An Evaluation of Extension Vulnerabilities of Google Chrome

Md. Mesbahul Islam
Department of Computer Science
University of Helsinki
Helsinki, Finland
mesbahul.islam@helsinki.fi

ABSTRACT
Browser extension is a third-party software module that extends a browser’s functionality and enhances the browsing experience of users. As extension can directly interact with untrusted web contents, it’s becoming one of the most common security vulnerable points and one of the primary choices for web attackers. In Firefox and IE, extensions and web scripts run under same browser process and extensions run with full user privileges which can result in exploiting arbitrary code by malicious web operators. To mitigate extension vulnerabilities, Google Chrome introduced a new extension model with features of privilege separation, isolated world and least privilege in 2009. In this report I will survey the Chromium browser’s extension security model and present an evaluation of the security features against malicious website operators and network attackers.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous;
D.4.6 [Security and Protection]: Subjects—Access controls, Invasive software (e.g., viruses, Trojan horses)

General Terms
Browser Security

Keywords
Browser extension, least privilege, isolated world, permission.

1. INTRODUCTION
At first web browsers were basic means of surfing Internet and web experience for people. Later browser extension was introduced which is, a small third-party software module, developed for enhancing browsing experience and expanding browser’s functionalities. All the time a secure browser is desirable to its users. Increasing popularity of the browser extensions start attracting the attackers to make the extension as a security vulnerable vector and thus increase the attack through browser extension. With this increasing attacks, lots of effort was made to propose better secure extension model over the time.

A secure browser protects network, computer system or data from threats such as phishing or malware, which could steal confidential data or make harm to the system. A phishing attack happens when attacker masquerades as some other honest website and tricks the user into confusing a dishonest website with that honest website. The user then can share sensitive and personal information with the attacker which will consequently cause harms. On the other hand, malware is installed into user’s machine, often without their consent, intended to harm computer or steal sensitive data from computer.

Traditionally, browser extensions and the browser itself run in the same process heap such as in IE and Firefox. As both are running in the same process space, malicious web contents can exploit buggy extensions in the browser to steal sensitive data or cause serious harm to the underlying operating system as the operating system share the same user privilege as the browser. Google Chrome came up with security policies, strong isolation between websites and extensions, separation of privilege within an extension, and permission of extension system, in 2009 [1] to prevent extension vulnerabilities. Google Chrome’s extension made from multi-component architecture which contains content script, core extension and optionally native binary.

Google Chrome is built upon Chromium’s modular architecture which has two modules: browser kernel and rendering engine. The rendering engine is responsible for converting requests into rendered bitmap whereas browser kernel is responsible for interacting with underlying operating system. Both modules execute under separate process and the rendering engine executes in a sandbox.

The paper is organized as follows. Section 2 describes attacks on extensions and threat model for Chrome extension model. Section 3 presents the Chrome extension architecture and its security model. Evaluation of the Chrome extension security model will be described in section 4. Finally, concluding remarks will be presented in section 5.

2. ATTACKS ON EXTENSIONS
As browser extensions can interact directly with web content, extensions are at risk of attacks while interacting with untrusted web content. The risk comes from either malicious web operators or active network attackers [1]. The attacker tries to corrupt the extension and uses its privileges to cause harm to the user. For example, an attacker can corrupt the extension to achieve one of the following goals:

- Persistent Malware. The attacker tries to install malicious software on user’s machine so that the soft-
In Firefox, browser extensions have the full access to browser internals and they run with the same privileges as the browser has. So, if an attacker can infect the extension, then it is possible to change the functionality of the browser, modify web sites’ behavior or even access to the file system. Previous study found that [1], only 3 of 25 popular Firefox extensions required full system access whereas remainder were over-privileged. This high level of privilege increased the severity of extension vulnerabilities.

In 2009 at DEFCON, Liverani and Freeman presented attacks against popular Firefox extensions [9]. One of the attacks was against Skype (<3.8.0.188) extension. If there is any phone number in a web page, Skype browser extension rewrites that phone number into hyper-links and there will be a green call button beside that link. If a user clicks on that button, Skype will launch and call that number. In this point the attacker gains privilege of user’s browser and exploits malicious code in user’s browser to corrupt browser extension or read sensitive data from browser.

3. GOOGLE CHROME EXTENSIONS

Chrome uses Chromium architecture which has two modules [2]: browser kernel and rendering engine which run in two separate protection domain. Rendering engine interacts with web with restricted privilege under sandbox and responsible for converting HTTP responses and user inputs into rendered bitmap. On the other hand, browser kernel interacts with underlying operating system as a user. Figure 1 shows a high level view of Chromium architecture where...
browser kernel treats the rendering engine as a black box [2].

Google Chrome’s extension model is multi-component architecture model. It consists of content script, core extension and native binary. To prevent extension vulnerabilities Chrome applies privilege separation between extension components, between content scripts and web contents and Chrome browser grants a restricted set of privileges. In the following sub-sections, I will present Chrome’s extension architecture and corresponding security model [1, 8].

3.1 The Extension Architecture

Chrome Extension’s Architecture was designed to mitigate the vulnerabilities of benign but buggy extensions. A chrome extension is divided into multiple contents. Usually an extension includes content script and core extension. Figure 2 is showing a basic architecture of Chrome extension.

- A content script is written in JavaScript and can interact directly with the web contents but run with less or no privilege. A content script is injected into a web site when the web page is loaded. The content script is isolated from the associated web pages as their process run in different process heap but they share same underlying DOM (Document Object Model). Although both the content script and websites access same DOM, they do not exchange variables or JavaScript pointers.

- The core extension can’t interact directly with the website contents and run with bulk of extension privileges. It includes one or more background pages which run in separate process. A core extension can only interact with web contents by invoking XMLHttpRequest or via content scripts.

- A native binary is optionally included with an extension which can interact directly with the host machine and run with full user privilege. In absence of a native binary, the core extension can interact with the host machine with full user privilege.

Chrome extension architecture has multi-layer security as the components of an extension are separated by strong process boundary. As they run under sandboxed process, they cannot access all of the operating system’s services. Moreover, the content scripts and the core extension can only communicate by passing message through a secure channel. If an attacker wants to gain the full privilege of user’s operating system then the attacker need to compromise the extension to send the malicious input from content script to core extension and then from core to native binary where the malicious input can exploits buggy code. But this is a very difficult process as the attacker needs to break through multiple security boundaries.

3.2 Chrome Extension’s Security Model

To mitigate the extension vulnerabilities, Google Chrome followed three security mechanisms: least privilege, privilege separation, and strong isolation. In this section, we will describe and evaluate each of these mechanisms.

**Least Privilege**

Chrome defines a set of permissions which includes accessing websites, executing scripts, and accessing different browser modules like tabs, bookmarks, history, cookies etc. According to the least privilege, each extension in Chrome browser runs with a restricted set of permissions which require to declare in a manifest.json file that comes packaged with the extension. Thus, an attacker, who compromises an extension to exploit some buggy code, can only gain access to those permissions that the vulnerable extension already has.

Consider the manifest file of Google Mail Checker [1, 5]:

```json
"permissions": [  
  "alarms",  
  "tabs",  
  "webNavigation",  
  "http://*.google.com/*",  
  "https://*.google.com/*" ]
```

In this file we can see that Google Mail Checker needs permission for accessing sub-domain of google.com and the Tabs API. Attacker can’t access bank account because he has no permission in such bank website.

**Privilege Separation**

Moreover Chrome separates privileges among the components of an extension to make it difficult for the malicious web contents to get access to the extension’s privileges. Usually an extension is made of content script and core extension. Content scripts can interact directly with the web contents and run with no privilege. Content scripts cannot directly access the browser modules but only can send a JSON message postmessage via the core extension. On the other hand, core extension can’t access directly or modify
Figure 2: An Extension’s components, each with progressively more privileges and less exposure to malicious web content [1, 4].

web contents but run with extension’s full privilege. A core extension can communicate with web contents through content scripts or invoking XMLHttpRequest. The native binary can only interact with the core extension and has the full user privilege to run arbitrary code or access any files.

**Strong Isolation**

The chrome extension architecture uses multi-process architecture and sandboxing technology for strong isolation. In Chrome, every website or web application opens in different tabs is correspond to a separate and independent process. So, if one tab crashes others will remain unaffected. The sandbox [10] is designed to build a restricted environment around each process. For example, if a user visits a malicious website or opens a familiar website which is currently hacked, the browser’s sandbox will prevent user’s machine and other processes from further damages. As the malicious content is running within a sandbox, it will not harm other tabs or machine’s data also. Closing the browser tab containing malicious content will be removed without making any harm to other processes and to the machine’s data.

The chrome extension architecture uses three levels [1, 8] of isolation mechanism to isolate extension contents from each other and from web sites. **First**, Chrome follows the same origin policy (SOP) to give access privileges to web sites. The origin of an extension is defined by including a designated public key to the extension. By default, an extension can access all resources of it’s own origin but need cross origin permission to access another origin’s resources. According to the extension security model an extension needs to inject a content scripts to a website to access. If the website does not belong to the extension’s origin then cross-origin permission is also needed to access the website. Cross-origin permissions are also defined in the manifold.json file of the extension. To understand this level of isolation, consider following sample code [6]:

```html
hello.html
==========
<html>
<button id="mybutton">click me</button>
<script>
    var greeting = "hello, ";
    var button = document.getElementById("mybutton");
    button.personName = "Bob";
    button.addEventListener("click", function() {
        alert(greeting + button.personName + "." );
    }, false);
</script>
</html>
```

Now, suppose this content script was injected into above web page:

```html
var greeting = "hola, ";
var button = document.getElementById("mybutton");
button.personName = "Roberto";
button.addEventListener("click", function() {
    alert(greeting + button.personName + "." );
});
```
Now, if button is pressed, there will be two greetings. We see that both are running in different JavaScript environment. Please refer to Figure 3 for better understanding of isolated world.

Second, the components of an extension run in different process space and so they are isolated from each other. When a content script injects in a web page, the script runs in the process space of the web sites. At the same time the core extension and the native binary run in their own processes.

Finally, in the third level, the content scripts and website’s scripts run in different JavaScript engine, which is called isolated world. Content script is interesting features of extension which can do useful things like download all image from a web page or convert and unlink URL to link element by interacting with the DOM (Document Object Model) of a web page. A DOM represents a tree like structure of a HTML page. The DOM is shared between the page’s JavaScripts and content scripts associated with the interacting extension. Although both are sharing same DOM, they are isolated from each other and executes under different process heap [7]. Since, the extensions have access to privileged data and APIs, the isolated world mechanism makes sure that the web page cannot use the DOM as a way of accessing extensions as well as those privileged data and APIs.

4. EVALUATION OF EXTENSION’S SECURITY

To make browser more secure the Chrome team presented a new security model of browser extension in 2009 [1] with the security mechanisms of least privilege, privilege separation, and permission. After that an evaluation had been done [4] on the extensions whether the mechanisms are sufficient for protecting browser extensions from web attacks. It was found that Chrome extensions security architecture can mostly prevent web attacks but failed to prevent network attacks. In this section I will present the evaluation of the extensions security mechanisms.

4.1 Evaluation of Isolation Mechanism

This mechanism is intended to protect content scripts from malicious web contents. In isolated world concept, the content scripts and the web site Javascript run under different Javascript engine. So, they can’t access each other functions or variables. This is also helps both to run different script version in their own process heap.

It is often seen that the extension developers are not security experts. So the developers face four security challenges [4] when writing extensions which are interacting with web contents. These are using data as HTML, using eval to execute untrusted data, click injection, prototyping.

Using Data as HTML. Inserting untrusted data as HTML in web page in one of the strong developments mistakes because it allows the untrusted data to run as code. Isolated world mechanism helps to prevent this type of mistakes. If data contains script which is inserted in a web page by content script, the script is executed in the web site’s process space instead of the extension’s. As a result, the content script can read data from one web site and copy the data to another web site which may introduce a security breach on the
Using eval. Another potential vulnerability can be introduced by using eval to execute untrusted data. Eval is very dangerous because it can execute any arbitrary data for example a script or expression. Consider a situation where an extension reads data from a website containing malicious code in that data and then the data is executed within content script by eval. Now, the resulting script will run in content script’s isolated world. So, in this situation isolated world does not mitigate the extension’s vulnerability due to the use of eval.

Click Injection. A click injection attack can be initiated by a website by invoking an extension’s click event handler. Actually this website tricks the extension to invoke the handler whereas this is not a user request at all. This situation can be happened when an extension registers some event handler for DOM elements from a website which can be untrusted. The isolated world does not mitigate this attack at all. This type of attack does not run any buggy code instead it controls the content script’s behavior.

Finally the authors [4] stated that isolated world can prevent prototypes and capabilities attack as this mechanism provide heap separation.

4.2 Evaluation of Privilege Separation

The goal of the privilege separation mechanism is to protect the core extension from attacks. The isolated world provides the first level security for core extension and the privilege separation comes into action when isolated world failed. That means privilege separation is very effective in absence of isolated world.

An extension is usually made of content script and core extension where the core extension runs with most of the privileges. The core extension can not interact directly with web contents. To do so core extension needs to communicate via content script or invokes XMLHttpRequest. The content scripts and core extension communicate through secure IPC channel. So, if an attacker wants to exploit malicious code in core extension to gain the privilege, then the attacker need to compromise content scripts to pass the code through message channel which is difficult in chrome extension architecture.

Now consider, the attacker wins to communicate with core extension and run arbitrary code, then attacker get extension’s privileges. But as an extension are issued only restricted set of permission so still now the attacker can’t get full user privilege.

It is an expectation of Chrome extension architecture that a malicious website first enter extension through content script, it can’t interact directly with core extension. But it is possible to attack the core extension without going through content scripts. Web site metadata is the way to access core extension directly as core extension can access these information (i.e. history, bookmarks etc.). If the metadata contains inline script then it can be a core extension vulnerability. In this situation privilege separation does not prevent it.

As mentioned earlier, that core extension can communicate web contents by invoking XMLHttpRequest, so it is clear that core extensions are vulnerable to direct network attack. Direct network attacks are the largest core extension vulnerabilities. Of the 50 core extension vulnerabilities, 44 vulnerabilities stem from HTTP scripts or HTTP XMLHttpRequest [4, 8].

4.3 Evaluation of Permission

The permission mechanism is intended to reduce the security breaches of core extension. If an attacker compromises content scripts to run arbitrary code in core extension then the permission mechanism helps to limit the severity of damage.

But the permission failed to protect core extension if the extension comes packaged with important permissions. It was shown that [8], Chrome extensions pose serious threats, including both information dispersion and harvesting due to the request of serious permission.

5. CONCLUSION

Now-a-days website attackers exploit malware in the browser extensions and it is increasing day by day. In this study, I have studied the security architecture of Google Chrome extension model and presented an evaluation of the architecture based on previous studies. It is shown that the isolated world mechanism is the most efficient to prevent the browser extensions from most of the malware attacks. This mechanism mitigates the vulnerability of content script. The privilege separation mechanism is the second layer safety for core extension but it’s usefulness come to front when the isolated world failed to protect. Finally, permission is also an important feature for limiting malicious attack on user’s machine.

It can be said that, the extension developers are not always security experts. So, sometimes using of dangerous code like inline scripts, innerHtml, eval etc. negate the benefit of the security mechanism. It is also stated that, Chrome’s extension model is intended to prevent web attackers rather than network attackers. There are no security mechanism to protect core extension from direct network attack or website metadata attack. So, there is scope of improving in the field of preventing network attacks. Finally, 94% of the most serious attacks can be removed [4] by banning insecure coding practice like ban on HTTP scripts and inline scripts.

6. REFERENCES


