Chrome Extension Pentest Toolkit

This repository contains a Chrome extension that facilitates the demonstration and testing of various web security exploits. Follow the setup instructions below to get started.

Bachelor	Toegepaste Informatica
Keuzetraject Afstudeerrichting	Cyber Security
Academiejaar	2023 - 2024
Student	Thor Demeestere
Interne begeleider	Matthias Blomme
Externe promotor	Tobias Chielens

Licensing

As required by the GNU General Public License v3.0, the source code will be made publicly available on my GitHub. Everyone is permitted to use, distribute, and modify the code, provided that any unmodified or modified versions remain licensed under the same license. Additionally, patents can be obtained only if such patents are licensed for everyone's free use. If any of these conditions are violated, the user's rights are terminated; however, rights can be reinstated upon cessation of the violating behavior. This policy ensures that the software remains free and open, and that its derivatives are bound to the same freedoms and protections.

Disclaimer

This Chrome extension and associated exploits are intended for educational and testing purposes only. Do not use these exploits on any system or website without proper authorization. The developers and contributors of this repository are not responsible for any misuse or damage caused by these exploits. Always ensure that you have appropriate permissions before conducting any security testing.

Table of contents

Why I Started Developing a Chrome Extension

It all began with my frustration at school, specifically with the cumbersome process of viewing documents on LEHO. I had to scroll three times just to see a single page in its entirety.
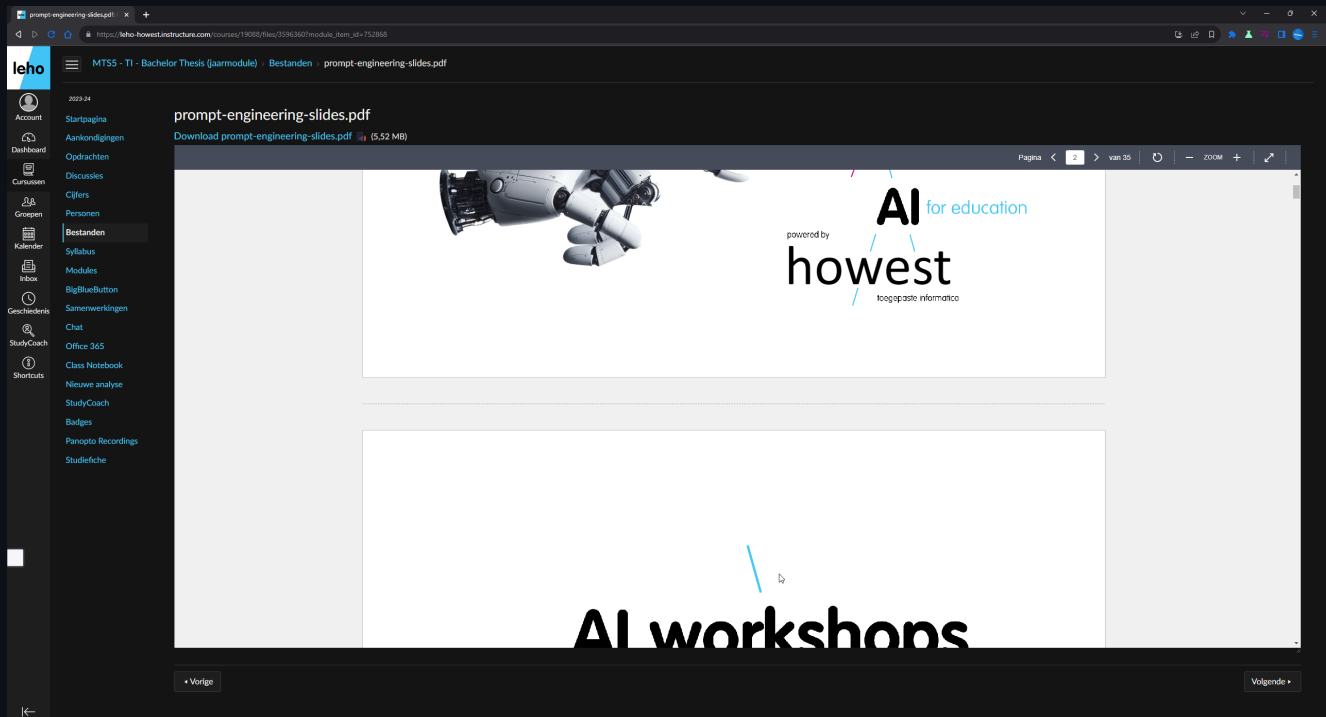

This led me to write a small piece of JavaScript and execute it in the console. Later, I evolved this script into a bookmarklet. However, I still found it inconvenient to open a document and press the bookmark each time. Thus, I embarked on a quest to find a way to automatically execute JavaScript whenever a LEHO document was opened. This journey led me to the world of browser extensions.

Although there were existing extensions like Tampermonkey that could potentially meet my needs, they didn't perform exactly as I wanted it. They were either too complex for my simple requirements or failed to function in the desired manner. As a result, I decided to develop my own extension.

Why a chrome extension?

The biggest reason: chrome is my default browser. Afterwards, when thinking more about it: the chrome browser itself has a market share of ~65%, safari ~15% (but i don't have a MacBook so this wouldn't be feasible), firefox ~5% and the last ~15% are the other browsers and a lot of those are chromium based.
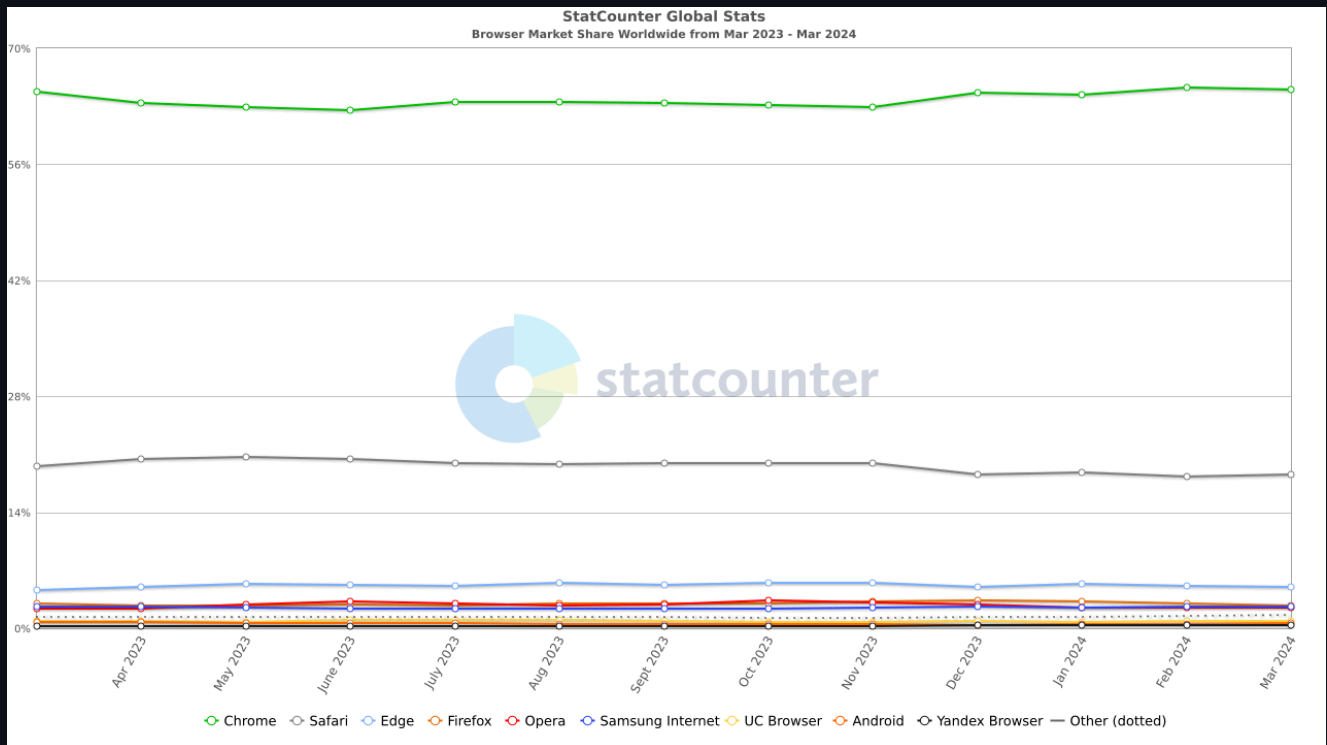
Although initially i tried to develop the extensions for both Firefox and Chromium-based browsers i found Google's documentation to be superior. Furthermore, after spending hours trying to make it work on Firefox without success, I decided to focus on Chromium-based browsers, which also offered the advantage of wider applicability due to their increasing market share.

What are extensions actually

Extensions are JavaScript programs that run either in the background or within the same execution environment as a webpage.
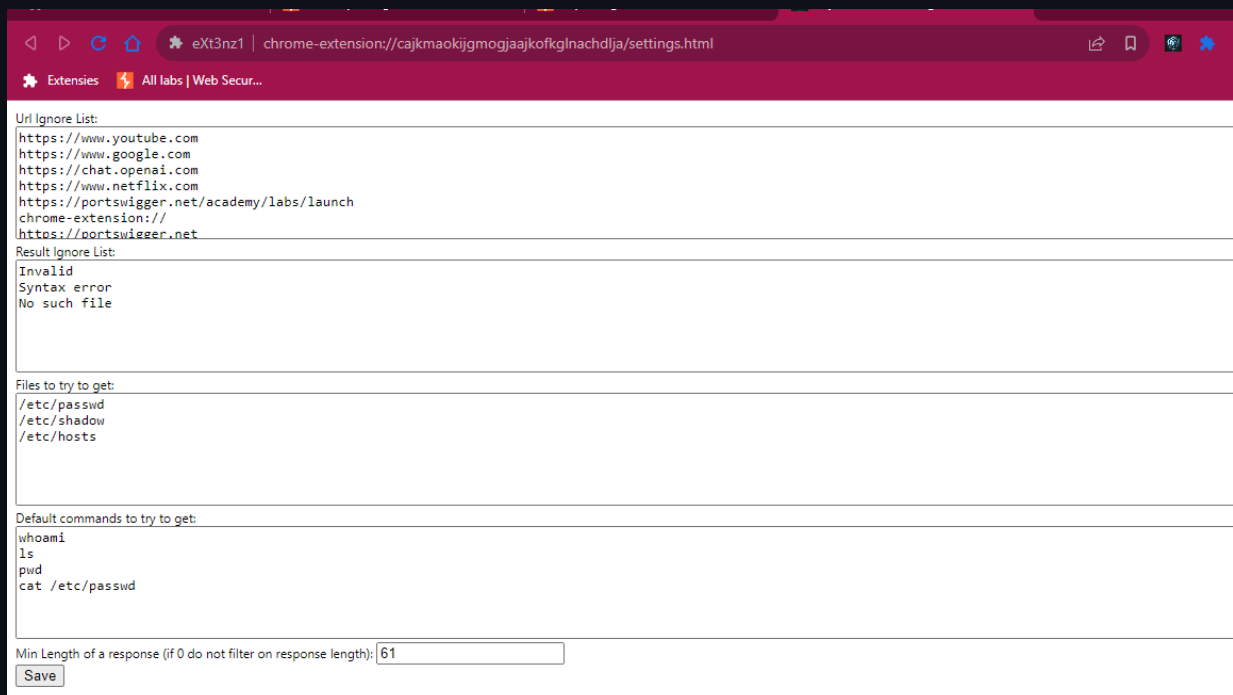
Development process

The first automatic exploit I worked on involved an XML attack, which I learned about from an article. After successfully implementing this exploit, I took the time to modularize my code, breaking it down into separate files and folders for better organization and maintenance. This also enabled me to add a "module" to insert new exploits when they got developed.

Key Components of the Extension

Before i start explaining the exploits there are some things we need to get familiar with:

- **manifest.json**: This file acts as the backbone of a Chrome extension. It configures the extension by declaring its major components and requesting the necessary permissions to function properly.
- **background.js**: Think of this as the JavaScript that powers the entire browser extension environment. While it cannot interact directly with the content of each tab, it can intercept requests and manage broader extension activities.
- **offscreen.html**: This HTML file exists within the same operational context as a browser tab but remains invisible to the user. It serves as a hidden layer where certain extension processes can run in the background.
- **listener.js (previously offscreen.js)**: Associated with offscreen.html, this JavaScript file is dedicated to monitoring specific events or "attacks" within the extension's hidden realm. It is where we listen for and respond to various triggers.
- **content.js**: This script runs in the context of each browser tab, directly interacting with the web content. It is isolated to the tab it operates in, ensuring that its actions are tab-specific and do not interfere with the broader browser or other tabs.
- **popup.html/js**: This is the interface you see when you click the extension's icon, where the results for the current tab are displayed.
- **settings.html/js (previously options.html/js)**: In this section, we configure settings, primarily filters for results and URLs to ignore during a penetration test.
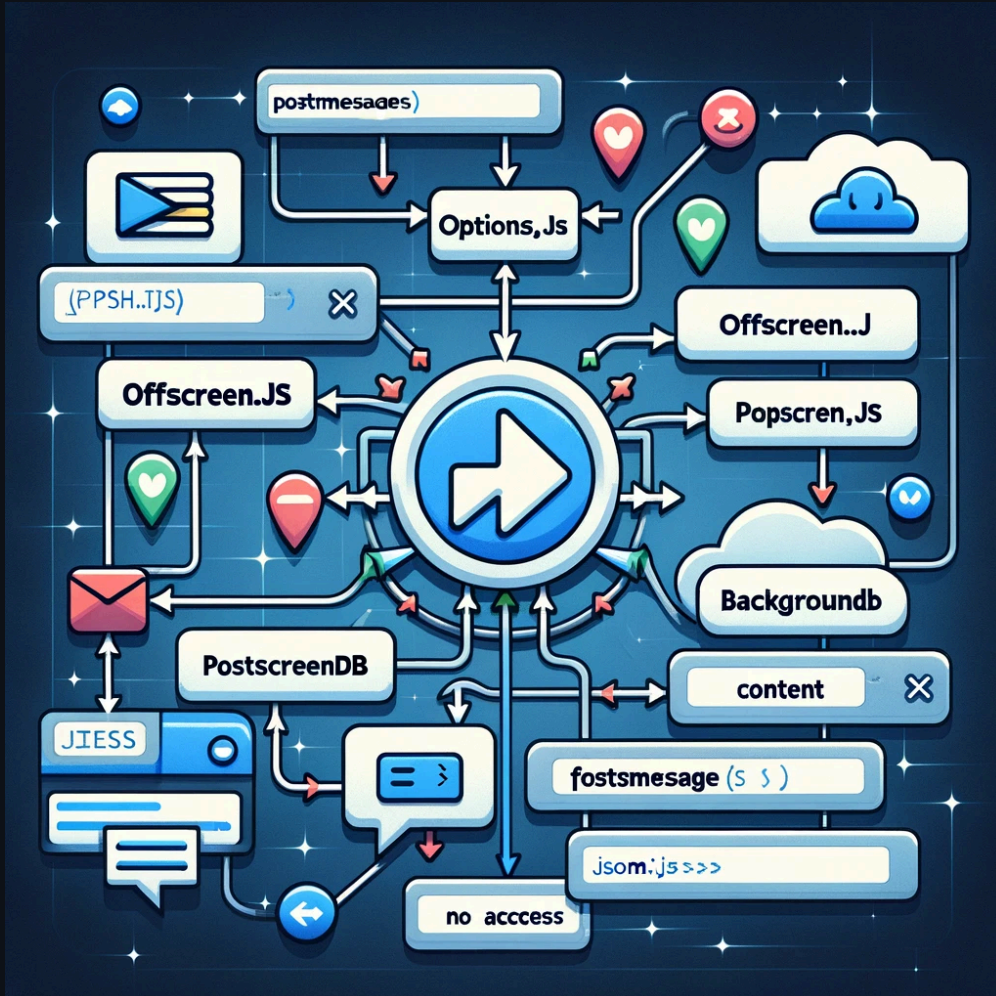
Issues

Scope

During the development process, I encountered multiple challenges related to the scope of JavaScript variables and methods, as well as storage.

As depicted, the structure is complex, but I will simplify it:

Firstly, regarding JavaScript scopes:

- You have a browser which is interacted with via background.js. It contains tabs that are managed through content.js.
- Each of these components has its own scope and communicates with others via post messages.
- Additionally, an offscreen HTML page is started up, loaded from the background, and has its own scope.

Except for content.js, all of these are modules, which allows the use of the import method. This is useful for separating different functions into different files. However, since content.js is not a module, I had to duplicate some methods that I wanted to use in my content scripts.

Popup and settings pages also have their own scopes and are reloaded each time they are opened, so they don't need to directly share data (except for stored data).

Then, regarding the storage scope:

- While implementing localStorage and cookies is straightforward, they can only be accessed from content.js.
- We needed storage that could be accessed from each part of the extension. Therefore, I opted for IndexedDB. Although relatively slow, it gets the job done.

Results

Initially, I always opened the extension's development console and logged each response. While this is easy and fast for checking functionality during development, it is not suitable for the final product. Therefore, I explored methods to save the results and subsequently needed to determine how to display them. In the end, I decided to store them in IndexedDB and display the results of the current page when clicking the extension's icon.

Tools and Methodology

There where also tools, that i used, to debug, to understand, to explore the internet:

For examining and modifying requests, I utilized Burp Suite by Dafydd Stuttard, a tool that lets you intercept, view, and modify the requests sent from your browser. My approach involved intercepting every request using background JavaScript. This method allowed for real-time monitoring and modification of requests before they were completed, offering insights into potential exploits.

Chrome DevTools: This integrated set of tools is built directly into the Google Chrome browser, allowing developers to edit pages on-the-fly and diagnose problems quickly, which helps in understanding the detailed workings of web applications. I used Chrome DevTools extensively for inspecting HTML, CSS, and JavaScript, observing network activity, and managing browser storage among other things.

Google Documentation: Google's comprehensive documentation resources were invaluable for understanding best practices and how to effectively use various APIs and services. I frequently referred to Google's developer documentation to ensure that I was using the most up-to-date and efficient methods available.

ChatGPT: For synthesizing information, explaining complex concepts in simpler terms, spellchecking, and generating images, I used ChatGPT. It was particularly helpful not only for quick summaries and clarifications on technical topics but also for creating visual content that aided in illustrating points and enhancing presentations. The ability to generate images helped in visualizing concepts that were otherwise abstract and difficult to convey through text alone.

Each tool played a critical role in enhancing my capability to develop, troubleshoot, and optimize my applications efficiently.

Chrome Extension Exploit Identifier and Status Table

This is an overview of all the exploits and the status of how far they have been implemented yet and also links to info about the exploits.

Exploit Identifying name in project	Exploit Link	Description	Implementation status
post_formbody_attack	Exploit Link	This exploit demonstrates an XXE (XML External Entity) attack by exploiting vulnerabilities in the `post_formbody_attack` scenario.	☑️
post_xmlbody_attack	Exploit Link	This exploit showcases the exploitation of XXE to retrieve files by targeting the `post_xmlbody_attack` scenario.	☑️
blind_xxe_with_data_retrieval_via_error_messages	Exploit Link	This exploit aims to demonstrate blind XXE with data retrieval via error messages.	🟡 Pending
get_filepath	Exploit Link	This exploit demonstrates file path traversal vulnerabilities by targeting the get_filepath scenario.	☑️
postmessage_identifying	Info Link1;Info Link2	This identifies possible vulnerabilities through the exploitation of the postMessage method.	☑️
General Info		Site URL: example.com; IP Address: 192.0.2.1; Full URL: https://example.com; includes git folder	🟡 Pending

Exploiting and Refining

Generally, here's how the process works: When you visit a webpage and engage in activities like clicking buttons or other interactions, the extension detects specific identifiers or "footprints." If it recognizes any, and there are associated payloads, it sends these to an offscreen document. In this offscreen document, the extension tests the payloads, and then filters the results, which can be adjusted in the settings. Subsequently, the filtered results are stored in the IndexedDB, organized by the tab they originated from. Additionally, if there are any successful results, a counter is displayed on the extension's icon in the taskbar, also tab-specific.

XInclude Attacks

XInclude attacks occur when XML parsers on the server-side process input that includes XInclude statements. By default, XInclude attempts to parse included documents as XML. However, attackers can exploit this feature to include non-XML files such as "/etc/passwd" by specifying the parsing method as text. This vulnerability allows unauthorized file access through crafted XML input.

XML External Entity (XXE) Injection

XXE injection involves exploiting the feature in XML where external entities can be defined and used within the document. When XML input containing external entity declarations is processed, it can be manipulated to include data from system files like "file:///etc/passwd". This type of attack can lead to data exposure or retrieval of sensitive files.

Path Traversal

Path traversal vulnerabilities occur when input values (such as file paths) provided by a user are not properly sanitized, allowing attackers to navigate the server's directory structure. An example is input like "../../../../etc/passwd" which can lead to unauthorized access to critical system files.

PostMessage Vulnerabilities

The postMessage method is typically used in web applications to enable secure communication between windows or frames from different origins. However, if not properly implemented, it can be vulnerable to attacks. Malicious scripts can exploit postMessage to execute unauthorized actions or access sensitive data. Identifying and mitigating these vulnerabilities requires careful scrutiny of how messages are validated and handled within the application.

Setup

1. Clone this repository to a known location on your system, for example, `~/chrome-extension`.

2. Open Google Chrome.

3. Navigate to `chrome://extensions/` in the address bar.

4. Enable developer mode by toggling the switch located at the top right corner.

5. Click on the "Load unpacked" button and select the folder where you cloned this repository (`~/chrome-extension`).

6. The extension should now be loaded into Chrome.

7. Navigate to a webpage that you wish to test for vulnerabilities.

How to use

For now, you can visit websites and freely explore their features. If the extension detects a request that might be exploitable, it will automatically attempt certain actions. You can view the results in the extension's service.

1. Open Google Chrome.
2. Surf website you may legally pentest
3. After a while, if there are any exploits that succeeded, there will show a badge with the amount of exploits that succeeds

4. Then you can click on the popup and it will show the exploits with a button to copy the executed exploit

AI Prompts

I used various prompts for generating code and explaining how certain pieces of code work:

- Can you explain how this works ...

- Can you write an example for ...

Additionally, I primarily used prompts for improving and rewriting texts:

- Can you proofread this text and correct any errors: ...

I also used AI to generate images for my presentation.

References

- "All labs | Web Security Academy." https://portswigger.net/web-security/all-labs
- "API reference," Chrome for Developers. https://developer.chrome.com/docs/extensions/reference/api
- "Chat GPT." https://chat.openai.com/auth/login
- A. Leybourne, "Chrome Extension Icon Generator." https://alexleybourne.github.io/chrome-extension-icon-generator/
- "Extensions / Manifest v3 | Chrome for Developers," Chrome for Developers. https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3
- Scribbr, "Free Citation Generator | APA, MLA, Chicago | Scribbr," Scribbr, Mar. 21, 2024. https://www.scribbr.com/citation/generator
- "People of Twitter." https://twitter.com/home
- "Using IndexedDB - Web APIs | MDN," MDN Web Docs, Jan. 30, 2024. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB
- "Extensions / reference," Chrome for Developers. Available: https://developer.chrome.com/docs/extensions/reference
- C. Rai, "Exploiting PostMessage() - Chirag Rai - Medium," Medium, Jan. 06, 2022. Available: https://medium.com/@chiragrai3666/exploiting-postmessage-e2b01349c205
- DEFCONConference, "DEF CON 31 - Infinite Money Glitch - Hacking Transit Cards - Bertocchi, Campbell, Gibson, Harris," YouTube. Aug. 23, 2023. Available: https://www.youtube.com/watch?v=1JT_ITfK69Q

Honorable mentions

- Koen Koreman aka Koenk: The man, the myth, the legend, himself. Someone i look up to, one of the best docents i've ever had.
- Tobias Chielens: For the interesting internship.
- Matthias Blomme