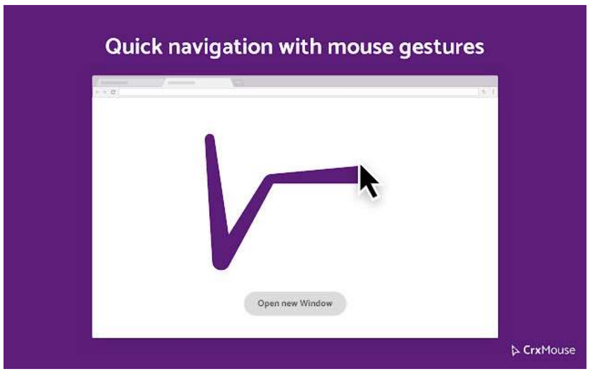
1. Introduction

Browser extensions provide additional functionality and customization for browsers. However, many previous studies show that browser extensions are often granted more privileges than necessary. Extensions can directly threaten the host system as well as web applications or bring in indirect threats to web sessions by injecting content into web pages. This project presents an empirical study of Firefox browser extensions and introduces a static analysis tool that can detect potential security vulnerabilities within extensions.

2. Case Study

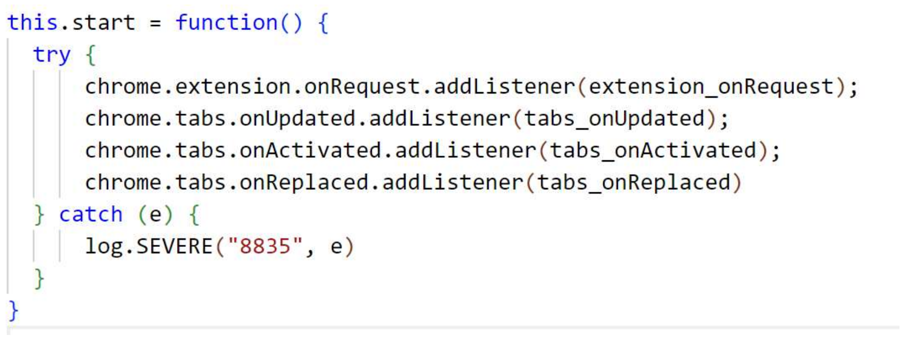
To understand the browser and extension better, and to know what kind of attack could happen, we look at browser extension security breaches that have occurred.

2.1. Gestures for Chrome



Gestures for Chrome is a popular browser extension that allows users to execute various commands and actions using mouse gestures. With Gestures, users can easily navigate through web pages, control media playback, close tabs, and perform other common tasks with just a click of the mouse.

During initialization, the extension sends the request to the s808 server, while "tabs_onUpdated", "tabs_onActivated" and "tabs_onReplaced" send requests to the s1808 server. The specific code is shown below:

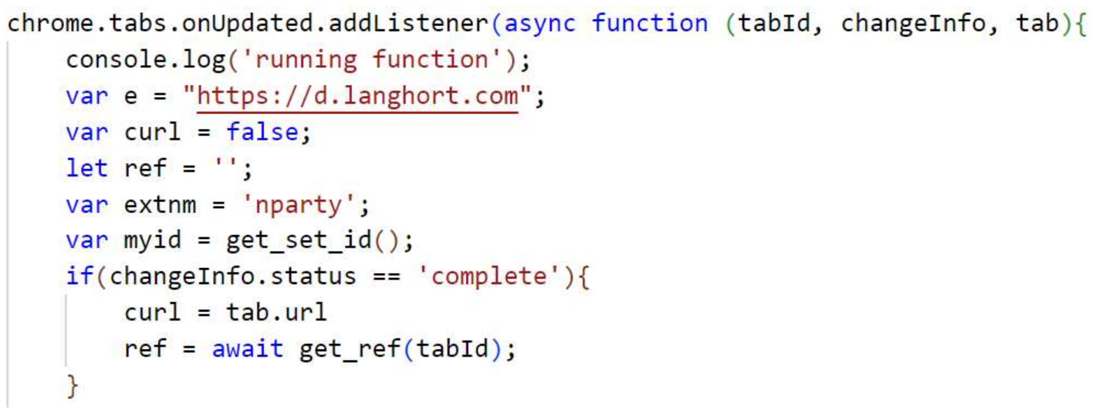


```
this.start = function() {
  try {
      chrome.extension.onRequest.addListener(extension_onRequest);
      chrome.tabs.onUpdated.addListener(tabs_onUpdated);
      chrome.tabs.onActivated.addListener(tabs_onActivated);
      chrome.tabs.onReplaced.addListener(tabs_onReplaced)
  } catch (e) {
      log.SEVERE("8835", e)
  }
}
```

The fact that user information is being sent to an unknown server is very concerning. The current URL and referrer of the browser undergo double base64 encoding, making it difficult for some ordinary users to detect. This inspires us to be mindful of extensions with base64 encoded string in their JavaScript code, as they may pose a risk of sending user information to unknown servers.

2.2. Netflix Party

Netflix Party enables users to watch Netflix shows together, but it also tracks their browsing activity by sending every visited website to the extension creator's servers. This allows the creators to insert code into eCommerce websites, modify cookies, and receive affiliate payments for any items purchased.



```
chrome.tabs.onUpdated.addListener(async function (tabId, changeInfo, tab){
    console.log('running function');
    var e = "https://d.langhort.com";
    var curl = false;
    let ref = '';
    var extnm = 'nparty';
    var myid = get_set_id();
    if(changeInfo.status == 'complete'){
        curl = tab.url
        ref = await get_ref(tabId);
    }
```

The figure above shows the code from B0.js, which is contained in Netflix Party XPI. Once this event triggers, the extension will set a variable called curl with the URL of the tab by using the tab.url variable. It creates several other variables which are then sent to d.langhort.com. Upon receiving the URL, langhort.com will check if it matches a list of websites that it has an affiliate ID for, and if it does, it will respond to the query. Then the extension would insert the URL received from the server as an iframe on the website being visited.

We can see that the harmful behavior of Netflix Party is achieved through interaction with “d.langhort.com”. Therefore, it is important to check for suspicious URLs in various code files (JavaScript, HTML) in the extension.

3. Design and Implementation

Based on the two case studies, it is evident that to determine the potential maliciousness of an extension, we need to analyze the manifest.json, JavaScript and HTML file extracted from *.XPI file.

To develop a tool for static analysis of extensions, we need to collect a sufficient number of Firefox extensions and then utilize the data to build and test the tool.

3.1. Extension Collection

We first download more than 8000 browser extensions from https://addons.mozilla.org. We crawl the extensions download using BeautifulSoup4.

BeautifulSoup3 is currently out of service, so we use BeautifulSoup4. BeautifulSoup4 is a python library that primarily crawls data from web pages, from HTML or XML files. It provides simple, Python-like functions to handle navigation, searching, modifying analysis trees, and so on. It is a toolbox that parses documents to provide users with data to grab. BeautifulSoup4 automatically converts the input document to Unicode encoding and the output document to utf-8 encoding.

Usage of bs4:

1) Import module

```
from bs4 import BeautifulSoup
```

2) Construct a Request object

```
html = "https://addons.mozilla.org/en-US/firefox/extensions/category/appearance/?page={}".format(each)
wb_data = requests.get(html)
soup = BeautifulSoup(wb_data.text, 'lxml')
```

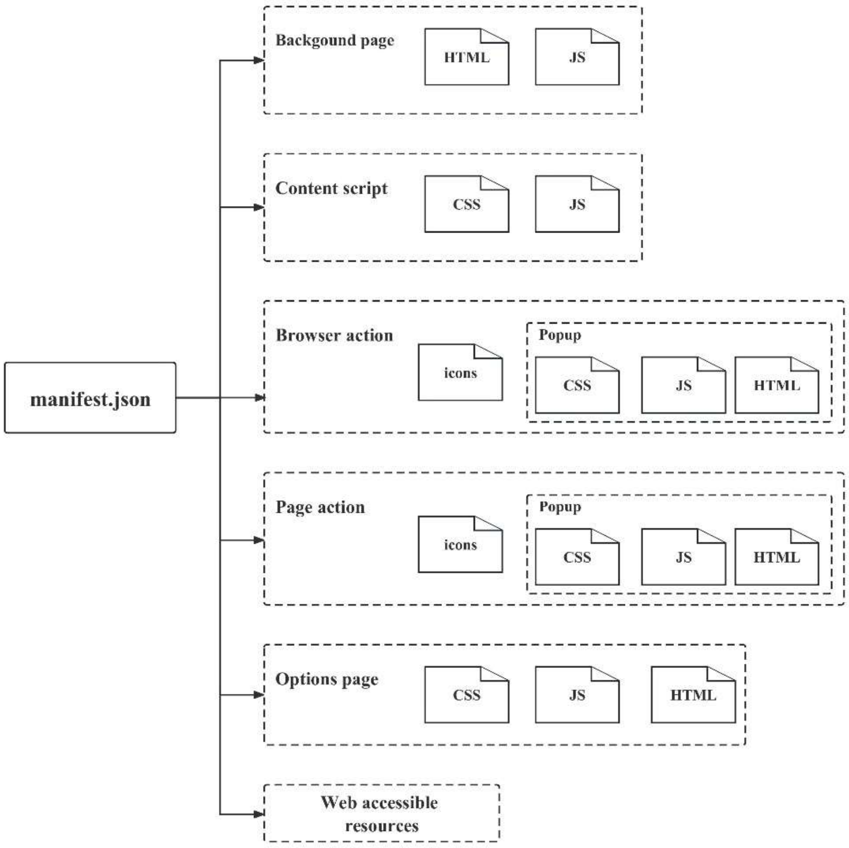
3) Find the corresponding tag using regular expressions

```
for tag in soup.find_all(href = re.compile(\
    "^/en-US/firefox/addon/[a-zA-Z0-9_-]+/\\?utm_source=\
        addons\\.mozilla\\.org&utm_medium=referral&utm_content=search$")):
```

4) Download extensions using the urllib library

```
for innertag in innersoup.find_all(href = re.compile\
                                   ("^https://addons\.mozilla\.org/firefox/downloads/file/.+\.xpi$")):
# Look for the downloaded url in the extension details page and use regular expression
            filename = innertag['href'].split('/')[-1]
            urllib.request.urlretrieve(innertag['href'], local_file_path + "\\" + filename)
```

3.2. Implementation

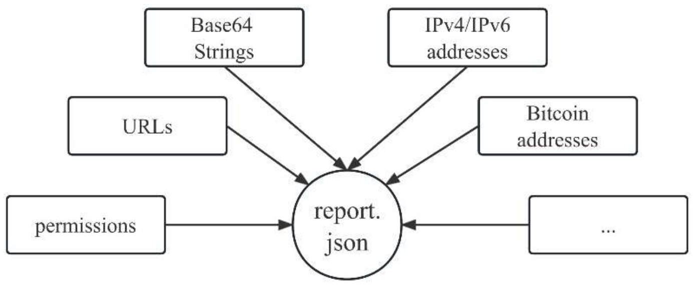


Our goal is to create a tool that analyzes extensions and produces an "analysis report" to assist users in deciding whether to install the extension on their browsers.

To achieve this goal, we need to comprehend the components of extensions and examine each of them. The structure of Firefox extensions is illustrated in the above figure.

The manifest.json file is the only file that presented in every extension, which contains basic metadata such as name, version, required permissions, and pointers to other files within the extension. Therefore manifest.json is the main focus of our analysis.

In addition to checking the manifest.json file, we also need to examine the JavaScript and HTML files in an extension. We specifically look for URLs, email addresses, Bitcoin addresses, IPv4/IPv6 addresses and base64 strings. If an extension contains any of these items in its code, it doesn't necessarily mean it's malicious, but it could raise a "red flag". We will list these URLs and addresses in the "analysis report".



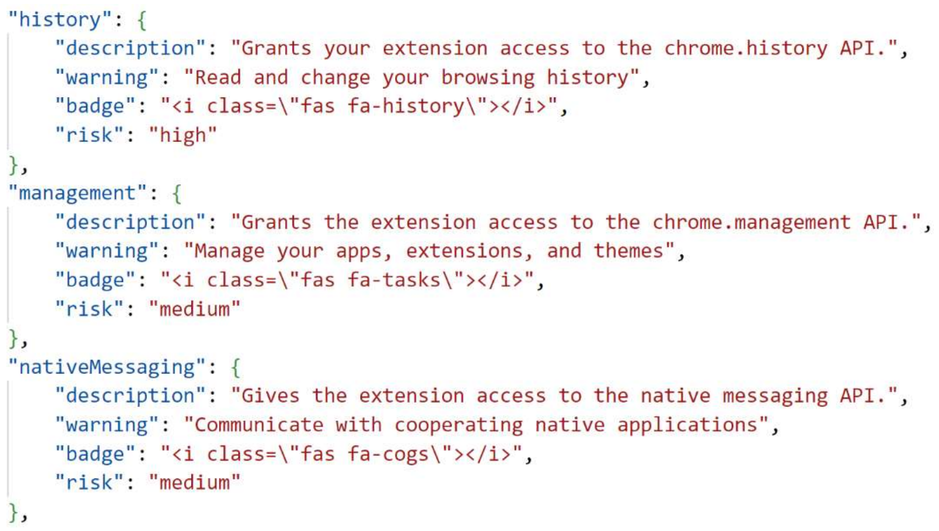
Our analysis report is a JSON file that contains various attributes, as shown in the figure above. Users can input a XPI file, retrieve the corresponding JSON file, and examine its contents. We do not provide a direct risk level assessment for a whole extension because it is not accurate to determine the nature of an extension based on a few indices only. However, we include the risk level of the permissions requested by an extension, all addresses, URLs, and base64-encoded strings found in its source code. Users can make their own judgments based on the report we provide. For example, if a user reviews our report.json and discovers that an extension will communicate with a domain they don't trust, they may decide not to install it.

Item	Description
name	Name of the extension
source	Source of extension. In our analysis, the value of 'source' is always 'Local'
type	Type of extension. In our analysis, the value of 'extracted' is always 'Local Firefox Extension'
manifest	The analysis result of 'manifest.json' within the XPI, including manifest version, background script, permissions, etc.
version	Version of extension
permissions	The requested permissions of the extension (same as the permissions in manifest)
urls	All URLs appeared in the JavaScript, HTML, or manifest.json files
emails	Email address appeared in the JavaScript, HTML, or manifest.json files
bitcoin_address	Bitcoin address appeared in the JavaScript, HTML, or manifest.json files
ipv4_address	Ipv4 address appeared in the JavaScript, HTML, or manifest.json files
ipv6_address	Ipv6_address appeared in the JavaScript, HTML, or manifest.json files
base64_string	Base64 encoded string appeared in the JavaScript, HTML, or manifest.json files.
comments	Comments appeared in the JavaScript, HTML, or manifest.json files.
domains	Domains appeared in the JavaScript, HTML, or manifest.json files.
files	Names of all files in the extension XPI
author	Author of the extension
description	Description of the extension

Above table displays the contents of "report.json", where "permissions" is a crucial item. Extensions can ask for permissions such as access to browsing history, cookies, and tabs. The table below shows some common permissions and their descriptions.

Permission	Description
activeTab	Allow the user to temporarily access the current active TAB while invoking the extension.
history	Permissions for browser history operations.
storage	Permissions for browser history operations.
cookies	Cookie query, modification, onChange monitoring
tabs	TAB permissions allow you to create, modify, and rearrange tabs.
browsingData	Permission to operate browser data, mainly used to clear browser data cookie storage.

To assess the risk level of permission requests made by an extension, we formulate a permissions.json file. This file defines the "risk level" for each type of permission request. Partial codes are shown below:

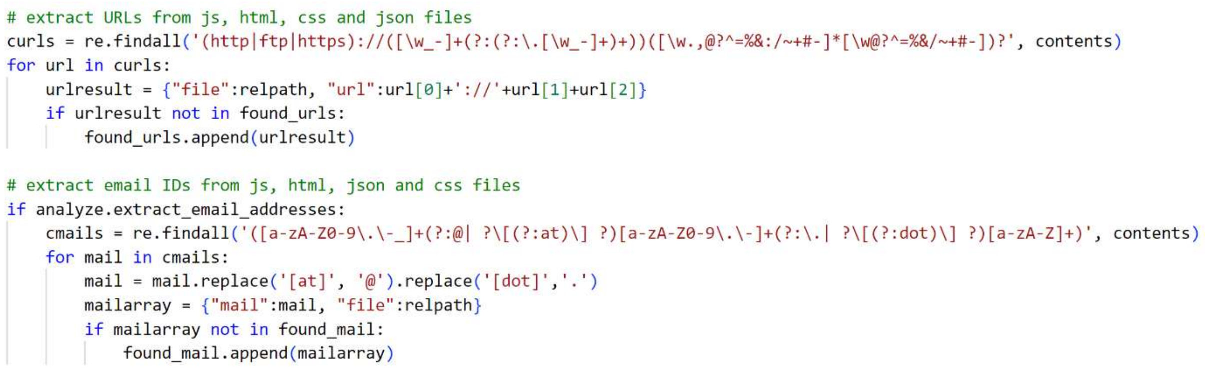


```
"history": {
    "description": "Grants your extension access to the chrome.history API.",
    "warning": "Read and change your browsing history",
    "badge": "<i class=\"fas fa-history\"></i>",
    "risk": "high"
},
"management": {
    "description": "Grants the extension access to the chrome.management API.",
    "warning": "Manage your apps, extensions, and themes",
    "badge": "<i class=\"fas fa-tasks\"></i>",
    "risk": "medium"
},
"nativeMessaging": {
    "description": "Gives the extension access to the native messaging API.",
    "warning": "Communicate with cooperating native applications",
    "badge": "<i class=\"fas fa-cogs\"></i>",
    "risk": "medium"
},
```

From the figure, we can see that the "history" permission is at high risk, while the "management" permission is at medium risk.

Referring to the permissions.json file, we can get the risk level of permission requests in manifest.json and display them in our report.

As for other attributes in report.json, such as URLs, domains, strings and addresses, we utilize regular expressions to scan all JavaScript and HTML files in the XPI file. We then add the search outcome to our report. Partial codes are shown below:

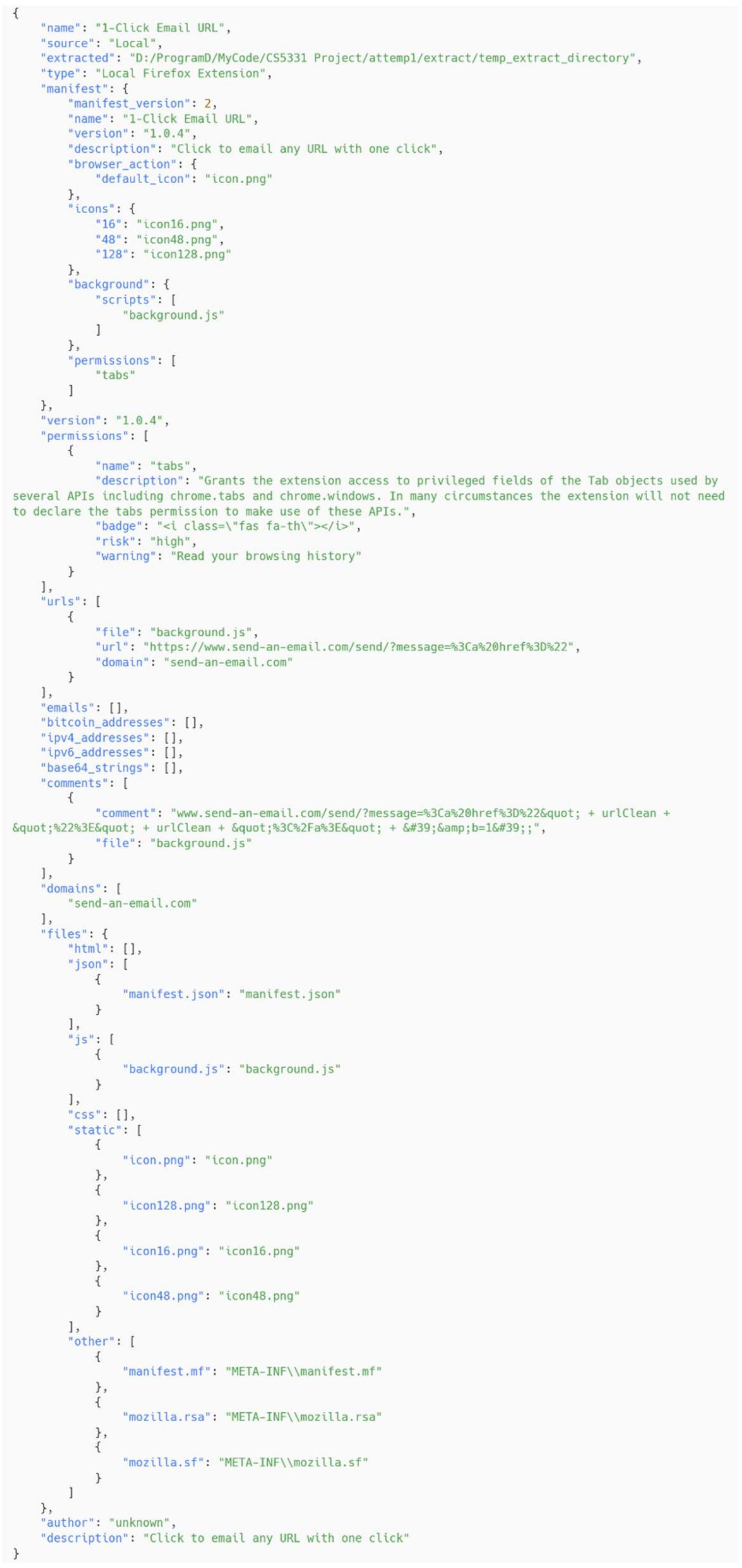


```python
# extract URLs from js, html, css and json files
curls = re.findall('(http|ftp|https)://([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:/~+#-]*[\w@?^=%&/~+#-])?', contents)
for url in curls:
    urlresult = {"file":relpath, "url":url[0]+'://'+url[1]+url[2]}
    if urlresult not in found_urls:
        found_urls.append(urlresult)

# extract email IDs from js, html, json and css files
if analyze.extract_email_addresses:
    cmails = re.findall('([a-zA-Z0-9\.\-_]+(?:@| ?\[(?:at)\] ?)[a-zA-Z0-9\.\-]+(?:\.| ?\[(?:dot)\] ?)[a-zA-Z]+)', contents)
    for mail in cmails:
        mail = mail.replace('[at]', '@').replace('[dot]','.')
        mailarray = {"mail":mail, "file":relpath}
        if mailarray not in found_mail:
            found_mail.append(mailarray)
```

4. Results and Insights

We utilize the tool developed in section 3 to analyze all the extensions we collected in section 2. Each extension has a result in the form of JSON as shown below.



```json
{
    "name": "1-Click Email URL",
    "source": "Local",
    "extracted": "D:/ProgramD/MyCode/CS5331 Project/attemp1/extract/temp_extract_directory",
    "type": "Local Firefox Extension",
    "manifest": {
        "manifest_version": 2,
        "name": "1-Click Email URL",
        "version": "1.0.4",
        "description": "Click to email any URL with one click",
        "browser_action": {
            "default_icon": "icon.png"
        },
        "icons": {
            "16": "icon16.png",
            "48": "icon48.png",
            "128": "icon128.png"
        },
        "background": {
            "scripts": [
                "background.js"
            ]
        },
        "permissions": [
            "tabs"
        ]
    },
    "version": "1.0.4",
    "permissions": [
        {
            "name": "tabs",
            "description": "Grants the extension access to privileged fields of the Tab objects used by
several APIs including chrome.tabs and chrome.windows. In many circumstances the extension will not need
to declare the tabs permission to make use of these APIs.",
            "badge": "<i class=\"fas fa-th\"></i>",
            "risk": "high",
            "warning": "Read your browsing history"
        }
    ],
    "urls": [
        {
            "file": "background.js",
            "url": "https://www.send-an-email.com/send/?message=%3Ca%20href%3D%22",
            "domain": "send-an-email.com"
        }
    ],
    "emails": [],
    "bitcoin_addresses": [],
    "ipv4_addresses": [],
    "ipv6_addresses": [],
    "base64_strings": [],
    "comments": [
        {
            "comment": "www.send-an-email.com/send/?message=%3Ca%20href%3D%22&quot; + urlClean +
&quot;%22%3E&quot; + urlClean + &quot;%3C%2Fa%3E&quot; + &#39;&amp;b=1&#39;;",
            "file": "background.js"
        }
    ],
    "domains": [
        "send-an-email.com"
    ],
    "files": {
        "html": [],
        "json": [
            {
                "manifest.json": "manifest.json"
            }
        ],
        "js": [
            {
                "background.js": "background.js"
            }
        ],
        "css": [],
        "static": [
            {
                "icon.png": "icon.png"
            },
            {
                "icon128.png": "icon128.png"
            },
            {
                "icon16.png": "icon16.png"
            },
            {
                "icon48.png": "icon48.png"
            }
        ],
        "other": [
            {
                "manifest.mf": "META-INF\\manifest.mf"
            },
            {
                "mozilla.rsa": "META-INF\\mozilla.rsa"
            },
            {
                "mozilla.sf": "META-INF\\mozilla.sf"
            }
        ]
    },
    "author": "unknown",
    "description": "Click to email any URL with one click"
}
```

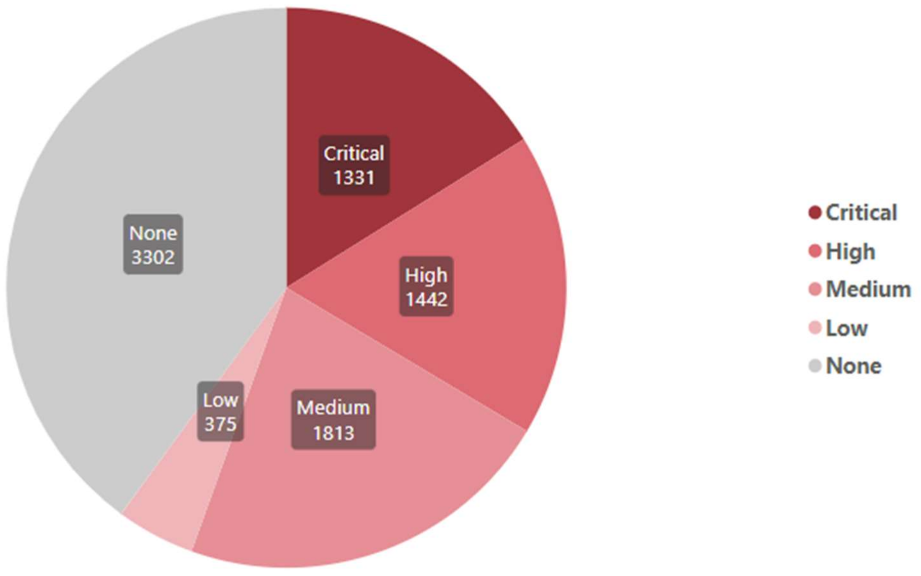
Next, we analyze the results obtained and derive insights from them.

4.1. Permissions

Each report.json file contains a “permissions” attribute, which may contain more than one permission requested by the extension. Each permission matches a risk level (critical, high, medium, low, or none).

To assess the overall risk level of an extension’s permission requests, we define it as the highest risk level among all the permissions requests of this extension based on the Wooden Bucket Theory.

We evaluate the overall risk level of all the extensions we collected, and the visualization is shown below. Of the 8,263 extensions, a total of 1,331 ask for permissions that were considered critical, 1,442 High, 1,813 Medium, 375 Low and only 3,302 None, which only accounts for 40 percent.



In response to this result, we think that many extensions apply for access to all URLs, request access to the browser's browsing history, and even request access to system files on the user's computer to provide convenient functionality, which provides a great user experience. However, from the perspective of security, these extensions "intrude" on users step by step, potentially stealing users' private information, which will bring great security risks.

4.2. URLs, Strings and Addresses

The report.json file shows whether there are any URLs, base64 encoded strings, or addresses in the extension's source code. However, the mere presence of these elements does not necessarily mean that an extension is malicious. But still, these URLs and addresses can provide valuable information on the type of attack that the extension may potentially carry out.

- URLs and domains: URLs and domains indicate that the extension could be communicating with a third-party server or website, sending user data to that party, and receiving commands that could potentially harm the user.
- Base64 encoded string: Base64 encoded string could be a sign of malicious behavior. Malware authors often use base64 encoding to hide malicious code or data, making it difficult for antivirus tools to detect.
- Email address: Email addresses can be used in phishing attacks, also there is possibility that the extension is sending sensitive information to remote server via email.
- Bitcoin address: Bitcoin address could be an indication of the extension being involved in cryptocurrency-related attacks. Attacker could use the user’s system to mine coins or try to steal coins from user’s account.
- IPv4/IPv6 address: IPv4 and IPv6 addresses are used to identify devices on the internet and enable communication between them. An extension that contains IPv4/IPv6 addresses could potentially connect to a malicious server without the user’s acknowledgement.

4.3. Section Summary

As previously mentioned, an extension may request high-risk permissions, include URLs, domains, and external addresses, and still be secure and benign. Our aim is to provide a brief overview of each extension and assess its potential security vulnerabilities. This will help users to know the extension better before they install it.

5. Conclusion

This project conducts an empirical study of Firefox browser extensions and introduces a static analysis tool that can detect potential security vulnerabilities in extensions. We first download over 8000 extensions from “addons.mozilla” using BeautifulSoup4. Then we extract the XPI file to get the codes in it, including manifest.json, JS and HTML files. Finally, we analyze these files using regular expressions and generate the report.json which provides users with extracted information about the extension.

From a security standpoint, we expect browser extensions to be pristine, devoid of any permission requests, external links, external addresses, or encoded strings. However, in terms of practicality and convenience, browser extensions will inevitably access personal data and communicate with external servers.

Although browser extensions may contain external sources or request permissions for legitimate purposes, it is important to be aware of the potential security risks associated with these actions.

In the future, we may consider analyzing the source code of an extension in more detail, providing more reliable security reports for users.