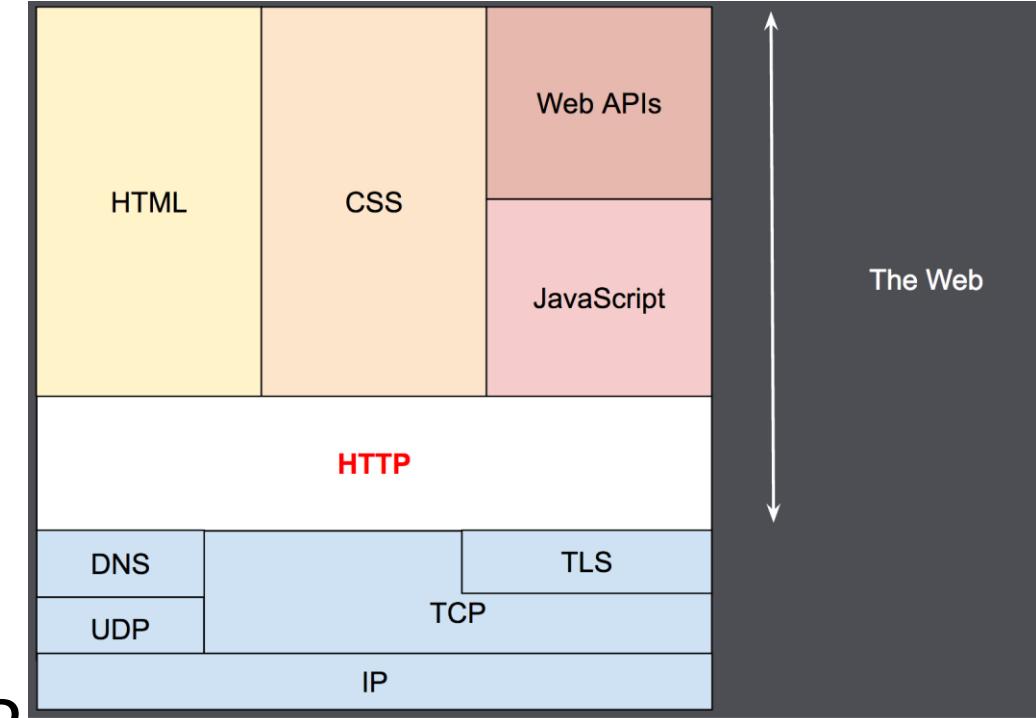
- It is the protocol that allows web servers and browsers to exchange data over the web.
- It is a request response protocol.
- It uses the reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

## The Basic Features of HTTP (Hyper Text Transfer Protocol):

- There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:
- **HTTP is media independent:** It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
- **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
- **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

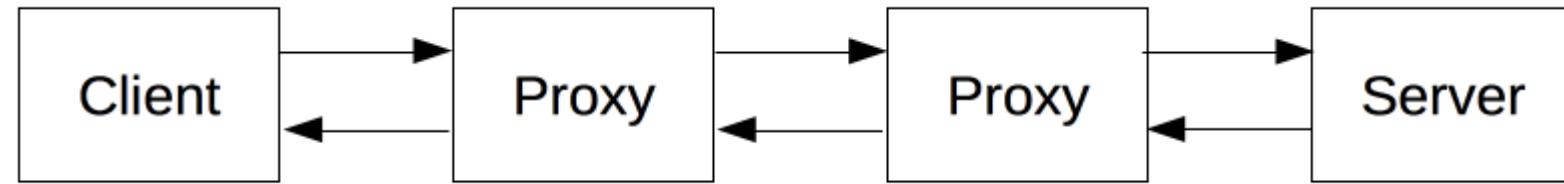
# HTTP in Application layer

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
- The messages sent by the client, usually a Web browser, are called *requests* and the messages sent by the server as an answer are called *responses*.
- Designed in the early 1990s, HTTP is an extensible protocol which has evolved over time.
- It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection, though any reliable transport protocol could theoretically be used.



# Components of HTTP-based systems

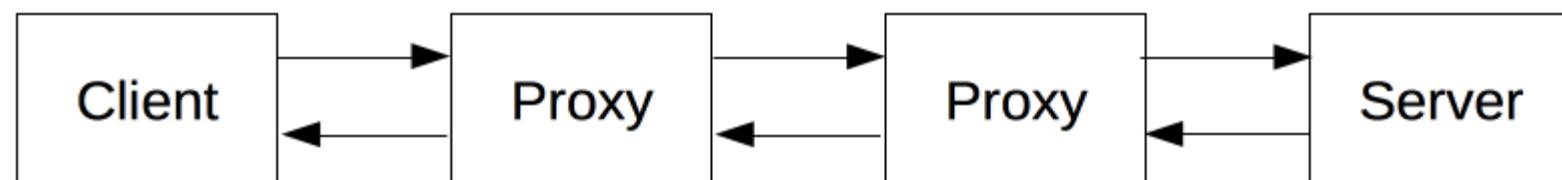
- Each individual request is sent to a server, which handles it and provides an answer, called the *response*.
- Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example



- In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more.
- Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer.
- Although important to diagnose network problems, the underlying layers are mostly irrelevant to the description of HTTP.

# Proxies

- Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches,
- Proxies may perform numerous functions:
  - caching (the cache can be public or private, like the browser cache)
  - filtering (like an antivirus scan or parental controls)
  - load balancing (to allow multiple servers to serve the different requests)
  - authentication (to control access to different resources)
  - logging (allowing the storage of historical information)



# HTTP flow

- When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

1. Open a TCP connection:

2. Send an HTTP message:

3. Read the response sent by the server,

4. Close or reuse the connection for further requests.

GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr

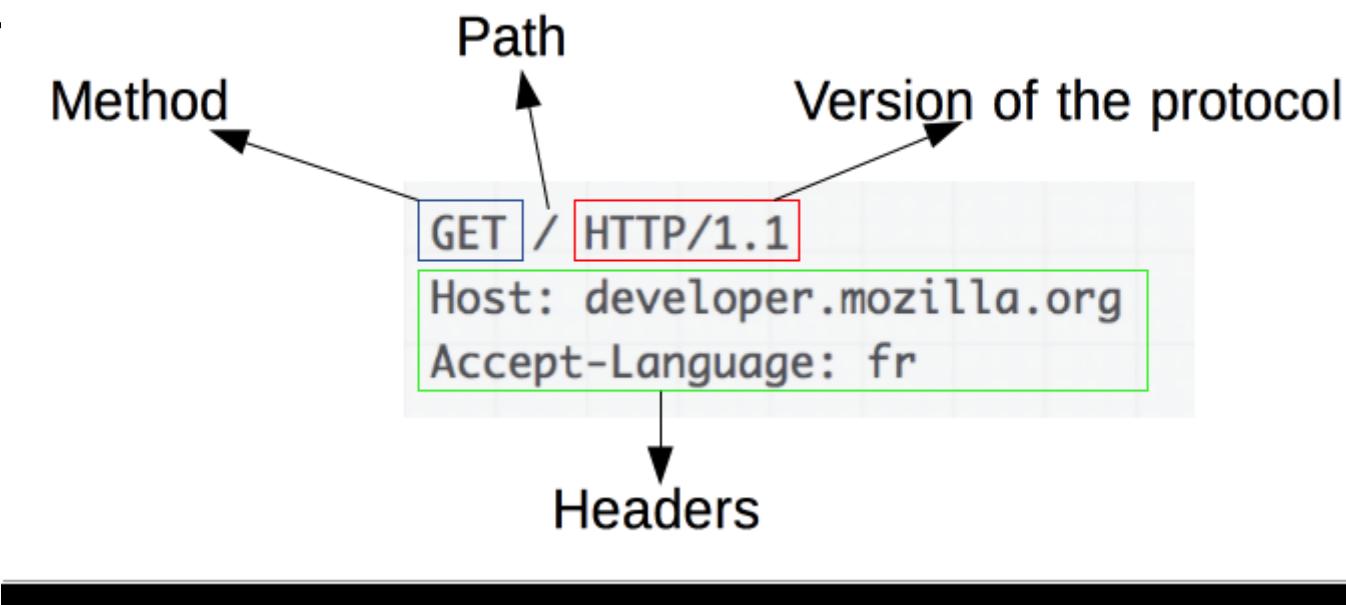
# HTTP request



HTTP Request	Description
<b>GET</b>	Asks to get the resource at the requested URL.
<b>POST</b>	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
<b>HEAD</b>	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
<b>TRACE</b>	Asks for the loopback of the request message, for testing or troubleshooting.
<b>PUT</b>	Says to put the enclosed info (the body) at the requested URL.
<b>DELETE</b>	Says to delete the resource at the requested URL.
<b>OPTIONS</b>	Asks for a list of the HTTP methods to which the thing at the request URL can respond

# HTTP Messages - Request

- In HTTP/2, these messages are embedded into a binary structure, a *frame*, allowing optimizations like compression of headers and multiplexing
- There are two types of HTTP messages, requests and responses, each with its own fc



# Get vs. Post

## Get vs. Post

In case of Get request, only limited amount of data can be sent because data is sent in header.

Get request is not secured because data is exposed in URL bar.

Get request can be bookmarked.

Get request is Idempotent . It means second request will be Ignored until response of first request is delivered

Get request is more efficient and used more than Post.

1 In case of post request, large amount of data can be sent because data is sent in body.

2 Post request is secured because data is not exposed in URL bar.

3 Post request cannot be bookmarked.

4 Post request is non-Idempotent.

5 Post request is less efficient and used less than get.



# GET vs POST

## The HTTP Method

Path to the source on Web Server      Parameters to the server      Protocol Version Browser supports  
GET /RegisterDao.jsp?user=ravi&pass=java HTTP/1.1

## The Request Headers

Host: www.javatpoint.com  
User-Agent: Mozilla/5.0  
Accept-text/xml,text/html,text/plain,image/jpeg  
Accept-Language: en-us,en  
Accept-Encoding: gzip,deflate  
Accept-Charset: ISO-8859-1,utf-8  
Keep-Alive: 300  
Connection: keep-alive

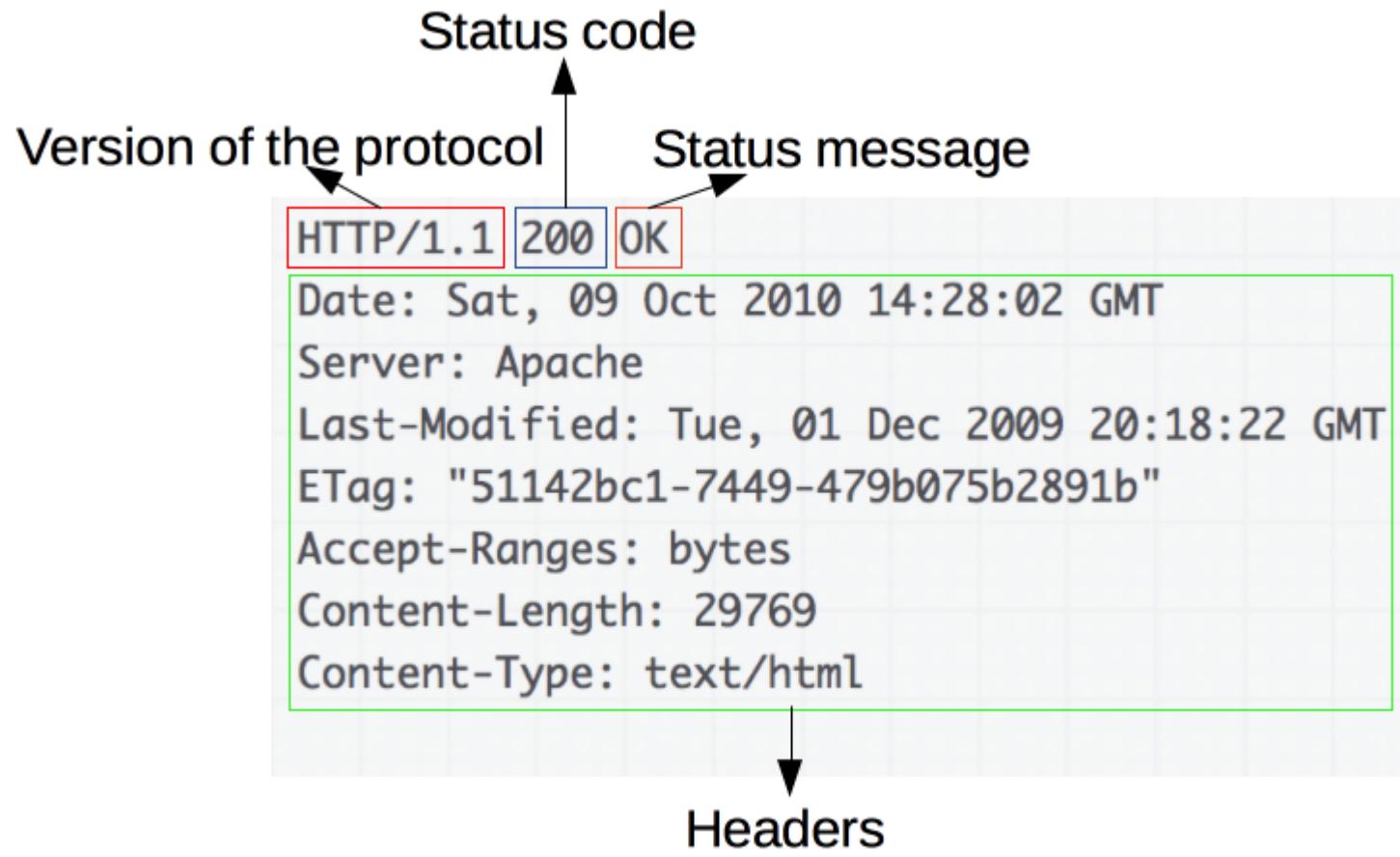
## The HTTP Method

Path to the source on Web Server      Protocol Version Browser supports  
Post /RegisterDao.jsp HTTP/1.1  
Host: www.javatpoint.com  
User-Agent: Mozilla/5.0  
Accept: text/xml,text/html,text/plain,image/jpeg  
Accept-Language: en-us,en  
Accept-Encoding: gzip,deflate  
Accept-Charset: ISO-8859-1,utf-8  
Keep-Alive:300  
Connection:keep-alive  
User=ravi&pass=java } Message body

# HTTP message - Response

Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was s
- A status message, a non-authoritative short
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched re



# APIs based on HTTP

- The most commonly used API based on HTTP is the XMLHttpRequest API, which can be used to exchange data between a user agent and a server. The modern Fetch API provides the same features with a more powerful and flexible feature set.



Dr.S.Udhayakumar  
Professor

- Server Side Scripting with PHP:
  - General Syntactic Characteristics
  - Primitives, Operations, and Expressions, Outputs
  - Control Statements
  - Arrays
  - Functions
  - Form handling, Form Processing and Business Logic
  - Connecting to Database
  - Dynamic Content

# What is PHP

- PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side.
- PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page.).
- PHP was created by **Rasmus Lerdorf** in **1994** but appeared in the market in 1995. **PHP 7.4.0** is the latest version of PHP, which was released on **28 November**. Some important points need to be noticed about PHP are as followed:



# About PHP

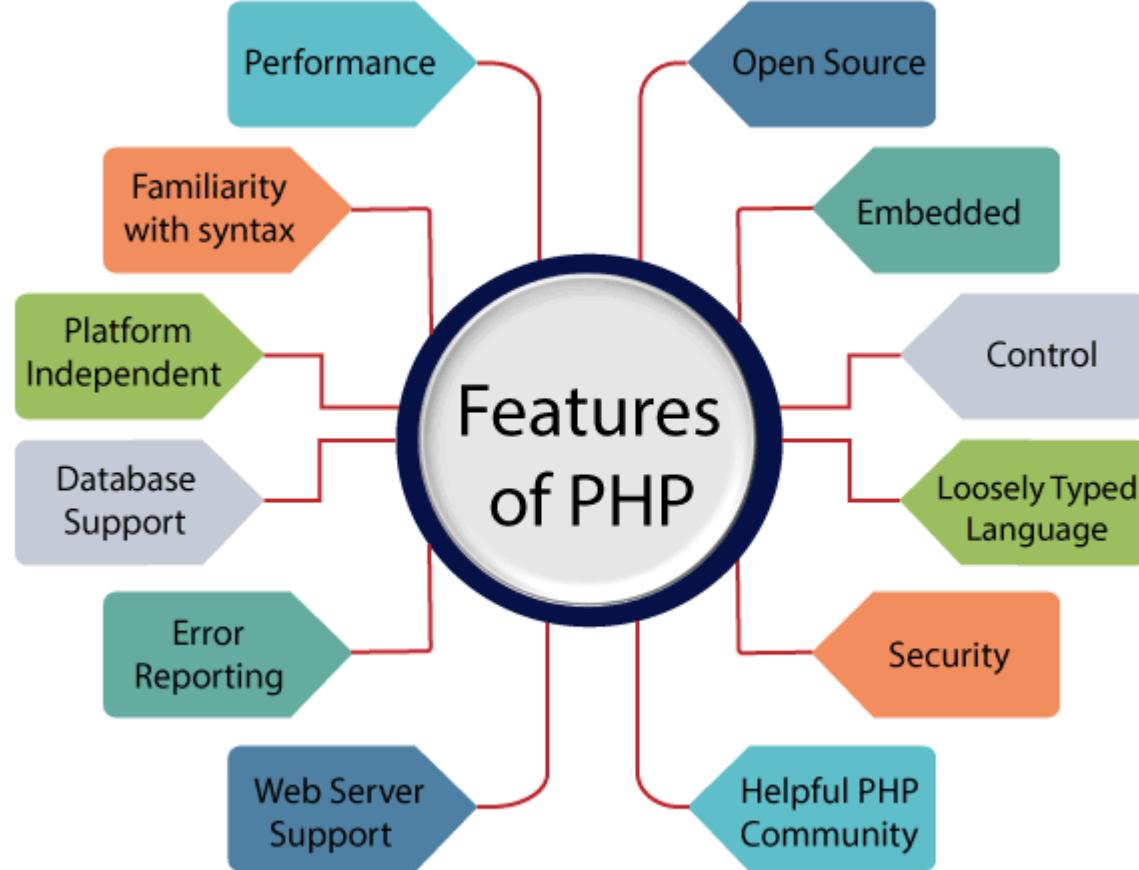
- PHP stands for Hypertext Preprocessor.
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.
- PHP is simple and easy to learn lang'



# Why use PHP

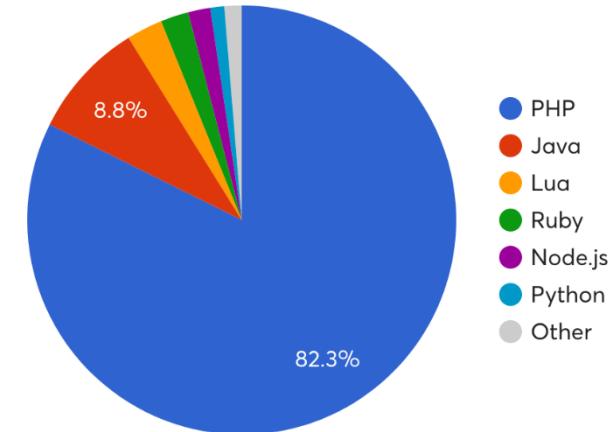
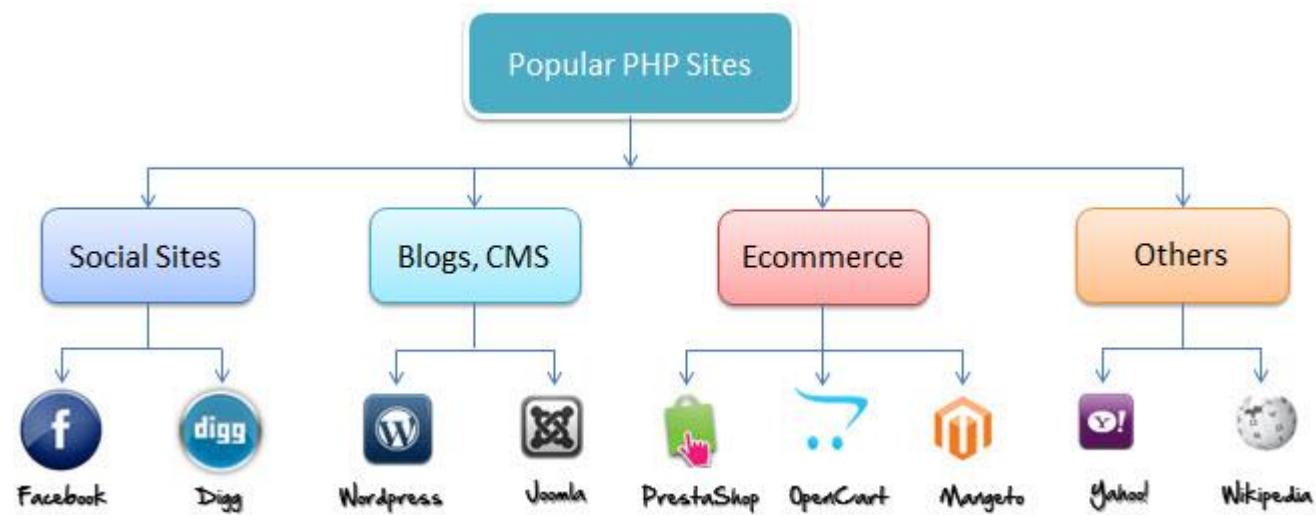
- PHP is a server-side scripting language, which is used to design the dynamic web applications with MySQL database.
  - It handles dynamic content, database as well as session tracking for the website.
  - You can create sessions in PHP.
  - It can access cookies variable and also set cookies.
  - It helps to encrypt the data and apply validation.
  - PHP supports several protocols such as HTTP, POP3, SNMP, LDAP, IMAP, and many more.
  - Using PHP language, you can control the user to access some pages of your website.
  - As PHP is easy to install and set up, this is the main reason why PHP is the best language to learn.
  - PHP can handle the forms, such as - collect the data from users using forms, save it into the database, and return useful information to the user. **For example - Registration form.**

# Feature of PHP



# What is PHP used for & Market share

- In terms of market share, there are over 20 million websites and application on the internet developed using PHP scripting language.



# How to run PHP

- **Install PHP**
  - To install PHP, we will suggest you to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems. There are many AMP options available in the market that are given below:
  - **WAMP** for Windows
  - **LAMP** for Linux
  - **MAMP** for Mac
  - **SAMP** for Solaris
  - **FAMP** for FreeBSD
  - **XAMPP** (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail, etc.

# Download

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl. Just download and start the installer. It's that easy.

## XAMPP for Windows 7.3.29, 7.4.21 & 8.0.8

Version	Checksum	Size
7.3.29 / PHP 7.3.29	<a href="#">What's Included?</a> <a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Download (64 bit)</a> 158 Mb
7.4.21 / PHP 7.4.21	<a href="#">What's Included?</a> <a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Download (64 bit)</a> 159 Mb
8.0.8 / PHP 8.0.8	<a href="#">What's Included?</a> <a href="#">md5</a> <a href="#">sha1</a>	<a href="#">Download (64 bit)</a> 160 Mb

[Requirements](#) [Add-ons](#) [More Downloads »](#)

Windows XP or 2003 are not supported. You can download a compatible version of XAMPP for these platforms [here](#).

## Documentation/FAQs

There is no real manual or handbook for XAMPP. We wrote the documentation in the form of FAQs. Have a burning question that's not answered here? Try the [Forums](#) or [Stack Overflow](#).

- [Linux FAQs](#)
- [Windows FAQs](#)
- [OS X FAQs](#)
- [OS X XAMPP-VM FAQs](#)

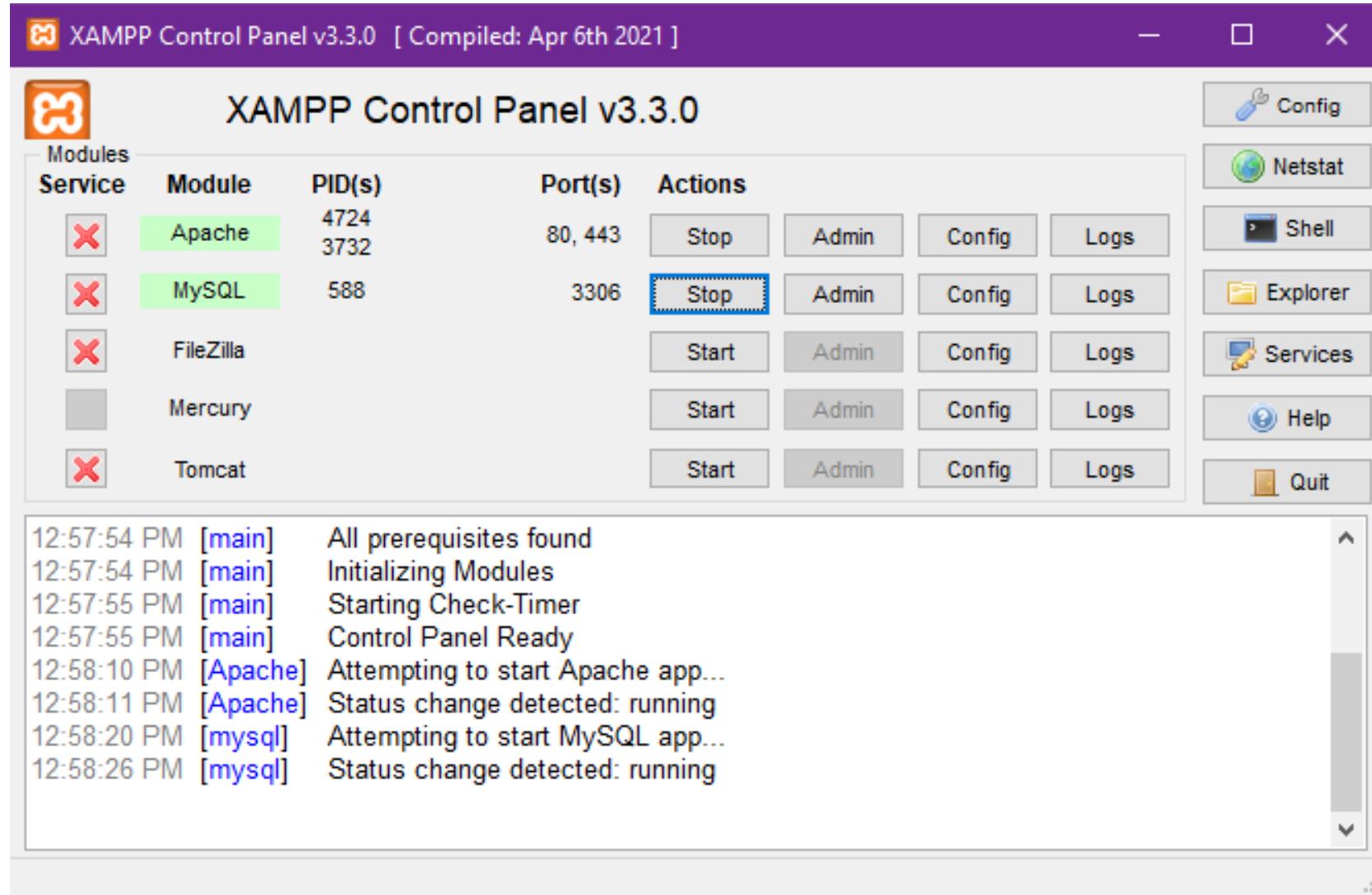
## Add-ons



Bitnami provides a free all-in-one tool

[Cookie Settings](#)

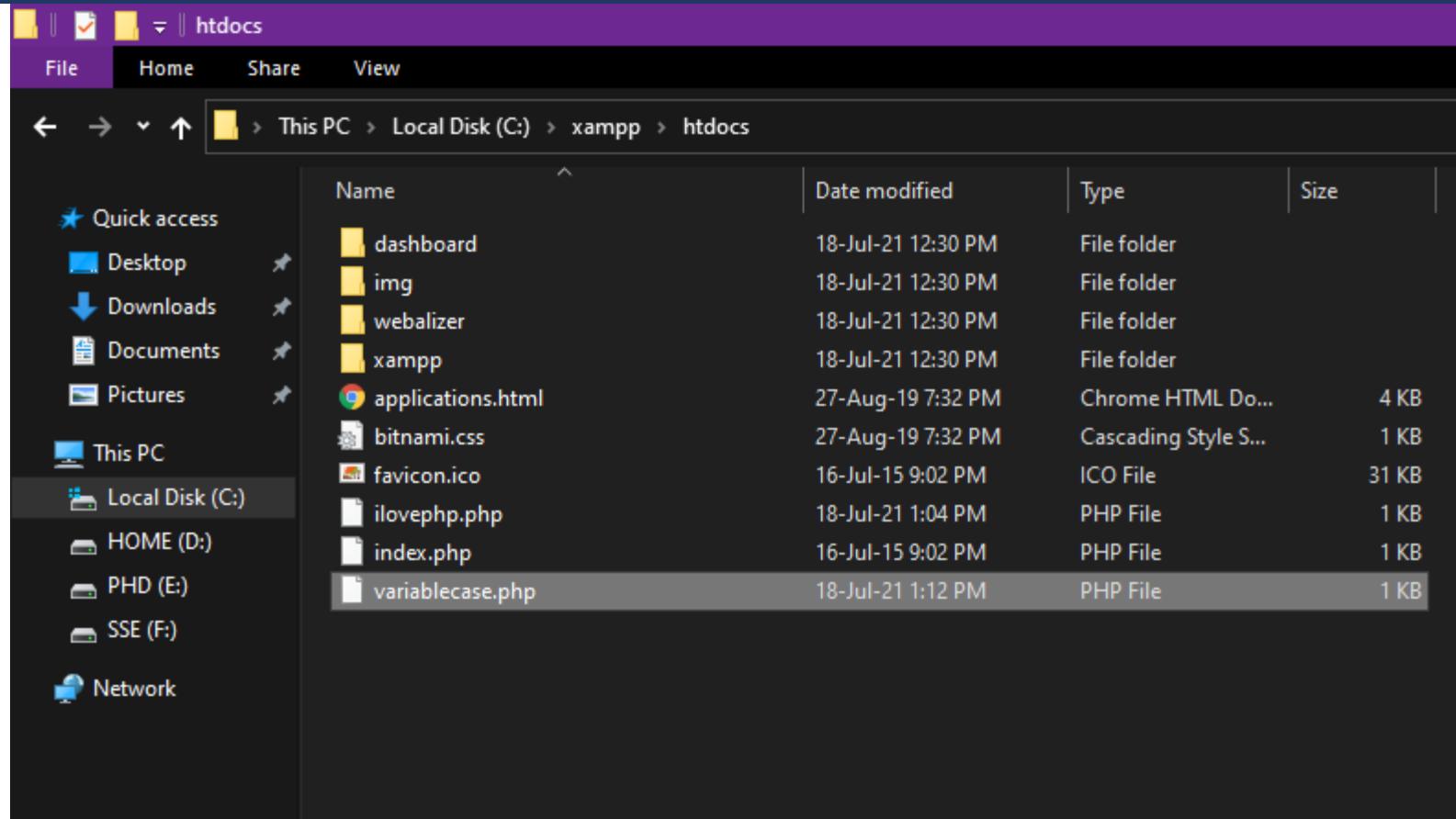
# Start the XAMPP control panel



# HTML + PHP or only PHP

- Generally, a PHP file contains HTML tags and some PHP scripting code. It is very easy to create a simple PHP example.
- To do so, create a file and write HTML tags + PHP code and save this file with .php extension.

# PHP code location

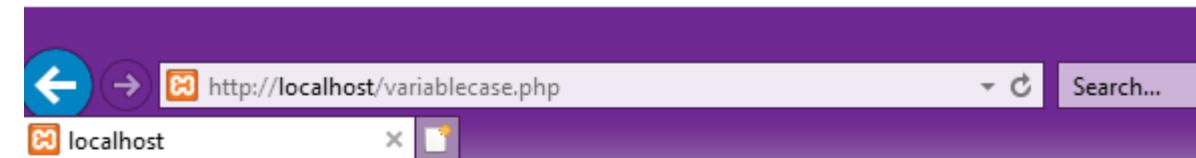


- **htdocs;** this is the web root directory. All of our PHP codes will be placed in this directory.
  - **mysql** – this directory contains all the information related to MySQL database engine, by default it runs on port 3306.
  - **php** – this directory contains PHP installation files. It contains an important file named php.ini. This directory is used to configure how PHP behaves on your server.
- By default,** the Apache web server runs on **port 80**.

# PHP Case Sensitivity

- In PHP, keyword (e.g., echo, if, else, while), functions, user-defined functions, classes are not case-sensitive.
- However, all variable names are case-sensitive.

```
<html>
  <body>
    <?php
      $color = "black";
      echo "My car is ". $ColoR . "<br>";
      echo "My dog is ". $color . "<br>";
      echo "My Phone is ". $COLOR . "<br>";
    ?>
  </body>
</html>
```



**Notice:** Undefined variable: ColoR in C:\xampp\htdocs\variablecase.php on line 5  
My car is  
My dog is black

**Notice:** Undefined variable: COLOR in C:\xampp\htdocs\variablecase.php on line 7  
My Phone is

# PHP Echo

- PHP echo is a language construct, not a function. Therefore, you don't need to use parenthesis with it. But if you want to use more than one parameter, it is required to use parenthesis.
- The syntax of PHP echo is given below:
  - `void echo ( string $arg1 [, string $... ] )`
- PHP echo statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:
  - echo is a statement, which is used to display the output.
  - echo can be used with or without parentheses: `echo()`, and `echo`.
  - echo does not return any value.
  - We can pass multiple strings separated by a comma (,) in echo.
  - echo is faster than the print statement.

# Examples of Echo

## PHP echo: printing string

```
1.<?php  
2.echo "Hello by PHP echo";  
3.?>
```

## PHP echo: printing multi line string

```
1.<?php  
2.echo "Hello by PHP echo  
3.this is multi line  
4.text printed by  
5.PHP echo statement  
6.";  
7.?>
```

## PHP echo: printing escaping characters

```
1.<?php  
2.echo "Hello escape \"sequence\" characters";  
3.?>
```

## PHP echo: printing variable value

```
1.<?php  
2.$msg="Hello JavaTpoint PHP";  
3.echo "Message is: $msg";  
4.?>
```

# PHP Print

- Like PHP echo, PHP print is a language construct, so you don't need to use parenthesis with the argument list. Print statement can be used with or without parentheses: print and print().
  - **Unlike echo, it always returns 1.**
- The syntax of PHP print is given below:
  - **int print(string \$arg)**
- PHP print statement can be used to print the string, multi-line strings, escaping characters, variable, array, etc. Some important points that you must know about the echo statement are:
  - print is a statement, used as an alternative to echo at many times to display the output.
  - print can be used with or without parentheses.
  - print always returns an integer value, which is 1.
  - Using print, we cannot pass multiple arguments.
  - print is slower than the echo statement.

1.<?php

```
2.print "Hello by PHP print ";
3.print ("Hello by PHP print()");
4.?>
```

1.<?php

```
2.$msg="Hello print() in PHP";
3.print "Message is: $msg";
4.?>
```

This is an error for Print

1.<?php

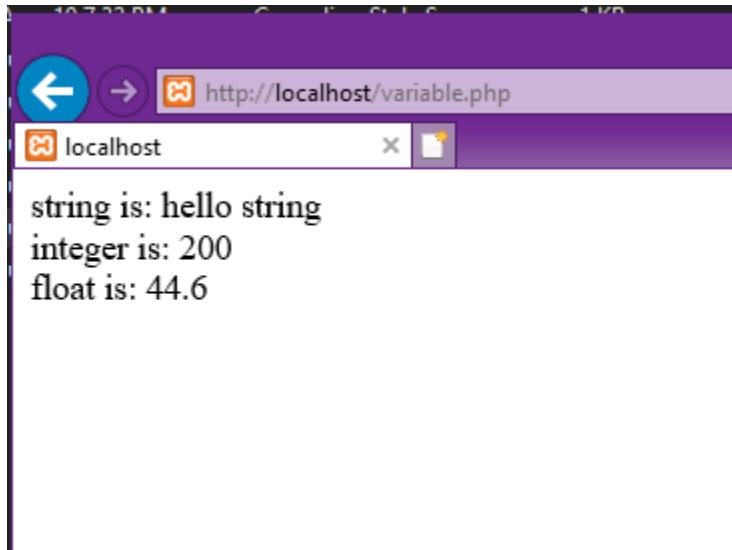
```
2. $fname = "Gunjan";
3. $lname = "Garg";
4. print "My name is: ".$fname,$lname;
5.?>
```

# PHP Variables ( syntax \$variablename=value; )

- In PHP, a variable is declared using a **\$ sign** followed by the variable name. Here, some important points to know about variables:
  - As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyzes the values and makes conversions to its correct datatype.
  - After declaring a variable, it can be reused throughout the code.
  - Assignment Operator (=) is used to assign the value to a variable.
- Rules for declaring PHP variable:
  - A variable must start with a dollar (\$) sign, followed by the variable name.
  - It can only contain alpha-numeric character and underscore (A-z, 0-9, \_).
  - A variable name must start with a letter or underscore (\_) character.
  - A PHP variable name cannot contain spaces.
  - One thing to be kept in mind that the variable name cannot start with a number or special symbols.
  - PHP variables are case-sensitive, so \$name and \$NAME both are treated as different variable.

# PHP Variable: Declaring string, integer, and float

```
1.<?php  
2.$str="hello string";  
3.$x=200;  
4.$y=44.6;  
5.echo "string is: $str <br/>";  
6.echo "integer is: $x <br/>";  
7.echo "float is: $y <br/>";  
8.?>
```



# PHP Variable: Sum of two variables

```
1.<?php  
2.$x=5;  
3.$y=6;  
4.$z=$x+$y;  
5.echo $z;  
6.??>
```

## PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

## PHP Variable: Rules

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

```
1.<?php  
2.$a="hello";//letter (valid)  
3.$_b="hello";//underscore (valid)  
4.  
5.echo "$a <br/> $_b";  
6.??>
```

# PHP Variable Scope

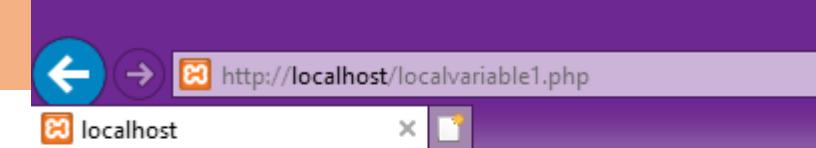
- The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."
- PHP has three types of variable scopes:
  - Local variable
  - Global variable
  - Static variable

# Local variable

- The variables that are declared within a function are called local variables for that function.
- These local variables have their scope only in that particular function in which they are declared.
- This means that these variables cannot be accessed outside the function, as they have local scope.

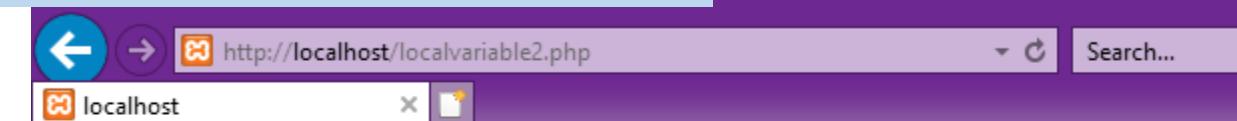
```

1.<?php
2. function local_var()
3. {
4.     $num = 45; //local variable
5.     echo "Local variable declared inside the function is: ". $num;
6. }
7. local_var();
8.?>
```



```

<?php
function mytest()
{
    $lang = "PHP";
    echo "Web development language: " . $lang;
}
mytest();
//using $lang (local variable) outside the function will generate an
error
echo $lang;
?>
```



# Global variable

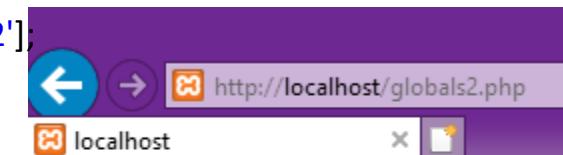
- The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program.
- To access the global variable within a function, use the **GLOBAL** keyword before the variable. However, these variables can be directly accessed or used outside the function without any keyword.
- Therefore there is no need to use any keyword to access a global variable outside the function.

```

1.<?php
2. $name = "Sanaya Sharma";      //Global Variable
3. function global_var()
4. {
5.   global $name;
6.   echo "Variable inside the function: ". $name;
7.   echo "<br>";
8. }
9. global_var();
10. echo "Variable outside the function: ". $name;
11.?>
```

```

1.<?php
2. $num1 = 5;    //global variable
3. $num2 = 13;   //global variable
4. function global_var()
5. {
6.   $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
7.   echo "Sum of global variables is: " . $sum;
8. }
9. global_var();
10.?>
```



Sum of global variables is: 18

# PHP \$ and \$\$ Variables

- The **\$var** (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.
- The **\$\$var** (double dollar) is a reference variable that stores the value of the \$variable inside it.

```
1.<?php
2.$x = "abc";
3.$$x = 200;
4.echo $x."<br/>";
5.echo $$x."<br/>";
6.echo $abc;
7.?>
```

abc	
200	
200	

# PHP Constants

- PHP constants are name or identifier that **can't be changed** during the execution of the script except for magic constants, which are not really constants.
- PHP constants can be defined by 2 ways:
  - Using define() function
  - Using const keyword
- Constants are similar to the variable except once they defined, they can never be undefined or changed. They remain constant across the entire program.
- PHP constants follow the same PHP variable rules.
- **For example**, it can be started with a letter or underscore only.
- Conventionally, PHP constants should be defined in uppercase letters.

# PHP constant: define()

- Use the define() function to create a constant. It defines constant at run time. Let's see the syntax of define() function in PHP.
- **define(name, value, case-insensitive)**
  - **name:** It specifies the constant name.
  - **value:** It specifies the constant value.
  - **case-insensitive:** Specifies whether a constant is case-insensitive. Default value is false. It means it is case sensitive by default.

```
1.<?php  
2.define("MESSAGE","Hello WIMS PHP");  
3.echo MESSAGE;  
4.?>
```

Hello WIMS PHP

# PHP constant: const keyword

- PHP introduced a keyword **const** to create a constant. The const keyword defines constants at compile time. It is a language construct, not a function.
- The constant defined using const keyword are **case-sensitive**.

```
1.<?php
2.const MESSAGE="Hello const by JavaTpoint PHP";
3.echo MESSAGE;
4.?>
```

- Constant() function
- There is another way to print the value of constants using constant() function instead of using the echo statement.

# Constant vs Variables

Constant	Variables
Once the constant is defined, it can never be redefined.	A variable can be undefined as well as redefined easily.
A constant can only be defined using define() function. It cannot be defined by any simple assignment.	A variable can be defined by simple assignment (=) operator.
There is no need to use the dollar (\$) sign before constant during the assignment.	To declare a variable, always use the dollar (\$) sign before the variable.
Constants do not follow any variable scoping rules, and they can be defined and accessed anywhere.	Variables can be declared anywhere in the program, but they follow variable scoping rules.
Constants are the variables whose values can't be changed throughout the program.	The value of the variable can be changed.
By default, constants are global.	Variables can be local, global, or static.

# Magic Constants

- Magic constants are the predefined constants in PHP which get changed on the basis of their use. They start with double underscore (`__`) and ends with double underscore.
- They are similar to other predefined constants but as they change their values with the context, they are called **magic** constants.
- There are **nine** magic constants in PHP. In which eight magic constants start and end with double underscores (`__`).
  - `__LINE__`
  - `__FILE__`
  - `__DIR__`
  - `__FUNCTION__`
  - `__CLASS__`
  - `__TRAIT__`
  - `__METHOD__`
  - `__NAMESPACE__`
  - [ClassName::class](#)

# PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types (predefined)
2. Compound Types (user-defined)
3. Special Types

## PHP Data Types: Scalar Types

It holds only single value. There are 4 scalar data types in PHP.

- 1.[boolean](#)
- 2.[integer](#)
- 3.[float](#)
- 4.[string](#)

## PHP Data Types: Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

- 1.[array](#)
- 2.[object](#)

## PHP Data Types: Special Types

There are 2 special data types in PHP.

- 1.[resource](#)
- 2.[NULL](#)

# PHP Indexed Array

- PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

- There are two ways to define indexed array

- 1st way:

- `$season=array("summer","winter","spring","autun`

- 2nd way:

- `$season[0]="summer";`
  - `$season[1]="winter";`
  - `$season[2]="spring";`
  - `$season[3]="autumn";`

```
1.<?php
```

```
2.$season=array("summer","winter","spring","autum
```

```
3.echo "Season are: $season[0], $season[1], $season  
eason[3];
```

```
4.?>
```

```
1.<?php
```

```
2.$season[0]="summer";
```

```
3.$season[1]="winter";
```

```
4.$season[2]="spring";
```

```
5.$season[3]="autumn";
```

```
6.echo "Season are: $season[0], $season[1], $season[2] and $  
eason[3];
```

```
7.?>
```

# PHP Associative Array

- We can associate name with each array elements in PHP using => symbol.
  - There are two ways to define associative array:
  - 1st way:
    - `$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");`
  - 2nd way:
    - `$salary["Sonoo"]="350000";`
    - `$salary["John"]="450000";`
    - `$salary["Kartik"]="200000";`
- ```
1.<?php
2.$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
3.echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
4.echo "John salary: ".$salary["John"]."<br/>";
5.echo "Kartik salary: ".$salary["Kartik"]."<br/>";
6.?>
```

# PHP Multidimensional Array

- PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row \* column.

```
1.$emp = array ( array(1,"sonoo",400000), array(2,"john",500000), array(3,"rahul",300000) );
```

# PHP Multidimensional Array Example

| Id   | Name  | Salary |
|--|-------|--------|
| 1<br>1.<?php<br>2.\$emp = <b>array</b>   | sonoo | 400000 |
| 2<br>3. (<br>4. <b>array</b> (1, "sonoo",400000),  | john  | 500000 |
| 3<br>5. <b>array</b> (2, "john",500000),<br>6. <b>array</b> (3, "rahul",300000)<br>7. );<br>8. <b>for</b> (\$row = 0; \$row < 3; \$row++) {<br>9. <b>for</b> (\$col = 0; \$col < 3; \$col++) {<br>10. echo \$emp[\$row][\$col]." ";<br>11. }<br>12. echo "<br/>";<br>13.}<br>14.?> | rahul | 300000 |

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

# PHP Array Functions : 1) PHP array() function

- PHP provides various array functions to access and manipulate the elements of array.
  - The important PHP array functions are given below.
    - 1) PHP array() function
      - PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.
    - **array** **array** ([ mixed \$... ] )

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Season are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output:

Season are: summer, winter, spring and autumn

## 2) PHP array\_change\_key\_case() function

- PHP array\_change\_key\_case() function changes the case of all key of an array.
- Note: It does not change the value of the array.

**Syntax**

```
array array_change_key_case ( array $array [, int $case = CASE_LOWER ] )
```

### Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

### Output:

```
Array ( [SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000 )
```

# 3) PHP array\_chunk() function

- PHP array\_chunk() function splits array into chunks. By using array\_chunk() method, you can divide array into many parts.

## Syntax

```
array array_chunk ( array $array , int $size [, bool $preserve_keys = false ] )
```

## Example

```
<?php
$Salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_chunk($Salary,2));
?>
```

## Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 )
)
```

# 4) PHP count() function

- PHP count() function counts all elements in an array.

## Syntax

```
1.int count ( mixed $array_or_countable [, int $mode = COUNT  
_NORMAL ] )
```

## Example

```
1.<?php  
2.$season=array("summer","winter","spring","autumn");
```

```
3.echo count($season);
```

```
4.?>
```

Output: 4

# 5) PHP sort() function

- PHP sort() function sorts all the elements in an array.

## Syntax

1.bool sort ( **array** &\$array [, int \$sort\_flags = SORT\_REGULAR ] )

## Example

```
1.<?php
2.$season=array("summer","winter","spring","autumn");
3.sort($season);
4.foreach($season as $s )
5.{  
6. echo "$s<br />";
7.}  
8.?>
```

## Output:

```
1.autumn
2.spring
3.summer
4.winter
```

# 6) PHP array\_reverse() function

- PHP array\_reverse() function returns an array containing elements in reversed order.

## Syntax

**1.array** array\_reverse ( **array** \$array [, bool \$preserve\_keys = false ] )

## Example

```
1.<?php
2.$season=array("summer","winter","spring","autumn");
3.$reverseseason=array_reverse($season);
4.foreach( $reverseseason as $s )
5.{  

6. echo "$s<br />";
7.}  

8.?>
```

# PHP String

- PHP string is a sequence of characters i.e., used to store and manipulate text. PHP supports only 256-character set and so that it does not offer native Unicode support. There are 4 ways to specify a string literal in PHP.
  - single quoted
  - double quoted
  - heredoc syntax
  - newdoc syntax
- single quoted
  - 1.<?php
  - 2. \$str='Hello text within single quote';
  - 3. echo \$str;
  - 4.?>

- Double Quoted

- Now, you **can't use double quote directly inside double quoted string.**

```
1.<?php
2.$str1="Using double "quote" directly inside double quoted st
ring";
3.echo $str1;
4.?>
```

In double quoted strings, **variable will be interpreted.**

```
1.<?php
2.$num1=10;
3.echo "Number is: $num1";
4.?>
```

Number is: 10



## Session : PHP - Form

Dr.S.Udhayakumar  
Professor

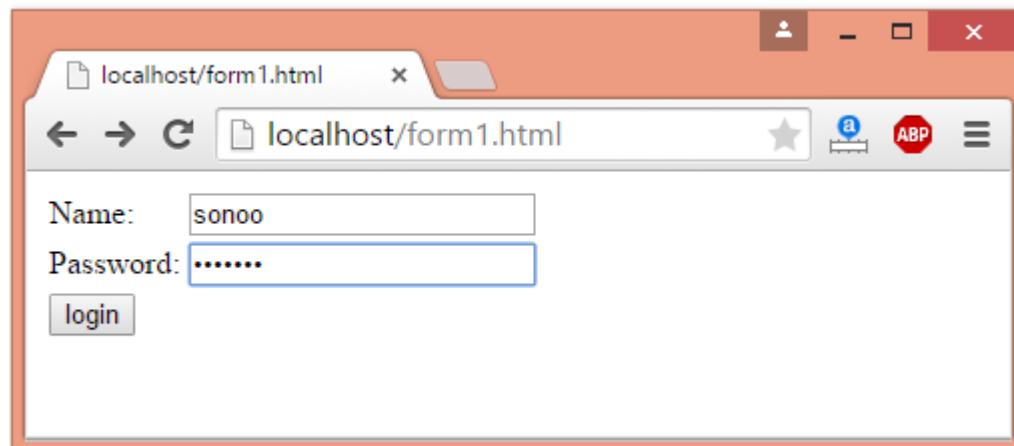
# PHP Forms

- To get form data, we need to use PHP superglobals `$_GET` and `$_POST`.
- The form request may be get or post. To retrieve data from get request, we need to use `$_GET`, for post request `$_POST`.

## PHP Get Form

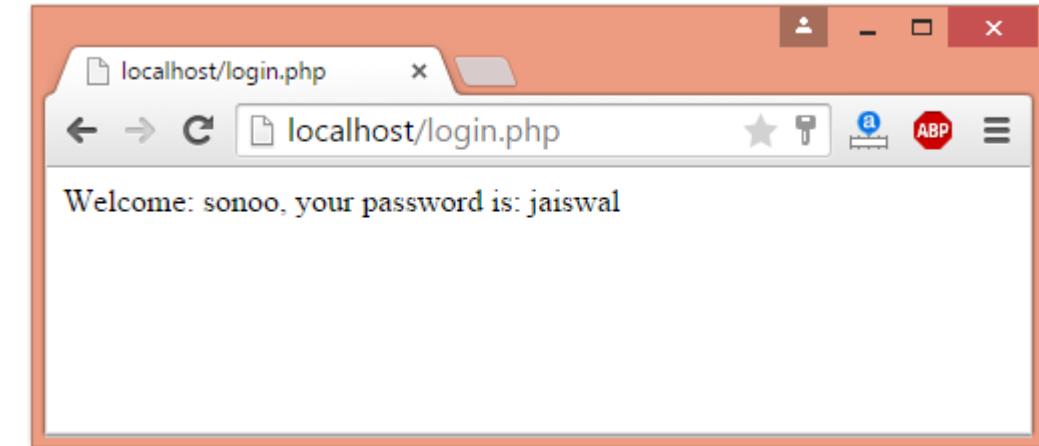
- Get request is the default form request. The data passed through get request is visible on the URL browser so it is not secured. You can send limited amount of data through get request.
- Let's see a simple example to receive data from get request in PHP.

```
1.<form action="welcome.php" method="get">
2.Name: <input type="text" name="name"/>
3.<input type="submit" value="visit"/>
4.</form>
```



### welcome.php

```
1.<?php
2.$name=$_GET["name"];//receiving name field value in $name variable
3.echo "Welcome, $name";
4.?>
```



# PHP Post Form

- Post request is widely used to submit form that have large amount of data such as file upload, image upload, login form, registration form etc.
- The data passed through post request is not visible on the URL browser so it is secured. You can send large amount of data through post request.

```
1.<form action="login.php" method="post">
2.<table>
3.<tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
4.<tr><td>Password:</td><td> <input type="password" name="password"/></td>
5.<tr><td colspan="2"><input type="submit" value="login"/> </td></tr>
6.</table>
7.</form>
```

```
1.<?php
2.$name=$_POST["name"];//receiving name field value in $name variable
3.$password=$_POST["password"];//receiving password field value in $password variable
4.
5.echo "Welcome: $name, your password is: $password";
6.?>
```

# PHP Include and Require

- PHP allows us to create various elements and functions, which are used several times in many pages.
- It takes much time to script these functions in multiple pages.
- Therefore, use the concept of **file inclusion** that helps to include files in various programs and saves the effort of writing code multiple times.
- "PHP allows you to include file so that a page content can be reused many times.
- It is very helpful to include files when you want to apply the same HTML or PHP code to multiple pages of a website." There are two ways to include file in PHP.
  - include
  - require

# include and require

- Both **include** and **require** are identical to each other, except failure.
  - **include** only generates a warning, i.e., E\_WARNING, and continue the execution of the script.
  - **require** generates a fatal error, i.e., E\_COMPILE\_ERROR, and stop the execution of the script.
- Advantage
  - **Code Reusability:** By the help of include and require construct, we can reuse HTML code or PHP script in many PHP scripts.
  - **Easy editable:** If we want to change anything in webpages, edit the source file included in all webpage rather than editing in all the files separately.
  -

*File: menu.html*

```
1.<a href="http://www.javatpoint.com">Home</a> |  
2.<a href="http://www.javatpoint.com/php-tutorial">PHP</a> |  
3.<a href="http://www.javatpoint.com/java-tutorial">Java</a> |  
4.<a href="http://www.javatpoint.com/html-tutorial">HTML</a>
```

*File: include1.php*

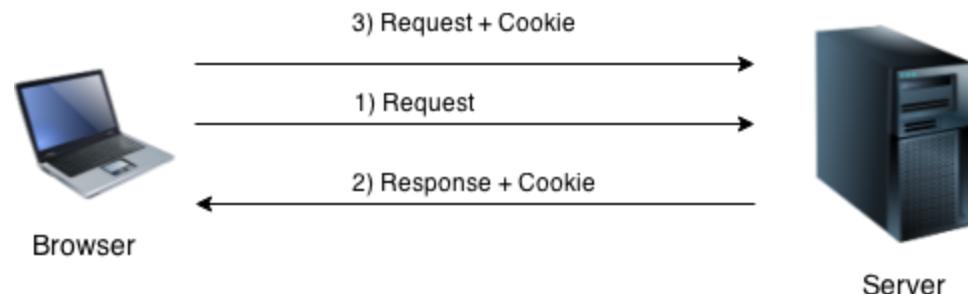
```
1.<?php include("menu.html"); ?>  
2.<h1>This is Main Page</h1>
```

*File: require1.php*

```
1.<?php require("menu.html"); ?>  
2.<h1>This is Main Page</h1>
```

# PHP Cookie

- PHP cookie is a small piece of information which is stored at client browser. It is used to recognize the user.
- Cookie is created at server side and saved to client browser. Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.



In short, cookie can be created, sent and received at server end.

# PHP setcookie() function

- PHP setcookie() function is used to set cookie with HTTP response. Once cookie is set, you can access it by \$\_COOKIE superglobal variable.
- Examples of Cookie
  - 1.setcookie("CookieName", "CookieValue");/\* defining name and value only\*/
  - 2.setcookie("CookieName", "CookieValue", time()+1\*60\*60); //using expiry in 1 hour(1\*60\*60 seconds or 3600 seconds)
  - 3.setcookie("CookieName", "CookieValue", time()+1\*60\*60, "/mypath/", "mydomain.com", 1);

# PHP \$\_COOKIE

- PHP \$\_COOKIE superglobal variable is used to get cookie.

1.\$value=\$\_COOKIE["CookieName"];//returns cookie value

File: cookie1.php

```
1.<?php
2.setcookie("user", "amrita");
3.?
4.<html>
5.<body>
6.<?php
7.if(!isset($_COOKIE["user"])){
8. echo "Sorry, cookie is not found!";
9.} else {
10. echo "<br/>Cookie Value: " . $_COOKIE["user"];
11.
12.?
13.</body>
14.</html>
```

Output: Sorry, cookie is not found!

*Sorry, cookie is not found!*

Firstly cookie is not set. But, if you *refresh* the page, you will see cookie is set now.

Output: Cookie Value: Sonoo

# PHP Delete Cookie

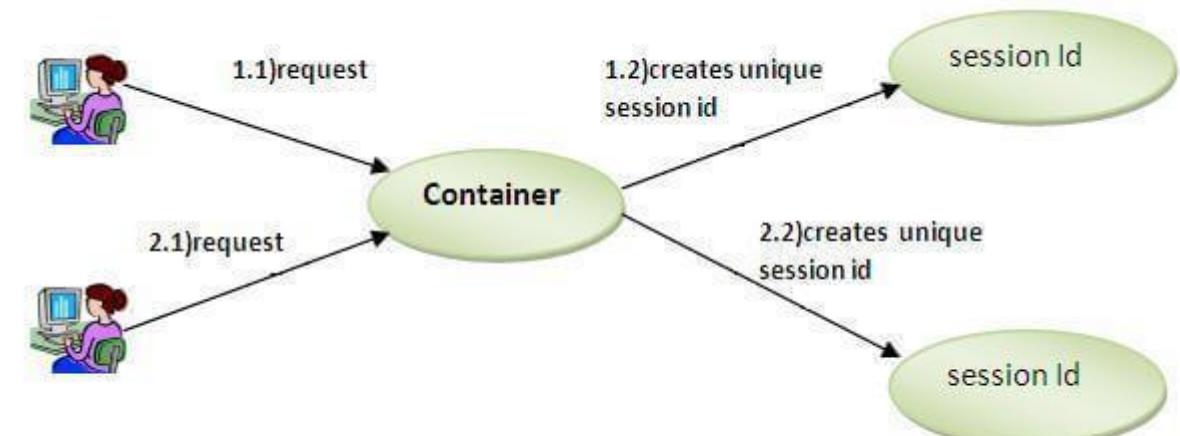
- If you set the expiration date in past, cookie will be deleted.

*File: cookie1.php*

```
1.<?php  
2.setcookie ("CookieName", "", time() -  
3600); // set the expiration date to one hour ago  
3.?>
```

# PHP Session

- PHP session is used to store and pass information from one page to another temporarily (until user close the website).
- PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.
- PHP session creates unique user id for each browser to recognize the user and avoid conflict between multiple browsers.
- 



# PHP MySQL Connect

- Since PHP 5.5, `mysql_connect()` extension is deprecated. Now it is recommended to use one of the 2 alternatives.
  - `mysqli_connect()`
  - `PDO::__construct()` ( PHP Data Object)
- **PHP `mysqli_connect()`**
- PHP `mysqli_connect()` function is used to connect with MySQL database. It returns resource if connection is established or null.
- **Syntax**
- **resource `mysqli_connect (server, username, password)`**
- PHP `mysqli_close()`
- PHP `mysqli_close()` function is used to disconnect with MySQL database. It returns true if connection is closed or false.
- Syntax
- `bool mysqli_close(resource $resource_link)`

# PHP MySQL Connect Example

```
1.<?php
2.$host = 'localhost:3306';
3.$user = "";
4.$pass = "";
5.$conn = mysqli_connect($host, $user, $pass);
6.if(! $conn )
7.{  
8. die('Could not connect: ' . mysqli_error());  
9.}  
10.echo 'Connected successfully';
11.mysqli_close($conn);
12.?>
```

# PHP MySQLi Create Database Example

```
1.<?php
2.$host = 'localhost:3306';
3.$user = "";
4.$pass = "";
5.$conn = mysqli_connect($host, $user, $pass);
6.if(! $conn )
7.{  
8. die('Could not connect: ' . mysqli_connect_error());
9.}  
10.echo 'Connected successfully<br/>';
11.  
12.$sql = 'CREATE Database mydb';
13.if(mysqli_query( $conn,$sql)){
14. echo "Database mydb created successfully.";
15.}else{
16.echo "Sorry, database creation failed ".mysqli_error($conn);
17.}  
18.mysqli_close($conn);
19.?>
```

Connected successfully  
Database mydb created successfully.

# PHP MySQLi Create Table Example

```
1.<?php
2.$host = 'localhost:3306';
3.$user = "";
4.$pass = "";
5.$dbname = 'test';
6.
7.$conn = mysqli_connect($host, $user, $pass,$dbname);
8.if(!$conn){
9. die('Could not connect: '.mysqli_connect_error());
10.}
11.echo 'Connected successfully<br/>';
12.
13.$sql = "create table emp5(id INT AUTO_INCREMENT,name VARCHAR(20) NOT
NULL,
14.emp_salary INT NOT NULL,primary key (id))";
15.if(mysqli_query($conn, $sql)){
16. echo "Table emp5 created successfully";
17.}else{
18.echo "Could not create table: ". mysqli_error($conn);
19.}
20.
21.mysqli_close($conn);
22.?=
```

Connected successfully  
Table emp5 created successfully

# WEB APP SECURITY



```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => {
                  console.log(value)
                }}
              </ProductConsumer>
            </div>
          </div>
        </div>
      <React.Fragment>
```

## Course Outcomes

**CO1** Apply client-side web development to design interactive front-end web user interfaces.

**CO2** Use server-side web application concepts to develop back-end web server application

**CO3** Identify and mitigate various client-side web application security vulnerabilities

**CO4** Identify and mitigate various server-side web application security vulnerabilities

# Syllabus

- Web application development – Introduction - Architecture – Client-side technologies and frameworks – HTML – CSS – Javascript - Ajax/Fetch - Data interchange formats – XML, JSON. Server-side scripting and technologies - development – technologies - Handling client requests – Database connectivity – Sessions – Cookies.
- Web application vulnerabilities – Client-side Vulnerabilities - Cross Site Scripting (XSS) - Cross Site Request Forgery (CSRF) - Cross-origin resource sharing (CORS) - Clickjacking. Server-side Vulnerabilities - SQL injection - OS command injection - Directory traversal - Authentication - Server-side request forgery (SSRF)
- **Textbook(s)**

# Textbook and Reference books

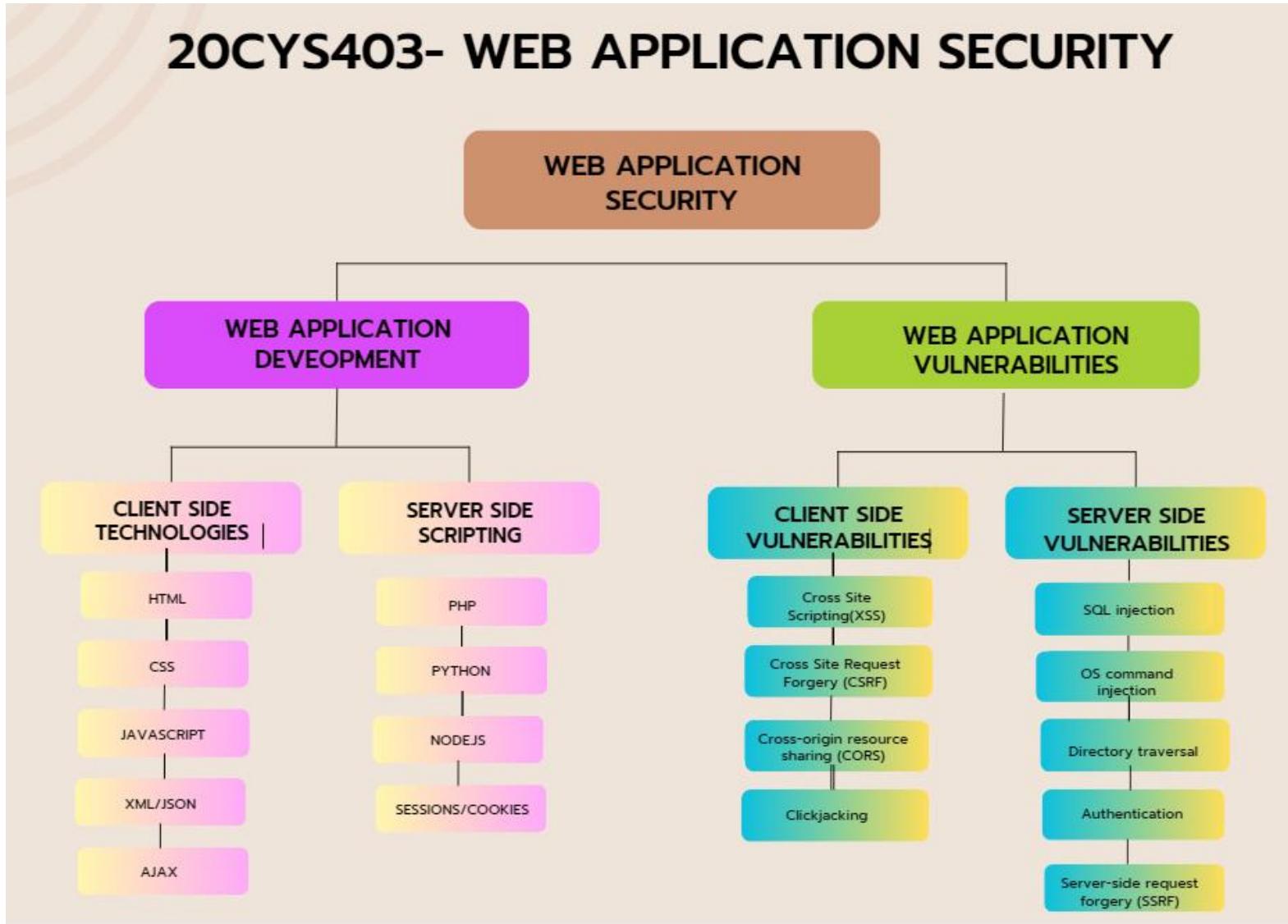
- **Textbook(s)**

1. Robin Nixon, Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites, Fifth Edition, O'Reilly Media, Inc.; 2018.
2. Dafydd Stuttard, and Marcus Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition, John Wiley & Sons; 2011

- **Reference(s)**

1. John Paul Mueller, Security for Web Developers: Using Javascript, Html, and CSS, O'Reilly, 2015
2. Andrew Hoffman. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Shroff Publishers & Distributors Pvt. Ltd, 2020.
3. Richa Gupta, Hands-on Penetration Testing for Web Applications, BPB Publications (29 March 2021)
4. Ivan Ristic, Bulletproof TLS and PKI, Second Edition: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Feisty Duck Ltd; 2nd edition, 2022
5. <https://web.stanford.edu/class/cs253/>
6. <https://nptel.ac.in/courses/128106006>
7. <https://www.rapid7.com/fundamentals/web-application-security/>
8. <https://owasp.org/>
9. <https://www.udemy.com/course/web-application-security/>

# Concept Map



# Web application vulnerabilities - Introduction

## WEB APPLICATION SECURITY VULNERABILITIES



|   |   |  |   |  |   |
|---|---|--|---|--|---|
|  Check Point | ▼ |  Fortinet                    | ▼ |  Palo Alto Networks | ▼ |
|  Gen Digital | ▼ |  Cisco                       | ▼ |  Crowdstrike        | ▼ |
|  Microsoft   | ▼ |  Trend Micro                 | ▼ |  IBM                | ▼ |
|  Darktrace   | ▼ |  FireEye                     | ▼ |  Kaspersky Lab      | ▼ |
|  Proofpoint  | ▼ |  Sophos                      | ▼ |  Rapid7             | ▼ |
|  Zscaler     | ▼ |  Imperva                     | ▼ |  Juniper Networks   | ▼ |
|  Tenable   | ▼ |  Internet Security Systems | ▼ |  Cybereason       | ▼ |
|  McAfee    | ▼ |  CyberArk                  | ▼ |  KnowBe4, Inc.    | ▼ |

# Web application vulnerabilities

- A **website vulnerability** is a
  - software code flaw/ bug,
  - system misconfiguration, or
  - some other weakness in the website/ web application or its components and processes.
- Web application vulnerabilities enable attackers to gain unauthorized access to systems/ processes/mission-critical assets of the organization.
- Having such access, attackers can
  - orchestrate attacks,
  - takeover applications,
  - engage in privilege escalation to exfiltrate data,
  - cause large-scale service disruption, and so on.

# How Can Website Vulnerabilities be Exploited?

- Website vulnerabilities are unavoidable, and most website/ web applications will have a few vulnerabilities. The two key concerns for organizations should be the exploitability factor associated with the vulnerabilities.
- Web app vulnerabilities are exploitable when there are no proper security measures in place to prevent attackers from finding and taking advantage of vulnerabilities. The factors that affect the exploitability of a vulnerability are the complexity associated with exploitation and the availability of active/ known exploits.
- Based on this, the risk associated with the vulnerability is calculated and vulnerabilities are categorized into critical, high, medium, and low risk. The critical and high-risk vulnerabilities must be fixed and protected on a high-priority basis.

# Some facts and figures

- 75% of attacks leveraged web application vulnerabilities that were known at least for 2 years!
- 80% of exploits were published even before the CVE (Common Vulnerabilities and Exposure) related to that exploit was made public.

**This means, instead of organizations steering ahead of attackers, attackers had the first-mover advantage in most exploits.**



**OWASP**

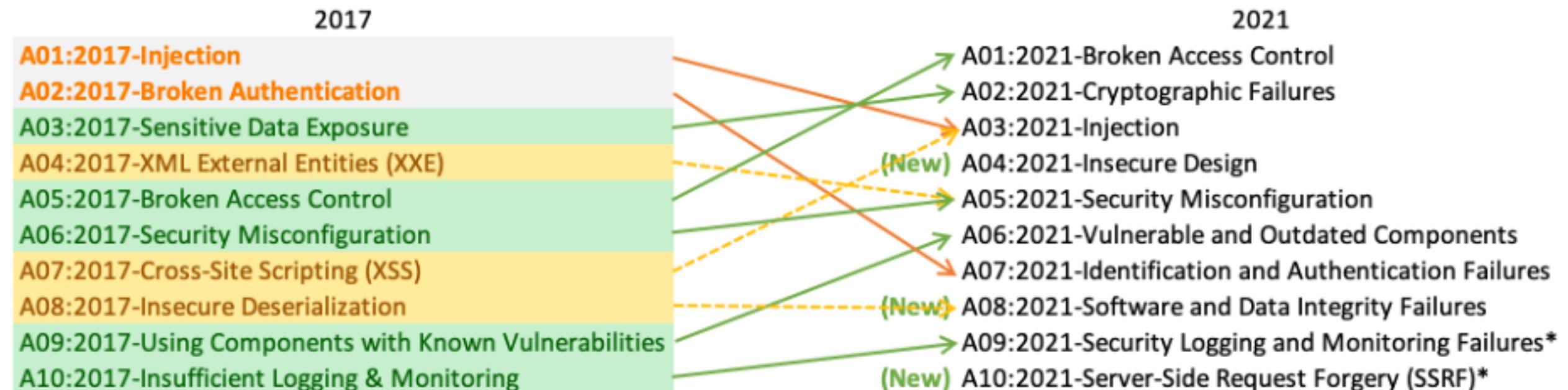
Open Web Application  
Security Project

# Open Web Application Security Project (OWASP)

- The Open Web Application Security Project (OWASP) is a non-profit global community that strives to promote application security across the web.
- A core OWASP principle is that their knowledge base is freely and easily accessible on their website.
- With its tens of thousands of members and hundreds of chapters, OWASP is considered highly credible, and developers have come to count on it for essential **web application security**, and **API security** guidance.

# What is the OWASP Top 10?

- Every few years, OWASP revises and publishes its list of the top 10 web **application vulnerabilities**.
- The list includes not only the OWASP Top 10 threats but also the potential impact of each vulnerability and how to avoid them.
- The comprehensive list is compiled from a variety of expert sources such as security consultants, security vendors, and security teams from companies and organizations of all sizes.
- It is recognized as an essential guide to web application security best practices.
- The OWASP Top 10 is largely intended to raise awareness.
- However, since its debut in 2003, enterprises have used it as a de facto industry AppSec standard. I
- If we look at the document closely, it specifically calls out the number of **CWE's** (Common Weakness Enumeration) attached with it.



\* From the Survey

**A1** Injection

**A2** Broken Authentication

**A3** Sensitive Data Exposure

**A4** XML External Entity (XXE)

**A5** Broken Access Control

**A6** Security Misconfiguration

**A7** Cross-Site Scripting (XSS)

**A8** Insecure Deserialization

**A9** Using Components with Known Vulnerabilities

**A10** Insufficient Logging and Monitoring

# Other Web Application Threats

**01** Directory Traversal

**07** Cookie Snooping

**13** Denial-of-Service (DoS)

**02** Unvalidated Redirects  
and Forwards

**08** Hidden Field Manipulation

**14** Buffer Overflow

**03** Waterhole Attack

**09** Authentication Hijacking

**15** CAPTCHA Attacks

**04** Cross Site Request Forgery

**10** Obfuscation Application

**16** Platform Exploits

**05** Cookie/Session Poisoning

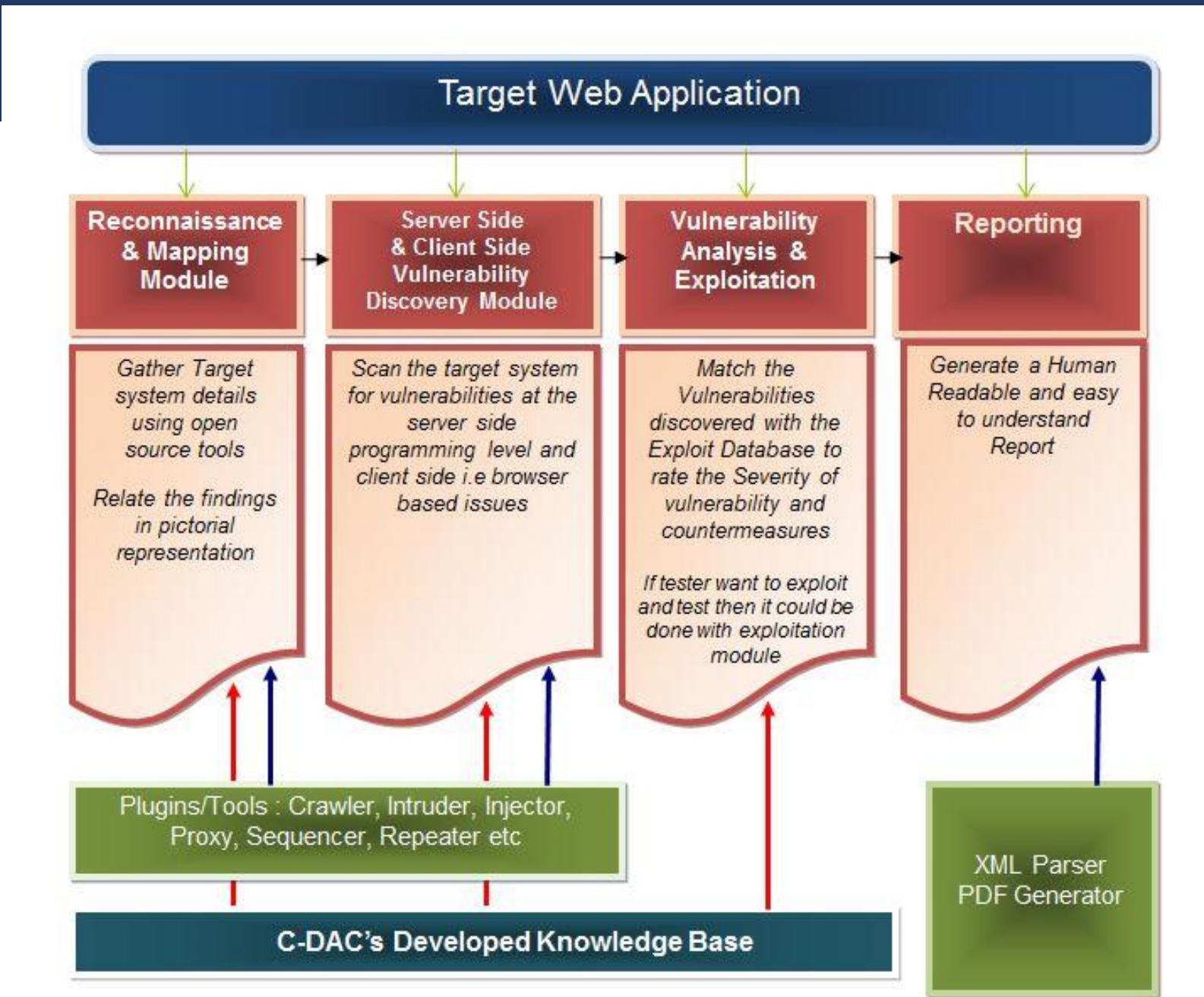
**11** Broken Session Management

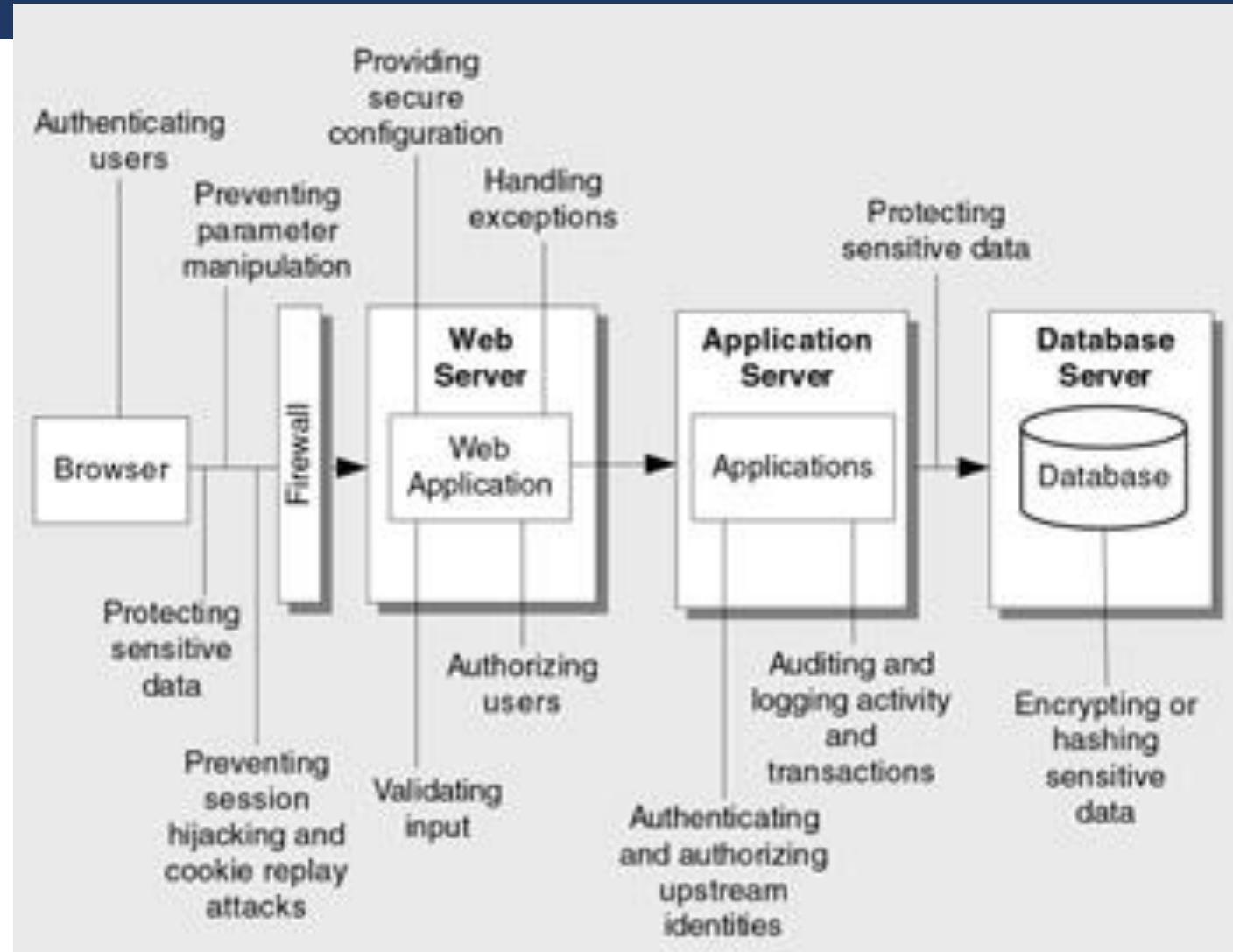
**17** Network Access Attacks

**06** Web Services Attack

**12** Broken Account Management

**18** DMZ Protocol Attacks





**Table 4.1: Web Application Vulnerabilities and Potential Problem Due to Bad Design**

| Vulnerability Category   | Potential Problem Due to Bad Design   |
|--------------------------|---|
| Input Validation         | Attacks performed by embedding malicious strings in query strings, form fields, cookies, and HTTP headers. These include command execution, cross-site scripting (XSS), SQL injection, and buffer overflow attacks. |
| Authentication           | Identity spoofing, password cracking, elevation of privileges, and unauthorized access.   |
| Authorization            | Access to confidential or restricted data, tampering, and execution of unauthorized operations.   |
| Configuration Management | Unauthorized access to administration interfaces, ability to update configuration data, and unauthorized access to user accounts and account profiles.  |
| Sensitive Data           | Confidential information disclosure and data tampering.   |
| Session Management       | Capture of session identifiers resulting in session hijacking and identity spoofing.  |
| Cryptography             | Access to confidential data or account credentials, or both.  |
| Parameter Manipulation   | Path traversal attacks, command execution, and bypass of access control mechanisms among others, leading to information disclosure, elevation of privileges, and denial of service.                                 |
| Exception Management     | Denial of service and disclosure of sensitive system level details.   |
| Auditing and Logging     | Failure to spot the signs of intrusion, inability to prove a user's actions, and difficulties in problem diagnosis.   |

# XSS

## Cross-site Scripting

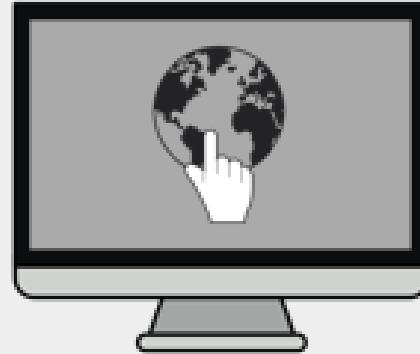
# Cross-Site Scripting (XSS)

- Exploits vulnerabilities in dynamically generated web pages
- Allows attackers to inject client side scripts into web pages viewed by other users
- Occurs when invalidated input data is included in dynamic content
- Can inject malicious scripts or web content by hiding it within legitimate requests

# Cross-Site Scripting (XSS)

1. Hacker injects trusted website with malicious script

1



TRUSTED WEBSITE

2. Victim visits trusted website and triggers malicious script

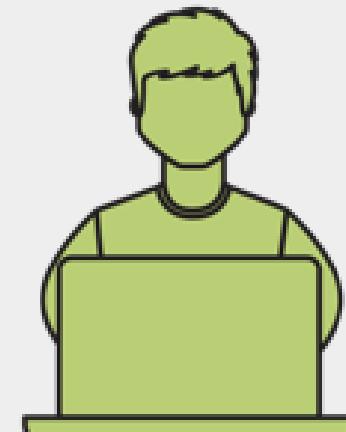
2



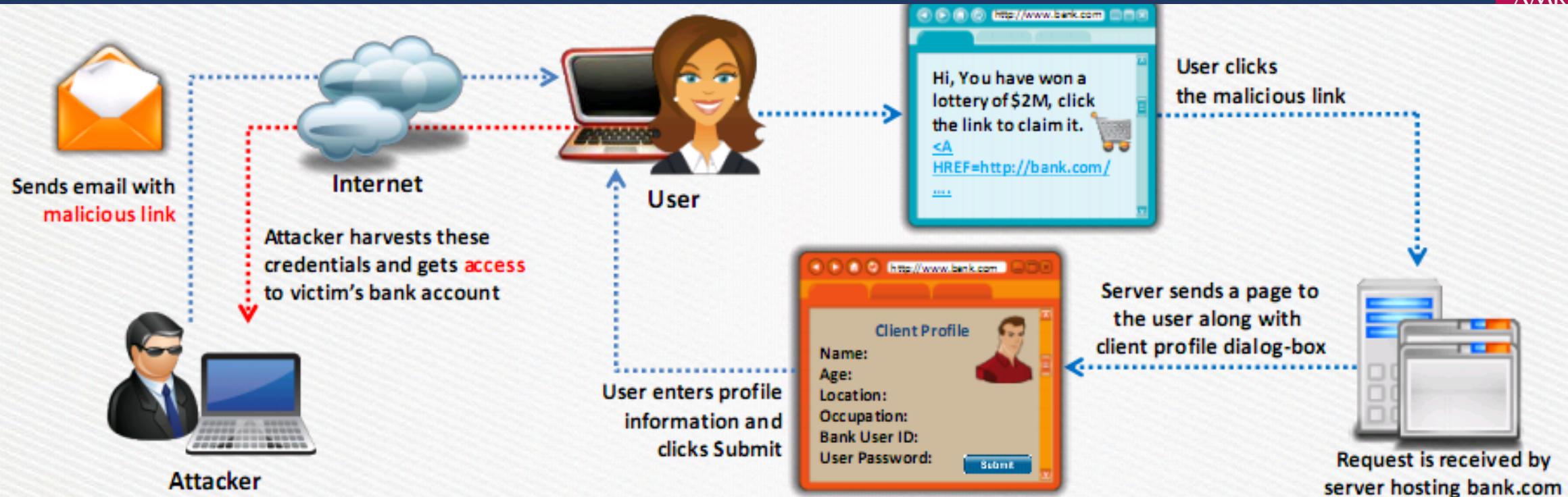
HACKER

3. Victim's browser executes malicious script and unknowingly forwards desired information (session token, cookie, etc.) to hacker

3



VICTIM



- In this example, the attacker crafts an email message with a malicious script and sends it to the victim:  
`<A HREF=http://bank.com/registration.cgi?clientprofile=<SCRIPT>  
maliciouscode</SCRIPT>>Click here</A>`
- When the user clicks on the link, the URL is sent to **bank.com** with the malicious code
- The legitimate server hosting **bank.com** website sends a page back to the user including the value of **clientprofile**, and the malicious code is executed on the client machine

# Definition of XSS attack

- XSS comes under the category of code injection attacks. It is one of the most severe security vulnerabilities that exist in the web applications.
- It is ranked among the top 10 web application vulnerabilities by the OWASP. In this attack, the attacker injects judiciously crafted malicious scripts into the vulnerable web application.
- The origin of the XSS attack is the inappropriate filtering of the data being entered by any user, due to which an attacker easily introduces malicious code into the Web pages.
- These malicious scripts run on the client-side in the user's Web browser.
- It enables an attacker to evade Same-Origin-Policy (SOP) that helps in separating the content of different web application.

- XSS attack does not cause any harm to the web application, rather it targets the end-users of the web application. Once it is successfully executed, the attacker can gain access to sensitive information like cookie information, session token, etc. It may be triggered as an initial step to launch other cyber-attacks like phishing and to infect the benign user device with malware so that the infected device becomes a part of the botnet army. These bots then will be exploited to launch Distributed Denial of Service (DDoS) attacks on a massive scale.

- This attack usually exploits vulnerabilities in the web application that are built with a range of programming languages comprising PHP, Java, ASP.net, etc. Among all, PHP is a highly used language. An attacker can craft malicious code using a variety of programming languages consisting of JavaScript, VBScript, ActiveX, Flash. Most browsers use JavaScript to support dynamic web pages, therefore, attackers frequently exploit the JavaScript language to craft malicious attack vectors for XSS attacks. This drives XSS attacks as one of the most recurrently occurring and venomous attacks.

# Consequences of XSS attack

- XSS attacks can cause destructive consequences. Some of the most common and severe effects are described below:
- **Cookie stealing:** It is possible for an attacker to steal the cookie sent by the server containing session ID and take control of the user account and may perform malicious activities such as sending spam messages to user's friends etc.
- **Account hijacking:** Attackers can steal sensitive information like financial account credentials or bank account login details for the use of their benefits. If account is hijacked, the attacker has access to the OSN server and database system and thus has complete control over the OSN Web application.
- **Misinformation:** This is a threat of credentialled misinformation. It may include malwares which may track the user like traffic statistics, leading to loss of privacy. Moreover, these may also alter the content of the page, leading to loss of integrity.
- **Denial-of-Service Attack:** Data availability is utmost important functionality provided by any enterprise. But the XSS attack can be used to redirect the user to some other fake web page so that he/she cannot access the legitimate website, whenever the user makes a request to that web page. Thus, the attacker successfully launches the DDoS attack. Malicious scripts may also crash the user browser by indefinitely blocking the service of the Web application through pop-ups.
- **Browser exploitation:** Malicious scripts may redirect the user browser to attacker's site so that attacker can take full control of user's computer and use it to install malicious programs like viruses, Trojan horses etc. and may get access to user's sensitive information.
- **Remote Control on System:** Once the XSS attack vector is executed on the victim's machine, it will open a way for the attacker to inject different malwares that help to gain remote access to the victim's system. Thereafter, the system may perform malicious activity on the Internet or become part of the network to launch different attacks such as the botnet army.
- **Phishing:** When the user clicks on the malicious link sent by the attacker, then it may redirect the user to the fake web site designed by the attacker to gain access to sensitive information such as the user's login credentials.

# How is XSS Being Performed?

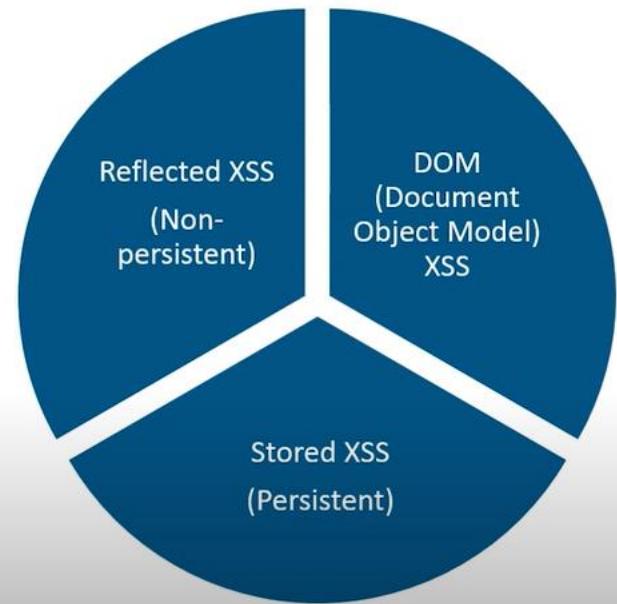
## WHAT IS CROSS SITE SCRIPTING?

Cross Site Scripting (XSS) is a Code Injection attack executed on the client-side of a Web Application.

- Attacker injects malicious script through the web browser
- The malicious script is executed when the victim visits the web page or web server
- Steals Cookies, Session tokens and other sensitive information
- Modify the contents of the Website



# TYPES OF CROSS SITE SCRIPTING



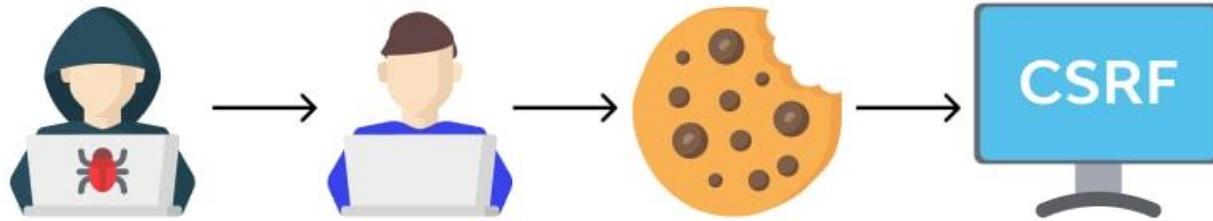


# HOW TO USE CROSS SITE SCRIPTING?

Reflected XSS  
(Non-persistent)

- Script is executed on the victim side
- Script is not stored on the server

## CSRF - Cross site request forgery attack



<https://owasp.org/www-community/attacks/csrf>

- <https://www.synopsys.com/glossary/what-is-csrf.html>

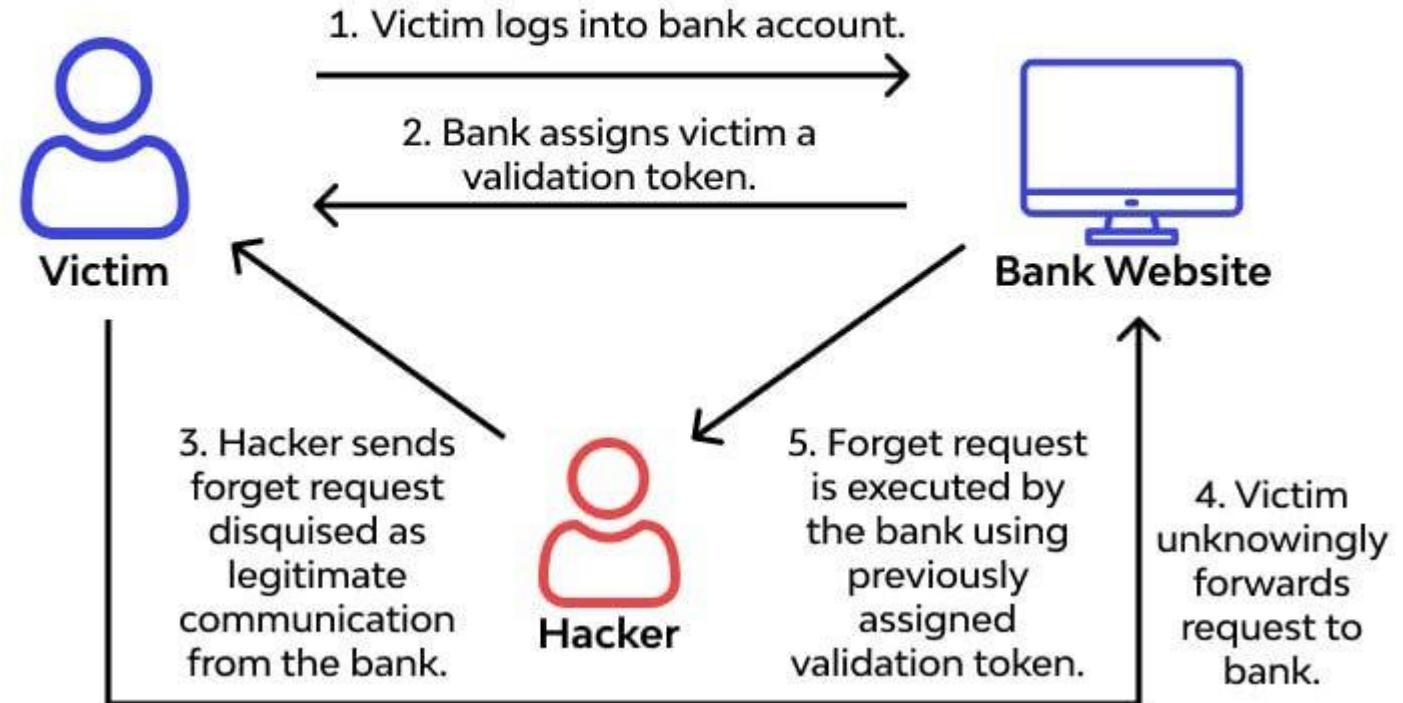
# Cross-Site Request Forgery (CSRF)

- Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated.
- CSRF attacks exploit the trust a Web application has in an authenticated user. (Conversely, cross-site scripting (XSS) attacks exploit the trust a user has in a particular Web application).
- A CSRF attack exploits a vulnerability in a Web application if it cannot differentiate between a request generated by an individual user and a request generated by a user without their consent.

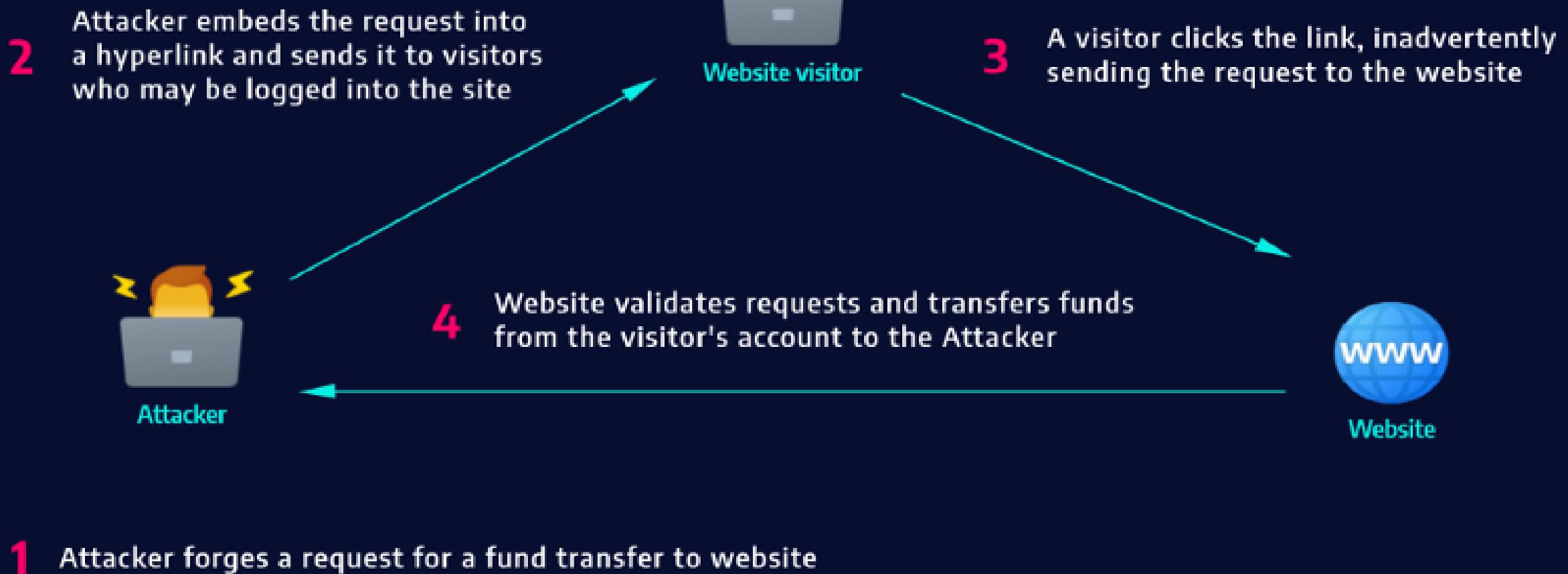
# Scenario of CSRF

- Cross site request forgery (CSRF), also known as XSRF, Sea Surf or **Session Riding**, is an attack vector that tricks a web browser into executing an unwanted action in an application to which a user is logged in.

## Cross-Site Request Forgery



CSRFs are typically conducted using malicious social engineering, such as an email or link that tricks the victim into sending a forged request to a server. As the unsuspecting user is authenticated by their application at the time of the attack, it's impossible to distinguish a legitimate request from a forged one.



- When a user tries to access a site, the browser often automatically includes the credentials in the request, to make the login process more convenient. These credentials may include the user's session cookie, basic authentication credentials, IP address, and Windows domain credentials.
- The risk inherent in this mechanism is that attackers can easily impersonate the user. Once a user passes the site's identity verification, the site cannot differentiate between a forged request and a legitimate user request.

# CSRF attack

- In a CSRF attack, an attacker assumes the victim's identity, and uses it to perform actions on behalf of the user, without their consent. Attackers typically follow this process:
  1. They use social engineering techniques to persuade the victim to click a link via email, chat message, or similar form of communication.
  2. Either the malicious link itself, or a web page the user visits, triggers a request to the targeted site
  3. The request supposedly comes from the user, and takes advantage of the fact that the user is already signed into the website.
  4. The website acknowledges the request and performs the attacker's requested action, without the user's knowledge or consent.

# What can be done once CSRF attack is successful.

- CSRF attacks typically attempt to change server state, but can also be used to gain access to sensitive data.
- If an attacker successfully performs a CSRF attack against the victim's account,
  - they can transfer funds,
  - purchase a product,
  - modify account information such as the shipping address,
  - modify the password, or
  - any other action available when the user is signed in.

# How the CSRF attack is done? Use Case 1

- Let's say that Bob has an online banking account on samplebank.com. He regularly visits this site to conduct transactions with his friend Alice. Bob is unaware that samplebank.com is vulnerable to CSRF attacks. Meanwhile, an attacker aims to transfer \$5,000 from Bob's account by exploiting this vulnerability. To successfully launch this attack:
  1. The attacker must build an exploit URL.
  2. The attacker must also trick Bob into clicking the exploit URL.
  3. Bob needs to have an active session with samplebank.com.
  - Let's say that the online banking application is built using the GET method to submit a transfer request. As such, Bob's request to transfer \$500 to Alice (with account number 213367) might look like this:
  - GET `https://samplebank.com/onlinebanking/transfer?amount=500&accountNumber=213367` HTTP/1.1

# How the CSRF attack is done? Use Case 1

- Aligning with the first requirement to successfully launch a CSRF attack, an attacker must craft a malicious URL to transfer \$5,000 to the account 425654:
- <https://samplebank.com/onlinebanking/transfer?amount=5000&accountNumber=425654>
- Using various social engineering attack methods, an attacker can trick Bob into loading the malicious URL. This can be achieved in various ways. For instance, including malicious HTML image elements onto forms, placing a malicious URL on pages that are often accessed by users while logged into the application, or by sending a malicious URL through email.
- The following is an example of a disguised URL:
- `<img src = "https://samplebank.com/onlinebanking/transfer?amount=5000&accountNumber=425654" width="0" height="0">`
- Consider the scenario that includes an image tag in an attacker-crafted email to Bob. Upon receiving it, Bob's browser application opens this URL automatically—without human intervention. As a result, without Bob's permission, a malicious request is sent to the online banking application. If Bob has an active session with samplebank.com, the application would treat this as an authorized amount transfer request coming from Bob. It would then transfer the amount to the account specified by an attacker.

# How the CSRF attack is done? Use Case 2

- Imagine your application has a /user/email route that accepts a POST request to change the authenticated user's email address.
- Most likely, this route expects an email input field to contain the email address the user would like to begin using.
- Without CSRF protection, a malicious website could create an HTML form that points to your application's /user/email route and submits the malicious user's own email address:

```
<form action="https://your-application.com/user/email"
method="POST">
<input type="email" value="malicious-
email@example.com">
</form>
<script>
document.forms[0].submit();
</script>
```

If the malicious website automatically submits the form when the page is loaded, the malicious user only needs to lure an unsuspecting user of your application to visit their website and their email address will be changed in your application.

# HOW DO I PREVENT CSRF?

1. To prevent this vulnerability, we need to inspect every incoming POST, PUT, PATCH, or DELETE request for a secret session value that the malicious application is unable to access.
2. Preventing CSRF requires the inclusion of an unpredictable token in the body or URL of each HTTP request. Such tokens should at a minimum be unique per user session, but can also be unique per request. (The easiest way to add a non-predictable parameter is to use a secure hash function (e.g., SHA-2) to hash the user's session ID. To ensure randomness, the tokens must be generated by a cryptographically secure random number generator.)
3. The preferred option is to include the unique token in a hidden field. The unique token can also be included in the URL itself, or a URL parameter.
4. Check Referrer field of each request.
5. Use Captcha on all critical page.

# CORS

Cross Origin Resource Sharing



<https://aws.amazon.com/what-is/cross-origin-resource-sharing/#:~:text=your%20CORS%20requirements%3F-,What%20is%20Cross%2DOrigin%20Resource%20Sharing%3F,reources%20in%20a%20different%20domain.>

# Cross-Origin Resource Sharing

- Cross-origin resource sharing (CORS) is a mechanism for integrating applications. CORS defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.
- This is useful because complex applications often reference third-party APIs and resources in their client-side code.
- For example, your application may use your browser to pull videos from a video platform API, use fonts from a public font library, or display weather data from a national weather database.
- CORS allows the client browser to check with the third-party servers if the request is authorized before any data transfers.

# Cross-origin resource sharing important

- In the past, when internet technologies were still new, cross-site request forgery (CSRF) issues happened. These issues sent fake client requests from the victim's browser to another application.
- For example, the victim logged into their bank's application. Then they were tricked into loading an external website on a new browser tab.
- The external website then used the victim's cookie credentials and relayed data to the bank application while pretending to be the victim. Unauthorized users then had unintended access to the bank application.
- To prevent such CSRF issues, all browsers now implement the same-origin policy.

# Same-origin policy

- Today, browsers enforce that clients can only send requests to a resource with the same origin as the client's URL. The protocol, port, and hostname of the client's URL should all match the server it requests.
- For example, consider the origin comparison for the below URLs with the client URL `http://store.aws.com/dir/page.html`.

URL	Outcome	Reason
<code>http://store.aws.com/dir2/new.htm</code> /	Same origin	Only the path differs
<code>http://store.aws.com/dir/inner/othe</code> <code>r.html</code>	Same origin	Only the path differs
<code>https://store.aws.com/page.html</code>	Different origin	Different protocol
<code>http://store.aws.com:81/dir/page.h</code> <code>tml</code>	Different origin	Different port ( <code>http://</code> is port 80 by default)
<code>http://news.aws.com/dir/page.htm</code> /	Different origin	Different host

- So, the same-origin policy is highly secure but inflexible for genuine use cases.
- Cross-origin resource sharing (CORS) is an extension of the same-origin policy. You need it for authorized resource sharing with external third parties. For example, you need CORS when you want to pull data from external APIs that are public or authorized. You also need CORS if you want to allow authorized third-party access to your own server resources.

# How does cross-origin resource sharing work?

- In standard internet communication, your browser sends an HTTP request to the application server, receives data as an HTTP response, and displays it. In browser terminology, the current browser URL is called the current origin and the third-party URL is cross-origin.
- When you make a cross-origin request, this is the request-response process:
  - The browser adds an origin header to the request with information about the current origin's protocol, host, and port
  - The server checks the current origin header and responds with the requested data and an Access-Control-Allow-Origin header
  - The browser sees the access control request headers and shares the returned data with the client application
  - Alternatively, if the server doesn't want to allow cross-origin access, it responds with an error message.

# Cross-origin resource sharing example

- For example, consider a site called <https://news.example.com>. This site wants to access resources from an API at [partner-api.com](https://partner-api.com).
- Developers at <https://partner-api.com> first configure the cross-origin resource sharing (CORS) headers on their server by adding [new.example.com](https://news.example.com) to the allowed origins list. They do this by adding the below line to their server configuration file.

```
Access-Control-Allow-Origin: https://news.example.com
```
- Once CORS access is configured, [news.example.com](https://news.example.com) can request resources from [partner-api.com](https://partner-api.com). For every request, [partner-api.com](https://partner-api.com) will respond with Access-Control-Allow-Credentials : "true." The browser then knows the communication is authorized and permits cross-origin access.
- If you want grant access to multiple origins, use a comma-separated list or wildcard characters like \* that grant access to everyone.

# What is a CORS preflight request?

- In HTTP, request methods are the data operations the client wants the server to perform. Common HTTP methods include GET, POST, PUT, and DELETE.
- In a regular cross-origin resource sharing (CORS) interaction, the browser sends the request and access control headers at the same time. These are usually GET data requests and are considered low-risk.
- However, some HTTP requests are considered complex and require server confirmation before the actual request is sent. The preapproval process is called **preflight** request.

# Complex cross-origin requests

- **Cross-origin requests are complex if they use any of the following:**
- Methods other than GET, POST, or HEAD
- Headers other than Accept-Language, Accept, or Content-Language
- Content-Type headers other than multipart/form-data, application/x-www-form-urlencoded, or text/plain
- So, for example, requests to delete or modify existing data are considered complex.

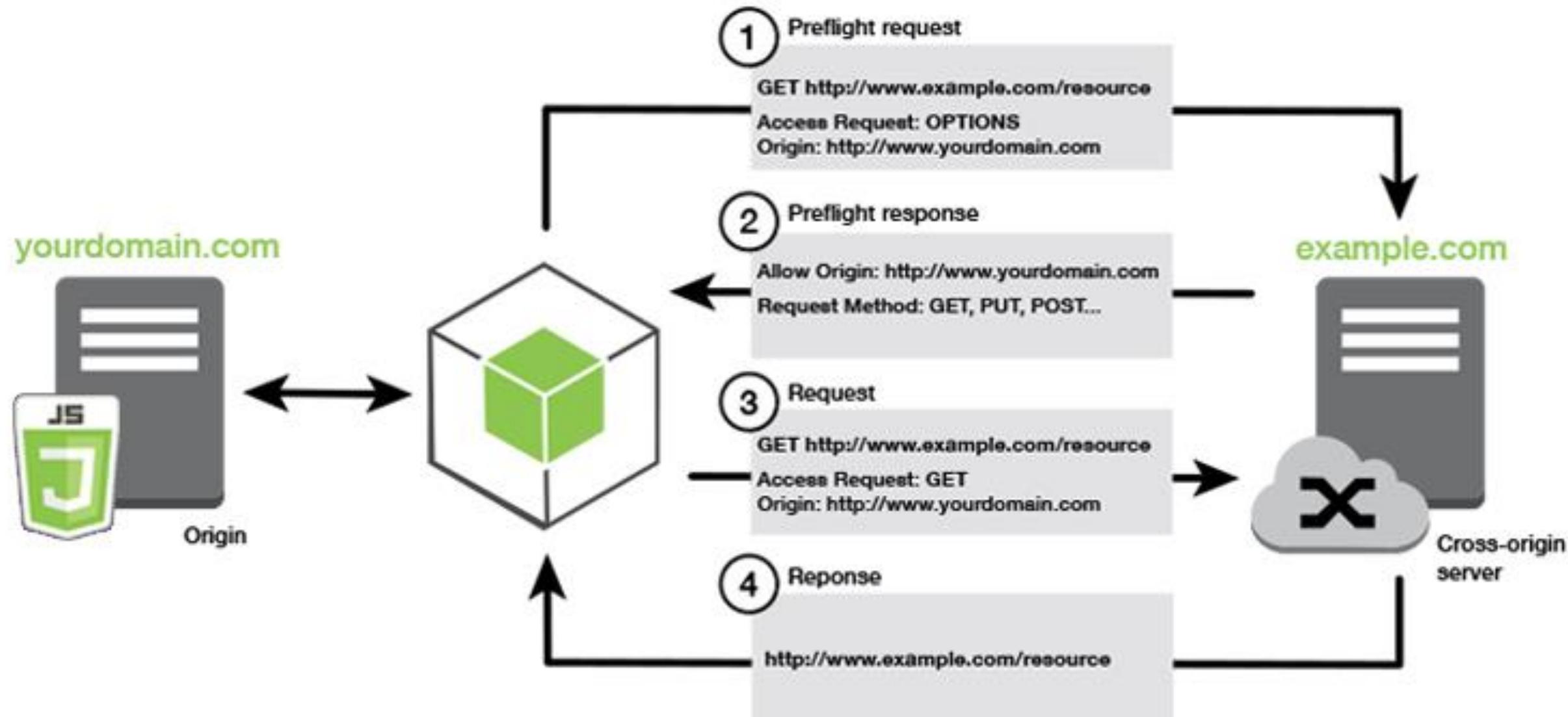
# How preflight requests work

- Browsers create preflight requests if they are needed. It's an OPTIONS request like the following one.
  - `OPTIONS /data HTTP/1.1`
  - `Origin: https://example.com`
  - `Access-Control-Request-Method: DELETE`
- The browser sends the preflight request before the actual request message. The server must respond to the preflight request with information about the cross-origin requests the server's willing to accept from the client URL. The server response headers must include the following:
  - `Access-Control-Allow-Methods`
  - `Access-Control-Allow-Headers`
  - `Access-Control-Allow-Origin`

# An example server response is given below. An

- HTTP/1.1 200 OK
- Access-Control-Allow-Headers: Content-Type
- Access-Control-Allow-Origin: <https://news.example.com>
- Access-Control-Allow-Methods: GET, DELETE, HEAD, OPTIONS
- The preflight response sometimes includes an additional Access-Control-Max-Age header. This metric specifies the duration (in seconds) for the browser to cache preflight results in the browser. Caching allows the browser to send several complex requests between preflight requests. It doesn't have to send another preflight request until the time specified by max-age elapses.

# An example server response is given below.



# What are some CORS best practices?

- You should note the following when you configure cross-origin resource sharing (CORS) on your server.
- **Define appropriate access lists**
  - It is always best to grant access to individual domains using comma-separated lists. Avoid using wildcards unless you want to make the API public. Otherwise, using wildcards and regular expressions may create vulnerabilities.
  - For example, let's say you write a regular expression that grants access to all sites with the suffix permitted-website.com. With one expression, you grant access to api.permitted-website.com and news.permitted-website.com. But you also inadvertently grant access to unauthorized sites that may use domains like maliciouspermitted-website.com.
- **Avoid using null origin in your list**
  - Some browsers send the value null in the request header for certain scenarios like file requests or requests from the local host.
  - However, you shouldn't include the null value in your access list. It also introduces security risks as unauthorized requests containing null headers may get access.

# Clickjacking

- Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website.
- Clickjacking is an attack that fools users into thinking they are clicking on one thing when they are actually clicking on another. Its other name, user interface (UI) redressing, better describes what is going on. Users think they are using a web page's normal UI, but in fact there is a hidden UI in control; in other words, the UI has been redressed. When users click something they think is safe, the hidden UI performs a different action.

# Consider the following example:

- Typically, clickjacking is performed by displaying an invisible page or HTML element, inside an iframe, on top of the page the user sees. The user believes they are clicking the visible page but in fact they are clicking an invisible element in the additional page transposed on top of it.
- The invisible page could be a [malicious page](#), or a legitimate page the user did not intend to visit – for example, a page on the user’s banking site that authorizes the transfer of money.
- There are several variations of the clickjacking attack, such as:
- **Likejacking** – a technique in which the Facebook “Like” button is manipulated, causing users to “like” a page they actually did not intend to like.
- **Cursorjacking** – a UI redressing technique that changes the cursor for the position the user perceives to another position. Cursorjacking relies on vulnerabilities in Flash and the Firefox browser, which have now been fixed.

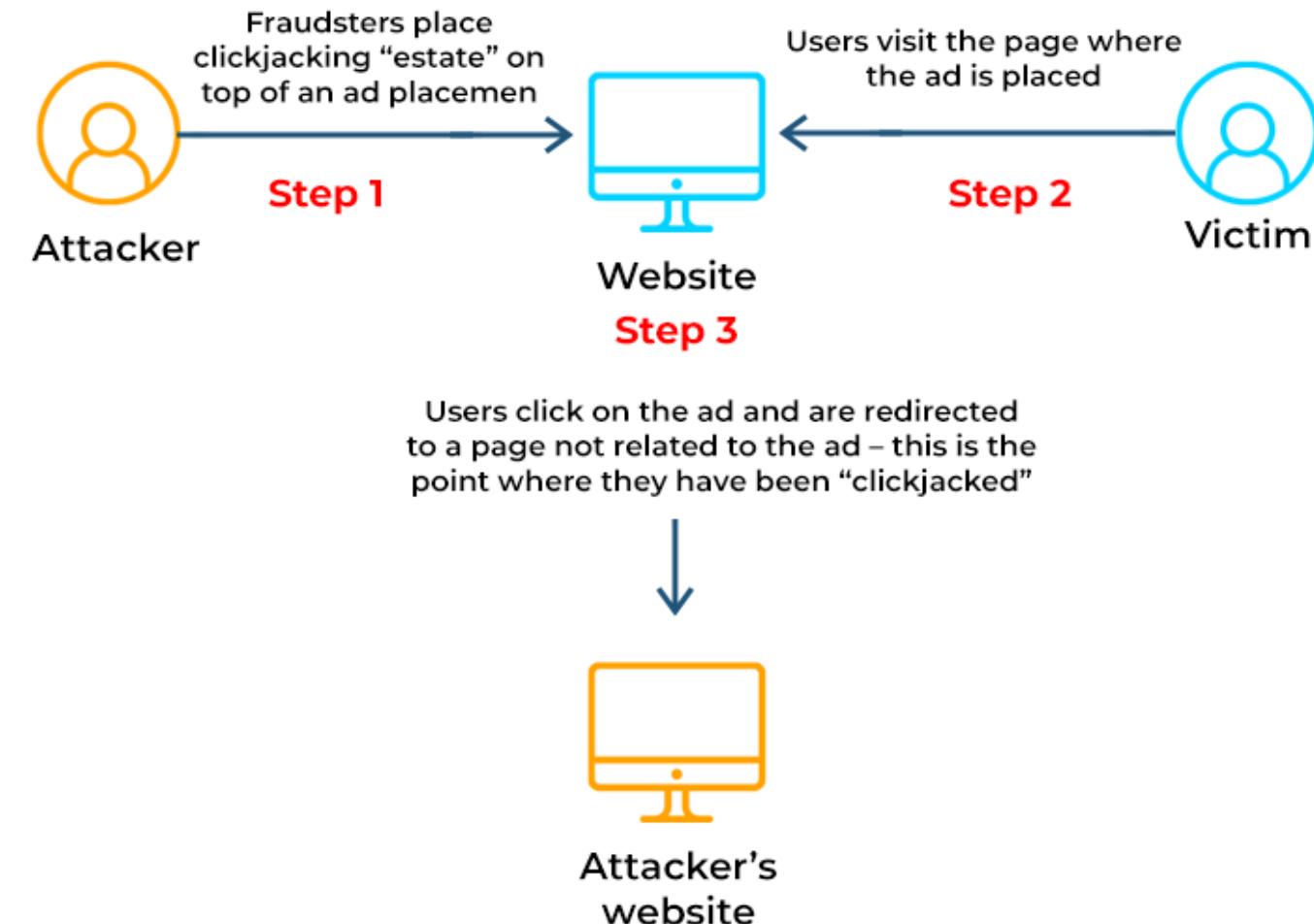
# Clickjacking attack Use Case 1

1. The attacker creates an attractive page which promises to give the user a free trip to Tahiti.
2. In the background the attacker checks if the user is logged into his banking site and if so, loads the screen that enables transfer of funds, using query parameters to insert the attacker's bank details into the form.
3. The bank transfer page is displayed in an invisible iframe above the free gift page, with the “Confirm Transfer” button exactly aligned over the “Receive Gift” button visible to the user.
4. The user visits the page and clicks the “Book My Free Trip” button.
5. In reality the user is clicking on the invisible iframe, and has clicked the “Confirm Transfer” button. Funds are transferred to the attacker.
6. The user is redirected to a page with information about the free gift (not knowing what happened in the background).

# Use Case 2

- This example illustrates that, in a clickjacking attack, the malicious action (on the bank website, in this case) cannot be traced back to the attacker because the user performed it while being legitimately signed into their own account.

## HOW CLICKJACKING WORKS



# How to construct a basic clickjacking attack

- Clickjacking attacks use CSS to create and manipulate layers.
- The attacker incorporates the target website as an iframe layer overlaid on the decoy website.
- An example using the style tag and parameters is shown.

```
<head>
    <style>
        #target_website {
            position:relative;
            width:128px;
            height:128px;
            opacity:0.00001;
            z-index:2;
        }
        #decoy_website {
            position:absolute;
            width:300px;
            height:400px;
            z-index:1;
        }
    </style>
</head>
...
<body>
    <div id="decoy_website">
        ...decoy web content here...
    </div>
    <iframe id="target_website" src="https://vulnerable-website.com">
    </iframe>
</body>
```

# How to construct a basic clickjacking attack

- The target website iframe is positioned within the browser so that there is a precise overlap of the target action with the decoy website using appropriate width and height position values.
- Absolute and relative position values are used to ensure that the target website accurately overlaps the decoy regardless of screen size, browser type and platform.
- The z-index determines the stacking order of the iframe and website layers.
- The opacity value is defined as 0.0 (or close to 0.0) so that the iframe content is transparent to the user.
- Browser clickjacking protection might apply threshold-based iframe transparency detection (for example, Chrome version 76 includes this behavior but Firefox does not).
- The attacker selects opacity values so that the desired effect is achieved without triggering protection behaviors.

# Clickjacking with prefilled form input

- Some websites that require form completion and submission permit prepopulation of form inputs using GET parameters prior to submission.
- Other websites might require text before form submission.
- As GET values form part of the URL then the target URL can be modified to incorporate values of the attacker's choosing and the transparent "submit" button is overlaid on the decoy site as in the basic clickjacking example.

## METHODS USED FOR CLICKJACKING



# 1. Cursorjacking

- Cursorjacking is a technique for manipulating the cursor on your computer screen. It shifts from where you think it is to a different location.
- This method is most commonly used to attack Adobe Flash and Firefox flaws. These issues, fortunately, have been resolved.
- Cursorjacking misleads visitors by displaying the pointer with an offset on a modified cursor picture. Compared to the actual cursor location, the projected cursor is displaced to the right.
- For example, when the user moves their mouse, the event triggers the listener, which causes the false mouse pointer (the visible one) to move.
- When you select the YES button, you will be taken to the Tweet page. Hackers initially used this method to get around NoScript's ClearClick Security.

## 2. Browserless clickjacking

- Browserless clickjacking is a method for replicating traditional clickjacking in programs that do not operate in an internet browser.
- Because of how pop-up alerts work, browserless clickjacking is common on Android smartphones.
- Between the time the notice is issued and the time it appears on the screen, there is a short delay with pop-up alerts.
- The attackers take advantage of the brief delay to build a dummy icon that you can touch beneath the real notification.

### 3. Password manager attack

- Particular security features autofill credentials unsafely for the HTTP edition of HTTPS-saved certificates.
- These password managers also fill in the information in iFrames, known as a password manager assault. When password sync was used on several machines, many password managers did not defend against iFrame and redirection-based threats, exposing other passwords.
- Traditional browsers, unlike password managers, are secure and do not autofill information.
- If the protocol deployed on the existing login page varied from the protocol applied when users stored passwords, browsers would not share autofill information. In iFrames, neither browsers nor data autofill.

## 4. CookieJacking

- CookieJacking is a type of clickjacking wherein the user's web address bar cookies are taken.
- This is done by duping the user into executing a seemingly harmless operation on the malicious site (typically dragging an item). When users accept the activity, they unwittingly select and transmit the cookie data to the hacker.
- After that, the attacker can employ a CSRF attack to impersonate the victim on the website. In this technique, a hacker steals the cookies from your browser.
- Typically, victims are asked to move an innocuous item, but the attacker copies the full content of their cookies and gives it to them.

# 5. Mousejack

- Mousejack is a set of flaws that may impact most non-Bluetooth ergonomic keyboards and mice. A broadcast transceiver, usually a tiny USB dongle, is used to ‘link’ these devices to a host device. Because the interface is wireless and cursor and keystrokes are transmitted over the sky, it is feasible to attack a victim’s computer by sending specially-crafted radio waves with a device that costs little than \$15.
- An attacker can target devices from a distance of around 100 meters. Without visually being next to a targeted device, the attacker can take command of it and write arbitrary text or execute preset commands. As a result, it is feasible to carry out hostile acts quickly while remaining undetected. The mousejack attack involves sending decrypted keystrokes to a target machine.
- Most mouse motions are sent unsecured, but keystrokes are frequently encrypted (to avoid eavesdropping that is being entered). The mousejack flaw, on the other hand, exploits affected recipient dongles and their accompanying software, enabling an attacker’s decrypted keystrokes to be forwarded on to the windows OS as if the user had written them properly.

# 6. Filejacking

- Filejacking is a method in which attackers take advantage of the web browser's capacity to traverse through a machine and access its files to steal personal information and compromise data security.
- This is accomplished by persuading the user to create an operational file server using the browser's folder choice interface.
- Assailants use this technique to access their targets' computers and steal files and private information.

# 7. Nested clickjacking

- For nested clickjacking to work, a malicious web window must be inserted between two frames of the original, benign web page. These frames are referred to as the enclosed page and the upper window page.
- A flaw in the X-Frame-Options HTTP header allows nested clickjacking to occur. When the X-Frame-Options signal is set to SAMEORIGIN, the browser only examines the two innocent layers. It ignores the harmful layer in the middle, allowing attackers to take advantage of the flaw.

## 8. Likejacking

- Likejacking is a type of clickjacking fraud that uses the Facebook “like” feature. Scammers use the “like” button to post an enticing video, picture, or coupon deal.
- This spreads the scam by posting the discount to all users’ acquaintances’ Facebook pages.
- The more “likes” the post receives, the more it circulates. Although the impact of these frauds is unknown, some online security experts suspect scammers are attempting to log in to a Facebook profile or private information of victims.

# How to prevent clickjacking attacks

- Clickjacking is a browser-side behavior and its success or otherwise depends upon browser functionality and conformity to prevailing web standards and best practice.
- Server-side protection against clickjacking is provided by defining and communicating constraints over the use of components such as iframes.
- However, implementation of protection depends upon browser compliance and enforcement of these constraints.
- Two mechanisms for server-side clickjacking protection are
  1. X-Frame-Options and
  2. [Content Security Policy](#).

# Defenses against clickjacking: X-Frame-Options

- ## X-Frame-Options?

- use the X-Frame-Options HTTP header. It allows an application to specify whether frame use is simply denied, via the DENY value, or the use of frames is allowed, by the SAMEORIGIN or ALLOW-FROM values. Mainstream modern browsers do support this header option, but other browsers may not.

- Possible X-Frame-Options:

- X-Frame-Options: DENY
- X-Frame-Options: SAMEORIGIN
- X-Frame-Options: ALLOW-FROM <https://example.com/>

# Defenses against clickjacking: Content Security Policy (CSP)

- Use Content Security Policy (CSP) and its frame-ancestors directive. This directive allows the application developer to disallow all frame use or specify where it is allowed, similar to X-Frame-Options. CSP is not available in all browsers, and browser plugins and add-ons may be able to bypass the policy. If both the X-Frame-Options header and CSP frame-ancestors are used, browsers are supposed to prefer CSP's directives, but not all will.
- Possible CSP frame-ancestor settings:
  - Content-Security-Policy: frame-ancestors 'none'
  - Content-Security-Policy: frame-ancestors 'self'
  - Content-Security-Policy: frame-ancestors example.com

# WEB APP SECURITY



```
render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product">
            <div className="row">
              <ProductConsumer>
                {(value) => {
                  console.log(value)
                }}
              </ProductConsumer>
            </div>
          </div>
        </div>
      <React.Fragment>
```

## Course Outcomes

**CO1** Apply client-side web development to design interactive front-end web user interfaces.

**CO2** Use server-side web application concepts to develop back-end web server application

**CO3** Identify and mitigate various client-side web application security vulnerabilities

**CO4** Identify and mitigate various server-side web application security vulnerabilities

# Syllabus

- Web application development – Introduction - Architecture – Client-side technologies and frameworks – HTML – CSS – Javascript - Ajax/Fetch - Data interchange formats – XML, JSON. Server-side scripting and technologies - development – technologies - Handling client requests – Database connectivity – Sessions – Cookies.
- Web application vulnerabilities – Client-side Vulnerabilities - Cross Site Scripting (XSS) - Cross Site Request Forgery (CSRF) - Cross-origin resource sharing (CORS) - Clickjacking. Server-side Vulnerabilities - SQL injection - OS command injection - Directory traversal - Authentication - Server-side request forgery (SSRF)
- **Textbook(s)**

# Textbook and Reference books

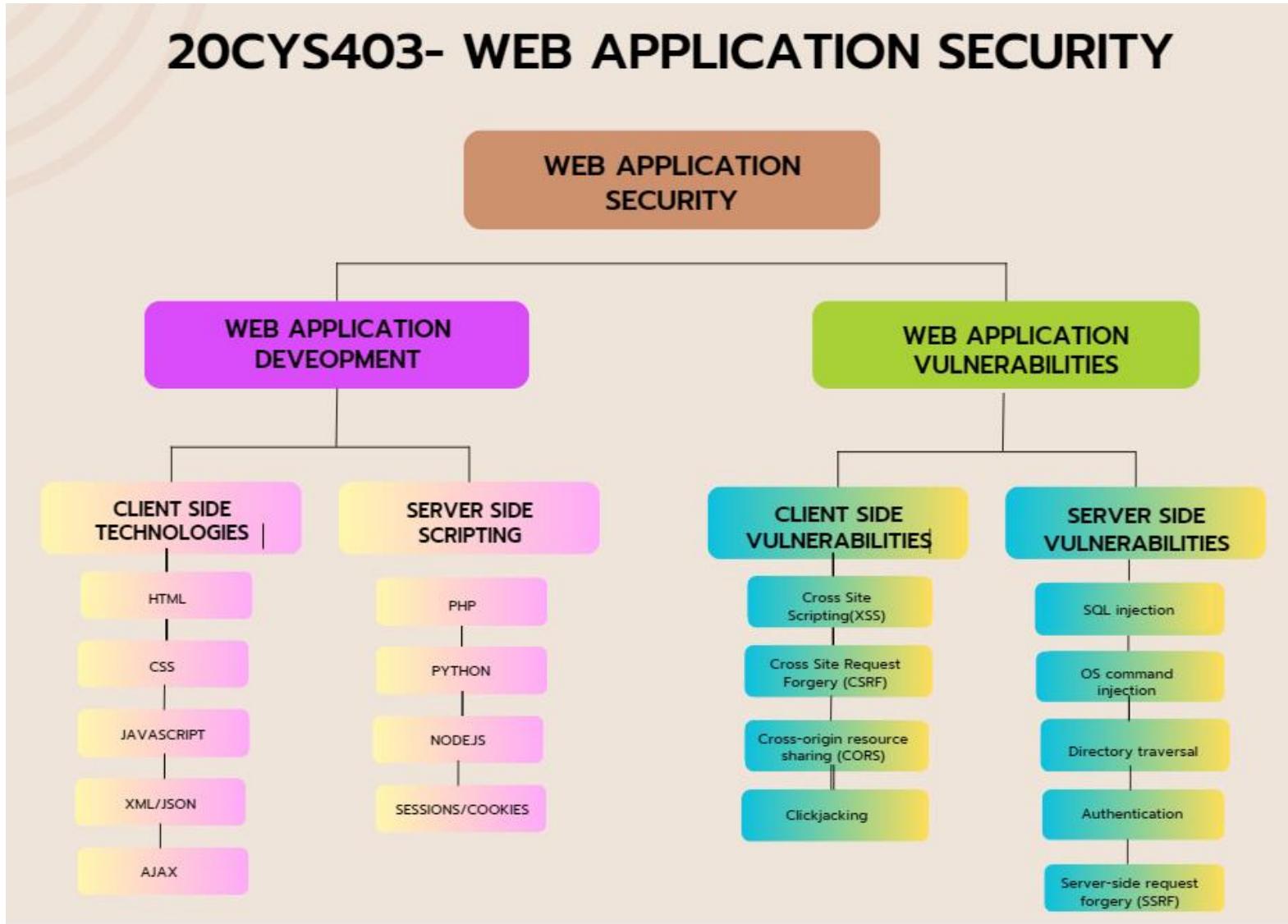
- **Textbook(s)**

1. Robin Nixon, Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites, Fifth Edition, O'Reilly Media, Inc.; 2018.
2. Dafydd Stuttard, and Marcus Pinto, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition, John Wiley & Sons; 2011

- **Reference(s)**

1. John Paul Mueller, Security for Web Developers: Using Javascript, Html, and CSS, O'Reilly, 2015
2. Andrew Hoffman. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Shroff Publishers & Distributors Pvt. Ltd, 2020.
3. Richa Gupta, Hands-on Penetration Testing for Web Applications, BPB Publications (29 March 2021)
4. Ivan Ristic, Bulletproof TLS and PKI, Second Edition: Understanding and deploying SSL/TLS and PKI to secure servers and web applications, Feisty Duck Ltd; 2nd edition, 2022
5. <https://web.stanford.edu/class/cs253/>
6. <https://nptel.ac.in/courses/128106006>
7. <https://www.rapid7.com/fundamentals/web-application-security/>
8. <https://owasp.org/>
9. <https://www.udemy.com/course/web-application-security/>

# Concept Map



# SQL Injection Attack



[www.educba.com](http://www.educba.com)

- Whenever the application interacts with the database server and requests for some data then the attackers may interfere in between and get access to those things and the data that is being retrieved or being sent from and to the database server is called an **Injection attack in SQL**.
- **What can SQL Injection do?**
  - Bypass a web application's authorization mechanisms and extract sensitive information
  - Easily control application behavior that's based on data in the database
  - Inject further malicious code to be executed when users access the application
  - Add, modify and delete data, corrupting the database, and making the application or unusable
  - Enumerate the authentication details of a user registered on a website and use the data in attacks on other sites

# Non-Technical Explanation of the SQL Injection Vulnerability

- Imagine a fully-automated bus that functions based on instructions given by humans through a standard web form. That form might look like this:
- Drive through <route> and <where should the bus stop?> if <when should the bus stop?>.
- **Sample Populated Form**
- Drive through **route 66** and **stop at bus stops** if **there are people at the bus stops**.
- Values in bold are provided by humans and instruct the bus. Imagine a scenario where someone manages to send these instructions:
- Drive through **route 66** and **do not stop at bus stops** and **ignore the rest of this form**. if there are people at the bus stops.
- The bus is fully automated. It does exactly as instructed: it drives up route 66 and does not stop at any bus stop, even when there are people waiting. Such an injection is possible because the query structure and the supplied data are not separated correctly. The automated bus does not differentiate between instructions and data; it simply parses anything it is fed.
- SQL injection vulnerabilities are based on the same concept. Attackers are able to inject malicious instructions into benign ones, all of which are then sent to the database server through a web application.

# Technical Explanation of SQL Injection Vulnerability

- As the name suggests, an SQL injection vulnerability allows an attacker to inject malicious input into an SQL statement.
- To fully understand the issue, first have to understand how server-side scripting languages handle SQL queries.
- For example, let's say functionality in the web application generates a string with the following SQL statement:
  - `$statement = "SELECT * FROM users WHERE username = 'bob' AND password = 'mysecretpw'"`;
  - This SQL statement is passed to a function that sends the string to the connected database where it is parsed, executed and returns a result.

# Notice the statement contains some new, special characters:

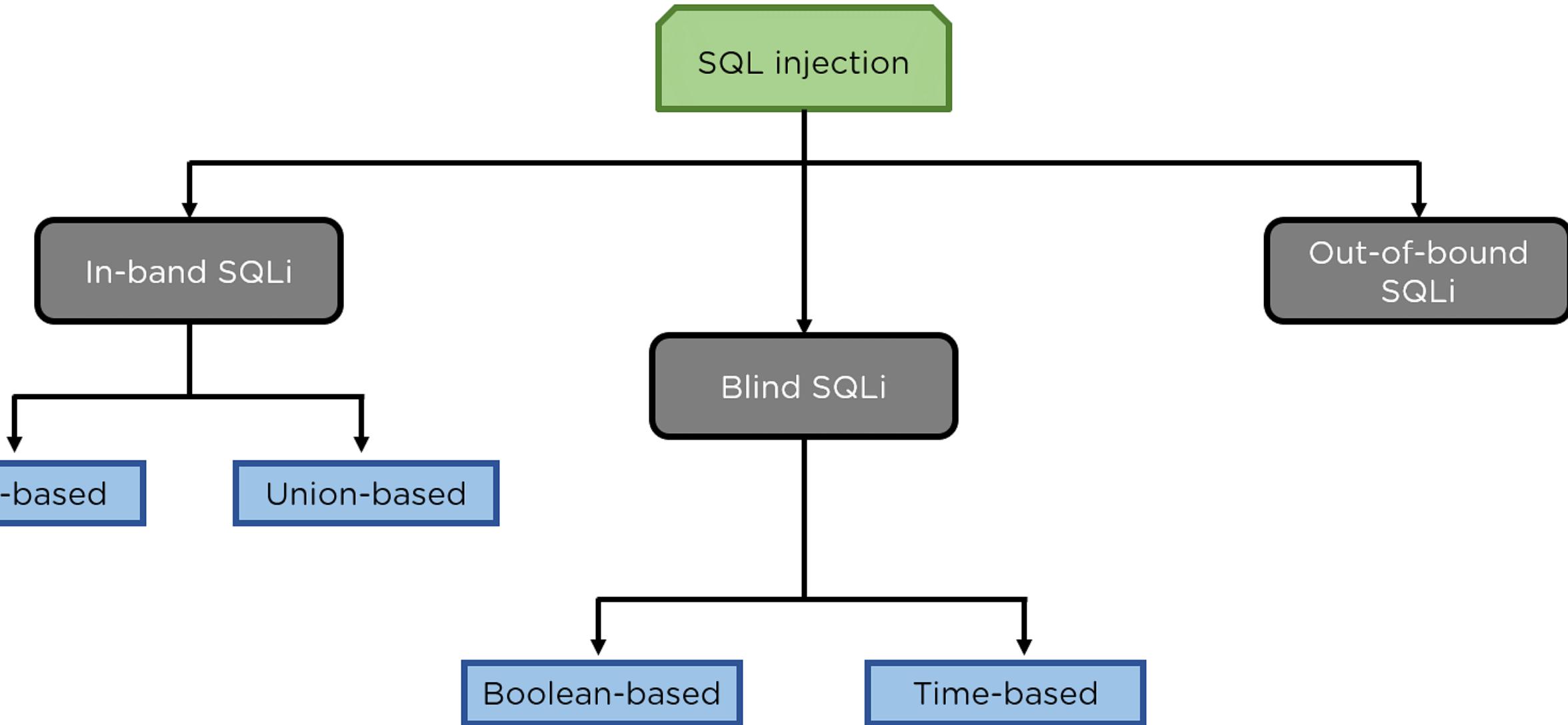
- As you might have noticed the statement contains some new, special characters:
  - a) \* (asterisk) is an instruction for the SQL database to return all columns for the selected database row
  - b) = (equals) is an instruction for the SQL database to only return values that match the searched string
  - c) ' (single quote mark) is used to tell the SQL database where the search string starts or ends

# Modified SQL

- Now consider the following example in which a website user is able to change the values of '\$user' and '\$password', such as in a login form:
- `$statement = "SELECT * FROM users WHERE username = '$user' AND password = '$password"';`
- An attacker can easily insert any special SQL syntax inside the statement, if the input is not sanitized by the application:
- `$statement = "SELECT * FROM users WHERE username = 'admin'; -- ' AND password = 'anything"'; = 'anything"';`
- What is happening here? The green part ('admin'; --) is the attacker's input, which contains two new, special characters:
  - ; (semicolon) is used to instruct the SQL parser that the current statement has ended (not necessary in most cases)
  - (double hyphen) instructs the SQL parser that the rest of the line (shown in light grey above) is a comment and should not be executed
- This SQL injection effectively removes the password verification, and returns a dataset for an existing user – 'admin' in this case. The attacker can now log in with an administrator account, without having to specify a password.

# The Different Types of SQL Injection Vulnerability

- Attackers can exfiltrate data from servers by exploiting SQL Injection vulnerabilities in various ways.
- Common methods include retrieving data based on:
  - errors,
  - conditions (true/false) and
  - timing .



# In-band SQL Injection

- It is the most common SQL Injection attack. Usually occurs when an attacker is able to use the same communication channel to both launch the attack and gather results. The two most common types of in-band SQL Injection are:
  - **Error-based SQL Injection** – It is a technique that relies on error messages thrown by the database server to obtain information about the structure of the database. Sometimes, this simple attack is more than enough for an attacker to enumerate an entire database.
  - **Union-based SQL Injection** – This technique leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.

# Example of Error-based SQL Injections:

- Adding SQL syntax to user input: [Source: https://www.geeksforgeeks.org/error-based-sql-injections/](https://www.geeksforgeeks.org/error-based-sql-injections/)
- In this SQL injection, a hacker inserts a malicious query to get an error that displays a message containing sensitive information about the database.
- A hacker might try writing a SQL command in any input field like a single quote, double-quote, or any other SQL operator like OR, AND, NOT.
- *For Example, for a URL of a site that takes a parameter from the user,*
- *then in that case: <https://www.example.org/index.php?item=123>*
- *Then here attacker can try inserting any SQL command or operator in the passes value,*
- *as: ['https://www.example.org/index.php?item=123'](https://www.example.org/index.php?item=123)*

# Example of Error-based SQL Injections .....

- In this case, a database could return some error like this, If you have an error in your SQL syntax, check the manual corresponding to your [MySQL](#) server version for the right syntax to use near “VALUE.”
- This message gives the attacker information like the database used in SQL, the syntax that caused an error, and where the syntax occurred in the query.
- For a professional hacker with experience, this will be enough to tell him that the server is insecurely connected to a database and can plan additional SQL injection attacks that will cause damage.
- An attacker can try several queries using commands like grep extract in input fields and see adding which commands return an error.

# Union-based SQL Injection:

- Union-based SQL injection involves the use of the UNION operator that combines the results of multiple SELECT statements to fetch data from multiple tables as a single result set. The malicious UNION operator query can be sent to the database via website URL or user input field.

```
https://sqlil.com/users/id=geek 'UNION SELECT * FROM users, courses --
```

- The above URL contains a malicious SQL query that can fetch records of all users. “–” at the end ignores all subsequent statements.

- Source:<https://www.geeksforgeeks.org/what-is-sql-injection-union-attacks/>

# Inferential SQL Injection (Blind SQLi)

- In this type of injection, no data is actually transferred via the web application. So, the attacker will not be able to see the result of an attack. Here, attacker reconstructs the database structure by sending payloads, observing the web application's response and the resulting behavior of the database server. The two types of inferential SQL Injection are:
  - **Boolean-based SQL Injection** – In this technique application is forced to return a different result depending on whether the query returns a TRUE or FALSE result. Based on the result, the content within the HTTP response will change, or remain the same.
  - **Time-based SQL Injection** – It is a technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The time website takes to respond will indicate to the attacker whether the result of the query is TRUE or FALSE.

# Blind SQL Injection

- A blind SQL Injection attack comes into the picture when the targeted web application is vulnerable to SQL Injection, but the twist is that the HTTP responses don't contain any database results or errors which are supposed to be retrieved as per the query.
  - Blind SQL Injection can be used to get sensitive data from the database servers. Most probably, the hacker asks true or false (1 or 0) queries to the application database and studies those responses based upon the answers of applications.
- 
- Source: <https://www.geeksforgeeks.org/sqlbit-automatize-boolean-based-blind-sql-injections/>

# For Example:

- `http://geeksforgeeks.org/items.php?id=2`
- Application transfers the following query to the database:
  - `SELECT title, description, body FROM items WHERE ID = 2`
- The hacker may then try to inject a malicious query that returns ‘false’:
  - `http://geeksforgeeks.org/items.php?id=2 and 1=2`
  - Now the SQL query should look like this:
    - `SELECT title, description, body FROM items WHERE ID = 2 and 1=2`
  - If the web application is vulnerable to SQL Injection, then it probably will not return anything. To make sure, the hacker will inject a query that will return ‘true’:
    - `http://geeksforgeeks.org/items.php?id=2 and 1=1`

# Out-of-band SQL Injection

- These types of SQL Injection attacks are the least common and generally the most difficult to execute. They usually involve sending the data directly from the database server to a machine that is controlled by the attacker. Out-of-band techniques offer the attacker an alternative to In-band or Blind SQL Injection attacks, especially if the server responses are not very stable.
- So, server-scripting languages are not able to determine if or not the SQL query string is malformed. All that they can do is send a string to the database server and wait for the interpreted response.

# How can SQL Injection be prevented?

- Discover SQL Injection vulnerabilities by routinely testing applications both using static testing and dynamic testing
- Avoid and repair injection vulnerabilities by using parameterized queries and Object Relational Mappers (ORMs). This types of queries specify placeholders for parameters so that the database will always treat them as data rather than part of a SQL command.
- Remediate SQL Injection vulnerabilities by using escape characters so that special characters are ignored.
- Mitigate the impact of SQL Injection vulnerabilities by enforcing least privilege on the database, this way each software component of an application can access and affect only the resources it needs.
- Use a Web Application Firewall (WAF) for web applications that access databases. This can help identify SQL injection attempts and sometimes help prevent SQL injection attempts from reaching the application as well.



Source: <https://www.acunetix.com/blog/web-security-zone/os-command-injection/>

# OS command injection

- OS command injection (operating system command injection or simply command injection) is a type of an injection vulnerability.
- The payload injected by the attacker is executed as operating system commands.
- OS command injection attacks are possible only if the web application code includes operating system calls and user input is used in the call.
- They are not language-specific – command injection vulnerabilities may appear in all languages that let you call a system shell command: C, Java, PHP, Perl, Ruby, Python, and more.

# Working of OS command injection

- The operating system executes the injected arbitrary commands with the privileges of the web server.
- Therefore, command injection vulnerabilities on their own do not lead to full system compromise.
- However, attackers may be able to use privilege escalation and other vulnerabilities to gain more access.

**Note:** Command injection is often confused with [code injection](#). Code injection vulnerabilities let the attacker inject code in the programming language in which the web application is built.

# Command Injection Example

- The developer of the example PHP application wants the user to be able to see the output of the Windows ping command in the web application.
- The user needs to input the IP address and the application sends ICMP pings to that address.
- Unfortunately, the developer trusts the user too much and does not perform input validation.
- The IP address is passed using the **GET** method and then used in the command line.

```
<?php
$address = $_GET["address"];
$output = shell_exec("ping -n 3 $address");
echo "<pre>$output</pre>";
?>
```

The attacker abuses this script by manipulating the **GET** request with the following payload:

<http://example.com/ping.php?address=8.8.8.8%26dir>

# Command Injection Example

- The `shell_exec` function executes the following OS command:
  - `ping -n 3 8.8.8.8&dir.`
  - The `&` symbol in Windows separates OS commands.
  - As a result, the vulnerable application executes an additional command `(dir)` and displays the command output (directory listing) on-screen:

Pinging 8.8.8.8 with 32 bytes of data:

Reply from 8.8.8.8: bytes=32 time=30ms TTL=56

Reply from 8.8.8.8: bytes=32 time=35ms TTL=56

Reply from 8.8.8.8: bytes=32 time=35ms TTL=56

Ping statistics for 8.8.8.8:

Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 30ms, Maximum = 35ms, Average = 33ms

Volume in drive C is OS

Volume Serial Number is 1337-8055

Directory of C:\Users\Noob\www

(...)

# Command Injection Special Characters

- You can use different special characters to inject an arbitrary command. The simplest and most common one for Linux is the semicolon (;) and for Windows, the ampersand (&). However, the following payloads for the ping.php script will also work:
  - address=8.8.8.8%3Bwhoami (; character, Linux only)
  - address=8.8.8.8&26whoami (& character, Windows only)
  - address=8.8.8.8%7Cwhoami (| character)
  - address=invalid%7C%7Cwhoami (|| characters, the second command is executed only if the first command fails)
  - address=8.8.8.8&26&26whoami (&& characters)
  - %3E(whoami) (> character, Linux only)
  - %60whoami%60 (` character, Linux only, the result will be reported by the ping command as an error)

# Command Injection Methods

- **Arbitrary command injection**
- Some applications may enable users to run arbitrary commands, and run these commands as is to the underlying host.
- **Arbitrary file uploads**
- If an application allows users to upload files with arbitrary file extensions, these files could include malicious commands. On most web servers, placing such files in the webroot will result in command injection.
- **Insecure serialization**
- Server-side code is typically used to deserialize user inputs. If deserialization is performed without proper verification, it can result in command injection.
- **Server-side template injection (SSTI)**
- Many web applications use server-side templates to generate dynamic HTML responses. This makes it possible for attackers to insert malicious server-side templates. SSTI occurs when user input is embedded in a template in an insecure manner, and code is executed remotely on the server.
- **XML external entity injection (XXE)**
- XXE occurs in applications that use a poorly-configured XML parser to parse user-controlled XML input. This vulnerability can cause exposure of sensitive data, server-side request forgery (SSRF), or denial of service attacks.

# What kind of damage can OS Command injections cause?

- An attacker successfully exploiting an OS Command injection vulnerability could:
  1. Infiltrate your local network
  2. Access sensitive data
  3. Upload or download certain data or malware
  4. Create custom scripts
  5. Run those scripts or other applications as administrators
  6. Edit user security levels

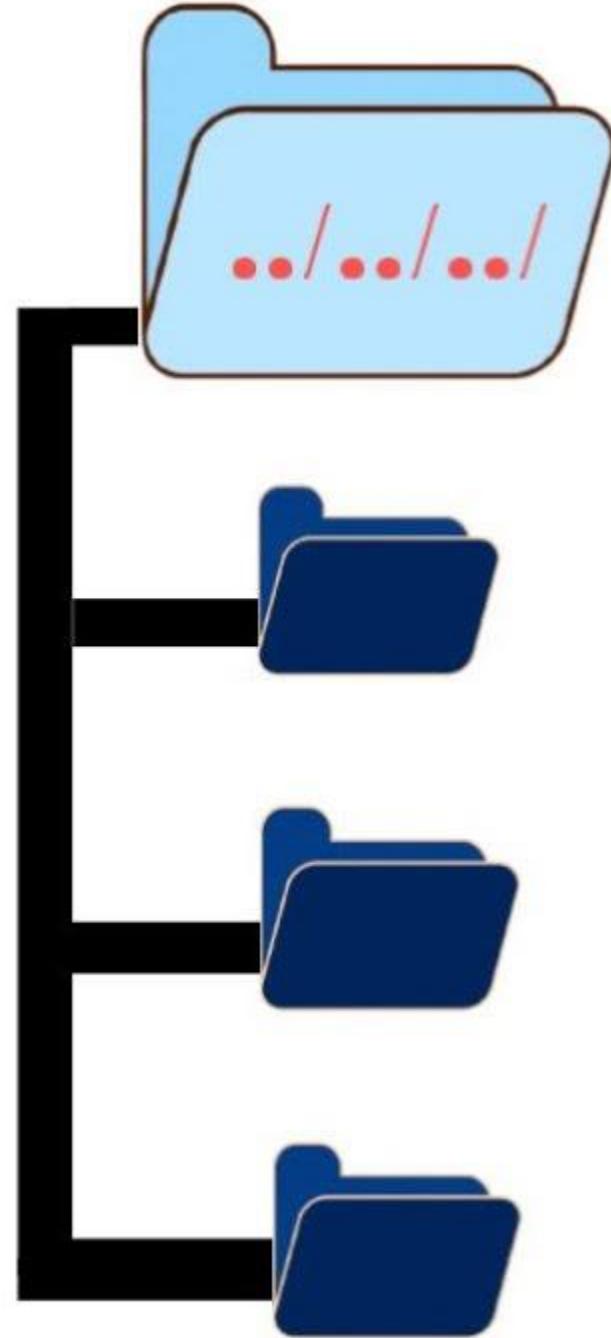
# Command Injection Prevention

- **Avoid system calls and user input**—to prevent threat actors from inserting characters into the OS command.
- **Set up input validation**—to prevent attacks like XSS and SQL Injection.
- **Create a white list**—of possible inputs, to ensure the system accepts only pre-approved inputs.
- **Use only secure APIs**—when executing system commands such as execFile()
- **Use execFile() securely**—prevent users from gaining control over the name of the program. You should also map user input to command arguments in a way that ensures user input does not pass as-is into program execution.

# Create an Application that can prevent OS command injection for the following areas

- **DDoS Protection**—maintain uptime in all situations. Prevent any type of DDoS attack, of any size, from preventing access to your website and network infrastructure. [rate limiting](#)
- **CDN**—enhance website performance and reduce bandwidth costs with a CDN designed for developers. Cache static resources at the edge while accelerating APIs and dynamic websites.
- **Bot management**—analyzes your bot traffic to pinpoint anomalies, identifies bad bot behavior and validates it via challenge mechanisms that do not impact user traffic. [captcha](#)
- **API security**—protects APIs by ensuring only desired traffic can access your API endpoint, as well as detecting and blocking exploits of vulnerabilities. [api serialization](#)
- **Account takeover protection**—uses an intent-based detection process to identify and defends against attempts to take over users' accounts for malicious purposes. [2 FACTOR](#)
- **RASP**—keep your applications safe from within against known and zero-day attacks. Fast and accurate protection with no signature or learning mode.
- **Attack analytics**—mitigate and respond to real security threats efficiently and accurately with actionable intelligence across all your layers of defense.

# Path TraversAl Attack



# Directory Traversal Attacks

- Properly controlling access to web content is crucial for running a secure web server.
- Directory traversal or Path Traversal is an HTTP attack that allows attackers to access restricted directories and execute commands outside of the web server's root directory.
- Web servers provide two main levels of security mechanisms
  - Access Control Lists (ACLs)
  - Root directory

# Access Control Lists (ACLs)

- An Access Control List is used in the authorization process.
- It is a list which the web server's administrator uses to indicate which users or groups are able to access, modify or execute particular files on the server, as well as other access rights.

# Root directory

- The root directory is a specific directory on the server file system in which the users are confined. Users are not able to access anything above this root.
- For example: the default root directory of IIS on Windows is
  - C:\Inetpub\wwwroot
- and with this setup, a user does not have access to
  - C:\Windows but has access to C:\Inetpub\wwwroot\news
- and any other directories and files under the root directory (provided that the user is authenticated via the ACLs).
- The root directory prevents users from accessing any files on the server such as C:\WINDOWS\system32\win.ini on Windows platforms and the /etc/passwd file on Linux/UNIX platforms.
- This vulnerability can exist either in the web server software itself or in the web application code.
- In order to perform a directory traversal attack, all an attacker needs is a web browser and some knowledge on where to blindly find any default files and directories on the system.

## Example Scenario 1 : Reading arbitrary files via path traversal

- Imagine a shopping application that displays images of items for sale. This might load an image using the following HTML:

```

```

- The loadImage URL takes a filename parameter and returns the contents of the specified file.
- The image files are stored on disk in the location

```
/var/www/images/
```

To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file.

In other words, the application reads from the following file path:

```
/var/www/images/218.png
```

## Example Scenario 1 : Reading arbitrary files via path traversal ...

- This application implements no defenses against path traversal attacks. As a result, an attacker can request the following URL to retrieve the /etc/passwd file from the server's filesystem:

```
https://insecure-website.com/loadImage?filename=../../../../etc/passwd
```

- This causes the application to read from the following file path:

```
/var/www/images/../../../../etc/passwd
```

The sequence .. is valid within a file path, and means to step up one level in the directory structure. The three consecutive .. sequences step up from /var/www/images/ to the filesystem root, and so the file that is actually read is: /etc/passwd

On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server, but an attacker could retrieve other arbitrary files using the same technique.

On Windows, both .. and ..\ are valid directory traversal sequences. The following is an example of an equivalent attack against a Windows-based server:

```
https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini
```

# How to prevent a path traversal attack

- The most effective way to prevent path traversal vulnerabilities is to avoid passing user-supplied input to filesystem APIs altogether. Many application functions that do this can be rewritten to deliver the same behavior in a safer way.
- If you can't avoid passing user-supplied input to filesystem APIs, we recommend using two layers of defense to prevent attacks:
- Validate the user input before processing it. Ideally, compare the user input with a whitelist of permitted values. If that isn't possible, verify that the input contains only permitted content, such as alphanumeric characters only.
- After validating the supplied input, append the input to the base directory and use a platform filesystem API to canonicalize the path. Verify that the canonicalized path starts with the expected base directory.

# How to prevent a path traversal attack

- Below is an example of some simple Java code to validate the canonical path of a file based on user input:

```
File file = new File(BASE_DIRECTORY, userInput);
if (file.getCanonicalPath().startsWith(BASE_DIRECTORY)) {
    // process file
}
```



# Authentication Vulnerabilities

# Authentication vulnerabilities

- According to [Statista](#), more than 1,000 data breaches in 2020 exposed over 155 million records. These breaches resulted in an average cost of [\\$3.86 million](#).
- It's even more nerve-racking to know that over [82% of breaches were caused by authentication issues](#) — stolen or weak credentials.

# What Are Authentication Vulnerabilities?

- Authentication vulnerabilities are issues that affect authentication processes and make websites and applications susceptible to security attacks in which an attacker can masquerade as a legitimate user.
- Authentication is a vital part of any website or application since it is simply the process of recognizing user identities.

## How Do Authentication Vulnerabilities Emerge?

- There are several ways through which authentication vulnerabilities can arise, the most common of which are neglecting to address areas of risk, errors in code or logic, and poor user choices which can combine with the previous two.
- If an application has poor brute-force protection, attackers can take advantage of this to gain access to even well-protected accounts or cheap bulk access to poorly protected ones. Likewise, if there are logic or coding errors, malicious users may be able to bypass some or all of the authentication process.

# How Could Authentication-Based Vulnerabilities Impact You?



- Authentication vulnerabilities have serious repercussions — whether it's because of weak passwords or poor authentication design and implementation.
- Malicious users can use these vulnerabilities to get access into systems and user accounts to:
  - Steal sensitive information
  - Masquerade as a legitimate user
  - Gain control of the application
  - Destroy the system completely

# 11 Most Common Authentication Vulnerabilities

## 1. Flawed Brute-Force Protection

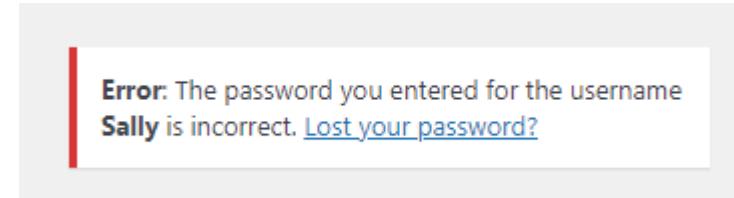
- A brute-force attack, such as a dictionary attack, is an attempt to gain illegal access to a system or user's account by entering large numbers of randomly generated or pregenerated combinations of usernames and passwords until they find one that works.
- If there's a flawed brute-force protection system such as a flaw in the authentication logic, firewall, or secure shell (SSH) protocol, attackers can hijack login credentials and processes, compromising the security of user credentials.

## 2. Weak Login Credentials

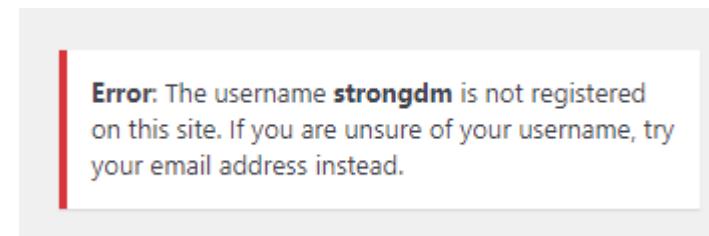
- When users register for an account on a site or application that uses password-based logins, they're prompted to create a username and password.
- However, if the password is predictable, this can lead to vulnerabilities in the authentication process. Predictable usernames can make it easier for attackers to target specific users.
- Rather than using a full brute-force attack, the attackers will look for accounts with easy-to-guess passwords, which are used far too often. . They'll try common credentials like "admin," "admin1," and "password1." With no restrictions on weak passwords, even sites protected against brute-force attacks can find themselves compromised.

# 3. Username Enumeration

- Username enumeration is not exactly an authentication vulnerability. But, it can make an attacker's life easier by lowering the cost for other attacks, such as brute-force attacks or weak credential checks.
- This process of username enumeration confirms whether or not a username is valid. For example:



- In this case, the username is correct but the password isn't.



- Here, the username is invalid.
- The problem with username enumeration is that attackers can tell what usernames are valid. Then, they can try to hack valid user accounts using brute-force techniques without wasting their time and money testing a multitude of invalid account names.

# 4. HTTP Basic Authentication

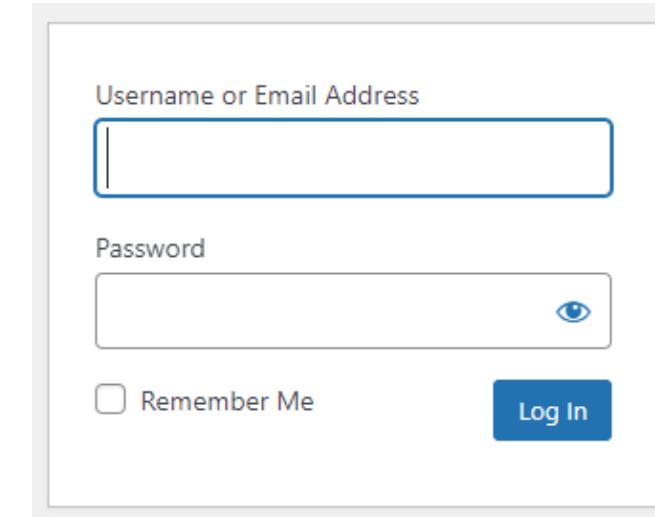
- This classic web authentication protocol is easy to implement, however, it is not without its risks.
- HTTP basic authentication is simple, sending a username and password with each request. However, if appropriate security protocols such as TLS session encryption are not used for all communication, the username and password information can be sent in the clear, making it easy for attackers to steal the credentials.
- Because the included credentials contain so little context, they can easily be misused in attacks such as cross-site request forgeries (CSRF). Also, because they are included with every single request, modern browsers normally cache this information indefinitely, with minimal ability to "log out" and prevent a local attacker from reusing the credential at some future point.

# 5. Poor Session Management

- A vulnerability in the management of session identifiers leads to hijacking of valid authenticated sessions. This is one of the common web vulnerabilities to bypass passwords.
- There are several session mismanagement vulnerabilities such as no session timeouts, exposure of session IDs in URLs, session cookies without the Http-Only flag set, and poor session invalidation.
- If attackers can seize control of an existing session, they easily get into a system by assuming the identity of an already-authenticated user, bypassing the authentication process entirely.

# 6. Staying Logged In

- A "Remember me" or "Keep me logged in" checkbox beneath a login form makes it super easy to stay logged in after closing a session. It generates a cookie that lets you skip the process of logging in.
- However, this can lead to a cookie-based authentication vulnerability if an attacker can predict a cookie or deduce its generation pattern.
- They can use malicious techniques like brute-force attacks to predict cookies, and cross-site scripting (XSS) to hack user accounts by allowing a malicious server to make use of a legitimate cookie.
- If a cookie is poorly designed or protected, attackers may be able to obtain passwords or other sensitive (and legally protected) data such as user addresses or account information from a stored cookie.



The image shows a standard web-based login interface. At the top, there is a light gray header bar. Below it, the main content area has a white background. It contains two input fields: one for 'Username or Email Address' and one for 'Password'. The 'Password' field includes a small blue eye icon to its right, which likely serves as a toggle for password visibility. Below these fields is a 'Remember Me' checkbox followed by a 'Log In' button. The entire form is enclosed in a thin gray border.

# 7. SQL Injection

- SQL injection is an attack vector that uses malicious SQL code input in an unexpected way to manipulate and access a database.
- SQL injections can enable attacks on authentication mechanisms by stealing relevant data (such as poorly protected password hashes) from an unprotected database. They can also bypass authentication mechanisms if the injected SQL code is executed by an internal (and already authorized) tool that failed to sufficiently validate external input.

# 8. Unsecure Password Change and Recovery

- Sometimes, users forget or just want to change their passwords and click the "Forgot password" or "Lost your password" links.
- user-login-example-with-password-reset
- The password reset process poses an authentication vulnerability if an application uses a weak password recovery mechanism such as easy security questions, no CAPTCHAs, or password reset e-mails with overly long or no timeouts.
- If the password recovery functionality is flawed, attackers can potentially use brute-force techniques or access to other compromised accounts to gain access to user accounts and credentials that are well-protected under normal circumstances.

# 9. Flawed Two-Factor Authentication

- While two-factor authentication (2FA) is effective for secure authentication, it can cause critical security issues if not well-implemented.
- Attackers can figure out the four- and six-digit 2FA verification codes through SIM swap attacks if they are sent through SMS. Some two-factor authentication is also not truly two-factor; if a user is attempting to access sensitive information on a stolen phone using cached credentials, a "second factor" that sends a message to that same phone adds no additional security.
- Two-factor authentication vulnerabilities can also occur if there's no brute-force protection to lockout an account after a specific number of attempted logins.

# 10. Vulnerable Authentication Logic

- Logic vulnerabilities are common in software applications. This occurs as a result of poor coding or design that affects authentication and authorization access, and application functionality.
- Flawed application logic can be due to the abuse of functionality, weak security measures, or a skipped step in the verification procedure.
- For example, an application can prompt a user to answer a security question the logic deems as something "only the correct person would know." But questions like the user's birthday or mother's maiden name are often easy to discover. This vulnerability makes it easy for cyber attackers to bypass authentication and gain illegal access to such accounts.

# 11. Human Negligence

- According to Shred-it 2020 report, up to 31% of C-suite executives reported employee negligence to be the second major cause of their data breaches.
- Human error can result in serious authentication vulnerabilities that are far easier to take advantage of than brute-force attacks, SQL injections, and authentication bypasses. This negligence includes actions such as:
  - Leaving a computer on and unlocked in a public place
  - Losing devices to theft
  - Leaking sensitive information to strangers
  - Writing bad code

# How to Prevent Authentication Vulnerabilities?

- Here are eight best practices to prevent authentication-based vulnerabilities and keep critical information safe.
- 1. Implement a reliable brute-force protection system:** Brute-force attacks can be prevented by enforcing account lockouts, rate limiting, IP-based monitoring, application firewalls, and CAPTCHAs.
  - 2. Enforce a secure password policy:** Do this by creating a password checker that tells users how strong their passwords are in real-time. You can also implement passwordless authentication using standards like FIDO2 to mitigate the risk and stress of managing passwords.
  - 3. Apply HTTP strict transport security (HSTS):** This forces web sessions to use TLS encryption, preventing sensitive information from being accessed in transit.
  - 4. Consider disabling username enumeration:** By generating the same error for a login failure whether the username was valid or invalid, you force an attacker to brute-force not just the set of possible passwords, but also the set of likely usernames, rather than sticking to the ones they know are valid.

# How to Prevent Authentication Vulnerabilities?

**5. Modify cookie headers:** Modifying cookie headers protects them against malicious attacks. Using the HttpOnly and SameSite tags when setting cookie headers prevents them from XSS and CSRF attacks, respectively.

**6. Scrutinize your coding on verifications:** This is important for detecting any vulnerabilities in your code.

Overall, periodically audit your code to discover logic flaws and authentication bypass and strengthen your security posture.

**7. Use parameterized statements:** You can prevent SQL Injection attacks through input validation and parameterized queries. They are safer to avoid directly putting user-provided input directly into SQL statements.

**8. Implement proper multi-factor authentication:** Using multi-factor authentication is more secure than password-based mechanisms. However, you need solid code and secured generation of verification codes to effectively implement this form of authentication.

# SERVER - SIDE REQUEST FORGERY (SSRF)



<https://www.invicti.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/>

<https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>

# Server-Side Request Forgery (SSRF)

- A Server-Side Request Forgery (SSRF) attack involves an attacker abusing server functionality to access or modify resources.
- The attacker targets an application that supports data imports from URLs or allows them to read data from URLs.
- URLs can be manipulated, either by replacing them with new ones or by tampering with URL path traversal.
- Typically, attackers supply a URL (or modify an existing one) and the code running on the server reads or submits data to it. Attackers can leverage URLs to gain access to internal data and services that were not meant to be exposed – including HTTP-enabled databases and server configuration data.
- Once an attacker has tampered with the request, the server receives it and attempts to read data to the altered URL. Even for services that aren't exposed directly on the public internet, attackers can select a target URL, which enables them to read the data.

# Some common risks of SSRF : Data Exposure

- One of the most prevalent examples of an SSRF attack is gaining access to Amazon EC2 instance credentials. If an IAM role is attributed to an EC2 instance, the provisional credentials may be accessed by completing a request to:

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/{role-name}
```

- The level of damage that can be caused by the attacker depends on the level of access given to the IAM role. The higher the privileges of the role, the bigger the breach.
- For application servers, this tends to indicate that, at the least, a cybercriminal will be capable of retrieving customer information. If excessive permissions are granted to the IAM role, attackers may be able to execute code remotely on EC2 instances within the target's AWS account.

# Some common risks of SSRF : Reconnaissance

- A common security practice used to minimize the attack surface from external networks is to limit the use of public-facing servers.
- The remaining servers are reserved for internal communication. SSRF allows attackers to carry out scans and collect information about internal networks.
- Once an attacker has gained access to the server, they can use this information to compromise other servers within the network.

# Some common risks of SSRF : Port Scans or Cross Site Port Attack (XSPA)



- SSRF attacks don't always return data to the attacker. Response times or other metadata, however, can allow an attacker to determine if a request was successful or not. If a port and a host can be pinpointed, the attacker could port scan the application server's network by leveraging this metadata in a Cross-Site Port Attack (XSPA).
- The timeout for a network connection generally remains unchanged, irrespective of the host or port. An attacker could thus attempt a request they know will fail, and use this as a baseline for future response times. Requests that are successful will tend to be markedly shorter than this baseline, and occasionally longer if the connection established is not secured by one of the parties.
- By working this way, attackers can fingerprint the services being carried out on the network, which allows them to initiate protocol smuggling attacks.

# Some common risks of SSRF : :Denial of Service (DoS)

- The volume of requests received by the internal servers is typically lower than the traffic to public-facing servers. Thus, they are configured to incur lower bandwidth.
- Cybercriminals may utilize SSRF to flood the internal servers with large amounts of traffic to take up their bandwidth, which results in an internal DoS attack.
- Aside from these prevalent attacks, cybercriminals may employ SSRF to carry out malicious or unauthorized actions or to embed malware. The greater knowledge organizations have about these risks, the more alarming it becomes.

# Remote Code Execution (RCE)

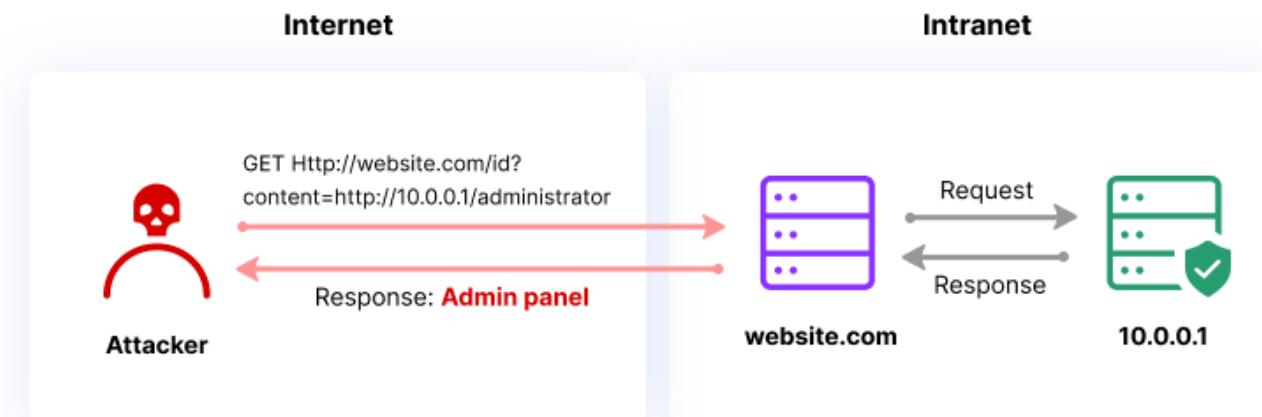
- Certain modern services are intended to be entirely interfaced via HTTP queries. Unlimited control of the URL may therefore allow the cybercriminal to exploit certain services, which could result in anything—even remote code execution on the core server (a well-known example is Redis).

# Types Of SSRF Attacks

- SSRF attacks commonly exploit a trust relationship,
  - within the server itself (known as a **server SSRF attack**) or
  - between the server and other back-end systems (known as a **back-end SSRF attack**).

# Server SSRF Attacks

- In a server SSRF attack, attackers exploit a process in which a browser or other client system directly accesses a URL on the server.
- The attacker will replace the original URL with another, typically using the IP 127.0.0.1 or the hostname “localhost”, which point to the local file system on the server.
- Under this hostname the attacker finds a file path that leads to sensitive data.



# How Server SSRF works

- For example, on a weather website, the web application queries its server for current weather forecasts to display. It can do this using a REST API, passing a URL with an API request from the user's browser to the server. The request might look like this:

```
POST /meteorology/forecasts HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 113
weatherApi=http://data.weatherapp.com:8080/meteorology/forecasts/check%3FcurrentTime%3D6%26cityId%3D1
```

The attacker can change this to the following:

```
weatherApi=http://localhost/admin
```

- This will cause the server to display the contents of the /admin folder to the attacker. Because the request is executed within the server's file system, it bypasses ordinary access controls and exposes the information even though the attacker is unauthorized.

# Back-End SSRF attacks

- Another variant on SSRF is when a server has a trusted relationship with a back-end component.
- If, when the server connects to that component, it has full access rights, an attacker can forge a request and gain access to sensitive data, or perform unauthorized operations.
- Back-end components often have weak security because they are considered to be protected inside the network perimeter.
- Continuing the previous example, the attacker could replace the API call with:

```
weatherApi=http://192.168.12.5/admin
```

- If the server connects to a back-end component on IP address 192.168.12.5 and is allowed to access the /admin directory on that component's file system, the attacker can similarly gain access and view the content of the directory.

# Mitigating Server-Side Request Forgery

- **Whitelists and DNS Resolution**
- A solid approach to avoiding SSRF is to whitelist the IP addresses or DNS names that your application requires access to. Only if a whitelist approach is not suitable should you use a [blacklist](#). It is essential that you authorize user input effectively. For instance, do not permit requests to private IP addresses (which are non-routable).
- **Response Handling**
- To stop response information from reaching the cybercriminal, you should make sure that the response that is received conforms to what is anticipated. On no account should the raw response body received from the request initiated by the server be transferred to the client.

# Mitigating Server-Side Request Forgery.....

- **Disable Unused URL Schemas**
- If your application is solely reliant on HTTPS or HTTP to initiate requests, permit only these URL schemas. By disabling unused URL schemas, you deny attackers the ability to utilize the application to carry out requests via potentially harmful schemas, including dict://, file:///, and gopher://.
- **Authentication on Internal Services**
- Services like Redis, MongoDB, Elasticsearch, and Memcached do not demand verification by default. A cybercriminal may gain access to certain services without verification, using SSRF vulnerabilities. Thus, to enforce web application security, it is desirable to allow verification whenever you can, including for local network services.