An Analysis on Google Chrome

December 6, 2010

Tony Lei, Alfred Lam

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

Tony: fious@interchange.ubc.ca, Alfred: alfred89@interchange.ubc.ca

ABSTRACT: Google Chrome is a more recent browser with some unique features (sandboxing feature for example). Chrome is finding more users every year. As its popularity increases, more attention to its features is inevitable. This report will feature an analysis on its pop-up blocker, JavaScript blocker, sandboxing feature, safe browsing, and auto-update process. The pop-up blocker is tested for its thoroughness and if a bypass is possible. JavaScript is analyzed on specific usages in different websites. Sandboxing is explained and a work-around is looked upon. Chrome's safe browsing design and updating process is examined and possible design improvements are searched into.

I. INTRODUCTION

Google Chrome is a web browser released in late 2008. Chrome's first intention was to be a fast and secure browser. Chrome creates individual processes as each tab opens (sandboxing). This also makes it fast since each this means each webpage is handled as an individual process. As for security, Chrome openly advertises their sandboxing, safe browsing and auto-update feature. Most of our approaches were done through examining Chrome's current design and trying to find a loop-hole or a bypass through their design. This report will focus on the following five security features; popup blocker, JavaScript blocker, sandboxing, safe browsing, and auto update. We found some events that were deemed an unintentional behavior for Chrome, such as the pop-up blocker (JavaScript is able to pop up a pop-up). The update process is automated and runs in the background. If malicious activities were to happen in the background relating to the update process, the user may not notice. This is not a secure design as they're not questioning assumptions. Google assumes pop-ups are launched by the body section of the HTML, and updates will work. Chrome grants users to turn off all JavaScript through an option, to stop malicious JavaScript, but over 75% of websites use JavaScript [6]. This security feature would violate the psychological acceptability design principal, as users would not want to turn on this feature as their website browsing experience would be hindered by a large amount (Websites like hotmail will be blocked). Changes will need to be made if users are not secure or happy. This analysis is significant because the number of Chrome users have been steadily growing.

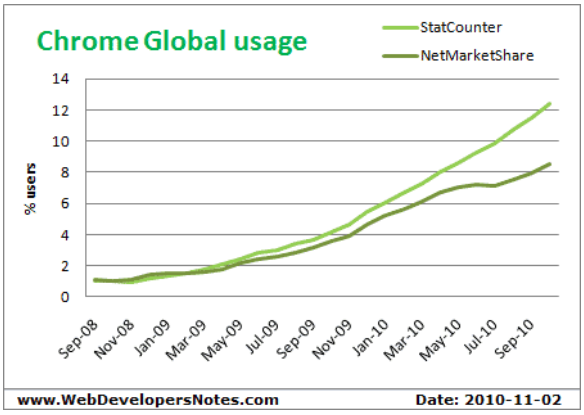

Fig. 1 - This graph shows the growth of Chrome usage since its release http://www.webdevelopersnotes.com/articles/chrome_usage_statistics.php

Google reported that there are over 70 million users in May 2010 [5]. Therefore it is very important to study the security implementations that are protecting these users.

II. ANALYZED SYSTEM

Google Chrome on default enables their Safe Browsing feature. With Safe Browsing turned on, a warning will appear whenever the user visits a suspected website containing phishing and/or malware [1]. There are a total of five different warnings depending on the detected website:
An example is shown in Figure 2.

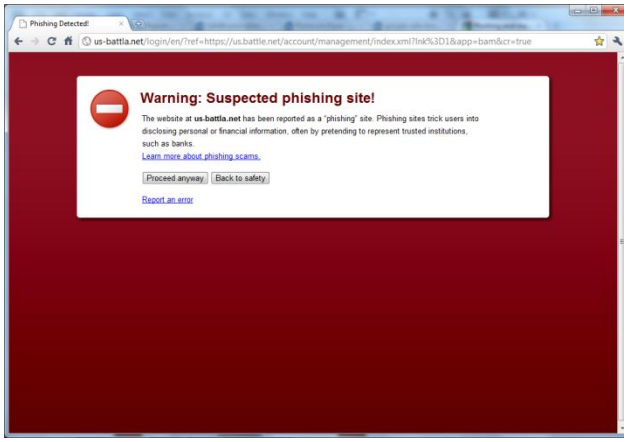

Fig. 2 - Warning: Phishing Site

"If [Google] finds a website that looks like it's a phishing page, it gets added to a list of suspected phishing websites" [2]. Websites that comprehend potentially malicious activity will be tested on a virtual machine on Google's side. Certain events will be monitored, such as viruses being installed, and if the check finds an event did occur, the website "will be added to a list of suspected malware-infected websites" [2].
For Safe Browsing to operate properly, Chrome downloads a list of information about websites that may contain malicious software or engage in phishing. The list does not contain the full URL of suspicious websites. Rather, each URL is hashed with SHA-256 and truncated to the first 4-bytes [1],[3]. This design saves space and bandwidth of users. The downloaded data comes in the form called a chunk. There are two types of chunks, add and sub chunks. Add chunks contain new hash prefixes for the client to match against, while the sub chunks tell the client to disregard particular hash prefixes from an add chunk. Sub chunks allow false positives to be removed from the blacklist.
This data design grants the client to download the blacklist incrementally, and gives the sending-server flexibility in deciding which chunks to send first. Each chunk belongs to a particular list. Chrome performs an update request to get the new blacklist data from the server every few minutes [3].
As the user browses the Web, each URL is hashed and truncated, then compared to the partial-hashed list the browser downloaded.
Each website is represented as host-suffix / path-prefix expressions.
Ex.
"**http://www.host.com/service/login.html**", the expressions "**host.com/**" and "**host.com/service/**" would both be expressions [3].

Each expression is hashed and a 4-byte hash prefix of the expression is generated, and then compared to the downloaded blacklist locally [3]. Collisions alone aren't sufficient to block the URL, thus the browser will contact Google's server to get the full, specific URL (full 32-byte hash) of the website page. Chrome can then determine if the user is visiting a risky website, and warn the user about it [1][2].

Google Chrome's Sandboxing feature isolates each new tab into processes. This allows each website to run faster, and also prevents JavaScript from communicating between Tabs. It also helps boost reliability of the browser. As explained on Chrome's website, "if an

individual tab freezes or crashes, the other tabs are unaffected [7]."
The pop-up blocker feature can be turned off or on in Chrome's options; same with JavaScript blocker. With these turned on, pop-ups should not appear and websites containing JavaScript should not operate as intended.

III. RELATED WORK

We have found many articles online that analyzed Google Chrome but none focused much on the security of the browser. Also, these articles are all outdated, as most of them were written in September of 2008 to introduce the browser when it was first released. We did find one security article that addresses an update issue.
In this article, the author from PC Magazine, Larry Seltzer, addresses the issue of Chrome performing updates without the user's consent, and also how the feature can't be turned off. He also states that "the program is stored in a user-writeable directory" [4]. However he did not provide any solutions and this is what our report will address.
We did not find any more articles about Chrome's pop-up blocker, JavaScript blocker, safe browsing, and sandboxing. This report will be addressing something that has not been focused on before. The behavior of other browsers in a similar situation will not be covered in this document.

IV. ANALYSIS

IV.1.0 Safe Browsing Update Process

Chrome checks for updates regularly in the background while the browser is in use. In the update process, the client contacts the Safe Browsing server via HTTP and sends a list of all the chunks the browser has [3].
Ex.

goog-malware-shavar:a:1-30,42
goog-malware-shavar:s:5-15

This means the client has all the goog-malware-shavar add chunks between 1-30, inclusive, and chunk 42. It also has sub chunks 5-15. If the client wants data for a list, but does not have any chunks for it yet, then the request will just include the list name

Ex. Googpub-phish-shavar:

The response for the update request does not hold new chunk data for the client, but contains a series of redirected URLs (which include new add and sub chunks) for the client to download. Using this design, it allows the data to be stored on proxy servers, which is not true if the update response held the data. As mentioned, the client will fetch each redirect URL and store the results in its local database (update response may instruct the client to delete chunks too). Both the "update response and redirect URL data are signed by the server, using a key that the client has previously obtained. This allows "the client to authenticate the source of the data, and detect whether it has been tampered with" [3]. Figure 3 depicts the update process.

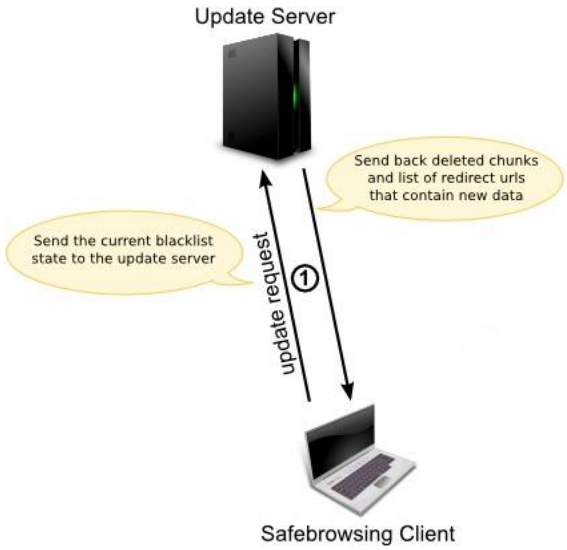

Fig. 3 - Update Process, taken from [3]

IV.1.1 Block Update Request

Chrome is installed in a directory where the user has full write access [4]. Chrome is able to perform updates without any UAC (User Account Control) prompts. Using Microsoft Network Monitor 3.4, it was able to track

exactly what Chrome was doing. Figure 4 shows an example of Chrome auto-updating.

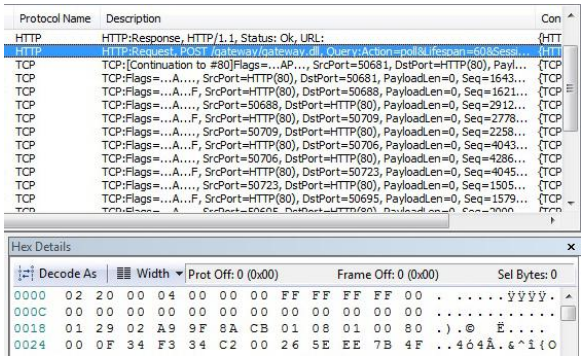

Fig. 4 - Packet Sniffing Chrome

An adversary, after gaining access to your HTTP packets, could block update requests without you knowing anything is wrong. Figure 5 shows explicitly which part of the flowchart would be blocked.

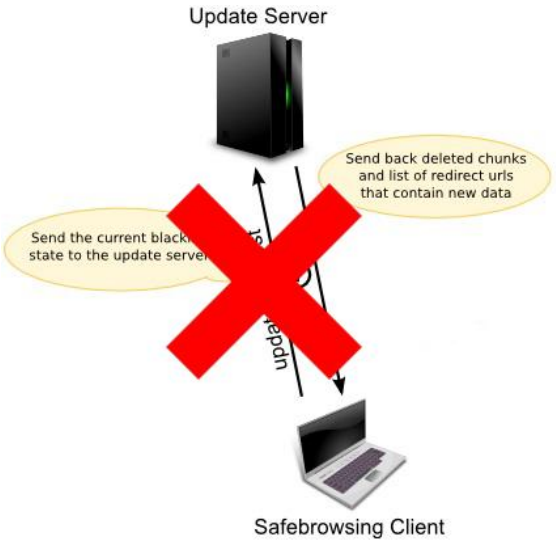

Fig. 5 - Blocking Update Request

IV.2.0 Hash Retrieval

Looking back at downloaded blacklist of URLs stored in Chrome. The browser requests for a full 32-byte hash whenever a local collision with an expression from the visited URL is matched with a blacklisted hashed expression. Figure 6 shows a diagram of how it works.

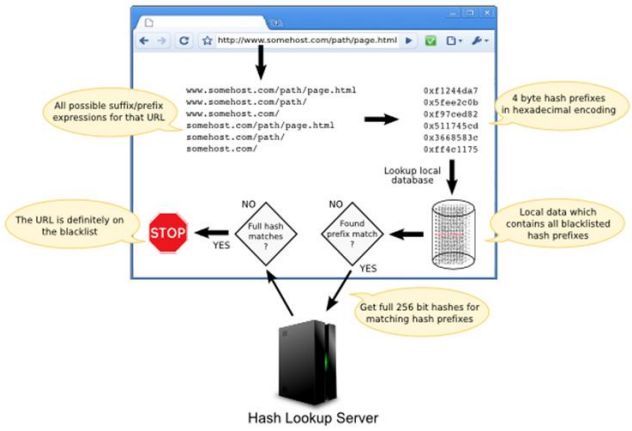

Fig. 6 - Looking Up a URL, taken from [3].

Similarly to their update process, the request for the hash lookup may be blocked, and Chrome will venture into websites that should have otherwise been verified as a risky website.

IV.3.0 Sandboxing

Google Chrome's Sandboxing feature isolates each new tab into processes. However, a work around for this feature caused the whole system to crash. To do this, a pop-up window in JavaScript is created that keeps moving its location around the screen. We used an endless loop that constantly changes the x, and y coordinates and calls JavaScript's "moveTo(X,Y);" function. This function moves the popup window to point x and y. When the user clicks on a link, a troublesome window will pop up thereby creating two tabs. The popup will then draw significant amounts of CPU usage. As the user tries to resume using Google Chrome, the popup window will cause both tabs to crash, and the user then has to restart the application.

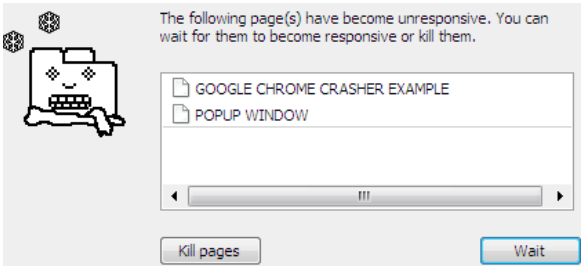

Fig. 7 - This image shows how we managed to crash 2 tabs with a malicious pop-up window Hence we were able to work around Google Chrome's sandboxing feature, as opening one tab caused it to affect another tab.

IV.4.0 JavaScript

Google Chrome protects it's users from malicious JavaScript by disabling it. However, this defense mechanism violates the design principle of psychological acceptability.

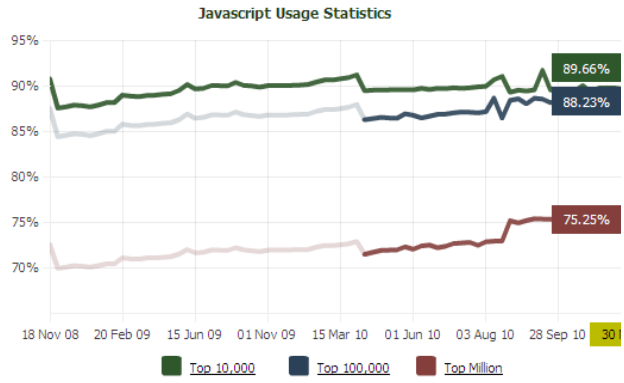

Fig. 8 - Percentage of JavaScript Users for top 10000, 100000, and Million sites from recorded on BuiltWith.com and Quantcast Top Million[6]

This table gives an estimate as to how much percentage of websites on the internet uses JavaScript. Hence it shows that a user, who has JavaScript blocker enable, will be unable to properly access more than 75.25% of the websites on the internet.

IV.5.0 Pop-Up

We have been using Chrome for the past few months and we realized that some websites still had popup windows even when we enabled the popup blocker. Therefore we looked into the functionality of the popup blocker. We tried to implement a popup, by adding this line of code in the body portion of a website:

```
<body
onLoad="window.open('http://www.example.com')">
```

We found that it does not work. The <body onLoad=> part also cannot call other JavaScript functions to resize or hide the window.

Therefore we concluded that Google Chrome disabled the body portion of the HTML code from calling any JavaScript functions. It also blocks hyperlinks from automatically opening a new window. However, we also tried to see if this blocks hyperlinks from opening new windows with JavaScript by adding this to the hyperlink code:

```
onclick="javascript:window.open"
```

We found that this can bypass the pop-up blocker and a new window will still be opened. Hence this shows that there is a flaw in the blocker as it does not stop all pop-ups from opening.

V. RESULTS

Through our trials, we were able to acquire several results. We were able to bypass the pop-up blocker by calling JavaScript's window.open function in the hyperlink portion of the HTML code.

Using Fiddler2, we were able to track Chrome when it's updating or retrieving hashed URL from Google's servers. Auto-responding to those hashed provided Chrome to operate as if nothing ever happened (no update request nor hash retrieval request).

V. DISCUSSION

Even after determining that pop-ups could still pop-up after the pop-up blocker being enabled, there is no serious security flaw. This may violate the psychological acceptability of users, but pop-ups are easily dealt with by closing them. A suggestion would be to disallow the HTML code in a hyperlink from calling JavaScript functions, similar to what they did with the <body>portion of the code, which blocked possible pop-ups to be opened.

For JavaScript blocking, instead of blocking all websites, a suggestion would be to introduce a blacklist of bad sites, and turning off JavaScript on those sites. However, if a blacklist were to appear, the update process would need to be considered.

Unfortunately, more time could and should have been spent in the analysis and experimenting with blocking HTTP communications between Chrome and Google's servers. This includes the update process and the looking up of an URL when a collision appears in the browser's blacklist. Since these processes happen in the background without the users consent, the user has to assume they're being protected without knowing if they're truly protected. This would violate the fail-safe default principal. The user would not be safe if updates were blocked. Research on the adversary getting access to the user's computer and network needs to be done. Rogue access points could be a viable entry point. A suggestion would be to notify the user and/or receive their consent similar to Windows 7 auto-updates. Tell the user their client is out-of-date, and allow them to choose to download updates or not, instead of running updates obviously to the user in the background. As long as the notification is persistent enough, but not annoying to users.

VII. CONCLUSION

Google Chrome is used by millions of users and this number is growing every day. Any security vulnerabilities would significantly affect these people and Google's reputation.

From our analysis, we are able to conclude that Google Chrome is a relatively safe browser to use. If an adversary or malicious program were to infiltrate the user's computer, Chrome users will receive minimal hints to an alternation of Chrome's update process and hash retrieval of blacklisted URLs. Although the JavaScript blocker violates Psychological Acceptability principle, it does not make the system vulnerable to attack if activated. However, we did by pass the pop-up blocker and this shows that the system is not perfect. Since the system is not perfect, we have reason to believe that it is possible that there are security vulnerabilities that have yet to be exploited.

ACKNOWLEDGMENT

We would like to thank Google, the Google Chrome Team, and the Chrome community for making a browser with modern flare. Documentation of the design was very helpful. Special thanks goes to EECE 412 Professor, Dr. Konstantin Beznosov and teaching assistant, San-Tsai Sun, for giving suggestions towards this project.

REFERENCES

[1] Google. "*Privacy and security settings: Phishing and malware detection*" (2010) [Online] Available: http://www.google.com/support/chrome/bin/answer.py?hl=en&answer=99020

[2] Ian Fette. "*Understanding Phishing and Malware Protection in Google Chrome*" Google. (2008-11-14) [Online] Available: http://blog.chromium.org/2008/11/understanding-phishing-and-malware.html

[3] Brian Ryner, Noe Lutz. "*SafeBrowsing Design*" Google. (2009-11-16) [Online] Available: http://code.google.com/p/google-safe-browsing/wiki/SafeBrowsingDesign

[4] Larry Seltzer. "Google Chrome's Security Practices Raise Eyebrows" *PCMAG.COM*. 05-18-2009 [Online] Available: http://www.pcmag.com/article2/0,2817,2347216,00.asp

[5] Leena Rao. (2010-05-19). "*Google I/O: Chrome Now 70 Million Users Strong*". TechCrunch. Available: http://techcrunch.com/2010/05/19/google-io-chrome-now-70-million-users-strong/

[6] Trends, BuiltWith. "*Javascript Usage Statistics*" (2010-11-30) [Online] Available: http://trends.builtwith.com/docinfo/Javascript

[7] Google. "*Google Chrome Features*". (2010) [Online] Available: http://www.google.com/chrome/intl/en/more/features.html