

SECURITY  
MADEIN.LU



circl.lu



cases.lu



c3.lu



Security Made In.Lu

CIRCL, Luxembourg, *Alexandre Dulaunoy*,

## *MISP Expansion*

### A Browser Extension For MISP

*Anselme Morisot*

Master 2 computer science

Information Systems Security

*Internship report from April 01 to September 30, 2021*

Université de Lorraine

MIM

*Francine Herrmann*



UNIVERSITÉ  
DE LORRAINE



UNIVERSITÉ  
DE LORRAINE

Anselme Morisot

# Plan

## **I) Introduction**

Special thanks	3
Introduction	3
The company	4

## **II) A word about :**

MISP	6
PGP	7
JavaScript	8
Browser Extensions	9
Browser Extensions Security	12

## **III) Stages of the project**

Thinking	14
Learning	16
Development	18
Security	21

## **IV) Conclusion**

<i>Bibliography</i>	25
---------------------	----

<i>Glossary</i>	26
-----------------	----

<i>Table of Appendices</i>	28
----------------------------	----

## Special Thanks

Before any development on this professional experience, it seems obvious to me to thank those who accompanied me during my internship.

Also, I would like to thank Alexandre Dulaunoy, my internship supervisor, for taking the time and having the patience to guide me but also for offering me the opportunity to complete this internship, as well as for all the means made available to ensure that the internship takes place in the best conditions.

I also thank the colleagues for their welcome even if the contacts were very limited due to the current health circumstances.

Finally, I would like to thank David Cruciani, second intern who will have accompanied me throughout this internship which was of great help at times.

## Introduction

As part of my Master 2 in computer science, specializing in Systems Security, I had to complete a six-month internship at the end of my apprenticeship.

So, from April 01 to September 31, I did an internship at [CIRCL](#), a department of *SecurityMadeIn.Lu*, and more specifically on the [MISP](#) platform.

In view of the current health circumstances, this internship was different from the internships I was able to do previously. In fact, this internship took place almost exclusively remotely. While some days were spent at the workplace, this only happened in the second half of the internship. In addition, the whole team was not present at the scene, so I was only able to meet some of the colleagues in person.

We had a group available to us on [Element](#) so we could communicate with other team members. We also had a [bbb](#) group for the video conferences in addition to the usual channels (messaging, etc ...).

For a cross-border worker, the advantages of being at home are not having the travel time to get to the site and not having to pay transport costs.

The disadvantages are having to pay for electricity and being alone.

As a result, two big major difficulties can emerge from this situation: the lack of motivation and the lack of mutual assistance, hence the fact of being vocal with David during working hours.

Regarding the project, I had the chance to work solo and start the project, so no upstream code to understand and no specific directive. In addition, most of the information required for the realization of this project was on the internet, so all I had to do was find it, understand it, but above all verify and test it. Because it is dangerous to blindly trust third-party code even if everyone knows that everything on the internet is true.

The initial topic was to compare the [PGP](#) encryption of web pages to see if they are trusted and eventually evolved into the comparison of a web page or its elements with the content of [MISP](#). This comparison is made using a browser extension for Chrome and Firefox.

Due to my specialization as well as the nature of the covered subject, it is obvious that the security of the extension had to be treated with special care. Note that, due to my lack of knowledge in web extension as well as the fact that JavaScript is not among my mastered languages, the majority of my working time is focused on the development part of the extension.

## The Company

The Computer Incident Response Center Luxembourg ([CIRCL](#)) is a government-driven initiative designed to provide a systematic response facility to computer security threats and incidents with a 14 members team.

[CIRCL](#) is the [CERT](#) (Computer Emergency Response Team) for the private sector, communes and non-governmental entities in Luxembourg.

[CIRCL](#) provides a reliable and trusted point of contact for any users, companies and organizations based in Luxembourg, for the handling of attacks and incidents. Its team of experts acts like a fire brigade, with the ability to react promptly and efficiently whenever threats are suspected, detected or incidents occur.

[CIRCL](#)'s aim is to gather, review, report and respond to cyber threats in a systematic and prompt manner.

*CIRCL* is operated by *SMILE* (Security Made In Lëtzebuerg), which is also the host organization for *CASES* and the Cyber Competence Center (C3) of Luxembourg.

*CIRCL*'s missions are to :

- Provide a systematic response facility to *ICT*-incidents
- Coordinate communication among national and international incident response teams during security emergencies and to help prevent future incidents
- Support *ICT* users in Luxembourg to recover quickly and efficiently from security incidents
- Minimize *ICT* incident-based losses, theft of information and disruption of services at a national level
- Gather information related to incident handling and security threats to better prepare future incidents management and provide optimized protection for systems and data
- Provide a security related alert and warning system for *ICT* users in Luxembourg
- Foster knowledge and awareness exchange in *ICT* security

*CIRCL* is the *CERT* for the private sector, communes and non-governmental entities in Luxembourg.

*CIRCL* is an accredited *CERT* of TF-CSIRT Trusted Introducer.

*CIRCL* is also a full member of *FIRST*.

*CIRCL* also provides a large range of services on a national scale :

- Incident Coordination and Incident Handling
  - Reporting of security incidents
  - Incident identification, triage, analysis and response
- Technical investigation:
- Incident correlation
- Malware analysis and reverse engineering
- System and network forensic analysis
- Security vulnerability assessment
- Information leak analysis and data mining
- Dynamic malware analysis platform
- *MISP* for private sector
- International and national CERT/CSIRT cooperation and also with Local Incident Response Teams (LIRT)
- Incident coordination might also include vulnerability handling and responsible vulnerability disclosure on incident reporter's request
- Training And Technical Courses (PDF Catalogue)

- Incident Handling Support Tools and Services
- URL Abuse to check and review security of URLs
- cve-search Common Vulnerabilities and Exposures (CVE) web interface and API
- IP address to ASN mapping whois service including 4 years of historical data
- Passive DNS, historical DNS records database (access on request, contact us)
- Passive SSL services, historical database of SSL certificate per IP address (access on request, contact us)
- Dynamic malware analysis platform (access on request, contact us)
- Threat indicators sharing platform for private sector - MISP (access on request, contact us)
- Network services exposure change detection
- Data Feeds and Early Detection Network
- Private and public organizations in Luxembourg can benefit from our early detection network by hosting a sensor in their unused network spaces
- [CIRCL](#) provides a contextual feed containing all software vulnerabilities including visibility ranking in Luxembourg

[CIRCL](#) is also working with private and public organizations in order to foster research in the security field. Don't hesitate to contact us, if you would like to discuss a research partnership.

[CIRCL](#) is actively working on different projects where contributors are welcome to participate and reuse software in their own organizations.

## A word about

This section is a grouping of notes, remarks, or even difficulties concerning the subjects dealt with and the themes tackled during this internship.

### MISP

**MISP** is a CIRCL project that is a threat intelligence platform to share, store and correlate targeted attack IOCs, threat intelligence, financial fraud information and vulnerability information.

The initial goal of **MISP** is to store information in a structured way to allow simple use of this data or even automated with an **IDS** or **SIEM**, as well as the sharing of this information. This sharing makes it possible to receive from trusted partners (other similar organizations are targeted by the same threat actor) information on a targeted threat thus allowing collaborative analysis and therefore saving time.

The features of **MISP** are:

- An efficient **IOC** database to store both technical and non-technical information on malware samples, incidents, attackers and intelligence as well as automatic correlation finding relationships between attributes and metrics of malware, attack campaigns or analysis.
  - Built-in sharing functionality to facilitate data sharing using different distribution models between different MISPs. Advanced filtering features can be used to meet the sharing policy of each organization.
  - An intuitive user interface allowing end users to create, update and collaborate on events and attributes / indicators and a graphical interface to seamlessly navigate between events and their correlations.
  - data storage in a structured format (allowing automated use of the database for various purposes) and exportable: IDS generation (Suricata, Snort and Bro are supported by default), OpenIOC, plain text, MISP XML or JSON output to integrate with other systems (network IDS, host IDS, custom tools).
  - Data sharing: automatic exchange and synchronization with other parties and trusted groups using **MISP** which allows a simple pseudo-anonymous mechanism to delegate the publication of events / indicators to another organization.
- Flexible API to integrate **MISP** with your own solutions. (PyMISP which is a Python library).
- Integrated encryption and signing of notifications via **PGP** according to user preferences.

The MISP Expansion project is based on collecting and displaying events.

**PGP**

**PGP** is cryptographic encryption software that allows you to check whether a message actually comes from the origin (via cryptographic signatures), as well as to encrypt messages so that a single recipient can read them. To do this, users create a pair of asymmetric encryption keys (one public, the other private).

The public key / private key pair can be provided by RSA or DSA (encryption algorithms).

**PGP** offers authentication, privacy, compression, and segmentation services, while being compatible with many email systems.

**PGP** was developed by Philip Zimmermann (president of the company PGP Inc) in 1991 then subsequently openPGP.

The difference between these two software is their right, **PGP** belongs to **PGP** Inc while openPGP is open-source. Note that there is a version called GnuPG or GPG, which is the GNU implementation of the openPGP standard.

In the context of the internship, OPENPGP is used for exchanges (emails) between users and automatic messaging.

## JavaScript

As browser extension is equal to web programming, the main language is javascript (along with HTML and CSS). in addition to understanding the Web API, i had some trouble to go along with the **sequential programming**. For exemple, if you go with return function inside an asynchronous one, you will have the 'undefined' error !

If there is a simple return function there is no problem as long as it is used in a synchronous function otherwise there is this error. Asynchronous function with JavaScript can be identified with the key word « asynch » in their definition.

However, all of the web extension API functions are asynchronous.

Here an example with the storage API

```
var a;  
a = myBrowser.storage.sync.get(OurObject, function() {});  
console.log('OurObject = ' + a);
```

we will see in the console after execution 'OurObject = undefined' instead of OurObject

Here the correct one to see OurObject value

```
myBrowser.storage.sync.get(OurObject, function(RES) {  
    console.log(RES.OurObject);  
});
```



So you can work inside the get function without trouble but when you need to retrieve the value to work with it, that's another story.

The way I used to deal with the asynchronous function (only when required) is the Promise object.

A promise is an object that represents the completion or failure of an asynchronous operation that is returned and to which callbacks are attached rather than passing callbacks to a function.

A promise allows the use of the function `Then()` which guarantees the upstream processing of information and which will not be undefined. A pass and fail system ensures that any errors are caught. Note that the promise does not transform an asynchronous function into a synchronous one but allows micro-scheduling of tasks inside a function. However, there are some functions that do not handle the promises such as the function `setTimeout` which necessarily takes a callback function. However, if a promise can be affiliated with callback functions thanks to `Then()`, have a promise and a callback at the same time because the error handling will no longer be valid (the promise will return the error but it will not be caught by the callback).

So I learned that promise object is the best way to work with an asynchronous function as if they were synchronous.

## **Browser Extension**

For the user, a browser extension is an add-on that can be found and added via the browser store and that allow to do some stuff on the web. For the developer, it is a folder containing code for the browser and added with some carabistouille !

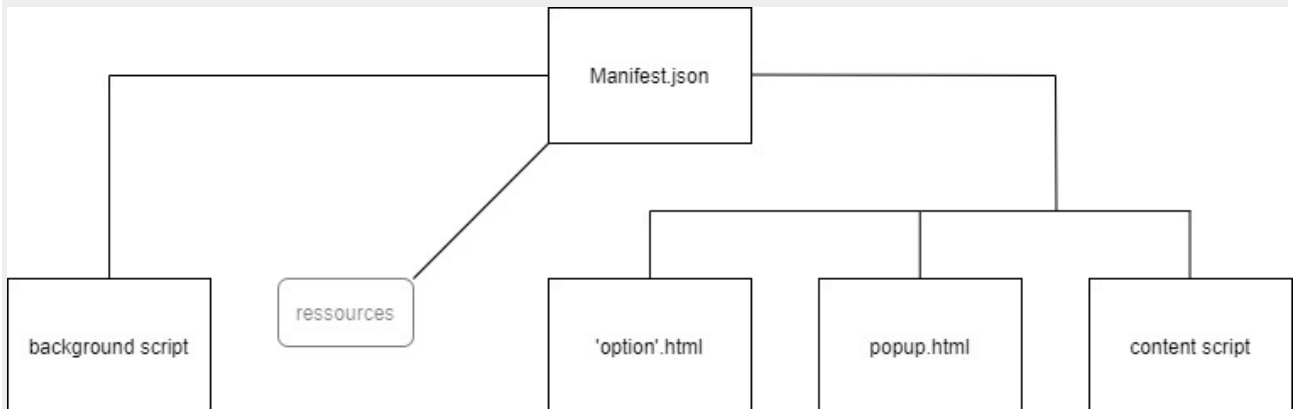
A browser extension is always made up of a base common to all browsers. Only APIs change from browser to browser (when it's not just the name).

Until 2020 November, extensions used the syntax of version 2 of the manifest (now using version 3). The version changes the keywords accepted in the manifest and therefore by extension the resulting possibilities for action.

So an extension consists of a manifest and up to 4 other parts (An extension with only a manifest is functional but will be unusable):

- Manifest (required)
- Background Script
- Content Scripts

- Browser Action (or PageAction)
- Options



The manifest is a json document which should be named « manifest », this is the entry point for all extension parameters / documents. It is mandatory to find there: the name of the extension, its version, the version of the manifest and 'browser\_specific\_settings' in the case of Firefox which generates a [uuid](#) (indeed, compared to chrome which keeps the [unpacked extensions](#), firefox removes extensions that do not come from the Firefox store each time you close the browser. The [uuid](#) therefore allows you to link parameters such as *options* or *storage*, which are not automatically deleted, with the same extension when it is again installed).

It is recommended to add the following elements: the description of the extension, the icons used to represent the extension in the browser and 'browser\_action' renamed to 'action' with version 3 of the manifest, which defines the behavior of the shortcut of the extension in the navigation bar of the browser (icon, management of the popup, deactivation/activation of the extension on a page).

There are other keys such as permissions, scripts used, resources etc ...

At this point, the manifest can be used as an extension although it can't do anything yet.

To be used the extension must be added manually as a development folder (the extension is called temporary extension) where the files are accessible (When modifying these files it is necessary to update the extension so that the files are changes are taken into account) or as an [packed extension](#) either by the user or by the browser store. On Chrome, if the extension is not added through the Chrome store then it will be present but inactive for security reasons.

To add an extension in a store, you must submit the files (after creating an account) to a verification imposed by the owner's browser. If the verification is validated, the extension is then available in the store and therefore can be loaded from it.

For other additions, in Chrome: open the browser settings then go to the "more tools" section then "extension" and finally activate developer mode.

It is possible from here to drag a [packed extension](#) (crx file), to compact an extension (to transcribe the file of the extension into a crx file) or to add an [unpacked extension](#) (select the extension folder).

Under Firefox, to load a [packed extension](#) you must load the xpi file. For an [unpacked extension](#), go to the following address : *about:debugging#/runtime/this-firefox* then select the extension folder in "load a temporary add-on".

Once the extension is loaded in the browser, it is possible to use it as long as it does something. This is where the other parts of an extension come in.

here is a non-exhaustive list of manifest key word :

« name » « version » « description » are info of the extension.

« manifest\_version » is the version of the manifest used in the extension. All APIs, security systems, features aren't available or are used differently depending on the manifest.

« icons » are all the icons used for the extensions.

« option\_ui » define option page properties.

« background » and « content » define the background script and content script pages.

« browser\_action » define extension behavior when user click on the icon shortcut.

« permission » define the authorization of the communication or the use of API and external url.

Browser Action / PageAction consists of the definition of the extension in the navigation bar of the browser. This allows you to define the icon and what happens when the user interacts with this icon such as opening a tab (new page or the extension's popup) or even causes changes to a page.

- Browser Action allows any type of interaction (defined by the developer) and on all web pages. If the extension is general, it is recommended to use Browser Action.

- Page Action is the reverse of Browser Action, it prohibits the global use of the extension and only authorizes it on certain pages (which the developer must define). The extension icon will be gray if the user is not on one of the authorized pages and no interaction will be possible. It is recommended to use Page Action when the extension should only be functional on a particular domain.

The popup or derived usage of Browser / Page Action is, in the extension, considered and exists only in the same context than a Content Script. With the exception that it only exists when the user clicks (in the case of a script) or when the focus is lost (in the case of opening a popup). It is necessary to specify that if the click causes the opening of another page / tab, the script only exists during the click but the page / tab remains open like a standard page.

Options is the part of the extension that allows the user to manage various settings according to the needs of the extension.

The pages (and therefore the scripts) of the options part are considered in the same context as a Content Script and therefore only exist as long as the pages are open.

Content Scripts are specific scripts that must be authorized for the urls on which they are used. Their particularities are to be executed in a context apart from the browser and the web page.

This therefore allows secure use of the scripts. However, the actions of Content Scripts are limited in order to guarantee this security. To begin with, a Content Script type script has access and therefore can modify the DOM of the body of a web page (hence prohibiting the use of [inline scripts](#)). It has no access with the browser itself and therefore with the Background Script except for sending and receiving messages with the *Message API* or Listeners with the *connect API*. (In terms of security, it is therefore up to the developer to pay attention to the messages that can be returned.) In addition, the APIs for retrieving information about the extension (such as *getUrl* or *Id*) which form the *Runtime API* which is one of the 3 APIs authorized in Content Scripts, the other two being *I18n* and *Storage*. Items from a Content Script only exist in the referenced pages and only last as long as those pages are open.

Note that Content Script (and runtime API) can't access to browser pages.

The Background Script is the browser-based script itself. It does not have direct access to the browser's web pages. One or more scripts can be loaded at the same time, it is however preferable, if the scripts are linked (import / export) to load them from an html page which takes into account the type = 'module' and to indicate the html page rather than the script array in the manifest.

Background scripts are the place to write code that needs to maintain long-term state, or perform long-term operations, regardless of the lifespan of any web pages or browser windows.

Background scripts are loaded as soon as the extension is loaded and remain loaded until the extension is deactivated or uninstalled or the browser is closed (all processes associated with the browser are terminated).

All web APIs can be used in these scripts, however, the associated permissions must be inserted in the manifest.json.

The Background Scripts being internal to the browser, the security vulnerabilities are those of the browser (in the case of Chrome or Firefox, regular updates guarantee the reliability and security of these browsers) and therefore the Background Script is only vulnerable by the browser itself on the condition that the developer does not do anyone in his code and that he respects the security rules of the extensions development.

## **Browser Extension Security**

If it is obvious that to secure an application the programmer must be particularly vigilant on his code as well as on the possibilities granted to the end user, there is, in the context of Web development, the Content Security Policy (CSP) as well as a guide of good practices in the protection of extensions which will guide the programmer throughout the realization of his project.

The CSP is a set of authorizations that guarantee security on the sites visited by being implemented directly on the servers but also on a case-by-case basis on the html (in the header of the file). It was extended to web extensions, where some properties being common such as resource externals (which are indicated in the manifest) added to the definition of the CSP rule.

The web CSP and the extension CSP are the same except that for an extension there is a rule applied by default prohibiting the following 3 actions:

- Inline scripting in Chrome App pages are forbidden. The restriction bans both `<script>` blocks and event handlers (`<button onclick="...">`).

This action forces the user to indicate a source file in the opening script tag of an html page while leaving this section blank. Likewise, a button cannot contain a script (this must be placed in the source script file).

- Use of external resources reference in any of the app files (except for video and audio resources) is forbidden. The same goes for an iframe.

This action forces the user to keep the resources in the extension folder (locally). If externals resources are required, developer must grab them using via XMLHttpRequest or Fetch and

serve them via blob or data (Video and audio can be loaded from remote services because they have good fallback behavior).

- Use of string-to-JavaScript methods like `eval()` and `new Function()` is forbidden.

Note that these rules can be override by adding a new rule authorizing these 3 actions.

CSP has rule for the ressources, for script, or the fetching urls. All of this rules are assigned with a right among 'self', 'none', '\*' and a list of url (can be empty) wich grant the proper autorisation.

Self only allow from the current file, none allow nothing, \* allow every sources and the urls list allow the urls page where extensions can proceed.

If one element is missing in the rule, it will use the «default-src » rule if present.

There is also some CSP default rules that into the manifest permission. They can be modified with the "content\_security\_policy" key

Note that if you use CSP in Html page, it will be overridden by the CSP in the manifest.

The Guide to Extending Good Security Practices is about tips and rules to secure a web extension.

The rules are :

- Ensure that third-party libraries are up to date, for obvious reasons
- Do not modify third-party libraries, because it is a significant indicator that a developer is trying to hide malicious code within code that is generally known and trusted.
- Create UI with extension components. It means that the developper must use extension browser APIs like context menu or browser\_action (in the worst case, use of iframes) but he must not use html elements.
- Use the standard extension content security policy, as saw before.
- Do not inject or incorporate remote scripts, because the code can be changed without user knowledge or consent, but add a copy of the script into the extension's code.
- Insert remote content safely, by using native DOM methods or JQuery's methods, try to avoid innerHtml methods.

And some tips :

- Share objects with in-page JavaScript with care, Firefox provides wrappedJSObject so a content script can access JavaScript objects created by page scripts. The danger here is that a malicious web page could, for example, modify the functions of JavaScript objects to run code of its own.

For more information, see Accessing page script objects from content scripts.

- Use `window.eval()` in content scripts with caution,same security breach as string,eval() ;

There is permission to allow extension to do things like urls where it can be executed, urls where Fetch API can be called or what API the extension can use.

Permission for API are called permission api et those for urls are called host permission.

With manifest version 2, extensions automatically receive host permissions for their own origin ('default: self').

## The stages of the project

This part is about the project itself and all the step from planning to completion

### Thinking

The goal of the [MISP](#) Expansion project is to increase the ergonomics of [MISP](#). By default, the user must, each time he wishes to make a request, log in to his [MISP](#) instance, manually enter the wide range of options as well as the various fields necessary to finally access the result. While with [MISP](#) Expansion, the user must fill in fields in the option menu then a right click for each search will be sufficient.

Starting from this postulate, the extension must therefore be able to :

- store data
- Allow the user to enter datas
- recover page's data to analyze with a simple click
- query [MISP](#) (and therefore connection)
- return the result

Each of these actions is therefore a design step involving more or less significant security risks, such as data storage which must be "protected".

To start, an already existing extension called Mitaka carried out 3 of these actions:

- retrieve data in one click
- query a site
- display of the result



However, the search part is simply to open the site on which the search is carried out and to pre-fill the input field with the value to be searched for (such as for example <https://hunter.io/email-verifier>) and the display being that of the site in question.

Only the data recovery part was therefore interesting here.

But in that case, why not just add **MISP** to the list of proposed sites ?

If the question and the idea are more than relevant there are several reasons why this is not so good.

- First of all, it is clearly indicated in the best practices guide that external dependencies should be avoided as much as possible and although the extension code is available to everyone on github, the extension itself would not belong to **CIRCI** and for example (this is not currently the case) nothing would prevent the analysis, collection and even the sale of the data entered by users by Mitaka. In addition, Mitaka should be updated regularly.
- This extension would require too big changes just to work with **MISP**, changes that the original designer may not approve of.
- Even though Mitaka's code is available on github, the plugin uses a lot of external dependencies that are not available. These external dependencies must also be updated if necessary (and there is no guarantee that this will be the case) and, like Mitaka, there is no guarantee regarding their evolution.
- And finally, it is necessary to have the credentials to connect to **MISP** and therefore to have to store them to shape where to save them on a basis that has no guarantee of security is not the most relevant choice.

with all these elements in mind, it was decided that the best choice is to start from scratch and be specific to **MISP** (no external dependencies if possible).

In terms of carrying out the stages of the extension, we have retained:

- a storage API called browser.storage (different from window.storage!) for data storage.
- the optional part of the extension for data entry.
- a context menu API for the click.
- the fetch API (new standard replacing XMLHttpRequest) for connection with **MISP**
- Notification API for quick response as well as a new window for further viewing from **MISP event**.

Note that all these decisions were not taken before the start of the project but according to the result obtained on the previous stages.



Before you can make a functional and secure extension, you must understand and appropriate the various elements that make it up.

The first step was to create an extension with a simple manifest then to embed it in the browser. To do this, I, after some research and digging right to left, created a json file called manifest containing the required default keys (name, version, version of the manifest) that I added to the Chrome browser. It was a bit more complicated to add it to the Firefox browser because this method is not intuitive compared to the Chrome browser.

The second step was to test how the extension work in the background. This requires going to the browser extensions page and then examining the background console. As the previous extension does not contain any background script, we have to create a JavaScript file and add it to the manifest to get our background script.

In this JavaScript file, it is important to start by applying 2 functions of Runtime API. These functions are `onInstall` and `onStarted` which allow you to initialize the elements used in the extension as well as its behavior during its installation for the first and when opening the browser for the second.

Once this action was completed, I tried doing some simple functions that use the Alarms API to set a timer that will increment a counter every 2 minutes.

Alarms API makes it possible to create an alarm every X minutes, to modify them, to delete them and has a listener to manage these alarms when they are triggered.

In our case, an alarm is created every 2 minutes when installing the extension.

When the browser opens an alarm listener call is made but having no alarm instantiated at that time an alarm will be created. Once this alarm is triggered, it will recreate itself then either create a second alarm at the end of the next minute, or do nothing if this alarm already exists. This second alarm will increment the counter and display it via a message in the console.

Although the Alarms API has no interest in this extension (and therefore will disappear), it allows it allows to acquire knowledge of the modules and syntax used in an extension.

Note that each API requires permission to add to the manifest.

The next step is a test on the browser's Storage API. A color variable in hexadecimal is created and stored in storage when installing the extension and then retrieved in the popup of our Browser Action. The only difficulty here was to retrieve the data due to the asynchronous property of the `get` function.

Storage API contains the following functions:

- set: define a variable and its value
  - get: get the value of a variable
  - getBitesInUse: get the size of space used by a variable
  - remove: remove a variable
  - clear: delete the entire storage (specific to each extension, clear will not delete data other than that of the extension in which the function is called)
  - onChange: checks via a listener if a variable sees its value change (when the listener is triggered, we can get in a callback function the data.OldValue and data.NewValue)
- The use of storage will subsequently store the values necessary for the operation of this extension such as the options of the fetch request.

The fourth step was to allow the user to choose the color among several proposed in the option menu via 'option' in the browser extension page then after clicking on a button (of function 'myBrowser.runtime.openOptionsPage (); ' ) in the popup browser action. This step makes it possible to understand the mechanism of the options and therefore subsequently allow the user to choose the options of the fetch request (and which will be stored so as not to have to choose them each time).

The penultimate step consists of a way to apply the color selected by the user on the body of the web pages visited (when possible) via the popup browser action (the one being active on all pages compared to Page action) .

To do this, we had to retrieve the color via storage and keep it in a button in the popup.

When the user clicks on this button, we load script on the page which will search for all the body type elements and change their background property to the chosen color.

Note that it is safer to use the DOM than the Tabs API executeScript function, but it requires the DOM knowledge of the page in question.

The last step consists in carrying out the fifth step automatically, without having to click on the button of our popup, and this via the contentscript.

To do this, the method used in the content script is the same as in the popup but is done once the page has loaded its elements: instead of button, onclick we use runtime.onLoad.

We must also add the content script in the manifest as well as its urls on which it can act (in our case <all\_urls>).

Then, finally, I replaced the automatic side with a user action. An eventListener on pressing the space bar will activate the color modification script.

This Listener being located in the background script, we had to send a message to the content script to activate our script. These communications are carried out via Runtime API

which has a function to send messages and a Listener, allowing a task to be executed according to the message obtained.

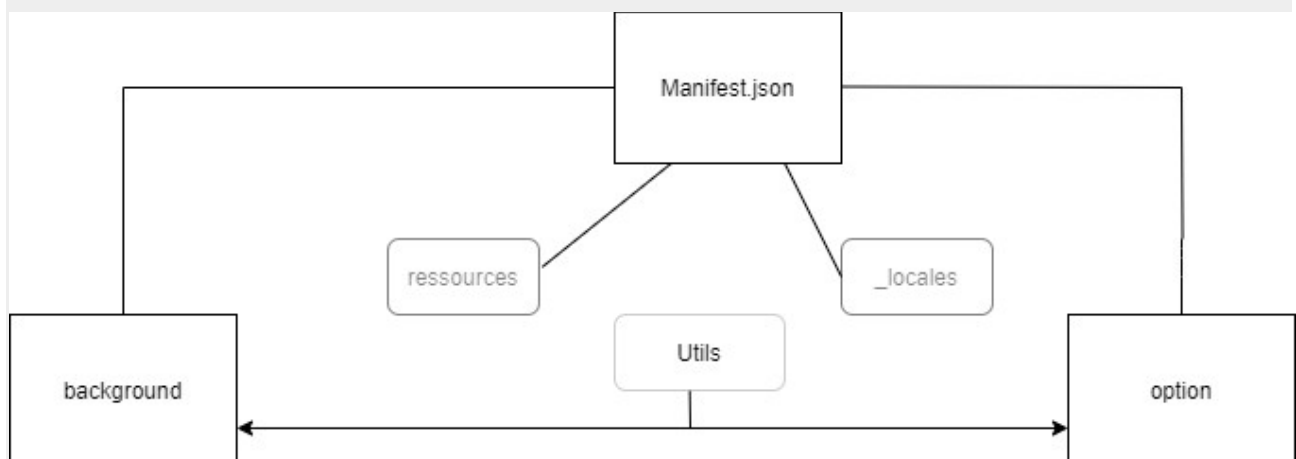
Messages communicate in the following way background → popup, content → background, popup → background. Note that the Tabs API send message function must be used to send message to content from background and popup can receive message only if it is opened.

Now that i have a good understanding of how to create an extension and also what i need to do, I started to develop the functionalities necessary for the success of the extension and then to secure it. Because while the APIs guarantee the security and reliability of the extension, there are still risks that must be taken into account and neutralized by the developer.

## Development

To solve the needs stated above, the extension uses different APIs in the background script part. It is made up of the following elements:

- The manifest
- A utility
- The resources
- The option part
- The background part
- \_locales



The resources folder is, as its name suggests, the folder that contains the extension's assets and in our case the icons used.

\_Locales is the folder containing the languages (English and French). i18n API allows you to define, thanks to a Json file, the texts to be displayed in the extension such as languages. A folder in \_locales for each language, in our case fr and en, will be used to display the texts (referenced with i18n). The Json file contains the different texts to display (keys) structured in an id for referencing with i18n, the content (the text to display) and possibly a description.

It is necessary to indicate in the manifest the language to be used by default, in the event that the browser language is not present in the folder \_locales.

Manifest contains the elements mentioned in the previous parts as well as the necessary permissions.

Permissions can also be requested directly from the user, especially for popups / notifications in the event that they prove to be redundant or intrusive.

The Utils folder contains functions and constants that are common to the various javascript files as well as the detection function of the browser used (so that the extension works in Chrome and Firefox without code modification). This can also be done with the 'polyfill' extension, but it does involve having external dependencies.

The options are used to store the data to perform the search with [MISP](#)

This folder consists of an html page (each html page is accompanied by its css and javascript files) of reception when the user accesses the options menu, of a page containing the different options during the request with [MISP](#) as well as a folder.

This folder contains an instance add page [MISP](#) (credentials) as well as a page for viewing the added instances and whether they can connect to [MISP](#).

For options in general, standard html / javascript code is used.

To add an instance, Storage API is used.

To check if the instance is valid Fetch API is used, if the response is accepted then the instance is valid, it is not otherwise.

The background Script of the extension is divided into 4 javascript files. Each file uses a different API:

- Context menu
- Fetch

- Storage
- The background part of the extension.

For the background part, there are the `OnInstall` and `OnStartup` functions which manage the storage state and the initialization of context menus and their listener as well as the management of access to options (via browser action, no popup).

The storage part manages the initialization and access to data in the storage during communication with [MISP](#).

The fetch part manages the communication with [MISP](#) by formatting the request to [MISP](#) standard.

Finally, the context menu part initializes the various context menus according to the selected data (text, url, etc.) to retrieve it and then process it (call to fetch).

Context menu API add items to the browser's context menu. There is different types of objects than context menu additions may apply to, such as images, hyperlinks, and pages. The "contextMenus" permission must be added into the extension's manifest to use the API. Also, a 16x16-pixel icon should be specified to display it next to the menu item. There is the create, remove and update function and also a event listener triggered when a context menu item is clicked.

Fetch API provides an interface for fetching resources across the network. It is now the standard replacing XMLHttpRequest. It offers similar features that are more flexible and more powerful.

Fetch provides a generic definition of Request and Response objects

The `fetch()` method takes an argument which is the path to the desired resource. It returns a Promise that resolves the Response for that request, whether successful or not.

Once the Response has been returned, there is a set of methods available to determine what the body content should be and how it should be manipulated (eg a response in Json).

A fetch is made up of a header which will be send to server and a method (POST or GET).

If the method is POST, a body can be added, containing the request option.

A credentials key can be added in case you want a specific use of user's cookies.

Now we have a working extension, datas are stored in the browser storage section thanks to Storage API. Those data can be modified by the user using the option menu. Context menu

API allow the user to select any data on the web page, then can query [MISP](#) about the selected data thanks to Fetch API.

To view the result of the query the user has the choice between a notification for a quick response or opening a new page for a more detailed response.

To notify the user, Notify API is used while Tabs API create a new tab if the user want the detailed response.

Note that Tabs API can execute script on the web page so developer must use the execute script function (from Tabs) with caution !

## Security

Now that we have a working extension, we have to check it security.

First of all, we have to ensure that our extension have the correct amount of privileges by checking the permission list in the manifest and remove all the unnecessary permission.

Note that ActiveTab permission grant privileges on the current tab to the extension while `<all_urls>` allow everything on a page from the extension. So when your extension need to carry out a common use, it is safer to add ActiveTab than `<all_urls>` in the manifest permission section.

But in our case, the fetch request apply of unknown url, so we have to use `<all_urls>`. It is the same with the Content Security Policy, where we can't restrict the html page interaction to allowed urls.

This is the reason why we have to redefine CSP in each html document, with `connect-src 'self'` or `script-src 'self'` if there is a fetch. As the result page html is just here to display the user fetch result and nothing more, it makes sense to use `default-src 'none'` on this document. (or at least `script-src 'none'` if there is a need of additional ressources).

All the step of this extension were done in accordance with the Extending Good Security Practices guide.

We use DOM when we need to modify a HTML page with javascript. DOM is safer than innerhtml cause characters are escaped (for your knoweldge, innerhtml must not be used to add field like style or script inside html tag) so

Unsafe values (like the response or user's input) are added with DOM function (as the guide said). The reason why innerhtml is used here is because it is more simple to deal with a dynamic table and a static css file.

As previously stated, we ensure the security in each of our html page by adding CSP rules. We have the background page containing all our background scripts, a fetch result page a show instance page, an option page showing the options, an other option page containing the fetch options and a last one that allow user to add MISP instance.

These pages should only be able to be applied on the browser so we have to add everything in our csp to 'self ' permission. As we don't use external ressources, we can even add 'none' permission except for « script-src » wich should be 'self' (script-src 'none' means there is no script allowed on our page). So we can for sure use « default-src 'none' » plus «script-src 'self' » instead of adding all the element one by one.

There is a few exception in our page, the first one is the background wich can , at any time, requested by the user, communicate with [MISP](#) so we have to add « connect-src 'url'» inside our CSP in the background script page (same goes for the MISP instance page). In our case, the url are unknow (the user have to add them) so we need to add \* instead of url. For all the other page, connect isn't added so default-src none is used.

In our fetch result page wich show the response of the user fetch call, there should not be anything at all except the DOM document so we have to set script-src to none (or in our case only put default-src to none).

Note we can add CSP inside our manifest and it will be applied to all page in our extension but it override all the CSP rules that already exist in our page wich means CSP rule should be the same in each page.

While viewing instances, the user can see the registered data. These data are displayed in the same way as on [MISP](#), url and response are normaly visible but the authentication keys are displayed with the first 4 and the last 4 characters of the key.

It is explicitly said that you should not store sensitive data with Storage API because stored datas are raw, but that what we have done in our extension : an url and an authantication key.

If there is no problem with the url, authentication should not be raw stored. This is why we used a data encryption function with [CBC](#), then we call an encrypt or decrypt function when it is needed.

This crypted data is set with a key of 42.

About the incoming value from our fetch function, those value are displayed in two way. The first one is a simple notification with the Notify API. As they are not customizable, characters are escaped, there is no possible processing of the data, there is no interaction

with user (except close the notification) and they are handle by the browser, this API is a safe way to display string. But that is all you can do, display text.

This is why there is another way to sho our fetch result : a dynamic html document that allow allow stuff Notify API doesn't, but there is also security threat.

However these threats are fixed by our security mesure.

Now we have checked the incoming datas, the extension permission and communication and even customize the CSP in our html documents. As we have done this verification and followed the Extending Good Security Practices guide, we can admit that our extension is safe.

Disclamer : there is always many way to increase the security level of a secure app, this extension is no exception.

## **Conclusion**

The basic functionalities of the project have been successfully implemented at the plugin.

The Firefox Store as well as the Chrome Store has this extension.



The extension allows you to perform the same action as MISP but simpler and more ergonomic. The user must add one or more MISP instances when installing the extension (can be modified later if necessary) then select the appropriate menu with left click each time he wishes to perform a search, instead of having to connect to your MISP instance then manually fill in a list of fields for each search.

In direct improvements, my level in JavaScript having increased during these 6 months, it is possible to improve the code written at the beginning of the internship such as for example using the shortcut '=>' or the promises to obtain a cleaner code of [sequential programming](#) (when this is necessary otherwise keep the [event programming](#)).

It is also possible to add more parameters in the options in order to provide a wider range of choices during a search.

It is also possible to migrate to a recently released V3 manifest now that everything works on the most stable version (V2) but not maintained.

This extension will have to be maintained in the future according to the needs of the users as well as their feedback but also according to the evolution of browsers.

In addition, actions could be added such as modifying the data of an event or one of the improvements mentioned above.

To conclude, I retained an extremely positive experience from this internship, from the discovery of a subject that I find most interesting (I had planned to launch into web extensions even without the internship) to the working conditions (going from 4h of public transport every day to 4h once every two weeks), but also the job satisfaction (I leave the appreciation of the word "good" to my internship supervisor) done!

This internship also allowed me to improve my understanding of web languages.

For the Safety part, the knowledge provided by the practical and theoretical courses of my training was useful to me. Computer security is very extensive, however, I regret not having been able to implement aspects other than website security.

## *Bibliography*

[https://fr.wikipedia.org/wiki/Pretty\\_Good\\_Privacy](https://fr.wikipedia.org/wiki/Pretty_Good_Privacy)

<https://stackoverflow.com/>

<https://github.com/ninoseki/mitaka>

<https://content-security-policy.com/>

*Some links for event/sequentiel/concurrent programming informations (not covered in this topic )*

[https://fr.wikipedia.org/wiki/Programmation\\_%C3%A9v%C3%A9nentielle](https://fr.wikipedia.org/wiki/Programmation_%C3%A9v%C3%A9nentielle)

[https://fr.wikipedia.org/wiki/Programmation\\_concurrente](https://fr.wikipedia.org/wiki/Programmation_concurrente)

[https://fr.wikipedia.org/wiki/Programmation\\_s%C3%A9quentielle](https://fr.wikipedia.org/wiki/Programmation_s%C3%A9quentielle)

## #JavaScript

<https://owasp.org/www-community/attacks/xss/>

<https://developer.mozilla.org/fr/docs/Web/JavaScript/>

[https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using\\_promises](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using_promises)

[https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Functions/Arrow\\_functions](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

<https://www.w3schools.com/>

<https://developers.google.com/web/fundamentals/security>

## # Extensions

<https://extensionworkshop.com/documentation/develop/build-a-secure-extension/>

*Only generic sites are given here, there are pages dedicated to each part used in this extension as well as general guides.*

<https://developer.chrome.com/docs/extensions/>

<https://developer.mozilla.org/fr/docs/Mozilla/Add-ons/WebExtensions>

## Glossary

**CIRCL** *Computer Incident Response Center Luxembourg*

**CERT** *Compteur Emergency Response Team*

**CASES** *Cyberworld Awareness & Security Enhancement Services*

**ICT** *Information and Communication Technologies*

**FIRST** *Forum of Incident Response and Security Teams.*

**MISP** *Malware Information Sharing Platform*, a Threat Sharing project of CIRCL

**Element** Free instant messaging software encrypted and licensed Apache 2.0

**bbb** Simple interface for open-source web conferencing server

**PGP** *Pretty Good Privacy*, data protection technique

**Event** Result of a search on MISP containing various information

**SIEM** *Security Information and Event Management*, tool for managing security events and information within a domain at the same time. A SIEM must collect, analyze, correlate, centralize and display these events / information

**IDS** *Intrusion detection System*, Mechanism making it possible to identify abnormal or suspicious activities on a target (network or host) and therefore to have a look at the successful and failed attempts of intrusions thanks to signature detections (recognition of malicious program) and anomaly detections (detection of behavior deviations)

**IOC** *Indicators Of Compromise*, artefact (virus signatures, particular IP addresses, hashes of malicious files) observed on a network or in an operating system revealing, with a high level of certainty, a computer intrusion.

**Sequential/Event programming** the 'code' is asynchronous/synchronous  
When something is running in a synchronized way, it is expected from that to finish before it move on to another task, But when it run asynchronously, it can move on to another task before it completes.

**Uuid** *Universally unique identifier*, identifiers allowing distributed systems to uniquely identify information without significant central coordination

they are unique and are encoded on 128 bits and are generated using pseudorandoms as well as the characteristics of a computer (hard disk number, MAC address, etc.).

**Packed extension** Extension added in raw in a browser (Chrome or Firefox). In Firefox, unpacked extensions are deleted when the browser is closed.

**Unpacked extension** An extension whose folders / files have been compressed into a file (xrc for Chrome xpi for Firefox).

**Inline Script** call to a script directly in the html page: `<script>Inline Script</script>`

**AES** *Advanced Encryption Standard*, is a symmetric block cipher algorithm, successor to the Data Encryption Standard (DES) algorithm, with symmetric (secret) key.

**CBC** *Cypher Block Chain*, is a method wich encrypt a text block by block by applying a XOR on a text with a key (called Initialization Vector) to get the first block, then the previous one is XORed with the next one until the last block.

## *Table of Appendices*

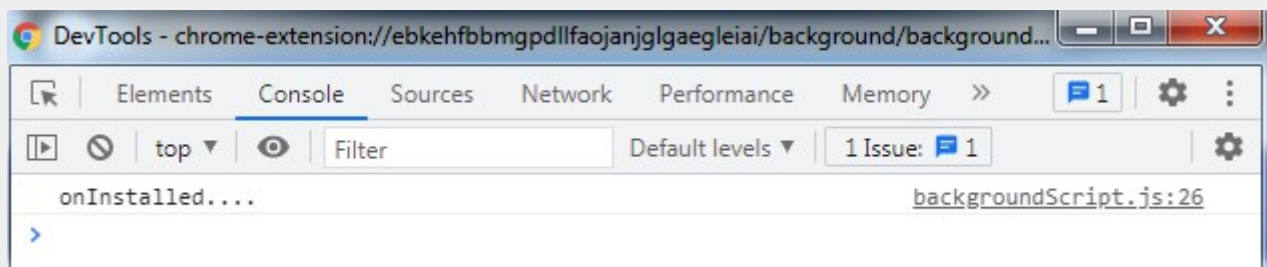
<i>annex 1</i> : Background Script installation Example	30
<i>annex 2</i> : Background Script alarm Example	31
<i>annex 3</i> : Background Script DIV Color Example	32
<i>annex 4</i> : Send and receive script with Fetch	33
<i>annex 5</i> : Test Event with 8.8.8.8 value	34

## *annex 1 : Background Script installation Example*

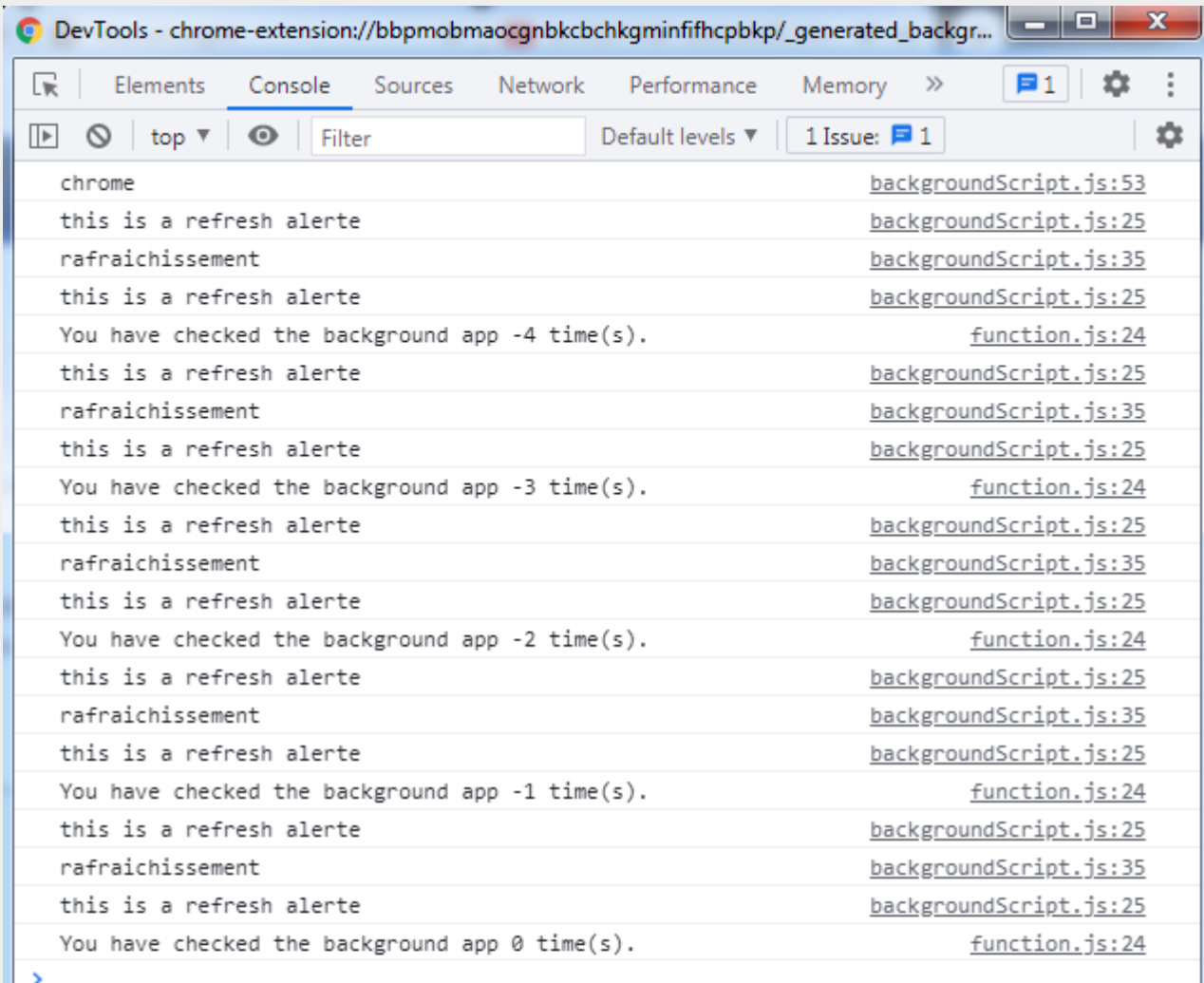
*when the extension is installed, we send a log to the console*

```
1  var myBrowser = findBrowser();
2
3  myBrowser.runtime.onInstalled.addListener(() => {
4    console.log('onInstalled....');
5    myBrowser.storage.sync.set({color: '#3aa757'}, function({}));
6
7    scheduleRequest(myBrowser);
8    scheduleWatchdog(myBrowser);
9    startRequest(myBrowser);
10   //clear(myBrowser);
11  });
12
13
14  myBrowser.runtime.onStartup.addListener(() => {
15    console.log('onStartup....');
16    startRequest(myBrowser);
17  });
18
```

*We see the log in the console.*



## annex 2 : Background Script alarm Example



chrome [backgroundScript.js:53](#)

this is a refresh alerte [backgroundScript.js:25](#)

rafraichissement [backgroundScript.js:35](#)

this is a refresh alerte [backgroundScript.js:25](#)

You have checked the background app -4 time(s). [function.js:24](#)

this is a refresh alerte [backgroundScript.js:25](#)

rafraichissement [backgroundScript.js:35](#)

this is a refresh alerte [backgroundScript.js:25](#)

You have checked the background app -3 time(s). [function.js:24](#)

this is a refresh alerte [backgroundScript.js:25](#)

rafraichissement [backgroundScript.js:35](#)

this is a refresh alerte [backgroundScript.js:25](#)

You have checked the background app -2 time(s). [function.js:24](#)

this is a refresh alerte [backgroundScript.js:25](#)

rafraichissement [backgroundScript.js:35](#)

this is a refresh alerte [backgroundScript.js:25](#)

You have checked the background app -1 time(s). [function.js:24](#)

this is a refresh alerte [backgroundScript.js:25](#)

rafraichissement [backgroundScript.js:35](#)

this is a refresh alerte [backgroundScript.js:25](#)

You have checked the background app 0 time(s). [function.js:24](#)

>

**Use of Alarm API.**

**cpt defined to -5 on extension install.**

**Each 1 minutes there is a refresh alarm saying the next minutes there is an other alarm, set an other refresh alarm and also a refreshment alarm if not already created at 2 minutes.**

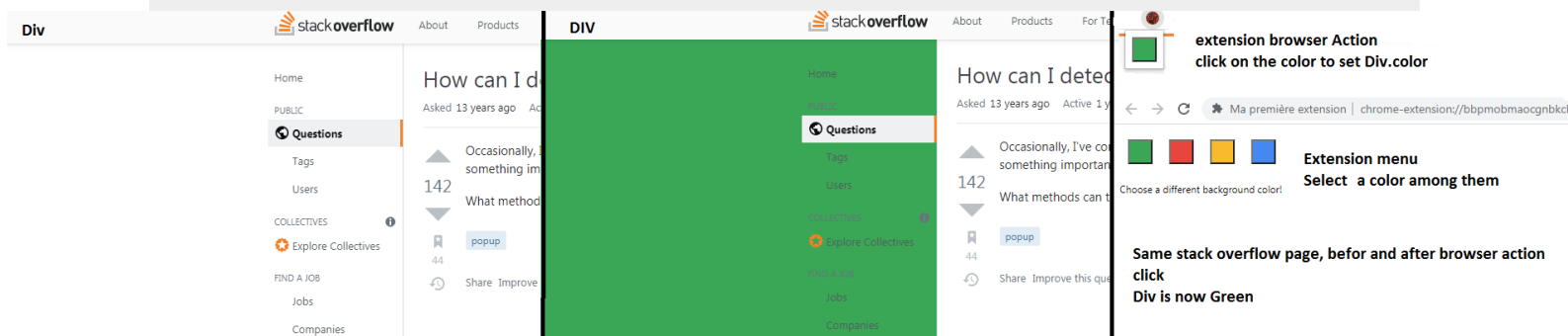
**So in each row (2 minutes) the refresh alarm (one per minute) and refreshment alarm (one per 2 minutes) are triggered. (1 time refreshment alert and 2 times refresh alert).**

**When refreshment alert is triggered,  $cpt = cpt + 1$  .**

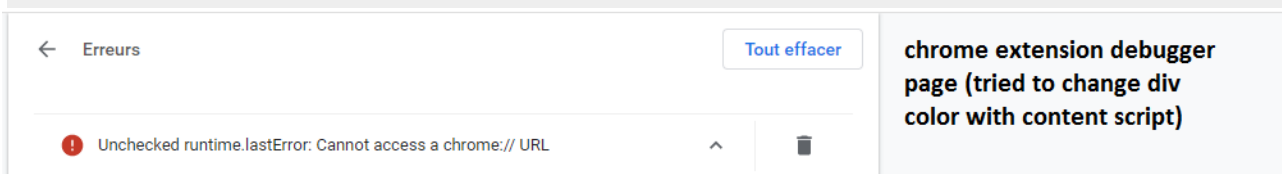
**if  $cpt = 5$  then  $cpt = -1$**

### *annex 3 : Background Script body Color Example*

```
3 myBrowser.storage.sync.get('color', function(data) {
4     changeColor.style.backgroundColor = data.color;
5     changeColor.setAttribute('value', data.color);
6 });
7
8 changeColor.onclick = function(element) {
9     let color = element.target.value;
10    myBrowser.tabs.query({active: true, currentWindow: true}, function(tabs) {
11        myBrowser.tabs.executeScript(
12            tabs[0].id,
13            {code: 'document.body.style.backgroundColor = "' + color + '"'});
14    });
15 };
```

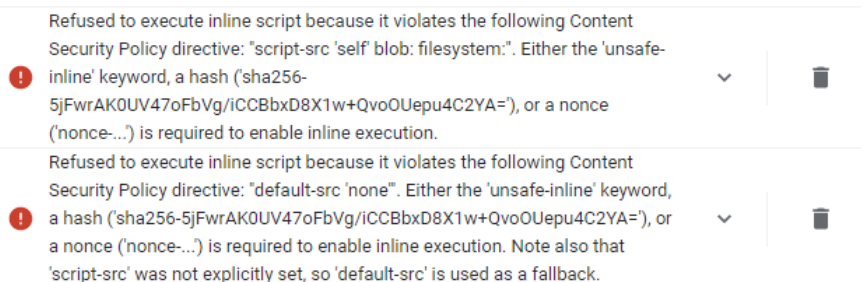
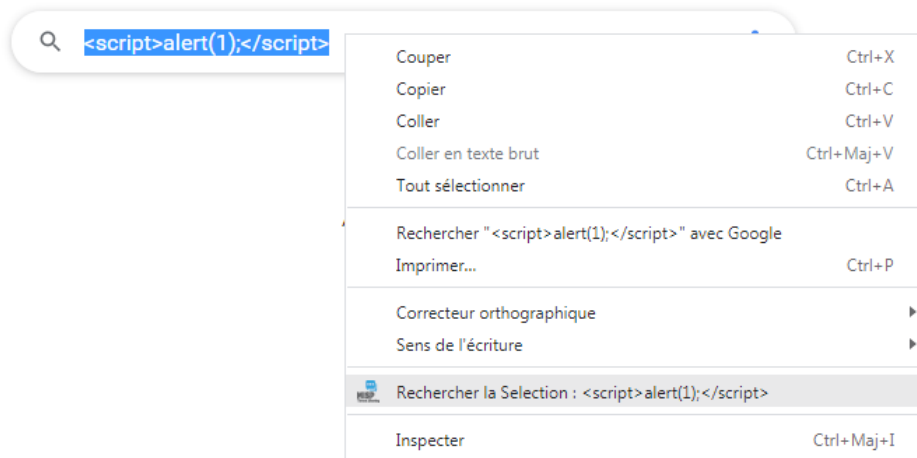


*Browser action and content script can't access private browser page.*





## annex 4 : Searching script with Fetch

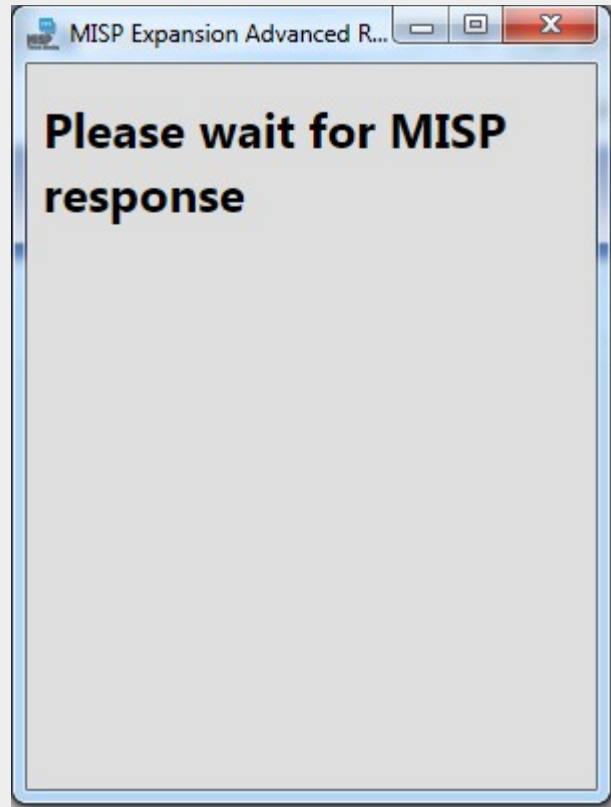


**forbidden use of inline script by default**

**Default-src set to none on the html document**

*Note we get this with innerHtml function. With the DOM attribut, it works fine*

## *annex 5 : Test Event with 8.8.8.8 value*



MISP Expansion Advanced Response

**Value: 8.8.8.8**

**Instance:** <https://misppriv.circl.lu>

Nombre d'événements dans la requête: 76

Nombre d'événements maximum affichés: 100

**Evenement : 1**

Nombre d'attributs dans l'événement: 18  
Nombre d'objets dans l'attribut Event: 6  
<https://misppriv.circl.lu/events/view/51b6cfa-6a0c-4592-b13a-05a70a000b01>

event\_info: Microsoft b54 IP list given under confidential terms. (We assume TLP:AMBER classification is fine) As you know Microsoft did the b54 Citadel action and was a bit counter productive as the list contains some false positives (as an example the 8.8.8.8 from Google was playing a role but Microsoft did not explain what role in Citadel infection) but also a list of known/unknown sinkholes operated by other parties. <http://www.abuse.ch/?p=5362> (for the feedback from Abuse.CH) The list might be still useful to use for finding previous "undetected" Citadel infection.

UUID: 51b6cfa-6a0c-4592-b13a-05a70a000b01

ID: 16600

event\_id: 182

**Evenement : 2**

Nombre d'attributs dans l'événement: 18  
Nombre d'objets dans l'attribut Event: 6  
<https://misppriv.circl.lu/events/view/51b6cfa-6a0c-4592-b13a-05a70a000b01543b8159-ab20-4664-a112-d44a950d210b>

event\_info: Test event for MISP 2.3

UUID: 543b8159-ab20-4664-a112-d44a950d210b

ID: 56066

event\_id: 583