Hacking Google Chrome OS

Taking Down the Browser from the Inside

Kyle Osborn, Application Security Specialist, WhiteHat Security
Matt Johansen, Application Security Team Lead, WhiteHat Security

August 2011

Introduction
Google Chrome has been called one of the most secure browsers. In fact, Google offered a $20,000 bounty on any research that leads to an attacker escaping the sandbox that Chrome provides.

”Sandboxing helps prevent malware from installing itself on your computer or using what happens in one browser tab to affect what happens in another. “ - Google Chrome and Browser Security - http:// www.google.com/chrome/intl/en/more/security.html

Google has emphasized a move away from Desktops to the Cloud, using their Chromium Browser in a Desktop format, under the Google Chrome OS. For users, this means: little to no hard drive usage, low CPU specs, overall cheaper hardware, and the security that the cloud provides.

Why a Web Hacker can Ignore the Sandbox
Google’s drive to move away from the desktop, and into the cloud results in desktop applications being replaced with HTML5 & JavaScript rich extensions. These new “desktop programs” seem to be more secure, because they do not have the classic vulnerabilities that desktop applications end services have--buffer/stack/heap overflows/underflows, format string attacks,plus many more. Since exploitation no longer leads to shell, the real dangers and implications of any exploit seem to be mitigated.

Unfortunately, this is not true. HTML and Javascript applications (Chrome Extensions) are now vulnerable to standard HTML and Javascript attacks. The most serious, in this situation, is Cross Site Scripting. By utilizing an XSS vulnerability in an extension, an attacker can pivot from that extension, and take advantage of the permissions given to it to attack and gain access to user information loaded in other tabs.

Chrome Extension APIs
Chrome Extensions are based on the same rules that websites are, like the Same Origin Policy (SOP). These extensions operate in a chrome-extension://<extensionID>/popup.html context. However, because this technically violates the Same Origin Policy, how do rich applications interact with web-based applications? How does an RSS feed reader access feeds from multiple websites, which should violate the SOP, but do not?
Google solved this problem by introducing extension APIs, which allow developers to choose what extensions have access to. These extensions are known as the ‘chrome.* APIs’

Extensions/Permissions in bold are potentially dangerous or vulnerable to attack.

Chrome Extension APIs:

chrome.bookmarks	**Extension can access and manipulate Chrome bookmarks**
chrome.browserAction	Extension can place a button in the addres bar
chrome.contextMenus	Extension can extend context menus in Chrome
chrome.cookies	**Extetion can access and manipulate cookies**
chrome.extension	Allows extensions to exchange messages between themselves
chrome.history	**Extention can add, delete and query recorded browser history**
chrome.i18n	Localization API
chrome.idle	**Extension can tell if the machine is in an "idling" state**
chrome.management	**Extension can monitor list of installed/running extensions**
chrome.omnibox	Extension can control Omnibox address bar
chrome.pageAction	**Icons inside address bar that represent actions on that page**
chrome.tabs	**Extension can read & control browser tabs (and potentially contents)**
chrome.windows	**Extension can read & control browser windows**

Extra Chrome extension permissions:

match pattern	**Allow extension to access scheme://hostname/path (adds Same Origin Policy exception)**
background	**Extension runs early and shuts down late, forcing Chrome to start at login**
chrome.experimental	Experiemental APIs, not allowed to be uploaded to the Webstore
geolocation	**Extension can use HTML5 geolocation without prompting user**
notifications	Extension canuse HTML5 notification API without permissions
unlimitedStorage	Extension has unlimited client-disde, localStorate data.

manifest.json

When developing extensions, the file manifest.json is denoted as the file that contains all the meta-information of the application. This includes title, version, resources, APIs it has access to, and permissions. The file is a JSON data array that includes all this information. There is next to no security information about what a developer should and should not add when creating this file.

XSS inside an extension, now what?

Once a Cross Site Scripting vector has been identified in an application, an attacker can take advantage of the extension APIs and permissions the developer has given them to access other websites regardless of any sandbox protections.

Examples

1) Execute javascript in all available tabs (with permissions from 'match pattern')

```
chrome.windows.getAll({"populate": true}, function (windows) {
     for (count in windows) {
          chrome.tabs.getAllInWindow(windows[count]['id'], function (tabs) {
               for (tabIndex in tabs) {
                    chrome.tabs.executeScript(tabs[tabIndex]['id'],
                         {code: "alert(document.domain)"});
```

2) If 'match pattern' is '*://*/*' (common mistake by developers)

```
test = new XMLHttpRequest();
test.open('get', 'http://mail.google.com/mail/')
test.send()
test.onreadystatechange = function () {
     if (test.readyState == 4 && test.status == 200) {
          alert(test.responseText) // alert text of gmail
     }
}
```

Conclusion

Next-generation operating systems, like Google Chrome, are raising new security issues. Now, we are seeing the evolution of the software security model into the browser extension trust model. Google Chrome OS raises security issues that apply across the board to cloud applications. How do you evaluate security on an application that you do not own? Google is working with extension developers to encourage them to be cognizant of security during the development process. For enterprises, the lesson here is to beware of extensions the users of Chrome OS machines are using and to realize that even though sensitive information isn't stored on the hard drive does not make it safe.

About WhiteHat Security, Inc.

Founded in 2001 and headquartered in Santa Clara, California, WhiteHat Security provides end-to-end solutions for Web security. The company's cloud technology platform and leading security engineers turn verified security intelligence into actionable insights for customers. Through a combination of core products and strategic partnerships, WhiteHat Security provides complete Web security at a scale unmatched in the industry. WhiteHat Sentinel, the company's flagship product line, currently manages more than 4,000 websites - including sites in the most regulated industries as well as top e-commerce, finance and healthcare companies.

To improve your organization's Web security, sign up for a custom, no-cost 30 day security evaluation by WhiteHat Sentinel SecurityCheck at www.WhiteHatSec.com