A Cloud-based Framework for Security Analysis of Browser Extensions

by

Saikat Das

A thesis submitted to the
Graduate Program in Electrical and Computer Engineering
in conformity with the requirements for
the degree of Master of Applied Science

Queen’s University
Kingston, Ontario, Canada

May 2016

Copyright © Saikat Das, 2016
Abstract

In today’s internet world, web browsers are an integral part of our day-to-day activities. Therefore, web browser security is a serious concern for all of us. Browsers can be breached in different ways. Because of the over privileged access, extensions are responsible for many security issues. Browser vendors try to keep safe extensions in their official extension galleries. However, their security control measures are not always effective and adequate. The distribution of unsafe extensions through different social engineering techniques is also a very common practice. Therefore, before installation, users should thoroughly analyze the security of browser extensions.

Extensions are not only available for desktop browsers, but many mobile browsers, for example, Firefox for Android and UC browser for Android, are also furnished with extension features. Mobile devices have various resource constraints in terms of computational capabilities, power, network bandwidth, etc. Hence, conventional extension security analysis techniques cannot be efficiently used by end users to examine mobile browser extension security issues. To overcome the inadequacies of the existing approaches, we propose CLOUBEX, a CLOUD-based security analysis framework for both desktop and mobile Browser EXtensions. This framework uses a client-server architecture model. In this framework, compute-intensive security analysis tasks are generally executed in a high-speed computing server hosted in a cloud
environment. CLOUBEX is also enriched with a number of essential features, such as client-side analysis, requirements-driven analysis, high performance, and dynamic decision making.

At present, the Firefox extension ecosystem is most susceptible to different security attacks. Hence, the framework is implemented for the security analysis of the Firefox desktop and Firefox for Android mobile browser extensions. A static taint analysis is used to identify malicious information flows in the Firefox extensions. In CLOUBEX, there are three analysis modes. A dynamic decision making algorithm assists us to select the best option based on some important parameters, such as the processing speed of a client device and network connection speed. Using the best analysis mode, performance and power consumption are improved significantly. In the future, this framework can be leveraged for the security analysis of other desktop and mobile browser extensions, too.
Acknowledgments

I am grateful to a number of people without whose help this thesis would have not been possible:

My supervisor, Prof. Mohammad Zulkernine, for his great support, guidance, persistent help, and patience over the last two years;

My family who has been a source of encouragement, confidence, unconditional love, inspiration, and wisdom through my entire life;

My colleagues at Queens Reliable Software Technology Group (QRST) who have been more than supportive during this effort.
# Contents

Abstract i

Acknowledgments iii

Contents iv

List of Tables vii

List of Figures viii

Chapter 1: Introduction 1
  1.1 Motivation .............................................. 1
  1.2 Research Problems ...................................... 2
  1.3 Overview of Proposed Approach .......................... 4
  1.4 Contributions ........................................... 5
  1.5 Organization of Thesis .................................. 6

Chapter 2: Background 7
  2.1 Browser Extensions ...................................... 7
  2.2 Common Browser Extension Security Issues ............... 8
  2.3 Mobile Extension Security Challenges .................... 10
  2.4 Different Browser Extension Ecosystems .................. 13
  2.5 Chrome Desktop Browser Extension Architecture .......... 14
  2.6 Firefox Extension Ecosystem ............................ 15
    2.6.1 Firefox Extension Architecture ...................... 16
    2.6.2 Types of Firefox Extensions ......................... 18
    2.6.3 Security Vulnerability Factors ....................... 18
    2.6.4 Firefox Extension Security Issues ................... 21
  2.7 Summary ................................................ 22

Chapter 3: Related Work 24
  3.1 Browser Extension Security .............................. 24
List of Tables

3.1 A qualitative comparison of extension security analysis techniques. . . 32

5.1 List of possible XML security configuration values. . . . . . . . . . . 55

6.1 Mobile and remote analysis execution times in milliseconds for different Firefox for Android extensions. . . . . . . . . . . . . . . . 83

6.2 Dynamic decision making algorithm results. . . . . . . . . . . . . . 90

A.1 Browser preference related security critical XPCOM interfaces & func-
tions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109

A.2 Login manager related security critical XPCOM interfaces & functions. 110

A.3 Cookie related security critical XPCOM interfaces & functions. . . . . 111

A.4 File related security critical XPCOM interfaces & functions. . . . . . 112

A.5 Location related security critical XPCOM interfaces & functions. . . 112

A.6 Network related security critical XPCOM interfaces & functions. . . 113
# List of Figures

2.1 Chrome extension architecture. ........................................... 15
2.2 Firefox extension architecture. ........................................... 17
4.1 Different components of CLOUBEX. ................................. 38
4.2 Different extension security analysis modes available in CLOUBEX. . 40
5.1 XPCOM classification based on different security issues. .............. 49
5.2 Extension analysis steps. ................................................... 56
6.1 Code snippet taken from Login Manager Plus. ........................ 76
6.2 Code snippet taken from FFSniff. .......................................... 81
6.3 Remote server analysis time and data transfer time distribution for 
three different sized extensions. ........................................... 85
6.4 Remote analysis times using Queen’s wireless and Wind’s 3G network. 86
6.5 Mobile analysis times in the HTC Desire and Moto G mobile devices. 88
6.6 Remote analysis times using the Oregon and Singapore cloud servers. 89
Chapter 1

Introduction

1.1 Motivation

A web browser is a software application that is primarily used to retrieve and display information to users. Nowadays, browsers are used for accessing many security sensitive applications, such as banking transaction software, social networking sites, e-commerce applications, and location-based services. Therefore, browser security is a critical issue. In addition to core browsing components, all popular desktop browsers, for example, Chrome, Firefox, Internet Explorer, Safari, and Opera, are enriched with extension and plugin features. Extensions are used to add more functionalities in browsers, such as for changing the look and feel of user interfaces, translating web pages into different languages, and managing confidential user credentials in a secure way. In many browsers (e.g., Firefox) extensions work in the over privileged chrome zone. Because of this elevated access, extensions play crucial roles for browser security.

1Throughout this thesis, by extensions, we mean both desktop and mobile browser extensions. We explicitly mention desktop extensions and mobile extensions, if we need to express their characteristics separately.

2Similarly, by Firefox extensions, we denote both the Firefox desktop and mobile browser extensions. We specifically use Firefox desktop extensions and Firefox for Android extensions to indicate them differently.
Extensions are responsible for many security issues, such as the stealing of confidential information, cross-site scripting, cross-context scripting, and man-in-the-middle attacks.

In modern world, mobiles are omnipresent. There are more than seven billion mobile devices in the world [45]. Despite the wide usage of different mobile apps, the browser still remains one of the most popular applications in mobile devices. The number of mobile web browser users is increasing in an exponential way. The number of mobile-only internet users has already exceeded the number of desktop-only internet users in the United States of America [34]. Extensions are frequently used in many mobile browsers, e.g., Firefox for Android [3], Dolphin [11], and UC Browser for Android [47]. Extensions cannot only be unsafe for desktop browsers, but are also security critical for mobile browsers. Different factors, such as the small screen size of mobile devices [52], the vast usage of mobile devices in the corporate environment, and the existence of powerful sensors, make mobile browser extension security issues much more dangerous than desktop browser extension security issues. Therefore, from the security perspective, mobile browser extensions deserve special attention.

1.2 Research Problems

Different security analysis approaches, for example, static analysis [55, 63], dynamic analysis [57, 60, 68, 76, 81, 85], and model-based analysis [79], are available for desktop browser extensions. However, till now very little attention is paid towards mobile browser extension security. Each traditional security analysis approach has its own advantages and disadvantages. The inadequacies of the existing approaches become
1.2. RESEARCH PROBLEMS

much more prominent when we need to analyze the security issues of mobile browser extensions. There are many inherent resource limitations of mobile devices, such as poor processing speed, insufficient storage, and limited network bandwidth. Resource-demanding security analysis tasks cannot be accomplished very efficiently in low-speed mobile devices. Mobile web users are very much concerned about the performance of their browsers. Therefore, detecting the security issues of mobile browser extensions by checking their runtime behaviors is also not an ideal option. For that reason, there should be a security analysis framework which can analyze both desktop and mobile browser extensions in an efficient manner.

Browser vendors try to follow some basic security review processes before placing extensions in their official repositories. However, on many occasions, they fail to detect the potential security issues of extensions and place the vulnerable extensions in the extension galleries. Once, Mozilla discovered serious security vulnerabilities in two popular extensions, Mozilla Sniffer and CoolPreviews, which were available in the Mozilla add-on site for a long time. The CoolPreviews extension was downloaded by almost 177,000 users from the Mozilla’s official extension repository before the security issue has been identified [17]. Some vendors (e.g., Mozilla) keep experimental extensions in their websites [19]. Those experimental add-ons do not undergo extensive security review processes and may have severe security flaws. Very often, end users make mistakes by completely relying on those experimental extensions. Sometimes, browser vendors take quite a long time to complete a full-fledged review process because of the long queue of extension review requests and some manual review steps. Finally, the official repositories are not the only sources of extensions. The sideloading of extensions from different unsafe repositories is also a common phenomenon.
Security requirements are very much user specific. For example, if a particular user always turns off his or her location device, then location specific security analysis is irrelevant for him or her. However, there is no fine-grained requirements-driven extension security analysis approach available in the market. Hence, there should be a parallel security analysis technique by which general users can efficiently analyze the security issues of browser extensions as per their specific needs before installing them in browsers.

1.3 Overview of Proposed Approach

Keeping in mind the various limitations of mobile devices, we design CLOUBEX, a CLOUd-based generic security analysis framework for Browser EXtensions. This framework helps users detect the security issues of both mobile and desktop browser extensions in an efficient way irrespective of device configurations and existing network conditions.

CLOUBEX is built on the principle of a client-server architecture model. In this framework, compute-intensive static analysis tasks are generally executed in a remote powerful server hosted in a cloud environment. There is a client analysis module by which users can perform the analysis tasks in mobile devices, if the particular devices have sufficient resources or if the users do not want to divulge their security requirements to the remote server. CLOUBEX is also equipped with a requirements-driven security analysis feature. In CLOUBEX, a decision making algorithm is used to suggest users the best possible analysis option based on some dynamic parameters, such as network connection speed, extension size, and the computing speed of a client device.
From the security point of view, the Firefox extension ecosystem is the most dangerous one [56]. A relatively weaker extension architecture and Mozilla’s lack of security standards give rise to many security problems. Hence, to evaluate the effectiveness of the proposed framework, we implement it for the security analysis of the Firefox extensions. By using this approach, we identify several security issues in different Firefox desktop and Firefox for Android mobile browser extensions. We can also achieve significant improvements in performance, power consumption, and average central processing unit (CPU) load, by executing the static analysis tasks in a high-speed cloud server and caching the analysis results in a database repository. In the future, with proper extension analysis techniques, this framework can also be adapted for other desktop and mobile browser extensions.

1.4 Contributions

In brief, the main contributions of this thesis are as follows [59]. We review different extension security analysis approaches. To address the shortcomings of the existing approaches, we propose CLOUBEX, a cloud-based comprehensive security analysis framework. By using this framework, the security issues of both desktop and mobile browser extensions can be analyzed more efficiently and effectively. With proper analysis techniques, this framework can be utilized for the security analysis of different desktop and mobile browser extensions. We can also integrate other existing static security analysis techniques with this framework and can further improve the performance of those techniques. Another advantage of the proposed framework is that users can define their security requirements prior to the analysis. The requirements-driven security analysis is missing in most of the existing security analysis approaches.
To assess the usefulness of the proposed framework, we implement a static security analysis technique for the Firefox desktop and Firefox for Android mobile browser extensions. Recently, Mozilla has made the signing process mandatory for installing an extension in the Firefox browser. It is expected that from now on, they need to analyze more extensions than before. In the future, Mozilla can also take advantage of this automated analysis technique to expedite their extension review processes. As of now, Mozilla does not provide any user specific analysis option. They can use the user-requirements-driven analysis concept and issue a user specific installation key during the signing process.

1.5 Organization of Thesis

The remainder of the thesis is organized as follows. In Chapter 2, we present background information on browser extension and corresponding security issues. In this chapter, we also discuss why mobile browser extension security issues are much more dangerous than desktop browser extension security issues. In Chapter 3, we survey some related work. We analyze the strengths and weaknesses of the existing approaches, and how the proposed technique can overcome the deficiencies of the available approaches. The proposed framework is presented in detail in Chapter 4. Chapter 5 describes the static security analysis technique used to identify malicious information flows and security vulnerabilities in the Firefox extensions. Chapter 6 presents the implementation details and explains the evaluation results. In Chapter 7, we summarize the thesis and discuss the limitations of the proposed approach and our future plans.
Extensions are used to implement additional features in browsers. However, the extra privileges of browser extensions introduce many security issues in desktop and mobile browsers. In this chapter, we present a basic overview of browser extensions, browser extension security issues, and different extension ecosystems. We also explore different characteristics of the Firefox extension ecosystem which is unanimously considered as the most vulnerable extension system in the browser extension world.

2.1 Browser Extensions

Browser vendors always focus on core browsing operations. Hence, most of the web browsers come with only basic browsing facilities. The development of additional components is normally handed off to third parties. It helps reduce potential bugs in a browser codebase. However, users are not always satisfied with the basic features. They want a much more sophisticated browsing experience. Browser extensions are used to add those extra functionalities. Extensions can change the browsing experience of end users in different ways, such as by modifying user interfaces, providing utility services, and managing users’ important information in an efficient way.
2.2. COMMON BROWSER EXTENSION SECURITY ISSUES

Usually, there are two zones in browsers, a privileged chrome zone and an unprivileged internet zone. Web pages are loaded in the unprivileged internet zone. Extensions work in the privileged chrome zone. As a result, extensions have elevated access. This behavior is consistent across different browsers. Extensions use the same process space shared by other browser components. They have access to different privileged APIs. For example, the Firefox extensions can access a set of highly privileged Cross Platform Component Object Model (XPCOM) APIs. By using those APIs, extensions can access different types of sensitive browser information and the underlying operating system.

Plugins are also third party software applications used to display additional content in browsers. A plugin only affects a specific page in which it is placed, whereas an extension can operate across all web pages. From that perspective, extension security issues are much more severe than plugin security issues. Extensions are widely exploited for accomplishing different kinds of security attacks.

Some security issues are common for both desktop and mobile browser extensions. Mobile browser extensions are also responsible for some additional security issues. The common browser extension security issues and the mobile browser extension specific security challenges are discussed in the following sections.

2.2 Common Browser Extension Security Issues

A brief description of security issues applicable for both desktop and mobile browser extensions is presented in this section. Extensions are frequently used to fetch different types of security sensitive browser information, such as users’ login credentials, cookies, bookmark, livemark, caches, sessions, and browser logs. They can even steal
2.2. COMMON BROWSER EXTENSION SECURITY ISSUES

critical information from the Document Object Model (DOM) and introduce many user interface (UI) related security issues by operating on the DOM.

By abusing flaws in extension code, malicious scripts can be injected through different channels, for example, URL and DOM properties. If security principles are not properly followed, then extensions can produce different types of cross-site scripting (XSS) issues. By default, web page scripts work in the unprivileged internet zone. On the other hand, extensions function in the privileged chrome zone. However, if the unprivileged web scripts are smuggled into the privileged zone through extension vulnerabilities, then those scripts can introduce different cross-context scripting (XCS) attacks.

Extensions can send cross-origin requests and divulge sensitive information to an outsider. Since extensions can send cross-origin requests, they can be maliciously used to initiate distributed denial-of-service (DDoS) attacks. Quite often, the communication between an extension and the external world is accomplished through an insecure HTTP channel. In such a scenario, sensitive data can be intercepted and modified in the communication channel itself. Therefore, extensions are responsible for many man-in-the-middle attacks. Extensions can download malicious files, e.g., malware, spyware, adware, and viruses. It is also possible to inject malware through extension code.

Extensions are not only capable of accessing sensitive information from the file system, but also can create, modify, and delete sensitive files and directories. Interaction with the operating system layer is a very powerful feature of extensions. Occasionally, extensions are used to run malicious executable processes. Extensions
2.3. MOBILE EXTENSION SECURITY CHALLENGES

can communicate with device sensors such as the location tracking system and the ac-
celerometer. Hence, they are responsible for introducing many sensor related security
issues.

Users’ online activities can be monitored by extensions. Sometimes, extensions
are purposefully used to alter browser user-agent information. However, by inspect-
ing the invalid combination of the information, attackers can understand the role
of extensions. The shortcomings of extensions can be exploited by the attackers as
additional fingerprinting features.

Additionally, extensions can be abused to produce other types of security attacks
such as phishing attacks, cross-site request forgery, and click fraud. These third
party software applications can also inject different types of security vulnerabilities
by changing the security settings of browsers. By impersonating benign extensions,
malicious extensions can trap users to disclose sensitive information. One malware
extension can control other benign extensions and invite various potential security
issues.

2.3 Mobile Extension Security Challenges

Mobile browser extensions introduce more security challenges than desktop exten-
sions. There are many reasons behind those additional security challenges as discussed
below.

Proliferation of mobile devices: Because of the portability and mobility fac-
tors, mobile devices are used for various types of next generation applications, such
as location-based services, enterprise, and defense applications. With the advent of
2.3. MOBILE EXTENSION SECURITY CHALLENGES

high-speed internet connection, people use mobile devices quite frequently for accessing different types of security sensitive applications, e.g., banking software and social networking sites. The Bring Your Own Device (BYOD) policy permits employees to use their own smartphones for accessing sensitive corporate information. Mobile browser extensions can be used to access and transmit different kinds of browser, operating system, and sensor specific critical information. The usage of mobile devices for a wide range of applications helps attackers exploit different mobile browser extension vulnerabilities more severely.

**Resource constraints:** Mobile devices have some constraints in terms of computing resources, memory, power, etc. In most of the mobile web browsers, security guidelines are not strictly followed [53]. The scarcity of resources does not facilitate the application of regular security updates on mobile browsers. Comprehensive mobile anti-virus tools are still not readily available in the market. Existing mobile anti-virus tools are signature-based and unable to detect new extension vulnerabilities. The lack of a keyboard in mobile devices encourages users to use mobile browser extensions extensively for storing different types of secret information, such as login credentials. Those extensions definitely provide some sort of conveniences to mobile users. However, at the same time, this practice creates a wide surface of security vulnerabilities, as the critical information is shared across the entire browser.

**Screen related security issues:** Because of the improper handling of display elements and the small screen size, mobile browsers are susceptible to different kinds of security attacks, e.g., click fraud, login cross-site request forgery (CSRF), phishing attacks, and password stealing [52]. As extensions can modify the look and feel
of browser interfaces, they are sometimes directly or indirectly liable for many mobile screen related security issues. There are mobile browsers which cannot properly manage overlapping elements. In those faulty mobile browsers, when a user clicks a top browser element, an action associated with a bottom element gets executed. A malevolent extension can purposefully place an attractive advertisement on top of a malicious web content and initiate a click fraud attack. In this kind of scenario, a user may click the benign content and unknowingly trigger a harmful action linked to the underneath content. Attackers normally get huge financial benefits from click fraud attacks [65]. To explore how mobile browser extensions can introduce different phishing attacks, we build an experimental Firefox for Android extension. This extension creates malicious cross-site elements and pushes the honest content of a login page out of the normal view of a mobile screen. A user may divulge his or her sensitive login credentials by placing the secret data in the wrong fields and clicking the malicious button created by the extension. The small address bar of mobile browsers and the launching capability of mobile extensions are also responsible for many phishing attacks.

**Strong sensors:** Device sensors are much stronger in mobile devices than desktops. A GPS-based location tracking system is used in mobile devices. It provides more accurate location information. Since extensions can interact with different types of mobile sensors, sensor related extension security issues are prominent in mobile devices.

**Scarcity of security knowledge:** The huge penetration of mobile devices among general people introduces additional security challenges because common people are
not very much aware of standard security practices.

Extension architectures are different for different browsers. As a result, all the above mentioned security issues may not be applicable for all browser extension ecosystems. In the next section, we discuss different browser extension ecosystems.

### 2.4 Different Browser Extension Ecosystems

Extensions are available for all popular desktop browsers, e.g., Firefox, Chrome, Internet Explorer, Safari, Opera, and RockMelt. Among those, Firefox has the most number of extensions. Chrome has also many useful extensions. Chrome’s permission based extension architecture is more secure than Firefox’s flat extension architecture model. Opera provides few extensions in the form of widgets. The Internet Explorer (IE) extensions consist of browser helper objects, toolbars, and ActiveX controls. The IE extensions are compiled to native code. As a result, the IE extensions are vulnerable to many traditional security issues, such as buffer overflows, format string attacks, and integer bugs.

Many mobile browsers, for example, Firefox for Android [3], Dolphin [11], UC browser for Android [47], and Maxthon [29], are also equipped with extension features. Those extensions provide mobile web users different kinds of functionalities. Skyfire, a fully-owned subsidiary of Opera, develops a mobile extension platform, named Horizon [24]. The Horizon extensions can be deployed on Android phones by original equipment manufacturers and telecom operators. An extension feature is experimentally supported in the 12-th release of the Opera Mobile for Android browser [35]. Microsoft unveils that they are also contemplating to incorporate an
extension support in the Windows Mobile 10 inbuilt browser [13]. With the introduction of sophisticated smartphones, high speed internet connection, and users’ high expectations, the mobile browser extension field has been growing very rapidly. Therefore, we should pay special attention towards mobile browser extension security.

In the next section, we present the Chrome extension architecture which is considered the most secure one in the existing browser extension space. Among all the extension systems, the Firefox extension ecosystem is regarded as the most vulnerable one. There exist a number of research work on the Firefox extension security issues [1, 56, 84]. We discuss the characteristics of the Firefox extensions in a separate section.

2.5 Chrome Desktop Browser Extension Architecture

The Chrome extension architecture is depicted in Figure 2.1. This architecture adheres to three standard security policies: privilege separation, isolation, and least privilege. The Chrome extension architecture comprises of different layers, such as content scripts, UI, and background pages [36]. The content scripts can interact with web pages. The UI pages are used to build extension user interfaces. The background pages contain core extension logic. The content scripts cannot directly communicate with the UI and background pages. A communication between a content script and a background page is accomplished via messages only. Because of this restricted communication model, remote malicious scripts cannot be easily used to access privileged extension APIs.
To segregate the content scripts from the web page scripts, the Chrome architecture uses a concept called isolated worlds. Each isolated world has its own representation of DOM. A content script can access and modify a DOM tree, but cannot access another script’s isolated world.

To use different privileged extension APIs and resources, developers need to explicitly mention all permissions in a manifest file. This least privilege policy model helps restrict extra privileges at installation time.

### 2.6 Firefox Extension Ecosystem

In this section, we first discuss the Firefox extension architecture in detail. There are three different types of Firefox extensions which are presented in the next subsection. Next, we analyze different factors that affect the security of the Firefox extensions.
Finally, we explore security issues associated with different Firefox components.

### 2.6.1 Firefox Extension Architecture

The Firefox extension architecture is presented in Figure 2.2. The main components of the Firefox extensions are XPCOM, XPConnect, JavaScript, and XML User Interface Language (XUL). XPCOM is a cross platform object model similar to Microsoft COM. The XUL files are used for creating user interfaces. Instead of using the XUL files, the Firefox for Android extensions use a native window for building and customizing user interfaces. Extension core behaviors are controlled by the JavaScript files. XPConnect is a binding language by which a JavaScript file can interact with the XPCOM components.

The Firefox extension files are packaged within a Cross-Platform Installer Module (XPI). In addition to the core JavaScript files, a traditional Firefox extension contains many other files and folders, such as `install.rdf` and `chrome.manifest`. The `install.rdf` file contains different meta-data information about an extension, e.g., unique identifier, target application, and update URL. The `chrome.manifest` file specifies the location of different folders, for example, `Locale`, `Content`, and `Skin`.

The Firefox extensions can access a large number of XPCOM interfaces which cannot be used by normal web page scripts. Therefore, the Firefox extensions are more privileged than the web page scripts and are not subject to the same origin policy. To enforce the principle of least authority, Firefox introduces an add-on SDK named Jetpack framework. However, the Jetpack core modules have more privileges than required [69]. As per the classical or traditional approach, it is still possible to bypass the Jetpack architecture and interact directly with the XPCOM components through
XPConnect. In fact, the Firefox extension developers are much more accustomed to the classical approach [76]. As a result, the Jetpack framework fails to bring sufficient security controls in the Firefox extension architecture. Developers can also create custom XPCOM components and Jetpack modules.

The Firefox extension architecture is not separated into different layers like the Chrome extension architecture. Hence, if remote malicious scripts are smuggled into extension code, then those scripts can easily access sensitive native layers such as the file system, the operating system, and device sensors. Firefox applies a one-way privilege separation mechanism in which extension namespaces are only hidden from web page scripts [61]. However, extensions can directly access web page content.

Additionally, in this Firefox extension architecture, one extension can control other
extensions. In the previous versions of Firefox, there was an option by which an extension could be silently installed without any user’s intervention. This unsafe control mechanism and silent installation process may add additional security vulnerabilities.

2.6.2 Types of Firefox Extensions

There are three different types of Firefox extensions: classical [22], bootstrapped [18], and add-on SDK extensions [16]. A classical or traditional extension uses XUL overlays. For the classical extensions, a browser restart is mandatory after an installation. Instead of using the XUL overlays, a bootstrapped extension manages its user interfaces programmatically. The bootstrapped extensions contain a bootstrap.js file which includes different life cycle functions, such as install, uninstall, startup, and shutdown. The Firefox browser invokes those functions automatically. As a result, after an installation, no restart is required for the bootstrapped extensions. To ease the development process and enhance the Firefox extension security, Mozilla introduces an add-on SDK framework. The add-on SDK extensions use many high-level and low-level JavaScript APIs to accomplish their intended functionalities. However, even after the introduction of the add-on SDK framework, the classical and bootstrapped extensions still dominate the market.

2.6.3 Security Vulnerability Factors

Among all the existing browser extensions, the Firefox extension ecosystem is the most vulnerable one mainly because of its insecure architecture, development flexibilities, lack of security policies, inefficient testing standards, side loading, and huge popularity. In the following paragraphs, we discuss each factor one by one.
Insecure architecture: Architecture plays a key role in the security of browser extensions. As discussed in Subsection 2.6.1, Firefox has a relatively weak extension architecture which leads to many security issues. The Chrome extension architecture has different layers, and remote scripts cannot directly access low-level sensitive layers such as the file system, the operating system, and device sensors. On the contrary, in the Firefox extension architecture, there is no such security boundary which could prevent unauthorized access of remote malicious scripts.

Development flexibilities: Developers can easily build different types of Firefox extensions either manually or by using the add-on software development kit (SDK). Firefox already comes with many in-built XPCOM interfaces and functions. Moreover, developers can implement powerful custom XPCOM components by themselves. They need not to be very proficient in low-level programming languages, e.g., C and C++. Mozilla provides an option to build custom XPCOM components using the JavaScript language [25]. It is also possible to build new Jetpack modules and directly access the XPCOM interfaces from those custom modules in an insecure way. This wide range of implementation flexibilities helps attackers build extensions for different malicious purposes.

Lack of security policies: Detailed security guidelines are not available for the Firefox extension developers. Their lack of security knowledge and insecure development practices introduce many security vulnerabilities in practice. The Firefox extension security policies are also relatively weaker. For example, the Dolphin extension developers can build extensions using the Dolphin garage SDK, but for installation, they need an API key from the vendor [40]. On the contrary, till the version 43,
Mozilla did not enforce signing. The Firefox extension developers can develop and install extensions without seeking any permission from Mozilla. Signing is also an expensive process. Therefore, the installation of the unsigned Firefox extensions has been a very common practice for a long time.

**Inefficient testing standards:** Mozilla’s testing standards have been proved to be inefficient on many occasions [17]. Once they have officially revealed that their virus scanner was not very effective [15]. Mozilla continuously tries to upgrade their testing standards. However, if a malicious extension is distributed from the official repository, then it is very difficult to uninstall it from all affected systems. Mozilla keeps many experimental add-ons in their extension gallery [19]. For those add-ons, detailed security policy compliances are not checked. Sometimes, end users blindly rely on the official repository and mistakenly install many vulnerable extensions.

**Side loading:** In addition to downloading extensions from the Mozilla’s official repository, users download extensions from different unauthorized web sites. Extensions can be distributed through different social engineering techniques. In the age of social networking, it is not a difficult task to convince users to install malicious extensions. However, those side loaded extensions are not always safe to use.

**Popularity:** From the usage perspective, the Firefox desktop browser is the third most popular browser [9]. As per the Google Play statistics, the Firefox for Android mobile browser has more than 100 million users [21]. There are more than fifteen thousand Firefox desktop browser add-ons available in the Mozilla’s official repository [46]. Some Firefox extensions (e.g., Adblock Plus) have millions of end users [31].
Because of this huge popularity of both the Firefox browsers and extensions, attackers are always tempted to target the Firefox extensions.

2.6.4 Firefox Extension Security Issues

All the common extension security issues, mentioned in Section 2.2, are prevalent for the Firefox extensions. Major security issues associated with different Firefox components are elaborated in the following paragraphs.

XPCOM related security issues: The XPCOM components provide a wide range of functionalities for the Firefox extensions. Hence, the XPCOM interfaces are frequently exploited by attackers for many malicious purposes, e.g., for stealing sensitive information, disclosing information to the outside world, downloading malicious files, executing harmful operating system processes, and accessing vital file system information. The XPCOM components are also susceptible to many memory related security issues.

Security issues in configuration files: Some elements in the install.rdf, for example, em:optionsURL and em:aboutURL, may contain malicious XSS payloads [12]. The em:updateURL element is used to control the update process of an extension. If an HTTP URL is used for the em:updateURL without configuring the em:updateKey element, then it is possible to initiate a man-in-the-middle attack [27]. In the earlier versions of Firefox, the em:hidden=true value was used to hide an extension in the add-on manager section [26]. There are many security loopholes in the chrome.manifest file, too. Different critical extension contents can be exposed to an untrusted user by specifying contentaccessible=yes in the manifest file [10].
Other security issues: The Firefox extensions can create many security issues by accessing and modifying the sensitive document object model (DOM). The `eval` function is used by the Firefox extensions mainly to get user inputs at runtime. This `eval` function is a potential injection point for remote malicious scripts [8]. In many cases, the `eval` function is responsible for many cross-site scripting attacks. If a JavaScript Object Notation (JSON) is used, then from the security perspective it should not be used within an `evalInSandbox`. A `wrappedJSObject` minimizes the security control by allowing users to access the JavaScript objects hidden by the wrappers [49]. It is possible to install malign programs, e.g., malware, spyware, and viruses, in a user’s machine through an extension installation process. A malicious extension can control other benign extensions and create many security issues.

Considering the range of functionalities, extra privileges, and vast usage, security issues introduced by the XPCOM components are the most critical ones.

2.7 Summary

Over-privileged browser extensions are responsible for many security issues, for example, the stealing of secret information, cross-site scripting, cross-context scripting, and man-in-the-middle attacks. In recent times, extensions are not only available in desktop browsers, but many mobile browsers, e.g., Firefox for Android and Dolphin, are also complemented with extension features. Mobile browser extension security issues are much more critical than desktop browser extension security issues because of several reasons, such as the wide usage of mobile devices in different security critical domains, various resource constraints of mobile devices, and strong sensors. All
extension ecosystems do not follow same security standards and principles. The Firefox desktop and Firefox for Android mobile extensions appear as the most vulnerable extensions because of a number of factors, such as an insecure Firefox extension architecture, and inefficient extension security testing methodologies followed by Mozilla. From the security perspective, users should pay special attention towards all vulnerable desktop and mobile browser extensions before using their services.
Chapter 3

Related Work

In the previous chapter, we briefly discussed the security issues of browser extensions. Before diving deep into our proposed approach, it is important to get an overview of different related work and their deficiencies, and why a distinct approach is required to address the existing problems. In this thesis, we propose a cloud-centric security analysis framework for analyzing mobile and desktop browser extensions. Considering the main components of CLOUBEX, we review the relevant work from three different perspectives. First, we explore the existing research work on browser extension security. Then, we compare our approach with various mobile anti-virus tools. Finally, we discuss the characteristics of different static security analysis techniques used for other programming languages, such as Java and PHP.

3.1 Browser Extension Security

We divide this section into four subsections. First, we present the empirical studies on desktop browser extension security and secure extension architecture proposed in literature. Second, different types of desktop extension security analysis approaches are elaborated. Third, we discuss an existing security analysis technique for mobile
3.1. BROWSER EXTENSION SECURITY

browser extensions. Finally, we present a qualitative comparison of different extension security analysis techniques.

3.1.1 Empirical Studies on Desktop Browser Extension Security

The browser extension security has been an area of interest for researchers for a long time. A number of empirical studies [1, 58, 62, 72, 78, 84] are accomplished on the browser extension security. Wang et al. [84] carry out an empirical study on the security issues of the Firefox extensions. The Firefox extensions are capable of stealing sensitive information from both the Firefox browser and underlying operating system. In addition to these direct threats, the Firefox extensions can introduce many indirect threats by injecting malicious content into web sessions. To mitigate the indirect extension threats, Wang et al. propose a technique which restricts the behaviors of the injected content.

Google implements a permission-based secure extension platform for its Chrome browser. Carlini et al. [58] perform an extensive evaluation of the Google Chrome extension architecture from the security point of view. To prevent direct web attacks on extensions, the Chrome extension architecture uses many standard security principles, such as privilege separation, isolation, and permissions. However, those security mechanisms are not sufficient enough to prevent direct network attacks, website meta-data attacks, and attacks on websites produced by extensions. After the in-depth study of the Chrome extension architecture, finally, they come to the conclusion that the existing vulnerabilities can be mitigated by prohibiting HTTP and inline scripts.
3.1. BROWSER EXTENSION SECURITY

3.1.2 Secure Extension Architecture

To overcome the security loopholes of the Firefox extension ecosystem, Barth et al. [56] propose a secure extension architecture model based on three important security principles: privilege separation, isolation, and explicit permission. Google Chrome follows the same security architecture recommended by Barth et al.

The secure architecture of browser extensions is widely studied in different other research work. The IBEX [63] framework is used for authoring, analyzing, and verifying browser extension security policies. This framework can be used by three different groups of people: extension developers, the administrators of extension hosting services, and end users. IBEX provides a set of high-level secure browser-independent extension APIs. A policy language is used for defining extension privileges. Before hosting extensions in public repositories, the administrators can verify the safety of extensions by a static verification tool. Both the extension code and user-defined policy file are verified to find out different extension security issues.

IBEX has a security analysis component, too. However, IBEX and the research done by Barth et al. are much more related to the secure architecture of browser extensions. On the other hand, CLOUBEX deals with detecting security issues in malicious and vulnerable extensions.

3.1.3 Security Analysis Approaches for Desktop Browser Extensions

There exist a number of security analysis approaches for desktop browser extensions, such as static analysis [55, 69], dynamic analysis [57, 60, 61, 68, 76, 81, 85], and model-based analysis [79].
3.1. BROWSER EXTENSION SECURITY

Static Analysis

In static analysis, extension code is analyzed without actually executing the extension. VEX [55], a popular security analysis tool, uses a static information flow analysis for detecting security vulnerabilities in the Firefox extensions. They consider five important source-to-sink information flows. The sensitive flows include content document data to `eval`, content document data to `innerHTML`, Resource Description Framework (RDF) to `innerHTML`, `evalInSandbox` object to `==` or `!=`, and method call on `wrappedJSObjects`. We also use a static analysis technique to analyze the security issues of the Firefox extensions. In VEX, only one sensitive XPCOM component, i.e., RDF is considered. Our comprehensive security analysis procedure covers a vast range of XPCOM components. VEX is designed mainly for the Firefox vendor, whereas CLOUBEX is enriched with a requirements-driven security analysis feature and can be simultaneously used by both end users and browser vendors.

Following the secure Chrome extension architecture, Mozilla introduces an add-on SDK named Jetpack framework for their Firefox extensions. The Jetpack framework comes with some core modules which satisfy the principle of least authority (POLA). However, there are some security loopholes in this framework, too. Karim et al. [69] investigate the ineffectiveness of the Jetpack extension framework in a detailed way. Moreover, the Firefox architecture allows to directly access the XPCOM components without using the Jetpack modules. For this reason, the add-on SDK is not widely adopted by the Firefox extension developers. In fact, out of the top 1,000 Firefox extensions, only 3.4% utilize the Jetpack approach [76]. Therefore, for the security analysis purpose, we consider only the traditional Firefox extensions in our approach.
3.1. BROWSER EXTENSION SECURITY

In general, a static analysis technique can check all possible execution paths, and it does not have any runtime overhead. Hence, the static analysis approach is relatively a better option when analysis tasks need to be executed in low-speed mobile devices. However, it is unable to detect security issues injected at runtime.

Dynamic Analysis

In dynamic analysis, extension security issues are detected at runtime. SABRE [60] uses an in-browser information flow analysis to identify confidentiality and integrity violations in JavaScript-based browser extensions. SABRE needs to strictly monitor each JavaScript instruction. It is reported that with the SunSpider benchmark suit, a SABRE-enabled browser has a significant overhead of 6.1 times. Djeric and Goel [61] recommend a similar taint-based system to defend privilege escalation attacks. A modified browser with their proposed approach has an overhead of 28% and 32% for the JavaScript and DOM tests, respectively.

In SENTINEL [76], users can specify security policies and avert different types of XPCOM related security attacks at runtime. The average performance overhead of SENTINEL is 7.5%. The modified browser also imposes a 59.2% delay at the time of launching the Firefox browser. Similar to SENTINEL, Wang et al. [85] propose a fine-grained access control framework. In their framework, access control policies are devised dynamically to impede the access of sensitive resources by browser extensions. The framework has an additional overhead of 7.567% with the SunSpider suit.

Ter Louw et al. [81] propose a code integrity checking mechanism and a runtime policy enforcement framework for browser extensions. The code integrity checking
3.1. BROWSER EXTENSION SECURITY

Technique blocks the installation of malicious extensions by malware. It can also detect illegal changes made to installed extensions. In their framework, an extension can be installed only after getting a valid authorization from a user. The runtime policy enforcement framework is used to check runtime behaviors of malevolent extensions, and subsequently, prohibit the access of security critical components from extensions. It is observed that their dynamic analysis approach has an overhead of around 17%.

Barua et al. [57] propose a runtime protection mechanism to prevent the JavaScript injection attacks in the Firefox extensions. The proposed approach uses a code randomization technique that transforms the source code of an extension. At runtime, this transformation helps distinguish the extension code from the malicious code because the derandomization technique can only transform back the extension code into valid code, but not the malicious code. To reduce false positives and negatives, the dynamic analysis technique is also integrated with a static analysis technique. The proposed approach has some performance overhead because of both the randomization and derandomization processes.

Hulk [68] is another dynamic security analysis tool. It first creates a dynamic environment to trigger extension behaviors. Then, the technique identifies the potential security issues of browser extensions by observing their runtime behaviors.

Since a dynamic analysis technique checks security issues at runtime, it can easily detect issues injected by remote scripts. For obfuscated extension code, the dynamic analysis approach is the most preferred option. However, there are some disadvantages. Sometimes, it cannot analyze all possible execution paths for all possible input values. From the aforementioned performance overhead information, it can be concluded that dynamic analysis approaches have extra runtime overhead. Hence, this
particular analysis technique is not an ideal option for analyzing extensions in mobile devices because the performance of mobile browsers is a genuine concern for end users. CLOUBEX is mainly built for browser end users. Therefore, the efficiency of the approach is a matter of utmost importance which motivates us to use a static analysis technique.

**Model-based Analysis**

In model-based analysis, first, a set of features from a number of widely used extensions are used to generate different models. Then, an unknown extension is tested against those models to detect the type of the extension. Shahriar et al. [79] propose a model-based detection approach for analyzing browser extension security issues. If models are properly built, then a model-based analysis provides much more precise results than a static analysis. However, building correct models is an error prone and time consuming job. A large number of extensions with varying behaviors is also required to build accurate models. The mobile browser extension field is comparatively new. As a result, we may not find sufficient mobile extension samples to experiment with a model-based security analysis approach.

**3.1.4 Mobile Browser Extensions Security Analysis Solution**

The mobile browser extension security field has not received adequate attention so far. Marston et al. [75] study the security issues of the Firefox for Android extensions. The approach may have some considerable performance impact, as extensions are analyzed in a mobile environment.
The disadvantages of using one of the existing approaches become much more noticeable when we need to consider mobile browser extensions because mobile devices have some resource constraints in terms of computing speed, memory, power, network connection, etc. CLOUBEX is designed for analyzing both desktop and mobile browser extensions. The proposed security analysis framework follows a client-server architecture model and addresses most of the challenges associated with the existing security analysis approaches. In the proposed approach, CPU-intensive security analysis tasks are generally executed in a high-speed server. However, the framework has the flexibility to execute the static analysis tasks either in the client device or in the server based on different conditions, such as a client device’s computational capabilities, network connection speed, and extension size.

3.1.5 A Qualitative Comparison

Because of the unavailability of the source code and required tools, we cannot directly compare our approach with the existing techniques quantitatively. However, in Table 3.1 we depict a qualitative comparison of different security analysis techniques.

3.2 Mobile Anti-virus Tools

Mobile anti-virus segment is still not mature enough because mobile devices have numerous resource constraints. There are only a few mobile anti-virus tools available in the market. Sophos Mobile Security for Android [42] can protect Android devices from malware. In Sophos, there is a cloud scan mode option [43] which enables the scanner to check mobile apps against the latest threat data stored in the Sophos Lab cloud database. However, Sophos is not able to detect new vulnerable extensions.
Table 3.1: A qualitative comparison of extension security analysis techniques.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Security Analysis Technique</th>
<th>For Both Desktop and Mobile Extensions?</th>
<th>Runtime Overhead</th>
<th>Requirement Driven Analysis</th>
<th>Extensible</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEX [55]</td>
<td>Static</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Beacon [69]</td>
<td>Static</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SABRE [60]</td>
<td>Dynamic information-flow-based</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Djeric and Goel [61]</td>
<td>Dynamic taint-based</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SENTINEL [76]</td>
<td>A dynamic policy enforcer</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Wang et al. [85]</td>
<td>A dynamic fine-grained access control framework</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ter Louw et al. [81]</td>
<td>A code integrity checking mechanism and a runtime policy enforcement framework</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Barua et al. [57]</td>
<td>A runtime protection mechanism and a points-to static analysis</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Hulk [68]</td>
<td>Dynamic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Shahriar et al. [79]</td>
<td>Model-based</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Marston et al. [75]</td>
<td>Static</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CLOUBEX</td>
<td>Static</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

On the contrary, CLOUBEX can analyze any type of extension by performing an in-depth analysis of extension code based on user-defined security requirements. Sophos provides a service to its users to upload a suspicious file to the Sophos Lab [44] for further verification. However, there is no well-defined and extension specific user interfaces in which users can define their security requirements. We are also not
3.3. JAVA, PHP, AND OTHER JAVASCRIPT STATIC ANALYSIS

sure whether they examine code manually or not. In the future, mobile anti-virus companies such as Sophos can utilize our tool to analyze extensions more efficiently.

Panda Mobile Security [37] is another popular mobile anti-malware tool. Similar to Sophos, Panda is also ineffective in detecting new unsafe extensions. By using Panda, it is possible to identify apps which use sensitive information such as location information, internet connectivity, and device storage. However, it only informs users that the Firefox for Android application tracks location information. Deciding whether an extension uses some sensitive information is beyond the scope of this mobile anti-virus tool. There are some other commercial mobile anti-virus tools, e.g., AVG mobile [7], Norton Mobile Security [33], and Avast Mobile Security [6]. However, all the mobile anti-virus tools are signature-based and can detect only known vulnerabilities.

3.3 Java, PHP, and Other JavaScript Static Analysis

Static analysis is not only proven to be an effective code inspection technique for extension code, but also it is widely used in several other fields. Benjamin et al. [73] use a context-sensitive pointer analysis to find out vulnerabilities in Java-based web applications. They are able to detect 29 security vulnerabilities which include a number of SQL injections, cross-site scripting, and HTTP splitting attacks. Taint Analysis for Java (TAJ) [82] is a static analysis tool used for large-scale industrial applications. TAJ is able to handle reflective calls, tainted flows through containers, and nested taint. A diverse set of features of the Java Enterprise Edition framework is also supported in TAJ. WALA [48] is used to analyze both Java and JavaScript applications. WALA is enriched with a number of essential features, such as inter-procedural
data flow analysis, context-sensitive tabulation-based slicer, pointer analysis, and call graph construction. Arzt et al. [54] propose a static taint analysis approach to analyze security issues in Android applications.

WebSSARI [64] applies a lattice-based static analysis mechanism to identify insecure information flow in PHP-based web applications. Pixy [67] is a similar type of static analysis tool used to identify cross-site scripting vulnerabilities in PHP scripts. In this tool, a flow-sensitive, context-sensitive, and inter-procedural data flow analysis is used to locate the vulnerable points in a program. Vogt et al. [83] propose a combined dynamic data tainting and static analysis approach to prevent cross-site scripting attacks.

JavaScript is not only the main component of the Firefox extensions, but also extensively used in different web-based applications. Madsen et al. [74] recommend a pointer and use analysis to effectively examine JavaScript code in the presence of framework and libraries. Kashyap et al. [71] show how the precision of a JavaScript static analysis can be improved using type refinement. Many other studies [66, 70, 77, 80] focus on the JavaScript static analysis.

We borrow some concepts from the existing static analysis research work. In our approach, we use a flow-sensitive and context-sensitive static taint analysis to detect malicious and vulnerable extensions. XPCOM provides a wide range of functionalities through its inbuilt interfaces and functions. Therefore, it is equally important to consider that large number of XPCOM interfaces and functions for identifying sensitive source-to-sink information flows. Instead of tightly integrating the XPCOM security model into the analysis component, we define security critical XPCOM sources and sinks in a different configuration file. This flexibility also helps incorporate new
sources and sinks at any point of time with minimal effort.

3.4 Summary

Different security analysis approaches, for example, static analysis, dynamic analysis, and model-based analysis, are proposed by researchers for analyzing desktop browser extension security issues. However, the mobile browser extension security research field is still at a very nascent stage. The existing desktop browser extension security analysis techniques cannot be efficiently applied for mobile browser extensions, as mobile devices have numerous resource constraints in terms of processing speed, storage, power, etc. Existing mobile anti-virus tools are also signature-based and can detect only known issues in mobile browser extensions.

Considering various limitations of mobile devices and the necessity of analyzing unknown extensions, we design CLOUBEX, a cloud-based security analysis framework for both desktop and mobile browser extensions. We use a static taint analysis technique to identify malicious information flows and other security vulnerabilities in browser extensions. Some important concepts of the proposed static analysis technique, such as flow-sensitivity and context-sensitivity, are influenced by the available static analysis approaches used for different programming languages. Additionally, the proposed technique is also enriched with a requirements-driven analysis feature.
Chapter 4

Security Analysis Framework

In the preceding chapter, we discussed the advantages and disadvantages of different extension security analysis approaches. Keeping in mind the deficiencies of the conventional approaches and the various resource limitations of mobile devices, we conceptualize CLOUBEX, a cloud-based extension security analysis framework. CLOUBEX is built on a client-server architecture model. The proposed framework is designed mainly for end users and it can be used for analyzing different types of desktop and mobile browser extensions. In this chapter, we present an architectural overview of CLOUBEX which is followed by the details of the client and server components of the proposed framework.

4.1 Architecture Overview

In CLOUBEX, there are two main components: client and server. The client and server components of CLOUBEX framework are depicted in Figure 4.1. In this framework, a number of mobile and desktop clients are connected with a nearby high-speed server over the internet. In general, the client application relies on the high-speed server for performing compute-intensive security analysis tasks in an efficient way.
Once the analysis tasks are executed by the high-performance computing server, the client application can retrieve the analysis results and display the reports to end users. The client application is also used to define different types of user-specific security requirements. However, it is not always beneficial to execute the tasks in the remote server because of several important factors, such as slow connectivity, large extension size, and users’ individual preferences. To address those particular situations, the proposed framework also provides an option to analyze extensions statically in the client component itself. If a particular extension is already analyzed, then it is advantageous to reuse that analysis reports for subsequent user requests, as the security analysis tasks are always very time consuming. A primary database, located in the same geographic location, is used for storing the analysis results for future references. As a fault tolerant measure, another database is also kept in standby mode. In the future, a cache server can be used to further improve the performance of the framework. All the server-side sub-components are hosted in a secure cloud environment.

The client and server components of the proposed framework are presented in detail in the following sections.

4.2 Client Component

In the proposed approach, there are two different types of clients: mobile client and desktop client. The mobile and desktop client applications are used for finding security issues in mobile and desktop browser extensions, respectively.

Security analysis tasks are always very resource-demanding and time consuming. Hence, for available extensions, the security analysis tasks are executed in a high-speed computing system in an offline mode. As a result, users can access the available
security analysis reports from the client application without waiting for a long analysis time. For mobile devices, this approach provides much more benefits because these devices are not at all compatible with the high-speed server for accomplishing the compute-intensive static analysis tasks.

Security requirements are very much user specific. For example, if a particular user always operates in the private browsing mode, then he or she should not require history or cookie related security analysis. An end user can skip location related security analysis, if the location tracking system is disabled by him or her. In CLOUBEX, end users can specify their security requirements from the client application and security analysis tasks are always executed based on the user-defined security requirements. The requirements-driven analysis also helps reduce security analysis time.

Using vendors provided add-on Software Development Kits (SDK), nowadays any
person can develop new extensions very easily. Therefore, it is not feasible to store security analysis reports for all extensions at the server side. In CLOUBEX, there is an option by which users can upload the newly developed extensions from the client application to the remote server. After uploading, analysis tasks are executed at the server end. Once the analysis tasks are completed, a user is able to check the security analysis reports for that particular extension in the client application.

Mobile internet connection is not always very fast. Sometimes it takes a long time to upload a large extension and in that case, different advantages of executing security analysis tasks in the high-speed server diminish. In modern times, smartphones are also equipped with high-speed computing resources and can handle CPU-intensive tasks very efficiently. Sometimes, users do not want to disclose their security requirements and extension code to the remote server. To address those particular situations, there is an extension analysis module in the client component. By this module, users can examine extensions in the client devices without uploading them to the remote server. The application logic of this extension analysis module would be different for different types of extensions.

Mobile network connection is very erratic in nature. As a fault tolerance measure, if somehow network connection gets disrupted during a remote analysis, then it is possible to switch the analysis mode and execute the analysis tasks in the client component.

To further improve the performance of the proposed framework, a nearest-server-finder module is incorporated in the client part. This module can calculate distances with all the active CLOUBEX servers hosted in different geographic locations based on the pre-configured latitude-longitude information, and subsequently, detect the
closest one. As network latency mostly depends on distance, by using this feature, we can reduce the response time to a great extent.

In CLOUBEX, there are three different analysis modes which are presented in Figure 4.2. If the extension security analysis reports are available at the server end, then users can directly fetch them without executing the analysis tasks again. If the required extension analysis reports are not available at the server end, then users can either upload the extensions to the remote server and execute the analysis tasks in the high-speed server, or can analyze the extensions in the client devices. The following dynamic decision making algorithm helps users identify the best analysis option.

4.2.1 Dynamic Decision Making Algorithm

To further improve the effectiveness of the framework, a dynamic decision making algorithm is incorporated in the client component. This algorithm suggests the best possible analysis option based on some dynamic parameters, such as extension size,
current network connection speed, and the computing speed of a client device.

Before the analysis of actual extensions, both the client and server components analyze some sample extensions, record different execution times, and store the data in configuration files. Here, we assume that the total client analysis time for the sample extensions of total size $S_s$ byte is $T_{cs}$. The total server analysis time for the same sample extensions is $T_{ss}$.

Now, if a new extension of size $S_n$ byte is analyzed completely in the client, then the expected client analysis time $T_{cn}$ can be calculated by determining the runtime complexity of the analysis algorithm and using the analysis time of the sample extensions. If the complexity of the security analysis algorithm is almost directly proportional to the size of an extension, then the following equation can be used to calculate $T_{cn}$.

$$T_{cn} = \left( \frac{T_{cs}}{S_s} \right) \times S_n$$  \hspace{1cm} (4.1)

However, this calculation will vary according to the complexity of the exact analysis logic.

To calculate the time required in the remote analysis mode, we also need to consider the data transfer time. To find out existing network connection speed, this decision making component first sends a small sample extension to the remote server. It is expected that the client network speed does not vary significantly within a short period of time. Hence, to eliminate this extra processing time, the network connection speed information is cached for a certain time period. If the cached bandwidth information expires, then only the decision making component sends the sample extension to the remote server to find out the available network bandwidth.
Here, we assume that the data transfer time for the sample extension of size $S_t$ is $T_t$. Then the data transfer time for the actual extension would be

$$T_n = (T_t / S_t) \times S_n \quad (4.2)$$

Similar to the expected client analysis time, the expected server analysis time $T_{sn}$ for the selected extension can be calculated in the same way

$$T_{sn} = (T_{ss} / S_s) \times S_n \quad (4.3)$$

Therefore, the total remote analysis time would be

$$T_{rn} = T_{sn} + T_n \quad (4.4)$$

Now, if $T_{rn} > T_{cn}$, then the client analysis would be the preferred option. It is very difficult to predict actual execution time of any program. A comprehensive and precise algorithm to predict the actual analysis time will also add significant overhead on the overall performance. Here, our intension is not to predict the actual analysis time, but to augment the usability of the approach. From that perspective, this simple algorithm is sufficient enough to decide which analysis mode would be a better option, when there is a significant time difference between the client analysis and the remote analysis.

### 4.3 Server Component

An extension analysis module also exists in the server component. For available extensions, this analysis module executes analysis tasks statically in an offline mode.
On the contrary, user provided new extensions are analyzed based on user-defined security criteria. However, those new extensions are analyzed again in detail at a later time and results are stored for future references.

To overcome long analysis time, detailed analysis results for a large number of extensions are stored in a database repository at the server side. Often database operations take some time. For enhancing the application performance, an in-memory cache server can be used at the server end.

Different server sub-components are deployed in a cloud environment. The physical location of a particular user may impact the overall response time. To address this problem, the server component is deployed in multiple geographic locations. It helps CLOUBEX users to always find a nearby server application and minimize the data transfer time as much as possible. There are some advantages of using the cloud resources over standalone computing and storage servers. These advantages are discussed in the following subsection.

4.3.1 Advantages of Using Cloud

There are many advantages of using the cloud resources over standalone servers. We can instantaneously get profound computing resources in a secure platform. The high-speed cloud computing resources help to use more precise and intensive security analysis algorithms. A standard cloud computing platform (e.g., Amazon Web Service) also provides a number of options for different compute-optimized servers which are specifically designed for high-performance computing jobs such as static code analysis, machine learning, and image processing. The performance of the proposed framework can be further improved by executing computationally expensive
static analysis tasks in parallel on many nodes.

Currently, a high number of browser extensions are available in the market and that number is growing almost exponentially every day. The cloud platform provides virtually unlimited amount of storage for storing security analysis reports for that large number of extensions. In the future, we can also think of using a cloud NoSQL database (e.g., DynamoDB) for storing the extension security analysis results. The NoSQL database can be expanded both horizontally and vertically without impacting the performance.

Another advantage of the cloud platform is its scalability. The number of users for browser extensions is significantly high. For example, the Adblock Plus Firefox extension has more than 22 million users [2]. If multiple users want to access the security analysis application at the same time, then the cloud platform can automatically provision extra computing resources.

Because of the cloud platform’s pay as you go pricing model, both the solution provider and users can achieve a significant cost advantage. At idle time, extra cloud computing resources can be automatically released. This auto de-provisioning feature helps reduce the cost considerably, since in cloud, users only need to pay for used resources. However, for standalone servers we always need to keep extra resources to meet unexpected demand at peek time. Pricing plays an important factor in our approach as the tool is designed for end users.

The cloud platform provides an option to deploy the server component in different geographic locations. To adhere to different security compliance standards, users may want to upload extensions to a particular location. That flexibility can be achieved by deploying the server-side application in different geographic locations. The global
4.4. SUMMARY

The infrastructure of the cloud platform also helps improve the response time by reducing network latency.

Extensions and users’ sensitive security requirements data can be encrypted both at rest and at transition. The encryption process can be managed both at server and client side. Moreover, well-established cloud platforms are compliant with different industry standard security requirements. We do not need to separately think of those security measures.

In addition, by using the cloud platform, we can get several other general benefits, for example, better infrastructure management, automatic backup and maintenance services, automatic fail-over, customized server configurations, standard security compliances, durability, and reliability.

4.4 Summary

CLOUBEX follows a client-server architecture. Low-performance mobile devices are not ideal choices for executing compute-intensive extension security analysis tasks. Hence, the static analysis tasks are normally offloaded to a nearby high-performance cloud server when users want to analyze extensions from their resource-constrained mobile devices.

In the remote analysis mode, slow network connectivity introduces some challenges from the performance point of view, as extension code also needs to be transferred to the remote server. Sometimes, users do not feel comfortable uploading their security sensitive analysis requirements and extension code to the remote location. To address those particular situations, an extension analysis module is incorporated in the client
application. The client application is also augmented with a decision making algorithm which suggests users the best possible analysis option based on some dynamic parameters, such as the processing speed of a client device and existing network bandwidth. The security analysis tasks are always very much time-consuming. Therefore, to overcome long analysis time, the analysis results are always stored in a database for future references.

All server-side components are hosted in a secure cloud environment. The cloud environment is selected over an in-house option because of several reasons, such as the availability of profound computing resources, unlimited database storage, scalability, and pay as you go pricing model.
Chapter 5

Static Security Analysis of Firefox Extensions

In the previous chapter, we presented the components of CLOUBEX in an elaborated way. In Chapter 2, we discussed why the Firefox extensions are the most unsafe extensions in the extension world. To evaluate the effectiveness of CLOUBEX, we implement the proposed framework for the security analysis of the Firefox extensions. There exist a number of security issues in the Firefox extensions. By using this prototype tool, we try to detect security issues introduced by the XPCOM components. In this chapter, we cover two important topics: the XPCOM related security issues that exist in the Firefox extensions and the static analysis technique used to detect those security issues.

5.1 XPCOM Related Security Issues

Mozilla constantly incorporates changes in the Firefox extension architecture to address different security issues. However, still there are many exploitable areas. It is also important to consider the security issues that existed in the older versions of Firefox because sometimes users are reluctant to upgrade their browsers. Considering the range of functionalities and vast usage, the XPCOM related security issues are the
most dangerous ones. Therefore, for the analysis purpose, we focus on those security issues. The implemented analysis technique performs a comprehensive and granular security analysis of the Firefox extensions.

There are more than 700 in built XPCOM interfaces [50]. Each XPCOM interface again consists of numerous XPCOM functions. Different unsafe interfaces have different types of security problems. Therefore, to clearly understand different types of security issues associated with this large number of XPCOM interfaces and functions, it is imperative to categorize the interfaces and functions from various security perspectives. The XPCOM classification is also necessary to analyze the XPCOM related security issues based on user-defined security requirements.

The classification of XPCOM, based on different security issues, is presented in Figure 5.1. The XPCOM interfaces can be used for accessing sensitive browser information such as login credentials, cookies, sessions, caches, bookmarks, browser history, and logs. They can also change browser security settings. XPCOM can interact with the operating system layer and different device sensors. Sensitive information can be disclosed to attackers by some vulnerable XPCOM interfaces and functions. XPCOM can even change browser user interfaces (UI) and introduce many UI related security issues. The inadvertent usage of the XPCOM interfaces and functions may give rise to other types of security issues as well.

5.2 Security Analysis Procedure

We use a static taint analysis to identify different sensitive source-to-sink information flows in a given Firefox extension. For the static analysis, we consider only the JavaScript files. A Firefox extension is nothing but an xpi formatted zip file. The
5.2. SECURITY ANALYSIS PROCEDURE

Figure 5.1: XPCOM classification based on different security issues.

The analysis application first unzips the xpi file and extracts all the JavaScript files for further processing. The analysis procedure consists of two important steps: read configurations and core analysis. The two important steps of the proposed security analysis technique are presented in the following subsections.

5.2.1 Read Configurations

To identify malicious information flows, the next important task is to define sensitive sources and sinks. In most of the existing research work, some specific source-to-sink information flows are considered. However, the XPCOM components provide a wide range of functionalities. As a result, there exist a large number of security sensitive XPCOM interfaces and functions. Instead of specifying the vulnerable XPCOM interfaces and functions in the application code, we organize, maintain, and control
them through an XML-based configuration file. Reading a file is an expensive process. It adds some overhead on the application response time. Therefore, instead of checking the configuration file multiple times for different functions, the application reads the file at the beginning and stores the information in a data structure for future references.

There are many advantages of using the XML-based security model. Mozilla continuously releases new Firefox versions. By updating the XML file, we can easily incorporate the new XPCOM interfaces and functions and remove the obsolete ones without touching and redeploying the application. The XML-based model is also quite useful for maintaining the earlier discussed XPCOM classification in a structured way. The security analysis application can check only the required categories and provide the analysis reports for specific user-defined requirements. The Firefox for Android extensions do not support all the available XPCOM interfaces (e.g., bookmark interfaces). Because of the usage pattern and mobility factor, some XPCOM interfaces are more vulnerable in the Firefox mobile version. For example, the geolocation related XPCOM interfaces are not security critical for the Firefox desktop extensions, as the location of a desktop is hardly changed over the time. However, in the Firefox for Android mobile extensions, the same location APIs can be exploited to reveal the sensitive location information of a particular user. Hence, it is important to apply different security models. We can achieve this by using two different configuration files for the Firefox desktop and mobile browser extensions.
The entire configuration file is divided into different security categories, such as browser preference, location, file system, and network. The categories are decided by the XPCOM classification model. Next, for each category we define a list of security critical XPCOM interfaces and functions. In the proposed framework, an end user can define his or her security requirements for the security analysis of a particular extension. The categorization of the XPCOM interfaces and functions helps filter the functions at the very first level and apply the analysis logic based on the user-defined security requirements. Different XPCOM functions need to be analyzed in different ways which are determined by different XML configuration values (All possible XML configuration values are listed in Table 5.1). In the following paragraphs, we elaborate the structure of the XML configuration file with some sample code snippets.

**Taint Analysis Related Configurations**

For the static taint analysis, it is necessary to define both sensitive sources and sinks. The value of the XML element `AnalysisType` is set to `TaintAnalysis` for all those XPCOM functions for which the taint analysis is required.

The XPCOM functions which are used to retrieve confidential information, for example, `getAllLogins`, are identified by the `XPCOMFunctionType` element value `SensitiveSource`. Sometimes, it is required to check function input parameters along with a function name, to decide whether the particular function is really used to fetch critical information or not. For instance, the Java method `checkPrefMethods` specified in the `CheckingMethod` element fulfills that extra analysis for the XPCOM function `getCharPref`. The required configurations for the `getAllLogins` XPCOM function is given below.
5.2. SECURITY ANALYSIS PROCEDURE

For all the sensitive sink related XPCOM functions, the XPCOMFunctionType value is set to SensitiveSink. During the analysis, it is checked whether any tainted values are disclosed to an untrusted recipient through the sink functions. A Java method, defined in the element CheckingMethod, takes a list of tainted variables as inputs and fulfills the required analysis. The open XPCOM function of the nsIXMLHttpRequest interface can be used to send information to a remote host through the HTTP GET protocol.

Security Vulnerability Related Configurations

There exist a number of XPCOM functions which are not directly relevant from the taint analysis perspective, but indirectly responsible for introducing different kinds of security vulnerabilities. Those functions are recognized by the AnalysisType element value VulnerabilityAnalysis. The resetPrefs function of the nsIPrefService XPCOM interface brings back the entire Firefox browser preference system to its initial state. This function may delete some security preferences set by a particular user. Similar to the taint analysis related XPCOM functions, for many vulnerable functions,
it is not adequate to consider only function names. The `setIntPref` function of the `nsIPrefBranch` interface is used to set a particular integer preference value. However, all integer preferences are not important from the security perspective. Therefore, it is equally essential to check the preference name for the `setIntPref` function. For such type of XPCOM functions, the `ExtraChecking` configuration value is set to `Y`. The method responsible for accomplishing that extra checking is mentioned in the `CheckingMethod` element. For all those vulnerable XPCOM functions, the analysis component either directly throws an alert specified in the `UserMessage` XML element or produces a custom message generated by the application.

```xml
<XPCOMFunctionName>setIntPref</XPCOMFunctionName>
<AnalysisType>VulnerabilityAnalysis</AnalysisType>
<ExtraChecking>Y</ExtraChecking>
<CheckingMethod>checkPrefMethods</CheckingMethod>
<XPCOMFunctionType>ChangeSecuritySetting</XPCOMFunctionType>
<UserMessage>NA</UserMessage>
```

There are many XPCOM functions which do not have direct impact over users. However, in the future, those functions may bring some vulnerabilities when used with some other malicious extensions. The `addLogin` function of the `nsILoginManager` XPCOM interface is used to store a login information in the browser login manager. This particular method neither discloses any sensitive information nor changes any security setting of the Firefox browser. However, the login information is shared across the browser and is accessible by other extensions as well. A malevolent extension can retrieve that critical login information stored by a benign extension. In addition to detect malicious information flows, the analysis component warns users about the usage of such risky functions. For this purpose, for each category, we keep a list of potential unsafe XPCOM functions in the configuration file.
5.2. SECURITY ANALYSIS PROCEDURE

<XPCOMFunctionName>addLogin</XPCOMFunctionName>
<AnalysisType>Vulnerability Analysis</AnalysisType>
<ExtraChecking>N</ExtraChecking>
<CheckingMethod>NA</CheckingMethod>
<XPCOMFunctionType>StoreSensitiveInformation</XPCOMFunctionType>
<UserMessage>Adding login info by addLogin is vulnerable for future activities</UserMessage>

Internal Processing Related Configurations

There are few XPCOM functions which are not directly liable for any security issues. However, those functions need to be reviewed for the analysis of other XPCOM functions. The <getBranch> function of the <nsIPrefService> interface is used to get a preference branch rooted at a specific node from the entire preference tree. If the branch root node name is passed as an input in the <getBranch> method, then for all the subsequent operations it is sufficient to mention only the child suffixes without giving the root node name. This <getBranch> function needs to be examined carefully to keep track of all the branch root nodes. As a result, the analysis component correctly identifies the preference names for the following preference related XPCOM functions. For those functions, the analysis procedure is controlled by a unique identifier value added in the <XPCOMFunctionType> element.

<XPCOMFunctionName>getBranch</XPCOMFunctionName>
<AnalysisType>Internal Processing</AnalysisType>
<ExtraChecking>N</ExtraChecking>
<CheckingMethod>NA</CheckingMethod>
<XPCOMFunctionType>SetPrefBranchValue</XPCOMFunctionType>
<UserMessage>NA</UserMessage>
Table 5.1: List of possible XML security configuration values.

<table>
<thead>
<tr>
<th>XPCOM Element Name</th>
<th>Possible Values and Their Significances</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPCOMInterfaceName</td>
<td>Any valid XPCOM interface name</td>
<td>nsILoginManager</td>
</tr>
<tr>
<td>XPCOMFunctionName</td>
<td>Any valid XPCOM function name</td>
<td>getCookieString (nsICookieService)</td>
</tr>
<tr>
<td>AnalysisType</td>
<td>TaintAnalysis denotes that the function is important for the static taint analysis.</td>
<td>getCookieString (nsICookieService)</td>
</tr>
<tr>
<td></td>
<td>VulnerabilityAnalysis means that the function may introduce security vulnerabilities.</td>
<td>setCharPref (nsIPrefBranch)</td>
</tr>
<tr>
<td></td>
<td>InternalProcessing signifies that the function demands an extra processing for the security analysis of other XPCOM functions.</td>
<td>getDefaultBranch (nsIPrefService)</td>
</tr>
<tr>
<td>ExtraChecking</td>
<td>Y stands for the invocation of an extra method for the analysis.</td>
<td>getCharPref (nsIPrefBranch)</td>
</tr>
<tr>
<td></td>
<td>N means that no extra method is required.</td>
<td>getCookieString (nsICookieService)</td>
</tr>
<tr>
<td>CheckingMethod</td>
<td>Any valid Java method name that is used for the extra analysis of the specific XPCOM functions. This value is set, only when the ExtraChecking value is set to Y.</td>
<td>The checkPrefMethods Java method is used for preference related analysis.</td>
</tr>
<tr>
<td>XPCOMFunctionType</td>
<td>For TaintAnalysis, SensitiveSource and SensitiveSink are used to define security critical source and sink functions, respectively.</td>
<td>SensitiveSource and SensitiveSink are mentioned for searchLogins (nsILoginManager) and send (nsIXMLHttpRequest), respectively.</td>
</tr>
<tr>
<td></td>
<td>For VulnerabilityAnalysis, different identifiers, such as ChangeSecuritySetting and StoreSensitiveInformation, are used.</td>
<td>ChangeSecuritySetting is specified for setIntPref (nsIPrefBranch).</td>
</tr>
<tr>
<td></td>
<td>For InternalProcessing, some specific identifiers are used to decide the course of actions.</td>
<td>A unique identifier SetVulnerableFile is used for initWithPath (nsIFile) to track all file related variables used for sensitive file locations.</td>
</tr>
<tr>
<td>UserMessage</td>
<td>A user friendly message that is displayed to a user to notify about the security issues.</td>
<td>&quot;The resetPrefs is vulnerable, as it deletes all user specific security settings.&quot; - message is used for resetPrefs (nsIPrefService).</td>
</tr>
</tbody>
</table>
5.2. SECURITY ANALYSIS PROCEDURE

5.2.2 Core Analysis

In this subsection, we mainly discuss the steps and algorithm of the core analysis. There are three main steps in the core analysis part. First, the analysis component generates a call graph and a program summary graph (PSG) for a JavaScript file. In the following two steps, all the XPCOM interfaces and functions, used in that particular extension code, are identified and analyzed based on the security policies defined in the XML configuration file.

(i) Generation of Call Graph and Program Summary Graph (PSG)

An extension JavaScript file normally consists of multiple functions. Hence, for an inter-procedural extension analysis, it is indispensable to establish connections among
all existing functions as well as actual, formal, and return parameters. At the beginning of the core analysis, the application traverses all nodes and builds a call graph for the JavaScript file. A call graph represents only relations between called and callee functions. Therefore, after completing the traversal, the analyzer forms a detailed program summary graph from the call graph and available function definitions. The generated program summary graph helps find out the XPCOM functions where the aliases of the actual XPCOM interface variables are used to invoke the functions.

(ii) Detection of XPCOM Interfaces

The application detects the XPCOM interfaces by checking the most common syntaxes used for the XPCOM instantiations [51].

In the first example, initially, a `file` component is created and stored in a variable named `aFile`. A component consists of multiple interfaces. To invoke an XPCOM function, the next task is to create an instance of the particular XPCOM interface. In the second line, an object of the `nsILocalFile` XPCOM interface is instantiated from the `file` component. After the second line, the `aFile` variable refers to the `nsILocalFile` interface, not the entire `file` component.

Instead of first creating an XPCOM component and then querying an interface, the XPCOM interface can be directly created in a shortcut way which is represented by the second example.

If a developer is not sure that an interface is supported by a particular component, then instead of using the `QueryInterface` method, the developer should use the `instanceof` operator to prevent an exception. If a component does not implement an interface, then the `instanceof` check returns false. For a supported interface, the
instanceof operation returns true, and as a side effect the variable also refers to the XPCOM interface afterwards. This type of interface instantiation pattern is depicted in the third example.

There exist a number of special XPCOM components which are called services. At a time, only one service component should exist. Therefore, instead of invoking the createInstance method, the getService method needs to be called to get a reference of a service component. As per the fourth example, the consoleService variable refers to the nsIConsoleService XPCOM interface which is a part of the console XPCOM service.

1. var aFile = Components.classes[‘@mozilla.org/file/local;1’].createInstance();
   if (aFile) aFile.QueryInterface(Components.interfaces.nsILocalFile);
2. var bFile = Components.classes[‘@mozilla.org/file/local;1’].createInstance(Components.interfaces.nsILocalFile);
3. var cFile = Components.classes[‘@mozilla.org/file/local;1’].createInstance();
   if (cFile instanceof Components.interfaces.nsILocalFile) {
     // do something
   }
4. var consoleService = Components.classes[‘@mozilla.org/consoleservice;1’].getService(Components.interfaces.nsIConsoleService);

The application not only detects the XPCOM interfaces, but also keeps track of all the JavaScript object reference variables and their aliases (e.g., in the above examples the JavaScript object reference variables are aFile, bFile, cFile, and consoleService) associated with the XPCOM interfaces at each program point.
(iii) Identification and Analysis of XPCOM Functions

Once an XPCOM interface is identified, the next important analysis task is to detect the XPCOM functions. Here, we first mention the common syntax used by the analyzer to identify different XPCOM functions. The function analysis algorithm is elaborated in the next few paragraphs. To invoke an XPCOM function, the following syntax is used.

\[ i.f(p1, p2, \ldots) ; \]

Here, \( i \) represents an XPCOM interface object reference variable or its alias and \( f \) denotes an XPCOM function of the associated XPCOM interface. The XPCOM function may contain a number of input parameters which are represented by \( p1, p2, \) etc.

For different XPCOM functions, the analysis component takes different actions. The main algorithm for finding the security issues and vulnerabilities of an extension is depicted in Figure 5.2.

The core analysis component considers mainly three types of XPCOM functions. The XPCOM functions which are relevant from the static taint analysis perspective are identified by the \texttt{AnalysisType} element value \texttt{TaintAnalysis}. There exist a number of functions which are not important for the taint analysis, but responsible for introducing different security vulnerabilities. Those function are recognized by the \texttt{AnalysisType} element value \texttt{VulnerabilityAnalysis}. Some functions are neither required for the taint analysis nor liable for any security vulnerabilities, but demand some extra processing for the analysis of other vulnerable functions. Those functions are marked by the \texttt{AnalysisType} element value \texttt{InternalProcessing}. At the first stage, the analysis component checks the \texttt{AnalysisType} element value associated
with a detected XPCOM function to decide the next course of actions.

For a taint analysis related XPCOM function, the application then checks the XPCOMFunctionType element value of the function. If the value is SensitiveSource, then the associated left hand side variable is tainted. If the value is SensitiveSink, then a different method is invoked to detect whether the tainted information is divulged to any untrusted recipient by the corresponding function. The list of tainted values is passed as an input parameter to the analysis method. If the method finds any privacy violation, then it reports back the malicious flow to the end user.

For other non-taint related vulnerable XPCOM functions (i.e., AnalysisType value VulnerabilityAnalysis), the analysis algorithm first determines whether an extra level of checking is required for that particular XPCOM function. If the ExtraChecking element value is Y, then the application calls a Java method which is defined in the CheckingMethod element. That extra method helps to identify whether the function is really abusive or not. The security analysis is context sensitive in nature, as it also checks function parameters for the relevant XPCOM functions. If the ExtraChecking element value is N, then the analysis component directly warns about the security vulnerability by a user-friendly message specified in the UserMessage element.

Finally, for the internal processing related XPCOM functions, the analysis component makes a decision about the next processing logic based on a unique identifier defined in the XPCOMFunctionType element.
5.3 Important Security Analysis Characteristics

The main characteristics of the aforementioned static analysis are taint analysis and requirements-driven analysis. In the following subsections, we discuss these two important characteristics in a detailed manner.

5.3.1 Taint Analysis

We use a taint analysis procedure to identify malicious source-to-sink information flows in the Firefox extensions. In Subsection 5.2.2, we discussed how the XPCOM functions are analyzed based on the XML security configuration values. In this subsection, we elaborate the static taint analysis mechanism. During the static analysis, the application only considers explicit data flows, but not implicit control flows. Considering the current nature of the Firefox extensions, it is a quite reasonable approach to monitor only the explicit data flows. The comprehensive taint analysis is accomplished by a flow-sensitive forward analysis technique.

If an XPCOM function is used to retrieve information from a critical source, then the associated JavaScript object is tainted. The criticality of a particular source is determined by the analysis technique and user-defined security criteria. For example, the login manager component is always treated as a security sensitive source. On the other hand, when an XPCOM function is used to get information from the file system, then user-defined security critical directories are also considered to decide whether the related object needs to be tainted or not.

If an XPCOM function returns a complex security sensitive object, then all properties of that object are conservatively treated as tainted. The getAllLogins method of the nsILoginManager interface returns an array of nsILoginInfo objects. The
nsILoginInfo object consists of multiple properties, such as username, hostname, and password. However, instead of considering only the password field, the application taints the entire nsILoginInfo object when the getAllLogins function is used.

The analysis application not only taints and maintains a list of tainted object variables at each program point, but also captures the propagation of all the tainted variables through program variables and functions. In assignment statements, the left-hand side variable is tainted if any of the right-hand side operands is tainted.

The analysis component considers all elements of an array as tainted, when the particular array is tainted. In a similar way, if a taint value is assigned to an array element, then from that program point the entire array is treated as tainted. Reassigning a non-critical value or using a new expression for a tainted variable results the removal of taint signature from that particular reference variable and all its properties.

The application also checks the program summary graph to capture the associated formal parameters which need to be considered for the inter-procedural source-to-sink information flow analysis. Similar to the formal parameters, the function return parameters are also examined for the analysis purpose.

Finally, if the analysis application finds that any tainted information is disclosed to an untrusted recipient through an XPCOM sink function, then it reports the malicious information flow to end users. To identify whether a specific remote host is trusted or not, the analysis application maintains a list of blacklisted URLs. Additionally, users can also blacklist remote hosts from the client application. In the next subsection, we discuss the requirements-driven analysis feature in detail.
5.3. IMPORTANT SECURITY ANALYSIS CHARACTERISTICS

5.3.2 Requirements-driven Analysis

Most of the conventional extension security analysis approaches follow some predefined generic security policies. However, security requirements are always very much user specific. A user who is always concerned about his or her private location information and switches off the location device, may not want to spend time analyzing location related extension security issues and vulnerabilities. If a user always operates in the private browsing mode, then cookie related security analysis is irrelevant for him or her. A browser preference value which is critical for one user, may not be relevant for another user from the security perspective because of the differences in their browsing behaviors. In CLOUBEX, there is an option by which an end user can specify broad level security requirements, such as cookie, location, and bookmark. For the analysis of an extension, only the user-defined security categories are considered. This requirements-driven security analysis not only reduces the security analysis time, but also produces the analysis report in a more compact and precise format.

Unlike other traditional approaches, the proposed technique also provides many other user specific options to achieve a more granular level analysis. It is essential to check whether any sensitive information is leaked to a remote host. The URL checking is also important to identify whether any malware is downloaded from an unsafe remote repository. However, not all remote URLs are unsafe. Through the client application, a user can whitelist and blacklist remote URLs based on his or her previous knowledge and personal choices. For a detailed function argument level analysis, a user can also define the names and types of security related information such
as important browser preferences and sensitive file repositories. All those user preferences are considered during the course of analysis. To improve the usability of the application, different security analysis options are prepopulated with standard values. However, users can always overwrite those values according to their requirements.

5.4 Analysis Challenges and Remedies

JavaScript is an object-based and prototype-based dynamic language. In general, there are many challenges in analyzing a JavaScript file. However, all of them are not relevant for the proposed analysis technique. The remaining challenges are overcome in some alternative ways. In the JavaScript code, object properties can be created and deleted based on demand. This issue is resolved conservatively by considering all properties as tainted when the associated object is sensitive in nature.

The `eval` function is extensively used in the JavaScript code. A user can dynamically inject executable malicious JavaScript instructions through the `eval` function. Although the usage of the `eval` function is not recommended in extension code from the security perspective, but still it is permitted. The main objective of this analysis is to identify the XPCOM related malicious flows and security vulnerabilities that exist in extension code. Therefore, the analysis of the `eval` function does not demand special attention.

The JavaScript language uses a large number of external APIs, e.g., DOM APIs and browser APIs. For the analysis of the Firefox extensions, we use a standard parser which provides us important low-level information such as the abstract syntax tree (AST) of the JavaScript file. This parser supports the latest ECMAScript standard libraries. Hence, in the future, it would not be a very difficult task to incorporate
the analysis of different external APIs, and subsequently, enhance the precision of the analysis technique.

5.5 Summary

We implement a static analysis technique to examine security issues in the Firefox for Android mobile and Firefox desktop browser extensions. The Firefox extensions have different kinds of security problems. We focus on detecting the XPCOM related security issues because the over-privileged XPCOM components can introduce many direct and indirect security threats.

The static analysis technique is controlled through an XML-based configuration file. The XML-based configuration file provides many advantages, such as the better administration of the Firefox extension security model.

The proposed analysis technique has two important steps. After extracting the JavaScript files, the analysis application reads the configuration file and stores the values in a data structure for future references. At the final stage, it detects the XPCOM interfaces and functions, and verifies those against the security related configuration values. If the application finds any malicious information flows or any potential security vulnerabilities, then it reports back to end users. The main characteristics of this static analysis approach are taint and requirements-driven analysis.
Chapter 6

Implementation and Evaluation

The main objective of our research is to come up with a comprehensive security analysis framework for both desktop and mobile browser extensions. We implement CLOUBEX framework for the security analysis of the Firefox extensions. To assess the effectiveness of the proposed framework, we analyze a number of Firefox for Android mobile and Firefox desktop extensions through various experiments. In this chapter, we first discuss the implementation part. Next, we describe the setup used for the experiments. Finally, we present and analyze the experimental results in a detailed manner.

6.1 Implementation

In this section, we cover three important topics. First, we discuss the implementation details of the core CLOUBEX components. Then, we explore the technologies and platforms used to implement the static security analysis technique. Till now, there is no standard security evaluation benchmark for the Firefox for Android extensions. Therefore, for the evaluation purpose, we develop a set of malicious Firefox for Android extensions which are presented in the last subsection.
6.1. IMPLEMENTATION

6.1.1 CLOUBEX

In CLOUBEX, there are two main components: client and server. The implementation details of the client and server components are described in the following paragraphs.

Android and Swing-based Client Component

An Android application is used as the mobile client. A client-server communication is accomplished using the Java Socket technology. If a particular remote analysis is not completed within a certain time due to slow network connectivity, then SocketTimeoutException occurs in the client application. As a fault tolerance measure, after such type of timeout exception, the analysis tasks are executed by an extension security analysis module integrated with the client application. The user interfaces of the Android application are controlled through different layout XML files.

In the Android client component, there is a utility feature by which the client application can choose the nearest server. It helps reduce the data transfer time and improve the overall application performance. The latitude and longitude information of the available cloud servers are preconfigured in the Android application. After fetching its own latitude and longitude information using the Android location APIs, the client component calculates the server distances using the Haversine formula \[23\] and finds the nearest server.

Similar to the Android mobile client application, a Java Swing-based desktop client is used for the Firefox desktop browser extensions. For better usability, this Swing application is wrapped within an exe file. A cross-platform software named
launchoj is used for wrapping the Swing-based Java application.

The security analysis algorithm parses all the JavaScript nodes returned by the parser. The number of the nodes is almost proportional to the size of the JavaScript files. As a result, the security analysis execution time is more or less directly proportional to the size of the JavaScript files. We also verify this by executing the security analysis logic on different sample extensions. Hence, the dynamic decision making algorithm can predict the security analysis execution time for an unknown extension from the size of the JavaScript files and the average analysis time of sample extensions using Equation 4.1. For this reason, before the first analysis, the client application analyzes some sample Firefox extensions and records execution times using Java’s standard library functions in a text file. Sometimes, the analysis execution time not only depends on the number of the JavaScript nodes, but also on other factors such as the type and complexity of the nodes. Therefore, to improve the precision of the decision making algorithm, we use a number of different sized sample extensions. The client application also sends a sample extension file to the remote server to find out existing network connection speed. Finally, the algorithm recommends the best analysis option based on the computing speed of the client and the existing network bandwidth.

**Amazon Cloud-based Server Component**

A Java-based application always runs in a high-speed computing server. After an analysis, the extension security analysis reports are also stored in a MySQL database using the Java Database Connectivity (JDBC) adapter. The server-side components are hosted in the Amazon Web Services (AWS) cloud. A compute-optimized Elastic
Implement Cloud (EC2) server instance and a Relational Database Service (RDS) MySQL instance are used as the computing server and database server, respectively. We choose the compute-optimized option because the static analysis algorithm is compute-intensive in nature.

6.1.2 Static Analysis Component

For the static analysis of the JavaScript files, we use a standard parser, named Rhino [41]. There exist a couple of reasons behind selecting Rhino. Rhino is a JavaScript parser, written completely in Java. The thin client application is Android-based which supports the Java APIs. Hence, we can use Rhino for building both the client and server analysis modules. The latest version of Rhino supports both the ECMAScript libraries and JavaScript 1.8 language. Therefore, it can correctly parse the Firefox extension JavaScript files. If Rhino does not support some modern syntaxes, then it is possible to modify the existing implementation of Rhino and incorporate new features, as Rhino is an open source tool. We prefer Rhino over other more complicated JavaScript parsers because the performance of Rhino is not an impending factor for the thin client Android application.

Rhino provides an abstract syntax tree (AST) on which we can build any type of sophisticated analysis logic. The visit function of the Rhino parser is used to traverse all nodes of the AST in a depth first manner [38]. During the traversal process, the analysis component analyzes each node based on its type. Finally, from the node information, the application detects the security issues using the comprehensive algorithm discussed in the previous chapter.
6.1. IMPLEMENTATION

6.1.3 Malicious Firefox for Android Extensions

There is no standard security evaluation benchmark for the Firefox for Android extensions. Therefore, to check how accurately and effectively the analysis component can detect different malicious source-to-sink information flows, we develop a number of Firefox for Android extension malware. We use a large number of security critical XPCOM interfaces and functions in the developed extensions (the details are provided in Appendix A).

Each extension contains two files: `install.rdf` and `bootstrap.js` [20]. The `install.rdf` file has different meta-data information, whereas the `bootstrap.js` file contains the actual extension code. After the implementation, we test each extension in a Firefox for Android browser. The XPCOM components provide a large number of interfaces and functions which can be exploited in different ways. Mainly keeping in mind the criticality of the XPCOM interfaces and functions, and their impact on a mobile user, we build the following extensions.

**Preference Explorer:** Mozilla provides a number of XPCOM APIs to control the Firefox browser preferences or properties. As those properties are the backbone of the Firefox browser, numerous security vulnerabilities can be introduced by using different preference related XPCOM interfaces and functions in an unsafe way. Sometimes, benign extensions store sensitive data such as a user’s individual choices as browser preferences. A malevolent extension can retrieve and leak those information to an untrusted user by utilizing the preference functions. Different preference related XPCOM interfaces and functions are explored in the Preference Explorer extension.
Login Manager Plus: Nowadays, a user needs to remember numerous username and password information for different websites. There exist a number of Firefox extensions, for example, Mobile Password Manager [30] and LastPass Password Manager [28], which provide users conveniences by storing confidential user credentials in the browser’s login manager component and populating those information in required fields when required. However, the stored information is not extension specific. An unsafe extension can always fetch the sensitive login information stored by some other benign extensions. Therefore, the login manager related XPCOM interfaces and functions always require intensive analysis from the security point of view. For mobile devices, those XPCOM functions are much more critical because the lack of a keyboard allures mobile users to use password extensions more frequently than desktop users. Different login manager specific XPCOM interfaces and functions are investigated through the Login Manager Plus extension.

Cookie Controller: In general, an HTTP cookie is used to store stateful information in a browser. For mobile websites, cookies are extensively used to provide quality services as users do not need to enter same information again and again. Similar to login information, cookie data can also be shared across the browser. The Cookie Controller extension is developed to analyze vulnerable cookie related XPCOM interfaces and functions.

File Administrator: In recent times, it is a very common practice to store sensitive files in mobile devices. The Bring Your Own Device (BYOD) policy encourages mobile users to use and store important corporate information in the mobile file system. XPCOM is highly privileged with a capability of accessing critical information from
the mobile file system. Different file related security sensitive XPCOM interfaces and functions are considered in the File Administrator extension.

**Location Finder:** Mobile devices are equipped with location-based sensors which enable users to use different types of location-based services. There are some location specific XPCOM interfaces and functions which can be misused to fetch a user’s current location information. The Location Finder extension is developed to showcase that XPCOM feature.

To understand how a Firefox for Android extension discloses critical information to an untrusted remote host, it is equally important to develop the sink component. The Firefox for Android extensions can transmit information through different channels, e.g., HTTP and socket. To explore the data transmission through HTTP and socket, we develop a servlet and a socket application, respectively. The servlet application can communicate using the HTTP GET and POST protocols.

### 6.2 Experimental Setup

The performance of the security analysis framework is strongly influenced by a mobile device’s computing speed. Nowadays, mobile processor speed normally varies from 1GHz to 2.3GHz. To evaluate how a mobile device’s computational capabilities can regulate the overall performance of the framework, we use two different mobile devices: a medium-speed Moto G and a low-speed HTC Desire. The Moto G mobile phone is equipped with a 1.2GHz quad core ARM Cortex A-7 CPU, 1GB RAM, and Android 4.4.4. The HTC Desire mobile phone has 1GHz Qualcomm Scorpion CPU, 576MB RAM, and Android 2.3.
In the proposed approach, the speed of network communication between a client and a server plays a crucial role. In our experiments, we use two different network connections: a wireless connection available on campus at Queen’s University, Canada, and a 3G connection provided by Wind Canada. The bandwidths of these networks are measured using speedtest.net. The network connection speed of Queen’s University’s network is around five times higher than that of the Wind’s network.

Amazon provides different types of server instances, such as general purpose, compute-optimized, memory-optimized, storage-optimized, and GPU. The security analysis technique is a compute-intensive task. Therefore, a compute-optimized c4.large EC2 instance [4] is used as the computing server. We choose the c4.large instance because it is the cheapest option in the compute-optimized category and at the same time efficient enough to handle intended security analysis tasks. A db.t2.micro RDS MySQL instance [5] is used as the database repository. In the prototype implementation, we keep only a few security analysis reports in the database. Hence, the free-tier db.t2.micro RDS instance can solve our purpose. However, in the future, a more sophisticated RDS instance can be leveraged based on the actual storage requirements. Both the computing server and database repository are located in the Oregon region. To evaluate how a distance factor can control the overall performance of the framework, we use servers located in the Singapore region, too.

The latest Rhino version is 1.7.7.1. This particular version does not support the JavaScript `arrow` function which is extensively used in the Firefox for Android extensions. However, the code available in the master branch of the Git repository supports this feature. Therefore to resolve this issue, we take the Rhino code from the master branch and build a jar from that codebase.
6.3. EXPERIMENTS WITH FIREFOX ANDROID EXTENSIONS

Android’s standard library functions and Qualcomm’s Trepn profiler are used for profiling the client application. Java’s standard library functions are used for profiling the server application.

6.3 Detection Experiments and Results for Firefox Android Extensions

Compared to the Firefox desktop extensions, the Firefox for Android extensions are relatively new in the browser extension field. There neither exists a security evaluation benchmark nor a blocked add-ons repository for the Firefox for Android extensions. To address this issue, we develop five malicious Firefox for Android extensions. To evaluate the effectiveness of the analysis technique, we carry out the experiments with 25 Firefox for Android extensions which include these five malware and 20 most popular Firefox for Android extensions taken from Mozilla’s official extension gallery[32].

The Rhino parser cannot parse Adblock Plus, Lastpass Password Manager, Adblock Edge, Adguard Adblocker, and Gtranslate. There are some syntax errors in Adblock Plus, Adblock Edge, and Gtranslate. The Rhino parser cannot parse the lastpassext JavaScript file of the Lastpass Password Manager extension because the file is totally obfuscated. Three JavaScript files of the Adguard Adblocker extension cannot be parsed because those files start with a byte order mark which is considered as an illegal character by the Rhino parser. The 80% parsing success rate and the nature of errors imply that Rhino is an optimum choice.

The taint-based static security analysis technique not only helps us detect malicious information flows, but also produces first-level warnings for many security vulnerabilities that exist in different official Firefox for Android mobile browser extensions. After executing the automated analysis technique, we manually verify the
relevant parts of the extension code. The manual inspection of the extension code confirms that no false positive alarm is generated by the analysis technique.

### 6.3.1 Detecting Malicious Information Flows

We execute the detection experiments with all the Firefox for Android extension malware that we build for the evaluation purpose. First, we analyze the experimental results for the Login Manager Plus extension in an elaborated way. A code snippet of this extension is presented in Figure 6.1. Here, the `getAllLogins XPCOM` function of the `nsILoginManager XPCOM` interface is used to fetch all login information stored in the browser login manager component. Then, the critical login information is divulged to an untrusted remote server by the `send` function of the `nsIXMLHttpRequest` interface. During examining this extension code, the analysis component first taints the variable `logins`, as in the configuration file the `getAllLogins XPCOM` function is considered as a taint related function. As per the taint analysis logic, each and every element of an array is tainted, if the array itself is tainted. Following this assumption, the security analysis mechanism taints the variable `allLogins`. For this particular extension, the analysis component reports a malicious source-to-sink information flow because the tainted `allLogins` value is leaked to an untrusted server. The remote server is blacklisted by us before executing the analysis algorithm. This experiment demonstrates how the analysis component can taint a sensitive object variable and generate an alert based on some user-defined security criteria.

The other four malicious extensions are developed to investigate how cookie, file, location, and preference related sensitive information can be retrieved by different XPCOM interfaces and functions. In all these extensions, the sensitive information
Figure 6.1: Code snippet taken from Login Manager Plus.

is sent to a remote server using different network channels and protocols, such as HTTP GET, HTTP POST, and socket. In all the circumstances, the security analysis technique can successfully identify and report the malicious flows.

We develop the File Administrator extension using multiple JavaScript functions. This extension uses a large number of XPCOM interfaces. Therefore, for better maintainability, we encapsulate all the XPCOM interface instantiation code in one function and then pass the XPCOM interface objects to a different function as input parameters. Before accomplishing the actual security analysis tasks, the application creates a program summary graph for performing an inter-procedural analysis. As a result, the analysis technique can correctly taint and detect the unsafe information flow that exists in the File Administrator extension.
We also check the analysis reports for different user preferences. For example, for the Location Finder extension, the analysis application triggers a security alarm only when the location preference is selected through the client application user interface. On the contrary, if a location device is switched off, then the location related security analysis can be skipped by not selecting the particular location checkbox.

In Chapter 3, we discussed that most of the mobile anti-virus tools are signature-based and cannot identify unknown security issues. To justify our claim, we first keep all the Firefox for Android extension malware, that we develop for the evaluation purpose, in an Android device. Then, we scan the device by two popular mobile anti-virus tools: Sophos and Panda. The mobile anti-virus tools cannot detect any security issues in these malware. On the contrary, using the implemented static analysis technique, we can successfully identify malicious information flows in all the malware.

6.3.2 Identifying Vulnerabilities in Official Extensions

Mozilla does not maintain any repository for the blocked Firefox for Android addons. It is assumed and highly expected that the official extensions do not have direct security impact over users, as those are validated before available in the repository. The framework is designed mainly for end users. We select 20 most popular Firefox for Android extensions from Mozilla’s official extension gallery and execute the security analysis technique on all of them. Our security analysis approach finds many indirect security vulnerabilities which are presented in the following paragraphs.

**Fingerprinting:** Quite often, extensions set HTTP request headers to accomplish their intended functionalities. By capturing and examining specific request header
6.3. EXPERIMENTS WITH FIREFOX ANDROID EXTENSIONS

values, it is possible to fingerprint extensions and identify a particular user’s browsing behaviors. In today’s world of advertisements, fingerprinting is a big security concern for all of us. The `setRequestHeader` XPCOM function of the `nsIXMLHttpRequest` XPCOM interface is extensively used in many Firefox for Android extensions, such as Flash Video Downloader, Https Everywhere, and Youtube Downloader. The `logStringMessage` function of the `nsIConsoleService` interface is used in Flash Video Downloader, Https Everywhere, and Youtube Downloader, to log important messages in the browser console. In the future, other malicious extensions can retrieve those important log messages and use the messages for fingerprinting.

**Shared information related:** It is not always safe to add cookie and login information in browser because the information is shared across the browser space and may be accessed by some malicious extensions. In the Self Destructing Cookies and Https Everywhere extensions, the `add` function of the `nsICookieManager2` interface is used to add critical cookie information. In Mobile Password Manager, the `addLogin` method of the `nsILoginManager` interface is used to store sensitive user login credentials.

**Preference related:** Properties are the backbone of both the Firefox browser and extensions. Resetting a critical browser property may introduce some sort of security vulnerabilities in a user’s future browsing activities. It is observed that the analysis application can detect different types of property setting functions, such as `setBoolPref`, `setCharPref`, and `setIntPref` in few extensions, for example, Flash Video Downloader and Youtube Downloader. It cannot generate any alarm because the properties used in these extensions are mainly extension specific and not important
from the security point of view. However, the detection of these sensitive functions justifies that the application is capable of finding preference related security vulnerabilities. It highlights another important aspect of the analysis technique. If the application throws alerts only based on the usage of the XPCOM functions, it may generate many false positives. We reduce such type of false positives by applying a more granular function argument level analysis. Similar to preferences, categories are also widely used throughout the Mozilla platform. The analysis application can successfully identify several XPCOM functions of the nsICategoryManager interface, e.g., addCategoryEntry and deleteCategoryEntry, in Https Everywhere and Bluehell Firewall.

6.4 Detection Experiments and Results for Firefox Desktop Extensions

Initially, our objective is to build a comprehensive security analysis framework for mobile browser extensions and evaluate the effectiveness of the framework through a security analysis technique. However, the security analysis approach, which we build for the Firefox for Android extensions, can be reused for the Firefox desktop extensions. The only requirement is to use a different security policy configuration file, as some XPCOM interfaces and functions are only supported by the Firefox desktop extensions and not by the Firefox for Android extensions. To assess how effectively the security analysis mechanism can detect the XPCOM related security issues in the Firefox desktop extensions, we also execute the experiments with a number of Firefox desktop extensions which are taken from different sources based on the information available in different existing literature, popular security blogs, and websites [14, 39, 55]. The Rhino parser can successfully parse all the selected
6.4. EXPERIMENTS WITH FIREFOX DESKTOP EXTENSIONS

Firefox desktop browser extensions and the security analysis technique can detect many known issues which are described in the following paragraphs.

6.4.1 Detecting Malicious Information Flows

The XPCOM interfaces are capable of divulging sensitive information to attackers. In the FFSniff 0.1 extension [14] code, the createTransport function of the nsISocketTransportService is used to disclose sensitive document password information to an untrusted remote server through the Simple Mail Transfer Protocol (SMTP). A sample code snippet of the FFSniff extension is depicted in Figure 6.2. Although the main purpose of the analysis is to find the XPCOM related malicious information flows, we extend it to check whether any critical data is fetched from the document object model (DOM). It enables us to detect the malicious flow that exists in the FFSniff extension code. This detection shows the flexibility and versatility of the analysis technique because using the basic Rhino parser we can track any kind of JavaScript syntaxes very easily.

In this regard, another important point needs to be mentioned that the analysis is extended to support the XMLHttpRequest JavaScript object which is heavily used in the Firefox extensions for sending information to remote servers.

6.4.2 Identifying Vulnerabilities

In the evaluation phase, the security analysis technique identifies the following security vulnerabilities in the experimented Firefox desktop extensions.

**Bookmark related:** The analysis component detects many bookmark related vulnerable XPCOM functions, such as insertBookmark and changeBookmarkURI, in the
Fast Dial 2.15 extension [39]. The bookmark related functions may be responsible for many phishing attacks because by using these functions, it is possible to insert spam URLs in the bookmark menu and lure users to visit those websites.

**Shared information related:** In the Add n Edit Cookies 1.0.0.2 extension, the `setCookieString` function of the `nsICookieService` is identified. This function is used to store vital cookie information which may be misused by some other malware.

**Preference related:** In the ProCon Firefox extension, several risky functions of the `nsIPrefBranch` interface, e.g., `setCharPref`, `setBoolPref`, `setComplexValue`, and `setIntPref`, are used to change different browser settings. It is essential to carefully check browser settings changed by these functions, as it is reported that the particular extension breaks the appearance of a web page.
6.5 Performance Evaluation

Although the security analysis technique can detect XPCOM related malicious information flows and security vulnerabilities in the Firefox desktop extensions, but from the performance point of view the analysis of desktop extensions is not that critical because a desktop environment does not have any resource constraints. To assess how efficiently mobile extensions can be analyzed by the framework, we demonstrate the experimental results for 15 mostly used Firefox for Android Extensions [32]. In this section, we first present a comparative performance analysis of different modes available in CLOUBEX. In the following subsections, we discuss how the performance of the security analysis framework is influenced by different parameters, such as network connection speed, the computational capabilities of a client device, and distance between a client and a server. Finally, we present the prediction results of the dynamic decision making algorithm, and how the performance improvements can help reduce both mobile power consumption and CPU load.

6.5.1 Comparative Performance Analysis

For each of the 15 Firefox for Android extensions, three different scenarios are investigated in a Moto G mobile phone using a wireless network connection available on campus at Queen’s University. In the first and the second scenario, the Firefox for Android extensions are analyzed in the Moto G mobile phone and the cloud server, respectively. In the third scenario, the client application uses the security analysis reports stored in a database repository.

It is important to compare the remote and mobile analysis execution times because a particular extension analysis report may not be available in the database
6.5. PERFORMANCE EVALUATION

Table 6.1: Mobile and remote analysis execution times in milliseconds for different Firefox for Android extensions.

<table>
<thead>
<tr>
<th>Extension Name</th>
<th>Mobile Unzip</th>
<th>Mobile Core Analysis</th>
<th>Mobile Total Execution</th>
<th>Remote Data Transfer</th>
<th>Server Unzip</th>
<th>Server Core Analysis</th>
<th>Remote Total Execution</th>
<th>Gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Video Downloader</td>
<td>6,185</td>
<td>47,621</td>
<td>53,912</td>
<td>4,078</td>
<td>286</td>
<td>2,849</td>
<td>7,229</td>
<td>86.59</td>
</tr>
<tr>
<td>U Block Origin</td>
<td>1,021</td>
<td>43,081</td>
<td>44,178</td>
<td>1,702</td>
<td>74</td>
<td>1,412</td>
<td>3,205</td>
<td>92.75</td>
</tr>
<tr>
<td>Stylish</td>
<td>764</td>
<td>5,030</td>
<td>5,886</td>
<td>1,088</td>
<td>33</td>
<td>216</td>
<td>1,364</td>
<td>76.83</td>
</tr>
<tr>
<td>Adblock For Youtube</td>
<td>124</td>
<td>326</td>
<td>481</td>
<td>536</td>
<td>3</td>
<td>11</td>
<td>560</td>
<td>-16.42</td>
</tr>
<tr>
<td>U Block</td>
<td>2,992</td>
<td>29,041</td>
<td>32,159</td>
<td>2,441</td>
<td>101</td>
<td>447</td>
<td>3,012</td>
<td>90.63</td>
</tr>
<tr>
<td>Self Destructing Cookies</td>
<td>91</td>
<td>2,341</td>
<td>2,468</td>
<td>615</td>
<td>4</td>
<td>54</td>
<td>701</td>
<td>71.60</td>
</tr>
<tr>
<td>Save From Net</td>
<td>385</td>
<td>135,950</td>
<td>136,377</td>
<td>930</td>
<td>15</td>
<td>1,785</td>
<td>2,748</td>
<td>97.99</td>
</tr>
<tr>
<td>Https Everywhere</td>
<td>2,654</td>
<td>6,067</td>
<td>8,918</td>
<td>1,791</td>
<td>94</td>
<td>90</td>
<td>1,997</td>
<td>77.61</td>
</tr>
<tr>
<td>Youtube Downloader 4k</td>
<td>6,045</td>
<td>50,247</td>
<td>56,373</td>
<td>5,443</td>
<td>292</td>
<td>2,896</td>
<td>8,656</td>
<td>84.65</td>
</tr>
<tr>
<td>Bluehell Firewall</td>
<td>38</td>
<td>989</td>
<td>1,045</td>
<td>764</td>
<td>3</td>
<td>89</td>
<td>874</td>
<td>16.36</td>
</tr>
<tr>
<td>Video Without Flush</td>
<td>72</td>
<td>703</td>
<td>797</td>
<td>592</td>
<td>3</td>
<td>27</td>
<td>643</td>
<td>19.32</td>
</tr>
<tr>
<td>UnMHT</td>
<td>1,242</td>
<td>167</td>
<td>1,518</td>
<td>1,094</td>
<td>27</td>
<td>9</td>
<td>1,142</td>
<td>24.77</td>
</tr>
<tr>
<td>PDF Viewer</td>
<td>1,524</td>
<td>241,688</td>
<td>243,415</td>
<td>2,717</td>
<td>70</td>
<td>3,023</td>
<td>5,831</td>
<td>97.60</td>
</tr>
<tr>
<td>Simple Youtube</td>
<td>25</td>
<td>538</td>
<td>628</td>
<td>643</td>
<td>1</td>
<td>12</td>
<td>669</td>
<td>-6.53</td>
</tr>
<tr>
<td>Pytpekepy</td>
<td>109</td>
<td>27,390</td>
<td>27,533</td>
<td>722</td>
<td>5</td>
<td>255</td>
<td>1,005</td>
<td>96.35</td>
</tr>
</tbody>
</table>

The execution times for the mobile and server analysis are presented in Table 6.1. Both the total and task specific execution times are mentioned in the table. The total execution times are slightly higher than the sum of the task specific execution times, as a few milliseconds is required for accomplishing other minor application tasks. Because of the high computing speed, a specific remote server core
analysis time is always much less than a corresponding mobile core analysis time. However, in the remote analysis mode, we need to consider the data transfer time. If a particular mobile analysis takes a considerable amount of time and the data transfer time is not very high for that particular extension, then the performance can be significantly improved by executing the analysis tasks in the remote powerful server. In our experiments, using the remote analysis mode, the performance is improved up to 97.99% over the mobile analysis option. It always takes some fixed amount of time to create a socket connection. Hence, for small extensions, the data transfer time is proportionately higher than the core analysis time. We may not get sufficient performance gain analyzing a simple and small extension at the server end. In the evaluation, for two small extensions, Adblock For Youtube and Simple Youtube, the mobile analysis appears as a better option. However, for small extensions, any of the two approaches can be used as both the mobile and remote analysis modes give satisfactory results.

The size and complexity of an extension strongly influence the performance gain factor. Figure 6.3 shows the server analysis time and data transfer time distribution for three different sized extensions. For large and complex extensions, such as Save From Net, a significant performance improvement can be achieved using the remote analysis mode. On the contrary, for small extensions (e.g., Adblock For Youtube) where the data transfer time dominates the core analysis time, the mobile analysis mode provides a better result.

In general, the time required to get database analysis reports is almost constant, and independent of the size and complexity of an extension. In this mode, users do not need to wait for a long analysis time. We execute this database report reuse
mode for 10 different extensions and observe an average execution time of 786 ms. The application takes some time to communicate with the database. This execution time is much better than the mobile and remote server analysis times required for very large and complex extensions. For any type of extension, the retrieval of existing reports is always the most preferred option, as neither the transfer of data nor the execution of compute-intensive analysis tasks is required in this mode.

6.5.2 Impact of Network Connection Speed

To evaluate how network connection speed controls the performance of the framework, we analyze the extensions using a different 3G internet connection provided by the Wind Canada. Figure 6.4 shows that in the slow Wind’s network, the remote analysis times are much higher than what are required in the fast Queen’s network.
We experience network speed fluctuation from time to time in the Wind network and that results some unexpected execution times for few extensions. Even, for some extensions, such as Flash Video Downloader, Https Everywhere, and Youtube Downloader 4k, the mobile analysis performance is better than the remote analysis performance in the Wind’s network. However, in all these scenarios, the dynamic decision making algorithm correctly identifies the best analysis option and helps us achieve a significant performance gain. Hence, if the data transfer time is very high due to slow network connectivity and a mobile has decent processing speed, then the mobile analysis emerges as a better choice.

Therefore, we can conclude that uploading a large unknown extension in a slow network is a challenge for end users. However, in reality the number of large and unknown extensions is very less. The unknown extensions also need to be uploaded
only once, as for subsequent requests the analysis component can reuse stored analysis reports. With the advent of 4G/LTE, network connection speed is not a major concern in most of the areas. A more comprehensive security analysis algorithm will generate much better results in the remote analysis mode because the data transfer time will remain the same. Users can also analyze extensions in mobile devices, if network connection is not at all suitable for uploading the extensions to the remote server.

6.5.3 Effect of Mobile Computing Speed

To check the impact of a mobile device’s computing speed, we analyze the extensions in a low-speed HTC Desire mobile phone. The HTC Desire cannot analyze all extensions correctly. For some large extensions, for example, Flash Video Downloader, Save From Net, Youtube Downloader 4K, and PDF Viewer, the analysis component throws a memory related exception, "Too deep recursion while parsing". It explains why a low-speed and memory-constrained mobile environment is not at all a good choice for analyzing a large extension. For other extensions, as depicted in Figure 6.5, the HTC Desire mobile analysis times are much higher than the Moto G mobile analysis times. Therefore, for low-speed mobile devices, the remote analysis is much more efficient than the mobile analysis.

6.5.4 Significance of Server-client Distance

Distance between a client device and a remote cloud server determines the data transfer time. We examine the application performance against two different cloud computing servers located in two different geographic locations: Oregon and Singapore. The nearest cloud server is the Oregon one, as in our evaluation the client device is
6.5. PERFORMANCE EVALUATION

Figure 6.5: Mobile analysis times in the HTC Desire and Moto G mobile devices.

located in Canada. Figure 6.6 shows that the remote analysis processes are much faster when we use the Oregon cloud server. We notice that the server analysis times are almost the same, as we use the same c4.large cloud instance in both the cases. Only because of the differences in the data transfer times, the performance can be improved in the range of 42.34%-81.35% using the Oregon server over the Singapore option.

6.5.5 Predictions of Dynamic Decision Making Algorithm

In almost all the experiments, where time difference between the mobile analysis and the remote analysis is significantly high, the dynamic decision making algorithm correctly identifies the best analysis option. The actual better approach and the corresponding dynamic decision making algorithm prediction for different extensions
are mentioned in Table 6.2. In more than 80% scenarios, the dynamic decision making algorithm provides correct results. It is observed that the algorithm cannot predict the best option for some cases where actual time difference between the remote and mobile analysis is very small. This minor deviation is quite acceptable, as the actual analysis execution time depends on so many complex factors, such as the complexity of the JavaScript files and variation in network speed, which are not considered in the implemented prediction algorithm.

The dynamic decision making algorithm calculates the computing speed of a client device in an offline mode. It only takes some time to measure existing network connection speed. This network speed calculation time is further minimized by caching the bandwidth information. Therefore, the algorithm execution time is almost negligible.

Figure 6.6: Remote analysis times using the Oregon and Singapore cloud servers.
### 6.5. PERFORMANCE EVALUATION

Table 6.2: Dynamic decision making algorithm results.

<table>
<thead>
<tr>
<th>Extension Name</th>
<th>Better Approach in Queen's Wifi</th>
<th>Prediction in Queen's Wifi</th>
<th>Better Approach in Wind 3G</th>
<th>Prediction in Wind 3G</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Video Downloader</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>U Block Origin</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Stylish</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Adblock For Youtube</td>
<td>Mobile</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>In Queen’s wifi network, actual time difference between two modes is only 79 ms</td>
</tr>
<tr>
<td>U Block</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Self Destructing Cookies</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Remote</td>
<td>In Wind 3G network, actual time difference between two modes is only 46 ms</td>
</tr>
<tr>
<td>Save From Net</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Https Everywhere</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Youtube Downloader 4k</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Bluehell Firewall</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Remote</td>
<td>In Wind 3G network, actual time difference between two modes is 1,071 ms</td>
</tr>
<tr>
<td>Video Without Flush</td>
<td>Remote</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>UnMHT</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>Mobile</td>
<td>In Queen's wifi network, actual time difference between two modes is only 376 ms</td>
</tr>
<tr>
<td>PDF Viewer</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
<tr>
<td>Simple Youtube</td>
<td>Mobile</td>
<td>Remote</td>
<td>Mobile</td>
<td>Mobile</td>
<td>In Queen’s wifi network, actual time difference between two modes is only 41 ms</td>
</tr>
<tr>
<td>Pytpekepy</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Correct predictions</td>
</tr>
</tbody>
</table>
6.5.6 Improvements in Mobile Power Consumption and CPU Load

By using Trepn, we can measure application specific average power in terms of current (mA) and CPU load. We can calculate actual power by multiplying three parameter values: current, voltage, and application execution time. The calculated results roughly reveal that the mobile analysis consumes more power than the remote analysis. The average consumption of current does not vary significantly in the remote and mobile analysis modes. However, a long execution time leads to a greater power consumption in the mobile analysis mode. We also confirm from the Trepn profiling data that in the mobile analysis mode, the average CPU load is around 25%, whereas in the remote analysis mode, it never crosses 15%. It happens because in the remote analysis mode, most of the intensive tasks are executed at the server end and the mobile device keeps itself in an idle mode during that time period.

6.6 Summary

We build CLOUBEX using various technologies and platforms. The mobile and desktop client applications are developed using Android and Java Swing, respectively. The Rhino JavaScript parser is used to implement the static analysis technique.

Using the static analysis technique, we successfully detect malicious information flows in five Firefox for Android malware extensions that we develop for the purpose of our evaluation. To assess the effectiveness of the analysis approach for unknown vulnerabilities, we conduct experiments with 20 most popular Firefox for Android official extensions. The Rhino parser cannot parse five official extensions because of some syntax errors that exist in different JavaScript files. In the remaining extensions, the analysis technique finds a number of fingerprinting and shared information related
security vulnerabilities. The analysis mechanism also identifies a malicious information flow in the FFSniff Firefox desktop extension and produces first-level warnings for many security vulnerabilities that exist in different Firefox desktop extensions.

In the prototype implementation, we consider only the XPCOM related security issues. However, in addition to using the XPCOM functions, a malware may introduce security vulnerabilities by using other JavaScript APIs (e.g., DOM APIs). For example, the analysis technique can detect the `GetDataSource` XPCOM function of the `nsIRDFService` interface and the `GetAllResource` function of the `nsIRDFDataSource` interface in the Fizzle and Beatnik 1.2 extensions [55]. These XPCOM functions are used to retrieve content from the RDF sources which may contain malicious scripts. However, the analysis technique does not report any security issue because the vulnerable RDF content is injected into a document using the `innerHTML` property of the DOM which is not analyzed in the prototype implementation. In the future, the analysis technique can be made more comprehensive by incorporating analysis logic for other JavaScript APIs.

The performance of the security analysis is improved up to 97.99% over the mobile analysis option by executing the compute-intensive security analysis tasks in a nearby high-performance cloud server. The computational offloading and better response time help us achieve significant improvements in average CPU load and mobile power consumption, respectively. Finally, from the evaluation results, we can conclude that the performance of the framework depends on several parameters, such as extension size, network connection speed, and the computing speed of a client device. However, using the dynamic decision making algorithm, we can select the best analysis option in most of the cases.
Chapter 7

Conclusion and Future Work

In this chapter, we summarize our research work and discuss the limitations of the proposed approach and our future plans.

7.1 Conclusion

In modern times, people widely use extensions in both desktop and mobile browsers. General users are not that much aware of different extension security issues. Therefore, there should be an efficient security analysis technique by which users can instantaneously check the security issues of extensions before installing them in browsers. However, because of the various resource limitations, existing desktop browser extension security analysis approaches cannot be directly used for analyzing mobile browser extensions.

In this thesis, we present CLOUBEX, a generic security analysis framework, which can be used for analyzing different types of desktop and mobile browser extensions. CLOUBEX is built based on a client-server architecture model. This comprehensive framework has a flexibility of analyzing extensions either in a client device or in a high-speed server based on existing network connection speed, extension size, as well
as the computing speed of the particular client device. To justify the effectiveness of CLOUBEX, we implement the proposed framework for the security analysis of the Firefox extensions. We analyze a large number of Firefox desktop and Firefox for Android mobile browser extensions. We notice that the analysis technique correctly detects many malicious source-to-sink information flows and other potential security vulnerabilities in different Firefox extensions. With the help of a dynamic decision making algorithm, the framework also assists us to select the best analysis option, and consequently, achieve a significant performance improvement.

7.2 Limitations

CLOUBEX can be used for analyzing different types of desktop and mobile browser extensions. However, there are few limitations. The proposed static analysis technique is not suitable for detecting security issues injected at runtime. For analyzing the Firefox extensions, we consider only the JavaScript files and Mozilla’s in built XPCOM interfaces. The native C++ code may introduce some additional security challenges. In addition to the Mozilla’s in built XPCOM interfaces, developers can also build custom XPCOM components. In the JavaScript language, a particular functionality can be accomplished in multiple ways using different syntaxes. In the implemented analysis technique, we consider the most common JavaScript syntaxes. Therefore, the analysis mechanism may introduce false negatives if some complicated JavaScript syntaxes are used in malware code. Most of the Firefox extensions are classical and bootstrapped extensions. However, it is also possible to invoke the XPCOM interfaces through the Jetpack core modules. The analysis of the Jetpack core modules is beyond our scope. While we think that the simple algorithm used for the
dynamic decision making is sufficient for identifying the best possible analysis option, it may not be always appropriate to predict the exact security analysis time.

The scarcity of the malicious Firefox for Android extensions imposes some threats to validity. Mozilla does not maintain a separate blocked repository for the Firefox for Android add-ons. For many blocked desktop add-ons, they keep only a sample file, not the actual malevolent extension. As soon as a security issue is identified in an extension, that particular extension gets removed from the official gallery. It is not a very wise idea to download malicious extensions from unsafe repositories in a local environment as those malware may crash the entire system. It is also observed that sometimes, Google blocks a user from searching, if he or she continuously looks for harmful extensions. However, to mitigate the aforementioned evaluation risks, we develop a number of malicious Firefox for Android extensions.

7.3 Future Work

As part of the future work, first, we will enhance the implemented static security analysis technique. In the existing implementation, we use a static taint analysis and consider only explicit information flows. Considering the type of the existing Firefox for Android extensions, it is sufficient to examine only the explicit data flows. However, we have a plan to use a fine-grained information flow control analysis. The analysis of the implicit control flows will help cover other possible use cases, and subsequently, enhance the accuracy of the technique. We will also address security issues other than the XPCOM ones. We believe, the comprehensive and fine-grained analysis of all the security issues will make the framework more pertinent, as in that case we will be able to get much more performance benefit.
In the framework, there is a dynamic decision making algorithm. The prediction of application execution time is completely a separate area of research. We will try to fine tune the algorithm in the future. We may reduce the algorithm execution time, while we get a correct prediction when the time difference between a mobile analysis and a remote analysis is not significantly high.

We will also try to leverage this framework for the security analysis of other mobile browser extensions, e.g., Dolphin and UC Android browser extensions. As other mobile browser extensions use different technologies, we will have to use different parsers for analyzing those extensions.
Bibliography


[79] Hossain Shahriar, Komminist Weldemariam, Thibaud Lutellier, and Mohammad Zulkernine. A model-based detection of vulnerable and malicious browser


Appendix A

Security Evaluation Benchmark

For the evaluation purpose, we develop five different malicious Firefox for Android extensions using a large number of security critical XPCOM interfaces and functions. The in-depth analysis and development of the malicious extensions also help us understand which interfaces and functions are supported in the Firefox for Android architecture. For the security evaluation benchmark, we consider six important security categories: preference, login manager, cookie, file system, location, and network. In the following sections, the security issues related to different XPCOM interfaces and functions are presented in tabular forms.

A.1 Browser Preference

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsIPrefBranch</td>
<td>clearUserPref</td>
<td>Vulnerable</td>
<td>It may delete a security related preference value set by a user.</td>
</tr>
<tr>
<td></td>
<td>deleteBranch</td>
<td>Vulnerable</td>
<td>It may remove one or many security related preference values referenced by the particular preference branch.</td>
</tr>
<tr>
<td></td>
<td>getBoolPref</td>
<td>Information leakage</td>
<td>It may be used to retrieve an important boolean preference value.</td>
</tr>
</tbody>
</table>
### A.1. BROWSER PREFERENCE

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCharPref</td>
<td>Information leakage</td>
<td>It may be used to retrieve an important string preference value.</td>
</tr>
<tr>
<td>getChildList</td>
<td>Information leakage</td>
<td>It may be used to retrieve one or more important preference values which fall under the specific root node.</td>
</tr>
<tr>
<td>getIntPref</td>
<td>Information leakage</td>
<td>It may be used to retrieve an important integer preference value.</td>
</tr>
<tr>
<td>lockPref</td>
<td>Vulnerable</td>
<td>Locking a preference may cause the removal of the security specific value set by a user.</td>
</tr>
<tr>
<td>resetBranch</td>
<td>Vulnerable</td>
<td>It may clear security specific preference values and reset the values to their default values.</td>
</tr>
<tr>
<td>setBoolPref</td>
<td>Vulnerable</td>
<td>It may change the value of a boolean preference which is critical from the security perspective.</td>
</tr>
<tr>
<td>setCharPref</td>
<td>Vulnerable</td>
<td>It may change the value of a string preference which is critical from the security perspective.</td>
</tr>
<tr>
<td>setIntPref</td>
<td>Vulnerable</td>
<td>It may change the value of an integer preference which is critical from the security perspective.</td>
</tr>
<tr>
<td>nsIPrefService</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBranch</td>
<td>Internal processing required</td>
<td>It is used to get a specific branch from the preference tree. After that, important preference values can be fetched from that particular branch.</td>
</tr>
<tr>
<td>getDefaultBranch</td>
<td>Internal processing required</td>
<td>The getDefaultBranch function is quite similar to the getBranch function. However, for the getDefaultBranch function, the branch returns only the default values.</td>
</tr>
<tr>
<td>resetPrefs</td>
<td>Vulnerable</td>
<td>It may delete security specific preference values, as it is used to completely re-initialize the preference system.</td>
</tr>
<tr>
<td>resetUserPrefs</td>
<td>Vulnerable</td>
<td>It may remove security specific preference values set by a user.</td>
</tr>
</tbody>
</table>

Table A.1: Browser preference related security critical XPCOM interfaces & functions.

There are few other functions, such as `getComplexValue (nsIPrefBranch)` and `readUserPrefs (nsIPrefService)`, which are also important for the security analysis of the Firefox extensions. However, those functions are not explored in the Preference Explorer extension.
A.2 Login Manager

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsILoginManager</td>
<td>addLogin</td>
<td>Vulnerable</td>
<td>It is used to store a security sensitive login information which may be accessed by some malicious extensions.</td>
</tr>
<tr>
<td></td>
<td>findLogins</td>
<td>Information leakage</td>
<td>It is used to get login information based on a criteria.</td>
</tr>
<tr>
<td></td>
<td>getAllLogins</td>
<td>Information leakage</td>
<td>It is used to retrieve all login information stored in the login manager.</td>
</tr>
<tr>
<td></td>
<td>searchLogins</td>
<td>Information leakage</td>
<td>It returns a list of matching logins.</td>
</tr>
</tbody>
</table>

Table A.2: Login manager related security critical XPCOM interfaces & functions.

In the login manager category, there exist a number of other functions, for example, `migrateAndAddLogin(nsILoginManagerIEMigrationHelper)` and `decrypt (nsILoginManagerCrypto)`, which are also very much relevant from the security point of view. There are some functions, such as `addUser(nsIPasswordManager)`, which are currently deprecated. However, a user may use an older version of Firefox. Hence, it is also essential to consider the deprecated functions for the security analysis of extensions. We do not explore those deprecated functions.

A.3 Cookie

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsICookieService</td>
<td>getCookieString</td>
<td>Information leakage</td>
<td>It is used to fetch a complete cookie string associated with a particular URI.</td>
</tr>
<tr>
<td></td>
<td>getCookieString FromHttp</td>
<td>Information leakage</td>
<td>This function is also used to retrieve a complete cookie string associated with a particular URI.</td>
</tr>
<tr>
<td></td>
<td>setCookieString</td>
<td>Vulnerable</td>
<td>It is used to store a vital cookie information which may be leaked by some other malevolent extensions.</td>
</tr>
</tbody>
</table>
setCookieString
FromHttp
This function may introduce a security vulnerability by storing a confidential cookie information.

table A.3: Cookie related security critical XPCOM interfaces & functions.

In addition to the above mentioned functions, the functions such as setCookie (nsICookieStorage) and getCookie(nsICookieStorage) also play important roles in introducing different types of security issues.

A.4  File

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsIFile</td>
<td>copyTo</td>
<td>Vulnerable</td>
<td>It may copy a file to a sensitive location.</td>
</tr>
<tr>
<td></td>
<td>create</td>
<td>Vulnerable</td>
<td>It may be used to create a new malicious file.</td>
</tr>
<tr>
<td></td>
<td>initWithPath</td>
<td>Internal processing required</td>
<td>It is required to check a file or directory location.</td>
</tr>
<tr>
<td></td>
<td>moveTo</td>
<td>Vulnerable</td>
<td>Similar to the copyTo function, it may transfer a file to a sensitive location.</td>
</tr>
<tr>
<td></td>
<td>renameTo</td>
<td>Vulnerable</td>
<td>It also moves a file, but within the same volume.</td>
</tr>
<tr>
<td></td>
<td>remove</td>
<td>Vulnerable</td>
<td>Using this function a critical file may be deleted.</td>
</tr>
<tr>
<td>nsIFileOutputStream</td>
<td>init</td>
<td>Internal processing required</td>
<td>It is important to identify the file output stream which is used to write data in a file.</td>
</tr>
<tr>
<td>nsIOutputStream</td>
<td>write</td>
<td>Vulnerable</td>
<td>It is used to write data in a file. Important file information may be accessed and divulged to a suspicious user by some malware.</td>
</tr>
<tr>
<td>nsIFileInputStream</td>
<td>init</td>
<td>Internal processing required</td>
<td>It is important to identify the file input stream which is used to read data from a file.</td>
</tr>
</tbody>
</table>
A.6. NETWORK

<table>
<thead>
<tr>
<th>nsIScriptableInput-Stream</th>
<th>init</th>
<th>Internal processing required</th>
<th>It is also required to detect the scriptable input stream which is used to read data from a file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td></td>
<td>Information leakage</td>
<td>It may be used to read data from a sensitive file.</td>
</tr>
</tbody>
</table>

Table A.4: File related security critical XPCOM interfaces & functions.

There are many other file related XPCOM functions, e.g., `copyToFollowingLinks` (nsIFile), `CopyToNative(nsIFile)`, and `moveToNative(nsIFile)`, which also need to be analyzed for the security analysis of the Firefox extensions.

A.5 Location

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsISupports/</td>
<td>getCurrentPosi-</td>
<td>Information</td>
<td>It is used to retrieve a user’s different types of location information, such as latitude,</td>
</tr>
<tr>
<td>nsIDOMGeo-</td>
<td>tion</td>
<td>leakage</td>
<td>longitude, and altitude.</td>
</tr>
<tr>
<td>Geolocation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>watchPosition</td>
<td>Information</td>
<td>Information</td>
<td>It is similar to the getCurrentPosition. However, it continues to call the callback with</td>
</tr>
<tr>
<td></td>
<td>leakage</td>
<td>leakage</td>
<td>updated position</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>information at a regular</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interval.</td>
</tr>
</tbody>
</table>

Table A.5: Location related security critical XPCOM interfaces & functions.

A.6 Network

<table>
<thead>
<tr>
<th>XPCOM Interface</th>
<th>XPCOM Function</th>
<th>Security Issue Type</th>
<th>Security Issue Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsIXMLHttpRequest</td>
<td>open</td>
<td>Information</td>
<td>It may be used to open a connection with an untrusted remote host. For a HTTP GET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>leakage</td>
<td>connection, this function needs to be analyzed to check whether any sensitive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>information is passed through the remote host URL.</td>
</tr>
</tbody>
</table>
### Table A.6: Network related security critical XPCOM interfaces & functions.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Function</th>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsISocket-TransportService</td>
<td>send</td>
<td>Leakage</td>
<td>For a HTTP POST connection, this function is used to specify data that needs to be transferred. Using this function, secret information can be sent to an untrusted recipient.</td>
</tr>
<tr>
<td>nsISocket-TransportService</td>
<td>createTransport</td>
<td>Internal processing required</td>
<td>This function is important to check a remote host address.</td>
</tr>
<tr>
<td>nsITransport</td>
<td>openOutputStream</td>
<td>Internal processing required</td>
<td>It needs to be analyzed to identify the output stream in which critical data is written for sending through a socket.</td>
</tr>
<tr>
<td>nsIOutputStream</td>
<td>write</td>
<td>Leakage</td>
<td>This function may be used to send the data through a socket channel.</td>
</tr>
</tbody>
</table>