service worker lab & Research

# introdction

In this lab I will be exploring service workers and showing a vulnerability (persistent XSS) lay dormant with in them.

I will by using some JavaScript based technologies like node.js nodemon.js and express.js all will be explained as we go along. A basic understanding of Java Script is not required but it can help you to better understand the lab

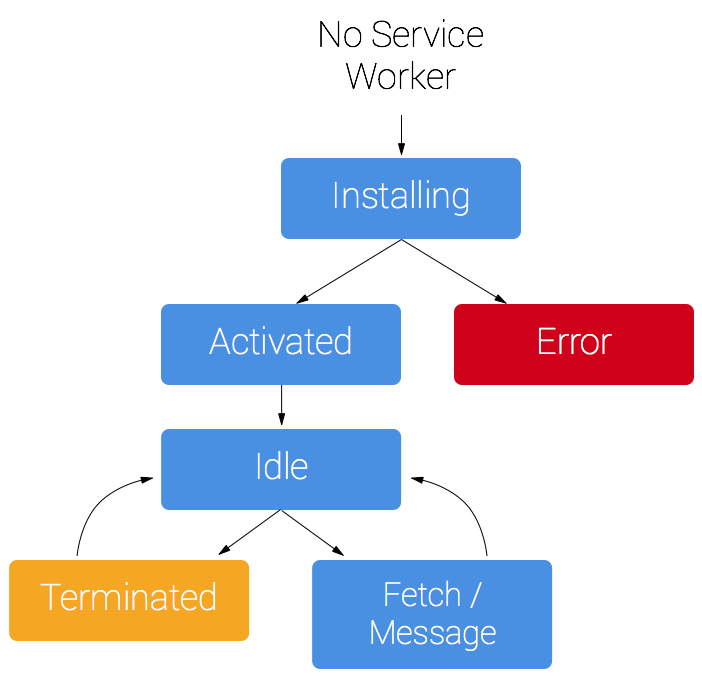
# what are Service WORKERS?

A service worker is a JavaScript file or script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. Today, they already include features like push notifications and background sync .The core feature discussed in this tutorial is the ability to intercept and handle network requests, including programmatically managing a cache of responses. In laymen terms it provides developers the ability to write web apps that can work of line and updates after internet connection is available.

## Things to note about a service worker:

1. It's a [JavaScript Worker](https://www.html5rocks.com/en/tutorials/workers/basics/), so it can't access the [DOM](https://www.w3schools.com/js/js_htmldom.asp) directly. Instead, a service worker can communicate with the pages it controls by responding to messages sent via the [postMessage](https://html.spec.whatwg.org/multipage/workers.html" \l "dom-worker-postmessage) interface, and those pages can manipulate the DOM if needed.
2. Service worker is a programmable network proxy, allowing you to control how network requests from your page are handled.
3. It's terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's onfetch and onmessage handlers. If there is information that you need to persist and reuse across restarts, service workers do have access to the [IndexedDB API](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API).
4. Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out [Promises, an introduction](https://developers.google.com/web/fundamentals/getting-started/primers/promises).

## The service worker Life cycle

A service worker has a lifecycle that is completely separate from your web page.

In order to install a service worker for a site. You will have to register it. This registration will cause the browser to start the worker installation step at the background.

Typically during the install step, we'll want to cache some static assets. If all the files are cached successfully, then the service worker becomes installed.

When installed, the activation step will follow and this is a great opportunity for handling any management of old caches.

After the activation step, the service worker will control all pages that fall under its scope, though the page that registered the service worker for the first time won't be controlled until it's loaded again. Once a service worker is in control, it will be in one of two states: either the service worker will be terminated to save memory, or it will handle fetch and message events that occur when a network request or message is made from your page.

## Important to know

During this lab we will be able to use service worker through localhost, but to deploy it on a site you'll need to have HTTPS setup on your server.

## setting uP

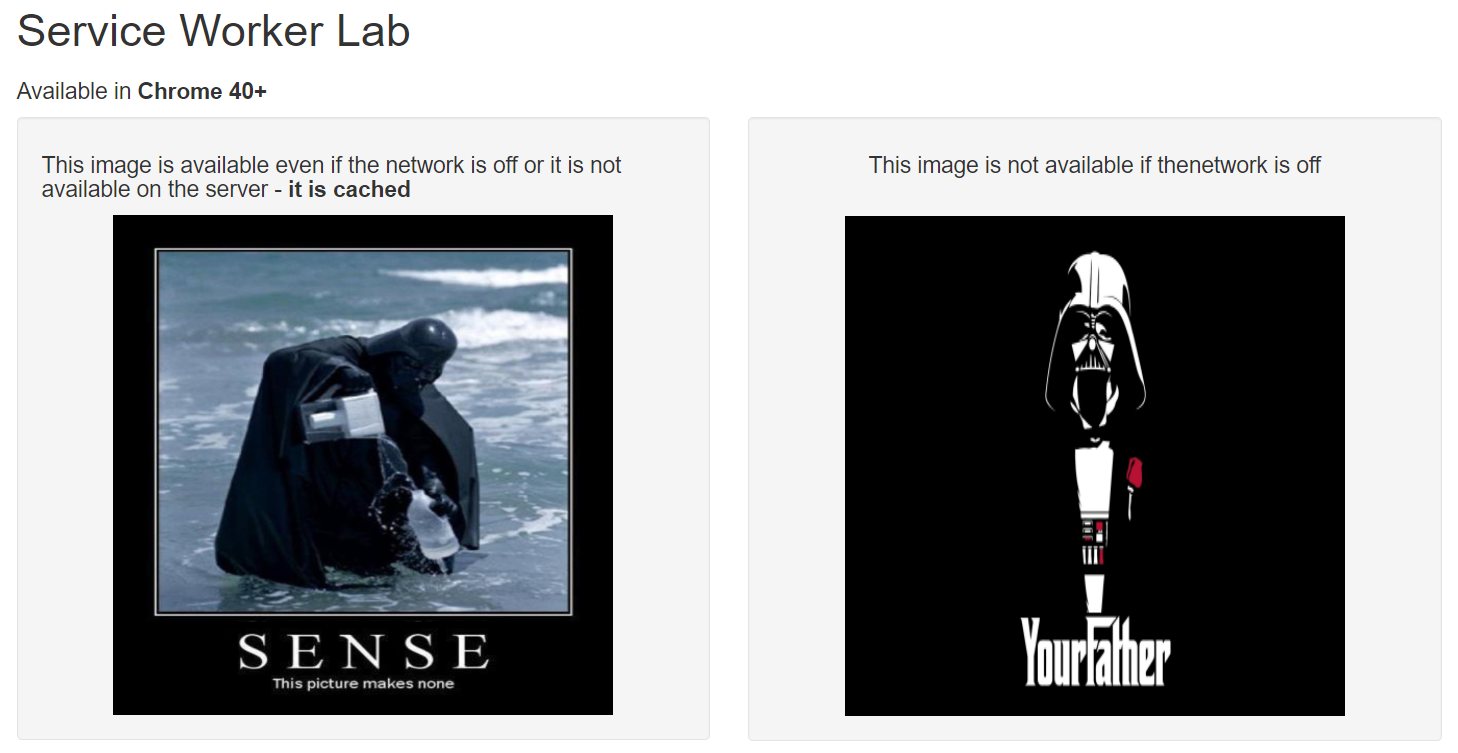
In this section we will set up our environment.

First thing to do is installing [Node.js](https://nodejs.org/en/) on out machine. Node.js is a JavaScript run time environment that means it's allow us to execute .js file outside of the browser.

A part of Node.js ecosystem Is Node Package Manager (NPM or npm). We will use this tool in order to include different code packages and libraries.

Just follow the instructions on [https://nodejs.org](https://nodejs.org/en/) and install node.

After installing node.js go in [here](https://github.com/TheFrohlich/service-worker-lab) and download the repository. Open the command line and cd in to the library containing the unzipped repository and type in: npm install , This will tall npm to install all of our dependencies that are listed in the package.json file. Run the command *node server.js* to start the server, if all is right you will see the message *"http Server is listening on 4000 "*.

In chrome bowser navigate to localhost:4000/index.html . You should see a page looks like this:

That's it our server is up and running.

## Task 1 – creating a service worker

As explained earlier as service worker (sw) is just a JavaScript file running at the background. Now we will create one and explore some of it usages.

Create a file called sw-1.js and using a text editor insert the code from the file appendix/service-worker-code.js

This is a standard service worker implementation as public by google + some minor changes to fit our file system

Now add this code to your index.js file

if ('serviceWorker' in navigator) {

window.addEventListener('load', function () {

navigator.serviceWorker.register('/sw-1.js',{scope:'/'}).then(function (registration) {

// Registration was successful

console.log('ServiceWorker registration successful with scope: ', registration.scope);

}, function (err) {

// registration failed :(

console.log('ServiceWorker registration failed: ', err);

});

});

}

Q.1: Explain the code above

## Task 2 – working Online and offline

One of the most powerful fetchers of the service worker is his ability to intercept fetch calls (AJAX) and then cache them in to the local cache.

In this task we will explore this fetcher.

Read the file sw-1.js and do the falling:

Q.2: Cut of your internet connection, go to change the name of the file cached-image.jpg to cached-image-1.jpg save it and refresh the page (F5).

Describe what happened and why.

Q.3: Reopen the network connection and open the browsers developer-tool (F12) navigate to the application tab. Find the Cache Storage tag and delete the cache. Restart the site, describe and explain the results. Now rename the

## Task 3 – Presistent XSS using service worker

XSS – **cross site scripting** is a form of attack including injecting a malicious JavaScript in to the client and use it for all sort of things (cookie stilling, info stilling, transforming the client in to a bot etc...).

Combining XSS with service worker will create a persistent XSS attack meaning the malicious code will be stored at the browser and in some forms of attack can only be remove by manually delete the cache.

Insert a script tag referring to the attack.js file like so:

<script src="./index.js"></script>

<script src="./attack.js"></script>

Now click the button. A popup will appear. Your page is now infected.

Q.4: Describe what happen after you refresh the page

Q.5: Get over to attack.js, explain what the malicious code is doing

Q.6: Delete the attack script tag and refresh the page, do you see any change? Explain.

Q.7: At the command ling running the server click ctrl + c to stop it and refresh the page do you see any change? Explain.

Q.8: Can you find a fix for this vulnerability?