Chapter 3

Web Application Technologies:

This chapter simply explains how web applications work by introducing the basic technologies behind them, like how browsers and servers communicate, how websites are built, and how data is formatted. Knowing these basics makes it much easier to understand and find security weaknesses in web applications, especially for beginners.

The HTTP Protocol:

What is HTTP?

HTTP is a set of rules that lets your web browser and a website talk to each other. When you type a website address or click a link, your browser uses HTTP to ask the website for a page, image, or video. The website then sends that information back, and your browser shows it to you. In simple terms, HTTP is how information moves back and forth on the web.

Can this be hacked?

Yes, because the data is sent as plain text, not encrypted.

How to protect?

Use HTTPS.

HTTP Requests:

An HTTP request is a structured message that tells the server what you want, where it is, how you're asking, and extra details about the browser and session.

An HTTP request is a text message made of a request line, headers, a blank line, and sometimes a body. The request line shows the method (like GET), the URL being requested, and the HTTP version. The headers give extra details such as the browser type, the website name, where the request came from, and any cookies. Together, these parts tell the server what resource is needed and how to handle the request.

HTTP Response:

An HTTP response is the message a server sends back after receiving a request. It starts with a line showing the HTTP version, a status code (like 200, meaning success), and a short message describing the result. After that come headers, which give extra details such as the server type, cookies to store, caching rules, content type, and content length. After a blank line, the response usually includes a message body, which contains the actual content, such as an HTML web page.

What are HTTP methods?

HTTP mainly uses two methods: GET and POST. GET is used to *get information*, and it sends data in the website address, which can be seen, saved, and logged, so it should not be used for sensitive information. POST is used to *perform actions* and sends data inside the request, making it more private and safer for things like forms or changes. Browsers also protect users by warning them before repeating a POST action, which helps prevent doing the same action twice.

Basic set of operations that can be used to interact with server:

- GET: retrieve a resource
- HEAD: No message body (response headers only)
- OPTIONS: what operations are available
- TRACE: loopback test (get same data)
- DELETE: remove a resource
- PUT: replace a resource
- POST: interact with resource (mostly add)
- PATCH: change part of a resource

URLS:

A URL is the address used to find and access a resource on the web. It usually includes the protocol (like HTTP or HTTPS), the website name, an optional port, the path to the resource, and sometimes parameters after a question mark. URLs can be written in a full (absolute) form with the complete address or in a shorter (relative) form that works within the same website. In simple terms, a URL is where the browser goes to get what you asked for.

What are HTTP headers?

These are metadata (key-value sent along with request and response).

An HTTP header is extra information sent along with an HTTP request or response to help the browser and website understand each other better. It doesn't contain the main content like a web page, but gives instructions and details, such as what kind of data is being sent, which browser is being used, whether the user is logged in, or how long the data can be saved. In simple words, HTTP headers are like notes attached to a package, telling the receiver how to handle what's inside.

Uses of HTTP headers:

1. Caching, authentication, manage state.

Types of headers:

1. Request headers: from client
2. Response headers: from server
3. Representation headers: encoding / compression
4. Payload headers: data

Cookies:

Cookies are small pieces of data that a website stores in a user's web browser to remember information about the user and their interaction with the website. They are automatically sent back to the website by the browser with each request, helping the website keep track of things like login sessions, user preferences, and activity.

How cookies work?

When you visit a website, the server sends a cookie to your browser. Your browser stores this cookie and then automatically sends it back to the same server every time you make another request to that website. This happens without you or the website needing to do anything extra.

Benefits of cookies:

- User authentication – Keep users logged in without asking for credentials on every page
- Personalization – Remember user preferences like language, theme, or settings
- Session management – Maintain user sessions across multiple web pages
- Improved user experience – Save shopping carts, form data, and recent activity
- Analytics and performance – Help websites understand user behavior and improve services
- Faster access – Reduce repeated data entry and server processing
- Security support – Help detect suspicious activity when used with proper settings

In short, cookies make websites more functional, user-friendly, and efficient.

For example:

- The server sends a cookie like this:

  Set-Cookie: tracking=ABC123

- Your browser saves it.

- On later requests, your browser sends it back like this:

  Cookie: tracking=ABC123

Cookies are usually written as name = value pairs. A website can send multiple cookies, and your browser will send them all back together in one request.

Cookie Options:

When a server creates a cookie, it can add extra rules that control how the cookie works:

- expires – Sets how long the cookie lasts.
  - If set, the cookie is saved on your device and used even after you close the browser.
  - If not set, the cookie is deleted when you close the browser.
- domain – Tells the browser which website(s) can use the cookie.
- path – Limits the cookie to specific pages or paths on the website.
- secure – The cookie is only sent over HTTPS (encrypted) connections.
- HttpOnly – Prevents JavaScript from accessing the cookie, which helps protect against attacks.

Types of cookies:

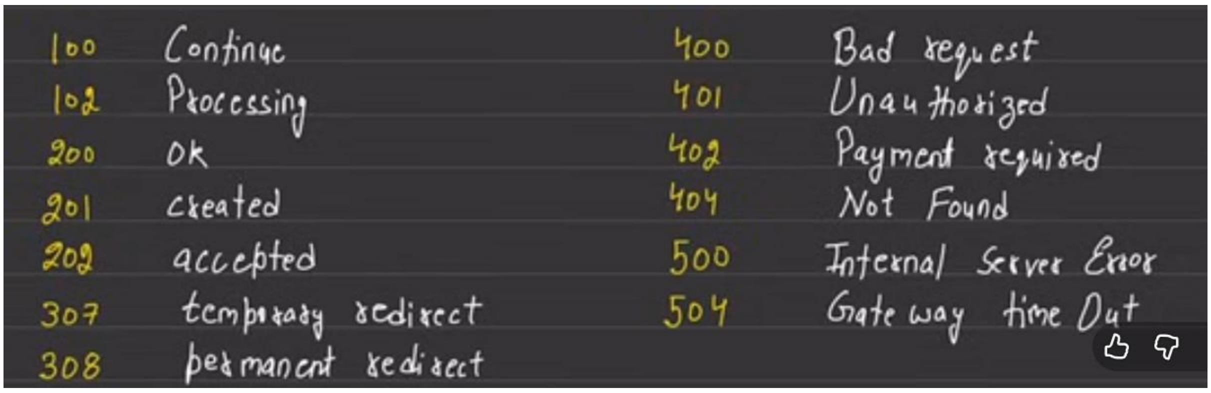
1. **Session cookies**
   a. Temporary cookies
   b. Deleted when the browser is closed
   c. Used to manage user sessions (e.g., login status)
2. **Persistent cookies**
   a. Stored on the device for a set time

   b. Remember preferences and settings across visits
3. **First-party cookies**
   a. Created by the website you are visiting
   b. Used for site functionality and personalization
4. **Third-party cookies**
   a. Created by other domains (e.g., advertisers)
   b. Commonly used for tracking and ads
5. **Secure cookies**
   a. Sent only over HTTPS connections
   b. Help protect sensitive data
6. **HttpOnly cookies**
   a. Not accessible via JavaScript
   b. Reduce risk of attacks like XSS
7. **SameSite cookies**
   a. Control when cookies are sent with cross-site requests
   b. Help prevent CSRF attacks

These types help websites balance functionality, personalization, and security.

<u>HTTP status code:</u>

- 1xx: Informational
- 2xx: Success
- 3xx: Redirection
- 4xx: Client error
- 5xx: Server error



100	Continue	400	Bad request
102	Processing	401	Unauthorized
200	OK	402	Payment required
201	created	404	Not Found
202	accepted	500	Internal Server Error
307	temporary redirect	504	Gate way time Out
308	permanent redirect		

<u>HTTPS:</u>

HTTPS (HyperText Transfer Protocol Secure) is a secure version of HTTP that is used to transfer data between a web browser and a website. **HTTP sends data over the internet using normal TCP**, which is not encrypted. This means that if an attacker is on the same network, they can see or steal the data being sent.

HTTPS is the secure version of HTTP. **It works the same way as HTTP, but the data is sent through a secure, encrypted connection using SSL (now called TLS).**

What is a Proxy?

A proxy is an intermediary server that sits between a client (like your browser) and a destination server (like a website).

Instead of connecting directly to the website:

- Your request goes to the proxy
- The proxy forwards it to the website
- The website's response comes back through the proxy to you

HTTP Proxies:

HTTP proxies are servers that act as a middleman between a web browser and web servers, handling HTTP and HTTPS traffic. They receive requests from the browser, forward them to the target website, and return the website's responses back to the browser. HTTP proxies can also inspect, modify, filter, cache, or block web traffic. In simple terms, an HTTP proxy controls and manages web communication between a user and websites.

HTTPS Proxies:

HTTPS proxies are proxy servers that handle secure (HTTPS) web traffic between a browser and a website.

How HTTPS proxies work:

- Your browser connects to the proxy
- The proxy creates a secure tunnel to the destination website
- Encrypted data passes through the proxy without being read

This is usually done using the CONNECT method, which allows the proxy to act as a tunnel rather than decrypting the data.

HTTP Authentication:

HTTP Authentication is a built-in way for websites to verify a user's identity using features provided directly by the HTTP protocol.

There are different authentication methods:

1. Basic Authentication

- Sends the username and password with every request
- The credentials are only Base64-encoded, not encrypted
- Easy to use but not secure unless combined with HTTPS

2. NTLM Authentication

- Uses a challenge-response process
- Based on Windows NTLM security
- Commonly used in corporate or intranet environments

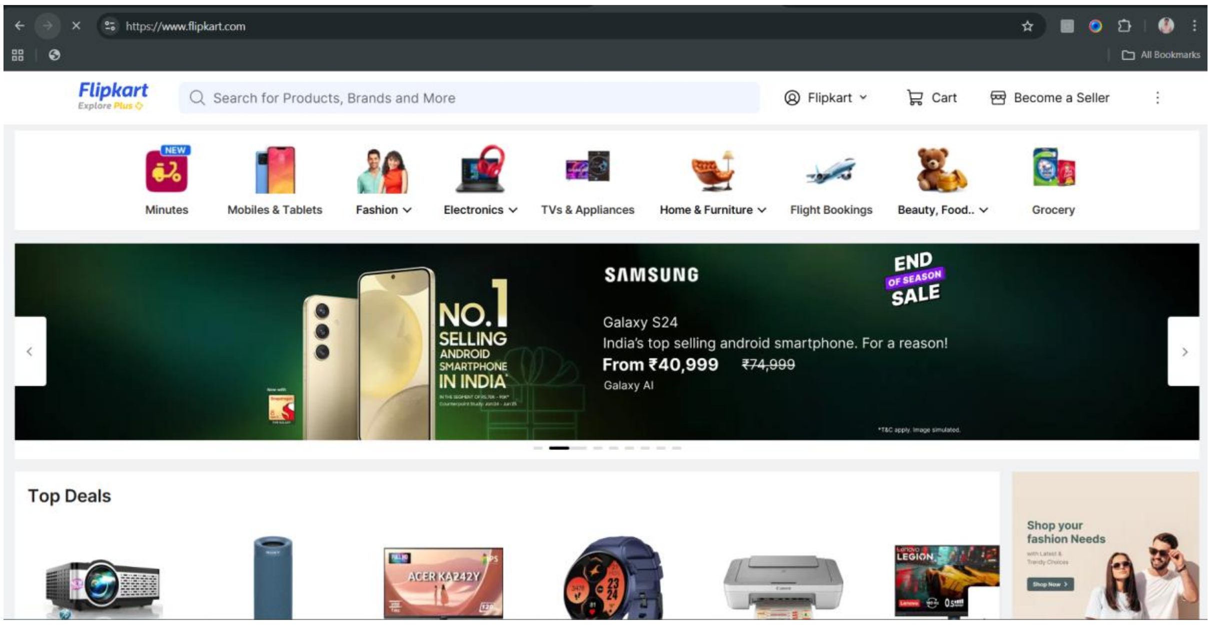
3. Digest Authentication

- Also uses a challenge-response method

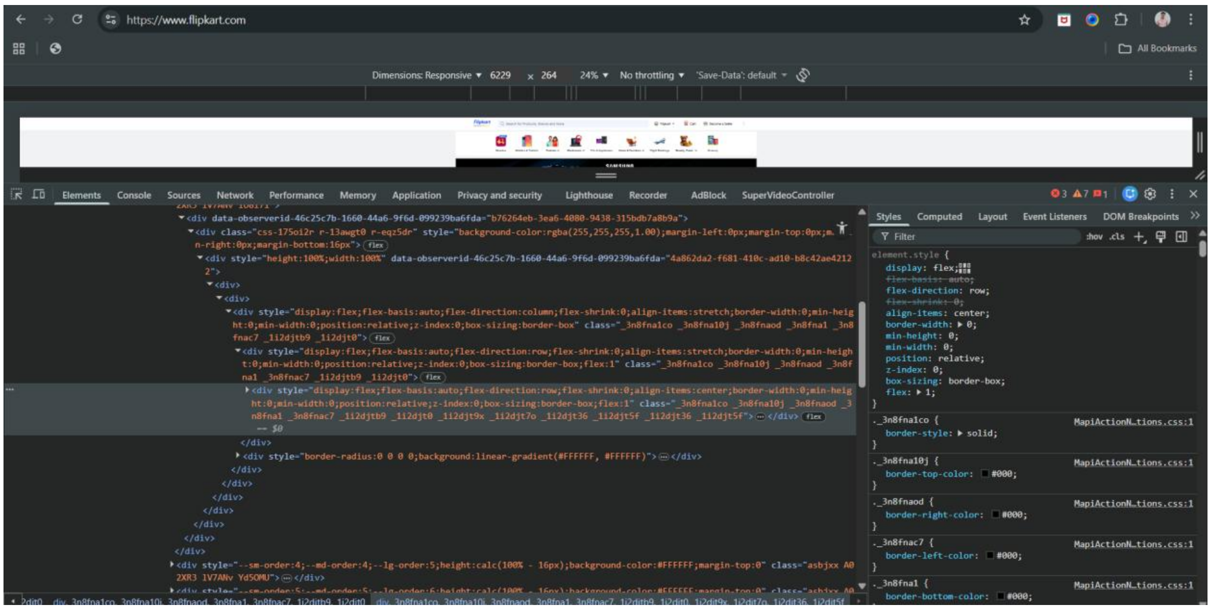
- Sends a hashed (MD5) version of the password instead of plain text
- More secure than Basic authentication

<u>How can I see cookies in my browser?</u>

Step1: Open the website



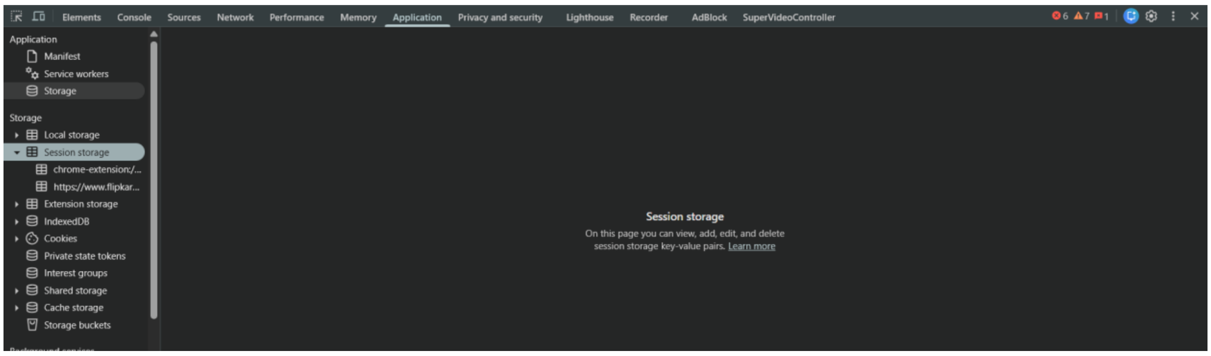
Step2: Right click and click on the “inspect”



```
▼<div data-observerid-46c25c7b-1660-44a6-9f6d-099239ba6fda="b76264eb-3ea6-4080-9438-315bdb7a8b9a">
 ▼<div class="css-175oi2r r-13awgt0 r-eqz5dr" style="background-color:rgba(255,255,255,1.00);margin-left:0px;margin-top:0px;m…
  n-right:0px;margin-bottom:16px"> flex
  ▼<div style="height:100%;width:100%" data-observerid-46c25c7b-1660-44a6-9f6d-099239ba6fda="4a862da2-f681-410c-ad10-b8c42ae4212
   2">
   ▼<div>
    ▼<div>
     ▼<div style="display:flex;flex-basis:auto;flex-direction:column;flex-shrink:0;align-items:stretch;border-width:0;min-heig
      ht:0;min-width:0;position:relative;z-index:0;box-sizing:border-box" class="_3n8fna1co _3n8fna10j _3n8fnaod _3n8fna1 _3n8
      fnac7 _1i2djtb9 _1i2djt0"> flex
      ▼<div style="display:flex;flex-basis:auto;flex-direction:row;flex-shrink:0;align-items:stretch;border-width:0;min-heigh
       t:0;min-width:0;position:relative;z-index:0;box-sizing:border-box;flex:1" class="_3n8fna1co _3n8fna10j _3n8fnaod _3n8f
       na1 _3n8fnac7 _1i2djtb9 _1i2djt0"> flex
   ...  ▶<div style="display:flex;flex-basis:auto;flex-direction:row;flex-shrink:0;align-items:center;border-width:0;min-heig
        ht:0;min-width:0;position:relative;z-index:0;box-sizing:border-box;flex:1" class="_3n8fna1co _3n8fna10j _3n8fnaod _3
        n8fna1 _3n8fnac7 _1i2djtb9 _1i2djt0 _1i2djt9x _1i2djt7o _1i2djt36 _1i2djt5f _1i2djt36 _1i2djt5f"> ⋯ </div> flex
         == $0
       </div>
      ▶<div style="border-radius:0 0 0 0;background:linear-gradient(#FFFFFF, #FFFFFF)"> ⋯ </div>
     </div>
    </div>
   </div>
  </div>
 </div>
</div>
</div>
▶<div style="--sm-order:4;--md-order:4;--lg-order:5;height:calc(100% - 16px);background-color:#FFFFFF;margin-top:0" class="asbjxx A0
 2XR3 1V7ANv Yd5OMU"> ⋯ </div>
```

```
element.style {
  display: flex;
  ~~flex-basis: auto;~~
  flex-direction: row;
  ~~flex-shrink: 0;~~
  align-items: center;
  border-width: ▶ 0;
  min-height: 0;
  min-width: 0;
  position: relative;
  z-index: 0;
  box-sizing: border-box;
  flex: ▶ 1;
}
._3n8fna1co {                          MapiActionN…tions.css:1
  border-style: ▶ solid;
}
._3n8fna10j {                          MapiActionN…tions.css:1
  border-top-color: ■ #000;
}
._3n8fnaod {                           MapiActionN…tions.css:1
  border-right-color: ■ #000;
}
._3n8fnac7 {                           MapiActionN…tions.css:1
  border-left-color: ■ #000;
}
._3n8fna1 {                            MapiActionN…tions.css:1
  border-bottom-color: ■ #000;
```

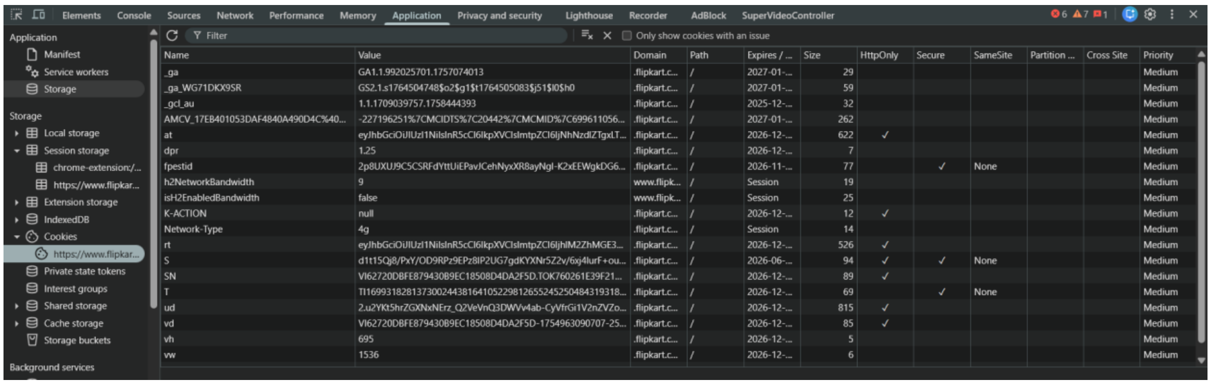
Step3: Go to the application tab



Session storage

On this page you can view, add, edit, and delete session storage key-value pairs. Learn more

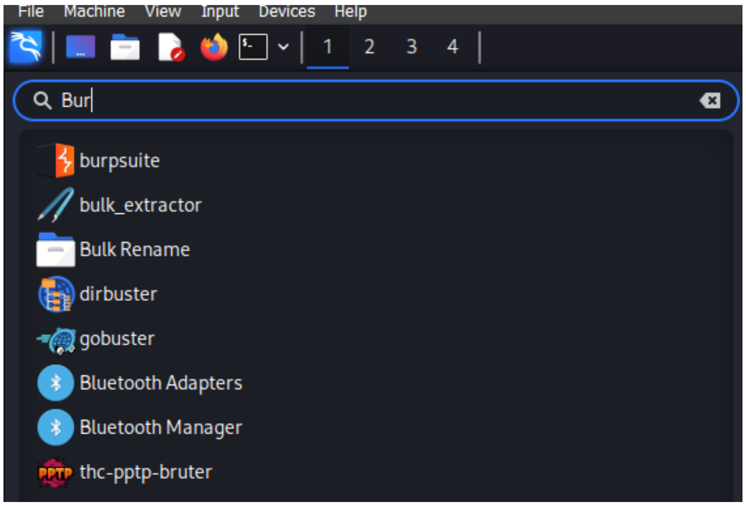
Step4: click on the left tab



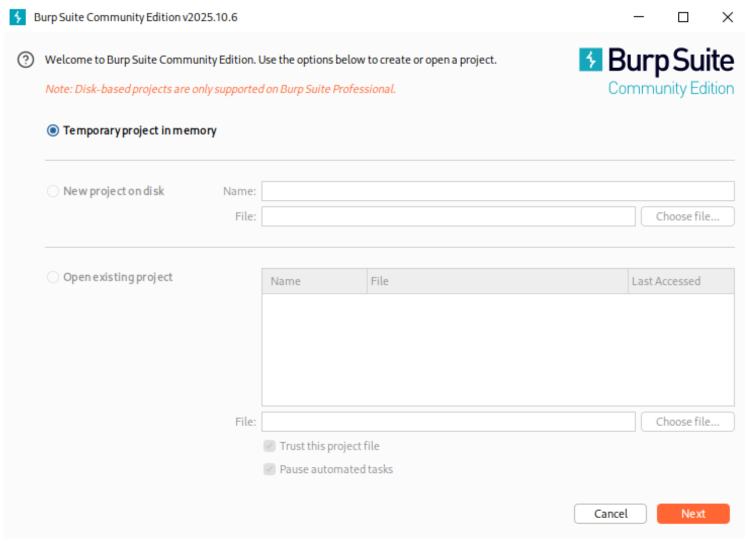
Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite	Partition ...	Cross Site	Priority
_ga	GA1.1.992025701.1757074013	.flipkart.c...	/	2027-01-...	29						Medium
_ga_WG71DKX9SR	GS2.1.s1764504748$o2$g1$t1764505083$j51$l0$h0	.flipkart.c...	/	2027-01-...	59						Medium
_gcl_au	1.1.1709039757.1758444393	.flipkart.c...	/	2025-12-...	32						Medium
AMCV_17EB401053DAF4840A490D4C%40...	-227196251%7CMCIDTS%7C20442%7CMCMID%7C699611056...	.flipkart.c...	/	2027-01-...	262						Medium
at	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjNhNzdlZTgxLT...	.flipkart.c...	/	2026-12-...	622	✓					Medium
dpr	1.25	.flipkart.c...	/	2026-12-...	7						Medium
fpestid	2p8UXUJ9C5CSRFdYttUiEPavJCehNyxXR8ayNgl-K2xEEWgkDG6...	.flipkart.c...	/	2026-11-...	77		✓	None			Medium
h2NetworkBandwidth	9	www.flipk...	/	Session	19						Medium
isH2EnabledBandwidth	false	www.flipk...	/	Session	25						Medium
K-ACTION	null	.flipkart.c...	/	2026-12-...	12	✓					Medium
Network-Type	4g	.flipkart.c...	/	Session	14						Medium
rt	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjhlM2ZhMGE3...	.flipkart.c...	/	2026-12-...	526	✓					Medium
S	d1t15Qj8/PxY/OD9RPz9EPz8IP2UG7gdKYXNr5Z2v/6xj4lurF+ou...	.flipkart.c...	/	2026-06-...	94	✓	✓	None			Medium
SN	VI62720DBFE879430B9EC18508D4DA2F5D.TOK760261E39F21...	.flipkart.c...	/	2026-12-...	89	✓					Medium
T	TI169931828137300244381641052298126552452504843193318...	.flipkart.c...	/	2026-12-...	69		✓	None			Medium
ud	2.u2YKt5hrZGXNxNErz_Q2VeVnQ3DWVv4ab-CyVfrGi1V2nZVZo...	.flipkart.c...	/	2026-12-...	815	✓					Medium
vd	VI62720DBFE879430B9EC18508D4DA2F5D-1754963090707-25...	.flipkart.c...	/	2026-12-...	85	✓					Medium
vh	695	.flipkart.c...	/	2026-12-...	5						Medium
vw	1536	.flipkart.c...	/	2026-12-...	6						Medium

How to intercept HTTP traffic?

Step1: Open the Burp Suite.



Step2: Following screen will appear.



Welcome to Burp Suite Community Edition. Use the options below to create or open a project.

Note: Disk-based projects are only supported on Burp Suite Professional.

[✓ Temporary project in memory]

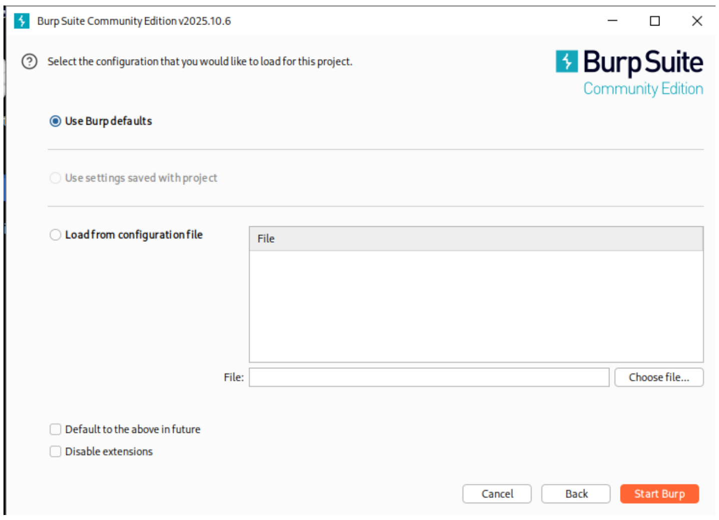
New project on disk — Name: File:

Open existing project — File:

Trust this project file

Pause automated tasks

Step3: Following screen will appear if you click on "next".



Select the configuration that you would like to load for this project.

[✓ Use Burp defaults]

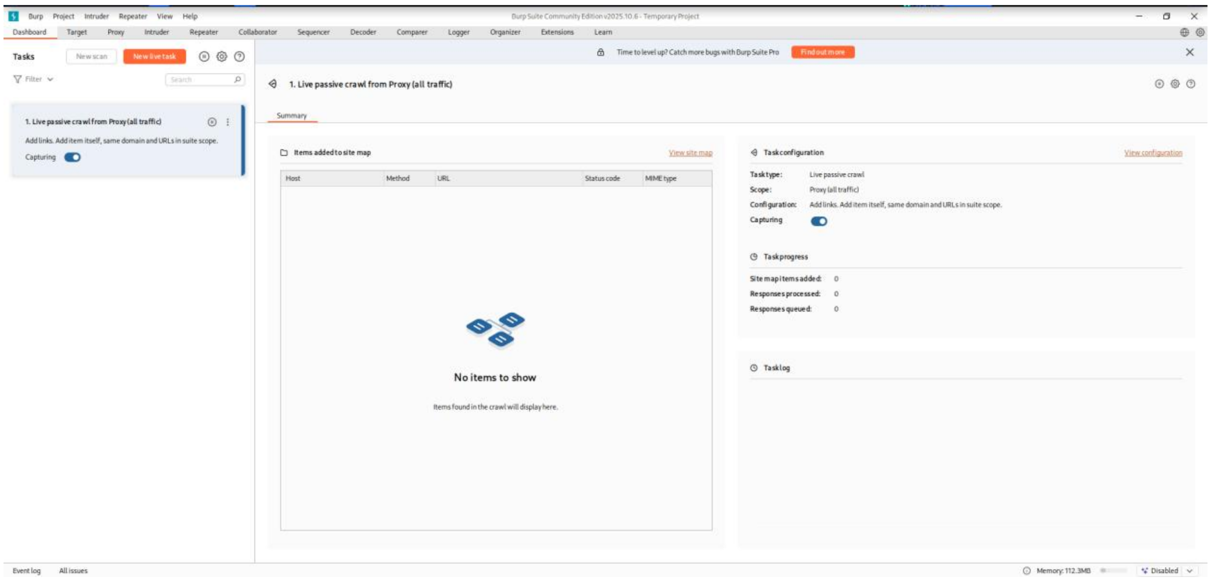
Use settings saved with project

Load from configuration file — File:

Default to the above in future

Disable extensions

Step4: Click on Start Burp, following screen will appear.



Time to level up? Catch more bugs with Burp Suite Pro

1. Live passive crawl from Proxy (all traffic)

Add links. Add item itself, same domain and URLs in suite scope.

Capturing

Items added to site map

No items to show

Items found in the crawl will display here.

Task configuration

Task type: Live passive crawl

Scope: Proxy (all traffic)

Configuration: Add links. Add item itself, same domain and URLs in suite scope.

Capturing

Task progress

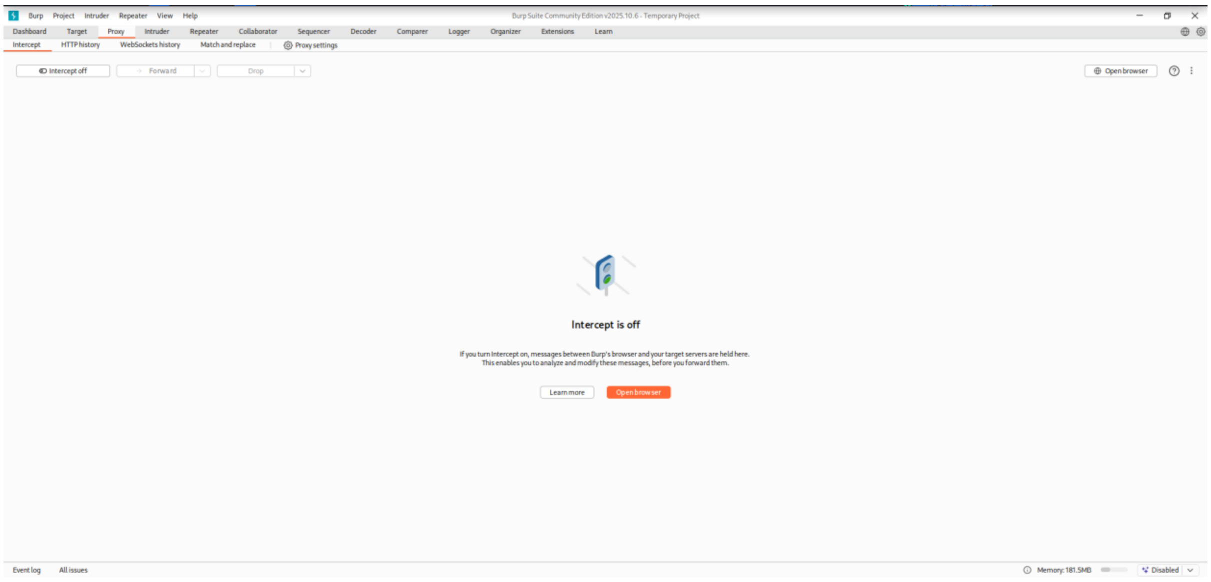
Site map items added: 0

Responses processed: 0

Responses queued: 0

Task log

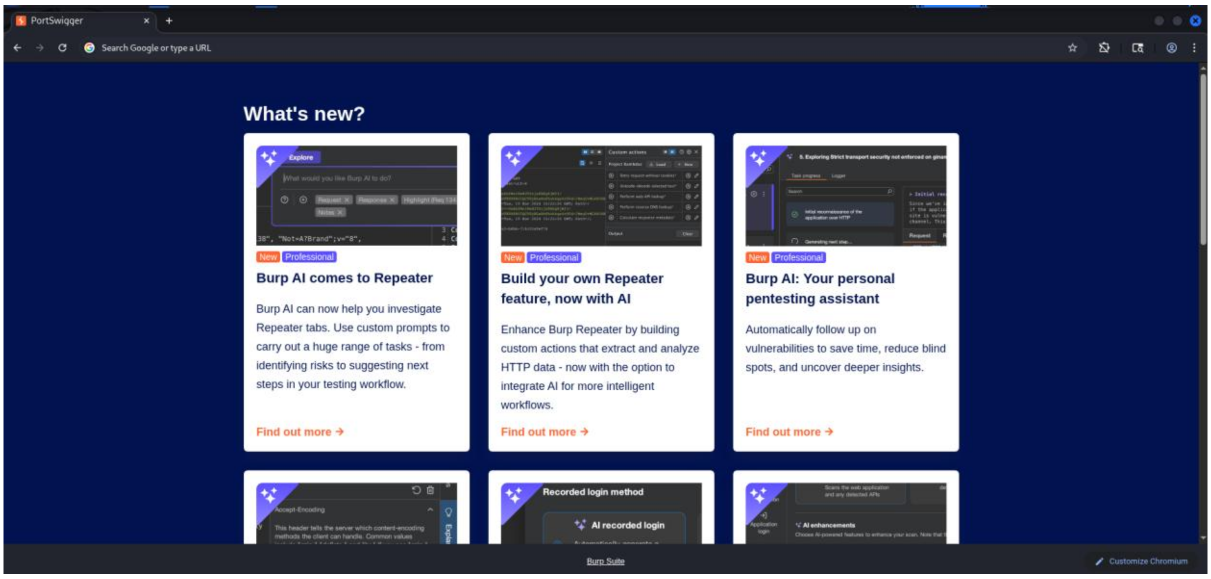
Step5: Go to the “proxy” tab.



Intercept is off

If you turn Intercept on, messages between Burp's browser and your target servers are held here.
This enables you to analyze and modify these messages, before you forward them.

Step6: Click on “Open Browser”, following screen will appear.



What's new?

Burp AI comes to Repeater

Burp AI can now help you investigate Repeater tabs. Use custom prompts to carry out a huge range of tasks - from identifying risks to suggesting next steps in your testing workflow.

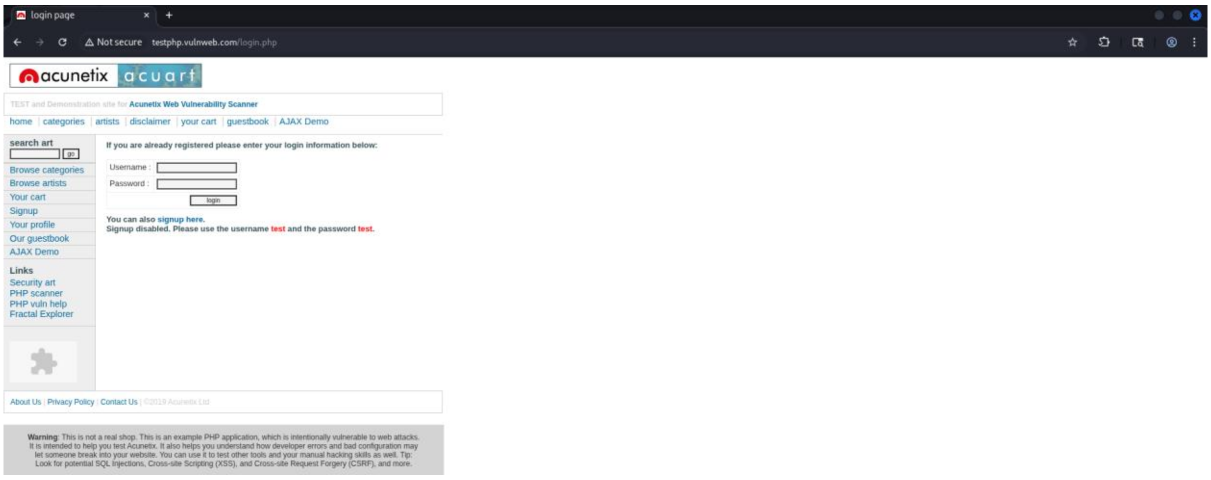
Build your own Repeater feature, now with AI

Enhance Burp Repeater by building custom actions that extract and analyze HTTP data - now with the option to integrate AI for more intelligent workflows.

Burp AI: Your personal pentesting assistant

Automatically follow up on vulnerabilities to save time, reduce blind spots, and uncover deeper insights.

Step7: In the search bar go to the following website.



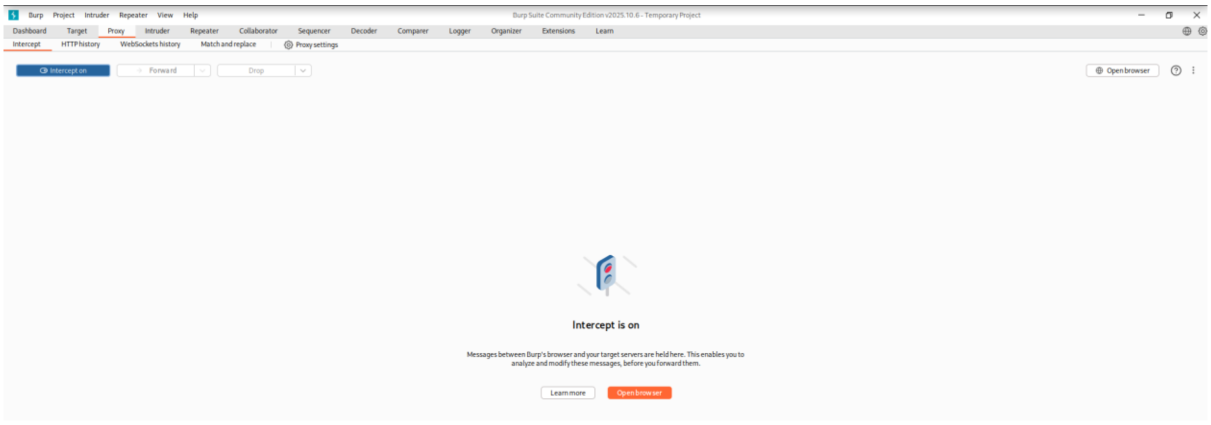
TEST and Demonstration site for Acunetix Web Vulnerability Scanner

If you are already registered please enter your login information below:

You can also signup here.
Signup disabled. Please use the username test and the password test.

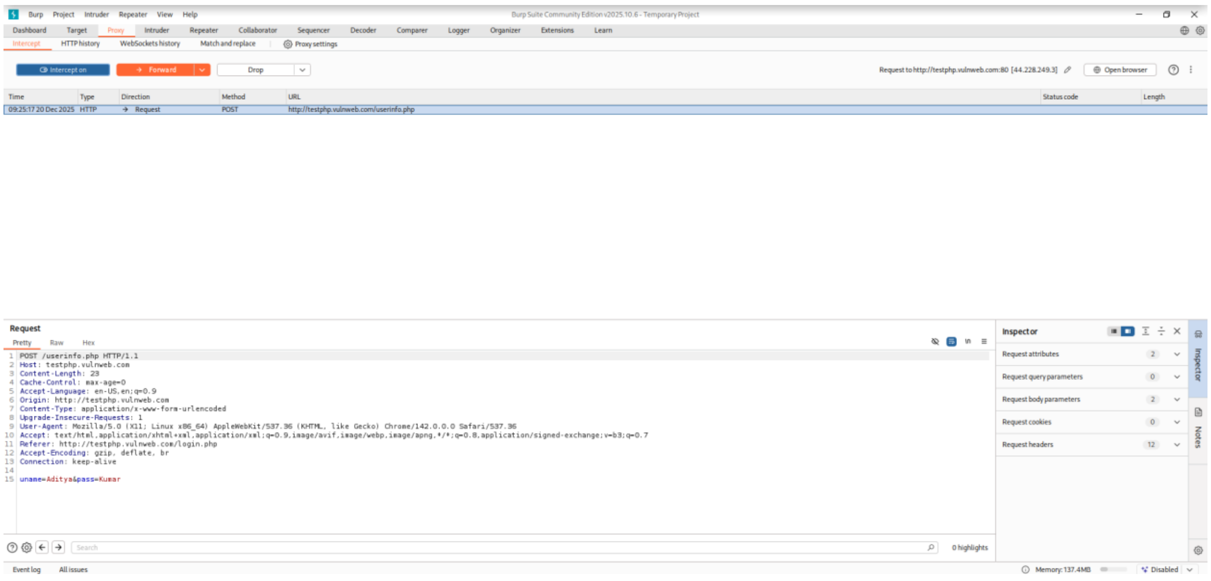
Warning: This is not a real shop. This is an example PHP application, which is intentionally vulnerable to web attacks. It is intended to help you test Acunetix. It also helps you understand how developer errors and bad configuration may let someone break into your website. You can use it to test other tools and your manual hacking skills as well. Tip: Look for potential SQL injections, Cross-site Scripting (XSS), and Cross-site Request Forgery (CSRF), and more.

Step8: Go to the Burp Suite and on the intercept.


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.10.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser
Intercept is on
Learn more
Open browser

Messages between Burp's browser and your target servers are held here. This enables you to analyze and modify these messages, before you forward them.

Step9: go back to the signup page, and enter any credentials. Click enter, the moment you click "enter" you will be redirected to the burp suite, with the following information.


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.10.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser
Request
Pretty
Raw
Hex
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 2
Request cookies 0
Request headers 12
Inspector
Notes
Event log
All issues
Memory: 137.4MB
Disabled

Request to http://testphp.vulnweb.com:80 [44.228.249.3]

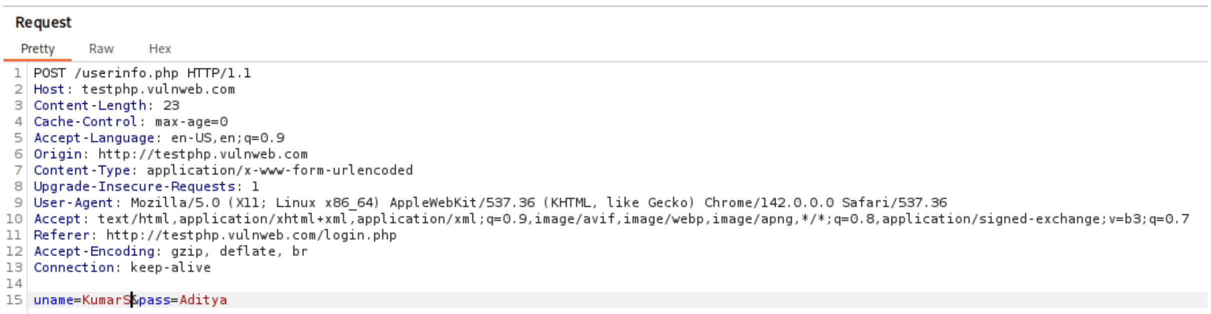
```
Time                Type  Direction  Method  URL                                   Status code  Length
09:25:17 20 Dec 2025 HTTP  → Request  POST    http://testphp.vulnweb.com/userinfo.php
```

```
POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Length: 23
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://testphp.vulnweb.com/login.php
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

uname=Aditya&pass=Kumar
```

Step10: We can edit and forward the request.

Edited request:

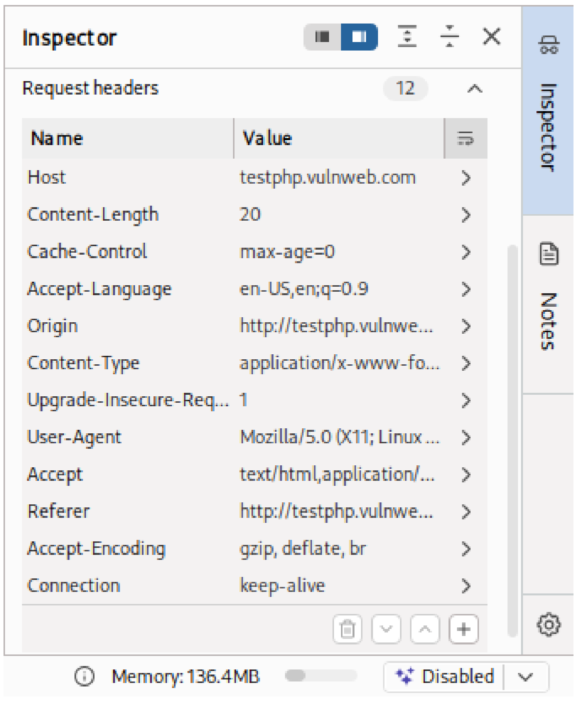


```
POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Length: 23
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://testphp.vulnweb.com/login.php
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

uname=KumarS&pass=Aditya
```

How to see HTTP Request headers?

We can see the headers when we intercept the traffic using burp suite:



Inspector

Request headers 12

Name	Value
Host	testphp.vulnweb.com
Content-Length	20
Cache-Control	max-age=0
Accept-Language	en-US,en;q=0.9
Origin	http://testphp.vulnwe...
Content-Type	application/x-www-fo...
Upgrade-Insecure-Req...	1
User-Agent	Mozilla/5.0 (X11; Linux ...
Accept	text/html,application/...
Referer	http://testphp.vulnwe...
Accept-Encoding	gzip, deflate, br
Connection	keep-alive

These are HTTP Request Headers sent by the client to the server as part of an HTTP POST request.

Web Functionality:

Server-Side Functionality:

Early websites only showed fixed (static) pages where everyone saw the same content. Modern websites still use some static content, but much of what users see is created dynamically by server-side programs. These programs run on the server, take input from users (such as URL parameters, cookies, form data, or even browser details), process it, and generate customized responses for each user. Server-side web applications use various technologies like programming languages, web frameworks, servers, databases, and other backend systems to provide this functionality.

The Java Platform:

The Java Platform (Enterprise Edition) is widely used for large, enterprise-level web applications because it supports scalable, modular, and reusable designs and runs on many operating systems. Java web applications use components like EJBs for complex business logic, POJOs for simple objects, Servlets to handle web requests, and web containers such as Tomcat or WebLogic to run the applications. Many applications also use open-source libraries for tasks like authentication, presentation, database handling, and logging, which save development time but can introduce security risks if those components contain vulnerabilities.

ASP.NET:

ASP.NET is Microsoft's web application framework and competes with the Java platform. It is built on the .NET Framework, allowing developers to write web applications using languages like C# or VB.NET. ASP.NET follows an event-driven programming style similar to desktop applications, and with tools like Visual Studio, it makes web development easy even for beginners. While it includes built-in protections against some common security issues, its ease of use means many applications are created by inexperienced developers who may overlook important web security risks.

PHP:

PHP started as a small personal project but has grown into a powerful language for building web applications, often used with free technologies in the LAMP stack (Linux, Apache, MySQL, PHP). Many popular open-source applications like forums, webmail, shopping carts, and wikis are built using PHP. Because PHP is free and easy to learn, it is widely used by beginners, and its earlier design made it easy to introduce security flaws. As a result, PHP applications have historically had a higher number of security vulnerabilities, sometimes due to issues in the applications themselves and sometimes due to flaws in the PHP platform.

Client-Side Functionality:

Client-side functionality provides the user interface that lets users interact with a web application and send input to the server, as well as display the server's responses. Since web applications run in browsers, they use common web technologies, but developers extend and combine these in different ways. The use of client-side technologies has evolved quickly, leading to more complex and interactive web applications.

HTML:

HTML is the basic language used to create web pages and define how content is structured and displayed in a browser. It uses tags to organize text and elements on a page. While it started as a simple formatting tool, HTML has evolved into a powerful language capable of building complex and interactive user interfaces.

Hyperlinks:

Hyperlinks are a common way for users to communicate with a server in web applications. When a user clicks a link, the browser sends a request to the server, often including preset parameters in the URL. These parameters are added by the server, not typed by the user, and they provide information such as an item ID or language choice. The server reads these values and uses them to decide what content to send back to the user.

Forms:

HTML forms allow users to send more flexible input to a web application than hyperlinks, such as usernames, passwords, or other data. When a form is submitted, the browser sends an HTTP request (usually using POST) that includes user-entered data, hidden fields, preset URL parameters, and cookies, all of which can affect how the server processes the request. Form data is typically sent using URL-encoded format or multipart format, depending on how the form is configured, and the server uses this information to control application logic and responses.

JavaScript:

JavaScript allows web applications to do more than just send data to the server by enabling processing directly in the user's browser. This improves performance by reducing unnecessary server requests and enhances usability by updating parts of a page without reloading it. JavaScript is commonly used to validate user input, change the interface dynamically, and control browser behavior through the page's structure (DOM). Modern applications often use AJAX, which lets pages send background requests to the server and update content smoothly, though these features can also introduce security risks if not implemented carefully.

<u>Thick Client Components:</u>

Some web applications use thick client components that go beyond JavaScript by adding powerful features through browser plug-ins or installed software. These components run custom code on the user's computer and can greatly extend what a web application can do. Common examples include Java applets, ActiveX controls, and Shockwave Flash objects, which provide advanced functionality but can also introduce security risks.

<u>State and Sessions:</u>

- **State**: Information about what a user is doing in a web application, such as login status or items in a shopping cart, remembered across multiple pages.
- **Session**: A server-side storage area that keeps a user's state data while they use a web application.
- **Cookies**: Small pieces of data stored in the user's browser that help identify the user and link them to their session.

Web applications need a way to remember what a user does as they move from one page to another. This is called state. Because HTTP does not remember users, applications store user data (like items in a shopping cart) in a session, usually on the server. Each time the user does something, the session is updated and used to show the correct information later. Some applications store this data on the user's device, but this can be risky if users change it. To recognize users across requests, applications give each user a unique session ID, most often stored in cookies.

<u>Encoding Schemes:</u>

Why Encoding is needed?

Web applications use different encoding methods to safely handle special characters and binary data because HTTP and HTML are text-based. When testing or attacking a web application, data often needs to be encoded correctly so it is processed as expected. Sometimes, changing how data is encoded can make the application behave in unexpected ways.

<u>URL Encoding:</u>

URL encoding is used to safely send characters in a URL that are not allowed or have special meaning. URLs can only use certain readable ASCII characters, so other characters are changed into a special format. In URL encoding, a character is replaced with a % followed by its two-digit hexadecimal ASCII code. For example, %3d means =, %20 means a space, and %25 means %. Also, the + symbol is often used to represent a space.

<u>Unicode Encoding:</u>

Unicode encoding is used to represent characters from all languages in web applications. One method uses 16-bit Unicode, where characters are sent as %u followed by a hexadecimal code (for example, %u2215 for /). Another method is UTF-8, which uses one or more bytes, each written as % plus a hexadecimal value (for example, %c2%a9 for ©). Unicode encoding is important in web security because it can sometimes be used to bypass input filters if the application processes Unicode but the filter does not handle it correctly.

<u>HTML Encoding:</u>

HTML encoding is used to safely display special characters in a web page. Some characters, like <, >, and &, have special meaning in HTML, so they must be encoded to be shown as text. This is done using HTML entities such as &lt; for < and &amp; for &, or by using numeric codes like &#34; or &#x22; for quotes. HTML encoding is important for security because it helps prevent attacks like cross-site scripting by making sure user input is treated as text, not code.

<u>Base64 Encoding:</u>

Base64 encoding is used to safely represent binary data using only readable ASCII characters. It is commonly used for things like email attachments, HTTP login details, cookies, and other web data. Base64 works by converting data into a fixed set of 64 characters and may add = signs at the end for padding. Web applications often use it to send or hide data, so it is useful to recognize and decode it.

<u>Hex Encoding:</u>

Hex encoding is another simple method that represents data using hexadecimal characters. It is often used to send binary data in cookies or parameters and is easy to identify and decode.