Detection of malicious Chrome extensions using CodeQL and ChatGPT

Carl Wang & Stella Carlsson (Group 27)

I. Background

Today, browser extensions are widely applied by users to enhance the functionality of their browser, namely Google Chrome, FireFox and Microsoft Edge. This accessibility comes with significant potential risks, since the extensions are granted with permissions of employing browser component, such as bookmarks, webpage data, browser history[1]. During the interaction of extensions and malicious web pages, extensions are possible to be abused to leak privacy information[2]. Moreover, the extensions themselves can also be malicious and attempt to steal user data[3]. Extensions consist of three main components: manifest file, content script, and service worker. The manifest file is used to request permissions, and the content script is a static JavaScript file combining with the viewing webpage, and the service worker is another JavaScript file which reacts to browser events, such as submitting a password. The code that attempts to steal user data can exist locally in the JavaScript files, but it can also be injected remotely. To prevent data leaks caused by remote code injection, Google introduced a new policy, Manifest V3, which disallows some dangerous operations, such as ***eval***, in the source code of the extensions. Unfortunately, it is not enough, malicious extensions remain existing[4][5].

Therefore, to detect potential malicious behaviors in browser extensions, a deep analysis of the extensions' source code is necessary. The selected tool to fulfill this need is CodeQL. CodeQL is a static analysis engine that parses source code in to a database and further analyzes it with queries. With proper queries, CodeQL is capable of detecting various issues in the code, including insecure API usage, code injections and secret leak[5][6].

However, CodeQL cannot give an accurate evaluation on the found risks, whether they are acceptable or unacceptable. With the introduction of ChatGPT, a determination of keeping or removing the extensions containing risky code can perhaps be made with a certain accuracy[7]. A study revealed that test results of ChatGPT 4 in Null Dereference bugs and Resource Leak bugs are better than the current leading bug detector in the scale of 12.86% and 43.13% respectively[8].

II. Goal

Our goal is to create a malicious extension detection program by combining CodeQL and ChatGPT. The code with potential risks is first selected using the queries in CodeQL, and then sent to ChatGPT to evaluate whether they are dangerous. Lastly, a report will be produced based on CodeQL results and ChatGPT responses.

III. Concept

A. Potentially dangerous operations

We have surveyed the existing literature to identify potentially dangerous JavaScript operations. For this project we only looked into Chrome extensions. As mentioned in the Background Chrome now uses the Manifest V3 policy. This policy already prevents some dangerous operations from being used by extensions such as `eval()`, which means it would be useless to look for these operations. We only focused on operations that are still allowed. So first off, we have the `chrome` operations such as for instance `chrome.storage` and `chrome.scripting`. These operations can in general be used to get and manipulate data of the browser, which enable malicious behavior. We also have some other operations that might be used to send stolen data such as `window.fetch` and `window.fetch`. Hence we have chosen known operations which can be used maliciously, however we have not done so in a systematic way since this isn't the primary goal of our study. This is only an example of the kinds of operations an analysis might want to be looking into. All chosen operations can be seen in Table 1.

B. CodeQL techniques

We chose to use CodeQL to look for predetermined potentially dangerous operations (see Table 1). We chose to do this because extensions have the unique ability to manipulate browsers through certain operations. This is useful and gives extensions their functionality, but if these privileges are used in malicious ways then the potential damage is big. Hence, we wanted to focus on identifying when these sorts of ”privileges” were used and then try to detect whether the usage was malicious or not. Since CodeQL can't really determine this, it was used as a tool to identify the usage of these powerful operations.

C. *ChatGPT Model*

We used the ChatGPT 4o model for this project. This was the most powerful GPT model available to us at the moment. Since the previous model ChatGPT 4 has been shown to be successful at detecting bugs we had reason to believe that this model would be at least as successful at this similiar task [8].

IV. Implementation

A. *Choice of extensions*

To evaluate the effectiveness of the program, two sets of Chrome extensions are selected. The first set contains several unsafe Chrome extensions, which were obtained from the files provided by the authors of Codex[5], which is called ”Malicious dataset” in this project. The other set of Chrome extensions was obtained from Google Chrome's official Github repository[9], which contains simple example extensions, this set is called ”Safe dataset” in this project.

B. *CodeQL queries*

In this project, the following JavaScript operations are extracted and evaluated:

chrome.	**Others**
cookies()	window.parent.postMessage()
history()	fetch()
bookmarks()	window.atob()
tabs()	window.btoa()
scripting()	String.fromCharCode
webRequest()	navigator.sendBeacon()
storage()	
identity()	
management()	
runtime()	
alarms()	
notification()	
declarativeNetRequest()	
webNavigation()	

Table 1: Operations chosen to be detected by CodeQL

CodeQL is used to find and allocate the positions of occurrences of above-listed operations. Furthermore, the specific operation name, for example chrome.tabs.create() is also stored in the produced csv file. For instance, after running the queries, the result csv file would contain information like this:

```
col0: FILE_NAME_WITH_PATH,
col1: LINE_NUMBER,
col2: "chrome.bookmarks.search",
col3: "chrome. ...      })"
```

The first column contains the file name and also the path traversal required for that file. The second column contains the line number where the operation is first called. The second column contains the operation's full name, and the last column contains the abstract code of that operation, how the call begins and ends.

C. *Automation Script*

A python script is used to automate the process of running CodeQL queries and extracting the corresponding static code from the extension source files.

The first step is quite straight forward, the CodeQL commands are called to run all queries on both databases created from the selected sets of Chrome extensions, and produce above-mentioned csv files for each query. Since these csv files contain the file name, the line number and the function name of the code that should be evaluated, the program goes to the line in the file, and starts reading once the function name is found, and stops when the parentheses are balanced. However, this introduces a problem. It is observed that source code of many Chrome extensions is compressed into nearly unreadable format to minimize their size. Many lines of code are placed in a single long line, therefore potentially include several occurrences of same function. To solve this, the number of occurrences in a single line is first counted, and then the code is extracted one by one. In the end, the code are combined with the csv files and a new csv file with combined information is produced.

D. *ChatGPT API*

In the design of our project, the combined result csv file should be sent to ChatGPT API after several prompt adjustments. However, this method is abandoned because it requires an additional cost of using ChatGPT API, based on the quantity of tokens used. Therefore, ChatGPT web application is used instead for this project. To demonstrate the ability of connecting the program and ChatGPT, a template is given in following code snippet.

```
from openai import OpenAI
client = OpenAI(api_key=
”OPENAI_API_KEY”)

...

message = (
”Prompt adjustments: for example: evaluate
the content of following csv file,
it includes code extracted from
different javascript files,
    you can see where the file
    comes from in the first column,
    for each js file,
    evaluate them as dangerous
    (malicious or exposes
    vulnerabilities
    to malicious websites)
    or safe."
)

for file, code in results.items():
    message += f"File: {file}\n"
    message += "\n".join(code)
    message += "\n\n---\n\n"

try:
    response =
    client.chat.completions.create(
        model="ChatGPT_MODEL",
        messages=[
            {"role": "system",
            "content":
            "You are an experienced
             static code security
             analyzer."},
            {"role": "user",
            "content": message}
        ],
        max_tokens=MAX_TOKENS,
        temperature=0
    )
    evaluation_result =
    response.choices[0].message.content
except Exception as e:
    evaluation_result =
    f"[API error: {e}]"

# Save response
with open(OUTPUT, "w",
    encoding='utf-8') as f:
    f.write(evaluation_result)
```

On ChatGPT web application, the model 4o is used. To make it understand its job, following messages were first sent to ChatGPT.

> *Now you are gonna play the role as static code analyzer, specifically for JavaScript code in Chrome extensions. I will send you a csv file containing the code and the filename. You need to carefully evaluate them(each code snippet) as "dangerous" or "safe". "dangerous" means the code is malicious, or it can be abused by malicious websites so it leaks sensitive information of the user. "safe" means the code is completely safe for the users. Give me the results in a list.*

V. Results

After running the automation script, 465 snippets were selected and stored in a csv file. All code snippets include the full function call of its corresponding function, however not the functions that bound them. Moreover, some of the parameters were only displayed as variable name, instead of the actual value/content of the parameters.

The first version the returned ChatGPT evaluation is unsatisfied. 158 out of 465 code snippets were evaluated as "dangerous". In detail, 113 out of 158 code snippets come from the malicious dataset. Compared to Codex, our result contains many more false positives, due to the fact that ChatGPT considers all function calls that involve functions that can be used to execute malicious operations or abused by malicious websites as dangerous code, without taking parameters into consideration.

Therefore, we adjust the prompt of ChatGPT so it takes parameters into account. This largely reduces the number of false positives, decreasing the total number of dangerous operations to 34, however produces a large number of false negatives. By exploring and comparing with the results from CodeX, we observed that the cause is the lack of context. For example, a call of ***chrome.bookmark.create*** itself might seem safe, but a ***fetch*** is called in way that our program does not include it in the code snippet. Since no actual url is extracted in the code snippet, ChatGPT would evaluate both calls as safe. To produce a more accurate result, we adjust the prompt so ChatGPT can also mark code snippets as "suspicious" when they show no evidence to be dangerous but can be harmful in certain cases.

Finally, following answer from ChatGPT is obtained:

Dataset	**Dangerous**	**Suspicious**
Malicious	18	37
Safe	12	8

VI. Discussion and Evaluation

A. Accuracy

To evaluate the accuracy of ChatGPT's evaluation, we compared the results with CodeX's results. It is observed that our results include many false positives, i.e. many cases of harmless code are classified as "dangerous" or "suspicious". This might be due to the fact that the authors of CodeX only provided one type of malicious action for each extension, while our program attempts to find all types of malicious actions in all extensions. For instance, in the extension that contains bookmark stealing code, the authors of CodeX only provided the results of bookmark stealing, however cookie stealing action was also detected by our program. The selected code snippet is:

```
chrome.cookies.get({url:
"https://shoppingcart.aliexpress.com/",
name:"xman_us_f"},
(e=>{i(e.value)}))
```

And it might steal sensitive data stored in by cookie if it is improperly used.

Our program also ignored some malicious operations due to the design flaws in CodeQL queries. For example, our query to find chrome functions can only find functions with format ***chrome.xxx.xxx***. This works for the most time, however it is recognized later that some malicious operations, such as ***chrome.storage.sync.set()*** are skipped.

Furthermore, the code extraction of our program sometimes ignores the context, for example, only the part before the arrow would be extracted in the case of ***function() => context***. This also largely affects the accuracy of the ChatGPT evaluation.

B. Improvements

First, the CodeQL queries should be improved, so it can find all potentially dangerous functions. This improvement is relatively simple, by focusing on the main classes can fix the problem, e.g. find all operations that starts with ***chrome.history*** instead of finding all operations that has the format of ***chrome.history.xxx***.

The second improvement is to find and extract the context and the variable value of the extracted code snippets. This requires more investigation in CodeQL, and may need to introduce dynamic analysis.

Lastly, the code evaluation of ChatGPT can also be improved by feeding it with a suitable number of examples. This improvement requires time to find related examples and to classify them into datasets.

Appendix

Contributions

The project was divided into several smaller parts to evenly distribute the workload. First, both Stella and Carl did the literature study and wrote the proposal together. Then Stella worked with CodeQL queries and Carl in the meantime worked with finding appropriate data sets. After that, Carl and Stella tested and improved the CodeQL queries together. In the second phase, Carl developed the automation script while Stella began to write the first sections of the report. Then Stella joined Carl and developed the automation script together, specifically in solving the problem of code extraction. After that, Stella and Carl adjusted ChatGPT and fetched the results together. Lastly, Carl and Stella continued working on the report until it is finished.

References

[1] O. Starov and N. Nikiforakis, “Xhound: Quantifying the fingerprintability of browser extensions,” in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017. doi: 10.1109/SP.2017.18 pp. 941–956.

[2] D. F. Somé, “Empoweb: empowering web applications with browser extensions,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 227–245.

[3] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, “Hulk: Eliciting malicious behavior in browser extensions,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 641–654.

[4] A. Nayak, R. Khandelwal, E. Fernandes, and K. Fawaz, “Experimental security analysis of sensitive data access by browser extensions,” in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 1283–1294.

[5] M. M. Ahmadpanah, M. F. Gobbi, D. Hedin, J. Kinder, and A. Sabelfeld, “Codex: Contextual flow tracking for browser extensions,” 2025.

[6] CodeQL, 2025. [Online]. Available: https://codeql.github.com/

[7] H. Li, Y. Hao, Y. Zhai, and Z. Qian, “Assisting static analysis with large language models: A chatgpt experiment,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2107–2111.

[8] M. M. Mohajer, R. Aleithan, N. S. Harzevili, M. Wei, A. B. Belle, H. V. Pham, and S. Wang, “Effectiveness of chatgpt for static analysis: How far are we?” in *Proceedings of the 1st ACM International Conference on AI-Powered Software*, 2024, pp. 151–160.

[9] G. Chrome, 2025. [Online]. Available: https://github.com/GoogleChrome/chrome-extensions-samples