Architectural Risk Analysis of Chromium

Mati Ullah Khan Mansoor Munib
Email: {matul986, manmu259}@student.liu.se
Supervisor: Shanai Ardi, {shaar@ida.liu.se}
Project Report for Information Security Course
Linköpings universitetet, Sweden

Abstract

Conventional risk analysis techniques do not necessarily cover all security aspects in software. Defects in a software design cannot be identified by simply looking for flaws in the code. Therefore, carrying out risk analysis at architecture level is important. In this project we have performed architectural risk analysis of Chromium which is an open source web browser project. The method followed to carry out the analysis is a best practice approach described by Gary McGraw in his book Software Security: Building Security In.

1. Introduction

Risk analysis is the process of identifying and assessing risks in a software project.

"Traditional risk analysis output is difficult to apply directly to modern software design" [1] and by applying these techniques to complex modern software one cannot identify all vulnerabilities and threats in a software.

"*Design flaws account for 50% of security problems*" [1]. So to identify and eliminate such design problems we have to perform risk analysis mechanism at an earlier stage in the software development life cycle. Risk analysis at such an early stage can significantly improve the overall security measure of any software.

Architectural risk analysis process is applied on the design of software to identify and assess design level flaws which cannot be found by simply analyzing the code.

In this project we perform Architectural Risk Analysis of Chromium which is an open-source browser project. The method we adopt is an architectural risk analysis best practice described by Gary McGraw in "*Software Security: Building Security in*" [1]. This method requires extensive knowledge about known attack patterns, vulnerabilities and security design principles. The process consists of three basic steps: attack resistance analysis, ambiguity analysis and weakness analysis.

We try to identify security flaws in the software design using the above mentioned steps. Predefined attack pattern and vulnerabilities are examined in attack resistance analysis. Ambiguity analysis focuses on identifying new risks by analysis of the software design. In weakness analysis the weakness in security due to dependence on external software is analyzed.

As the result of architectural analysis we will identify design level security flaws in Chromium.

2. Chromium Overview

2.1 Introduction

Chromium is an open-source browser project. Google chrome is built with open source code for chromium. We have worked with chromium version 0.2.149.27.

2.2 Architecture Overview

Developing a high level architecture view of target application is a prerequisite for performing the actual steps in risk analysis [1]. Building such an overview is very important because this high level description is used in the remaining steps of the analysis. The idea is to see the complete picture so that no details are missed during the analysis.

Figure 1 shows a high level overview of chromium which highlights all the important components of the software and the flow of information is explained below.

Chromium uses a separate process for each tab opened in the browser called renderer [3]. This approach is used to keep the whole application safe in case of a bug in any tab. Each renderer has a global render process which is responsible for communication with the browser process. Each RenderView corresponds to a single tab of content. It handles all navigation-related commands to and from the browser process.

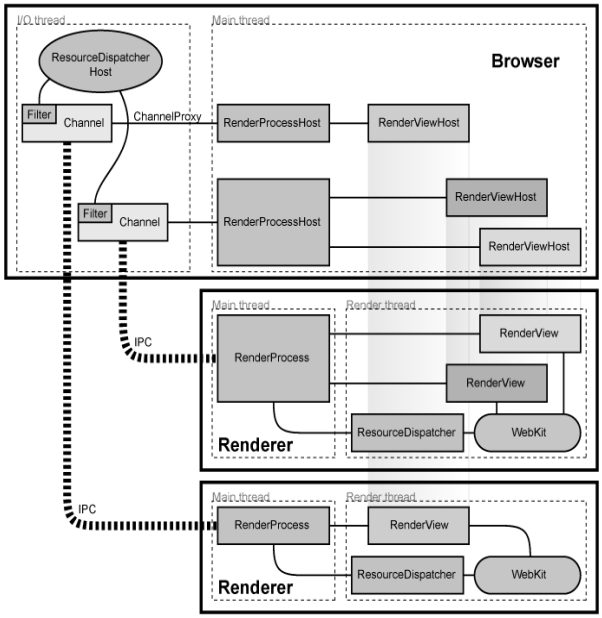

Figure 1 Architecture Diagram [2]

The main process in chromium design is “Browser” which manages all renderer processes. All network communication is handled by the main browser process. The browser process maintains a separate Render Process host responsible for communication with a renderer process. The RenderProcessHost maintains a RenderViewHost for each view in the renderer. Communication from the browser to a specific tab of content is done through these RenderViewHost objects, which know how to send messages through their RenderProcessHost to the RenderProcess and on to the RenderView.

WebKit is an open source web browser engine which is used to layout web pages. It has a ResourceLoader which is responsible for fetching data via the WebKit port. Chromium uses different types, coding styles, and code layout than the third-party WebKit code. The WebKit "glue" provides a more convenient embedding API for WebKit using chromium’s coding conventions and types.

Chromium is a multi-process architecture so there’s a lot of inter-process communication involved. It uses named pipes as main communication mechanism. A named pipe is assigned for communication between each renderer and the browser process. The browser process has two threads and I/O thread and one main thread. Messages from renderers are received at the I/O thread and resource request messages are directly forwarded to resource dispatcher rather than the main thread. [3]

2.3 Displaying a Webpage [3]

When the user types into or accepts an entry in the URL bar, the navigation controller is instructed to navigate to the respective URL

The NavigationController forwards the call to browser process which will create a new RenderViewHost if necessary, which will cause creation of a RenderView in the renderer process in case it is the first navigation.

The navigate request is forwarded to RenderViewHost. The NavigationController stores this navigation entry, but it is marked as "pending" because it doesn't know for sure if the transition will take place.

RenderViewHost sends a message via the RenderProcessHost to the new RenderView in the renderer process.

When told to navigate, RenderView may navigate, it might fail, or it may navigate somewhere else instead (for example, if the user clicks a link). RenderViewHost waits for a reply from the RenderView.

When the load is "committed" by WebKit (the server responded and is sending us data), the RenderView sends this reply, which is handled in RenderViewHost.

The NavigationController is updated with the information on the load. In the case of a link click, the browser has never seen this URL before. If the navigation was browser-initiated, as in the startup case, there may have been redirects that have changed the URL. The NavigationController updates its list of navigations to account for this new information. [3]

3. Attack Resistance Analysis

Attack resistance analysis requires knowledge about applying known attack patterns and demonstrating the attacks using exploit graphs. This step in the analysis aims at applying known attack patterns and vulnerabilities to the high level overview described in the previous section. First we identify known vulnerabilities using knowledge about existing attacks. Then we map these vulnerabilities to chromium application using attack patterns. This uncovers any flaws in the architecture. Finally we demonstrate the possibility of exploitation of the detected vulnerabilities using exploit graphs.

An exploit graph enables analysts to gain an overview about the exact sequence required to carry out an attack given any vulnerability [1]. It is essentially a flow chart which describes an attack and includes some basic details about the attack such as its delivery, access and actualization.

3.1 Results

Attack pattern is a plan of action for carrying an attack and is applicable to different software applications in

general. Some example attack patterns are command delimiters, argument injection, URL encoding etc [4]. In this step we first gained information about known vulnerabilities and attack patterns [4]. Then we applied some selected attack patterns to chromium to identify risks in the architecture. Finally we developed an overview of each applicable attack pattern using exploit graphs to demonstrate the possibility of the attack. The flaws we have identified are not a complete list of flaws in the application. Lots of risk analysis work has already been carried out on Chromium so the flaws mentioned here have also been discovered by other analysts. But we show here that these flaws can be efficiently identified by the risk analysis approach we have adopted. We have focused on some popular attack patterns to demonstrate the application of attack resistance analysis process. Identified flaws in chromium are listed below:

1. Chromium suffers from the famous buffer overflow vulnerability which is related to memory usage [5].
   We apply the attack pattern: "*Overflow variables*" [4] which confirms the flaw. The vulnerability is identified using the most basic approach of providing random long inputs to chromium [4]. When we supply a long title in a webpage and try to save it the long input title causes the overflow.
   The overflow is caused due to a boundary check error when a user tries to save a webpage which has sufficiently long title [5]. An attacker can exploit this vulnerability and gain control of the instruction pointer and using it he/she can run any arbitrary malicious code.
   The exploit graph which describes an attack scenario for this vulnerability is shown in figure 2. The graph shows different steps (circles) carried out by an attacker from delivering the attack to the actualization of the attacker's goals. To exploit this flaw an attacker may create a webpage containing harmful code and having a long title. Then he/she tricks the user into saving the page to deliver the attack as shown in the graph. When user tries to save the page overflow occurs and the attacker gains access to instruction pointer which enables malicious code execution. Thus in access stage the attacker gains access to an object of a protected class. Finally the attack actualizes when a call to a protected class method is made. [5]

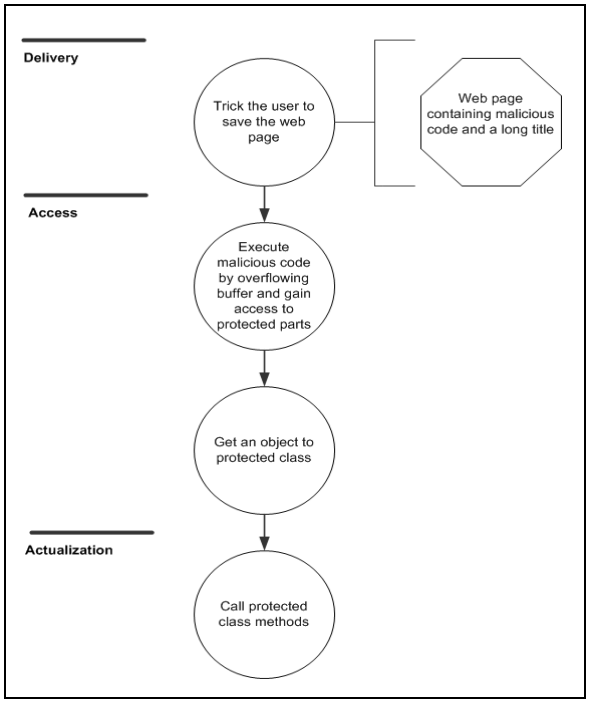

Figure 2 - Exploit graph for buffer overflow

2. Chromium is also vulnerable to denial of service attack. The vulnerability is listed on bugtraq [6] but we have applied the attack pattern "*Command delimiters*" [4] which identifies this vulnerability. In such an attack applications are made to execute commands which may cause unwanted behavior. Carriage return is one command delimiter that often passes through filters [4]. In this case we can use carriage return which causes memory exhaustion that leads to Denial of Service.
   The vulnerability exists in window.open function, when input to this function is exploited and carriage return(\r\n\r\n) is fed as input the browser generates several new windows causing memory exhaustion. [6]
   The exploit graph in figure 3 gives overview of an attack scenario for this vulnerability. The attacker can deliver the attack by overflowing the buffer and when the malicious code executes it gains necessary access to invoke the window.open function with the carriage return delimiter as inpurt parameter.

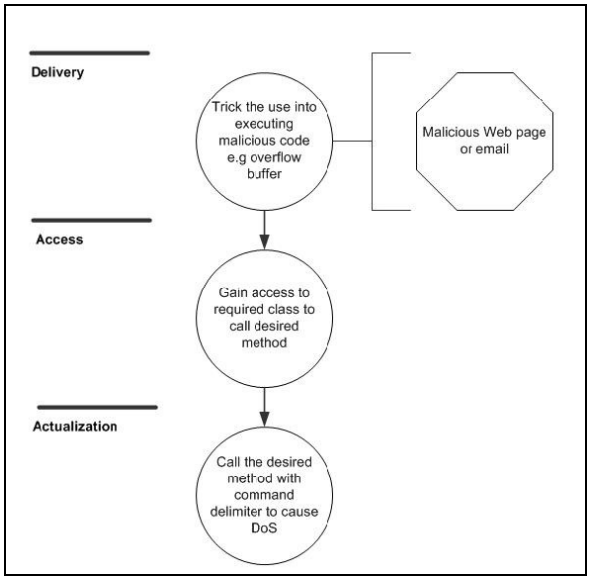

Figure 3 - Exploit graph for DoS

4. Ambiguity Analysis

Ambiguity analysis aims at identifying new risks by uncovering ambiguity and inconsistency in the design. This step requires knowledge about secure design principles. Attack resistance analysis only uncovers known vulnerabilities whereas ambiguity analysis detects new faults. This process demands creativity and at least two experienced analysts. Each of the analysts performs the analysis separately and discusses their understandings at the end.

4.1 Results

In ambiguity analysis we acquired knowledge about secure design principles [7]. Then we performed analysis of the chromium design independently. Finally we discussed our findings to unify our understandings.

The main idea behind chromium security design is to separate the renderer process from the main browser process. The browser process communicates with the operating system and the renderer process communicates with the web with restricted privileges. This leaves the attacker unable to get system level privileges by attacking the renderer.

However chromium renderer relies heavily on WebKit for rendering html to display a webpage. WebKit has many security flaws as we will see in weakness analysis. The renderer process does not secure these vulnerabilities in the WebKit which is a violation of the secure design principle i.e. "*secure the weakest link" [7]*. This principle states that the security is a chain and the software is as secure as the weakest link in the chain which in this case is the WebKit as we have found out during the weakness analysis that it has many security flaws.

Chromium has multiple processes so a lot of inter process communication is involved. The renderer process communicates with the browser process using named pipes. Messages received from the renderer at the browser process are not properly validated and checked and this could cause a breach in the renderer sandbox. The browser process receives messages from the renderer in the RenderProcessHost. The method *OnMessageRecieved* of *Listener* interface upon receiving a message from the renderer process simply unpacks the message. One of the easiest ways to get out of the sandbox is to take advantage of this insecure message unpacking. The design principle "defense in depth" [7] says that defense should be implemented at various levels so that if one level misses the error next level catches it. Chromium here depends on the sandbox created around the renderer but if an attacker gets into the renderer the message from renderer can carry malicious data to the browser and the sandbox will be breached.

5. Weakness Analysis

Weakness analysis aims at identifying weaknesses in design that arise due to assumptions about third party software. Knowledge about existing flaws in underlying frame works and other third party software used by the target application is required to perform weakness analysis. As shown in figure 4 applications nowadays are built upon layers of other software like .NET and J2EE frameworks and use outside libraries like DLLs or common language libraries. So the vulnerabilities in third party software affect the security of an application.

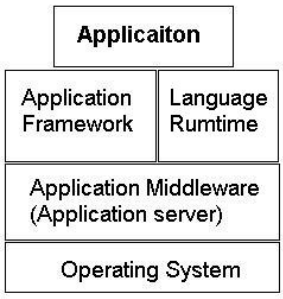

Figure 4 - Modern Application Architecture [8]

Weakness analysis requires examining the weak security provided by the framework and known vulnerabilities in COTS, network topology, platform, physical environment, build environment etc [1]. Then

analyze what assumptions the application makes about these dependencies.

Chromium is a complex application and has various external dependencies. However, we focus on the renderer part of the application. Renderer process uses WebKit, which is an open source web browser engine, for rendering any HTML or XHTML web page and communicates with browser for I/O. The impact of the WebKit flaws on the Renderer process is our main focus.

5.1 Results

In attack resistance analysis we first identified external software dependencies of chromium. We narrowed our focus to WebKit, identified existing flaws in WebKit. In the next step we analyzed what services chromium uses from the WebKit and what assumptions are made about WebKit security. Then we analyze the effect of WebKit security failure on chromium.

WebKit is vulnerable to a cross-site scripting vulnerability [9]. A remote user can create specially crafted HTML that, when loaded by the target user will trigger an invalid conversion in the WebKit code when rendering frame sets and execute arbitrary code on the target system. The code will run with the privileges of the target user. User's cookies can also be accessed by a remote user, if any, associated with an arbitrary site, access data recently submitted by the target user via web form to the site, or take actions on the site acting as the target user.

An attacker could easily trick users into launching an executable Java file by combining a flaw in WebKit with a known Java bug [10]. The problem is that, after a user double-clicks download at the bottom of the screen, this application is opened without any warning, which would allow a malicious hacker to easily execute any Java program on a user's machine.

Retrieving cookies from the user machine is an important feature of web application but in the WebKit if we set the cookie through http response they are either not stored at all or stored on the client side but not read. [11]

For writing Cascading Style Sheet (CSS) of a website we need to add prefix of the vendor in JavaScript. Following code is used to get the vendor prefix:

```
function getVendorPrefix()
{
  Var regex = /^(Moz|Webkit|Khtml|O|ms|Icab)(?=[A-Z])/;
  var someScript = document.getElementsByTagName('script')[0];
  for(var prop in someScript.style)
  {
            if(regex.test(prop))
              return prop.match(regex)[0];
  }
  return '';
}
```
[12]

By running above code in WebKit based browsers, empty string is returned.

Chromium uses browser management services from the WebKit which include Navigation Services, Rendering Services, DOM and JavaScript Management Services.

An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy. Due to the cross site scripting bug in WebKit, Chromium allows unauthorized access to user data.

Carpet bombing vulnerability of WebKit allows any harmful code to run without notifying the user. Users who are not yet familiar with Chrome's interface are convinced to click on download which appear at the bottom and can be fooled that the download is actually just part of the web page. When the user clicks the download this application is opened without any warning and starts exploiting the resources.

Cookies can be used for many things such as authenticating logins to Web sites and storing preferences for the Web sites, and they can be used for tracking where a user goes, whether within a Web site or between Web sites. But the bug in WebKit for setting and getting cookies makes Chromium's performance poor in browsing e-commerce application.

6. Risk Impacts and Mitigation suggestion

The risk matrix below ranks identified risks in chromium into different categories i.e. extreme, high, medium and low.

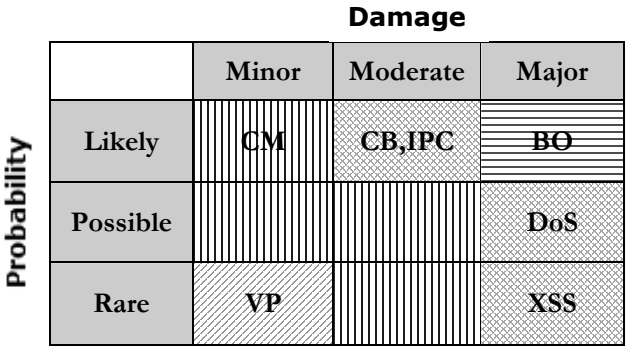

Figure 5 - Risk Matrix

Extreme risk – Immediate fix required	BO - Buffer Overflow
High risk – Fix required as soon as possible	CB - Carpet Bombing
Medium risk – Fix in near future	VP - Vendor Prefix
Low risk – Fix if resources available	CM - Cookies Management
	XSS - Cross Site Scripting
	DOS - Denial of Service
	IPC – Inter process

The ranking is based on the likeliness of an attack which depends mainly on ease of carrying out an attack and estimated damage caused by an attack.

Buffer overflow is easy to exploit and hence has a high likelihood. In case of this attack the attacker may run malicious code and can potentially gain complete control so it is ranked as extreme risk. Therefore, fixing this problem can reduce the overall risk to the application significantly.

DoS service attack is difficult to perform as compared to buffer overflow as it requires some additional knowledge about getting the command delimiter through the filter. Hence its probability is in neither rare nor likely but the damage in case of this attack is serious, so it is ranked as a high risk and fixing it as soon as possible is necessary.

Cross site scripting is possible due to a flaw in WebKit. It is difficult to carry out because attacker must know how to break the same origin policy of browser but in case the attacker is successful the attacker can gain elevated privileges and cause severe damage.

7. Conclusion and Future Recommendations

Performing risk analysis at an initial stage in the software development life cycle is very important. Design level flaws are hard to identify by code review. By applying this best practice architectural risk analysis technique we can efficiently identify some design level security faults in chromium. In the three step process we identified known vulnerabilities by applying attack patterns in attack resistance analysis, new vulnerabilities in design in ambiguity analysis and design flaws due to dependence on third party software in weakness analysis.

The risks we have identified do not cover all risks present in chromium; we have focused on specific areas in the risk analysis process because chromium is a huge software and extensive knowledge and experience is required in each step of the analysis. We have focused on some popular attack patterns in attack resistance analysis such as buffer overflow, DoS, click jacking. In ambiguity analysis our focus was few of the security design principles explained in "*Building Secure Software how to avoid security problems the right way*" [7]. In weakness analysis we focused on weaknesses in a renderer process due to its dependence on WebKit. We recommend a complete risk analysis on chromium covering all components and aspects so that a complete list of design level risks is formed.

References

[1] Gary McGraw: Software Security: Building Security in. ISBN 0-321-35670-5

[2] http://dev.chromium.org/developers/design-documents/multi-process-architecture, 26-03-2009

[3] http://dev.chromium.org

[4] Greg Hoglund, Gary McGraw: Exploiting software How to break code, ISBN 0-201-78695-8

[5] http://code.google.com/p/chromium/issues/detail?id=1414

[6] http://seclists.org/bugtraq/2008/Sep/0251.html

[7] John Viega, Gary McGraw: Building Secure Software how to avoid security problems the right way, ISBN 0-201-72152-X

[8] http://secappdev.org/handouts/2009/softwaresecuritytouchpointarchtecturalriskanalysis.pdf

[9] http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=520052

[10] http://www.readwriteweb.com/archives/security_flaw_in_google_chrome.php

[11] https://bugs.WebKit.org/show_bug.cgi?id=23520

[12] http://leaverou.me/2009/02/find-the-vendor-prefix-of-the-current-browser/#more-48