Keep our web applications safe

A security evaluation of Service Workers

Adam Erickson
Oscar Nielsen

9-23-2019

Supervisor : George Osipov
Examiner : Christer Bäckström
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Adam Erickson, Oscar Nielsen
Abstract

With the ever-expanding internet, finding new ways to increase the user experience are vital in order to keeping concurrent users on your web application. One way to achieve this could be to implement a Service Worker to unlock more capabilities of a web application.

The purpose of this paper is to evaluate what new security vulnerabilities can arise when implementing a Service Worker. This could then be used to evaluate if the technology has evolved far enough to be used by a wider audience of programmers and users. The analysis in this paper will be presented in a security matrix that is based on four experiments and a complementary literature study on web-based attacks. This paper found that some new vulnerabilities must be considered when implementing a Service Worker in a web application. The worst of these is the Living Outside of Scope, which can be used by an attacker to secretly hijack a victim's computer even when the application is shut down. This paper concludes that the technology has evolved far enough so that a secure web application with the use of the Service Worker is possible, but there are still some new vulnerabilities that can become a problem if not considered.
# Table of Contents

1. Introduction .......................................................................................................................... 1
   1.1. Motivation ...................................................................................................................... 1
   1.2. Aim ................................................................................................................................. 2
   1.3. Research question ......................................................................................................... 2
   1.4. Delimitations ................................................................................................................. 2
   1.5. Report overview .......................................................................................................... 2
   1.6. Glossary ......................................................................................................................... 2

2. Theory .................................................................................................................................. 5
   2.1. Service Worker Development ....................................................................................... 5
       2.1.1. Service Worker ....................................................................................................... 5
       2.1.2. Promise API .......................................................................................................... 6
       2.1.3. Fetch API .............................................................................................................. 8
       2.1.4. Cache API ........................................................................................................... 8
       2.1.5. Background Sync API ......................................................................................... 8
       2.1.6. Push API and Notification API ............................................................................ 9
       2.1.7. Offline Browsing ................................................................................................ 9
   2.2. Web Security ................................................................................................................ 10
       2.2.1. Essential Service Worker Security ....................................................................... 10
       2.2.2. Proxy/man in the middle ..................................................................................... 10
       2.2.3. Cross-site scripting ............................................................................................. 11
       2.2.4. Social engineering ............................................................................................... 12
       2.2.5. Cache Attack ........................................................................................................ 12
       2.2.6. Fixed Attacks ....................................................................................................... 13
       2.2.7. Service Worker Vulnerabilities ........................................................................... 13
   2.3. Related Works .............................................................................................................. 15

3. Method .................................................................................................................................. 17
   3.1. Security Matrix ............................................................................................................ 17
   3.2. Experiments .................................................................................................................. 17
       Experiment 1 - Worker in the middle ........................................................................... 17
       Experiment 2 - Bad eternity with XSS ......................................................................... 18
       Experiment 3 - Pushing it to the limits ......................................................................... 18
       Experiment 4 - Storage attack ..................................................................................... 18
3.3. Ethical considerations........................................................................................................18

4. Result ..................................................................................................................................21

4.1. Results from experiments .................................................................................................21
   Experiment 1 - Worker in the middle ..................................................................................21
   Experiment 2 - Bad eternity with XSS ...............................................................................21
   Experiment 3 - Pushing it to the limits ..............................................................................21
   Experiment 4 - Storage attack ............................................................................................22

4.2. Security matrix ..................................................................................................................24

5. Discussion ............................................................................................................................25

5.1. Method ...............................................................................................................................25

5.2. Security Matrix ................................................................................................................25
   5.2.1. Service Worker Origin Name - SWON ..................................................................25
   5.2.2. Lifetime of Cached Data - LCD .............................................................................25
   5.2.3. Bad Caching Strategies - BCS ...............................................................................25
   5.2.4. Persistence ................................................................................................................26
   5.2.5. Living Outside of Scope - LOS ...............................................................................26
   5.2.6. Service Worker Position - SWP .............................................................................26
   5.2.7. Old in Cache - OC ....................................................................................................26
   5.2.8. Cache Swap ...............................................................................................................26
   5.2.9. Free From CORS - FFC ..........................................................................................27

5.3. Attack mitigations ..............................................................................................................27

5.4. Future Work .......................................................................................................................27

6. Conclusion ............................................................................................................................29

Reference ..................................................................................................................................31
1. Introduction
This is a Bachelor thesis report for the Bachelor of Science in Computer Engineering specialization software engineering (16 hp) at Linköping University. The thesis work is a collaboration between the University and Zenon. Zenon is a consulting company with an interest in new technologies involving web applications and native applications (for example Android applications). The web applications used for the experiments can be found on GitHub\(^1\).

1.1. Motivation
Web developers want full control of the user experience when the user is online and offline, as both are important to deliver a good experience. In the past Application Cache (AppCache) was used to obtain offline functionality for the application. AppCache is an interface that uses a manifest file to decide what should be stored in cache \([1]\). Most developers were not fond of AppCache as it was hard to debug and easy to get wrong \([1,4]\). This made way for a new, simpler solution which is now known as Service Worker.

Service Worker is a script that runs between the application and the server and is used to increase functionality of the application. It can be used for, background sync, push notifications and caching \([2]\). Popular websites that use Service Workers today are YouTube\(^2\), Google\(^3\), Twitter\(^4\), and Reddit\(^5\). There are also some web applications that use Service Workers, for example, Gmail\(^6\) and Outlook\(^7\).

With an increasing number of stable connections, is the support for offline functionality important? Statista\(^8\) reveals a decrease in production of desktop computers by 59.2 million units from 2010 to 2017. This decrease is predicted to continue with another 13.3 million until 2022. This trend is a consequence of the increasing numbers of mobile units (laptop, tablet, and smartphone) in use. Wireless connections are used more on mobile units than on desktops and disconnections have a much higher rate, as a wireless connection is more easily disrupted. This trend must be followed with offline support if developers want to provide high-quality user experience.

One important functionality of Service Workers is that they can be utilized if the user has a slow or flaky connection \([3]\). What makes this feature possible is the position of the Service Worker: The Service Worker does not run in the web application or on the server, it instead works locally in the user's browser in a

\(^1\) https://github.com/oscni938/video_test
https://github.com/oscni938/book_test
https://github.com/oscni938/live

\(^2\) https://www.youtube.com/sw.js
\(^3\) https://www.google.com/_/chrome/newtab-serviceworker.js
\(^4\) https://twitter.com/push_service_worker.js
\(^5\) https://www.reddit.com/sw.js
\(^6\) https://mail.google.com/mail/u/0/sw.js?7020=1&7361=1&8202369=1
\(^7\) https://outlook.live.com/mail/sw.js
separate thread. This functionality combined with the additional features is the biggest difference between AppCache and Service Workers.

The purpose of this tool is not only to decrease the load on servers and the bandwidth but also to give the users a quality experience on the internet. This experience is not only when the users have good and stable connections but also the times when they are unstable or flakey. Some big companies like Google and Reddit have started to implement Service Workers, but the majority of web applications that could benefit from it today are not using this new technology. Is this because the technology is quite recent and not well-known yet or is it because new security problems occur when implementing the Service Worker?

1.2. Aim
The aim of this paper is to explore the safety aspect of Service Worker. The focus will lay on different attacks and safety mechanisms to prevent them. A literature study was done to identify potential vulnerabilities, and some complementary experiments were conducted to further investigate the Service Worker. The result is presented in the form of a security matrix created with literature-based criteria [7,21].

1.3. Research question
The research questions in this report are the following:

- Which new vulnerabilities can arise in a web application with the use of a Service Worker?
- What are some possible mitigations for Service Worker related vulnerabilities?

1.4. Delimitations
The following limitations have been made during the work:

- Our study will only focus on the browsers Mozilla Firefox (66.0.3) and Google Chrome (74.0.3729.108) because they have the biggest support for the technology.
- Our work on Service Workers safety will not be tested on mobile phones or tablets.

1.5. Report overview
The report structure is as follows. In section 1.6 Glossary we will go over some specific terminology that is important for the understanding the rest of the report. Section 2 will go over the introduction of Service Worker and its lifecycle. Section 2 will also cover the most important APIs, some basic Service Worker security, describe what offline browsing, and give an introduction to different attacks the Service Worker can be used in. Section 3 describes the method used in order to answer our research question. Section 4 contains the results. Section 5 discusses the result and gives some possible mitigations of emerging security problems. The final section will conclude the report.

1.6. Glossary
The following are some important terminologies:

**Application programmable interface (API):** API is a type of interface that has clearly defined methods for the communication and interactions between different software. An API can simplify the programming of the software because the developer does not need to have knowledge about the underlying systems.

**Request:** Request is the web applications way of asking for data and updates over the network. The request contains information about the client and about the data requested from the server.
**Response:** Response is the answer the web application gets after a request. The response consists of the server’s answer and also includes the request the client has sent.

**Browser cache:** Browser cache is a specific cache the browser uses. This cache is a storage space where the browser stores specific data, this is not to be mistaken for Cache API (which will be described in section 2.1.4 Cache API).

**CPU cache:** CPU cache is the storage space where the computer stores temporary data in order to decrease access time from the main memory. This is done by increasing the locality of data as it will be faster to access when it is sequential.

**Event:** An event happens when some value in the application is changed or something is received from the server. An event can be triggered, for instance when a button is pressed, or the server wants to update a picture on the application.

**Cross-Origin Request:** A Cross-Origin Request is when a web application requests data from a domain that is not its own.
2. Theory
The theory has been divided into three subsections. Information about Service Workers, some commonly used APIs and offline browsing will be described in the Service Worker Development subsection. Essential Service Worker security and possible attacks will be given in the Web Security subsection, and the Theory will be concluded with a Related Works section to present previous work.

2.1. Service Worker Development
In order to understand security problems, good underlying knowledge about how to develop a Service Worker is needed. This section Aims to grant that knowledge by describing the Service Workers most commonly used APIs.

2.1.1. Service Worker
Progressive Web Application (PWA) is a new concept that Sheppard claims is essential to creating user-friendly web applications [2]. PWA is not a framework or specific language. Instead, it is a set of guidelines to create a web application that is fast, reliable and engaging [2]. In order to utilize these new qualities some support is needed from the browsers. One functionality is offline browsing that can be used to increase locality and offload the server. Offline browsing is possible by using the API called Service Worker.

A Service Worker is an API that runs as a middleman in the user's browser (see figure 2.1), it is responsible for handling the Hyper Transfer Protocol Secure (HTTPS) requests and acts depending on the web application developers implemented functionality. (Note: the reason for only HTTPS will be mentioned in section 2.2.1)

Figure 1. Service Workers position in the system

The position of the Service Worker makes it possible to cancel HTTPS requests and instead use the browser's functionality to fulfill its goals, for instance, send push notifications without the involvement of the server. This happens asynchronously so the user does not experience any performance loss. The Service Worker specifies which request and response pairs to save in the Cache API. This feature gives the
developer full control over what is cached. The Service Worker can do this by listening for events and then act accordingly.

The lifecycle of the Service Worker follows a specific path. No changes in this path can be made since its vital for the functionality of the Service Worker. The lifecycle is the following:

**Downloading and registration**

The first step in the lifecycle handles the download to the browser, which happens automatically the first time the user loads the application. It is praxis to first check if the browser has support for Service Worker before registration, as it will raise errors if not supported. After the download is complete, a registration of the Service Worker is needed in order to use it, this can be done at any type of event.

**Installation**

In order to use the Service Worker, installation is needed, the Service Worker will be installed in the user's browser.

**Waiting**

Waiting is an optional event that lasts until the Service Worker has been activated, the waiting state is shown in the browser as stopped. The Service Worker might also be in this state when activated but is waiting for new tasks to complete.

**Activation**

The Service Worker is activated and ready to be used by the browser. Activation happens at predetermined events (for example, on receiving a push event) or right after installation.

**Updating**

The update event is triggered when a Service Worker is updated. In order to avoid conflicts with older versions of itself running in another tab, the updated version stuck in waiting. This updated version will only be active if all clients with the old version are closed or after 24 hours.

2.1.2. **Promise API**

Promise API is an API that enables asynchronous functions without the help of the old version callbacks. Kambona et al. describe that callbacks are used when different functions depend on each other answer, in order to determine if they should continue or stop [5]. A callback functions by delegating the workload to another function which will later notify its caller that it is done. Using callbacks can easily result in unreadable code because it is hard to grasp what functions are actually executed. A small example can be seen in figure 2 where a function has both a success callback and a failure callback. The function is not limited to only have two callbacks as figure 2.

```javascript
function Success(){ // this function is called if opening the door was successful
//continue as usual
}

function Fail(){ //this function is called if opening the door was unsuccessful
//give error
}
```

*Figure 2. callback*
The functionality of callbacks is still desirable because the application can act differently depending on the user's actions. These actions will be sent as status codes through an HTTP packet to the server, which can then respond as intended. Kambona et al. describe the following problem with callbacks: when they are nested together, the code becomes hard to read as seen in figure 3 [5]. Figure 3 is a chain of callbacks as the previous steps needs to be fulfilled to do the next step. To obtain desired functionality, using nested callbacks is unavoidable.

```javascript
var home = require('house');

home.load('step1', function(gotoDoor){
    home.prompt('Do step 2', function(){
        home.load('step2', function(takeOutKeys){
            home.prompt('Do step 3', function(){
                home.load('step3', function(openDoor){
                    home.prompt('Do step 4', function(takeOutKeys){
                        home.prompt('Do step 5', function(openDoor){
                            home.prompt('Do step 6', function(walkIn){
                                home.prompt('Done');
                            });
                        });
                    });
                });
            });
        });
    });
});
```

**Figure 3. Illustration of Callback Hell/Pyramid of Doom**

This problem was solved by creating the promise API that handles the callback as an object [7]. This object can either be resolved or thrown away, which simplifies fault handling and avoids situations like Callback hell / Pyramid of Doom. Error handling is possible by using error propagation, this mechanism stops the chain if an error occurs and gives the error message. With error propagation the functions lower down in the chain will never execute unless all links before it has finished. An example can be seen in figure 4, where someone tries to go through a door, which is the same as in figure 3 but is much easier to follow. The first promise object created in figure 4 is the .fcall() function and each .then() function creates their own promise to chain to its predecessor.

```javascript
//use Q.fcall in Q library to create a promise
Q.fcall(gotoDoor)
    .then(takeOutKeys)
    .then(openDoor)
    .then(function(response){
        walkIn(response)
    }, function(error){
        //catch all errors
    })
    .done();
```

**Figure 4. Chain of Calls using Promise API**

Promise API is important for Service Worker because all supported APIs uses promises. Service Worker uses these promises, for instance, to display a version of the web application when the server cannot be reached. As the promise to receive new data from the server is rejected and old data is used.
2.1.3. Fetch API

Fetch API is used to asynchronous get access to databases and/or servers. This gives the web application the ability to request new data without the need to update the site. Fetch API was created to replace tools as XMLHttpRequest. The difference between Fetch API and older tools are that Fetch API uses the Promise API instead of callbacks. The steps that the fetch function goes through are the following: the return value from the server is a promise object that contains a response object. The response object consists of the answer from the server and it also has access to the request status code to determine if the promise is fulfilled.

fetch() has only one mandatory argument which is the Uniform resource locator (URL) to the desired data, but fetch() can also take in other configuration parameters that can be used by the server, for example, headers. This makes Fetch API more flexible and developer-friendly than the older tools.

A Service Worker uses Fetch API by listening to the events the web application is sending. When a fetch is happening the Service Worker will act as a middleman (see figure 1.) and look through the request and then decide where the request should be sent. The response sent back by the server will also go through the Service Worker, where it decides if the response should be sent to the application or if it something else is to be done. The Service Worker can also modify these requests and responses if it is implemented in that way. The Service Worker can also take these requests and responses and use them in another API. This means that the Service Worker has full control over the network traffic of the user.

2.1.4. Cache API

Cache API is a direct update of AppCache and gives the developer full control over what is saved in Cache APIs cache. This cache should not be mistaken for the browser cache since they are two different storage spaces. The developer also needs to manage the data, otherwise, it will be indefinitely stored in Cache API.

The API can only save HTTP responses linked with its request. The request is a URL to the data it wants to access. The HTTP response can be all different types of data the browser supports, for example, JSON or Blob. To search if the data is stored in the cache the requests are used as a key and the linked response is given as the return value.

A Service Worker can be implemented in a wide range of ways but can be divided into two groups according to Cache API usage: static and dynamic cache. The principle for static cache is, when the application is first loaded are selected parts stored in the API cache, for example the login page or a page stating that the application is offline. The dynamic cache is used to cache parts during runtime depending on what the user is doing. These two types of caching can also be used together in the Service Worker if the developer sees a benefit to gain for the application.

2.1.5. Background Sync API

Background Sync API is used to store data in a queue if its unable to send it through the network. This is useful because the application can then be closed, as the Service Worker will go through the queue and send the data to the right destination when the network connection returns.

One implementation of this is sending messages. The user can send messages through the application even when offline and no error occurs, when it later goes back online these messages will be sent to the server or database and synchronize up with the user. Another useful implementation can be to get data from the server, for example if the web application has background sync the user can start the application then close it down and the Service Worker would notify the user when the application has finished loading.
The reason background sync works is because the Service Worker runs independent of the network connection and catches the sync event. Background sync is mostly used in combination with other APIs for example Cache API or IndexedDB (a database that is stored in the browser) to store requests that should be synchronized.

2.1.6. Push API and Notification API
Push API is a Service Worker specific API, which is used to asynchronously receive data from a server without sending a request. The Service Worker uses the Push API by first subscribing to get notifications from the server which is done by using a Voluntary Application Server Identification for Web Push (VAPID). The server sends a subscribe request to the browser that asks the user if it wants to subscribe to push notifications. The VAPID key will be stored on the application even if communication is refused by the user because the user is still able to change to the notification settings. This is similar to how web sockets function as it can be used to send data to users, but the Push API grants additional features. The Reason for using Push API is that the notifications will be sent to the screen even if the application is closed for instance, the user could be sent a notification if they received a mail. These notifications are sent by using the Notification API which is not only limited to sending text but also has support for icons, pictures and other kinds of data. The Notification API also contains options for customizing the notification sent to the client, for example, it can use different sorts of vibrations and flashing lights which mimics how a native application could be used. These two APIs are then combined and used together as illustrated in figure 5.

![Figure 5: Push API and Notification API working together.](image)

2.1.7. Offline Browsing
A developer should have full control over the user experience, even if the user has a flaky connection or no connection at all. This is the purpose of offline browsing. Offline browsing is possible because the device has saved important data which is shown when a server cannot be reached. The functionality for offline browsing has been known since before, but with the help of a service worker data can be stored dynamically something that gives the user an even better experience than before.

A user can become offline in a wide variety of ways, the common denominator in all these is that the connection between the server and client are lost. Because the Service Worker runs in the browser does it not matter how the user ended up offline as long as the browser is still up. According to Sheppard [2], you must in some way show that the user is offline. This is important because if the user tries to reach a point that is not available offline the site will seem broken.

Archibald delineates two types of offline sites depending on where the computational load is concentrated: on the server side or the client side [4]. On sites that you “look stuff up” on is the work done mostly by the server and on the sites that you “do stuff on” is the work done mostly by the client. Because these two types of sites exist, and they handle data differently are they important to consider before implementing how the offline functionality should work.
2.2. **Web Security**

This section will explain some of the essentials about using a Service Worker in a secure context, and the fundamentals of relevant web-based attacks. It will also describe Service Worker attacks that have been fixed and current vulnerabilities.

2.2.1. **Essential Service Worker Security**

As described earlier, the Service Worker is a very powerful tool in the field of web applications. But Service Workers also creates security risks, among other things because all modern browsers install Service Worker without the client's permission [7]. This is an intended procedure as it is believed that the Service Worker would often go unused otherwise.

Another basic security standard for using a Service Worker is that the site is using HTTPS, as HTTP does not meet the requirements for using a Service Worker [15]. The browser knows how to rely on an HTTPS website by using a third-party certificate. This certificate is used as a confirmation that the web application can be trusted to ensure that the server uses secure standards. HTTPS encrypts the communication protocol using Transport Layer Security (TLS) or Secure Socket Layer (SSL) to securely tunnel the data being sent. HTTPS sends encrypted data between client and server to ensure that a session hijacking does not result in a data leak.

2.2.2. **Proxy/man in the middle**

A proxy is a middleman that exists within the browser or as a layer in between the browser and the server, that aims to listen on network traffic or to interrupt it. The normal usage of a proxy usually consists of filtering web applications, blocking content or to be used as a virtual private network. But with the invention of proxies, a new web attack called man in the middle (MITM) became possible. A MITM attack is performed by exploiting the fact that HTTPS needs to send the public key at the start of the communication, which makes it possible for a middleman to intercept it and thus making it unsecure [8]. In order to trick the HTTPS system, the middleman uses a fake certificate which is used for all communication with the victim and the trusted certificate which was originally sent from the server is used for communication with the server as illustrated in figure 6 [8]. This can be avoided if the browser sends a warning to the victim and that the victim does not just skip through the warning. Callegati et al. describe that there are different versions of MITM attacks based on how the network is structured [8]. It is possible to use the routers switch to reroute all traffic through the attacker’s computer if the victim is on the same network as the attacker. This can be done on a local area network if the router is freely available or if the network is breached and a new user can be added [8]. The other possibility is that the attacker is located on the internet and has no close connection to the original router. This type of MITM is performed by modifying the address resolution protocol (ARP) and Domain Name System (DNS) to redirect the traffic through the attacker’s computer. This can be done by ARP poisoning, which is used to sniff and retrieve traffic sent to a switched LAN. By doing this the attacker gains knowledge of the victims IP address and Mac address and can use them to mimic the victim [8].
The threat of a MITM attack is that the victim unknowingly gains a “big brother” monitoring all traffic from the victim, which is not only limited to only data leaks as the attacker also has other potential ways to restrict internet access. This can generally be done by sending the users browser modified responses to other sites or outright blocking the request, which effectively blocks the victim from using the internet. The attacker could also impersonate the victim to other users on the internet which could go completely unnoticed by any other user on the internet.

2.2.3. Cross-site scripting
Cross-site scripting (XSS) was the second most prominent web-based attack during 2015 according to OWASP [21]. An XSS attack is performed by maliciously modifying a web applications code with a script which can be inserted through a text field, a URL or in other clever ways. In figure 7 you can see Googles search field which sends an alert prompt with the number one inside.

![Google Search](source:google.se)

An XSS attack is client based which means that security systems like the firewall cannot filter them, because the attack is executed on a trusted web application. This means that it is possible to make a web-based attack by forwarding an XSS infected web page to its victim. The goal of the XSS attack has historically been to take over a user's account/session, download of malicious programs or keylogging [21]. However,
this is not the only use for XSS, as it can also be used to make social manipulation by for instance using popups and distorted versions of the web page.

There are generally three categories of XSS attacks which are as follows: Reflected XSS is a non-persistent attack performed by modifying the Uniform Resource Identifier (URI) or HTTP parameters with attack code and sending the link to the victim. Stored XSS is when a user of a web page sends a message to someone where the message consists of the malicious code. This type of attack will happen to everyone who opens the message because the attack code will be saved on the webpage/server and there will not be a one-time attack. DOM XSS modifies a webpage's environment, which means that the webpage can act in unexpected ways, for instance, acquiring and sending the session cookie when the user tries to change the language. This method can be used to bypass the server’s XSS filter as it can be sent as a comment in the URL by adding a hash (#) symbol. Unlike reflected XSS and Stored XSS, DOM XSS not only utilizes server errors but can also force already existing data from the browser by avoiding the server.

2.2.4. Social engineering
Social engineering is an attack which targets the vulnerability that every computer system has in common: its user. A social engineering type attack has a plethora of attack vectors, but they all aim to exploit the gullibility, intimidation and/or curiosity of the users. Mouton et al. describe a four-step model on how to perform an attack, the steps are as follows: research, developing report & trust, exploit trust, utilize information. [18]. Mouton et al. state that this model lacks detail but can be used as an overview of the attack cycle [18]. Social engineering is not limited to using this model, for example, the attack could be as simple as planting malicious USB stick in a parking lot or tricking people to click a URL that either leads to a bad web application or a trusted web application with an XSS.

The common goal of all social engineering attacks is to gain access to private data or to gain access to the other user's computer, but the attacks can differ in where to attack and how the new access/data is used. According to Algarni et al. the most common place to perform a social engineering attack is on a social networking site (SNSs) as it can be used effectively to launch an attack [19]. Algarni et al. state that there are different viable tactics depending on what type of user is being targeted, for example, an experienced internet user will not be as gullible as a new user [19]. Phishing is one of the most common types of social engineering, as it can be performed in a plethora of different ways [20]. A phishing attack aims to gain access to private information in some way (for example passwords and banking information), which is usually performed by mimicking a trusted web application or by sending emails [20].

2.2.5. Cache Attack
Data leakage through cache attacks is a big problem because the cache is essential for good performance in modern computers. One type of attack that is causing big problems are side-channel-attacks. Side-channel-attacks are based on securing information by using a system, rather than abusing a weakness. Side-channel-attacks are using the system in a way it is meant to, this makes it hard to prevent and defend against. Even if some work has already been done to try and show the root of the information leakage, as described by He et al. about a Probabilistic Information Flow Graph, side-channel-attacks are still a big threat [9]. Jia et al. describe a case study where they use geo-interference attacks to obtain a user's location [10]. A geo-interference attack sniffs out location sensitive information left by other sites which give away the location of a user. This is possible because the attacker uses the time difference between a cache hit and a cache miss. This difference in time is a desired feature in the cache as a hit should take less time than a miss since you do not have to go through the network.
Another danger with cache is if the wrong information is stored. This is one way a hacker can force a user to a malicious site, this is possible through a DNS poisoning attack. DNS poisoning or spoofing is an attack on a vital part of accessing the internet. By corrupting data in a DNS cache a hacker can redirect the user to whichever IP address they please. This can then escalate since different DNS servers communicate with each other and then save responses in their cache. HTTPS helps a little with verification on a DNS attack since the false website will not have a verified certificate, but this defense relies on the user noticing that something is wrong. A new extension called Domain Name System Security Extensions (DNSSEC) was created instead. DNSSEC sign DNS lookups with cryptographic keys to ensure that the data has not been tampered with during transport. This extension makes DNS much more secure but has been slow to be adopted.

2.2.6. Fixed Attacks

One of the most threatening attacks based on a Service Worker is the MarioNet attack, which is abusing a new API called Periodic Sync [17]. The Periodic Sync API was thought to be able to periodically wake up the Service Worker without any interaction from a server, as it was designed to be able to synchronize data between the server and web browser while the web application was closed. This could be useful to get new content without starting the application, as it would request updates on a set interval. The goal of this attack was to become a puppeteer that could use the victim’s web browsers as a botnet for tasks such as DDoS, crypto mining, hash cracking and as a storage for illegal data [17]. The MarioNet attack was mitigated by not launching Periodic Sync API. Another similar attack to get a Service Worker to exceed its lifetime is by continuously updating itself during the activation event (Issue 805496) [13]. This bug could also have the same consequences that the MarioNet attack had. The bug was fixed by adding a delay to the update event which eventually made the Service Worker die [13]. Historically there have been a plethora of ways to break free from Cross-Origin Request (CORS) as it could be done for example by using canvas, subresource integrity and WebVTT [13]. There have also been more possibilities of using XSS with a Service Worker for example, in Issue 853937 the Payment Handler API was used to install a Service Worker as it imports JavaScript files from a specific origin to be able to handle payment [13]. There has also been a case where a Service Worker was XSS-ed in on Amazon.com subdomain to steal other users accounts [23]. To achieve this Hassam used a stored XSS to upload a Service Worker to the Amazon subdomain [23].

2.2.7. Service Worker Vulnerabilities

Shared storage came up as a potential threat while looking at different computer and browser cache problems, for example [10,11,16,22] describes the solution as a tradeoff between performance and safety. Since Cache API saves different caches based on origin this vulnerability is not a problem when implementing a Service Worker.

Side-channel attacks as described before can lead to information leakage, these vulnerabilities have been concluded to not affect either positively or negatively by introducing a Service Worker. Another big vulnerability that Kravchenko describes is outdated versions of protocols [7]. There are some Service Worker bugs exploitable in older versions of Chromium as shown in [13]. This means that outdated protocols are still a big vulnerability in correlation with the implementation of a Service Worker.

When implementing a Service Worker there are some potential vulnerabilities created, these have been named the following: Service Worker origin name, lifetime of cached data, bad cache strategy, persistence, Service Worker position, living outside of scope, old in cache.

---

9 [https://www.w3.org/TR/clear-site-data/#clear-cache](https://www.w3.org/TR/clear-site-data/#clear-cache)
**Service Worker Origin Name (SWON):** When a Service Worker is registered in the browser it will appear in the Service worker list, this can be seen by going to the developer tool and click Service Workers. The domain for the Service Worker is also visible here and creates a vulnerability for example, if the domain is geo-location specific as it could be used in an attack as described by Jia et al [10]. These domain names can give information about visited sites in a new seldom cleared way.

**Lifetime of Cached Data (LCD):** One important feature in Service Worker is offline browsing. This feature is only possible because the developer can choose what they want in the Cache API, and when the stored data should be deleted. This creates vulnerabilities since there is no time to live (TTL) for the cached data, as the information can be left and leaked. This property is exploitable as Nishtha et al. states that a good replacement strategy increases the security as it removes sensitive data when it is not in use [11].

**Bad Cache Strategy (BCS):** OWASP states that disabling caching of responses that may contain sensitive data as one way to stop sensitive data exposure [21]. This can be circumvented by a Service Worker since developers have full control over what should be put in the Cache API, even sensitive data might be stored. This vulnerability is created because the Service Worker gives control to its developers.

**Persistence:** Since the Service Worker gets registered in the browser it may become a persistent threat until its unregistered, as shown by Hussam constantly leaking Amazons CSRF tokens as soon as the user enters the application while having a rogue Service Worker [23]. This will make it possible to achieve a persistent attack, but it will be limited to the web application since the Service Worker will be stopped after the user has left the web application.

**Living Outside of Scope (LOS):** The Service Workers specification states a specific lifetime, which should not be able to be broken. If this lifetime is increased, as shown in [13] Issue 805496 or in Papadopoulos et al. [17] the Service Worker could secretly work on a user's computer. This may still be a matter of concern, even if issue 805496 is fixed and Papadopoulos et al. used an API that has not been developed, these are examples of a Service Worker exceeded their lifetime. LOS is dependent on the vulnerability persistence, meaning that if LOS is fixed persistence may remain but not the other way around. The difference between LOS and persistence is that persistence means that it is only when the user is on the application, while LOS can continue to work indefinitely.

**Service Worker Position (SWP):** Service Workers control all out and incoming communication, this feature is possible by the position of the Service Worker and enables a lot of desirable features, but it also creates a good target for a hacker. Callegati et al. talks about the consequences when an attacker gets in between the source and destination and manipulates the traffic [8]. This threat is a consequence of using Service Worker, which could give devastating result if not fully controlled.

**Old in Cache (OC):** If the Service Worker is implemented to access data in Cache API first and if the requested data is found and directly send to the client, an outdated version could be sent to the client. This can also force an older version of the web application to be reused even if an updated version exists on the server. This could lead to problems if, for example, the older version has an exploitable bug that the newer version has fixed. This is in close relation to the vulnerability LCD, as it also relates to the problem described by Nisht et al. [11].
2.3. Related Works

Service Worker is a new technology that has increased in popularity in recent years, because of this there is a need to get information out into the general public. This is done mostly through beginner guides like Sheppard's book [2], or through blog posts [1,12,15]. Scientific papers about Service Workers are few, but work has been done, for example Kravchenko's article regarding security [7], or Papadopoulos et al. about a potential attack called MarioNet [17]. While scientific papers regarding Service Worker specific attacks are few, there is a large body of papers regarding web attacks, for example, Mouton et al., Algarni et al., Gupta et al., and Wilcox et al. about social engineering attacks [18,19,20,27]. Different surveys have also been made on attacks on other important components, like Nishtha et al. about cache security [11], or Yoon et al. about different attacks using HyperText Markup Language (HTML) 5 [6].
3. Method
A security matrix has been developed as described in the security matrix subsection. Some experiments will be conducted to study which Service Worker functionalities that can be abused if regular implementation guidelines are followed. These experiments are described in the experiment subsection.

3.1. Security Matrix
A security matrix inspired by OWASP and Kravchenko will be developed to present potential vulnerabilities \[7, 21\]. The presented data is supported by the experiments and a complementary literature study on web-based attacks to outline other possible breaches of security. These attacks include cross-site scripting (XSS), cache attacks and proxy related attacks, as they are assumed to be the most critical attacks that can potentially be performed by a Service Worker. A proposal on how to mitigate these sorts of attacks will be discussed based on the security matrix.

The inspiration for potential vulnerabilities was taken from Chromium's bug reporting site [13] and official or other detailed implementation description sites, for example, W3C and Mozilla [14-15]. The implementation sites are used to show potential vulnerabilities if a developer has slavishly followed its instructions. The study will not be limited to known examples but will include attacks derived from the authors’ ingenuity.

Potential vulnerabilities will be studied through the experiments or by using academic papers such as [11] and other credible sources like [13]. The discovered vulnerabilities will be studied and graded according to three criteria as shown in table 1. The criteria are the following: Consequence: how big of an impact the attack will have. Complexity: how hard the attack will be to mitigate after it has happened. Difficulty: how experienced the attacker needs to be for performing the attack.

The vulnerabilities will then be divided into which APIs they occur through (approach) and will always be graded based on the worst-case scenario for consequences and prioritized based on combined score. The different type of attacks and potential measures will then be discussed in the discussion section. An example can be seen in Table 2. A total score is also calculated to get an overview on which vulnerability is the worst. The total score is calculated by adding the level from each criterion.

3.2. Experiments
During this project, a selection of web-based attacks will be performed to test the Service Worker ability to be used for attacking. The limitations of these experiments will be the authors limited time to study complex attack method, the attacks focus on making the Service Worker act in a way the specification document describes to be impossible. The experiments that will be conducted are the following:

Experiment 1 - Worker in the middle
The first experiments aim to evaluate the possibility to use a Service Worker to perform a man in the middle attack. This will be tested by sending all responses to a separate destination but will also check if the Service Worker can send CORS to redirect the user to another webpage/web application.

The client used during experiment 1 is of little interest, except the Service Worker which will try to perform a MITM attack. The Service Worker only needed to implement the fetch event listener as it is responsible for interrupting network requests and responses. As a Service Worker has access to Fetch API a separate fetch was called inside of the event listener and respondWith() to send a new response to the web application.
To fully test how to avoid CORS different local and online servers where used. The local server used was an Atom live server, which is a plugin for Atom that can host a local web application. The online servers were used for two different tasks: the first to try to break CORS and the second to be the domain the Service Worker could fetch content from.

Experiment 2 - Bad eternity with XSS
The second experiments aim is to install a Service Worker using XSS and also to check if any new approaches of conducting an XSS are introduced by implementing an unsecure Service Worker.

This experiment only needs a basic web application containing text fields and a button to add new text to the page. The possible XSS attacks are not based on the server. The web application will also not contain any extra XSS protection as it will be purely for testing the possibility of adding a Service Worker to a web application and importing scripts. To fairly perform this, experiment no default settings on the web browser will be changed. If this attack succeeds the Service Worker will always stay on the web application so the XSS attack becomes permanent.

Experiment 3 - Pushing it to the limits
The third experiment will try to extend the specified natural lifetime of a Service Worker by using the Push API, both with and without sending notifications.

The client side used for experiment 3 consists of a web application that registers a Service Worker and can subscribe to web push notifications. The Service Worker will only need an event listener for handling incoming data from the server.

The server used for sending data was a node.js server using web push library\(^\text{10}\) and express\(^\text{11}\) to start the server.

Experiment 4 - Storage attack
The fourth experiments will test to which extent the Cache API and IndexedDB are available to the user and other webpages, for example, if Twitters Service Worker can access Gmail’s data.

This experiment will use two different GitHub pages\(^\text{12}\) to try to access each other's Cache API, the possibility to access a completely different web application’s Cache API by copying its name. The indexedDB test needs a more hands-on approach where its accessibility was checked through opening the database with the browsers developer tools.

3.3. Ethical considerations
As a security measure for conducting the experiments. The web applications used in the experiments include a warning message and instructions on how to unregister the Service Worker. The experiments which are only conductible by using a published web application (e.g. experiment 4) used GitHub pages as it can restrict the access to the web application and be turned off when the experiment is finished. Disregarding the measures taken if someone accesses the web application it will not do anything harmful to the victim because those experiments will not make any lasting changes as they only affect the web application. The other experiments will be conducted using a local server on the host machine. This

\(^\text{10}\) https://github.com/web-push-libs/web-push
\(^\text{11}\) https://expressjs.com/
approach will contain the malicious Service Worker to the local client, as it will not access other pages on the internet.

As for the search of Service Worker implementations on different web applications, the authors will impartially review the web application. The authors are only interested in the security aspects of Service Workers and not its creator.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Level</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consequences</td>
<td>1</td>
<td>The user experiences disturbing behavior from the web application, for example getting drowned in notifications</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The attack adversely affects the user experience of a web application to a degree where the application becomes unusable.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>The user’s web browser is exposed to a data leak.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>The hacker has gained full control of the user’s data traffic without affecting the user experience.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Level</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>1</td>
<td>The vulnerability can be quickly mitigated, and no extra resources (for example money, server downtime, and personnel) are spent.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The vulnerability takes some time to be mitigated, and no extra resources are spent.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>The vulnerability can quickly be mitigated, and extra resources are spent.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>The vulnerability takes a long time to be mitigated, and extra resources are spent.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Level</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>1</td>
<td>The attack is only considered possible by a well-informed and experienced web attacker, who has advanced knowledge of Service Workers and its APIs.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The attack is considered possible with good knowledge of the structure of web applications as well as underlying components.</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>The attack is considered possible with basic knowledge of the structure of web applications.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>The attack is considered possible by a beginner who can follow a tutorial and/or copy code.</td>
</tr>
</tbody>
</table>

Table 1. criteria for security matrix

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Approach</th>
<th>Consequences</th>
<th>Complexity</th>
<th>Difficulty</th>
<th>Total score</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>API 1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>API 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td>API 1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2. example of security matrix’s appearance.
4. Result
The result is divided into two subsections, the result from experiments will describe the result of our experiments and then our security matrix will be presented.

4.1. Results from experiments
This section will present the vulnerability and attack vectors found during the experiments. This section will also show the outcome of the experiments.

Experiment 1 - Worker in the middle
Experiment one had a positive result in that it was possible to break free from CORS (this vulnerability will be called Free From CORS (FFC)) in a Service Worker in a few different ways. They are the following:

- CORS enabled
  One workaround is that the server it tries to fetch from has the CORS enabled, which is also needed on the client to not get blocked by the web browser. This means that information could still be sent through a fetch request to another web application, if both the browser and server has enabled CORS.

- Proxy server
  A proxy server could be used to circumvent the CORS validation, as the authorization proxy modifies the response header and passes the CORS enabled response to the client. This will only be possible if the application has the same domain as the proxy as it would otherwise be rejected by the browser.

- Cache API
  It is also possible to avoid CORS by using responses stored in the Cache API from another origin, which will be further explained in experiment 4. It was also possible to avoid CORS if the web application is on localhost, which could be a problem for web application locally hosted inside an organization.

The second part of the experiment (responding with another response) was also possible by using .respondWith(), as its return value is the response that gets sent to the application. This makes it possible to send any desired data to the application, but its use will be limited to the content available.

Experiment 2 - Bad eternity with XSS
The result of experiment two is that it was possible to register a Service Worker by using XSS in the console and in text fields, but this assumes that the web application has a Service Worker in its origin. This other Service Worker could be a completely different if the attacker has managed to get the Service Worker file on the origin. The Service Worker could also be edited by using a plugin, as these plugins can modify any part of a web application.

The new types of XSS that a Service Worker introduced is that the attack became persistent and that it can be used to import scripts independent of the web application.

Experiment 3 - Pushing it to the limits
It was possible to extend the lifespan of a Service Worker by using the Push API, but its success depends on notifications being allowed. This means the user needs to allow notifications from the web application (see figure 8). These Push messages could even be sent silently in the background. The messages sent silently from the Service Worker can only be seen using console, as illustrated in figure 9.
Google Chrome asks for permission to allow notifications, **Source:** Google Chrome

![Figure 8](image)

Silent notifications as seen in developer tools in the console, **Source:** Developer tool Google Chrome

The Service Worker does not need to be running to respond to events as the web browser wakes it up when a push is received from the server. This means an attacker can keep sending silent push notifications to keep a Service Worker alive until the attack is finished. To prevent these kinds of abuses Google Chrome has made sure the user will get a notice that the application is running in the background (see figure 10). Mozilla Firefox has not implemented any preventions against these kinds of attacks.

![Figure 10](image)

**Figure 10. Working in the background notice from Google Chromes Notification API,**

**Source:** Notifications on Windows 10

Experiment 4 - Storage attack

The Cache API could not, in general, be accessed from another origin even if they shared a name, except from a special case where the domain is the same (see figure 11). The Cache APIs in the special case where merged if they shared name, making it possible to change the other web application’s Cache API. This vulnerability will be called Cache Swap.
Another finding was that Google Chrome's developer tool gives access to a preview of both Cache API and IndexedDB, this preview is not available on Mozilla Firefox. The information could be any response that the developer has decided to store, for example an admin view of the application or messages.
4.2. Security matrix

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Approach</th>
<th>Consequences</th>
<th>Complexity</th>
<th>Difficulty</th>
<th>Total score</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWON</td>
<td>Service Worker</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>LCD</td>
<td>Cache API</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>BCS</td>
<td>Cache API</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Persistence</td>
<td>Push API</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Fetch API</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Notification API</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOS</td>
<td>Push API</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>SWP</td>
<td>Service Worker</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>OC</td>
<td>Cache API</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Fetch API</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache Swap</td>
<td>Cache API</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>FFC</td>
<td>Service Worker</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 3. Complete security matrix.
5. Discussion

The discussion section will evaluate the method, the security matrix, possible mitigations, and future work.

5.1. Method

The problematics of evaluating a Service Worker stems from the shortages of scientific research about it, as it is a rather young field of research. Thus, developers commonly use material and specification documents that are not necessarily reliable. Another point of interest is that the Service Worker is still under development, making it more susceptible to bugs as new APIs and other permissions are added. These bugs will probably be added to a bug reporting site, which will not be shown until a grace period has passed. This will be problematic as there probably are more vulnerabilities in a Service Worker that could not be studied and included in the security matrix.

The experiments could not test the full potential of some attacks that exploit the Service Worker, for example using a Service Worker as a botnet. This is because of the ethical considerations of using the victim's browser and that our experiments are limited to small web applications.

Additional levels could be added to increase the accuracy of the security matrix, as they currently are quite vast. Another way to increase the accuracy of the security matrix would be to separate more commonly occurring cases and worst-case scenarios, as currently only the worst-case is considered. This change could give a clearer view of the vulnerabilities.

5.2. Security Matrix

This section will discuss the security matrix in order to clarify the score. It will also present which countermeasures that are already in place as well as other possible mitigations.

5.2.1. Service Worker Origin Name - SWON

The worst case in SWON is that information about visited sites can be used in a social engineering type attack, for example the applications that the victim used would not be cleared by deleting the browser history as the Service Workers are not unregistered. Even if the data is of limited use it is still a data leakage that can be used to gain more critical data. This vulnerability can be mitigated by the web browser by only showing the running Service Workers or if the user systematically unregisters their Service Workers. To gain this information either a shoulder surfer or somebody with access to the computer needs to open an interface for checking Service Workers.

5.2.2. Lifetime of Cached Data - LCD

This vulnerability is strongly connected with OC but is more connected to data leakage than bad user experience. By implementing your own time to live for your cached data you mitigate the vulnerability but might lose some performance. Long lifetime of cached data is a planned feature of the Service Worker as the implementors wanted the developer to be in full control of the Cache API. The worst case for this vulnerability arises when critical data (for example a logged in section of the web application) is stored for a long period of time, as it increases the risk for the data to be leaked.

5.2.3. Bad Caching Strategies - BCS

This vulnerability is also created by giving developers full control over what should be in the Cache API. The vulnerability cannot be fixed, it instead needs more information about basic good praxis. This can be given by tutorial sites that plan on giving the information to new developers that try out the tool for the first
time. Some current tutorials lack this information. One possible mitigation for BCS is that the developer should implement a blacklist of URLs to effectively filter the content, which is put into the cache.

5.2.4. Persistence
The persistence of a Service Worker is a vital part of its concept but can be used with other types of attacks. The only difference compared to a regular XSS attack is that by performing an attack with a Service Worker gives the ability to gain persistence as the attack only needs to be successful once. This is because the Service Worker is registered to the web application and thus will always be running while the web application is being used. The worst case would be if the attacker gets a malicious Service Worker on a trusted site with many concurrent users. But this needs good knowledge from the attacker to be able to evade security measures. To mitigate persistent attacks could web browsers have a confirmation to register as well as use the Service Worker. This would be bad user experience and probably something not desirable.

5.2.5. Living Outside of Scope - LOS
LO$S$ has the highest total score of all the vulnerabilities, this is because it gives the attacker the ability to wake up the Service Worker at any time. The LOS vulnerability is only abusable if the user gives permission for notifications. This is a good countermeasure but since the browser saves the subscription only one confirmation is enough. This vulnerability is hard to detect since invisible notifications are possible to be sent both in Google Chrome and Mozilla Firefox. The worst-case scenario for LOS is the same as while using MarioNet, as they achieve the same state. The problem with mitigating LOS is that the vulnerability can be done by abusing the Push API as shown in experiment 3. The reason for why this is hard to mitigate is because the server might send data at any time for example if the user is following many people on Twitter, then there is a possibility to get a continuous stream of notifications on status updates are sent. One mitigation which could be implemented is to force the Push API to send a notification even if the Notification API is unused.

5.2.6. Service Worker Position - SWP
Since the Service Worker controls all traffic from and to the web application is this vulnerability high up in total score. The worst case for this vulnerability is a big safety concern, but this vulnerability is also the reason the Service Worker has the features desirable. Service Worker developers have done a lot of regulations to try and secure the Service Worker but the vulnerability to get a running MITM is still a possibility. The only reasonable way to mitigate SWP is to not use Service Workers because if this part is changed it will not be a Service Worker but rather a Web Worker.

5.2.7. Old in Cache - OC
This vulnerability is mostly related to a bad implementation of the Service Worker. This vulnerability can lead to bad user experience but can also cause some big security threats. If malicious data or Service Worker is stored locally the user can experience an attack even if an update has been made. The vulnerability can be mitigated by using the clear-site-data header\(^\text{13}\) as it forces deletion on all stored data related to the web application. The vulnerability can also be mitigated by implementing a TTL for regulating the lifetime of cached items.

5.2.8. Cache Swap
This vulnerability is probably caused by a special case (two GitHub pages on one account) in our experiment. We have tried to recreate the result in different ways but to no success. The only time the result could be recreated was by recreating experiment 4 on another GitHub account. We concluded that this is

\(^{13}\) [https://www.w3.org/TR/clear-site-data/](https://www.w3.org/TR/clear-site-data/)
probably a special case since CORS got blocked but the Cache API thought them as the same origin, but if it is possible to recreate the cache swap, could the security bug be used for devastating results. But security measures are already in place to try and mitigate other sources from changing the Cache API content. For example, the HTML standard for same origin is used. The cache swap can be mitigated by enforcing a stricter origin policy.

5.2.9. Free From CORS - FFC

This vulnerability could lead to a big consequence, a lot of countermeasures are already in effect to avoid this vulnerability. One of the countermeasures for FFC is that both the browser and the server needs to allow CORS except when using localhost. Localhost is allowed to be a special case since it cannot be accessed on the web. Since the control happens by more than one actor a total score low. To mitigate any further tries to break free from CORS the developer can include a filter on both the web application and the server. The filter should look for the CORS header in the request and response.

5.3. Attack mitigations

The mitigations that are to be discussed will be in a larger scope as most of the vulnerability can be used in more than one type of attack. These mitigations will be focused on how to hinder the installation and use of a rouge Service Worker or the attacks themselves.

As described by Yusof et al. a pattern content filtering method could be used to mitigate XSS type attacks using/adding an unwanted Service Worker [24]. This sort of filter would need to be added on every input field or as proposed by Kaur et al. and Wang et al. a Google Chrome extension could be used to do the filtering as it can check for scripts during runtime [25-26].

The single feasible approach to mitigate a social engineering attack is to educate the people using the internet on how to avoid them. Wilcox et al. propose that a possible solution would be to give information on IT security standards and what threats that the attacks can pose [27]. Another way to educate the users would be to follow a flowchart as stated by Mouton et al. or a state machine as proposed by Jamil et al. [28-29].

The traditional approach using an HTTPS for preventing a MITM attack will not be applicable to a Service Worker MITM. This is because the Service Worker is already residing in the web browser, which means that the content will be decrypted before it reaches the Service Worker. But it can be mitigated by monitoring data traffic or if the web browser decrypts the data after it has passed the Service Worker.

5.4. Future Work

Some vulnerabilities that have been discussed in this paper are specific to the Cache API. Could strategies for securing the browser and CPU cache also be used for Cache API?

One conclusion that could be drawn during this study was that even though all of them are caches the fundamental uses are too different. The Cache API is used for the specific reason that a particular application wants its data saved while browser and CPU cache is used more generally to decrease the cost for a lookup.

Some strategies could also be applicable. For example, the request can first go either to the database or to the cache. There are also some choices involved in deciding when and how to update the cache. Further research is required to investigate different safety aspects of the implementations.

Background Sync API is a useful API if some specific features are desired. When looking at Service Worker Security this API have been looked at, but a more thorough research on the impact of this API could be
made. This is because Background Sync API is used in a way that IndexedDB and Cache API save important data and as described before this data can be accessed easily through the browser.

Another point of interest when looking at security related to Service Workers is the consequences of using the same computer. A world were computers and mobiles are more and more available can this kind of subject be neglected or forgotten. A more detailed researched has to be made around Service Workers influence on these types of computer in order to get a full picture of potential threats that can be developed.
6. Conclusion

Overall the Service Worker is secure, but it can be abused by an attacker if they have the knowledge. The cache security is fully dependent on the implementation of the web browser and the web developer who coded the Service Worker. For this reason, a developer needs a clear understanding of how to implement a secure cache. The most prevalent threat which can arise is if a malicious Service Worker is registered on a popular web application that could exploit LOS to abuse the user secretly.

The four experiments show that it was possible to respond with a different response than the one that was requested. It was also discovered that there is a possible workaround on the CORS policy, which could be done by using a server proxy. The possibility to register a Service Worker using XSS if it existed on the same origin as the application can be seen in the second experiment. It was possible to extend a Service Workers lifetime to exceed the limited time of an application by using the Push API. The last experiment was successful in changing the stored data in the Cache API between two different applications, though this might have been a special case created by the setup used in the experiment. It can be concluded from these points that some of the vulnerabilities related to the Service Worker can be easy to exploit but may need some additional steps to be harmful.

To conclude, the technology has been worked with and evolved far enough that a secure web application with the use of the Service Worker is possible, but there are still some new vulnerabilities that can become a problem if not considered.
Reference