

**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCE**

**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION**

**BSc THESIS**

**Malicious Chrome Extensions**

**ILIAS A. RAFAIL**

|  |  |
| --- | --- |
| **Supervisor (or supervisors):** | **Name Surname, Title** |

**ATHENS**

**MONTH YEAR**



**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

**Κακόβουλες Επεκτάσεις του Chrome**

**Ηλίας Α. Ραφαήλ**

|  |  |
| --- | --- |
| **Επιβλέπων (ή Επιβλέπουσα ή Επιβλέποντες):** | **Όνομα Επώνυμο,** Τίτλος (π.χ Αναπληρωτής Καθηγητής) |

**ΑΘΗΝΑ**

**ΜΗΝΑΣ ΕΤΟΣ**

**BSc THESIS**

Malicious Chrome Extensions

**ILIAS A. RAFAIL**

**S.N.:** 1115201300149

|  |  |
| --- | --- |
| **SUPERVISOR:** | **Name Surname, Title** |

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

Κακόβουλες Επεκτάσεις του Chrome

**Ηλίας Α. Ραφαήλ**

**Α.Μ.:** 1115201300149

|  |  |
| --- | --- |
| **ΕΠΙΒΛΕΠΟΝΤΕΣ:** | **Όνομα Επώνυμο,** Τίτλος (π.χ Αναπληρωτής Καθηγητής) |

**ABSTRACT**

Today the use of extensions is widespread for users of all major web browsers such as Chrome. There is also a large increase in the number of personal data that search engines mediate, and as a result they become the target of criminals who, taking advantage of how extensions work, can gather various information from the user without this being perceived.

The goal of this thesis is to implement our own extension that a user can use to get  
informed about suspicious actions on extensions he wants to install from the Chrome webstore.

The way the extension works is initially connects to a Native Messaging Host, which sends information about the identity of the extension that the user has chosen to the application. The application will download the extension's source code and perform a static analysis of the code. Every time a suspicious action is detected, it will inform the user with a corresponding warning message and the file in which the action was detected.

**SUBJECT AREA**: Static analysis of Chrome extension code

**KEYWORDS**: π.χ. malicious/suspicious chrome extensions, chrome webstore, static analysis, native messaging host.

**ΠΕΡΙΛΗΨΗ**

Σήμερα η χρήση επεκτάσεων είναι διαδεδομένη στους χρήστες όλων των μεγάλων  
μηχανών αναζήτησης όπως είναι ο Chrome. Παρατηρείται επίσης μεγάλη αύξηση του πλήθους των προσωπικών δεδομένων τα οποία διατηρούν οι μηχανές αναζήτησης και ως αποτέλεσμα γίνονται στόχος εγκληματιών οι οποίοι εκμεταλλευόμενοι τον τρόπο λειτουργίας των επεκτάσεων μπορούν να συγκεντρώσουν διάφορες πληροφορίες από τον χρήστη χωρίς αυτό να γίνει αντιληπτό.

Στόχος της πτυχιακής εργασίας είναι να δημιουργήσουμε μια δική μας επέκταση την οποία θα μπορεί να χρησιμοποιήσει ο χρήστης για να ενημερωθεί αν υπάρχουν ύποπτες ενέργειες σε επεκτάσεις που θέλει να εγκαταστήσει από το webstore του Chrome.

Ο τρόπος με τον οποίο λειτουργεί η επέκταση είναι αρχικά να συνδέεται με έναν Native Messaging Host, ο οποίος στέλνει την πληροφορία για τη ταυτότητα της επέκτασης που έχει επιλέξει ο χρήστης στην εφαρμογή. Η εφαρμογή με τη σειρά της θα κατεβάσει τον πηγαίο κώδικα της επέκτασης και θα εκτελέσει στατική ανάλυση πάνω στον κώδικα αυτό. Κάθε φορά που εντοπιστεί κάποια ύποπτη ενέργεια θα ενημερώσει τον χρήστη με αντίστοιχο προειδοποιητικό μήνυμα αλλά και το αρχείο στο οποίο εντοπίστηκε η ενέργεια.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ**: Στατική ανάλυση στον κώδικα επεκτάσεων του Chrome

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**: κακόβουλες/ύποπτες επεκτάσεις του chrome, chrome webstore, στατική ανάλυση, native messaging host.

*Στη σελίδα αυτή αναφέρονται οι αφιερώσεις. Η σελίδα αυτή είναι προαιρετική.*

**ΕΥΧΑΡΙΣΤΙΕΣ (ή AKNOWLEDGMENTS)**

Στη σελίδα αυτή αναφέρονται οι ευχαριστίες. Η σελίδα αυτή είναι προαιρετική. Παρατίθεται παράδειγμα ευχαριστιών.

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα θέλαμε να ευχαριστήσουμε τους επιβλέποντες, αν. καθ .Ευστράτιο Γεωργιάδη, Γρηγόριο Παπάμαλο, Αναστασία Γούσιου, Ξενοφών Παπαδόπουλο, για τη συνεργασία και την πολύτιμη συμβολή του στην ολοκλήρωση της.

**CONTENTS**

[PREFACE 13](#_Toc514844758)

[1. INTRODUCTION 14](#_Toc514844759)

[1.1 Chrome Extension Composition 14](#_Toc514844760)

[1.2 Installing Extensions 14](#_Toc514844761)

[1.3 Access Control Settings 15](#_Toc514844762)

[1.3.1 Permissions 15](#_Toc514844763)

[1.3.2 Content Scripts 16](#_Toc514844764)

[1.3.3 Background Pages 16](#_Toc514844765)

[1.3.4 Content Security Policy 16](#_Toc514844766)

[2. RELATED WORK 17](#_Toc514844767)

[2.1 Hulk Architecture 17](#_Toc514844768)

[2.1.1 HoneyPages 17](#_Toc514844769)

[2.1.2 Fuzzer 17](#_Toc514844770)

[2.1.3 Malicious Behavior Detection 18](#_Toc514844771)

[2.2 Chroak Extension 18](#_Toc514844772)

[2.2.1 Information Gathered 19](#_Toc514844773)

[2.2.2 Attacks Performed 19](#_Toc514844774)

[3. DESIGN 21](#_Toc514844775)

[3.1 Architecture 21](#_Toc514844776)

[3.2 Native Messaging Host 21](#_Toc514844777)

[3.3 Parsers 21](#_Toc514844778)

[3.4 Rules 21](#_Toc514844779)

[4. IMPLEMENTAYION DETAILS 23](#_Toc514844780)

[4.1 Rule 1: Denying the access to Chrome’s extensions configuration page. 23](#_Toc514844781)

[4.2 Rule 2: DoS. 23](#_Toc514844782)

[4.3 Rule 3: Extension installation/uninstallation. 24](#_Toc514844783)

[4.4 Rule 4: HTTP response header security options. 24](#_Toc514844784)

[4.5 Rule 5: URL redirection. 25](#_Toc514844785)

[4.6 Rule 6: Block access to websites. 25](#_Toc514844786)

[4.7 Rule 7: Form submit requests. 26](#_Toc514844787)

[4.8 Rule 8: User’s CPU information. 26](#_Toc514844788)

[4.9 Rule 9: User’s number of displays. 27](#_Toc514844789)

[4.10 Rule 10: User’s sessions information. 27](#_Toc514844790)

[5. EVALUATION 28](#_Toc514844791)

[6. CONCLUSION 29](#_Toc514844792)

[REFERENCES 30](#_Toc514844793)

**LIST OF FIGURES**

[Figure 1: manifest.json file. 14](file:///C:\Users\ilias\Desktop\asd\report\Report.docx#_Toc514844215)

[Figure 2: Example of manifest containing API permissions, content scripts, background script and CSP. 15](#_Toc514844216)

[Figure 3: Access to Chrome’s extensions configuration page. 23](#_Toc514844217)

[Figure 4: DoS. 23](#_Toc514844218)

[Figure 5: Extension installation/uninstallation. 24](#_Toc514844219)

[Figure 6: HTTP response header security options. 24](#_Toc514844220)

[Figure 7: URL redirection. 25](#_Toc514844221)

[Figure 8: Access to websites blocked. 25](#_Toc514844222)

[Figure 9: Information from submit requests. 26](#_Toc514844223)

[Figure 10: User’s CPU information. 26](#_Toc514844224)

[Figure 11: User’s number of displays. 27](#_Toc514844225)

[Figure 12: User’s sessions information. 27](#_Toc514844226)

**LIST OF TABLES**

[Table 1: All available extension permissions. 16](#_Toc514844248)

[Table 2: “Non-special” extension permissions. 18](#_Toc514844249)

PREFACE

This thesis took place in Athens of Greece between… It is consisted by three parts, Reading, Implementation and Writing. For the first part, I read some papers in order to familiarize with related work and gather ideas for the subject. The implementation phase was about creating the extension and the application that would perform the analysis. I started creating the part that downloads the extension files (.crx file) the user has chosen and extract them. Then, I created two parsers, one for HTML files and the other for JavaScript, and I parsed each file of the extension. I used the parsing tree to check for code that matches with rules that indicate suspicious activity. In case of a match, I print a warning message that will inform the user about the suspicious action. The application is written in Python and can be further expanded by adding more rules. The final part is about the text which presents the way I worked on the project.

# INTRODUCTION

## Chrome Extension Composition

Google Chrome supports extensions written in JavaScript and HTML. Each extension also contains a mandatory manifest (.json file) that defines a set of properties such as the extension name, description and version number. The manifest is used by the browser to know the functionality offered by the extension and the list of permissions required to access the different parts of the extension API.

In order to be listed on the Chrome Web Store, an extension must have a manifest\_version of at least 2, which by default limits the source of scripts and objects used by the extension to “self”. That means functions like eval are disabled, inline JavaScript will not be executed and only local script an object resources can be loaded. In order to relax these limitations, an extension must specify the changes to the policy in the manifest file. The changes will warn users downloading the extension that modifications have been made and that the extension may be vulnerable to attacks.



Figure 1: manifest.json file.

## Installing Extensions

The Chrome Web Store is the official means for users to find and install extensions. But in addition to this, extensions can also be installed manually by the user or an external program. Installation of extensions outside the web store is referred as sideloading. In 2014 Chrome took steps to prevent sideloading by requiring all installed extensions to be hosted in the Chrome Web Store. However, it is still possible for programs to force silent installation of extensions, since the attacker already has control of the machine.

## Access Control Settings

Access control settings is a protection mechanism currently used by all browsers (based on Chromium, Firefox and Microsoft Edge). Browsers rely on a set of configuration options included in a manifest file.



Figure 2: Example of manifest containing API permissions,  
content scripts, background script and CSP.

### Permissions

The permission system is designed in the spirit of least privilege, with the goal of limiting the resources available to an extension. It determines which sites an extension can access, the allowed API call, and the use of binary plugins.

In Figure 2 example, the extension requests host permission for https://www.google.com/, which allows it to access cookies and webRequest APIs for the specified domains. Wildcarding can also be used where the extension requests access to \*://\*.facebook.com. This permission allows access to all subdomains of facebook.com via any URL scheme. Additionally, <all\_urls> is a special token used for matching any URL.

Table 1 shows all available permissions.

Table 1: All available extension permissions.

|  |  |  |  |
| --- | --- | --- | --- |
| activeTab | declarativeContent | notifications | system.storage |
| alarms | desctopCapture | pageCapture | tabCapture |
| background | downloads | power | tabs |
| bookmarks | fontSettings | printerProvider | topSites |
| browsingData | gcm | privacy | tts |
| clipboardRead | geolocation | proxy | ttsEngine |
| clipboardWrite | history | sessions | unlimitedStorage |
| contentSettings | identity | storage | webNavigation |
| contextMenus | idle | system.cpu | web.request |
| cookies | management | system.display | webRequestBlocking |
| debugger | nativeMessaging | system.memory |  |

### Content Scripts

Content scripts is a list that indicates JavaScript files that will run inside of the web page. The execution of a content script not only can modify the DOM tree of other scripts, but it can also issue authenticated web requests like POST.

In Figure 2, we can see 2 JavaScript files included, that will be run in the context of the page for any URLs matching the http://www.yahoo.com/ pattern.

### Background Pages

Background pages often contain the logic and state an extension needs for the entirety of the browser session and do not have any visibility to the user.

Figure 2 shows how background.js is specified as a background page.

### Content Security Policy

A Content Security Policy (CSP) header is specified by servers and used by the browser in order to determine the sources from which it can include objects of the page. CSP can also specify other options, such as whether to allow the page to perform an eval or to embed inline JavaScript.

Figure 2 shows an example where the extension specifies its CSP in order to include source from foo.com and execute eval.

# RELATED WORK

Modern web browsers are characterized by third-party add-ons that extend their functionality. Chrome browser provide a rich API where extensions can make network requests, access the local file system, get low-level information about running processes and many more. Although a permission system is used by Chrome in order to curtail an extension’s privileges and avoid misuse, malicious extensions can allow attackers to gain access to a wide range of private, sensitive data and computer resources.

At following, we refer related work has been done on detecting malicious behavior in browser extensions and a series of attacks by which malicious extensions can steal data, track user behavior and collude to elevate their privileges. In special, we refer to Hulk which is a tool for detecting malicious behavior in Google Chrome extensions.

## Hulk Architecture

Hulk is a dynamic analysis system that detects malicious behavior in Chrome extensions. It dynamically loads extensions in a monitored environment and observes the interaction of the extension with the loaded web pages. It uses a set of heuristics to identify potentially dangerous behavior and label extensions as malicious, suspicious or benign.

### HoneyPages

There are extensions that their activation is based on the content of a web page. In order to analyze such cases, Hulk uses HoneyPages which are specially crafted pages that dynamically create an environment for the extension to perform the actions it needs. That means when an extension queries for the presence of a specific element, such as an iframe DOM element, the HoneyPage will create the element, inject it in the DOM tree and return it to the extension.

### Fuzzer

Extensions can register callbacks that respond to certain browser-level events using an event-based model such as the chrome.webRequest API. HoneyPages are not able to trigger callbacks for network events that require special properties like a specific URL or HTTP header. Therefore, Hulk uses event handler fuzzing where all event callbacks are invoked with mock event objects. At the same time, a HoneyPage is loaded in the active tab which enables Hulk to monitor all changes that the extension attempts to make on the page.

### Malicious Behavior Detection

In Hulk’s presentation there were cases of extensions abusing Chrome’s extension API. Specifically, by monitoring the chrome.management.uninstall API calls, malicious behavior was detected where an extension uninstalled other extensions.

A malicious extension can also interfere with tabs that point to the extension configuration page (chrome://extensions) either by replacing the URL with a different one, or by removing the tab completely. As a result, the malicious extension denies the access to Chrome’s extension configuration page and the user is unable to uninstall any extension.

Using callbacks in the webRequest API, a malicious extension can manipulate HTTP headers. Extensions can use the webRequest API to effectively perform a man-in-the-middle attack on HTTP requests and responses before they are handled by the browser. This behavior is often malicious (or at least dangerous) since there are cases of extensions that remove security-related headers, such as Content-Security-Policy or X-Frame-Options, through the use of callbacks such as webRequest.onHeadersReceived and webRequestInterval.eventHandled.

In addition to dropping security related headers, extensions can change or add parameters in URLs before the HTTP request is sent. Such suspicious behavior is common, especially among extensions that request permissions on shopping related sites such as Amazon, eBay, and others.

## Chroak Extension

Chroak extensions was developed in order to demonstrate cases where the Chrome extension permissions model fails to effectively inform the user of an extension’s capabilities. There is a subset of permissions that are not considered potentially malicious or noteworthy by Chrome. Table 2 shows the permissions that Chrome has deemed as “non-special”.

Table 2: “Non-special” extension permissions.

|  |  |  |  |
| --- | --- | --- | --- |
| activeTab | declarativeContent | notifications | system.storage |
| alarms | desctopCapture | pageCapture | tabCapture |
| background | downloads | power | tabs |
| bookmarks | fontSettings | printerProvider | topSites |
| browsingData | gcm | privacy | tts |
| clipboardRead | geolocation | proxy | ttsEngine |
| clipboardWrite | history | sessions | unlimitedStorage |
| contentSettings | identity | storage | webNavigation |
| contextMenus | idle | system.cpu | web.request |
| cookies | management | system.display | webRequestBlocking |
| debugger | nativeMessaging | system.memory |  |

While Google Chrome seems to imply that no malicious activity can be conducted through requesting these “non-special" permissions and there is no indication to the end-user that an extension uses these permissions, Chroak uses these extension permissions to perform its attacks. In special, the permissions that were used are: background, clipboardWrite, fontSettings, notifications, power, sessions, system.cpu, system.display, system.memory, system.storage and tabs. Notifications and tabs are the only special permissions used.

The extension's homepage shows up every time a user opens a new tab. It has action buttons which a user can click to self-inflict (and undo) malicious actions, and it has a list of information that the extension knows about the user's computer at any given time.

### Information Gathered

Chroak extension managed to gather information from the following permissions:

* chrome.tabs: Number of tabs and open windows.
* chrome.system.cpu: Information about the user’s CPU (number of processors, processor activity time, processor architecture and CPU model).
* chrome.system.display: Information about user’s displays.
* chrome.sessions: User’s chrome sessions and number of connected devices.
* chrome.system.memory: User’s total memory and current memory usage.
* window: User’s IP address, network connection status, operating system, system language and chrome version.
* chrome.system.storage: Number, types and names of mass storage devices attached to the user’s computer.

### Attacks Performed

The attacks carried out by the extension are:

* Power Settings: Using chrome.power, the extension can prevent user’s computer from suspending on its own. This could result in a user’s laptop battery to be drained faster than expected.
* Notifications: Using chrome.notifications, the extension can spam the user with notifications that keep reappearing when closed.
* Clipboard: Using chrome.clipboardWrite, the extension can write arbitrary text to the system clipboard. This can cause major annoyance to the user or it can trick the user into navigating to a malicious URL.
* Font Size Modification: Using chrome.fontSettings, the extension can change the default font size. This can not only be annoying for the user, but also can result in a form of DoS.
* DoS: The denial-of-service attack was executed using no special permissions, where the extension closed all open tabs and kept closing all newly-opened tabs or windows. This makes Chrome impossible to use and the only way to fix this is to restart in dev mode or remove the extension.
* Phishing: Phishing was the only attack carried out by the extension using the special permission chrome.tabs. When a new tab with a specific URL is opened by the user, the extension silently redirects that tab to an arbitrary URL.

# DESIGN

## Architecture

The application consists of four parts. The first part will receive the message/information from the user. That message is the URL which contains the ID of the extension chosen by the user from the Chrome web store.

The second part is the one that will use the extension ID to download the source code of the extension. Once the .crx file is downloaded, it will extract all its files in the “\extensions” directory.

The third part is about parsing the source code. After all .js and .html file are listed with their absolute path, the parsing process is started. Each file according to its type, is parsed and the Abstract Syntax Tree is generated.

The fourth and final part is where the static analysis is performed. We try to match parts of the source code (using the AST) with rules which indicates suspicious activity and inform the user through warning messages. At the end, when the analysis is finished, all files included in the extensions folder are deleted.

## Native Messaging Host

Extensions and apps can exchange messages with native applications using a messaging passing API. The native application must register a native messaging host that knows how to communicate with the extension. Chrome starts the host in a separate process and communicates with it using standard input and standard output streams.

We use this technique to send the extension ID that the user chose to the native application which will download the source code and run the static analysis.

## Parsers

As we know, extensions support JavaScript and HTML. Malicious code can be found in both, therefore we used two parsers, one for each type. We use the Abstract Syntax Tree as a representation of the abstract syntactic structure of the source code written in each file of the extension.

## Rules

Now that we have the Abstract Syntax Tree (AST), we can start adding rules. As rule we describe a set of elements, such as call expressions, variables, keywords etc., that indicate suspicious activity when they are found in the source code.

In order to define a rule, we use lists and flags.

The callees list contains all callees of a call expression. For example, if the call expression is chrome.webRequest.onBeforeRequest.addListener, the list will contain: [chrome, webRequest, onBeforeRequest, addListener].

The CallExpression list contains all the call expressions or in other words the list of the callees that we described before.

The decs list contains all the callees of a variable declaration. For example, if the variable declaration is var document.forms, the list will contain: [document, forms].

The Vars list contains all the variable declarations, or in other words the decs list that we already described.

# IMPLEMENTAYION DETAILS

## Rule 1: Denying the access to Chrome’s extensions configuration page.

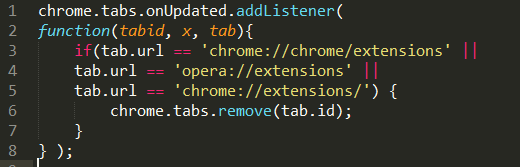


Figure 3: Access to Chrome’s extensions configuration page.

This rule detects denial of service that targets the extensions configuration page. The malicious code uses a listener to detect what URLs the user visits. When it is the configuration page for the extensions, it automatically removes the specific tab. As a result, the user is not able to disable or remove any extension installed.

## Rule 2: DoS.

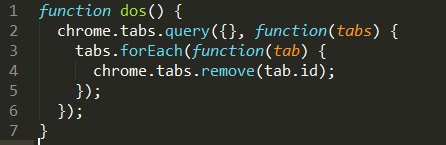


Figure 4: DoS.

This rule is similar to the previous one but the denial of service targets each tab the users tries to open. This prevents the user from using any Chrome’s service.

## Rule 3: Extension installation/uninstallation.

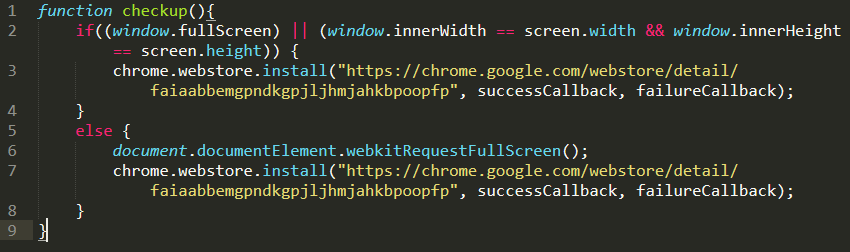


Figure 5: Extension installation/uninstallation.

This rule detects whether the extension installs or uninstalls other extensions. For the extension installation the chrome.webstore.install(url, successCallback, failureCallback) function is used (as we can see in the above picture) whereas, for the extension uninstallation the function used is chrome.management.unistall(id).

## Rule 4: HTTP response header security options.

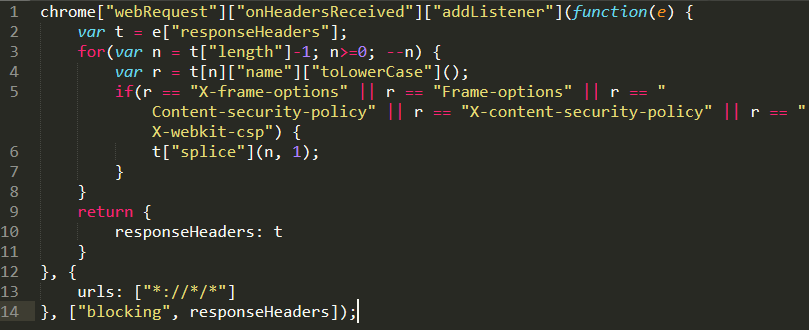


Figure 6: HTTP response header security options.

This rule detects malicious code used to remove security options from HTTP response header. It uses a listener to get all headers received and it checks for security options like X-frame-options, Frame-options, Content-security-policy, X-content-security-policy or X-webkit-csp to remove them.

## Rule 5: URL redirection.

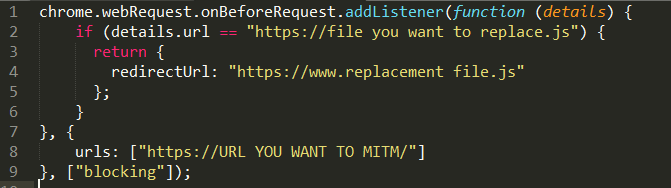


Figure 7: URL redirection.

This rule is about URL redirection where a listener is used for the URLs the user visits. When the URL matches the page that the attacker has targeted, the extension redirects this URL to an arbitrary one. This attack can be described as a Man-In-The-Middle attack or a phishing technique.

## Rule 6: Block access to websites.

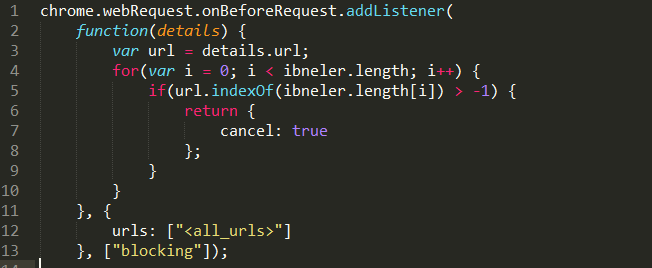


Figure 8: Access to websites blocked.

This rule detects whether the extension prevents the user from accessing specific websites. In this case, the attacker uses a listener for the URLs the user visits and blocks the access to them.

## Rule 7: Form submit requests.



Figure 9: Information from submit requests.

This rule is about malicious code that steals information from submit requests. An event listener is added for each form of the page visited by the user. When there is a submit event, the formSubmit function is triggered which creates an XMLHttpRequest to send all the names and values of the form.

## Rule 8: User’s CPU information.

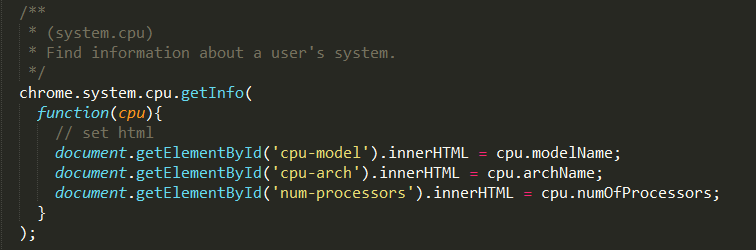


Figure 10: User’s CPU information.

This rule uses the system.cpu permission to get information about the user’s CPU.

## Rule 9: User’s number of displays.

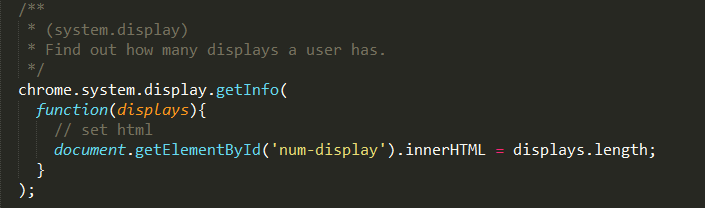


Figure 11: User’s number of displays.

This rule uses the system.display permission to find out how many displays the user has.

## Rule 10: User’s sessions information.



Figure 12: User’s sessions information.

This rule uses the sessions permission to get information about the user’s connected devices.

# EVALUATION

# CONCLUSION

Στο τέλος της εργασίας υπάρχουν τα συμπεράσματα που προκύπτουν από την έρευνα.

# REFERENCES

1. *What are extensions?* <https://developer.chrome.com/extensions>
2. *Native Messaging.* <https://developer.chrome.com/apps/nativeMessaging>
3. Alexandros Kapravelos, University of California, Santa Barbara; Chris Grier, University of California, Berkeley, and International Computer Science Institute; Neha Chachra, university of California, San Diego; Christopher Kruegel and Giovanni Vigna, University of California, Santa Barbara; Vern Paxson, University of California, Berkeley, and International Computer Science Institute. *Hulk: Eliciting Malicious Behavior in Browser Extensions.* <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-kapravelos.pdf>
4. Lujo Bauer, Carnegie Mellon University; Shaoying Cai, Singapore Management University and Institute for Infocomm Research; Limin Jia, Carnegie Mellon University; Timothy Passaro, Carnegie Mellon University; Yuan Tian, Carnegie Mellon University; *Analyzing the Dangers Posed by Chrome Extensions.* [*https://www.ece.cmu.edu/~lbauer/papers/2014/cns2014-browserattacks.pdf*](https://www.ece.cmu.edu/~lbauer/papers/2014/cns2014-browserattacks.pdf)
5. Iskander Sanchez-Rola, Deustotech and University of Deusto; Igor Santos, Deustotech and University of Deusto; Davide Balzarotti, Eurecom; *Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies.* <http://paginaspersonales.deusto.es/isantos/papers/2017/2017-sanchez-rola-extensions-usenix.pdf>
6. Ryan Chipman, Tatsiana Ivonchyk, Danielle Man, Morgan Voss. *Security Analysis of Chrome Extensions.* <https://courses.csail.mit.edu/6.857/2016/files/24.pdf>