

# **Penetration Testing**

## **Web Storage (User Experience)**

(English)

**Author:**

Abdulrahman Abdullah

You can find me on LinkedIn:

<https://www.linkedin.com/in/abduc>

Arabic version :

<https://www.exploit-db.com/docs/50020>

<https://packetstormsecurity.com/files/163187/Penetration-Testing-Web-Storage-User-Experience.html>

Translated by: @emyovfelipovsky

**Disclaimer:** All mentioned exploits on this paper had been responsibly disclosed and are now patched. The information and exploits on this paper has been published for **research** purposes only. Use these at your own discretion, the author cannot be held responsible for any **damages** caused. The views expressed on this site are our own and do not necessarily reflect those of our employers.

Index:

0x00 = whoami

0x01 = Introduction

0x02 = What is JavaScript (JS)

0x03 = JavaScript Engines

0x04 = HTML5 Web Storage

0x05 = How to Discover it and Exploit it

0x06 = Reflected XSS to Stored XSS with Lab

0x07 = Tips

0x08 = References

# Introduction

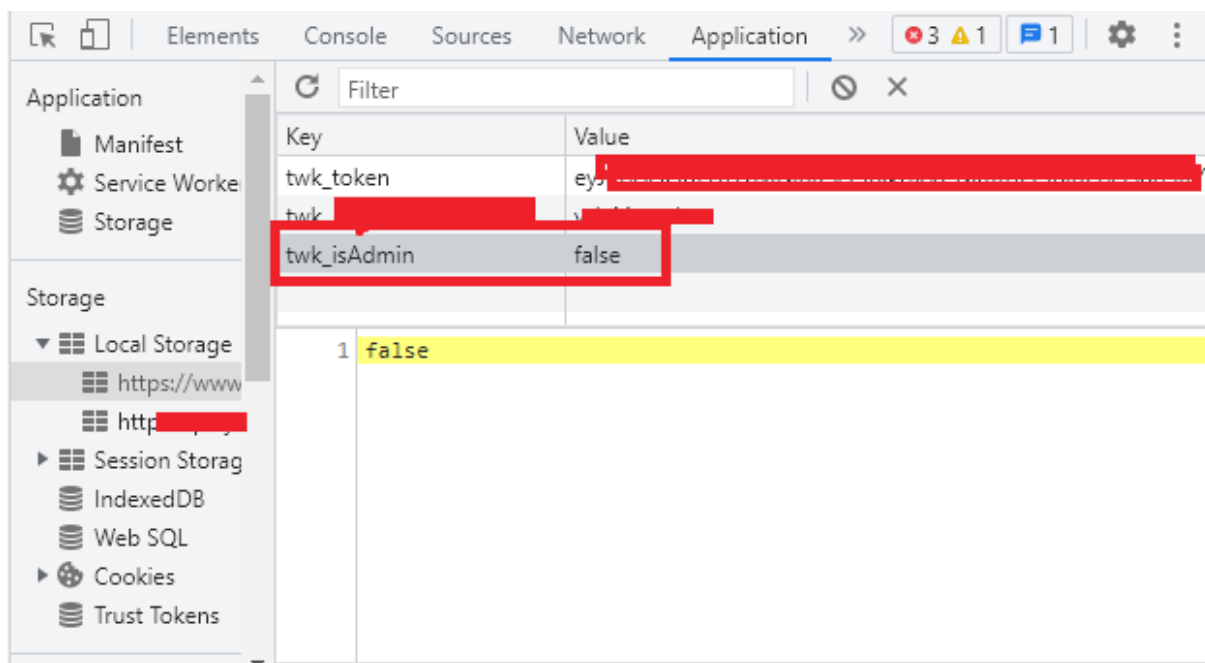
In the name of Allah the Merciful

Peace, mercy, and blessings of Allah upon you.

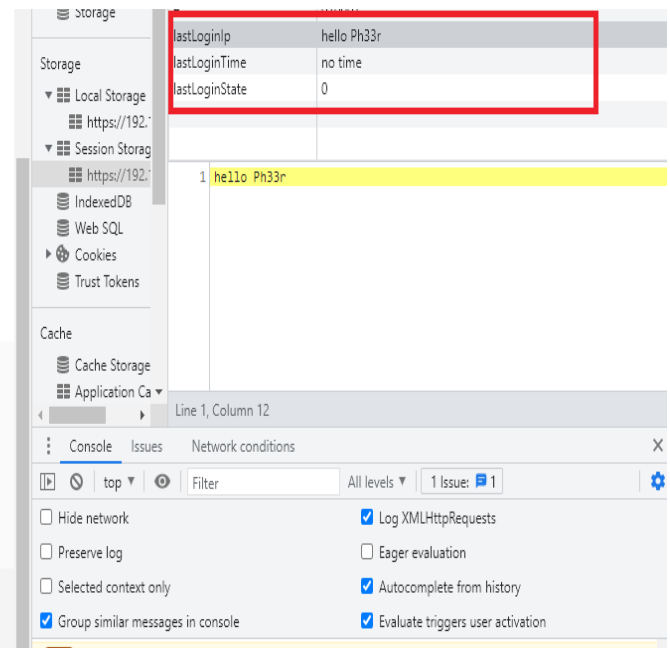
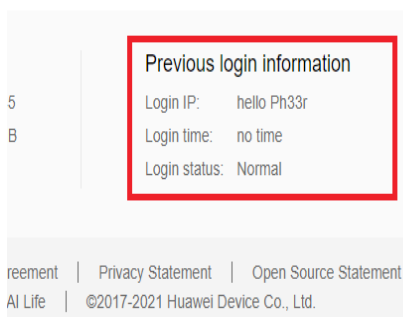
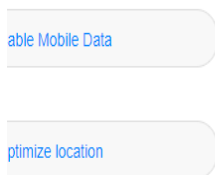
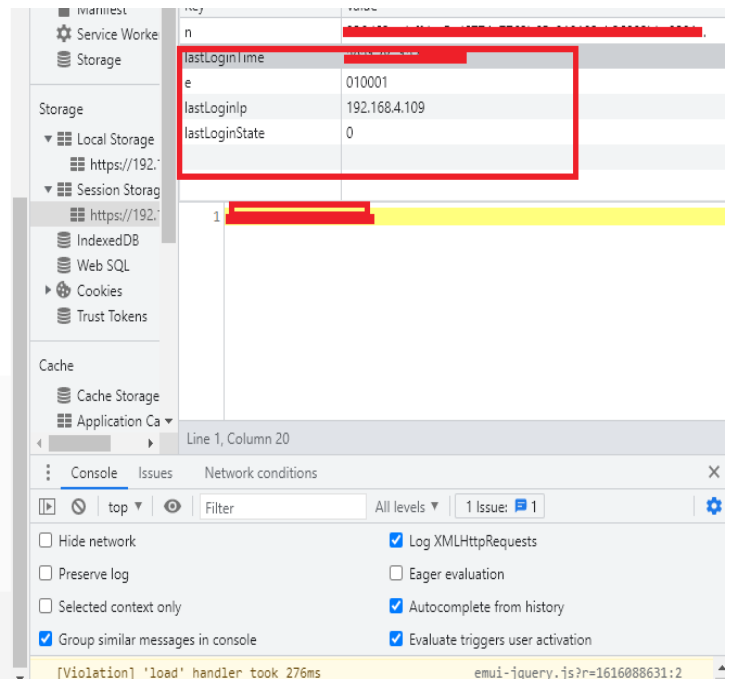
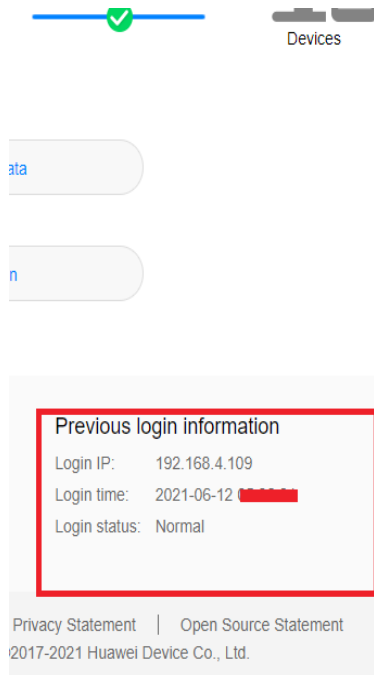
## 0x01= Introduction

JavaScript is considered the hidden treasure for web application penetration testers. In this paper, we present a novel approach in discovering web storage **vulnerability**. To the best of our knowledge, tools such as Zed Attack Proxy (ZAP) and the Burp Suite toolkit do not **check for this** type of **vulnerability**,

For our literature review on the different ways to exploit this vulnerability, we **took a look at** the Open Web Application Security Project ([OWASP](#)) [HTML5 Security Cheat Sheet](#). On the Storage API section, only two local storage based attacks are discussed. These aim to either steal data saved in local objects, or load malicious data into them. However, we were able to discover several other Web Storage vulnerabilities. Here are few examples:



# Introduction



# Introduction

**Do we have your attention?** Are you now interested in a penetration testing technique that, up until the writing of this paper, performs tests that cannot be done by other pentesting tools? This is because the aforementioned technique relies on values stored as strings in the browser. If these values are not passed **by the browser** to the application in the request, the application or **site** will use default values. Hence, it can only be tested through the browser using old school manual techniques.

Next, we take a deep dive into the details **of this technique** . Then, we explore different methods of exploitation.

# What is JavaScript (JS)



## 0x02 = What is JavaScript (JS)

JavaScript is a high level programming language used, in principle, **in web browsers** to create highly interactive web pages. It is currently developed by Netscape and Mozilla.

As defined in Wikipedia, **JS is a high level programming language used in web browsers to add interactivity to your web pages** in addition to many other utilizations. JS first gained popularity among amature coders and it quickly got noticed by professional programmers who further developed it to become the language that draws the most attention since 2015 to this day.

JavaScript files are created by adding the extension .js to explicitly define a file as a JS file.

## JavaScript in Web Applications

The **effect** of JS on web application programming is substantial. It establishes the foundation of any web developer. You can hardly find a web developer who can't deal with JavaScript code. Furthermore, JS is equally utilized in web design as it is in web development. It offers tremendous engaging features to both web designers and developers. This is especially true with the **release** of modern JS libraries such as Vue.js, Angular, React, and not to mention, jQuery..

# What is JavaScript (JS)

## What makes JavaScript Special (important)

JS is used on the client side however **and** is characterized by several features and techniques. These are:

- It **executes** on the client side: for example, it can be used to validate user input before sending a web request to the server.
- It's a relatively easy to learn and close to English.
- It is an independent programming language and fundamentally unrelated to Java which is a common misconception.
- It offers high control over web browsers.
- It is fast and highly interactive.
- It features rich interfaces. For example, you can drag and drop components to enhance your interface with the required elements.
- It is a functional programming language.
- It can be used to edit and update CSS and HTML files.
- JS can be used to interact with browser cookies as well as ajax and comet requests.
- **Saving Data on devices's Local Storage (important)**

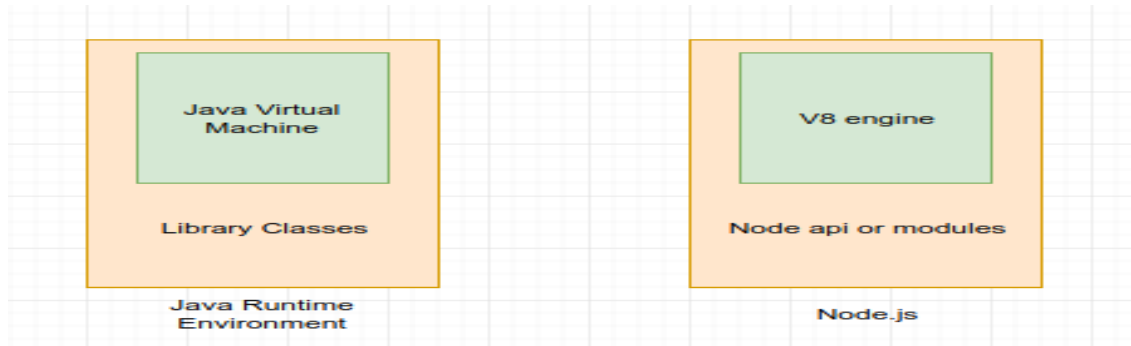
## JS Browser Limitations

When working with web browsers, there are some limitations on coding with JS. This aims to protect users' data and files from malicious websites. For example:

- Visited websites cannot gain access to user's files unless the user uploads such files to the site using the File Input element or the Drag and Drop feature.
- The same is true when different websites are visited using different tabs on the same web browser. A JS code running on a web page on one tab cannot access another site's data on another opened tab.
- When a web page that has a JavaScript code is loaded at the client's side, that JS code can access and exchange data with the site's web server which sent the page, the origin server, in a simple and direct way. However, this becomes more complex if the site is loaded from a different origin server. In other words, a JS code on a google.com page can not access or exchange data on exploit-db.com unless sufficient permission is granted by the latter. This policy is known as the Same-origin Policy and it is created to confine malicious JS code in an isolated domain that it can't escape.

# JavaScript Engines

## 0x03 = JavaScript Engines



Node.js is a JavaScript runtime environment. Sounds great, but what does that mean? How does that work?

The Node run-time environment includes everything you need to execute a program written in JavaScript.

Node.js came into existence when the original developers of JavaScript extended it from something you could only run in the browser to something you could run on your machine as a standalone application.

Now you can do much more with JavaScript than just making websites interactive.

JavaScript now has the capability to do things that other scripting languages like Python can do.

Scripting languages are designed to perform tasks to control developed applications and are processed through an interpreter, rather than a compiler.

### Node.js

"Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine"

JS is run on Chrome using its V8 engine, an open-source engine written in C++, as every browser runs its own engine. This is the same engine used by Node.js.

Chrome's V8 can be run independently or by including it in any application written in C++. Moreover, it has add-ons allowing you to write C++ code then make it available to be used in JavaScript.

This engine takes JS code as input then it outputs it as machine code to be instantly executed. This is because machine code is a low-level programming language, more like a group of 1s and 0s, that a computer can execute directly without the need to interpret it or convert it to another language first.

For more information on JS Engines, please refer to:

[https://en.wikipedia.org/wiki/JavaScript\\_engine](https://en.wikipedia.org/wiki/JavaScript_engine)



# HTML5 Web Storage

## 0x04 = HTML5 Web Storage



There are several advantages of storing data directly in a web browser. Most importantly, getting quick and direct access to the database independently of the network. There are currently four approaches, in addition to stopping, to store data on the client side:

1. Cookies
2. Local Storage
3. Session Storage
4. IndexedDB
5. 5. WebSQL (insignificant)

# HTML5 Web Storage

## Cookies

This is the classic approach to store data: saving simple data strings to a file. Traditionally, cookies are sent to the user, stored then returned to the server in subsequent requests. Cookies can be used to manage user accounts and track user information.

Cookies can be created, read, updated, and deleted using the following syntax:

```
// Create

document.cookie = "token=0x00";

document.cookie = "token_age=25;max-age=31536000;secure";

// Read (All)

console.log( document.cookie );

// Update

document.cookie = "token_age=08;max-age=31536000;secure";

// Delete

document.cookie = "token=0x00;expires=Thu, 01 Mar 1337 00:00:01 GMT";
```

## Advantages of using cookies

- Cookies can be used to communicate with the server.
- Cookies can be set to expire automatically without having to manually delete them.
- Cookies are generally supported in all major web browsers.

## Disadvantages of using cookies

- Cookies have a limited number and size.
- Cookies can only be stored as strings.
- Potential security vulnerabilities.

# HTML5 Web Storage

## Local Storage

Yes, Web Storage **uses** APIs (Application Programming Interfaces) including an API that stores key value data in the browser. This API solves the problem - by providing an easy and safe API to store simple data in cookies.

Although technically, we can only store strings in local storage, JSON files can be stored **tightly**. With Cookies, for example, this allows us to store more complex data locally.

Cookies can be created, read, updated, and deleted in Local Storage using the following syntax:

```
// Create
const user = { name: 'Ph33r', age: 08 }
localStorage.setItem('user', JSON.stringify(user));

// Read (Single)
console.log( JSON.parse(localStorage.getItem('user')) )

// Update
const updatedUser = { name: 'Ph33r', age: 08 }
localStorage.setItem('user', JSON.stringify(updatedUser));

// Delete
localStorage.removeItem('user');
```

## Advantages of using Local Storage when compared to cookies

- Local storage provides a simpler and easier interface for storing data.
- Local Storage is safer.
- Local Storage allows more data to be stored.

## Disadvantages of using Local Storage

- Local Storage allows only strings to be stored.

# HTML5 Web Storage

## Session Storage

This is the second type of APIs, a Session Storage API. It is quite similar to Local Storage. The difference is, with Local Storage, there is no expiration time setting for the data stored in it. With Session Storage, however, the saved data will be discarded at the end of the session.

**Session** can be created, read, updated, and deleted in Session Storage using the following syntax:

```
// Create

const user = { name: 'Ph33r', age: 08 }

sessionStorage.setItem('user', JSON.stringify(user));


// Read (Single)

console.log( JSON.parse(sessionStorage.getItem('user')) )


// Update

const updatedUser = { name: 'Ph33r', age: 08 }

sessionStorage.setItem('user', JSON.stringify(updatedUser));


// Delete

sessionStorage.removeItem('user');
```

# HTML5 Web Storage

## IndexedDB API

This is a more complex, yet comprehensive, solution for browsers to store data. IndexedDB API is an “API to store a large amount of data organized on the client side and use indexes on this data for high-performance retrieval” (Mozilla). This is a JavaScript-based, object-oriented database that allows us to store and retrieve data easily.

## How to use IndexedDB to Create an Application Offline

Compared to the other methods of storing in browsers, using IndexedDB is more complex. Before we can create/read/update/delete any data, we first need to open the database and create a data repository.

```
function OpenIDB() {  
  
    return idb.open('SampleDB', 1, function(upgradeDb) {  
        const users = upgradeDb.createObjectStore('users', {  
            keyPath: 'name'  
        });  
    });  
}
```

To create (or update) data, we need to follow the following steps:

```
// 1. Open up the database  
OpenIDB().then((db) => {  
  
    const dbStore = 'users';  
  
    // 2. Open a new read/write transaction with the store within the database  
    const transaction = db.transaction(dbStore, 'readwrite');  
    const store = transaction.objectStore(dbStore);  
  
    // 3. Add the data to the store  
    store.put({  
        name: 'Ph33r',  
        age: 08  
    });  
  
    // 4. Complete the transaction  
    return transaction.complete;  
});
```

# HTML5 Web Storage

To read the data, we need to follow the below steps:

```
// 1. Open up the database
OpenIDB().then((db) => {
  const dbStore = 'users';

  // 2. Open a new read-only transaction with the store within the database
  const transaction = db.transaction(dbStore);
  const store = transaction.objectStore(dbStore);

  // 3. Return the data
  return store.get('Ph33r');
}).then((item) => {
  console.log(item);
})
```

To delete the data, we need to follow the next steps:

```
// 1. Open up the database
OpenIDB().then((db) => {
  const dbStore = 'users';

  // 2. Open a new read/write transaction with the store within the database
  const transaction = db.transaction(dbStore, 'readwrite');
  const store = transaction.objectStore(dbStore);

  // 3. Delete the data corresponding to the passed key
  store.delete('Ph33r');

  // 4. Complete the transaction
  return transaction.complete;
})
```

# HTML5 Web Storage

## DB Advantages

- The most complex and structured data can be handled in each database.
- There can be several "databases" and "tables".
- More storage space.
- We can control how we interact with it.

## DB Disadvantages

- Very complex compared to other web storage APIs.

Let's now talk about Local Storage, Session Storage, IndexedDB.

Before we start to get to know the different exploitation methods stated above, we need to know that the difference between Local Storage and sessionStorage is that with sessionStorage as soon as the page or browser is closed, any stored data gets all cleared. In contrast, with Local Storage the data is saved by the browser and is not deleted without the user's own intervention. Most programmers prefer to use this method to store some data about the users experience such as storing the **users' language preference, do they**, prefer the dark or the light theme, do they like certain products etc. There are various ways to utilize stored data depending on the web application and the programmer.

## IndexedDB

IndexedDB is a database stored in the user's web browser and can only be viewed by the user himself. We will explain IndexedDB in more detail in the lab prepared for this paper.

# Discover and Exploit

## 0x05 = How to Discover and Exploit it?

There are two approaches to detect this kind of vulnerability. An easy one and a **difficult** one. Using the complex approach, prior knowledge of JavaScript code is required. You need to be able to read JS code and **trace** why the data was stored and whether it is displayed to the user as opposed to stored in log files for further processing by data and user experience analysts or not displayed.

We categorize the detection techniques of **this type** of vulnerabilities into two groups: easy and **difficult**.

### Section One: The Easy Approach

We experiment with adding HTML code within a Web Storage then check whether or not it is displayed to the user within the web browser.

Additionally, let's not forget that some websites store tokens within the Web Storage -.

Therefore, when a penetration test is performed, **the pentester can try modifying cookie values, such as changing**, the price of a product or tampering with the **site** permissions so that these modified values that are stored in the cookies are not verified. Whereas the values stored in the browser are verified.

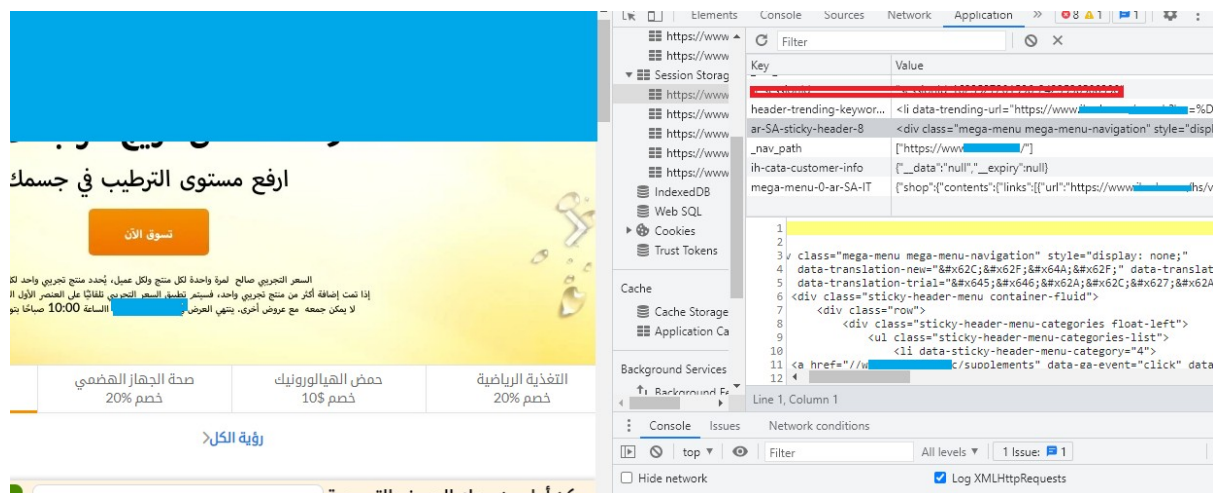
There are several scenarios that I found which will be explained in the second part, God willing.

Here's a simple example where a website stores the user's language **preference inside** sessionStorage. For instance, if the user chooses Arabic or any language other than English, the language, the language preference will be stored in sessionStorage. Now if you were to use BurpSuite or Zed Attack Proxy (ZAP) to test the website, it will only show you, in the server Response, the default, i.e. English.

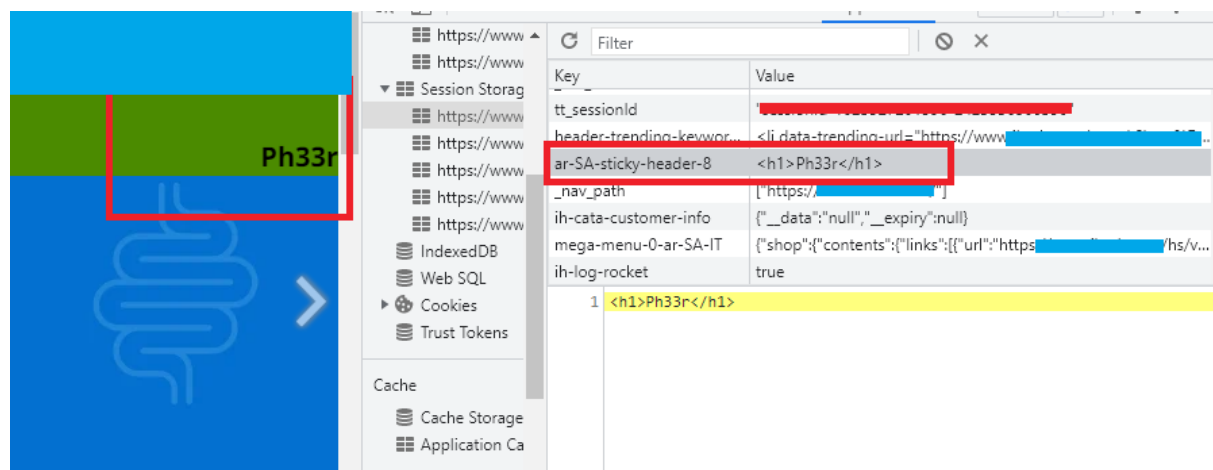
Therefore, we **should manually test** the website using the web browser to detect this type of vulnerability so that the default value is not displayed, **and the user experience stored value is displayed**. Let's follow the example in the **screenshots** below.



# Discover and Exploit



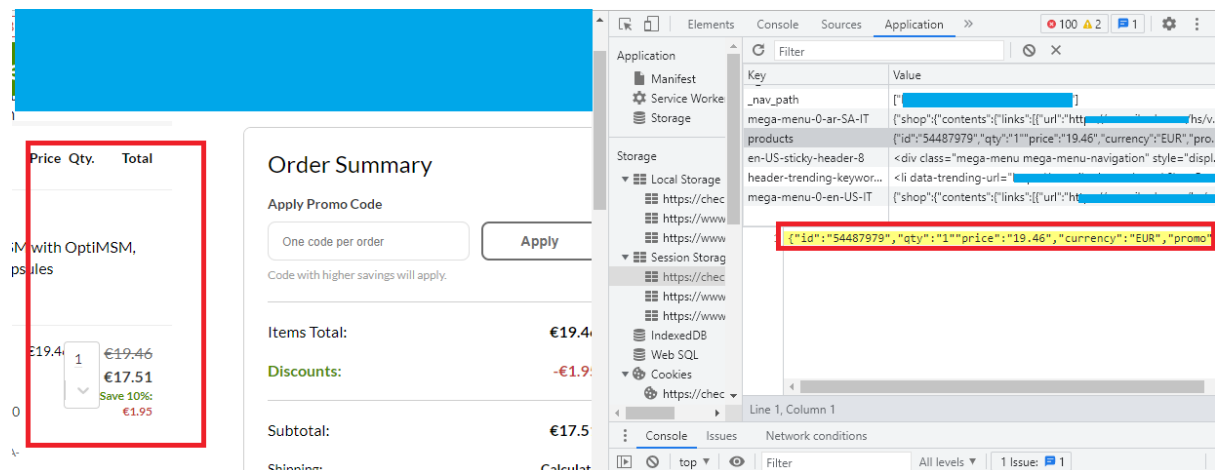
We inject our code now and update the page. Do we notice any change?



Operation successful! Now this is a Self-XSS vulnerability.

# Discover and Exploit

What about Business Logic vulnerabilities?



Price Qty. Total

Price	Qty.	Total
€19.46	1	€19.46
€17.51		
Save 10%: €1.95		

Order Summary

Apply Promo Code

One code per order

Apply

Code with higher savings will apply.

Items Total: €19.46

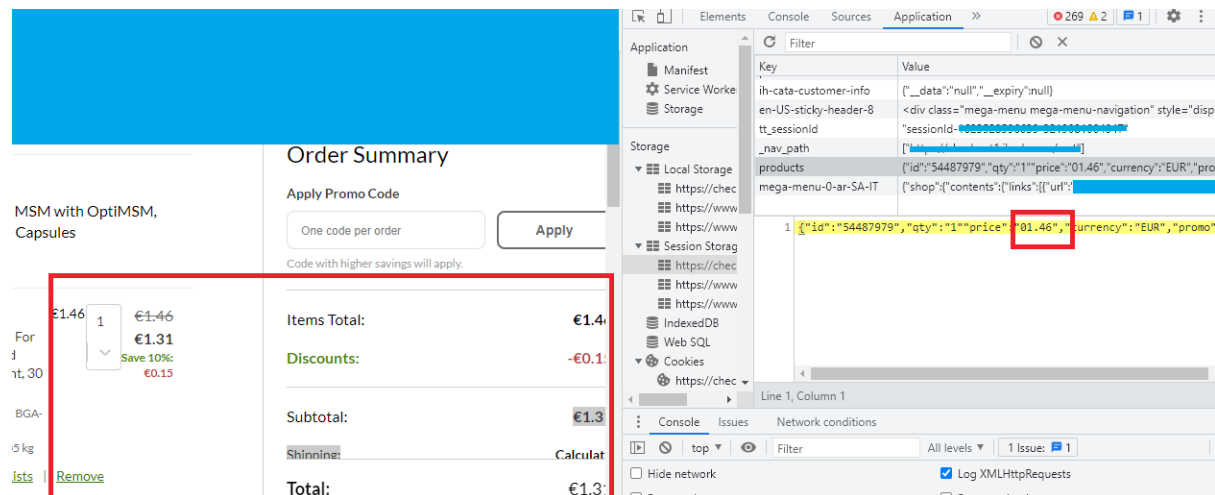
Discounts: -€1.95

Subtotal: €17.50

Shipping: Calculate

Application State:

```
{
  "products": [
    {
      "id": "54487979",
      "qty": "1",
      "price": "19.46",
      "currency": "EUR",
      "promo": "10%"
    }
  ],
  "sessionStorage": {
    "products": [
      {
        "id": "54487979",
        "qty": "1",
        "price": "19.46",
        "currency": "EUR",
        "promo": "10%"
      }
    ]
  }
}
```



MSM with OptiMSM, capsules

Price Qty. Total

Price	Qty.	Total
€1.46	1	€1.46
€1.31		
Save 10%: €0.15		

Order Summary

Apply Promo Code

One code per order

Apply

Code with higher savings will apply.

Items Total: €1.46

Discounts: -€0.15

Subtotal: €1.30

Shipping: Calculate

Total: €1.30

Application State:

```
{
  "products": [
    {
      "id": "54487979",
      "qty": "1",
      "price": "01.46",
      "currency": "EUR",
      "promo": "10%"
    }
  ],
  "sessionStorage": {
    "products": [
      {
        "id": "54487979",
        "qty": "1",
        "price": "01.46",
        "currency": "EUR",
        "promo": "10%"
      }
    ]
  }
}
```

# Discover and Exploit

## Section Two: The **Difficult** Approach

This approach will be explained in detail in another paper as some companies have not yet finished patching some vulnerabilities. After the bugs have been patched, section two will be published in a different paper which will include how these vulnerabilities were discovered.

### Part 2

00x01 = blhblh

00x02 = code review

00x03 = code review 2

00x04 = Blind XSS for (UX Analysts)

00x05 = 2 Tokens in (cookies, localStorage) .. !!

00x06 = Key Pollution ( storage )

00x07 = tips

# Exploit and Lab

## 0x06 = Reflected XSS to Stored XSS with Lab

How can I take advantage of the Self-XSS vulnerability in privilege escalation? Initially, you must find a Reflected XSS vulnerability. Through it, we can modify any value stored in the browser for the same site. This is done by injecting our payload inside any Web Storage by updating the values within it. Here's an example:

```
const updatedUser = { name: '<h1>Ph33r</h1>', age: 08 }

localStorage.setItem('user', JSON.stringify(updatedUser));
```

Now that we've got our Reflected XSS vulnerability and we know that the site also stores all user experience values in IndexedDB or localStorage, the first step we have to take is to get to the value key that we need to change.

For example, if we have inside the:

```
localStorage.getItem('Desired value key') OR
sessionStorage.getItem('Desired value key')
```

The screenshot shows a web browser with a shopping site. The sidebar menu is titled "ارفع مستوى الترطيب في جسمك" and includes a "تسوق الآن" button. The main content area displays various products with discounts. The browser's developer tools are open, showing the "Application" tab with a list of storage items. The "ar-SA-sticky-header-8" item is selected, showing its value as a complex HTML structure. The console shows a message "Log XMLHttpRequests".

```
sessionStorage.getItem('ar-SA-sticky-header-8')
```

# Exploit and Lab

After we get to the key and make sure it's the right key, we update the value inside the key.

```
sessionStorage.setItem('ar-SA-sticky-header-8', "code html")
```

Let's now assume we found a Reflected vulnerability, all we have to do now is inject the code that enables us to update the key value in a hidden way in the following way:

```
http://exploit-db.com/index?xssReflected=<script>sessionStorage.setItem('ar-SA-sticky-header-8',  
"code html")</script>
```

It is preferable to hide the payload within the same value while not changing the original value. As an example, when the victim browses the site, the payload is executed on his device. Every time the site is used, especially if the payload was injected into localStorage or IndexedDB in a hidden way, the payload will not change until it is deleted by the user.

## Lab

To prepare a small lab that doesn't need a local server, all you have to do is download and browse the files within any browser except Microsoft Internet Explorer.

There is an easy level lab with two XSS vulnerabilities to which you can apply all that's been mentioned.

<https://github.com/Ph33rr/webstorage>

# Tips and References

## 0x08 = Tips

- Storing data in Web Storage depends on the user experience in most websites:
  - Preferred language, preferred style, preferred currency, etc.
- Before **checking** the website, you should change all the default settings **and where they are stored** in the browser then test the values.
- At some online shopping websites, products added to the cart are stored in Web Storage if you visited as a *guest*. However, if you signed-in a *user* your data is stored in the database. You should check this before the penetration test.
- You can also find tokens stored in Web Storage. If you don't find them, try to use *Remember Me...*!!

## 0x09 = References

[https://www.researchgate.net/publication/](https://www.researchgate.net/publication/259081595_An_Investigation_into_Possible_Attacks_on_HTML5_IndexedDB_and_their_Prevention)

[259081595\\_An\\_Investigation\\_into\\_Possible\\_Attacks\\_on\\_HTML5\\_IndexedDB\\_and\\_their\\_Prevention](https://www.researchgate.net/publication/259081595_An_Investigation_into_Possible_Attacks_on_HTML5_IndexedDB_and_their_Prevention)

[https://en.wikipedia.org/wiki/JavaScript\\_engine](https://en.wikipedia.org/wiki/JavaScript_engine)

<https://en.wikipedia.org/wiki/JavaScript>

[https://www.w3schools.com/html/html5\\_webstorage.asp](https://www.w3schools.com/html/html5_webstorage.asp)

<https://academy.hsoub.com/programming/html/html5/%D8%A7%D9%84%D8%AA%D8%AE%D8%B2%D9%8A%D9%86-%D8%A7%D9%84%D9%85%D8%AD%D9%84%D9%8A-local-storage-%D9%81%D9%8A-html5-r362/>

<https://arabicprogrammer.com/article/3636627125/>

<https://www.tutomena.com/what-is-javascript/>

<https://medium.com/free-code-camp/what-exactly-is-node-js-ae36e97449f5>