



Browser Security Comparison

A Quantitative Approach

Document Profile

Version 0.0

Published 12/6/2011

Revision History

Version	Date	Description
---------	------	-------------

0.0	12/26/2011	Document published.
-----	------------	---------------------

Contents

Authors.....	v
Executive Summary.....	1
Methodology Delta	1
Results.....	2
Conclusion.....	2
Introduction	3
Analysis Targets	4
Analysis Environment.....	4
Analysis Goals	4
Browser Architecture.....	5
Google Chrome	5
Internet Explorer.....	5
Mozilla Firefox.....	6
Summary	6
Browser Comparison.....	8
Historical Vulnerability Statistics	8
Browser Comparison.....	8
Issues with Counting Vulnerabilities.....	9
Issues Surrounding Timeline Data	10
Issues Surrounding Severity.....	11
Issues Unique to Particular Vendors.....	11
Data Gathering Methodology	13
Update Frequencies.....	13
Publicly Known Vulnerabilities.....	16
Vulnerabilities by Severity	17
Time to Patch	18
URL Blacklist Services.....	20
Comparing Blacklists	20
“Antivirus-via-HTTP”	20
Multi-Browser Defense	20
Comparing Blacklist Services.....	21

Comparison Methodology	21
Results Analysis	21
Conclusions	25
Anti-exploitation Technologies	26
Address Space Layout Randomization (ASLR)	26
Data Execution Prevention (DEP)	26
Stack Cookies (/GS)	26
SafeSEH/SEHOP	26
Sandboxing	27
JIT Hardening	28
Browser Anti-Exploitation Analysis	31
Browser Comparison	32
Google Chrome	34
Microsoft Internet Explorer	45
Mozilla Firefox	58
Browser Add-Ons	67
Browser Comparison	68
Google Chrome	69
Internet Explorer	80
Firefox	89
Add-on summary	97
Conclusions	98
Bibliography	100
Appendix A – Chrome Frame	I
Overview	I
Decomposition	II
Security Implications	III
Risk Mitigation Strategies	V
Conclusion	V
Bibliography	VI
Appendix B	I
Google Chrome	I



Internet Explorer.....	XIII
Mozilla Firefox.....	XVIII
Tools.....	I

Authors

Listed in alphabetical order:

- Joshua Drake (jdrake@accuvant.com)
- Paul Mehta (pmehta@accuvant.com)
- Charlie Miller (charlie.miller@accuvant.com)
- Shawn Moyer (smoyer@accuvant.com)
- Ryan Smith (rsmith@accuvant.com)
- Chris Valasek (cvalasek@accuvant.com)

Executive Summary

Accuvant LABS built criteria and comparatively analyzed the security of Google Chrome, Microsoft Internet Explorer, and Mozilla FireFox. While similar comparisons have been performed in the past, previous studies compared browser security by considering metrics such as vulnerability report counts and URL blacklists. This paper takes a fundamentally different approach, examining which security metrics are most effective in protecting end users and evaluating those criteria using publicly available data and independently verifiable techniques.

Methodology Delta

Most attempts to compare the security of different vendors within a software class rely on statistical analysis of vulnerability data. The section entitled [Historical Vulnerability Statistics](#) and its subsections examine publicly available vulnerability data and discuss why such an approach is limited in its usefulness for comparatively assessing security.

In contrast, we believe an analysis of anti-exploitation techniques is the most effective way to compare security between browser vendors. This requires a greater depth of technical expertise than statistical analysis of CVEs, but it provides a more accurate window into the vulnerabilities of each browser. Accuvant LABS' analysis is based on the premise that all software of sufficient complexity and an evolving code base will always have vulnerabilities. Anti-exploitation technology can reduce or eliminate the severity of a single vulnerability or an entire class of exploits. Thus, the software with the best anti-exploitation technologies is likely to be the most resistant to attack and is the most crucial consideration in browser security.

An important difference between this paper and previous studies is that we've made our data and the tools used to derive the data available for scrutiny. Previous attempts have been made to compare [Historical Vulnerability Statistics](#) and [URL Blacklist Services](#); however, those studies' conclusions have differed wildly from this paper's results, and the difference in outcomes arises largely from the choice of data sources. We believe our own data is correctly representative of the population and have made it, along with our tools and methodologies, available to test this belief. Finally, we invite others to examine the tools for issues, or to extend and improve on them to encompass more criteria.

We hope this paper presents readers with a definitive statement as to which browser is currently the most secure against common attacks, and provides criterion that vendors may use to measure and improve the security posture of their browsers. Finally, it is our hope that this is helpful to others who work to evaluate browser security, and that they will reciprocate the open nature of this effort to help eliminate unverifiable data and conclusions.

Results

The following graph shows the results of our analysis:

Criteria	Chrome	Internet Explorer	Firefox
Sandboxing	✓	●	✗
Plug-in Security	✓	●	✗
JIT Hardening	✓	✓	✗
ASLR	✓	✓	✓
DEP	✓	✓	✓
GS	✓	✓	✓
URL Blacklisting	✗	✗	✗

- ✓ Industry standard
- Implemented
- ✗ Unimplemented or ineffective

Conclusion

The URL blacklisting services offered by all three browsers will stop fewer attacks than will go undetected. Both Google Chrome and Microsoft Internet Explorer implement state-of-the-art anti-exploitation technologies, but Mozilla Firefox lags behind without JIT hardening. While both Google Chrome and Microsoft Internet Explorer implement the same set of anti-exploitation technologies, Google Chrome's plug-in security and sandboxing architectures are implemented in a more thorough and comprehensive manner. Therefore, we believe Google Chrome is the browser that is most secured against attack.

Introduction

From the cellular phone to the desktop, the web browser has become a ubiquitous piece of software in modern computing devices. These same browsers have become increasingly complex over the years, not only parsing plaintext and HTML, but images, videos and other complex protocols and file formats. Modern complexities have brought along security vulnerabilities, which in turn attracted malware authors and criminals to exploit the vulnerabilities and compromise end-user systems. This paper attempts to show and contrast the current security posture of three major Internet browsers: Google Chrome, Microsoft Internet Explorer and Mozilla Firefox.

The following sections (Anti-Exploitation Technologies, Browser Anti-Exploitation Analysis and Browser Add-Ons) cover anti-exploitation technologies for the browsers and their add-ons. First a general discussion of anti-exploitation technologies, followed by more detailed information and comparisons of each browser's anti-exploitation and add-on capabilities. Lastly, our conclusions based on the aforementioned information and comparisons.

All information enumeration techniques that were automated are provided in a separate archive, so results can be reproduced, analyzed and challenged by third parties if so desired.

We concluded the research for this paper in July 2011. Changes and updates may occur after this paper is released. We may attempt to update the paper or develop errata to deal with the security evolution of each assessed browser.

Finally, readers should understand that, while Google funded the research for this paper, Accuvant LABS was given a clear directive to provide readers with an objective understanding of relative browser security.

The views expressed throughout this document are those of Accuvant LABS, based on our independent data collection.

Analysis Targets

The following targets were selected for analysis. These targets were selected for their market share. As of July, 2011 a combination of Google Chrome, Microsoft Internet Explorer and Mozilla Firefox represent 93.4% of all users accessing the Internet [W3_Schools_Market_Penetration]. While other browsers would have been interesting to compare, in the interest of time they were excluded from this study.

Google Chrome

Google, Inc. develops the Google Chrome web browser. Google released the first stable version of Chrome on December 11, 2008. Chrome uses the Chromium interface for rendering, the WebKit layout engine and the V8 Java Script engine. The components of Chrome are distributed under various open source licenses. We included Google Chrome versions 12 (12.0.724.122) and 13 (13.0.782.218) in our evaluation.

Microsoft Internet Explorer

Microsoft develops the Internet Explorer web browser. Microsoft released the first version of Internet Explorer on August 16, 1995. Internet Explorer is installed by default in most current versions of Microsoft Windows, and components of Internet Explorer are inseparable from the underlying operating system. Microsoft Internet Explorer and its components are closed source applications. We evaluated Internet Explorer 9 (9.0.8112.16421).

Mozilla Firefox

Mozilla develops the Firefox web browser. Mozilla released the first version was released on September 23, 2002. Firefox uses the Gecko layout engine and the SpiderMonkey JavaScript engine. The components of Firefox are released under various open source licenses. Firefox 5 (5.0.1) was evaluated for this project.

Analysis Environment

All targets were analyzed while running on Microsoft Windows 7 (32-bit). MacOS X, Linux and other operating systems were excluded from the analysis to simplify analysis tasks, provide timely and relevant information, and to increase applicability for the majority of users. Windows 7 was chosen over other variants in order to compare the latest operating system supported security measures. While it is regrettable that other environments and targets were excluded from the analysis, the sheer magnitude of material to cover combined with the pace that browser technologies evolve led to these constraints.

Analysis Goals

The goal of our analysis was to provide a relevant and actionable comparison of the security of the three web browsers. Additionally, since there are several other papers that address this goal, we have included similar metrics in our analysis. While some of these parity metrics have noted flaws, it was our goal to expose those flaws so readers would be aware of them and not view their omission as oversight.

Browser Architecture

Browsers have evolved over time, taking on characteristics that were classically the domain of the operating system. Recent browser architecture uses a combination of multi-process and multi-threaded architecture to provide security barriers and trust zones. In the following sections, we will describe individual browsers' process architecture and trust zones and how these browsers function across process boundaries.

Google Chrome

Chrome uses a medium integrity broker process that manages the UI, creates low integrity processes and further restricts capabilities by using a limited token for a more comprehensive sandbox than the standard Windows low integrity mechanism. These processes are created for rendering tabs, hosting plug-ins and extensions out of process and GPU acceleration. The broker process creates named pipes for inter-process communication.

chrome.exe	1036	Medium	
chrome.exe	3052	Low	-type=extension -lan
chrome.exe	3848	Low	-type=plugin -plugin
chrome.exe	5288	Low	-type=renderer -lang
chrome.exe	5680	Low	-type=gpu-process --

- Multi-Process
- Low Integrity Limited Token
 - Comprehensive sandboxing
- Out of Process
 - Renderer
 - Plug-ins (Flash, Silverlight, etc.)
 - Extensions
 - GPU acceleration
- Process Based Site Isolation

The extensive use of sandboxing limits both the available attack surface and potential severity of exploitation. A compromised renderer process would only have access to the current process and what is made available through the broker process IPC mechanism. The compromised process would need a method of privilege escalation from low integrity with a limited token in order to persist beyond the process.

Internet Explorer

Internet Explorer uses the "loosely coupled IE" [MSDN_LCIE] model where the UI frame and tabs are largely independent of each other, which allows for the browser tab processes to function at low integrity. A medium integrity broker process creates the low integrity tabs used for browsing, hosting ActiveX controls, GPU acceleration and manages activity independent of tabs such as downloads and toolbars.

ieexplore.exe	3240	Medium	"C:\Program Files (x86)\
ieexplore.exe	4588	Low	"C:\Program Files (x86)\
ieexplore.exe	5580	Low	"C:\Program Files (x86)\
ieexplore.exe	3960	Low	"C:\Program Files (x86)\

- Multi-Process
- Low Integrity
 - Sandboxing
- Out of Process
 - Tabs
- In-Process Plug-ins
- Crash & Hang recovery

In the event of a crash, the tab is automatically reloaded the first time, allowing malicious content multiple attempts to succeed, or have an unsuccessful exploit attempt go unnoticed. A tab compromised by an exploit would have read access to the file system and any low integrity process, including other browser tabs. The compromised process would need a method of privilege escalation from low integrity to persist beyond the browser session.

Mozilla Firefox

Firefox uses a single process medium integrity browser process which contains the entire browsing session including all tabs, add-ons, GPU acceleration and more in a single address space, with the exception of plug-ins like Flash and Silverlight. Plug-ins are hosted out of process and independent of each other at medium integrity. A crash in the browser process would take down the entire browser and all plug-in processes. Alternatively, a crash in a plug-in process would be isolated to that single process.

firefox.exe	3912	Firefox	Medium
plugin-container.exe	6104	Plugin Container f...	Medium

- Single-Process Browser
- Out of Process Plug-ins
 - Medium Integrity
 - Flash, Silverlight
- In-Process Add-Ons

A compromised browser or plug-in process would not require privilege escalation to persist beyond the browser process.

Summary

The following screen shot shows the different browsers as they appear after browsing common sites. It is easy to see the different processes that are spawned and the different integrity levels for each process.

firefox.exe	360	Medium	"C:\Users\Paul\AppData\Local\Mozilla\Firefox\Mozilla Firefox 6\firefox.exe"
plugin-container.exe	3064	Medium	"C:\Users\Paul\AppData\Local\Mozilla\Firefox\Mozilla Firefox 6\plugin-container.exe" --channel=360.f
chrome.exe	5880	Medium	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe"
chrome.exe	2072	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	2976	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	2996	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	4244	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	2324	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	5180	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
chrome.exe	3592	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=renderer -lang=en-Gl
rundll32.exe	2296	Medium	C:\Windows\system32\rundll32.exe "C:\Users\Paul\AppData\Local\Google\Chrome\APPLIC~1\1307
chrome.exe	3240	Low	"C:\Users\Paul\AppData\Local\Google\Chrome\Application\chrome.exe" -type=plugin --plugin-path=
iexplore.exe	5732	Medium	"C:\Program Files (x86)\Internet Explorer\iexplore.exe"
iexplore.exe	4476	Low	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" SCODEF:5732 CREDAT:203009
iexplore.exe	4132	Low	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" SCODEF:5732 CREDAT:137475
iexplore.exe	4000	Low	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" SCODEF:5732 CREDAT:203012
iexplore.exe	3216	Low	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" SCODEF:5732 CREDAT:137506
iexplore.exe	6076	Low	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" SCODEF:5732 CREDAT:137562

Figure4. Browser processes overview

The table below shows the processes by function and the integrity levels granted to each. A process with a higher integrity level represents a greater value for an attacker to compromise; however, with most of the higher integrity processes, an attacker can only interact with a very small attack surface.

Process Name	Pid	Integrity Level	Limited Token	Description
chrome.exe	5880	Medium	No	Chrome Main Broker
chrome.exe	2072	Low	Yes	Chrome Renderer
chrome.exe	3956	Low	Yes	Sandboxed Flash plug-in
iexplore.exe	5732	Medium	No	IE UI Frame
iexplore.exe	4476	Low	No	IE Low Integrity Browser
firefox.exe	360	Medium	No	Firefox browser
plug-in-container.exe	3064	Medium	No	Plug-in container for Firefox

Figure5. Browser security overview

With multiple processes and limited communication channels between processes, modern browsers provide a unique exploitation target. Merely compromising the browser, in some cases, is not enough for a compromise to persist past the life of the browser process. The following sections look at how these security barriers are implemented in order to determine which browsers provide the strongest resistance to compromise.

Browser Comparison

Category	Google Chrome	Internet Explorer	Mozilla Firefox
Vulnerability Patching	✓	✓	✓
Safe browsing API	✓	✓	✓
Sandboxing	✓	●	✗
JIT Hardening	✓	●	✗
Plug-in Architecture	✓	✓	✓

- ✓ First rate implementation
- Implementation has deficiencies
- ✗ Not implemented

Figure6. Browser comparison

Historical Vulnerability Statistics

One of the key factors to browser security is ensuring the browser is up-to-date and has the latest security patches. Each browser vendor has devised its own update methodology; relying on their own infrastructure to deliver updates. Furthermore, vendors have their own processes and procedures for handling, tracking, fixing and ultimately disclosing vulnerability information. Many statistics can be collected and analyzed by examining data from the execution of these processes. However, these statistics can be misleading when used to compare the relative security posture of the software. By analyzing the aforementioned points in finer detail, we hope to shed some light on the nuances of each vendor's approach, and the relative ease with which these statistics can be misappropriated to arrive at a conclusion.

Browser Comparison

For some of the other cross-browser test cases in this paper, the results are clear-cut. A browser's architecture or defensive model either blocks a given attack vector, or it does not. As described in this section of this document, it is difficult to draw provably unbiased conclusions when each browser project's datasets differ in so many ways. A great deal of data is available, but the true quality of that data and its usefulness as a metric of browser security is questionable.

In general, a move toward greater transparency in the security update process would benefit consumers, and create a level playing field if metrics such as vulnerability severity and the timeline from disclosure to release of updates are to be truly beyond the realm of being merely marketing material. While Accuvant LABS did not approach Microsoft for internal statistics on privately identified vulnerabilities and vulnerabilities with undisclosed remediation timelines, it is likely that these statistics exist, and could open the door for an unambiguous debate about each project's true response time.

Issues with Counting Vulnerabilities

In the past, studies have compared browser security by comparing the number of advisories that affect each browser within a specific period. Advisory comparisons may be quite popular due to the availability of data but problems arise when vendors issue advisories in order to advise users to install patches, not to generate statistical vulnerability information. Since the intent of issuing advisories and that of collecting statistics regarding numbers of advisories differ, problems arise during statistical analysis. Vendors may fold several unique vulnerabilities into a single advisory, fold unacknowledged vulnerabilities and one or more acknowledged vulnerabilities into a single advisory or issue a code fix for a software defect without announcing that the defect has security implications. These situations introduce errors into any numeric analysis of comparative browser security as a result of asymmetry between use and intent. Although they do not adversely affect an end-user whose goal is to patch, this asymmetry weakens the foundation of any propositions extrapolated from the data.

Every advisory that a vendor releases requires time and effort to document. If the vendor can fold multiple vulnerabilities into a single advisory, the amount of time and effort expended is reduced while still allowing end-users to understand the need to patch. Accuvant made an effort to mitigate this issue by using semi-manual analysis consisting of regular expression searches and manual review of the advisory text. While some errors may still exist, many were fixed within the collected data.

Some vendors will discover vulnerabilities internally and release fixes for these vulnerabilities alongside patches for publicly reported vulnerabilities. Microsoft has stated that their policy is to not report internally discovered vulnerabilities [MSDN_SilentPatches]. Additionally, it is not beyond the realm of possibility that a patch meant to address one vulnerability closes a completely separate one that was never discovered. In order to properly account for both of these scenarios, every patch would have to be analyzed to determine each issue that was intentionally patched, and whether the patch closes issues that would have otherwise existed. It is generally accepted that it is impossible to find every vulnerability for a sufficiently complex system, and even in this reduced case, the likelihood of misses is intuitively high. Accuvant did not account for this type of error within the dataset.

For browsers such as Firefox and Google Chrome, patches are issued in order to address software defects alongside security patches. As an example, if a font is rendered improperly within the browser, an update may be released to render the font correctly. However, by modifying the code, unless the developer is aware of all potential implications of their patch, the developer may inadvertently mitigate an undiscovered vulnerability in the code.

If a developer could predict every implication of changing a small piece of code, there would be no need to put a piece of software through QA for even the smallest code change. Therefore, it is likely safe to assume that there are vulnerabilities that have been addressed but are not represented within the data based on this scenario. Due to the complexity and time required to mitigate this error, Accuvant did not account for this type of error within the dataset.

Though this is not a thorough and complete account of possible errors within the dataset, they are representative of issues surrounding vulnerability counting. While setting up statistical measures for

advisories and drawing conclusions from these measures is logically attractive and provides a cute graphic, vulnerability counts within software are neither ordinal nor can a complete set be derived. However, in the interest of parity with other documents comparing browser security, the following sections will display statistical measures of the ameliorated data.

Issues Surrounding Timeline Data

Another seemingly useful measure of vulnerability data is timeline information. When a vulnerability is first reported or exploited in the wild and patched by the vendor seem like interesting and security relevant metrics. The only sources of timeline data, outside of the software vendor companies, are the public advisories and bug tracking systems. The intent of advisories is to notify end users that they should patch and the intent of bug tracking systems is to ensure bugs are reported and remediated, whereas our use is to derive meaningful statistics. Again, due to this asymmetry, there are issues that arise when extracting timeline information.

The first issue with timeline information stems from extracting the information from bug tracking systems. Since bug tracking systems are used for the purpose of ensuring bugs are patched, the participants may perform actions that obfuscate the time information. One example is bug duplication. If a vulnerability is reported twice in the tracking system, and the disclosure points to the most recent bug instance, then the date will be off. Another example: a vendor may receive notification of a vulnerability and begin work without immediately entering the vulnerability into the bug tracking database. In this scenario, the data will suggest a patch window of shorter duration than what actually took place. Accuvant made no attempt to ameliorate this discrepancy.

The second issue with timeline information stems from non-reporting. Microsoft does not make their bug tracking database public and the only source of vulnerability information is contained within the security advisories. However, the Microsoft security advisories do not provide timeline information. Third parties such as VeriSign iDefense and HP TippingPoint provide a timeline of disclosure, and Accuvant used these third party timelines.

The third issue with timeline information surrounds 0-day exploitation. Generally, when a vulnerability is exploited in the wild without vendor notification, the public only learns of the exploitation when a third party makes the exploitation known. A vendor may learn of the exploitation prior to the public and begin working on a patch. If the vendor does not admit to prior knowledge of exploitation, or provide a timeline, then the best date that can be derived is the date the public was informed. Accuvant used the public date for all 0-day exploitation timelines.

While these three issues are representative of problems encountered when extracting timeline data, this is by no means an exhaustive list. Without a vendor implementing strict and rigorous cataloging of when vulnerability information is first received, it is impossible to determine the exact time it takes to patch.

Issues Surrounding Severity

The severity of issues is another metric that appears interesting to compare. If one browser has more “critical” patched vulnerabilities, one might assume that particular browser is less secure because the other browsers do not have as much critical vulnerability. Another individual might assume that the browser with more patched critical vulnerabilities is more secure because the other browsers may have more undiscovered critical vulnerabilities. However, the truth of the matter is far more complex.

There are no solid industry accepted metrics for rating the criticality of vulnerabilities for every possible environment. CVSS, DREAD and several other vulnerability ranking systems are available; however, all of them include subjective components to arrive at an overall score. Additionally, each vendor may choose their own ranking methodology to arrive at a ranking for their advisories. These facts weaken any cross-browser comparisons unless each vulnerability is analyzed and ranked by a single person and all subjective criteria are removed.

Another issue involves making judgment calls regarding the severity of vulnerabilities. If a vulnerability cannot be exploited, it is easy to say that the severity of the vulnerability is low. However, since each vulnerability is unique and exploitation of vulnerabilities is an art, many of these judgment calls can be flawed. One such example MS-08-001 [MSDN_MS08001], and the resulting paper released by Immunity at [Immunity_Exploitability_Index]. Given that even a vendor can misunderstand the implications of vulnerabilities, it is easy to see that a third party may not be qualified to provide a precise severity label.

Another issue surrounds vulnerability chaining. Since vulnerabilities are really just pieces of code that allow an attacker to perform operations that were not intended, a single operation may not qualify as high severity. However, if many low severity unintended operations can be combined in unique ways, then the overall chain of operations may qualify as high severity.

Comparing vulnerabilities across vendors can lead to many issues because of a fundamental difference in how these vulnerabilities are ranked. Applying a ranking system can be subjective, and errors made due to novel exploitation strategies. An issue’s severity in isolation may be very different than the same vulnerability combined with others. Therefore, any security conclusions drawn based on severity metrics are going to be subjective.

Issues Unique to Particular Vendors

Each vendor also presented unique issues when collecting vulnerability data. The following subsections describe problems with individual browsers.

Internet Explorer

As previously discussed, collecting data for Internet Explorer was particularly challenging due to the closed nature of development at Microsoft. Beyond the challenges of data collection, we encountered several other difficulties during research and data collection.

In several Microsoft security bulletins, some CVEs are mentioned as having been publicly disclosed without any public reference. In some cases, Microsoft may have been alerted to information from an obscure source. In these cases, it was not possible to obtain a valid tracking date.

One considerable piece of complexity that is specific to collecting data for Internet Explorer is the way that Microsoft breaks down their security bulletins into various products. For example, when vulnerabilities are reported in Microsoft's JScript and VBScript engines, Microsoft creates a separate bulletin for that product. Despite the fact that these products directly affect the security posture of Internet Explorer, no Internet Explorer security bulletin was released. This differs from Chrome and Firefox, who both ship their own respective JavaScript engines. Accuvant included a number of Microsoft Security Bulletins that affect critical browser components in the interest of data amelioration.

Conversely, some vulnerabilities that were exploitable via Internet Explorer were not included. One such issue was CVE-2009-2495. We did not include the bulletin containing this CVE since it affects Visual Studio and additional third party applications built with Visual Studio. We did not include bulletins that were for non-essential or non-default Windows components.

Firefox

Despite the open nature of Firefox development, we encountered several issues while collecting data. First, Mozilla tends to group many issues together under the generic heading "Crashes with evidence of memory corruption" [Mozilla_Crashes_Evidence]. Fortunately, Mozilla includes all related bug numbers for these advisories. This inclusion allowed Accuvant to split these issues apart based on bug number, tracking each one individually.

The last Mozilla specific issue occurred when gathering bug report dates. Accuvant encountered tickets that were not accessible. It is possible that tickets were never opened despite the issues having been publicly disclosed. For these twelve bugs, time-to-patch information is not available.

Chrome

Unlike Mozilla and Microsoft, the Chrome team does not release formal security advisories. Instead, security relevant bugs that are fixed are posted to the Chrome Stable Release blog. For releases prior to 3.0.195.25, detailed bug fix information is available from the development channel release notes [Chromium_Release]. When gathering data from the Chromium release notes, Accuvant excluded posts that did not contain any security fixes or those that included only an updated Flash Player.

Another issue that cropped up deals with Chrome's version scheme. For the sake of consistency, Accuvant devised a custom milestone numbering scheme derived from the first two parts of the version number and a counter. The counter is incremented for each security-relevant release. For example, the second security fix release for Chrome 10 would be called "m10.0u2".

Although Chrome ships with a customized version of Flash Player, vulnerabilities affecting Flash will not be included in the analysis. Flash was excluded in order to present only the vulnerabilities inherent to the Chrome browser.

Similar to Firefox data collection, date information was gathered from the public Chrome bug tracker. Unfortunately, a large number bugs were not publicly accessible. In those cases, the dataset was augmented with data supplied by Google.

Data Gathering Methodology

Accuvant attempted to generate a dataset that was granular to the individual vulnerability level to avoid issues arising from vendors folding multiple vulnerabilities into a single CVE. After gathering information about advisory releases, discussed further in the “Security Updates” section below, Accuvant proceeded to examine each issue individually. For each issue, the following information was collected and manually checked for consistency: vendor bug identifier, CVE identifier, severity, date reported and date disclosed.

The resulting dataset, which covers the period between January 1, 2009 and June 28, 2011, was used throughout the rest of this section. The dataset includes versions of Firefox from 2.0 to 5.0, versions of IE from IE6 to IE9 and all stable releases of Chrome.

Update Frequencies

When designing a security update program, each vendor has policies and procedures in place to perform QA and, subsequently release the updates to end users. Browser development teams operate on a pre-set schedule for major version releases. This preset schedule is apparent within the data collected.

In addition to major releases, browser manufacturers also routinely provide updates that specifically address security vulnerabilities and other urgent issues. In some rare cases, such as when widespread attacks are taking place on the Internet, vendors will issue emergency updates. These emergency updates differ from periodic updates because the quality assurance cycle faster than usual, and end-user communication needs to reach a wide audience. Different vendors have varying difficulties in executing emergency patch updates, and this shows in the data.

The following sections provide some analysis for the patch data. The differences between vendors are demonstrative of different development practices and overhead in the patching process. Although it is tempting to derive conclusions from the graphs, the only fair conclusion is that they are just different.

Internet Explorer

By examining the frequency of Microsoft Security Bulletins with the title “Cumulative Security Update for Internet Explorer”, as seen in Figure 6, one can deduce that the IE team typically aspires for a two-month release cycle. In some cases, such as MS09-034 or MS10-002, Microsoft deviated from their cycle. Both of these deviations were necessitated by outside pressure. Other than those examples, Microsoft’s release process for bulletins with the title “Cumulative Security Update for Internet Explorer” is very regular.

IE Security Updates

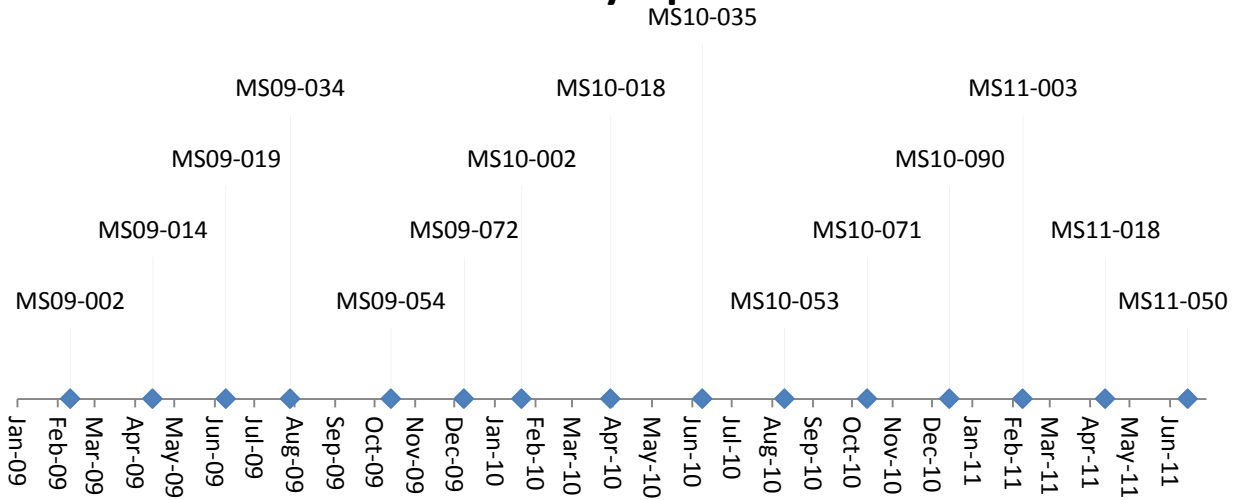


Figure 1. Cumulative Security Update for Internet Explorer

As previously noted, Microsoft tends to split components that directly affect Internet Explorer from Internet Explorer-related advisories. When all Internet Explorer-related updates are included within the timeline, the overall impression garnered from the graphs is that updates occur much less regularly.

IE and Related Security Updates

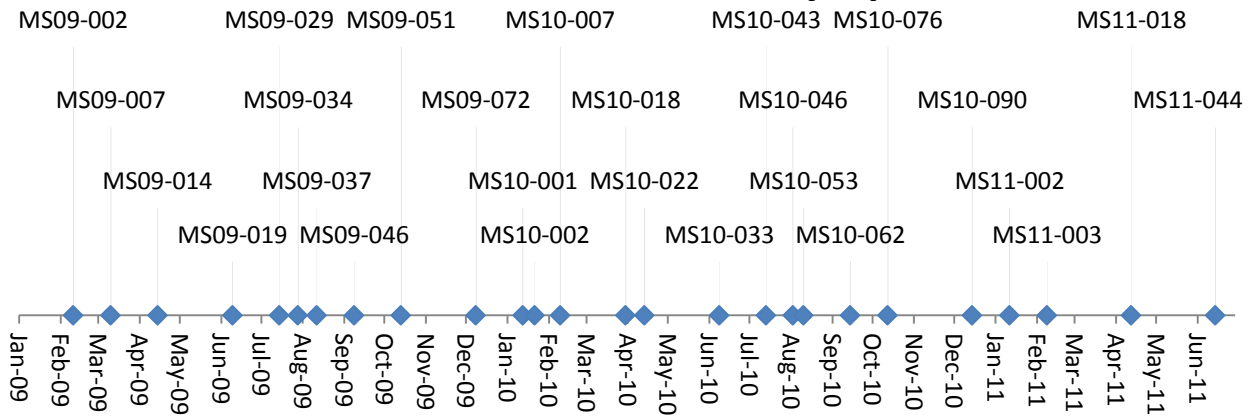


Figure 2. Updates not released under "Cumulative Security Updates for Internet Explorer"

This irregularity may be an artifact resulting from the divisions between development groups at Microsoft, or it may be due to different quality assurance processes applied to particular patches. In either case, a less regular update schedule has no direct impact on security. While it may be harder to apply updates on a non-scheduled basis, this difficulty is indicative of issues in patch deployment infrastructure rather than something that is intrinsic to the browser.

Firefox

The Firefox team is less predictable when releasing updates for its suite of products. As seen in Figure 9, Firefox has no pre-set pattern that determines release updates. In some instances, Mozilla has released updates in quick succession, within only a few days. Other times, up to three months passed without an update release. Note that this data treats multiple advisories released on the same day as a single update event. In some cases, Mozilla has released as many as 15 advisories on the same day.

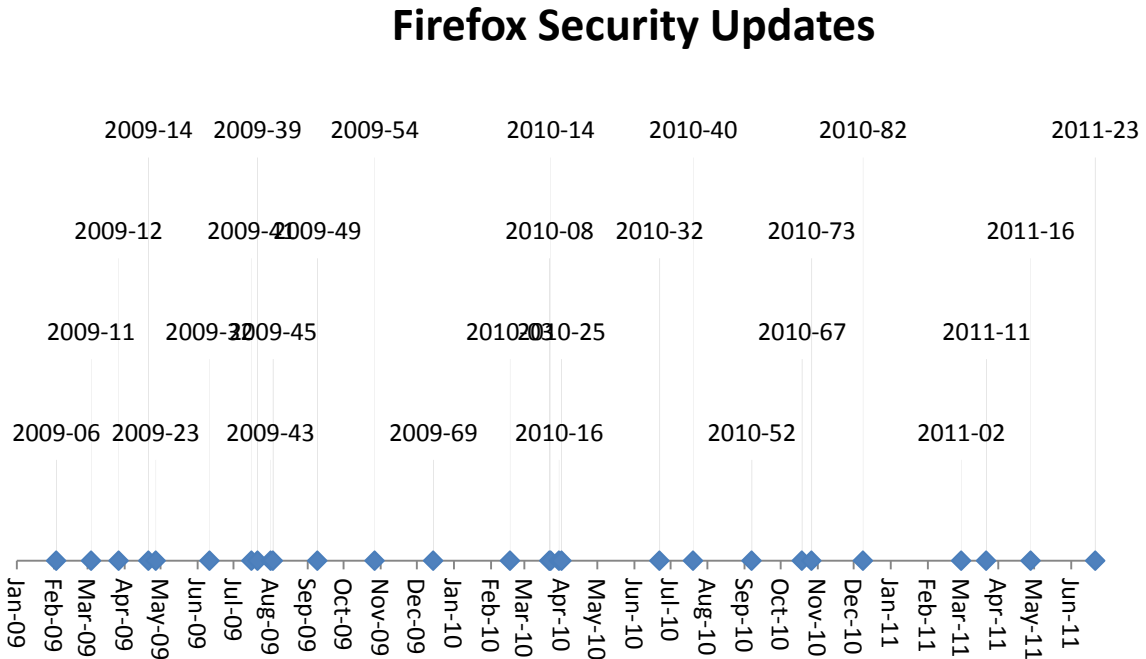


Figure 3. Mozilla Foundation security advisories affecting Firefox over time

The graph in Figure 8 is far less regular than either one of the Microsoft Internet Explorer graphs. This irregularity most likely stems from a fundamentally different approach to development, and a fundamentally different organization structure. However, these differences cannot be used to draw any security relevant conclusions.

Chrome

Google, like Mozilla, does not have a rigid update release schedule. Based on the data in Figure 9, Chrome tends to release updates more frequently than both Mozilla and Microsoft. Note that this data does not include Flash-only or non-security updates.

Chrome Security Updates

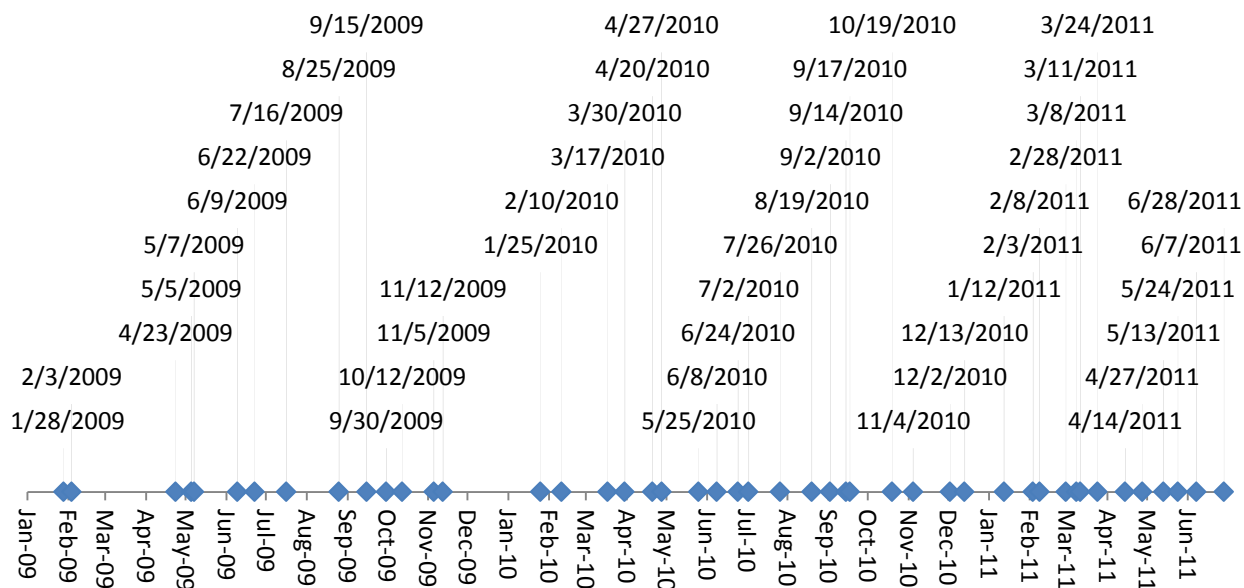


Figure 4. Chrome Security Update over time

The graph in Figure 9 appears more regular than Firefox but less regular than Internet Explorer's update graphs. The similarities with Firefox might stem from a more similar approach to development and a more similar corporate structure when compared to Microsoft. The increased regularity when compared to Firefox's update release may be due to differences in quality assurance testing. However, no security conclusions can be drawn from any of these graphs.

Reflections

Over the past 54 months, many updates have been for released for each browser. Chrome has conducted 47 update events. Mozilla has conducted 29, although the number of individual advisories reached 178. Microsoft has only conducted 27 update events, with 62 individual bulletins, due to their more rigid update release cycle.

While each vendor has different practices and procedures, all of them are roughly comparable. Chrome clearly stands out as being the most frequently updated of the three; based strictly on the number of update events, regularity of updates, and method by which the browser itself updates.

Given all this information, we can conclude that the browsers are different. Development methodologies, corporate structure and patch release infrastructure all play a role in making dissimilar graphs. However, none of these pieces of information can be used to draw a security related conclusion.

Publicly Known Vulnerabilities

Vulnerabilities within web browsers have become an increasingly common way for an attacker to compromise an end user's system. It seems intuitive that a larger number of patched vulnerabilities

imply that a particular browser is less secure; however, this is not the case. The reason is that the number of patched vulnerabilities does not indicate the number of vulnerabilities within a given code base. As an example, consider the following chart:

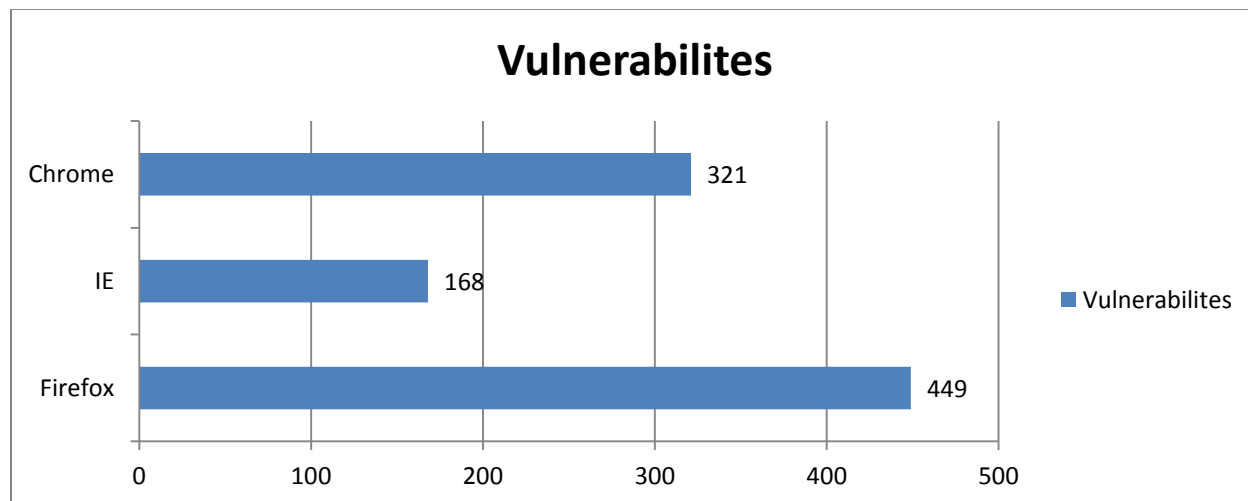


Figure 5. Total vulnerability counts for each browser

The chart depicts the total number of vulnerabilities patched within the period of the dataset. A naïve interpretation would be that Firefox is the least secure, Chrome is in the middle and Internet Explorer is the most secure. However, what this could indicate is that Firefox has the most vulnerabilities because researchers have an easy time exploiting the vulnerabilities and thus pay more attention to Firefox. Chrome may have the second most because they offer a bounty program so researchers pay more attention. Internet Explorer may have the least because they require more quality assurance overhead before creating a patch. The point is, any conclusion drawn from the data is speculation and the data does not aid in discovering which browser is most secure.

Vulnerabilities by Severity

Another way to look at the data is to look at the number of vulnerabilities in each browser broken down by severity. This breakdown seems attractive because if one browser has more highly critical vulnerabilities compared to the others, then it would appear to be less secure. However, another argument would be that a browser with more highly critical vulnerabilities disclosed puts an emphasis on fixing these vulnerabilities as soon as possible. In rebuttal, the browser with the most high severity vulnerabilities may have a bad architecture that contributes to more severe vulnerabilities. The truth of the matter is far more complex, and these uncertainties are better documented in the [Historical Vulnerability Statistics](#) section of this paper.

As a concrete example of these issues, consider the following chart:

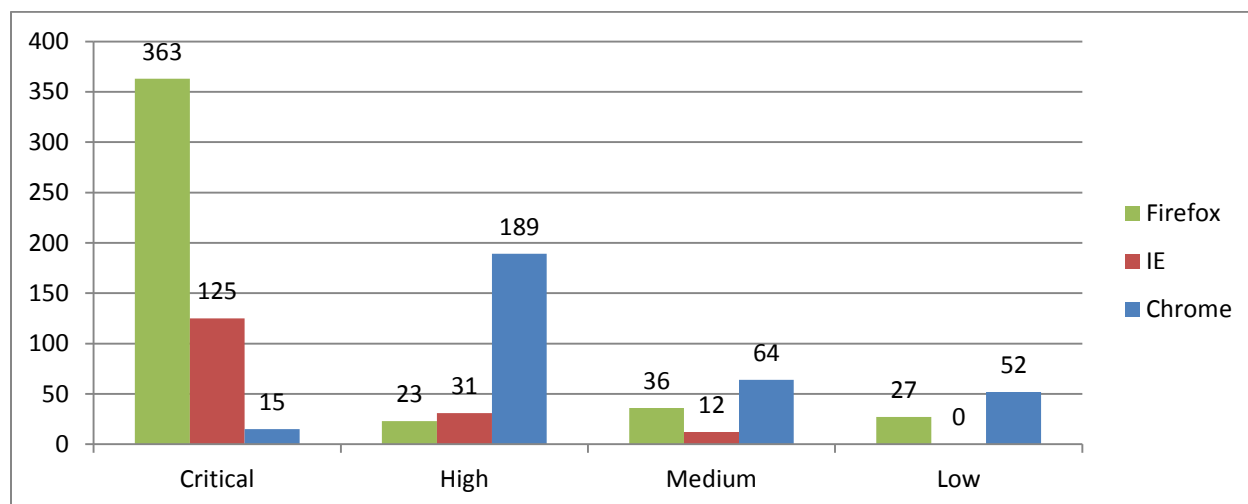


Figure 6. Vulnerabilities by severity for each browser

The differences between browsers are quite dramatic. Firefox, Internet Explorer and Chrome all appear to have a very different severity profile. A naïve determination might be that Firefox has the worst security, Internet Explorer is in the middle and Chrome has the best security. However, since risk ratings are designed to convey urgency for the end user to patch, the only real conclusion that can be drawn is that Mozilla applies a higher risk rating to convey their message and Google feels comfortable rating their vulnerabilities with a lesser severity. Any conclusions drawn from this type of data regarding the inherent security posture of the code base are ill founded.

Time to Patch

The amount of time it takes for a vendor to go from vulnerability awareness to a fix can be seen as a security commitment indicator. However, the reality is not so simple. Internet Explorer has such a deep integration with the Windows operating system that a change in Internet Explorer can have repercussions throughout a much larger code base. In short, the average time to patch is less indicative of a commitment to patch, as it is of complications with providing a good patch.

In Figure 12 below, it is clear that Microsoft's average time to patch is the slowest. To be fair, this information was based on a much smaller sample set than Firefox and Chrome. Even worse, it may be possible that the advisories for these vulnerabilities had timeline information only because of the fact that they had taken so long to patch.

Firefox comes in second, taking an average of 50 days less than Microsoft to issue a patch. The browser with the fastest average time to patch is Chrome. With an average of 53 days to patch vulnerabilities, they are nearly three times faster than Firefox and slightly more than four times faster than Microsoft.

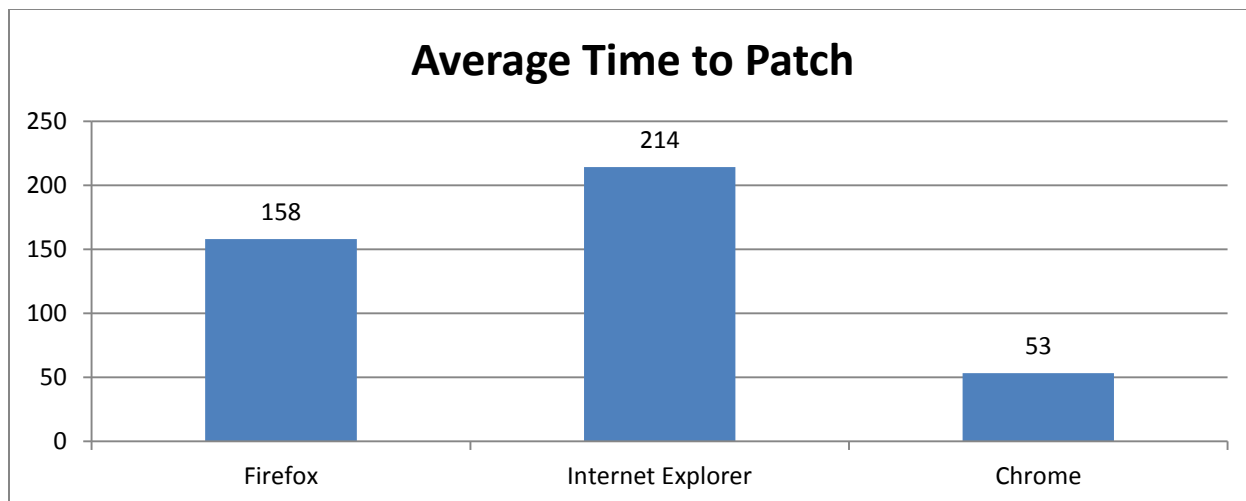


Figure 7. Average time to patch for all three browsers

Time to patch is not a good indicator of a browser’s susceptibility to compromise. Some vendors may prioritize patching efforts to address high impact vulnerabilities quickly, while neglecting less severe vulnerabilities. Some vendors may address “easy fix” vulnerabilities quickly and neglect more severe vulnerabilities. Additionally, the only metric that can be tracked is the date a vendor was made aware of a vulnerability or the date it was detected in the wild, which neglects 0-day vulnerabilities and skews the metric for vulnerabilities that took time to detect in the wild. Finally, the quality of the data that could be collected is great for Chrome, good for Firefox and terrible for Internet Explorer. Since these issues cannot be corrected, making strong security comparisons between browsers on that basis is not feasible.

What it does show is the respective vendor’s efficiencies in their response processes for vulnerabilities that we can track. Google’s update mantra for Chrome is “Release Early, Release Often” and this is reflected within their lower average time to patch. Firefox is slightly less efficient at delivering updates to end users, and according to the data, Internet Explorer is the least efficient. However, both Firefox and Internet Explorer’s code bases are more heavily integrated with other products. Therefore, the additional overhead may be due to coordination of releases and additional QA to ensure stable patches.

URL Blacklist Services

The stated intent of URL blacklisting services is to protect a user from him or herself. When a link is clicked inadvertently, via a phishing email or other un-trusted source, the browser warns the user “are you sure?” and displays a warning that the site might be unsafe based on a list of unsafe URLs regularly updated as new malware sites go live and are taken offline. Microsoft’s URL Reporting Service (from here forward, “URS”), formerly “Phishing Filter”, referred to in the browser application as “SmartScreen Filter”, was the first to provide this feature, with Google’s Safe Browsing List (“SBL”) following suit later, utilized initially by Mozilla Firefox, and now by Chrome as well as Safari.

Both services utilize functionally similar approaches, storing a local copy of hashed URLs in the blacklist, and sending the hash value of a URL to a public web service for validation if it doesn’t exist in the local table. Google’s API is publically documented and accessible to anyone who wishes to develop a client within terms-of-use constraints, while Microsoft’s is proprietary and specific to the Internet Explorer browser only.

Comparing Blacklists

URL blacklisting is another area where metrics are challenging, not in that the metrics are difficult to generate, but in that in our analysis, neither Google’s Safe Browsing service nor Microsoft’s URS appears to provide a fully comprehensive snapshot of all malware in the wild at any given point in time. Other blacklist and early-warning services, such as those used for botnet detection or spam prevention, also differ greatly in content, so this isn’t entirely unexpected. An apt analogy might be Signals Intelligence in the military. Two monitoring stations tracking enemy communications in two geographic areas both intercept some enemy radio traffic, but neither station picks up every single message, so neither has a complete picture.

“Antivirus-via-HTTP”

Like antivirus, URL blacklists implement a negative security model, or an antipattern-based approach (“that which is not expressly denied is permitted”, as opposed to “that which is not expressly permitted is denied”). This means that URL blacklists do not protect well against customized payloads created for a specific target, or against small-batch propagation to a limited user population.

However, URL blacklists do provide a deterrent against mass deployment of fast-flux malware to large user populations, with the benefit of rapid updates due to the realtime delivery of these services. As with other blacklist services like SMTP Realtime Blackhole Lists, URL blacklists provide one part of a larger set of defensive measures that helps to improve the overall security posture of the browser.

Multi-Browser Defense

Another criterion to consider in the case of URL blacklists is the fact that while MS URS was implemented to protect against threats targeting Internet Explorer, Google’s SBL primarily is in use to defend against attacks targeting the other three major browsers. While multi-browser attacks are increasingly common, attacks specific to Internet Explorer still outnumber those targeting the other three browsers with less market share. While not material to this paper *per se*, it is worth noting that by

definition, the number of URLs blacklisted in Microsoft's URS should be higher, based on the MS URS' stated purpose.

Comparing Blacklist Services

A previous third-party study of blacklist services used an undisclosed set of sample URLs for the generation of browser tests. Samples were from a number of private sources, and results appeared to skew heavily toward Microsoft's URS.

For our purposes, Accuvant used four public sources for active malware URLs: MalwareDomains, MalwarePatrol, BLADE and MalwareBlackList. This approach has the advantage of providing public attribution of sources, de-emphasizing private feeds and undisclosed sources that may favor one blacklist over another. In particular, since Microsoft licenses several private feeds to populate the URS list, Accuvant LABS wanted to ensure that our test dataset did not mirror Microsoft's too closely. Likewise, our analysis didn't make use of Google's internal SBL source material either. Our intent was to replicate a fairly broad sample of malware URLs in the wild, with minimal bias toward either blacklist being evaluated.

Comparison Methodology

Accuvant LABS performed daily downloads of the current blacklists from the malware URL sources above, removed duplicates and utilized browser automation to request each URL with Internet Explorer 9, recording whether the URL was reported unsafe by the MS URS service. Because Chrome and Firefox both utilize the Google SBL, an API client queried the Safe Browsing API during the same period, again recording the results for each page requested.

In the interest of comparing features that are at parity in all three browsers under review, we did not evaluate Microsoft's application reputation component, though it should be noted that Chrome has recently implemented [similar functionality](#) as well.

Testing took place over an eight-day period, from July 23, 2011 through July 30, 2011, with an average of 5960 URLs per day. Of these samples, an average of 3086 per day was live and responding during the test period. Dead hosts were discarded from the sample set as not posing a threat during the testing period.

Results Analysis

Overall, neither service identified a majority of URLs from the diverse sample set. On average, both services identified nearly an identical number of URLs, though the URLs identified differed. Over the course of testing, 42 URLs present in the MS URS were also flagged by Google's SBL, while no SBL URLs were identified at any time that was in the MS URS. This demonstrates that both services use substantially different data sources, and that no one service appears to have a truly comprehensive dataset of all malware present on the web.

Gathering intelligence about malware URLs is generally performed by running honeypots and spam-traps, and harvesting URLs from malware captured in the wild. Since no authoritative source exists, it is likely that each organization gathering data is getting one part of the overall picture. Based on

Accuvant’s analysis, no party is performing this data collection comprehensively. During the course of testing, our test environment was infected numerous times by malware that was not in the database of either URL blacklist service.

The table below lists the daily results of testing, averages, and the number of total URLs versus confirmed-live URLs in the sample set. Overall, both URL blacklists performed roughly the same in terms of number of URLs identified as malware, with minor variances each day.

Date	7/23	7/24	7/25	7/26	7/27	7/28	7/29	7/30	Average
Google SBL Matches	409	411	411	422	393	396	397	404	405
Microsoft URS Matches	361	336	364	371	401	447	499	450	404
Total URLs	5684	5724	5738	6128	6145	6089	6149	6025	5960
Live URLs	2993	2948	3040	3416	3128	3043	3115	3003	3086

Figure 8. URL blacklists over time

The daily detail below shows the gap between the numbers of live URLs provided versus those identified by either service.

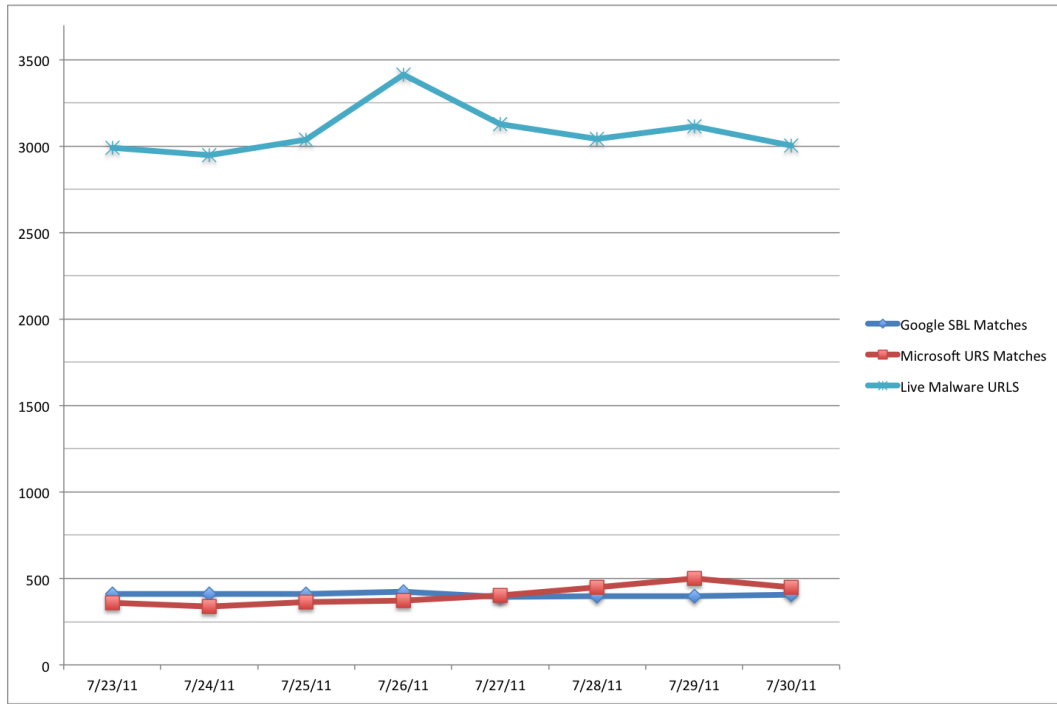


Figure 9. Malware URL vs. sample set

The table below shows the rolling daily averages of the two blacklist services, showing an overall trend toward near parity in the number of URLs identified.

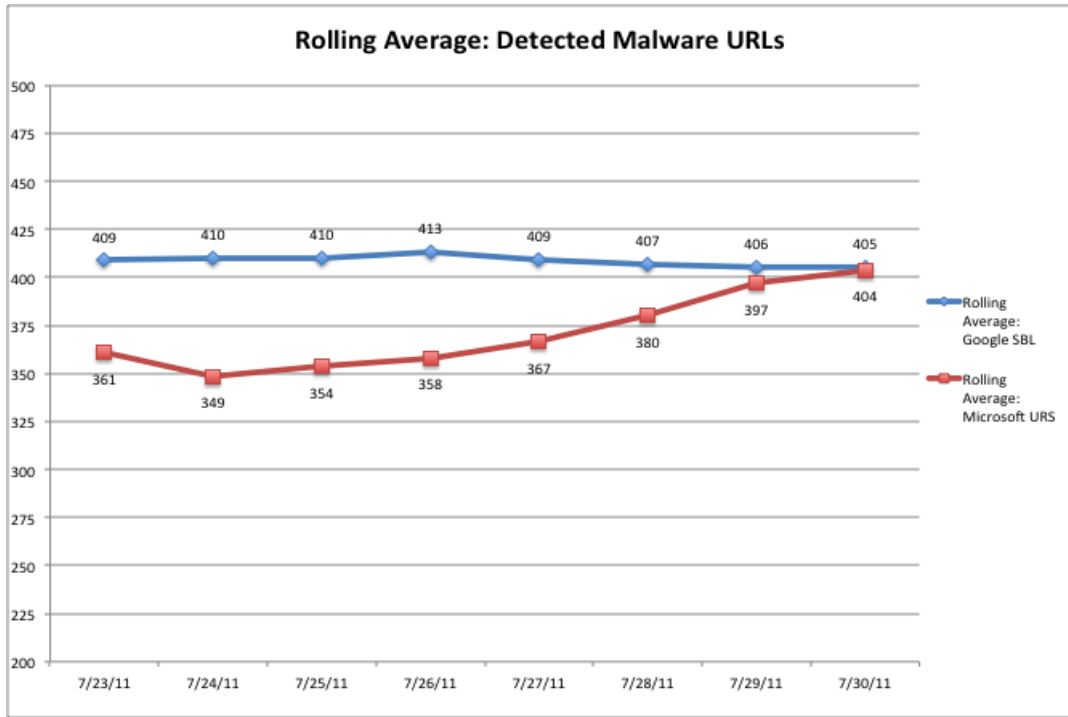


Figure 10. Average detected malware URLs

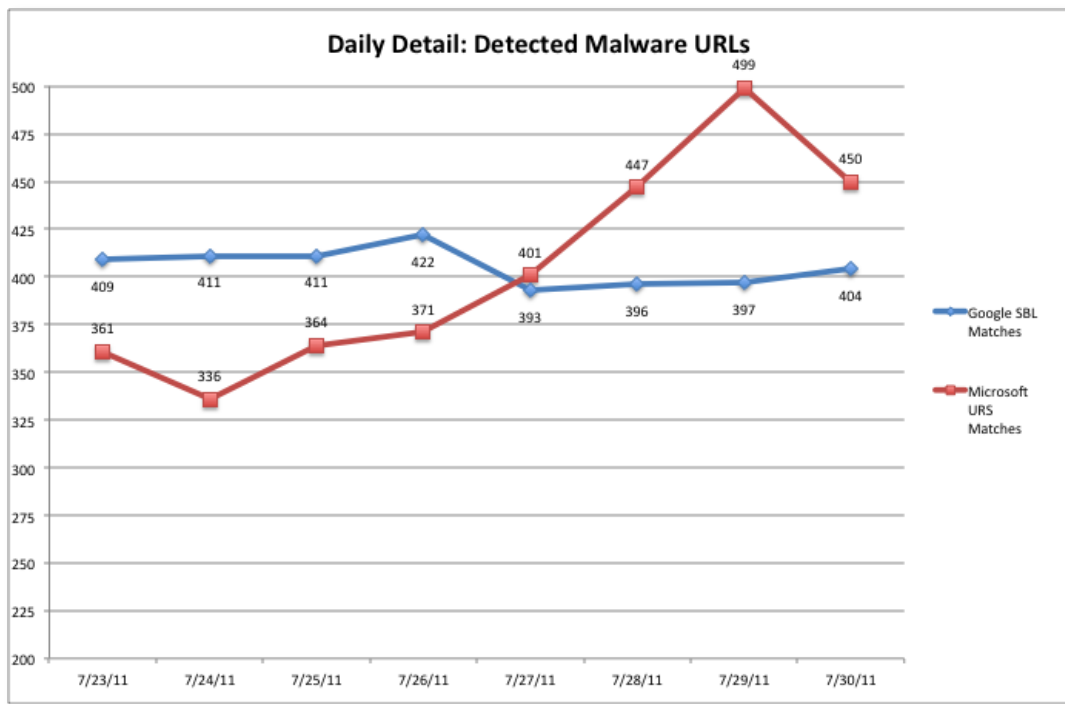


Figure 11. Daily detected malware URLs

In the daily detail view, it's clear that on one day, July 29, a large update was made to the MS URS, possibly due to a specific threat that was identified, or a weekly update. Again, this demonstrates that data sources for both services appear to be quite different. The trend lines seem to indicate that Google's SBL undergoes more incremental updates, whereas the MS SBL may be receiving updates in batches, though a longer sample period (several months or more) would be required to confirm this.

Conclusions

Based on our testing, it seems clear that no URL blacklisting service is fully comprehensive, and that any antipattern-based defensive measure is, by definition, imperfect. As with antivirus, the question is not whether the pattern-based detection will fail, but when and how. As such, blacklisting services should be considered a part of the overall browser defense model, rather than the only perimeter an attacker must traverse.

Other defenses discussed elsewhere in this paper, such as exploit mitigation and other approaches to limiting the extent of the damage from a given payload, are likely a better criteria for browser security than simple pattern matching alone.

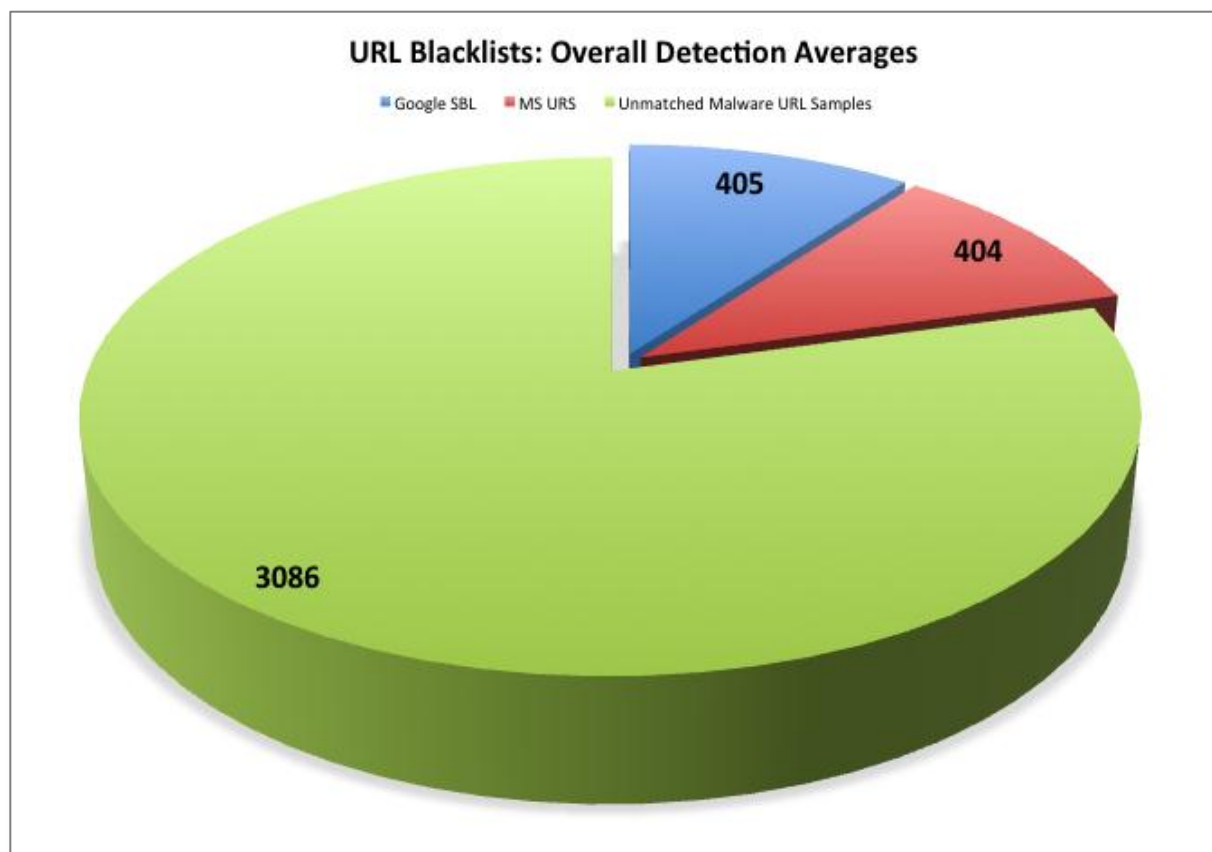


Figure 12. Blacklist overview

Anti-exploitation Technologies

The premise of this paper was to evaluate the overall security of each web browser selected. We achieved this by evaluating security controls independently and formulating a conclusion based on the security controls in place. This section provides information on distinct security controls and their relevance within this paper.

Address Space Layout Randomization (ASLR)

Address Space Layout Randomization (ASLR) attempts to make it harder for attackers to answer the question ‘*where do I go*’. By taking away the assumption of known locations (addresses), the process implementing ASLR makes it much more difficult for an attacker to use well-known addresses as exploitation primitives. One key weakness of ASLR is the ability for one module to ruin it for the rest, a weak link in an overall strong chain. During analysis, each executable used by a browser was evaluated to ascertain its ability to implement proper randomization.

Data Execution Prevention (DEP)

One of the first steps in compromising a system is achieving arbitrary code execution, the ability run code provided by the attacker. During traditional exploitation scenarios, this is achieved by providing the compromised application with *shellcode*, data furnished by the attacker to be run as code. Data Execution Prevention (DEP) addresses the problem of having data run as code directly. DEP establishes rules that state: “Only certain regions of memory in which actual code resides may execute code. Safeguard the other areas by stating that they are non-executable”. Our audit included querying each browser process about its ability to establish a DEP policy at run time.

Stack Cookies (/GS)

Due to common programming errors, archaic APIs and trusted user input, stack-based buffer overflows have been leveraged to gain code execution on Intel-based architectures for over 30 years. Microsoft compilers (all three browsers tested were compiled with Microsoft Visual Studio 2005 or greater) have the ability to put a *stack cookie* on the stack at compile time. This cookie can be validated, certifying the stack variables’ integrity upon returning to the caller. The /GS mechanism can re-order the variables on the stack as an attempt to prevent overflow-able variables from tainting other local variables, avoiding a future change in code execution [Microsoft_GS]. Executables used and installed by each browser were examined for characteristics of being compiled with /GS. Unfortunately, this is a flawed process, due to the nature of /GS.

Note: Although a library may have been compiled with the stack cookie feature, if it has no functions that meet the /GS requirements, then there will be no trace of the compilation feature.

SafeSEH/SEHOP

Other addresses used for code execution, other than the saved return address, became necessary due to the advent of the /GS compilation flag. The next logical candidate was the Structured Exception Handling (SEH) information residing on the stack. These exception handlers could be overwritten to execute data disguised as code at an address of the attacker’s choosing; completely circumventing the security attempts of the stack cookie. SafeSEH was designed to ensure that only the addresses of

validated exception handlers could be executed. Unfortunately, SafeSEH requires full code rebuilds with the SafeSEH compiler option enabled. The limitations of SafeSEH brought on the invention of Structured Exception Handler Overwrite Protection (SEHOP). Instead of validating that an image contained safe exception handlers, the exception handler code changed; validating the entire chain before dispatching an exception [Microsoft_SEHOP]. Because SEHOP was disabled by default on Windows 7 SP1 [Microsoft_SEHOP_KB], no additional testing regarding SEH overwrite exploit mitigation was completed.

Sandboxing

A sandbox is a mechanism of isolating objects/threads/processes from each other in an attempt to control access to various resources on a system. At the time of this writing, Google Chrome and Microsoft Internet Explorer both implement security restrictions that are considered a *sandbox*. The following entries describe the unit tests used to assess sandbox effectiveness. Although not comprehensive, the tests provide good insight into the overall protection provided by each sandbox.

File System

A proper sandbox should attempt to limit certain processes from accessing files and directories that may contain vital system information or used in a context that could result in executing code of the attacker's choosing. We augmented Chrome's file test cases, resulting in full read/write testing of integral Windows files and directories.

Registry

Limiting access to the Windows Registry is integral to maintaining system integrity. By limiting access to the registry, the sandbox can ensure that sensitive information cannot be obtained, altered or added. We chose to test a variety of registry hives with the maximum permissions available.

Network Access

Although file system and registry access may be limited to an attacker, it is still important to ensure that information cannot be leaked via the network. We tested the sandbox's ability to limit outbound network access along with determining if a port could be bound to the current process for listening.

Resource Monitoring

Certain techniques are prevalent within most spyware utilities. Malware authors may need the ability to read portions of the screen (i.e. take screenshots) or log input from the keyboard. We included tests that attempted to read pixels on the current display along with attempts to log keyboard input.

Processes/Threads

While it's necessary for many processes and threads to run concurrently on a system, arbitrary access to them is debatable. A sandboxed process should have very limited access to other processes and threads on a system. Our test cases enumerated the security permissions for every thread and process on a system from the perspective of the sandboxed process.

Handles

Windows keeps track of entire important objects (windows, buttons, files, etc.) for future reference within the system. Each object is tracked via a unique HANDLE. By enumerating all the handles on the

system and validating access permissions, we determined how processes from inside the sandbox communicate with other objects running on the operating system.

Windows Clipboard

The Windows clipboard provides functionality to permit multiple applications to transfer data [Microsoft_Clip]. By limiting the ability to set and receive data via the clipboard, a sandbox can reduce the likelihood that attacker-supplied data will be used in a malicious manner. Our tests evaluated the capabilities of the browser process to use the clipboard functionality.

Windows Desktop

Most people are familiar with the Windows desktop because it is the first thing they see after login; however, desktops also group windows together in the same security context [Chrome_Sandbox]. We tested the functionality to change and create desktops to evaluate process isolation.

System Wide Parameters

Alteration of system wide parameters by an unauthorized user could lead to an undesirable effect on system stability and security. We conducted tests to evaluate security constrictions around getting and setting system wide parameters.

Windows Messages

Windows messages are fundamental to inter-window communication, but unprivileged processes should be limited to where these messages are sent. We put test cases in the harness to determine if broadcast messages could be sent to all other windows (on the same desktop) via the sandboxed process.

Windows Hooks

Windows hooks are used to monitor various types of system events. The hooking functionality adds a hook to the chain in anticipation of performing an action based on standard windows events [Microsoft_SWH]. The same hooking functionality can also be used by malware authors; for example, hooking keyboard actions to monitor user input. Our tests determined if Windows hooks are permitted via SetWindowsHookEx() API.

Named Pipes

Named pipes are one-way or two-way pipes used for client/server communication [Microsoft_Pipes], which can also be used for local Inter Process Communication (IPC). Since named pipes are used for communication, reducing the set of named pipes that the browser can talk to reduces the overall attack surface for a potential attacker. Our test harness assessed some well-known named pipes on Windows 7 (32-bit).

JIT Hardening

JIT engines by necessity emit executable code, often at predictable locations in an application's address space. However, the presence of predictable code can weaken the security of a piece of software by simplifying the process of exploiting vulnerabilities elsewhere in the same address space. Technologies

like ASLR and DEP already exist for compiled binaries, but are not effective protections for JIT engines. As such, different mechanisms would be necessary to achieve a comparable effect.

- JIT code must currently be emitted in-process.
- Scripting engines provide a robust method that exploits often use to prepare the address space in order to be successful.
- JIT compilation bridges the distinction between data and code, which reduces the effectiveness of standard mitigation techniques, such as DEP.

JIT hardening is important because it can reduce the exploitability and impact of vulnerabilities in **other** software within the same address space. As a result, the larger the scope of the process, the more important JIT hardening becomes.

JIT Hardening Techniques

Codebase Alignment Randomization

The code emitted by JIT engines can begin with a random number of **NOP** or **INT 3** instructions to randomize the alignment of the instructions within. This prevents the prediction of specific instructions within emitted code.

Emitted Instructions	Hex Encoding
nop	90
nop	90
nop	90
push ebp	55
mov ebp, esp	8BEC
push esi	56

Figure 13. Example of codebase alignment randomization

Instruction Alignment Randomization

Even if the codebase offset is randomized, the internal alignment of basic blocks may allow for the accurate prediction of instructions. To prevent this, NOP instructions can be randomly inserted during compilation to randomize the alignment of subsequent instructions.

Constant Blinding

User controllable values can be obfuscated by XOR encoding the constant values with a random cookie during compilation and emitting two instructions that will de-obfuscate the value at runtime. This prevents constant values from being present in executable memory, therefore cannot be used to seed code that could be used during a later stage of an exploit.

Value	Emitted Instructions	Resulting Hex Code
0x02222222	mov eax, 89EF3D74 xor eax, 8BCD1F56	b8743def89 35561fcd8b

0x22222222	mov eax, A9EF3D74 xor eax, 8BCD1F56	b8743defa9 35561fcd8b
0x12345678	mov eax, 99F9492E xor eax, 8BCD1F56	b82E49f999 35561fcd8b

Figure 14. Example of constant binding

Constant Folding

The possible values that can be emitted as instructions are limited by instead emitting the folded value. The result is that only even constant values will appear as instruction operands.

Script	Emitted Instruction
x = 1;	mov eax,00000002
x = 0x1111;	mov eax,00002222

Figure 15. Example of constant folding

Memory Page Protection

If the code emitted by a JIT engine is not modified after the initial compilation, it will only require the PAGE_EXECUTE memory protection. This will result in a crash if targeted by a memory leak or memory corruption. If the JIT engine requires that the code be updated dynamically, the page protection can be temporarily changed to PAGE_EXECUTE_READWRITE for the modification. PAGE_EXECUTE_READWRITE is the least secure memory protection.

Resource Constraints

A constraint can be placed on the total executable allocations allowed by the JIT engine. The total size of compiled code is often very small. The source is likely malicious if large amounts of code are being emitted. Placing a constraint on the total executable memory prevents the bypass of ASLR and DEP through address space exhaustion.

Additional Randomization

The JIT engine can attempt to specify a random address at which to allocate executable memory manually instead of using the default OS behavior. ASLR does randomize the base address to not be completely predictable, but the significance of this decreases for many allocations where multiple large allocations will often result in a contiguous block of memory which then becomes predictable. Additional randomization can prevent the spraying of large amounts of code at predictable addresses.

Guard Pages

If the memory page protections must be PAGE_EXECUTE_READWRITE, guard pages can be placed before each region of executable memory to protect against memory corruption from crossing page boundaries.

Browser Anti-Exploitation Analysis

Each of the browsers selected for the study were put through rigorous tests, including but not limited to, statistical vulnerability analysis, plug-in architecture review, malware prevention analysis and simulated sandbox review. These tests attempt to give an accurate representation of the browser's overall security, not that of a singular, narrow scope. Although not all possible permutations could be achieved, a representative number of tests were performed to give the readers of this paper a view into the holistic security of each browser.

An additional note, the sandbox testing was performed by modifying the *sandbox* project that resides in the Google Chrome source tree. By augmenting tests and logic to the Chrome sandbox testing harness, we were able to easily integrate sandbox measurement code into the current architecture. Also, by compartmentalizing the test harness into a single module (DLL), it can be used by other third party testing utilities if desired.

By overwriting and adding the following files to the *sandbox_poc* project in the Google Chrome source tree, one will be able to reproduce our results; through the creation of the *pocdll.dll* library:

- **pocdll.cc**
 - This original library was altered to add additional measurements to the test harness. The exported *Run(logfile)* function can be called after opening a log file of the assessors choosing.
- **cv.cc**
 - Code that contains Accuvant specific test material to be used by *pocdll.dll*.
- **processes_and_threads**
 - Code that contains process and thread tests. A test for 'CreateProcess()' was added.

Browser Comparison

Sandbox Result	Chrome	Internet Explorer	Firefox
Read Files	✓	✗	✗
Write Files	✓	●	●
Read Registry Keys	✓	●	●
Write Registry Keys	✓	✓	●
Network Access	✓	✗	✗
Resource Monitoring	✓	●	✗
Thread Access	✓	●	●
Process Access	✓	●	●
Process Creation	✓	✗	✗
Clipboard Access	✓	✓	✗
System Parameters	●	●	✗
Broadcast Messages	✓	✗	✗
Desktop & Windows Station Access	✓	✗	✗
Windows Hooks	✗*	✗	✗
Named Pipes Access	✓	●	✗



Action was blocked



Action was partially blocked



Action was allowed

*Isolated Desktop and Window Station

Figure 16. Sandbox overview

JIT Hardening Techniques	Chrome	Internet Explorer	Firefox
Codebase Alignment Randomization	✗	✓	✗
Instruction Alignment Randomization	✗	✓	✗
Constant Folding	✓	✓	✗
Constant Blinding	✓	✓	✗
Resource Constraints	✓	✓	✗
Memory Page Protection	✗	✓	✗
Additional Randomization	✓	●	✗
Guard Pages	✓ *	●	✗

- ✓ Technique was implemented
- Technique was not necessary
- ✗ Technique was not implemented

* Chrome 14

Figure 17. JIT hardening overview

Although there was a plethora of tests performed on all the browsers, a general conclusion about each browser can be derived from the figure above. Google Chrome prevents processes in the sandbox from doing much of anything, and even if permission is granted, it is limited to the alternate desktop. Microsoft Internet Explorer generally allows read access to most objects on the operating system, while only preventing a hand full of system modification changes. Mozilla Firefox, on the other hand, is only limited by the medium integrity under which it runs; permitting read, write and system change capabilities associated with regular, non-administrator users.

Google Chrome

ASLR Results

Accuvant examined each binary installed or loaded during browser startup to determine its ASLR compatibility. The *pefile* python library was used to check the **OPTIONAL_HEADER.DllCharacteristics** attribute to determine if a given module's address space would be randomized by the loader.

All the binaries that were currently loaded and being used in the browser were ASLR compatible, leaving only one installation file (*GoogleUpdater.exe*) incompatible with ASLR. For a full listing, please see the [Google Chrome ASLR Results](#) in [Appendix A](#).

Note: We are aware that the list in Appendix A may be missing binaries and will attempt to update it if new modules are discovered. Also note that this omits any third party/plugin modules.

DEP Results

As mentioned previously, Data Execution Prevention (DEP) prevents attackers from executing their *data* as *code*. By limiting execution rights to certain address spaces, DEP greatly reduces the attack surface. The default DEP policy for Windows 7 (32-bit) is **OptIn** [Microsoft_DEP]; meaning that the module will either have to be compiled with the /NXCOMPAT flag set or DEP will need to be enabled via **NtSetInformationProcess()** [Uninformed_DEP] (Windows XP & Windows 2003) or **SetProcessDEPPolicy()** [Microsoft_SPDEP] (Windows Vista and later).

Code: Please see *dep.cc* in the Google Chrome project.

```
// Flags being used as per SetProcessDEPPolicy on Vista SP1.
ULONG dep_flags;
switch (enforcement) {
case DEP_DISABLED:
    // 2
    dep_flags = MEM_EXECUTE_OPTION_DISABLE;
    break;
case DEP_ENABLED:
    // 9
    dep_flags = MEM_EXECUTE_OPTION_PERMANENT | MEM_EXECUTE_OPTION_ENABLE;
    break;
case DEP_ENABLED_ATL7_COMPAT:
    // 0xD
    dep_flags = MEM_EXECUTE_OPTION_PERMANENT | MEM_EXECUTE_OPTION_ENABLE |
                MEM_EXECUTE_OPTION_ATL7_THUNK_EMULATION;

    break;
default:
    NOTREACHED();
    return false;
}

HRESULT status = NtSetInformationProc(GetCurrentProcess(),
                                     ProcessExecuteFlags,
                                     &dep_flags,
                                     sizeof(dep_flags));
```

Figure 18. Chrome DEP being enabled

Process Examination: Process Explorer shows DEP enabled for the browser and rendering processes.

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name	DEP	ASLR	Integrity
vmtoolsd.exe	1408		6,504 K	3,924 K	VMware Tools Core Service	VMware, Inc.	n/a		
VMUpgradeHelper.exe	1500		2,232 K	588 K	VMware virtual hardware up...	VMware, Inc.	n/a		
TPAutoConnSvc.exe	1796		1,928 K	2,092 K	TPAutoConnect Printer Creat...	ThinPrint AG	n/a		
TPAutoConnect.exe	292		2,080 K	2,708 K	TPAutoConnect User Agent	ThinPrint AG			Medium
taskhost.exe	276		6,996 K	2,880 K	Host Process for Windows T...	Microsoft Corporation	DEP (permanent)	ASLR	Medium
SearchIndexer.exe	192		37,000 K	11,796 K	Microsoft Windows Search L...	Microsoft Corporation	n/a	ASLR	
SearchProtocolHost.e...	3736		1,480 K	4,904 K	Microsoft Windows Search P...	Microsoft Corporation	DEP (permanent)	ASLR	Medium
SearchFilterHost.exe	2312		952 K	3,420 K			n/a		
svchost.exe	784		3,764 K	920 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	3320		75,668 K	19,484 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
taskhost.exe	3208		6,044 K	2,700 K			n/a		
taskhost.exe	4032		3,328 K	8,008 K			n/a		
lsass.exe	512		2,776 K	3,064 K	Local Security Authority Proc...	Microsoft Corporation	n/a	ASLR	
sm.exe	520		1,300 K	1,084 K			n/a		
cars.exe	404		9,732 K	2,000 K			n/a		
conhost.exe	392		540 K	204 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
winlogon.exe	440		1,484 K	552 K			n/a		
explorer.exe	708		37,008 K	33,532 K	Windows Explorer	Microsoft Corporation	DEP (permanent)	ASLR	Medium
VMwareTray.exe	1200		2,244 K	1,756 K	VMware Tools tray application	VMware, Inc.			Medium
VMwareUser.exe	1752		8,588 K	8,620 K	VMware Tools Service	VMware, Inc.			Medium
procexp.exe	2340		11,600 K	7,784 K	Sysinternals Process Explorer	Sysinternals - www.sysinter...	DEP (permanent)	ASLR	Medium
chrome.exe	3580		12,924 K	29,340 K	Google Chrome	Google Inc.	DEP (permanent)	ASLR	Medium
chrome.exe	2632		14,188 K	23,620 K	Google Chrome	Google Inc.	DEP (permanent)	ASLR	Low

CPU Usage: 1.56% | Commit Charge: 29.10% | Processes: 39 | Physical Usage: 34.99%

Figure 19. Chrome permanent DEP enabled

GS Results (Stack Cookies)

A test was run to determine if a module was compiled with the /GS compiler option for each item installed by Google Chrome. Although flawed*, this simple test attempted to see if a stack cookie named *object* was referenced within a binary. While having stack cookies doesn't completely prevent exploitation, it does make writing an exploit more difficult.

While the majority of the modules used by Google Chrome are compiled with stack cookies, our IDA script to detect /GS presence could have false-negatives due to a lack of debugging symbols for certain libraries or modules that contain no code. We recognize that currently, this is a flawed process and will develop a new /GS checking process in the future.

For a full listing of the stack cookie results for Google Chrome please reference the [Google Chrome GS Results](#) section in [Appendix A](#).

***Note:** A module may not have the need for a stack cookie, which is determined by stack variable usage [Microsoft_GS].

JavaScript JIT Hardening

Constant Blinding

During compilation, each constant value is XOR encoded with a random cookie that is securely generated per code chunk. Only values greater than 0x7FFF are obfuscated this way.

Script	Emitted Instruction	Hex Value
x = 0x00000001;	mov eax,2	B802000000
x = 0x00000002;	mov eax,4	B804000000
x = 0x00000003;	mov eax,6	B806000000
x = 0x0000007F;	mov eax,0FEh	B8FE000000
x = 0x0000007FF;	mov eax,0FFEh	B8FE0F0000
x = 0x00007FFF;	mov eax,0FFFEh	B8FEFF0000
x = 0x00008000;	mov eax, 82C82646h xor eax, 82C92646h	B84626C882 354626C982
x = 0x0000FFFF;	mov eax, 82C8D9B8h xor eax, 82C92646h	B8B8D9C882 354626C982
x = 0x00011111;	mov eax, 82CB0464h xor eax, 82C92646h	B86404CB82 354626C982
x = 0x11111111;	mov eax, A0EB0464h xor eax, 82C92646h	B86404EBA0 354626C982

Figure 20. Chrome V8 constant binding

Constant Folding

Instruction operands will only contain even values.

Emitted Value = constant value << 1;

Script	Emitted Instruction	Hex Value
x = 0x00000001;	mov eax,3	B802000000
x = 0x00000002;	mov eax,5	B804000000
x = 0x00000003;	mov eax,7	B806000000
x = 0x0000007F;	mov eax,0FEh	B8FE000000
x = 0x0000007FF;	mov eax,0FFEh	B8FE0F0000
x = 0x00007FFF;	mov eax,0FFFEh	B8FEFF0000

Figure 21. Chrome V8 constant folding

When combined with Constant Blinding results, only even values of less than 0x8000 can be emitted as instruction operands.

Resource Constraints

V8 allows the application to manage the resource constraints as required. Chrome allows V8 to allocate a maximum of about 540 MB per process, of which 128 MB may be executable.

```
ResourceConstraints::ResourceConstraints()
: max_young_space_size_(0),
  max_old_space_size_(0),
  max_executable_size_(0),
  stack_limit_(NULL) { }

bool SetResourceConstraints(ResourceConstraints* constraints) {
  i::Isolate* isolate = EnterIsolateIfNeeded();

  int young_space_size = constraints->max_young_space_size();
  int old_gen_size = constraints->max_old_space_size();
  int max_executable_size = constraints->max_executable_size();
  if (young_space_size != 0 || old_gen_size != 0 || max_executable_size != 0) {
    // After initialization it's too late to change Heap constraints.
    ASSERT(!isolate->IsInitialized());
    bool result = isolate->heap()->ConfigureHeap(young_space_size / 2,
                                                old_gen_size,
                                                max_executable_size);

    if (!result) return false;
  }
  if (constraints->stack_limit() != NULL) {
    uintptr_t limit = reinterpret_cast<uintptr_t>(constraints->stack_limit());
    isolate->stack_guard()->SetStackLimit(limit);
  }
  return true;
}
```

Figure 22. api.cc v8.googlecode.com

Guard Pages (Introduced in Chrome 14)

Guard pages are used to protect against memory corruption across region boundaries.

04E80000	Private	1,472 K				1 Read/Write
0EF00000	Private	128 K	128 K	60 K	60 K	2 Execute/Read/Write
DEF00000	Private	8 K	8 K			Read/Guard
DEF02000	Private	120 K	120 K	60 K	60 K	Execute/Read/Write
151C0000	Private	52 K	52 K	44 K	44 K	2 Execute/Read/Write
151C0000	Private	8 K	8 K			Read/Guard
151C2000	Private	44 K	44 K	44 K	44 K	Execute/Read/Write

Figure 23. Chrome VMMap showing V8 guard pages

Additional Randomization

V8 attempts to randomize the address of executable memory manually before using the default OS behavior, which results in significantly less predictable regions of executable memory.

```

void* OS::Allocate(const size_t requested,
                  size_t* allocated,
                  bool is_executable) {
    // The address range used to randomize RWX allocations in OS::Allocate
    // Try not to map pages into the default range that windows loads DLLs
    // Use a multiple of 64k to prevent committing unused memory.
    // Note: This does not guarantee RWX regions will be within the
    // range kAllocationRandomAddressMin to kAllocationRandomAddressMax
#ifdef V8_HOST_ARCH_64_BIT
    static const intptr_t kAllocationRandomAddressMin = 0x0000000080000000;
    static const intptr_t kAllocationRandomAddressMax = 0x0000003FFFFFF0000;
#else
    static const intptr_t kAllocationRandomAddressMin = 0x04000000;
    static const intptr_t kAllocationRandomAddressMax = 0x3FFF0000;
#endif

    // VirtualAlloc rounds allocated size to page size automatically.
    size_t msize = RoundUp(requested, static_cast<int>(GetPageSize()));
    intptr_t address = 0;

    // Windows XP SP2 allows Data Execution Prevention (DEP).
    int prot = is_executable ? PAGE_EXECUTE_READWRITE : PAGE_READWRITE;

    // For executable pages try and randomize the allocation address
    if (prot == PAGE_EXECUTE_READWRITE &&
        msize >= static_cast<size_t>(Page::kPageSize)) {
        address = (V8::RandomPrivate(Isolate::Current()) << kPageSizeBits)
            | kAllocationRandomAddressMin;
        address &= kAllocationRandomAddressMax;
    }

    LPVOID mbase = VirtualAlloc(reinterpret_cast<void*>(address),
                               msize,
                               MEM_COMMIT | MEM_RESERVE,
                               prot);
    if (mbase == NULL && address != 0)
        mbase = VirtualAlloc(NULL, msize, MEM_COMMIT | MEM_RESERVE, prot);

    if (mbase == NULL) {
        LOG(ISOLATE, StringEvent("OS::Allocate", "VirtualAlloc failed"));
        return NULL;
    }

    ASSERT(IsAligned(reinterpret_cast<size_t>(mbase), OS::AllocateAlignment()));

    *allocated = msize;
    UpdateAllocatedSpaceLimits(mbase, static_cast<int>(msize));
    return mbase;
}
  
```

Figure 24. Chrome platform-win32.cc (v8.googlecode.com)

Sandbox Results

All sandbox testing was performed from inside the lowest privileged process (rendering/render process) by attempting to open new resources. Therefore, the results only reflect a rendering process attempting to access resources not previously opened.

File System

Testing attempted to access certain system directories and files via the sandboxed browser process (sometimes referred to as the *renderer/rendering process*).

Note: Permissions may have overlapped due to generic and specific permission checks. This is done to give a general overview accompanied by precise security permissions.

Permission	%SystemDrive% %SystemRoot% %ProgramFiles% %AllUsersProfile% %UserProfile% %Temp% %SystemRoot%\System32 %AppData%
ZERO	BLOCKED
GENERIC_READ	BLOCKED
GENERIC_WRITE	BLOCKED
FILE_ADD_FILE	BLOCKED
FILE_ADD_SUBDIRECTORY	BLOCKED
FILE_APPEND_DATA	BLOCKED
FILE_CREATE_PIPE_INSTANCE	BLOCKED
FILE_DELETE_CHILD	BLOCKED
FILE_LIST_DIRECTORY	BLOCKED
FILE_READ_ATTRIBUTES	BLOCKED
FILE_READ_DATA	BLOCKED
FILE_READ_EA	BLOCKED
FILE_TRAVERSE	BLOCKED
FILE_WRITE_ATTRIBUTES	BLOCKED
FILE_WRITE_DATA	BLOCKED
FILE_WRITE_EA	BLOCKED
WRITE_DAC	BLOCKED

Figure 25. Chrome directory permissions

Files

Permission	%SystemRoot%\explorer.exe %SystemRoot%\Cursors\arrow_i.cur
ZERO	BLOCKED
GENERIC_READ	BLOCKED
GENERIC_WRITE	BLOCKED
GENERIC_EXECUTE	BLOCKED
FILE_EXECUTE	BLOCKED
FILE_READ_ATTRIBUTES	BLOCKED
STANDARD_RIGHTS_EXECUTE	BLOCKED
SYNCHRONIZE	BLOCKED
FILE_READ_DATA	BLOCKED
FILE_READ_EA	BLOCKED
STANDARD_RIGHTS_READ	BLOCKED
FILE_APPEND_DATA	BLOCKED
FILE_WRITE_ATTRIBUTES	BLOCKED
FILE_WRITE_DATA	BLOCKED
FILE_WRITE_EA	BLOCKED
STANDARD_RIGHTS_WRITE	BLOCKED
WRITE_DAC	BLOCKED

Figure 26. Chrome file permissions

Registry

A select few registry hives and keys were accessed from inside the sandbox. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Note: The *MAXIMUM_ALLOWED* permission by the rendering process is read-only.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	BLOCKED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	BLOCKED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	BLOCKED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	BLOCKED

Figure 27. Chrome registry permissions

Network Access

The ability for a browser to access the Internet is vital to its operation, but creation of network sockets for reading, writing and listening could permit an attacker to communicate readable information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	BLOCKED
Send ()	N/A
Recv ()	N/A
Listen ()	N/A

Figure 28. Chrome network accessibility

Note: If WSAStartup() fails, then none of the other tests are applicable due to the inability to start up network sockets. Otherwise, reading/writing and listening (port 88) are attempted.

Resource Monitoring

Recording keystrokes, registering hotkeys and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in their attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire user information (obviously, there are various other techniques.)

Action	Result
GetPixel ()	BLOCKED
RegisterHotKey ()	BLOCKED
GetAsyncKeyState ()	BLOCKED

Figure 29. Chrome Resource monitoring

Threads

Access to other threads running on the system could be used to escalate privileges or trampoline onto different parts of the system. The sandbox test harness provides functionality that tests the access privileges of every thread currently alive on the system. Since this list can vary too greatly, only a list of threads and the privileges granted will be supplied. Please see 'chrome_sandbox_results.txt' in the attachment for a full listing. In addition, the threads used in the sandbox testing were **omitted** from the results.

Process:Thread ID	Permission Granted
N/A	N/A

Figure 30. Chrome threads permission granted

Note: There are no entries because access was denied for all threads running.

Processes

Processes, like threads, can be used to escalate privileges or trampoline onto different parts of the system. The harness provides functionality to test the access privileges of every process currently alive on the system. Only the processes granted certain access have been listed. Please see 'chrome_sandbox_results.txt' in the attachment for a full listing. Also, the processes used in the sandbox testing were **omitted** from the results.

Process	Permission Granted
N/A	N/A

Figure 31. Chrome processes permission granted

Note: There are no entries because access was denied for all processes running.

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization and privilege level as the compromised application. This could permit a plethora of other opportunities that could be used for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with "C:\Program Files\Internet Explorer\iexplore.exe" was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	BLOCKED

Figure 32. Chrome CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them to gain privileges.

The number of handles on a system may vary, but note that the typical Desktop (\Default) and WindowStation (\WinSta0) are not present due to the Chrome sandbox. For more granular information on handling test cases, please see 'chrome_sandbox_results.txt' in the attachment.

Windows Clipboard

The Windows Clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to *GET* and *SET* information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	BLOCKED
SetClipboardData (CF_TEXT)	BLOCKED

Figure 33. Chrome Clipboard Access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	BLOCKED
OpenWindowsStation ("winsta0")	BLOCKED
OpenDesktop ("Default")	ERROR_FILE_NOT_FOUND [due to failed OpenWindowsStation()]

Figure 34. Chrome Desktop/WindowStation access

System Parameters

It should be obvious that an attacker could use system wide parameters to his advantage. These can control screen saver parameters, menu parameters and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	BLOCKED

Figure 35. Chrome SystemParametersInfo()

Note: Only a single system parameter was checked for brevity's sake.

Windows Message Broadcasts

By sending a Windows message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could interpret the broadcast message differently, due to expecting a varying number of parameters [MSDN_Broad]. This could cause operating system instability and exploitation scenarios. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_TIMER)	BLOCKED

Figure 36. Chrome send broadcast message

Windows Hooks

Windows Hooks are a procedure used to monitor certain types of system events on the same desktop as the calling thread [Microsoft_Hooks]. These same hooks have historically been used by malware to do such things as monitor keyboard input and other nefarious tasks. We checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 37. Chrome set Windows hooks

Note: Since Google Chrome creates an alternate desktop, the hooking mechanisms are limited to those threads on the alternate-desktop (i.e. renderer/sandbox desktop).

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows operating system [Microsoft_Pipes]. While the ability to communicate between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We attempted to enumerate all the named pipes for a system for permissions testing. If that were not possible, we would iterate through a list of ‘well-known’ pipes for the Windows 7 (32-bit) operating system in an attempt to validate permissions.

Named Pipe	PIPE_ACCESS_INBOUND PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	BLOCKED
\\.\pipe\ntsvcs	BLOCKED
\\.\pipe\scerpc	BLOCKED
\\.\pipe\protected_storage	BLOCKED
\\.\pipe\plugplay	BLOCKED
\\.\pipe\epmapper	BLOCKED
\\.\pipe\eventlog	BLOCKED
\\.\pipe\atsvc	BLOCKED
\\.\pipe\wkssvc	BLOCKED
\\.\pipe\keysvc	BLOCKED
\\.\pipe\trkwks	BLOCKED
\\.\pipe\srvsvc	BLOCKED

Figure 38. Chrome named pipe access

Summary

It is apparent that the Chrome sandbox prohibits the ability of the rendering process to do much of anything. There aren’t any easily viable ways for malware to gain persistence or communicate with the outside world. Any permissible actions, such as hooking windows messages, are mitigated by the fact that an alternate Windows desktop is used for rendering content. Out of the three browsers examined,

it is obvious that Google Chrome has the most stringent constraints when it comes to interacting with the operating system from a sandboxed process.

Microsoft Internet Explorer

ASLR Results

Accuvant examined each binary installed or loaded during browser startup to determine its ASLR compatibility. The *pefile* python library was used to check the **OPTIONAL_HEADER.DllCharacteristics** attribute to determine if a given module's address space would be randomized by the loader.

All the binaries that were currently loaded and being used in the browser were ASLR compatible, although it is quite difficult to predict all modules that Internet Explorer is capable of using. For a full listing, please see the [Internet Explorer ASLR](#) results in [Appendix A](#).

Note: We are aware that this list may be missing binaries and will attempt to update it if new modules are discovered. Also, note that this omits any third party/plugin modules.

DEP Results

As mentioned previously, Data Execution Prevention (DEP) prevents attackers from executing their *data* as *code*. By limiting execution rights to certain address spaces, DEP greatly reduces the attack surface. The default DEP policy for Windows 7 (32-bit) is **OptIn** [Microsoft_DEP]; meaning that the module will either have to be compiled with the /NXCOMPAT flag set or DEP will need to be enabled via **NTSetInformationProcess()** [Uninformed_DEP] (Windows XP & Windows 2003) or **SetProcessDEPPolicy()** [Microsoft_SPDEP] (Windows Vista and later).

Code: Please see *ieplorer.exe* disassembly.

```
.text:00402577 ; __stdcall Get_SetProcessDEPPolicy(x)
.text:00402577 _Get_SetProcessDEPPolicy@4 proc near ; CODE XREF: wWinMain(x,x,x,x)+100↑p
.text:00402577 ; wWinMain(x,x,x,x)+138↑p
.text:00402577 and dword ptr [esi], 0
.text:0040257A push offset aKernel32_dll_0 ; "Kernel32.DLL"
.text:0040257F call ds:__imp_GetModuleHandleW@4 ; GetModuleHandleW(x)
.text:00402585 test eax, eax
.text:00402587 jz short locret_40259B
.text:00402589 push offset aSetprocessdepp ; "SetProcessDEPPolicy"
.text:0040258E push eax ; hModule
.text:0040258F call ds:__imp_GetProcAddress@8 ; GetProcAddress(x,x)
.text:00402595 test eax, eax
.text:00402597 jz short locret_40259B
.text:00402599 mov [esi], eax
.text:0040259B
```

```
.text:00401348 loc_401348: ; CODE XREF: wWinMain(x,x,x,x)+F6↑j
.text:00401348 lea esi, [esp+30h+SetProcessDEPPolicy]
.text:0040134C mov [esp+30h+SetProcessDEPPolicy], ebx
.text:00401350 call _Get_SetProcessDEPPolicy@4 ; Get_SetProcessDEPPolicy(x)
.text:00401355 cmp [esp+30h+SetProcessDEPPolicy], ebx
.text:00401359 jz short loc_401365
.text:0040135B push edi
.text:0040135C call [esp+34h+SetProcessDEPPolicy]
.text:00401360 mov byte ptr [esp+30h+var_24+3], 1
```

Figure 39. Internet Explorer DEP being enabled

Process Examination: Process Explorer shows DEP enabled for the browser and rendering processes.

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name	DEP	ASLR	Integrity
services.exe	508		3,320 K	5,044 K			n/a		
svchost.exe	636		2,456 K	4,948 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	704		2,580 K	4,688 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	752		13,440 K	9,872 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	852		36,560 K	39,564 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
dwm.exe	1444		1,240 K	2,992 K	Desktop Window Manager	Microsoft Corporation	DEP (permanent)	ASLR	Medium
svchost.exe	892		12,368 K	12,680 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	1040		5,724 K	7,132 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	1136		10,592 K	6,192 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
spoolsv.exe	1232		6,504 K	8,332 K	Spooler SubSystem App	Microsoft Corporation	n/a	ASLR	
svchost.exe	1268		8,532 K	6,572 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
vmtoolsd.exe	1492		5,660 K	6,060 K	VMware Tools Core Service	VMware, Inc.	n/a		
VMUpgradeHelper.exe	1548		2,176 K	3,492 K	VMware virtual hardware up...	VMware, Inc.	n/a		
TPAutoConnSvc.exe	1916		1,932 K	4,180 K	TPAutoConnect Printer Creat...	ThinPrint AG	n/a		
TPAutoConnect.exe	2212		1,988 K	5,116 K	TPAutoConnect User Agent	ThinPrint AG			Medium
svchost.exe	1304		3,544 K	5,512 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
svchost.exe	1188		66,204 K	26,460 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
SearchIndexer.exe	1768		40,728 K	33,804 K	Microsoft Windows Search I...	Microsoft Corporation	n/a	ASLR	
taskhost.exe	500		6,848 K	5,524 K	Host Process for Windows T...	Microsoft Corporation	DEP (permanent)	ASLR	Medium
svchost.exe	3484		1,708 K	4,416 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
lsass.exe	528		2,556 K	5,572 K	Local Security Authority Proc...	Microsoft Corporation	n/a	ASLR	
lsm.exe	556		1,108 K	2,464 K			n/a		
csrss.exe	416		9,588 K	3,240 K			n/a		
conhost.exe	2220		532 K	1,840 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
conhost.exe	1344		784 K	3,228 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
winlogon.exe	464		1,552 K	3,296 K			n/a		
explorer.exe	352		33,112 K	43,996 K	Windows Explorer	Microsoft Corporation	DEP (permanent)	ASLR	Medium
VMwareTray.exe	776		2,220 K	3,840 K	VMware Tools tray application	VMware, Inc.			Medium
VMwareUser.exe	1756		7,924 K	11,096 K	VMware Tools Service	VMware, Inc.			Medium
iexplore.exe	2996		7,112 K	15,544 K	Internet Explorer	Microsoft Corporation	DEP (permanent)	ASLR	Medium
explore.exe	2744		39,824 K	37,624 K	Internet Explorer	Microsoft Corporation	DEP (permanent)	ASLR	Low
procexp.exe	1956		8,988 K	16,752 K	Sysinternals Process Explorer	Sysinternals - www.sysinter...	DEP (permanent)	ASLR	Medium
cmd.exe	2188		1,904 K	2,440 K	Windows Command Processor	Microsoft Corporation	DEP (permanent)	ASLR	Medium

Figure 40. Internet Explorer permanent DEP enabled

GS Results (Stack Cookies)

A test was run to determine if a module was compiled with the /GS compiler option [Microsoft_GS] for each item used by Internet Explorer. Although flawed*, this simple test attempted to see if a stack cookie named *object* was referenced within a binary. While having stack cookies doesn't completely prevent exploitation, it does make writing an exploit more difficult.

While all but two of the modules used by Internet Explorer are compiled with stack cookies, our IDA script to detect /GS presence could have false-positives due to a lack of debugging symbols for certain libraries or libraries without any code. We recognize that, currently, this is a flawed process and will develop a new /GS checking process in the future.

For a full listing of the stack cookie results for Internet Explorer, please reference the [Internet Explorer GS](#) section in [Appendix A](#).

***Note:** A module may not have the need for a stack cookie, which is determined by stack variable usage [Microsoft_GS].

JavaScript JIT Hardening

Codebase Alignment Randomization

The first code chunk emitted does not randomize its alignment, but when emitting code into an existing buffer, IE9 will prepend a random number (0-0x0F) of **INT 3** instructions prior to copying the compiled instructions into executable memory. This randomizes the offset of each subsequent instruction and has the additional benefit of crashing if ever executed.

Address	EmitBufferManager::GetAllocation	Comment
10021734	cmp byte ptr [ebp+arg_C], al	
10021739	call Math::Rand(void)	Get a random value
1002173E	and eax, 0Fh	Mask off the lower 4 bits

Figure 41. Internet Explorer alignment randomization

Hex Value	Instruction	Hex Value	Instruction
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
CC	int 3	CC	int 3
55	push ebp	55	push ebp
8BEC	mov ebp,esp	8BEC	mov ebp, esp
81FC54C91A03	cmp esp,31AC954h	81FC54C91A03	cmp esp,31AC954h
0F8F0F000000	jg 028D59F1	0F8F0F000000	jg 009259C9
6818FD5E02	push 25EFD18h	68086C5F02	push 25F6C08h
6854090000	push 954h	6854090000	push 954h

Figure 42. Internet Explorer code alignment randomization of the same script

Constant Blinding

Each constant value is XOR encoded with a separate cookie that is securely generated during compilation. Only values greater than 0x7FFF are obfuscated this way.

Script	Emitted Instruction	Hex Value
x = 0x00000001;	mov eax,3	b803000000
x = 0x00000002;	mov eax,5	b805000000
x = 0x00000003;	mov eax,7	b807000000
x = 0x0000007F;	mov eax,0FFh	b8ff000000
x = 0x000007FF;	mov eax,0FFFh	b8ff0f0000
x = 0x00007FFF;	mov eax,0FFFh	b8ffff0000
x = 0x00008000;	mov eax,17D562B2h xor eax,17D462B3h	b8b262d517 35b362d417
x = 0x0000FFFF;	mov eax,8D36BBD0h xor eax,8D37442Fh	b8d0bb368d 352f44378d
x = 0x00011111;	mov eax,0E9E6C90Ah xor eax,0E9E4EB29h	b80ac9e6e9 3529ebe4e9
x = 0x11111111;	mov eax,0BE262007h xor eax,9C040224h	b8072026be 352402049c

Figure 43. Internet Explorer constant binding

Constant Folding

Instruction operands will only contain odd values.

Emitted Value = (constant value << 1) + 1;

Script	Emitted Instruction	Hex Value
x = 0x00000001;	mov eax,3	b803000000
x = 0x00000002;	mov eax,5	b805000000
x = 0x00000003;	mov eax,7	b807000000
x = 0x0000007F;	mov eax,0FFh	b8ff000000
x = 0x000007FF;	mov eax,0FFFh	b8ff0f0000
x = 0x00007FFF;	mov eax,0FFFh	b8ffff0000

Figure 44. Internet Explorer constant folding

When combined with Constant Blinding results, only odd values of less than 0x07FFF can be emitted as instruction operands.

Memory Page Protection

Memory protection for the regions of emitted code is PAGE_EXECUTE, which will protect against both memory corruption and memory leaks targeting the emitted code.

The region is initially allocated as PAGE_EXECUTE. It is later marked PAGE_EXECUTE_READWRITE for the minimal period required to copy in the compiled code and return the protections to PAGE_EXECUTE.

JSCRIPT9!EmitBufferManager::CommitBuffer:	Comment
push eax	
push 40h	
push esi	
push ebx	
mov [ebp+lpAddress], ebx	
mov [ebp+dwSize], esi	
call Virtual Protect	Set protections to PAGE_EXECUTE_READWRITE
push esi	
push ecx	
push edx	
push ebx	
call memcpy_s	Copy compiled into PAGE_EXECUTE_READWRITE region
mov ecx, [ebp+dwSize]	
mov edx, [ebp+lpAddress]	
mov eax, [ebp+fOldProtect]	
push eax	
push 10h	
push ecx	
push edx	
call VirtualProtect	Set protections back to PAGE_EXECUTE

Figure 45. Internet Explorer memory page protections

Additionally, when the page protections are returned to PAGE_EXECUTE, the size of the memory is reduced to the minimal size. After this, the memory protection is not modified.

Resource Constraints

Constraints are placed on the total committed executable memory to prevent malicious content from spraying excessive amounts of executable memory.

The maximum total committed memory with protection PAGE_EXECUTE is 7MB (0x00700000).

BackgroundCodeGenThread::MainProc(void *)+A4	Comment
mov ecx, [ecx+1F8h]	Get size of currently committed PAGE_EXECUTE
cmp dword ptr [ecx+334h], 700000h	Compare against max PAGE_EXECUTE memory
jnb loc_10068A8E	If above, error out

Figure 46. Internet Explorer PAGE_EXECUTE allocation constraints

There is no constraint placed on the total committed PAGE_READWRITE protected memory.

Instruction Alignment Randomization

NOP equivalent instructions are randomly inserted into the stream during compilation to randomize the alignment of subsequent instructions.

Address	Instruction	Address	Instruction
026101B5	mov eax, 84564C4Bh	026101B5	mov eax, 689B3476h
026101BA	xor eax, 0A6746E68h	026101BA	xor eax, 4AB91655h
026101BF	mov ecx, [ebx+4]	026101BF	mov ecx, [ebx+4]
026101C2	cmp ecx, ds:7319018h	026101C2	cmp ecx, ds:7319018h
026101C8	jnz loc_26104F8	026101C8	jnz loc_26104F9
026101CE	mov ecx, [ebx+8]	026101CE	mov ecx, [ebx+8]
026101D1	movzx edx, word ptr 731901Eh	026101D1	movzx edx, word ptr 731901Eh
026101D8	lea ecx, [ecx]	026101D8	mov [ecx+edx*4], eax
026101DA	mov [ecx+edx*4], eax		

Figure 47. Internet Explorer random NOP instructions

Guard Pages

IE9 has no need for guard pages because after initialization, regions that contain emitted code are protected with PAGE_EXECUTE, which remains for the lifetime of the script and protects against corruption and information leaks. It can be modified briefly while appending additional code to the current region.

Additional Randomization

The low constraint on PAGE_EXECUTE allocations combined with codebase and instruction alignment randomization are effective enough that additional randomization is not required.

Sandbox Results

All sandbox testing was performed from inside the lowest privileged process (rendering/render process) by attempting to open new resources. Therefore, the results only reflect a rendering process attempting to access resources not previously opened.

Note: The sandbox test harness specifically ignored the function hooks set in place by IEShims.dll due to the way most shellcode locates and executes functions.

File System

We attempted to access certain system directories and files via the sandboxed browser process (sometimes referred to as the *renderer/rendering process*).

Note: Permissions may have overlap due to generic and specific permission checks. This is done to give an overview accompanied by precise security permissions.

Permission	%SystemDrive% %SystemRoot% %ProgramFiles% %AllUsersProfile% %UserProfile% %Temp% %SystemRoot%\System32 %AppData%
ZERO	GRANTED
GENERIC_READ	GRANTED
GENERIC_WRITE	BLOCKED
FILE_ADD_FILE	BLOCKED
FILE_ADD_SUBDIRECTORY	BLOCKED
FILE_APPEND_DATA	BLOCKED
FILE_CREATE_PIPE_INSTANCE	BLOCKED
FILE_DELETE_CHILD	BLOCKED
FILE_LIST_DIRECTORY	GRANTED
FILE_READ_ATTRIBUTES	GRANTED
FILE_READ_DATA	GRANTED
FILE_READ_EA	GRANTED
FILE_TRAVERSE	GRANTED
FILE_WRITE_ATTRIBUTES	BLOCKED
FILE_WRITE_DATA	BLOCKED
FILE_WRITE_EA	BLOCKED
WRITE_DAC	BLOCKED

Figure 48. Internet Explorer directory permissions

Files

Permission	%SystemRoot%\explorer.exe %SystemRoot%\Cursors\arrow_i.cur
ZERO	GRANTED
GENERIC_READ	GRANTED
GENERIC_WRITE	BLOCKED
GENERIC_EXECUTE	GRANTED
FILE_EXECUTE	GRANTED
FILE_READ_ATTRIBUTES	GRANTED
STANDARD_RIGHTS_EXECUTE	GRANTED
SYNCHRONIZE	GRANTED
FILE_READ_DATA	GRANTED
FILE_READ_EA	GRANTED
STANDARD_RIGHTS_READ	GRANTED
FILE_APPEND_DATA	BLOCKED
FILE_WRITE_ATTRIBUTES	BLOCKED
FILE_WRITE_DATA	BLOCKED
FILE_WRITE_EA	BLOCKED
STANDARD_RIGHTS_WRITE	GRANTED
WRITE_DAC	BLOCKED

Figure 49. Internet Explorer file permissions

Registry

A select few registry hives and keys were accessed from inside the sandbox. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Note: The *MAXIMUM_ALLOWED* permission for a **low integrity** process is read-only.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	GRANTED

Figure 50. Internet Explorer registry permissions

Network Access

The ability for a browser to access the Internet is vital to its operation, but creating a new socket for reading, writing and listening could permit an attacker to communicate read-able information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	GRANTED
Send ()	GRANTED
Recv ()	GRANTED
Listen () [port 88]	GRANTED

Figure 51. Internet Explorer network accessibility

Resource Monitoring

Recording keystrokes, registering hotkeys and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire user information (obviously, there are various other techniques.)

Action	Result
GetPixel ()	GRANTED
RegisterHotKey ()	GRANTED
GetAsyncKeyState ()	BLOCKED

Figure 52. Internet Explorer resource monitoring

Threads

Access to other threads running on the system could be used to escalate privileges or trampoline onto different parts of the system. The sandbox test harness provides functionality that tests the access privileges of every thread currently alive on the system. Since this list can vary too greatly, only a list of threads and the privileges granted will be supplied.

Note that a majority of the threads could **not** be accessed, but those that could generally only permitted *SYNCHRONIZE* and *THREAD_QUERY_LIMITED_INFORMATION* access (there were some that had full access). Please see 'ie9_sandbox_results.txt' for a full listing. Also, the threads used in the sandbox testing were **omitted**.

Process:Thread ID	Permission Granted
N/A	N/A

Figure 53. Internet Explorer threads permission granted

Processes

Processes, like threads, can be used to escalate privileges or trampoline onto different parts of the system. The harness provides functionality to test the access privileges of every process currently alive on the system. Only processes that granted certain access are listed. In addition, processes specific to VMWare, the sandbox test harness and instances of **low integrity** Internet Explorer were **omitted**. Please see 'ie9_sandbox_results.txt' in the attachment for a full listing.

Process	Permission Granted
taskhost.exe	PROCESS_QUERY_LIMITED_INFORMATION
taskhost.exe	PROCESS_TERMINATE
dwm.exe	PROCESS_QUERY_LIMITED_INFORMATION
dwm.exe	PROCESS_TERMINATE
dwm.exe	SYNCHRONIZE
explorer.exe	PROCESS_QUERY_LIMITED_INFORMATION
explorer.exe	PROCESS_TERMINATE
explorer.exe	SYNCHRONIZE
conhost.exe	PROCESS_QUERY_LIMITED_INFORMATION
conhost.exe	PROCESS_TERMINATE
conhost.exe	SYNCHRONIZE
iexplore.exe	PROCESS_QUERY_LIMITED_INFORMATION
iexplore.exe	PROCESS_TERMINATE
iexplore.exe	SYNCHRONIZE

Figure 54. Internet Explorer processes granted

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization and privilege level as the compromised application. This could permit a plethora of other opportunities for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with "C:\Program Files\Internet Explorer\iexplore.exe" was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	GRANTED

Figure 55. Internet Explorer CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them gain privileges.

For more granular information on handling test cases, please see 'ie9_sandbox_results.txt' in the attachment.

Windows Clipboard

The Windows clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read potentially sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to *GET* and *SET* information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	BLOCKED
SetClipboardData (CF_TEXT)	BLOCKED

Figure 56. Internet Explorer clipboard access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	GRANTED
OpenWindowsStation ("winsta0")	GRANTED
SetProcessWindowStation ("winsta0")	GRANTED
OpenDesktop ("Default")	GRANTED

Figure 57. Internet Explorer desktop/WindowStation access

System Parameters

It should be obvious that an attacker could use the system wide parameters to his advantage. These parameters can control screen savers, menus and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	BLOCKED

Figure 58. Internet Explorer SystemParametersInfo()

Note: Only a single system parameter was checked for brevity's sake.

Windows Message Broadcasts

By sending a Windows Message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could interpret the broadcast message differently; due to expecting a varying number of parameters [MSDN_Broad]. This could cause many operating system instability and exploitation scenarios. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_TIMER)	GRANTED

Figure 59. Internet Explorer send broadcast message

Windows Hooks

Windows hooks are a procedure used to monitor certain types of system events on the same desktop as the calling thread [Microsoft_Hooks]. These same hooks historically have been used by malware to do such things as monitor keyboard input and other nefarious tasks. We checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 60. Internet Explorer set Windows hooks

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows operating system [Microsoft_Pipes]. While the ability to communicate between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We attempted to enumerate all the named pipes for a system for permissions testing. If that were not possible, we would iterate through a list of ‘well-known’ pipes for the Windows 7 (32-bit) operating system in an attempt to validate permissions. For a complete listing, please see ‘ie9_sandbox_results.txt’ in the attachment.

Named Pipe	PIPE_ACCESS_INBOUND	PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	GRANTED	INDETERMINATE [Resource not Available]
\\.\pipe\ntsvcs	GRANTED	BLOCKED
\\.\pipe\scerpc	GRANTED	BLOCKED
\\.\pipe\protected_storage	GRANTED	GRANTED
\\.\pipe\plugplay	GRANTED	BLOCKED
\\.\pipe\epmapper	GRANTED	BLOCKED
\\.\pipe\eventlog	GRANTED	BLOCKED
\\.\pipe\atsvc	GRANTED	BLOCKED
\\.\pipe\wkssvc	GRANTED	GRANTED
\\.\pipe\keysvc	GRANTED	BLOCKED
\\.\pipe\trkwks	GRANTED	BLOCKED
\\.\pipe\srvsvc	GRANTED	GRANTED

Figure 61. Internet Explorer named pipe access

Summary

Internet Explorer permits the low privileged browser processes limited interaction with the operating system by permitting read-only accessibility to most resources. While the ability to prevent attacker persistence through limiting write access is notable, the attacker still has the ability to read and export information to the outside world via network sockets. In addition, since there is no alternate desktop and desktops can be created; there is an ability to communicate with other non-sandboxed window objects. Named pipes are also accessible for reading and writing, which could result in additional attack surfaces used for privilege escalation (i.e. escalating from low integrity to a higher integrity). Overall, Internet Explorer does a satisfactory job of preventing malware persistence, but lacks the ability to lock down browser processes that may be exploited to gain further privilege level by using available system resources.

Mozilla Firefox

ASLR Results

Each binary installed or loaded during browser startup was investigated to determine its ASLR compatibility. The *pefile* python library was used to check the **OPTIONAL_HEADER.DllCharacteristics** attribute to determine if a given module's address space would be randomized by the loader.

All the binaries that were currently loaded and being used by the browser were ASLR compatible, although it is difficult to predict all modules that could be loaded. For a full listing, please see the [Mozilla Firefox ASLR](#) results in [Appendix A](#).

Note: We are aware that this list may be missing binaries and will attempt to update it if new modules are discovered. Also, note that this omits any third party/plugin modules.

DEP Results

As mentioned previously, Data Execution Prevention (DEP) prevents attackers from executing their *data* as *code*. By limiting execution rights to certain address spaces, DEP reduces the attack surface. The default DEP policy for Windows 7 (32-bit) is **OptIn** [Microsoft_DEP]; meaning that the module will either have to be compiled with the `/NXCOMPAT` flag set or DEP will need to be enabled via `NTSetInformationProcess()` [Uninformed_DEP] (Windows XP & Windows 2003) or `SetProcessDEPPolicy()` [Microsoft_SPDEP] (Windows Vista and later).

NXCOMPAT: Bit 0x100 Set in DllCharacteristics.

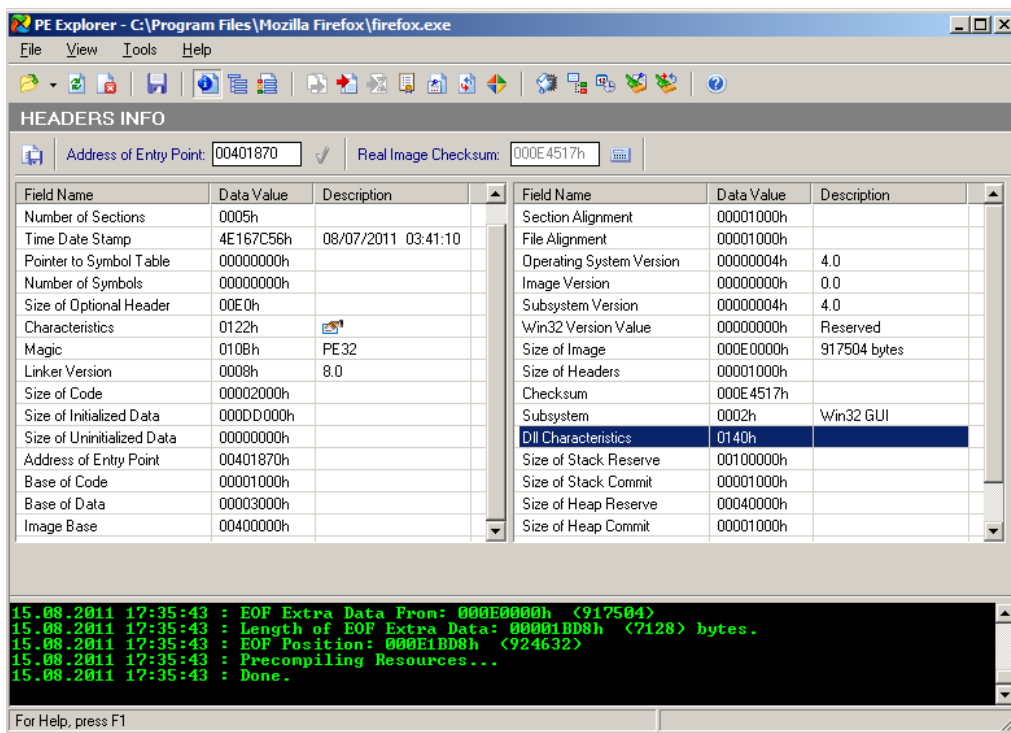
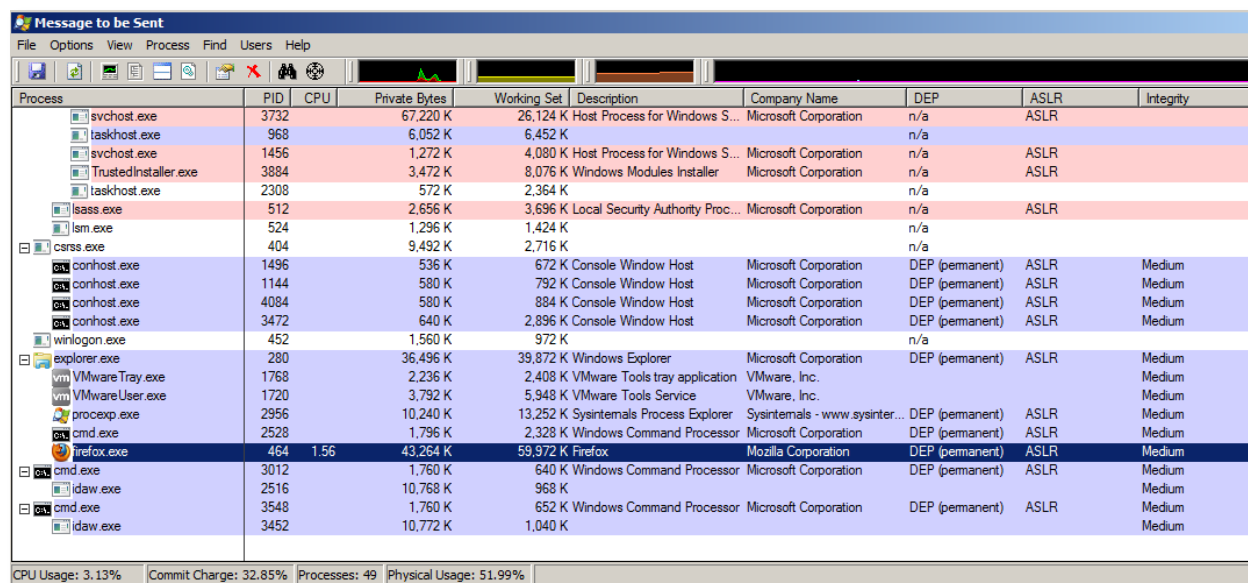


Figure 62. Firefox DEP enabled

Process Examination: Process Explorer shows DEP enabled for the browser.



Process	PID	CPU	Private Bytes	Working Set	Description	Company Name	DEP	ASLR	Integrity
svchost.exe	3732		67,220 K	26,124 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
taskhost.exe	968		6,052 K	6,452 K			n/a		
svchost.exe	1456		1,272 K	4,080 K	Host Process for Windows S...	Microsoft Corporation	n/a	ASLR	
TrustedInstaller.exe	3884		3,472 K	8,076 K	Windows Modules Installer	Microsoft Corporation	n/a	ASLR	
taskhost.exe	2308		572 K	2,364 K			n/a		
lsass.exe	512		2,656 K	3,696 K	Local Security Authority Proc...	Microsoft Corporation	n/a	ASLR	
lsm.exe	524		1,296 K	1,424 K			n/a		
csrss.exe	404		9,482 K	2,716 K			n/a		
conhost.exe	1496		536 K	672 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
conhost.exe	1144		580 K	792 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
conhost.exe	4084		580 K	884 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
conhost.exe	3472		640 K	2,896 K	Console Window Host	Microsoft Corporation	DEP (permanent)	ASLR	Medium
winlogon.exe	452		1,560 K	972 K			n/a		
explorer.exe	280		36,496 K	39,872 K	Windows Explorer	Microsoft Corporation	DEP (permanent)	ASLR	Medium
VMwareTray.exe	1768		2,236 K	2,408 K	VMware Tools tray application	VMware, Inc.			Medium
VMwareUser.exe	1720		3,792 K	5,948 K	VMware Tools Service	VMware, Inc.			Medium
proccexp.exe	2956		10,240 K	13,252 K	Sysinternals Process Explorer	Sysinternals - www.sysinter...	DEP (permanent)	ASLR	Medium
cmd.exe	2528		1,796 K	2,328 K	Windows Command Processor	Microsoft Corporation	DEP (permanent)	ASLR	Medium
firefox.exe	464	1.56	43,264 K	59,972 K	Firefox	Mozilla Corporation	DEP (permanent)	ASLR	Medium
cmd.exe	3012		1,760 K	640 K	Windows Command Processor	Microsoft Corporation	DEP (permanent)	ASLR	Medium
idaw.exe	2516		10,768 K	968 K					Medium
cmd.exe	3548		1,760 K	652 K	Windows Command Processor	Microsoft Corporation	DEP (permanent)	ASLR	Medium
idaw.exe	3452		10,772 K	1,040 K					Medium

CPU Usage: 3.13% | Commit Charge: 32.85% | Processes: 49 | Physical Usage: 51.99%

Figure 63. Firefox permanent DEP enabled

GS Results (Stack Cookies)

A test was run to determine if a module was compiled with the /GS compiler option [Microsoft_GS] for each item used by Mozilla Firefox. Although flawed*, this simple test attempted to see if a stack cookie named *object* was referenced within a binary. While having stack cookies doesn't completely prevent exploitation, it does make writing an exploit more difficult.

All modules tested that were used by Mozilla Firefox are compiled with stack cookies.

For a full listing of the stack cookie results for Mozilla Firefox, please reference the [Mozilla Firefox GS results](#) section in [Appendix A](#).

***Note:** A module may not have the need for a stack cookie, which is determined by stack variable usage [Microsoft_GS].

JavaScript JIT Hardening

Firefox does not implement any JIT hardening techniques.

Sandbox Results

At the time of this writing, Mozilla Firefox does not implement any type of formal sandbox technology. It relies on the Windows integrity level given to the application on start-up (by default, this is *medium*). The following results are shown, not as sandbox results, but the inverse, the browser's capabilities when no sandbox or additional integrity restrictions are implemented.

File System

Certain directories and files were accessed attempting to show the browsers file system access capabilities.

Note: Permissions may overlap due to generic and specific permission checks. This is done to give an overview accompanied by precise security permissions.

Permission	%SystemDrive%	%SystemRoot% %ProgramFiles% %SystemRoot%\System32	%UserProfile% %Temp% %AppData%
ZERO	GRANTED	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED	GRANTED
GENERIC_WRITE	GRANTED	BLOCKED	GRANTED
FILE_ADD_FILE	GRANTED	BLOCKED	GRANTED
FILE_ADD_SUBDIRECTORY	GRANTED	BLOCKED	GRANTED
FILE_APPEND_DATA	GRANTED	BLOCKED	GRANTED
FILE_CREATE_PIPE_INSTANCE	GRANTED	BLOCKED	GRANTED
FILE_DELETE_CHILD	GRANTED	BLOCKED	GRANTED
FILE_LIST_DIRECTORY	GRANTED	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED	GRANTED
FILE_TRAVERSE	GRANTED	GRANTED	GRANTED
FILE_WRITE_ATTRIBUTES	GRANTED	BLOCKED	GRANTED
FILE_WRITE_DATA	GRANTED	BLOCKED	GRANTED
FILE_WRITE_EA	GRANTED	BLOCKED	GRANTED
WRITE_DAC	GRANTED	BLOCKED	GRANTED

Figure 64. Firefox directory permissions I

Permission	%AllUsersProfile%
ZERO	GRANTED
GENERIC_READ	GRANTED
GENERIC_WRITE	GRANTED
FILE_ADD_FILE	GRANTED
FILE_ADD_SUBDIRECTORY	GRANTED
FILE_APPEND_DATA	GRANTED
FILE_CREATE_PIPE_INSTANCE	GRANTED
FILE_DELETE_CHILD	BLOCKED
FILE_LIST_DIRECTORY	GRANTED
FILE_READ_ATTRIBUTES	GRANTED
FILE_READ_DATA	GRANTED
FILE_READ_EA	GRANTED
FILE_TRAVERSE	GRANTED
FILE_WRITE_ATTRIBUTES	GRANTED
FILE_WRITE_DATA	GRANTED
FILE_WRITE_EA	GRANTED
WRITE_DAC	BLOCKED

Figure 65. Firefox directory permissions II

Files

Permission	%SystemRoot%\explorer.exe %SystemRoot%\Cursors\arrow_i.cur
ZERO	GRANTED
GENERIC_READ	GRANTED
GENERIC_WRITE	BLOCKED
GENERIC_EXECUTE	GRANTED
FILE_EXECUTE	GRANTED
FILE_READ_ATTRIBUTES	GRANTED
STANDARD_RIGHTS_EXECUTE	GRANTED
SYNCHRONIZE	GRANTED
FILE_READ_DATA	GRANTED
FILE_READ_EA	GRANTED
STANDARD_RIGHTS_READ	GRANTED
FILE_APPEND_DATA	BLOCKED
FILE_WRITE_ATTRIBUTES	BLOCKED
FILE_WRITE_DATA	BLOCKED
FILE_WRITE_EA	BLOCKED
STANDARD_RIGHTS_WRITE	GRANTED
WRITE_DAC	BLOCKED

Figure 66. Firefox file permissions

Registry

A select few registry hives and keys were accessed. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Note: The *MAXIMUM_ALLOWED* permission by a **medium integrity** process is read-only for all hives and keys other than *HKEY_CURRENT_USER*, which is read/write.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	GRANTED

Figure 67. Firefox registry permissions

Network Access

The ability for a browser to access the Internet is vital to its operation, but creating a new socket for reading, writing and listening could permit an attacker to communicate read-able information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	GRANTED
Send()	GRANTED
Recv()	GRANTED
Listen() [port 88]	GRANTED

Figure 68. Firefox network accessibility

Resource Monitoring

Recording keystrokes, registering hotkeys, and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in their attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire user information (obviously, there are various other techniques).

Action	Result
GetPixel()	GRANTED
RegisterHotKey()	GRANTED
GetAsyncKeyState()	GRANTED

Figure 69. Firefox resource monitoring

Threads

Access to other threads running on the system can be used to escalate privileges or trampoline onto different parts of the system. The sandbox test harness provides functionality that tests the access privileges of every thread currently alive on the system. Since this list can vary greatly, only a list of threads and the privileges granted will be supplied.

Since there is no sandbox in place, all threads that are running with **medium integrity** will grant *full-access* to the Firefox process. Please see 'firefox5_nosandbox_results.txt' for a full listing.

Process:Thread ID	Permission Granted
N/A	N/A

Figure 70. Firefox threads permission granted

Processes

Processes, like threads, can be used to escalate privileges or trampoline onto different parts of the system. Just like threads, all processes that are **medium integrity** will be fully accessible by the Firefox process. Due to the number of accessible processes, they have been omitted from this paper. Please see 'firefox5_nosandbox_results.txt' for a full listing.

Process	Permission Granted
N/A	N/A

Figure 71. Firefox processes permission granted

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization and privilege level as the compromised application. This could permit a plethora of other opportunities that could be used for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with "C:\Program Files\Internet Explorer\iexplore.exe" was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	GRANTED

Figure 72. Firefox CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them gain privileges.

For more granular information on handling test cases, please see 'firefox5_nosandbox_results.txt' in the attachment.

Windows Clipboard

The Windows clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to *GET* and *SET* information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	GRANTED
SetClipboardData (CF_TEXT)	GRANTED

Figure 73. Firefox Clipboard access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	GRANTED
OpenWindowsStation ("winsta0")	GRANTED
SetProcessWindowStation ("winsta0")	GRANTED
OpenDesktop ("Default")	GRANTED

Figure 74. Firefox Desktop/WindowStation permissions

System Parameters

It should be obvious that an attacker could use the system wide parameters to his advantage. These parameters can control screen savers, menus and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	GRANTED

Figure 75. Firefox SystemParametersInfo()

Note: Only a single system parameter was checked for brevity's sake.

Windows Message Broadcasts

By sending a Windows message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could interpret the broadcast message differently; due to expecting a varying number of parameters [MSDN_Broad]. This could cause many operating system instability and exploitation scenarios. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_TIMER)	GRANTED

Figure 76. Firefox send broadcast message

Windows Hooks

Windows hooks are a procedure used to monitor certain types of system events on the same desktop as the calling thread [Microsoft_Hooks]. These same hooks historically have been used by malware to do such things as monitor keyboard input and other nefarious tasks. We have checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 77. Firefox set Windows hooks

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows operating system [Microsoft_Pipes]. While this ability to communicate between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We attempted to enumerate all the named pipes for a system for permissions testing. If that were not possible, we would iterate through a list of ‘well-known’ pipes for the Windows 7 (32-bit) operating system attempting to validate permissions.

Named Pipe	PIPE_ACCESS_INBOUND	PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	GRANTED	INDETERMINATE [Resource not Available]
\\.\pipe\ntsvcs	GRANTED	GRANTED
\\.\pipe\scerpc	GRANTED	GRANTED
\\.\pipe\protected_storage	GRANTED	GRANTED
\\.\pipe\plugplay	GRANTED	GRANTED
\\.\pipe\epmapper	GRANTED	GRANTED
\\.\pipe\eventlog	GRANTED	GRANTED
\\.\pipe\atsvc	GRANTED	GRANTED
\\.\pipe\wkssvc	GRANTED	GRANTED
\\.\pipe\keysvc	GRANTED	GRANTED
\\.\pipe\trkwks	GRANTED	GRANTED
\\.\pipe\srvsvc	GRANTED	GRANTED

Figure 78. Firefox named pipe access

Summary

Firefox has yet to implement a formal sandbox at the time of this writing, relying solely on the default Windows integrity level (medium). This leaves many doors open for attackers when it comes to persistence, external communication and additional operating system resources used for privilege escalation. Out of all the browsers tested for exploit and persistence mitigation, Firefox contains the most potential from an attacker’s perspective.

Browser Add-Ons

Browser add-ons are software components, typically written by third parties that add to the functionality of the browser. Such new functionality may include the ability for browsers to play video, scan for viruses, view particular file types, make modifications to displayed HTML, and many other tasks. These add-ons include components such as plug-ins, extensions and themes. The exact types of add-ons available vary from browser to browser. Common examples of add-ons include the Firefox NoScript extension and plug-ins that allow viewing of files normally used by Microsoft Office, Shockwave Flash and Adobe Acrobat Reader.

Adding new code to the browser represents several potential security risks. One may be that the new code itself may be malicious. In this case, it is important to consider whether the add-ons are reviewed in some manner before being offered. This helps prevent malicious add-ons from being installed. Another important issue regarding malicious add-ons is whether they alert the user that they are being installed or that they can be silently installed in the background. Clearly, if add-ons can be installed without user notification/interaction, it can result in code being placed on the system without user knowledge.

Another way that add-ons introduce security risk is by introducing new security vulnerabilities into the browser, which may be exploited by attackers. A number of factors play into how plug-in risk is handled by the browser. Add-ons may be sandboxed to prevent compromised code from causing significant harm. (This helps in malicious add-on cases, as well). Another risk to consider is whether including add-ons weakens the overall security posture of the browser by weakening exploit mitigation technologies such as ASLR or DEP.

Yet another risk to consider is if the plug-ins can be silently activated or require user interaction. As stated previously, permitting add-on installation without user acknowledgment presents a scenario where unauthorized code can stealthily be installed on the system under the guise of a browser 'add-on'.

Lastly, we should take the concept of add-on management into consideration. If a vulnerability is discovered in a third party browser plug-in, then it should be easily remedied through an upgrade or patch release. Unmanageable add-ons can result in upgrade and patching neglect, leaving users with old, vulnerable plug-ins for extended periods.

First, each browser's extension methodology was identified. Then, the most critical ways to extend the browser were observed; how they were installed, what types of changes the add-ons could make, extension limitations and add-on review procedure. We next built custom plug-ins to test feature limitations by including in them the sandbox test harness used in the browser testing section. Lastly, we examined each add-on's update and management mechanisms.

Browser Comparison

Feature	Chrome	Internet Explorer	Firefox
Pre-install Warning	✓	✓	✓
Auto Updates	✓	✓	✓
Permission Model	✓	✗	✗
Sandbox	●	✗	✗
Read Files	✗	✗	✗
Write Files	●	●	●
Read Registry Keys	✗	✗	✗
Write Registry Keys	●	●	●
Network Access	✗	✗	✗
Resource Monitoring	✗	●	✗
Thread Access	●	●	●
Process Access	●	●	●
Process Creation	✗	✗	✗
Clipboard Access	✗	✓	✗
System Parameters	✗	●	✗
Broadcast Messages	✗	✗	✗
Desktop & Windows Station Access	✗	✗	✗
Windows Hooks	✗	✗	✗
Named Pipes Access	✗	●	✗

- ✓ Good
- Acceptable
- ✗ Bad

Figure 79. Browser comparison

As you can see from the table above, the capabilities of browser plug-ins/add-ons differ greatly from the browser itself, due to their demanding nature (i.e. interacting with the browser itself). A redeeming feature is that many of the plug-ins don't come pre-installed and must be installed by the end user. The security of add-ons should be considered just as high a priority as the browser.

Google Chrome

Installation

There are multiple ways to add code to Chrome, including installable web apps, themes, extensions and plug-ins. These are all referred to as *apps for Chrome*. An app can load DLLs by using the NPAPI interface used by Firefox plug-ins. All extensions come in a single .crx file, which is a cryptographically signed, zip file.

A theme contains no dynamic content and therefore poses no real security concern.

A hosted web app is a simple wrapper around an actual web site on the Internet. A packaged web app is a web application that comes in a .crx file and can use Chrome extension features. A packaged web app contains all the HTML, CSS, and other files needed to view the web application. They have almost all the functionality offered by extensions, except they cannot add buttons to the address bar. In addition, no native code can be included in installable web apps.

Extensions have the most access to elements of the browser and may include native code. Instead of just centering on a single web site or application, extensions are designed to affect how all web sites function within the browser.

Plug-ins includes any extension or other mechanism, such as a Firefox plug-in, that includes native code.

User interaction is required when installing add-ons from the Chrome Web Store, see the figure below. Notice that the prompt includes information on what access the extension requires.

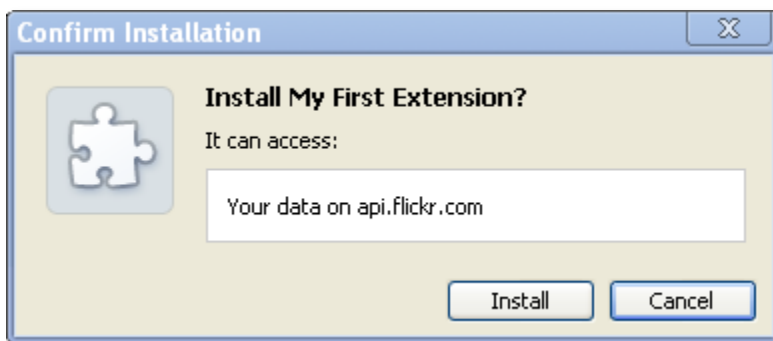


Figure 80. Packaged extensions must declare permissions

This information can be provided because, in order to use most of the APIs available to Chrome extensions, the extension must declare which API it needs to access in its manifest file. (The only exception is unpacked extensions, but these can only be installed in Developer mode). In the example above, the extension needs to access data from the site <http://api.flickr.com>, but that is the only site whose data it can access. This extension's manifest file looked like:

```
{
  "name": "My First Extension",
  "version": "1.0",
  "description": "The first extension that I made.",
  "browser_action": {
    "default_icon": "icon.png"
  },
  "permissions": [
    "http://api.flickr.com/"
  ]
}
```

Figure 81. File extension manifest

A list of permissions that must be requested and approved before use is in the permissions section of the manifest. This list of permissions include things such as

- Plug-ins
- Bookmarks
- History
- Management
- Geolocation
- clipboardRead

If the extension includes native code, it needs the “Plug-ins” manifest entry and the warning indicates that it can do just about anything.

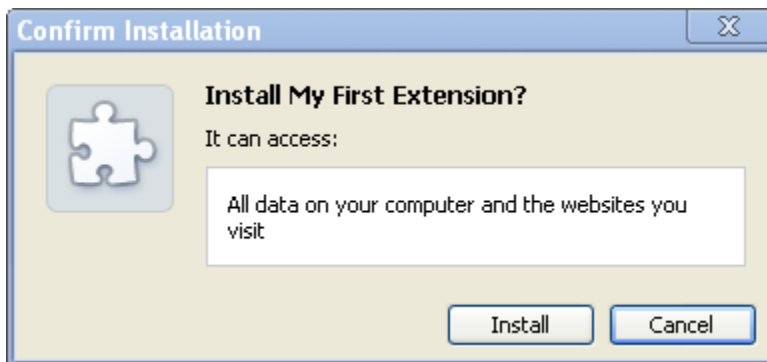


Figure 82. Chrome native code plug-in extensions throw a warning

Extensions that are loaded from somewhere besides the Chrome Web Store display an additional dialogue, see the figure below

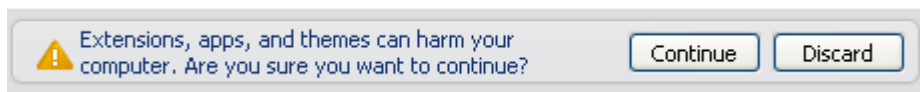


Figure 83. Chrome unsigned apps require user interaction

After this prompt, the extension then declares which API it needs in order to function.

This additional prompt encourages developers to use the Chrome Web Store. Furthermore, in order to reduce the problem of extensions simply requesting access to everything, the Chrome Web Store requires extensions with the Plug-in permission to sign a contract with Google.

Additionally, if an extension is found to be malicious or otherwise harmful, it can be removed from the Web Store and can also be remotely blacklisted and uninstalled from all Chrome browsers.

Any native code runs in a separate Chrome process while other extensions run in the renderer process. Once installed, extensions and plug-ins can be activated without user interaction, with only a few exceptions. Some plug-ins, such as Java, requires user interaction before it can be run, see figure 90. Google indicates that plug-ins with a history of security problems that are not widely used are blocked in this manner by Chrome to prevent websites from loading the plug-in and exploiting it.

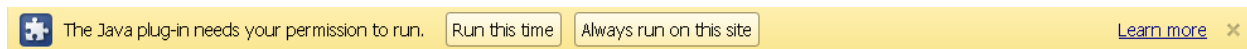


Figure 84. Chrome java approval pre-instantiation

Another interesting feature is that, while plug-ins and extensions are automatically updated, the underlying services, such as the Java runtime environment, may get out of date. When this happens, Chrome again warns you, giving you the option of updating or just running the out of date plug-in.



Figure 85. Chrome plug-in out of data indicator

Sandbox Results

While most Chrome add-ons run in the renderer sandbox (see earlier section), plug-ins do not. Therefore, when examining sandboxing for Chrome add-ons, we look at the worst-case scenario and look at the restrictions on native code plug-ins. Plug-ins do not run in a sandbox or with any restrictions, besides those imposed due to not being run as administrator. This is revealed by running the sandboxing script.

File System

We attempted to access certain system directories and files via the sandboxed plug-in.

Note: Permissions may overlap due to generic and specific permission checks. This is done to give an overview accompanied by precise security permissions.

Permission	%SystemRoot%, %ProgramFiles%, %SystemRoot%\System32	%AllUsersProfile%
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	BLOCKED	GRANTED
FILE_ADD_FILE	BLOCKED	GRANTED
FILE_ADD_SUBDIRECTORY	BLOCKED	GRANTED
FILE_APPEND_DATA	BLOCKED	GRANTED
FILE_CREATE_PIPE_INSTANCE	BLOCKED	GRANTED
FILE_DELETE_CHILD	BLOCKED	BLOCKED
FILE_LIST_DIRECTORY	GRANTED	GRANTED

Permission	%SystemRoot%, %ProgramFiles%, %SystemRoot%\System32	%AllUsersProfile%
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
FILE_TRAVERSE	GRANTED	GRANTED
FILE_WRITE_ATTRIBUTES	BLOCKED	GRANTED
FILE_WRITE_DATA	BLOCKED	GRANTED
FILE_WRITE_EA	BLOCKED	GRANTED
WRITE_DAC	BLOCKED	BLOCKED

Figure 86. Chrome add-on directory permissions

Permission	%SystemDrive%, %UserProfile%, %Temp%, %AppData%	%SystemRoot%\explorer.exe, %SystemRoot%\Cursors\arrow_i.cur
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	GRANTED	BLOCKED
GENERIC_EXECUTE	GRANTED	GRANTED
FILE_EXECUTE	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
STANDARD_RIGHTS_EXECUTE	GRANTED	GRANTED
SYNCHRONIZE	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
STANDARD_RIGHTS_READ	GRANTED	GRANTED
FILE_APPEND_DATA	GRANTED	BLOCKED
FILE_WRITE_ATTRIBUTES	GRANTED	BLOCKED
FILE_WRITE_DATA	GRANTED	BLOCKED
FILE_WRITE_EA	GRANTED	BLOCKED
STANDARD_RIGHTS_WRITE	GRANTED	GRANTED
WRITE_DAC	GRANTED	BLOCKED

Figure 87. Chrome add-on file permissions

Registry

A select few registry hives/keys were accessed from inside the plug-in. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	GRANTED

Figure 88. Chrome add-on registry permissions

Network Access

The ability for a browser to access the Internet is vital to its operation, but creating a new socket for reading, writing and listening could permit an attacker to communicate read-able information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	GRANTED
Send ()	GRANTED
Recv ()	GRANTED
Listen ()	GRANTED

Figure 89. Chrome add-on network accessibility

Resource Monitoring

Recording keystrokes, registering hotkeys and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in their attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire this information (obviously, there are various other techniques).

Action	Result
GetPixel ()	GRANTED
RegisterHotKey ()	GRANTED
GetAsyncKeyState ()	GRANTED

Figure 90. Chrome add-on resource monitoring

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization as the compromised application. This could permit a plethora of other options that could be used for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with "C:\Program Files\Internet Explorer\iexplore.exe" was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	GRANTED

Figure 91. Chrome add-on CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them gain privileges.

For more granular information on handling test cases, please see 'chrome-addons.txt' in the attachment.

Windows Clipboard

The Windows clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to *GET* and *SET* information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	GRANTED
SetClipboardData (CF_TEXT)	GRANTED

Figure 92. Chrome add-on Clipboard access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	GRANTED
OpenWindowsStation ("winsta0")	GRANTED
OpenDesktop ("Default")	GRANTED

Figure 93. Chrome add-on Desktop/WindowStation permissions

System Parameters

It should be obvious that an attacker could use the system wide parameters to his advantage. These parameters can control screen saver parameters, menu parameters and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

Note: Only a single system parameter was checked for brevity's sake.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	GRANTED

Figure 94. Chrome add-on SystemParametersInfo()

Windows Message Broadcasts

By sending a Windows message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could potentially interpret the broadcast message differently, each expecting a varying number of parameters [MSDN_Broad]. This could obviously cause operating system instability and exploitation scenarios by the handful. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_SETTEXT)	GRANTED

Figure 95. Chrome add-on send broadcast message

Windows Hooks

Windows hooks are a procedure used to monitor certain types of system events for a specific thread or all threads in the same desktop as the calling thread [Microsoft_Hooks]. These same hooks historically have been used by malware to do such things as monitor keyboard input and other nefarious tasks. We checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 96. Chrome add-on set Windows hooks

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows operating system [Microsoft_Pipes]. While this ability to communication between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We attempted to enumerate all the named pipes for a system for permissions testing. If that was not possible, we would iterate through a list of ‘well-known’ pipes for the Windows 7 (32-bit) operating system in an attempt to validate permissions.

Named Pipe	PIPE_ACCESS_INBOUND PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	GRANTED
\\.\pipe\ntsvcs	GRANTED
\\.\pipe\scerpc	GRANTED
\\.\pipe\protected_storage	GRANTED
\\.\pipe\plugplay	GRANTED
\\.\pipe\epmapper	GRANTED
\\.\pipe\eventlog	GRANTED
\\.\pipe\atsvc	GRANTED
\\.\pipe\wkssvc	GRANTED
\\.\pipe\keysvc	GRANTED
\\.\pipe\trkwks	GRANTED
\\.\pipe\srvsvc	GRANTED

Figure 97. Chrome add-on named pipe access

There are a couple of exceptions to the above sandbox tests. Most web browsers do not come with extensions or plug-ins by default. Chrome includes a plug-in for viewing PDFs and Flash content, two of the more popular formats on the web. As we mentioned above, Chrome cannot usually control the version of the underlying content parsers associated with a particular plug-in, but by including them within Chrome, they can control the versions of everything. In the case of the PDF viewer, it also allows Chrome to place it in the full sandbox, which it cannot typically do with plug-ins.

So, Chrome includes more attack surface by including the aforementioned plug-ins, but is then able to use its auto updating feature to ensure they are always current. Furthermore, the PDF plug-in lives in the full Chrome sandbox and the Flash plug-in is in a weaker sandbox, both in an attempt to limit the damage that could be done by attackers.

In the end, for the majority of users who would probably include the plug-ins anyway, this is a security win. For those few users who would not install both plug-ins this choice adds to the attack surface, which potentially decreases security. This second group of advanced users can manually disable those two plug-ins.

Exploit Mitigations and Manageability

As far as ASLR goes, the results are similar to Firefox, since Firefox plug-ins will load in Google Chrome. Chrome allows loading plug-ins with non-ASLR compatible DLLs, allowing attackers to know the location of executable code segments, hence defeating the exploit mitigation.

There are two ways to review, disable or remove extensions or plug-ins. Extensions can be managed by going to the configuration wrench, then selecting Tools and Extensions.

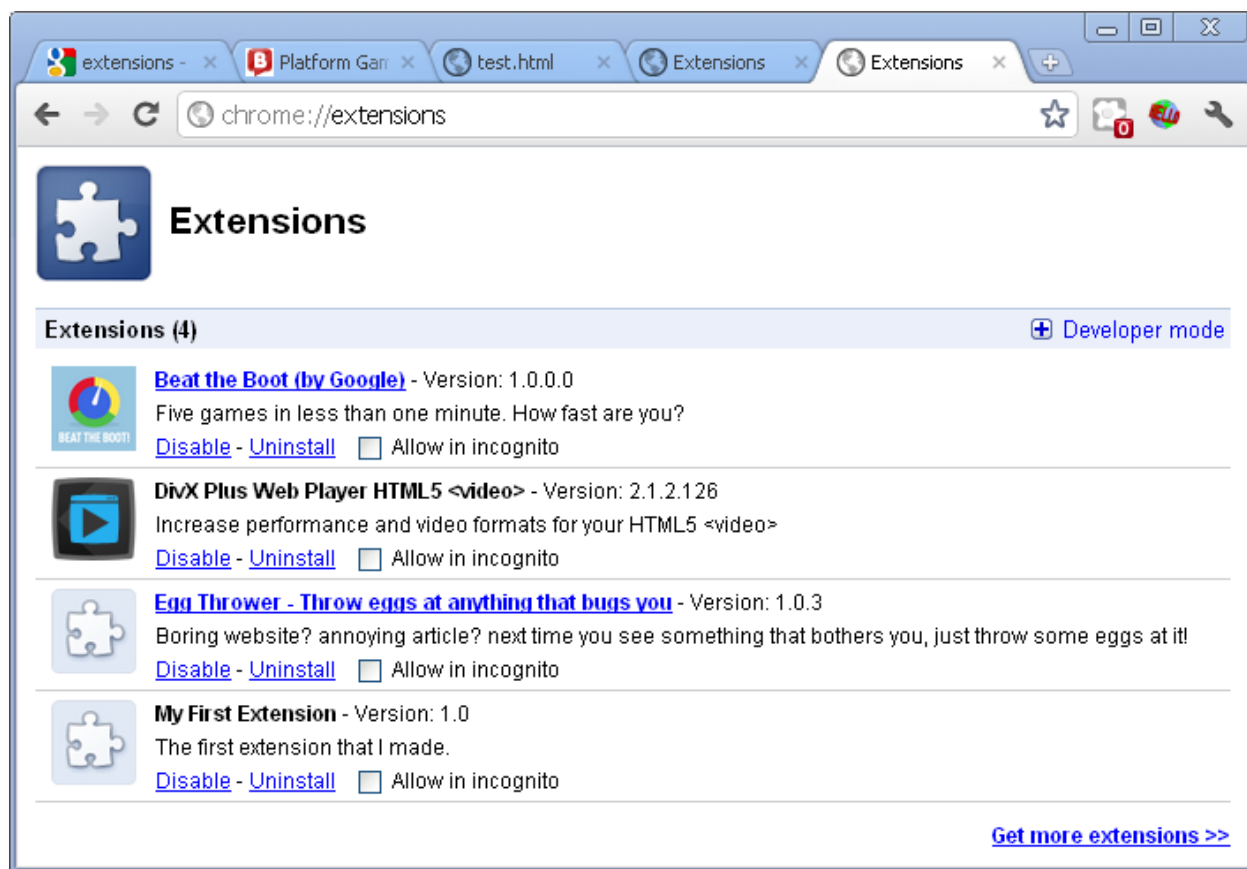


Figure 98. Chrome extensions may be disabled or allowed

There is also a task manager, found under the wrench and Tools menu that indicates what tabs are open, what extensions are running, and what plug-ins are loaded.

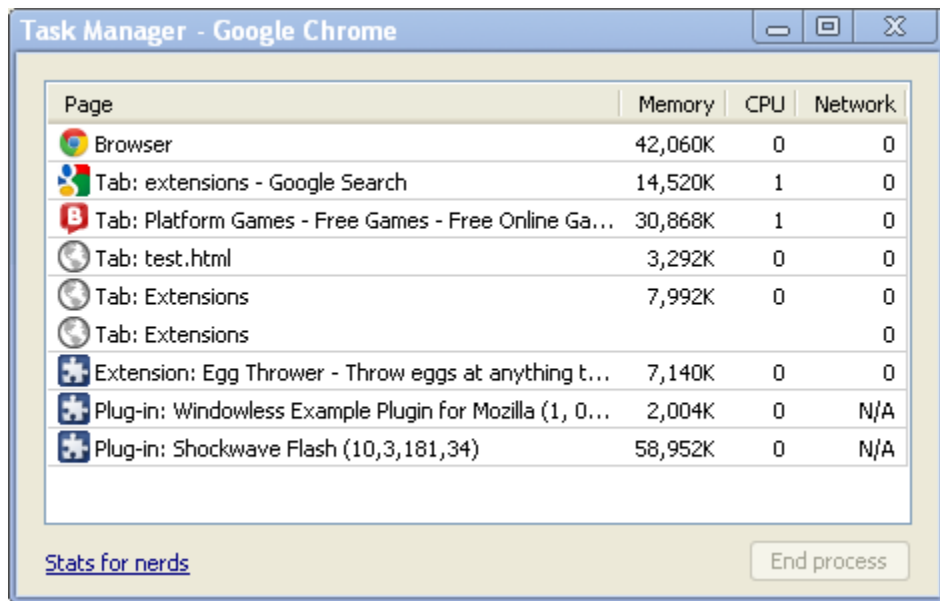


Figure 99. Chrome task manager

The only problem is the lack of installed plug-in listing and configuration content in the management interface and wrench menu. The only way to get the information is to navigate to “chrome://plug-ins” in the address bar. This menu is nice and indicates version information, where DLLs are stored, MIME types, and numerous other content, see Figure 106. However, having this information accessible from the extensions configuration screen would be more intuitive.

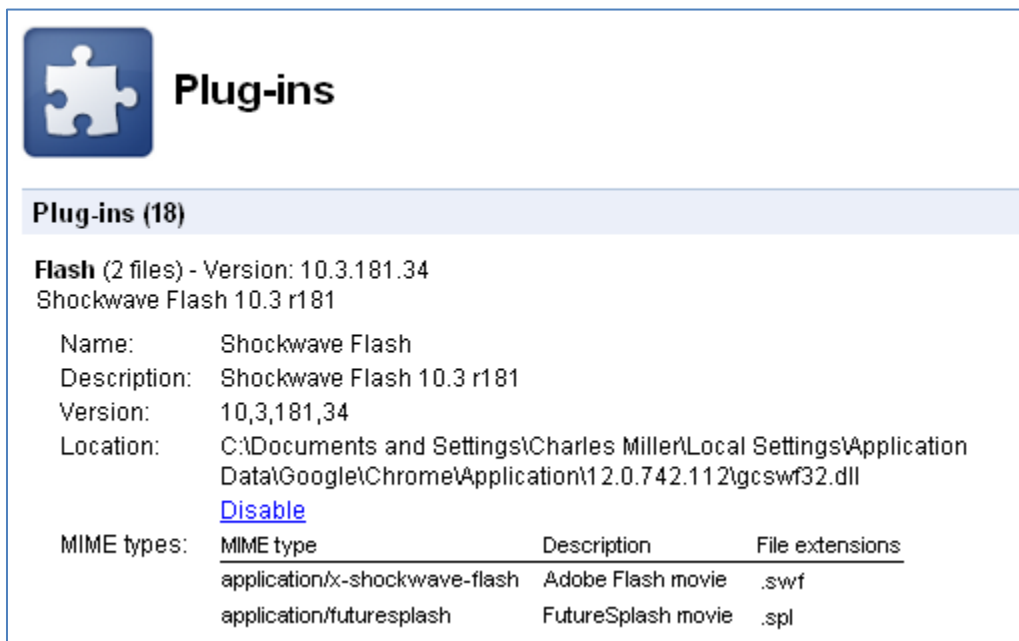


Figure 100. Chrome plug-in configuration menu

Chrome Add-on summary

- Installation requires user interaction
- Extensions must declare in a manifest which API they need to access
- Central location of extensions, more prompts if installed elsewhere
- Auto updates
- Extensions can be remotely uninstalled if malicious
- No sandboxing of plug-ins, except PDF and Flash
- Plug-ins can break ASLR
- Plug-ins can be activated silently (there are some notable exceptions that cannot)
- Plug-ins and extensions are managed separately

Internet Explorer

Installation

Of the browsers assessed, IE has the greatest number of ways to customize and extend its functionality. These fall into two major categories, browser extensions and content extensions. There are many different browser extensions. One is the Shortcut Menu Extension, which permits script or native code to be executed upon a right-click menu choice by the user. Secondly, there are toolbars or Explorer Bars that allow native code to be loaded into the browser to add toolbar functionality.

The final type of browser extension is the Browser Helper Object (BHO). BHOs let native code objects run in the same memory context as the browser and can perform any action on the available windows and modules. For example, a BHO could detect the browser's typical events, such as GoBack, GoForward and DocumentComplete; accessing the browser's menu and toolbar to make changes; creating windows to display additional information on the currently viewed page; and installing hooks to monitor messages and actions.

The other broad category of add-ons is content extensions. These extensions are responsible for adding the ability to parse and display new types of content. An example of a content extension is an Active Document. These extensions run native code when encountering a particular document type. The most well-known type of extension, ActiveX controls, are another example of content extensions. ActiveX controls are native code and have access to the browser internals. Other content extensions include Behaviors, Windows Forms Controls and Pluggable Protocol Handlers.

As you can see, there are numerous ways to extend Internet Explorer. In this section, we'll focus mainly on ActiveX, although most of the findings are valid for other types of extensions as well.

Installation of ActiveX controls requires user input and interaction. If an ActiveX control is not installed, Internet Explorer will display a bar that requires the user to click and select 'Install' to run the ActiveX control.

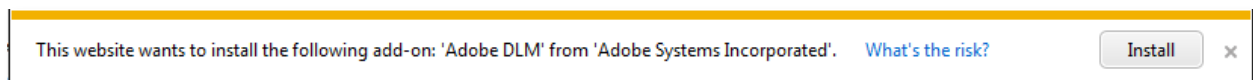


Figure 101. Internet Explorer page requires ActiveX control

After permission is granted, a second dialogue is presented which presents information about the control's digital signature.

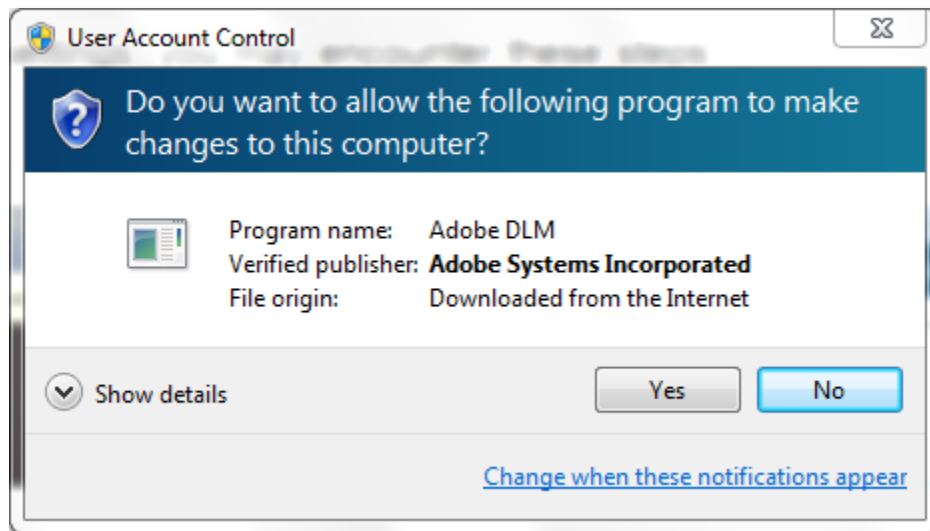


Figure 102. Internet Explorer control requesting installation

If the signature is invalid or missing, Internet Explorer will not continue the installation process.

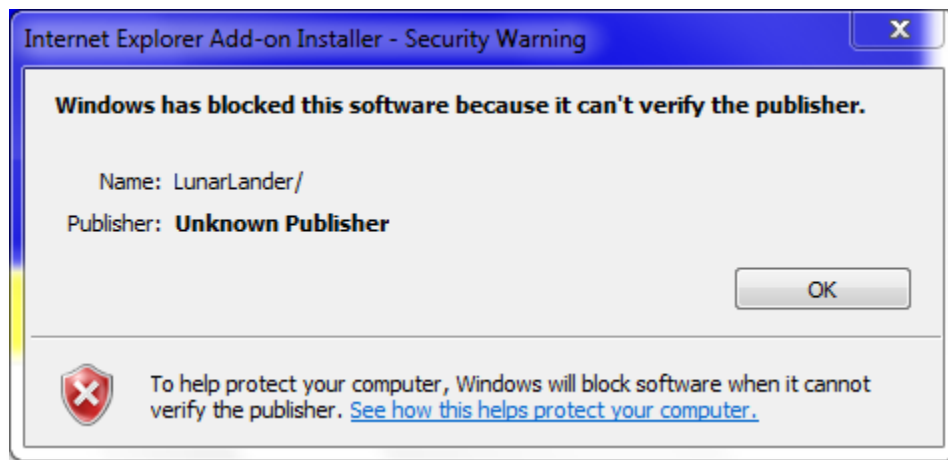


Figure 103. Internet Explorer bad signature

Automatic updates are possible with ActiveX controls. Whenever content that calls upon an ActiveX control is accessed, Internet Explorer checks with the site to see if a newer version is available. If an update is available, it will offer the ability to install the new version.

Sandbox Results

ActiveX controls run within the Internet Explorer process. The control runs in a permissive sandbox. There are some restrictions to what it can do, but not enough to stop the activities of an attacker significantly. The results of the sandbox testing script are detailed below.

File System

We attempted to access certain system directories and files via the sandboxed plug-in.

Note: Permissions may overlap due to generic and specific permission checks. This is done to give an overview accompanied by precise security permissions.

Permission	%SystemDrive%, %SystemRoot%, %ProgramFiles%, %SystemRoot%\System32	%AllUsersProfile%
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	BLOCKED	GRANTED
FILE_ADD_FILE	BLOCKED	GRANTED
FILE_ADD_SUBDIRECTORY	BLOCKED	GRANTED
FILE_APPEND_DATA	BLOCKED	GRANTED
FILE_CREATE_PIPE_INSTANCE	BLOCKED	GRANTED
FILE_DELETE_CHILD	BLOCKED	BLOCKED
FILE_LIST_DIRECTORY	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
FILE_TRAVERSE	GRANTED	GRANTED
FILE_WRITE_ATTRIBUTES	BLOCKED	GRANTED
FILE_WRITE_DATA	BLOCKED	GRANTED
FILE_WRITE_EA	BLOCKED	GRANTED
WRITE_DAC	BLOCKED	BLOCKED

Figure 104. Internet Explorer add-on directory permissions

Permission	%UserProfile%, %Temp%, %AppData%, %AllUsersProfile%	%SystemRoot%\explorer.exe, %SystemRoot%\Cursors\arrow_i.cur
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	GRANTED	BLOCKED
GENERIC_EXECUTE	GRANTED	GRANTED
FILE_EXECUTE	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
STANDARD_RIGHTS_EXECUTE	GRANTED	GRANTED
SYNCHRONIZE	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
STANDARD_RIGHTS_READ	GRANTED	GRANTED
FILE_APPEND_DATA	GRANTED	BLOCKED
FILE_WRITE_ATTRIBUTES	GRANTED	BLOCKED
FILE_WRITE_DATA	GRANTED	BLOCKED
FILE_WRITE_EA	GRANTED	BLOCKED
STANDARD_RIGHTS_WRITE	GRANTED	GRANTED
WRITE_DAC	GRANTED	BLOCKED

Figure 105. Internet Explorer add-on file permissions

Registry

A select few registry hives/keys were accessed from inside the add-on. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	GRANTED

Figure 106. Internet Explorer add-on registry permissions

Network Access

The ability for a browser to access the internet is vital to its operation, but creating a new socket for reading, writing, listening could permit an attacker to communicate read-able information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	GRANTED
Send()	GRANTED
Recv()	GRANTED
Listen()	GRANTED

Figure 107. Internet Explorer add-on network accessibility

Resource Monitoring

Recording keystrokes, registering hotkeys and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in their attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire user information (obviously, there are various other techniques).

Action	Result
GetPixel ()	GRANTED
RegisterHotKey ()	GRANTED
GetAsyncKeyState ()	BLOCKED

Figure 108. Internet Explorer add-on resource monitoring

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization as the compromised application. This could permit a plethora of other options that could be used for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with "C:\Program Files\Internet Explorer\iexplore.exe" was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	GRANTED

Figure 109. Internet Explorer add-on CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them gain privileges.

For more granular information on handling test cases, please see 'ie-addons.txt' in the attachment.

Windows Clipboard

The Windows clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read potentially sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to *GET* and *SET* information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	BLOCKED
SetClipboardData (CF_TEXT)	BLOCKED

Figure 110. Internet Explorer add-on Clipboard access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	GRANTED
OpenWindowsStation ("winsta0")	GRANTED
OpenDesktop ("Default")	GRANTED

Figure 111. Internet Explorer add-on Desktop/WindowStation permissions

System Parameters

It should be obvious that an attacker could use the system wide parameters to his advantage. These parameters can control screen savers, menus and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

***Note:** Only a single system parameter was checked for brevity's sake.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	BLOCKED

Figure 112. Internet Explorer add-on SystemParametersInfo()

Windows Message Broadcasts

By sending a Windows message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could interpret the broadcast message differently; due to expecting a varying number of parameters [MSDN_Broad]. This could cause many operating system instability and exploitation scenarios. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_TIMER)	GRANTED

Figure 113. Internet Explorer add-on sends broadcast message

Windows Hooks

Windows hooks are a procedure used to monitor certain types of system events on the same desktop as the calling thread [Microsoft_Hooks]. These same hooks historically have been used by malware to do such things as monitor keyboard input and other nefarious tasks. We have checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 114. Internet Explorer add-on set Windows hooks

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows operating system [Microsoft_Pipes]. While this ability to communicate between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We attempted to enumerate all the named pipes for a system for permissions testing. If that were not possible, we would iterate through a list of 'well-known' pipes for the Windows 7 (32-bit) operating system attempting to validate permissions.

Named Pipe	PIPE_ACCESS_INBOUND	PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	GRANTED	GRANTED
\\.\pipe\ntsvcs	GRANTED	BLOCKED
\\.\pipe\scerpc	GRANTED	BLOCKED
\\.\pipe\protected_storage	GRANTED	GRANTED
\\.\pipe\plugplay	GRANTED	BLOCKED
\\.\pipe\epmapper	GRANTED	BLOCKED
\\.\pipe\eventlog	GRANTED	BLOCKED
\\.\pipe\atsvc	GRANTED	BLOCKED
\\.\pipe\wkssvc	GRANTED	GRANTED
\\.\pipe\keysvc	GRANTED	BLOCKED
\\.\pipe\trkwks	GRANTED	BLOCKED
\\.\pipe\srvsvc	GRANTED	GRANTED

Figure 115. Internet Explorer add-on named pipe access

Exploitation Mitigations and Manageability

ActiveX controls do not have to have ASLR enabled. Therefore, they can weaken the exploit mitigation strategies used by the browser. However, by default Visual Studio builds ActiveX controls with ASLR enabled. Even with this, some common plug-ins ship with non-ASLR compatible DLLs.

Examples include DLLs from the following plug-ins:

- Java
- DivX

Web pages can silently load and use ActiveX controls that are already installed in Internet Explorer. Microsoft does offer ActiveX filtering, which can be found under the Tools->Safety menu, as shown below:

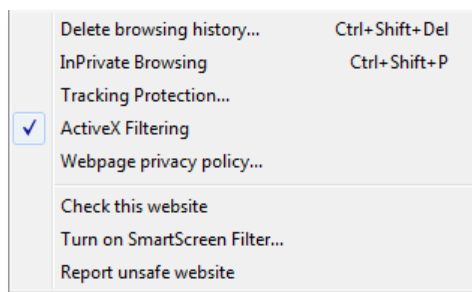


Figure 116. Internet Explorer enable ActiveX filtering

This feature is NOT on by default. ActiveX filtering blocks controls from being launched by sites unless explicitly given permission to do so.

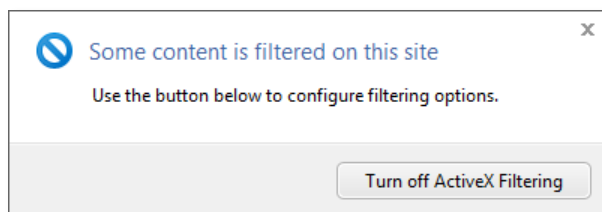


Figure 117. Internet Explorer ActiveX filtering permissions dialog

Luckily, there exists a central location to manage all add-ons for Internet Explorer; residing in Tools->Manage add-ons, see Figure 124. This interface provides all the functionality needed to manage add-ons for Internet Explorer.

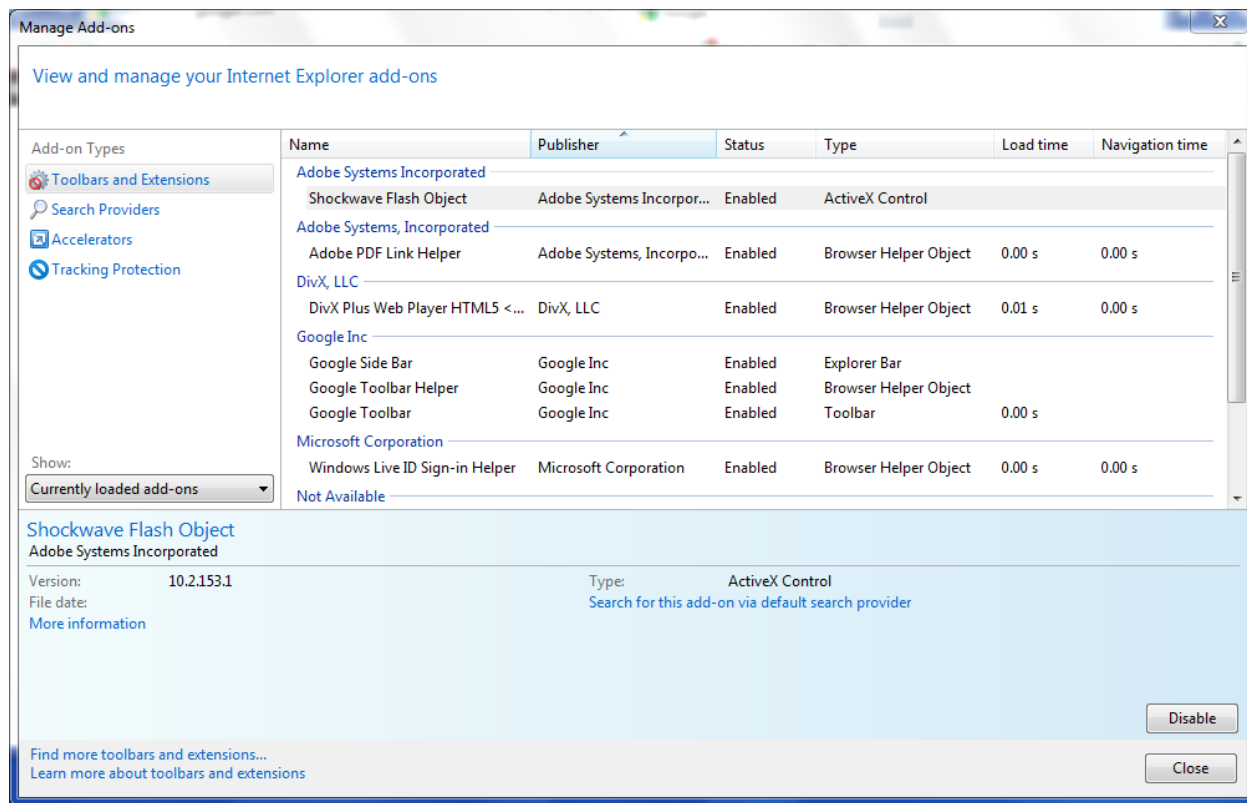


Figure 118. Internet Explorer add-ons

Internet Explorer Add-On Summary

- Installation requires user interaction
- No review of add-ons
- Auto updating available
- No sandboxing
- Can break ASLR
- Plug-ins can be activated silently
- Good manageability

Firefox

Installation

Firefox has a few ways to customize or extend the functionality of the browser. These include personas, themes, extensions and plug-ins. Personas are simply images that the browser displays; which present no apparent security implications. Themes, sometimes called skins, consist of a .jar file containing a collection of .css, .xml and image files. Again, these have no dynamic content, so are relatively benign. Things start to get a little more interesting with extensions.

Firefox extensions come in an .xpi file. An xpi file is really a zip file containing files of the following types: css, xml, image files, JavaScript and other types. Extensions can also include XPCOM typelibs (type libraries), which are binary interface description files. Extensions are intended to extend existing functionality in the browser. Firefox extensions are installed entirely within the browser. Normally, they present a dialogue to the user before installation and make the user wait five seconds before accepting the risk, see Figure 125. Note that if you install an extension from the “Search” screen, it does not ask for confirmation.

Firefox extensions are able to run arbitrary JavaScript code, but do not include native code. From a security perspective, in the worst case, it is like enabling a cross-site scripting vulnerability on every page you visit. That is, the attacker could change arbitrary HTML, read cookies, see form data, etc.

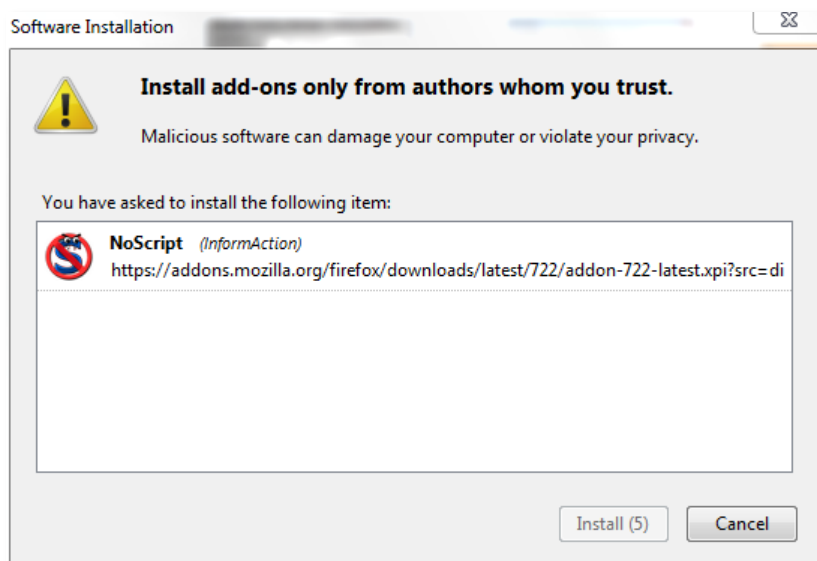


Figure 119. Firefox installing an extension

The final way to extend Firefox is through plug-ins. While extensions extend existing browser functionality, plug-ins introduces completely new behavior and is the mechanism used to extend the browser using native code. This has the highest potential security risk. Plug-ins can potentially access any files on the system, install software and make network connections. They can perform or install any of the things done by traditional malware. Due to the insecurities inherent in compiled code, plug-ins could introduce new security vulnerabilities, including new memory corruption problems, to the

browser. In addition, since they consist of entirely new code, they must include at least one library (DLL) to be installed either manually or by way of an installer outside of Firefox, see Figure 126. Firefox plug-ins run in a separate process from the main browser called **plug-in-container.exe**.

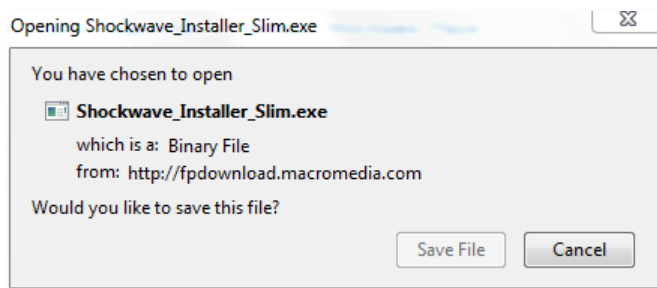


Figure 120. Firefox installing plug-ins resemble software installations

Installation of extensions and plug-ins normally comes from the official Mozilla site, addons.mozilla.org. If the extension is installed through the Mozilla site, updates will be pushed down to the browser when available. The browser, by default, is set to “Update Add-ons Automatically”, see Figure 127.

In order for add-ons to be hosted on addons.mozilla.org, they must pass a review process. For extensions not reviewed or hosted on addons.mozilla.org, the installation dialogue adds “Author not verified,” see Figure 128. There are still ways to allow automatic updating even if the add-on is not hosted on the Mozilla site. Once installed, a plug-in can be activated silently by web pages visited.

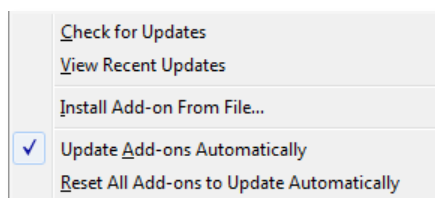


Figure 121. Firefox automatic add-on updates

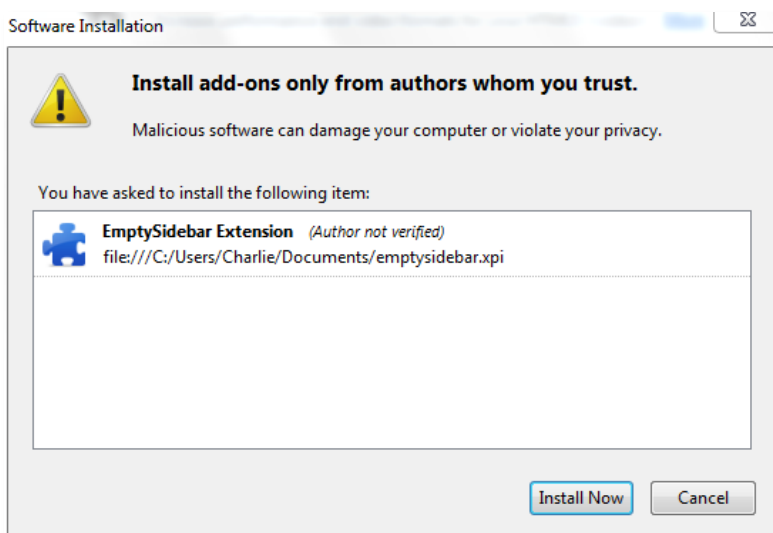


Figure 122. Firefox custom extension (Author not verified)

Sandbox Results

The results of running the sandbox test harness showed the script could perform whatever actions it wished to perform at the permissions the browser had, which is running with medium integrity level by default (Windows 7 32-bit).

File System

We attempted to access certain system directories and files via the sandboxed add-on.

Note: Permissions may overlap due to generic and specific permission checks. This is done to give an overview accompanied by precise security permissions.

Permission	%SystemDrive%, %SystemRoot%, %ProgramFiles%, %SystemRoot%\System32	%AllUsersProfile%
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	BLOCKED	GRANTED
FILE_ADD_FILE	BLOCKED	GRANTED
FILE_ADD_SUBDIRECTORY	BLOCKED	GRANTED
FILE_APPEND_DATA	BLOCKED	GRANTED
FILE_CREATE_PIPE_INSTANCE	BLOCKED	GRANTED
FILE_DELETE_CHILD	BLOCKED	BLOCKED
FILE_LIST_DIRECTORY	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
FILE_TRAVERSE	GRANTED	GRANTED
FILE_WRITE_ATTRIBUTES	BLOCKED	GRANTED
FILE_WRITE_DATA	BLOCKED	GRANTED
FILE_WRITE_EA	BLOCKED	GRANTED
WRITE_DAC	BLOCKED	BLOCKED

Figure 123. Firefox add-on directory permissions

Permission	%UserProfile%, %Temp%, %AppData%	%SystemRoot%\explorer.exe, %SystemRoot%\Cursors\arrow_i.cur
ZERO	GRANTED	GRANTED
GENERIC_READ	GRANTED	GRANTED
GENERIC_WRITE	GRANTED	BLOCKED
GENERIC_EXECUTE	GRANTED	GRANTED
FILE_EXECUTE	GRANTED	GRANTED
FILE_READ_ATTRIBUTES	GRANTED	GRANTED
STANDARD_RIGHTS_EXECUTE	GRANTED	GRANTED
SYNCHRONIZE	GRANTED	GRANTED

FILE_READ_DATA	GRANTED	GRANTED
FILE_READ_EA	GRANTED	GRANTED
STANDARD_RIGHTS_READ	GRANTED	GRANTED
FILE_APPEND_DATA	GRANTED	BLOCKED
FILE_WRITE_ATTRIBUTES	GRANTED	BLOCKED
FILE_WRITE_DATA	GRANTED	BLOCKED
FILE_WRITE_EA	GRANTED	BLOCKED
STANDARD_RIGHTS_WRITE	GRANTED	GRANTED
WRITE_DAC	GRANTED	BLOCKED

Figure 124. Firefox add-on directory/file permissions

Registry

A select few registry hives/keys were accessed from inside the add-on. These hives and keys represent locations that would be of interest to malware authors in an attempt to gain persistence.

Hive	Subkey	Permission	Result
HKEY_LOCAL_MACHINE	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_CURRENT_USER	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_USERS	NULL	MAXIMUM_ALLOWED	GRANTED
HKEY_LOCAL_MACHINE	Software\Microsoft\Windows NT\CurrentVersion\WinLogon	MAXIMUM_ALLOWED	GRANTED

Figure 125. Firefox add-on registry permissions

Network Access

The ability for a browser to access the Internet is vital to its operation, but creating a new socket for reading, writing and listening could permit an attacker to communicate read-able information to the outside world. The ability to initiate, read, write and listen on Windows sockets are listed below.

Action	Result
WSAStartup	GRANTED
Send ()	GRANTED
Recv ()	GRANTED
Listen ()	GRANTED

Figure 126. Firefox add-on network accessibility

Resource Monitoring

Recording keystrokes, registering hotkeys and attempting to read screen data (i.e. screen captures) are widely employed amongst attackers and spyware authors in their attempts to intercept and read user's confidential information. The sandbox test harness has three checks for methods that attempt to acquire user information (obviously, there are various other techniques).

Action	Result
GetPixel ()	GRANTED
RegisterHotKey ()	GRANTED
GetAsyncKeyState ()	GRANTED

Figure 127. Firefox add-on resource monitoring

Process Creation

An attacker might find it valuable to create a new process, even if that process has the same authorization and privilege level as the compromised application. This could permit a plethora of other opportunities that could be used for privilege escalation or data leakage. A simple example of calling the CreateProcess() API with “C:\Program Files\Internet Explorer\iexplore.exe” was used.

Executable	Permission Granted
C:\Program Files\Internet Explorer\iexplore.exe	GRANTED

Figure 128. Firefox add-on CreateProcess()

Note: We are aware that this is in a system directory and only presents one example, but deemed it appropriate for the limitations of this assessment.

Handles

Handles are used by the Windows operating system to keep track of content-specific identifiers. This permits applications to reference resources by handle, instead of, for example, process ID. Since handles are used to access resources, they must also contain security restrictions so that other applications, specifically those from the sandbox, may not use them gain privileges.

For more granular information on handling test cases, please see ‘ff-addons.txt’.

Windows Clipboard

The Windows clipboard enables different applications to share messages and data [Microsoft_Clip]. Not only could a compromised application read sensitive information from the clipboard, the attacker could also use flaws in the clipboard to gain further system access (i.e. sandbox escape) [Clip_Exploit]. During our tests, we attempted to GET and SET information to the clipboard.

Action	Permission Granted
GetClipboardData (CF_TEXT)	GRANTED
SetClipboardData (CF_TEXT)	GRANTED

Figure 129. Firefox add-on Clipboard access

Windows Desktop

The Windows desktop not only provides a display surface for user interaction, but also contains objects such as windows, menus and hooks (it is also a securable object). Windows messages are limited to communicating with other processes that reside on the same desktop; inter-desktop process communication is not operational [Microsoft_Desktop]. The ability to create, switch and open other desktops with varying permissions may also lead to privilege escalation scenarios [CVE-2009-1123].

Action	Permission Granted
CreateDesktop ()	GRANTED
OpenWindowsStation (“winsta0”)	GRANTED
OpenDesktop (“Default”)	GRANTED

Figure 130. Firefox add-on Desktop/WindowStation permissions

System Parameters

It should be obvious that an attacker could use the system wide parameters to his advantage. These parameters can control screen savers, menus and many other options [Microsoft_SysParam]. By limiting the ability to set these parameters, the sandbox can ensure that no underhandedness can be achieved by someone attempting to escape the sandboxed environment.

Action	Permission Granted
SystemParametersInfo (SPI_GETMOUSE) [GET]	GRANTED
SystemParametersInfo (SPI_SETMOUSE) [SET]	GRANTED

Figure 131. Firefox add-on SystemParametersInfo()

Note: Only a single system parameter was checked for brevity's sake.

Windows Message Broadcasts

By sending a Windows message with the 'HWND_BROADCAST' option set, an application effectively sends the same message to every top-level window. Each of these windows could interpret the broadcast message differently; due to expecting a varying number of parameters [MSDN_Broad]. This could cause many operating system instability and exploitation scenarios. We sent an example broadcast message to determine if it was permitted from within the sandbox. A great example of exploiting the Windows messaging system for authoritative gain would be a *shatter attack* [Wiki_Shatter].

Action	Permission Granted
SendMessage (HWND_BROADCAST, WM_TIMER)	GRANTED

Figure 132. Firefox add-on send broadcast message

Windows Hooks

Windows hooks are a procedure used to monitor certain types of system events on the same desktop as the calling thread [Microsoft_Hooks]. These same hooks historically have been used by malware to do such things as monitor keyboard input and other nefarious tasks. We have checked the ability to set system hooks.

Action	Permission Granted
SetWindowsHookEx (WH_KEYBOARD)	GRANTED

Figure 133. Firefox add-on set Windows hooks

Named Pipes

Named pipes are used for one-way or two-way communications within the Windows Operating System [Microsoft_Pipes]. While this ability to communication between client and server is an integral part of inter process communication, unbridled communications can be used to bypass sandbox protection mechanisms. For example, imagine an attacker has the ability to send data to a named pipe, which has a privilege and authorization level greater than the process that is sending data. We have attempted to enumerate all the named pipes for a system for permissions testing. If that was not possible, we iterate through a list of ‘well-known’ pipes for the Windows 7 (32-bit) operating system in an attempt to validate permissions.

Named Pipe	PIPE_ACCESS_INBOUND PIPE_ACCESS_OUTBOUND
\\.\pipe\lsass	GRANTED
\\.\pipe\ntsvcs	GRANTED
\\.\pipe\scerpc	GRANTED
\\.\pipe\protected_storage	GRANTED
\\.\pipe\plugplay	GRANTED
\\.\pipe\epmapper	GRANTED
\\.\pipe\eventlog	GRANTED
\\.\pipe\atsvc	GRANTED
\\.\pipe\wkssvc	GRANTED
\\.\pipe\keysvc	GRANTED
\\.\pipe\trkwks	GRANTED
\\.\pipe\srvsvc	GRANTED

Figure 134. Firefox add-on named pipe access

Exploit mitigations and manageability

Plug-ins have the potential for circumventing exploit mitigation technologies. Since plug-ins are DLLs, it is possible they are not compatible with ASLR, which will make this mitigation ineffective, since any non-randomized code segments can be utilized by an attacker. In Firefox’s case, it is possible for non-ASLR compatible plug-ins to exist. Furthermore, some common plug-ins ship with non-ASLR compatible DLLs.

Examples include DLLs from the following plug-ins:

- Java
- DivX
- Windows Media Player

Firefox provides an easy interface to manage add-ons in the Tools->Add-ons menu, see Figure 141. It allows the user to see all installed add-ons, disable or remove any extension, and disable any plug-in. It also allows the user to manually check for add-on updates, as well as view recent updates to any add-ons, see Figure 141. Finally, it allows the user to configure the extensions, when applicable.

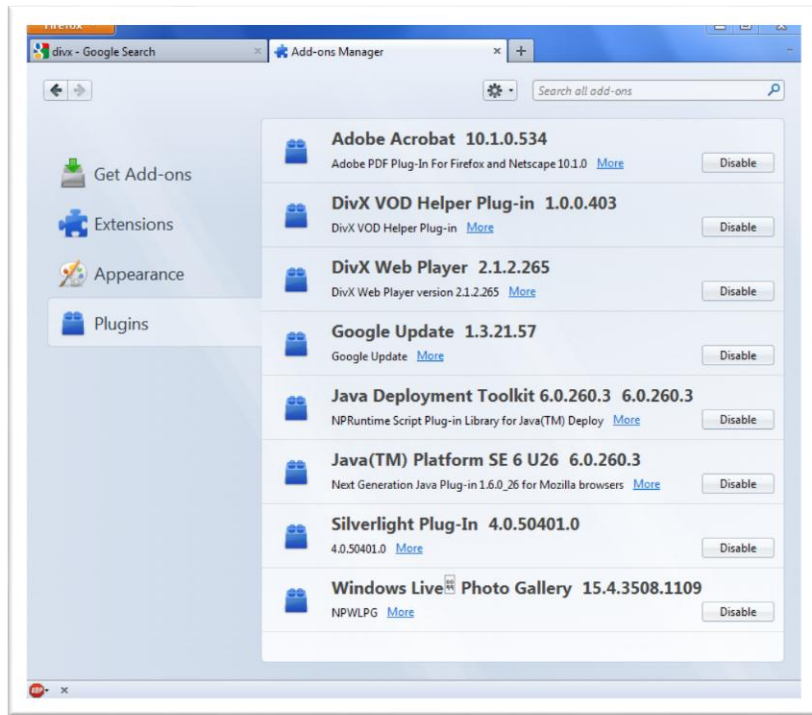


Figure 135. Firefox managing plug-ins

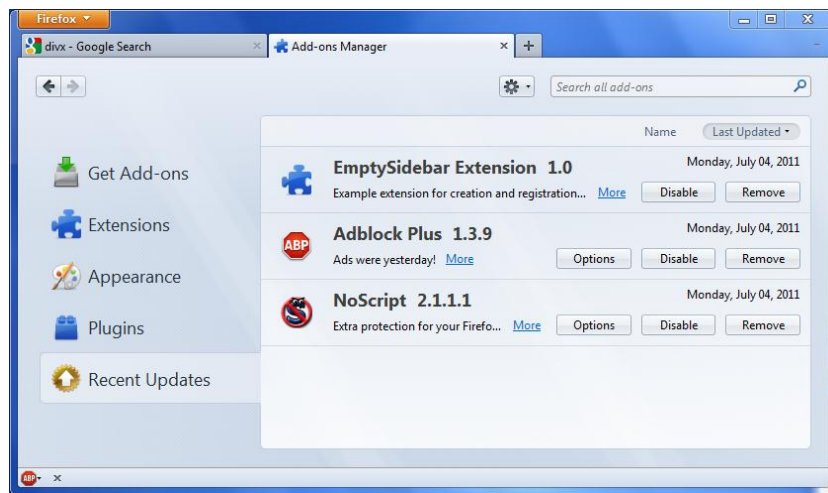


Figure 136. Firefox recent add-on updates

Firefox Add-on summary

- Installation requires user interaction
- Add-ons hosted at addon.mozilla.org are reviewed
- Auto update
- No sandboxing
- Can break ASLR
- Plug-ins can be activated silently
- Good manageability

Add-on summary

Chrome	IE	Firefox
<ul style="list-style-type: none"> • Installation requires user interaction • Extensions can be remotely removed • Extensions restricted to which API they require and are approved to use • Central location of extensions encouraged • Auto updates • No sandboxing, except PDF and Flash • Can break ASLR • Plug-ins can be activated silently, except some specifically • Plug-ins and extensions are managed separately. 	<ul style="list-style-type: none"> • Installation requires user interaction • No review of add-ons • Auto updating available • Limited sandboxing • Can break ASLR • Plug-ins can be activated silently • Good manageability 	<ul style="list-style-type: none"> • Installation requires user interaction • Add-ons hosted at addon.mozilla.org are reviewed • Auto update • No sandboxing • Can break ASLR • Plug-ins can be activated silently • Good manageability

Figure 137. Firefox add-on overview

None of the browsers allows non-reviewed add-ons to be silently installed. All of the browsers have some methods that allow add-ons updates. With the exception of Flash and PDF on Chrome, none of the browsers has significant sandboxing of add-ons or requires them to be ASLR compatible. With a few exceptions, add-ons can be activated silently by web pages on all browsers. Chrome’s management interface requires some extra work, but all the browsers have reasonable management interfaces.

Chrome made a couple of choices concerning specific plug-ins that adds to the overall security of the browser. Plug-ins such as Java cannot be used without user interaction. Others, like their PDF viewer or Flash, are included with the browser; they are able to ensure they are up to date with security patches and provide a limited sandbox.

Chrome is the only browser that supports a security model that restricts extension’s API use. Extensions must declare which API they plan to use, the user is shown this information and must approve it. Also, Chrome allows for the remote uninstallation of malicious extensions.

Conclusions

Browsers have become complicated, multi-content, multi-purpose applications in recent times. They are no longer primitive HTML parsers. New functionality has brought on new security concerns, which are currently being addressed by the iterative process of updates to each browser's codebase, as well as deterrent technologies such as exploit mitigation and URL blacklisting.

We have shown that there are various methods for collecting vulnerability statistics, many of which are plagued by abstractions such as multi-vulnerability CVEs, silent fixes and inconsistent data. While tracking vulnerabilities is useful for vendors and end users as a means to apply security fixes in a timely manner, the data currently available lacks sufficient consistency to be truly useful for evaluating browser security. There will always be browser vulnerabilities, but browser appraisal must be derived from metrics that can be accurately correlated. A move toward greater transparency in the security update process would likely benefit consumers, and create a level playing field if metrics such as vulnerability severity and timelines from discovery to release date of security fixes were openly disclosed. Were these timelines widely available, the data might open the floor for an unambiguous debate about each browser project's true response time.

At the time of this writing, sandboxes are quickly becoming standard best practice within many popular applications. They prevent certain applications and process from accessing functionality deemed inappropriate. We have shown that Google Chrome provided the most restrictive sandbox, limiting almost all interaction with the operating system to the *broker* process; leaving the *rendering* process to handle strictly rendering. Internet Explorer has made valiant first steps at a sandbox by using the *low integrity* functionality to restrict IE tabs' persistence abilities on the system in the event of a compromise. Unfortunately, the *low integrity* mechanism permits read access to most resources, unrestricted network accessibility and a multitude of ways to alter the system state. Lastly, Firefox has yet to implement any formal sandbox, relying solely upon the process running as *medium integrity*; giving it the ability to perform any action of a non-administrator.

Add-ons and plug-ins provide manufacturers and third parties with the ability to develop additional functionality for the browser. Additional functionality also brings on security risk, which can undermine overall browser security posture. While each browser has the ability to accurately manage plug-ins, auto-update of add-ons still require user interaction before installation, and Google Chrome is the only browser to partially sandbox any of its plug-ins. That said, all three of the major browsers provided exploit writers with a number of ways to circumvent exploit mitigation technologies, such as ASLR, by neglecting to ensure all add-ons adhere to the appropriate policies. Add-ons provide integral functionality to a browser, and will likely be commonplace for years to come, but strong default security restrictions will need to be put in place to ensure that these add-ons don't undermine the overall browser security model.

In the case of URL blacklisting services, gathering a realtime, comprehensive picture of all live malware propagated via websites on any given hour, day, or week may be Sisyphian task. This is not to say URL blacklisting services aren't useful, and providing these services is valuable as part of an overall browser security approach, but it's fairly clear it's neither an ironclad defense nor a particularly valuable criterion

of overall browser security posture. As with vulnerability statistics, it's likely that more transparency would benefit consumers and the community as a whole. If URS integrated the SBL, and vice versa, average users would benefit greatly, and the net benefit of that collaboration would be a safer userbase for both browsers, and a benefit to the Internet at large, which presumably is the stated intent of services of this type in the first place. We have shown that browsers tend to detect a majority of the items in their sample set, yet cross-sample detection varies quite drastically. We feel the only way to get the best results would be for all browser manufacturers to share their sample data.

In conclusion, overall browser security needs to be considered when attempting to compare browsers from a security standpoint. Drawing conclusions based solely on one category of protection, such as blacklisted URL statistics, doesn't give a valid perspective on which browser is most secure. Instead, they should be considered in the context of other mechanisms such as anti-exploitation technologies and malicious plug-in protection, which play a more important role in protecting end users from exploits and persistent malware. By these measures, we believe Google Chrome to be the web browser that is most secured against attack.

Bibliography

- **Autoupdate** – *Why Silent Updates Boost Security*
(<http://www.techzoom.net/publications/silent-updates/>)
- **Autoupdate_No** – *Welcome to the era of auto update*
(<http://www.zdnet.com.au/welcome-to-the-era-of-auto-update-339308123.htm>)
- **Blackhat_Smith09** – *The Language of Trust: Exploiting Trust Relationships in Active Content*
(<https://www.blackhat.com/html/bh-usa-09/bh-usa-09-speakers.html#Dowd>)
- **Chrome_AutoUpdate** – *Autoupdating*
(<http://code.google.com/chrome/extensions/autoupdate.html>)
- **Chrome_Blocked** – *Blocked plug-ins*
(<http://www.google.com/support/chrome/bin/answer.py?hl=en&answer=1247383>)
- **Chrome_ExtOverview**—*Google Chrome Extension Overview*
(<http://code.google.com/chrome/extensions/overview.html>)
- **Chrome_FAQ** – *Chrome Extension FAQ*
(<http://code.google.com/chrome/extensions/faq.html>)
- **Chrome_NPAPI** – *NPAPI Plug-ins*
(<http://code.google.com/chrome/extensions/npapi.html>)
- **Chrome_OutofDate** – *Out of date plug-ins*
(<http://www.google.com/support/chrome/bin/answer.py?answer=1181003>)
- **Chrome_PermWarn** – *Permission Warning*
(https://code.google.com/chrome/extensions/permission_warnings.html)
- **Chrome_Sandbox** – *Sandbox – The Chromium Project*
(<http://www.chromium.org/developers/design-documents/sandbox>)
- **Chromium_Finish** – *Extensions: One Step Closer to the Finish Line*
(<http://blog.chromium.org/2009/11/extensions-one-step-closer-to-finish.html>)
- **Chromium_Release** – *Release Notes*
(<http://dev.chromium.org/getting-involved/dev-channel/release-notes?offset=0>)
- **Clip_Exploit** – *Zero-Day Kernel Flaw Linked to Clipboard*
(<http://mcpmag.com/articles/2010/08/09/zero-day-windows-kernel-flaw-linked-to-clipboard.aspx>)
- **CVE-2009-1123**—*Microsoft Windows 'win32k.sys' Local Privilege Escalation Vulnerability*
(<http://www.securityfocus.com/bid/35121/info>)
- **Immunity_Exploitability_Index** -- *A Bounds Check on the Microsoft Exploitability Index*
(<http://download.microsoft.com/download/3/E/B/3EBDB81C-DF2F-470B-8A64-981DC8D9265C/A%20Bounds%20Check%20on%20the%20Microsoft%20Exploitability%20Index%20-%20final.pdf>)
- **Microsoft_BHO**—*Browser Help Objects: The Browser the Way You Want It*
(<http://msdn.microsoft.com/en-us/library/ms976373.aspx>)
- **Microsoft_Clip** – *About the Clipboard*
(<http://msdn.microsoft.com/en-us/library/ms649012%28v=vs.85%29.aspx>)

- **Microsoft_DEP** – DEP on Vista exposed!
(http://blogs.technet.com/b/robert_hensing/archive/2007/04/04/dep-on-vista-explained.aspx)
- **Microsoft_Desktop** – Desktops
(<http://msdn.microsoft.com/en-us/library/ms682573%28v=vs.85%29.aspx>)
- **Microsoft_GS** – GS cookie protection – effectiveness and limitations
(<http://blogs.technet.com/b/srd/archive/2009/03/16/gs-cookie-protection-effectiveness-and-limitations.aspx>)
- **Microsoft_Hooks** – SetWindowsHookEx function
(<http://msdn.microsoft.com/en-us/library/ms644990%28v=vs.85%29.aspx>)
- **Microsoft_IEArch** – Internet Explorer Architecture
(<http://msdn.microsoft.com/en-us/library/aa741312%28v=vs.85%29.aspx>)
- **Microsoft_Pipes** – Named Pipes
(<http://msdn.microsoft.com/en-us/library/aa365590%28v=VS.85%29.aspx>)
- **Microsoft_ExtSecurity** – Verified Security for Browser Extensions
(https://code.google.com/chrome/extensions/permission_warnings.html)
- **Microsoft_SEHOP** – Preventing the Exploitation of Structured Exception Handler (SHE) Overwrites with SEHOP
(<http://blogs.technet.com/b/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwrites-with-sehop.aspx>)
- **Microsoft_SEHOP_KB** -- How to enable Structured Exception Handling Overwrite Protection (SEHOP) in Windows operating systems
(<http://support.microsoft.com/kb/956607>)
- **Microsoft_SPDEP** – SetProcessDEPPolicy Function
(<http://msdn.microsoft.com/en-us/library/bb736299%28v=vs.85%29.aspx>)
- **Microsoft_SWH** – SetWindowsHookEx()
(<http://msdn.microsoft.com/en-us/library/ms644990%28VS.85%29.aspx>)
- **Microsoft_SysParam** – SystemParametersInfo Function
(<http://msdn.microsoft.com/en-us/library/ms724947%28v=VS.85%29.aspx>)
- **MSDN_Broad** – Remember what happens when you broadcast a message
(<http://blogs.msdn.com/b/oldnewthing/archive/2006/06/12/628193.aspx>)
- **MSDN_LCIE** – IE8 and Loosely-Coupled IE (LCIE)
(<http://blogs.msdn.com/b/ie/archive/2008/03/11/ie8-and-loosely-coupled-ie-lcie.aspx>)
- **MSDN_MS08001** – The case of the IGMP Critical
(<http://blogs.technet.com/b/srd/archive/2008/01/08/ms08-001-part-3-the-case-of-the-igmp-network-critical.aspx>)
- **MSDN_SilentPatches** – Additional Fixes in Microsoft Security Bulletin
(<http://blogs.technet.com/b/srd/archive/2011/02/14/additional-fixes-in-microsoft-security-bulletins.aspx>)
- **Mozilla_Add-on** – Add-on (Mozilla)
(http://en.wikipedia.org/wiki/Add-on_%28Mozilla%29)

- **Mozilla_Crashes_Evidence** -- *Mozilla Foundation Security Advisory 2008-15*
(<http://www.mozilla.org/security/announce/2008/mfsa2008-15.html>)
- **Mozilla_Extensions** – *Extensions*
(<https://developer.mozilla.org/en/Extensions>)
- **Mozilla_Policies** – *Add-on Policies*
(<https://addons.mozilla.org/en-US/developers/docs/policies>)
- **Mozilla_Wraps** – *Mozilla takes wraps off Firefox 1.5*
(<http://www.zdnet.com/news/mozilla-takes-wraps-off-firefox-15/145817>)
- **Uninformed_DEP** – *Bypassing Windows Hardware-enforced Data Execution Prevention*
(<http://www.uninformed.org/?v=2&a=4>)
- **W3_Schools_Market_Penetration** – *Browser Statistics*
(http://www.w3schools.com/browsers/browsers_stats.asp)
- **Wiki_Aurora** – *Operation Aurora*
(http://en.wikipedia.org/wiki/Operation_Aurora)
- **Wiki_Shatter** – *Shatter Attack*
(http://en.wikipedia.org/wiki/Shatter_attack)

Appendix A – Chrome Frame

Overview

Web pages began with static content delivered with very little structure. Over time, this basic structure has given way to incredibly complex, interactive and disparate structures. Many technologies that were devised to make the web more engaging have been conceived, flourished and disappeared within the relatively short time that the web has been around. Standards bodies have drafted large documents to detail the intricacies of the structure with the hope of being unambiguous; however, the results have proven less than perfect. A single web page rendered in both Microsoft Internet Explorer and Google Chrome can appear wildly different.

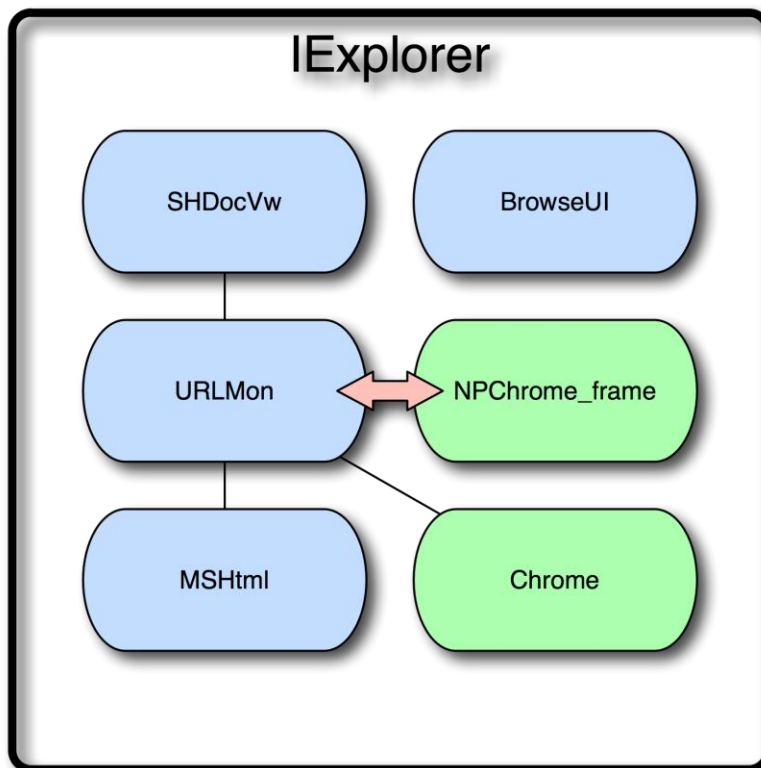
Beyond mere asymmetrical renderings, there are several other features of a browser that can differ. Visual Basic Script is a proprietary Internet Explorer technology; Google Chrome supports different Java Script constructs. One browser may offer more security features, another may already be integrated with the user's operating system. These nuances and more lead individuals to prefer one browser over another one. Sometimes the task of switching browsers is involved and requires switching browsers for different web sites in order to use the most compatible browser with a particular website. Google created Chrome Frame to open advancing web technologies to legacy browser users by allowing websites to be interpreted by either Google Chrome code or Internet Explorer code depending on the requirements.

Google's Chrome Frame is a browser plug-in that allows a Microsoft Internet Explorer user to take advantage of parts of the Google Chrome web browser. Once a user has installed Chrome Frame, they can replace parts of Internet Explorer with Google Chrome to leverage advantages in the Google Chrome browser, while still using Internet Explorer when a web page requires its use.

Chrome Frame may be the first browser plug-in that attempts to replace the web browser core. Generally, plug-ins are meant to add functionality beyond what a standard web browser offers. Given that the concept behind this plug-in is so unique, the security implications of using the Chrome Frame plug-in are not well studied. This paper attempts to address the security implications behind the Chrome Frame browser, first by providing a decomposition of the plug-in that outlines its mechanics; second, by mapping out the attack surface in order to illustrate what risks increase for users of this plug-in. Finally, this attack surface will be compared to that of other plug-ins in order to provide an idea of relative risk.

Decomposition

The following diagram illustrates how Microsoft Internet Explorer functions with Google's Chrome Frame installed.



The IExplorer process displays the traditional Internet Explorer frame that supports all of the controls within Internet Explorer that are not part of a web site or a third party add-on. Additionally, it hosts the SHDocVw active document container, also known as the WebBrowser control. SHDocVw, in turn, hosts Active Document controls.

Active Document controls are a part of the windows UI architecture that allows viewing and editing documents using the same controls within various active document containers. First, SHDocVw uses URLMon to determine the MIME type for the current URL. Next, SHDocVw uses that MIME type to determine the appropriate Active Document Control for displaying the URL.

Chrome Frame modifies how IExplorer works using application hooking techniques. The NPChrome_frame component is loaded either as a browser helper object using normal methods or by injection. Once loaded, it hooks various functions within URLMon. The hooking points serve a dual purpose, allowing Chrome Frame to detect when it is appropriate to use Chrome instead of MSHtml and to swap out the use of MSHtml for parsing text/HTML mime types and replace it with Chrome.

The Chrome and MSHtml components both serve the same purposes. They both parse HTML and render the result for the end user. They both can host controls to view non-HTML content and run JavaScript

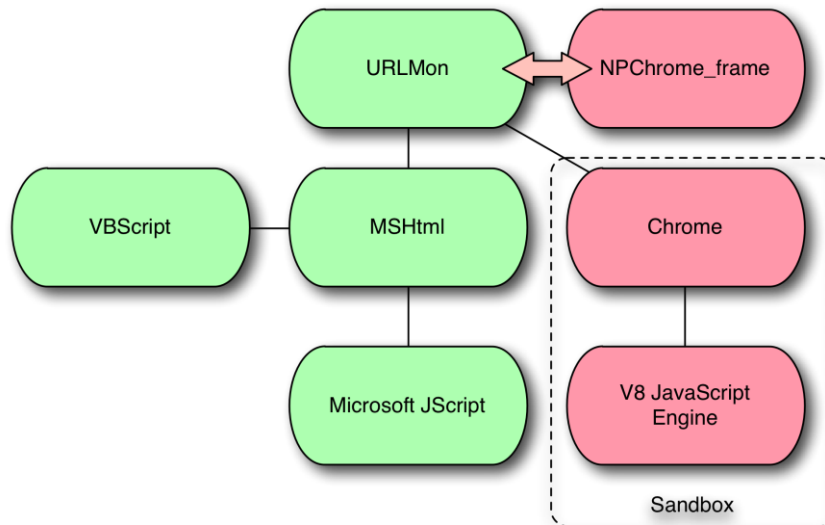
using their own JavaScript engines. The differentiator between the two is the code they use to parse and render HTML, and the code they use to run JavaScript. When the Chrome component is used, the results exhibit symmetry with how the Google Chrome browser interprets a web page and runs scripts. When the MSHTML component is used, sites will run as if Chrome Frame was never installed.

Security Implications

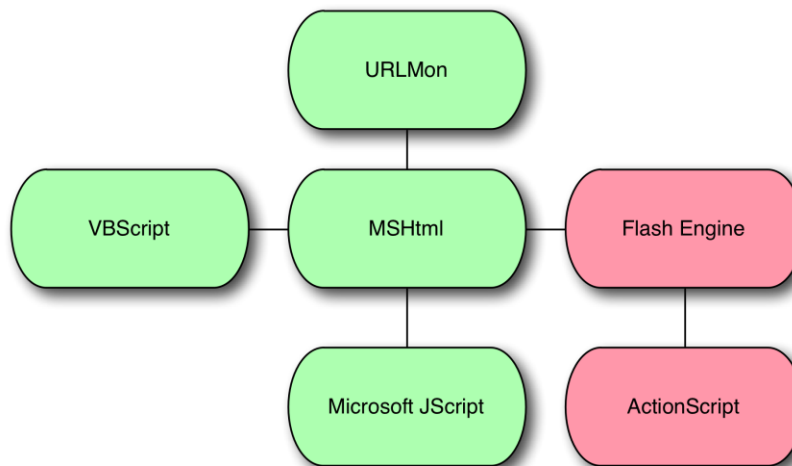
A common term used to describe security implications is *attack surface*. This term originates from physical combat where it is used to describe the surface area one combatant is exposing to another that could potentially be vulnerable to attack. Within the computer security domain, it describes the amount of code an attacker can supply with input. The theory being, if you reduce the amount of code an attacker can interact with, you reduce the population size of code that could present an exploitable condition.

If an end user adds a new component to a web browser, it increases the attack surface of that browser. This maxim is true for Adobe Flash, Apple QuickTime, Oracle's Java and Chrome Frame. The key tradeoff to consider is the increase in functionality versus the increase in attack surface. Adobe Flash allows the end user to consume flash-based content, Java allows the end user to consume Java-based content. The standard web browser already handles the majority of features in Google Chrome Frame. However, the features that aren't offered by the standard web browser may be compelling enough to accept the additional risk.

While the benefits are well understood by people considering deploying Chrome Frame, the increase in attack surface is not as well understood. Microsoft is quoted [**Microsoft_NetworkWorld**] as saying, "Google Chrome Frame running as a plug-in has doubled the attach[sic] area for malware and malicious scripts." Assuming that this can be rewritten as, "Google Chrome Frame running as a plug-in has doubled the attack surface for vulnerabilities" without losing any of the intended meaning, it shows that even Microsoft has a difficult time understanding the security implications of Chrome Frame. Given the assumption that an attack surface is calculated by the number of lines of code an attacker can interact with, Chrome Frame would have to have the same number of lines of code that can be interacted with by an attacker as Internet Explorer, which is clearly incorrect. In order to understand the issue more clearly, consider the following diagram.



The red components represent additional code with which an attacker can interact. Specifically, the two areas an attacker is most able to influence are the Chrome rendering engine and the V8 JavaScript engine. These components have had vulnerabilities in the past, and will continue to have vulnerabilities. However, consider the following diagram.



An attacker may interact with these additional components if Adobe's Flash product is installed. Qualitatively, they are not much different from Chrome. The Flash Engine supports layouts and object placement much like the Chrome rendering engine. The ActionScript component supports an interpreted language much like the V8 JavaScript Engine. The only real difference between the additional attack surface Chrome yields is the fact that the browser already has an HTML rendering engine and a JavaScript engine. Although the overlapped attack surface exists, it should be noted that Chrome Frame is run in a sandboxed environment, leaving the person deploying Chrome Frame to decide if deployment is worth the additional security risk.

Risk Mitigation Strategies

In the default configuration, Chrome Frame increases the risk to a browser in order to provide increased functionality much like other plugins. The reason for this increased risk is, in the default configuration, an attacker can choose which engine will render the HTML. However, this behavior can be changed with Group Policy settings. The following registry script shows an example:

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Policies\Google]

[HKEY_CURRENT_USER\Software\Policies\Google\Chrome]
"ChromeFrameRendererSettings"=dword:00000001

[HKEY_CURRENT_USER\Software\Policies\Google\Chrome\RenderInHostList]
"1"="http://www.RenderInIE.com*"
"2"="http://www.AlsoRenderInIE.com*"
```

With these settings defined, everything will render in Chrome rather than MSHTML unless a specific exception is made in the above white list. In the example above, pages matching the URLs http://www.RenderInIE.com* and http://www.AlsoRenderInIE.com* will render in IE. With this configuration scheme, the attacker can no longer choose which renderer will be used and therefore the attack surface is isolated to one HTML rendering engine or another. By isolating the rendering engine in this manner, additional attack surface is eliminated. The result of this elimination is that the attack surface will be equivalent to a standard user running IE or a standard user running Google Chrome depending on your configuration. Thus, any additional security concerns will be mitigated.

Conclusion

Installing Google's Chrome Frame with default permissions can increase your browser's attack surface using default settings, though the overall increase in attack surface is equivalent to other common browser plug-ins. Although there is duplicated attack surface from a browser rendering perspective, the sandbox that is integrated into Chrome Frame mitigates the security risk in certain situations. Users concerned by the increase in attack surface can mitigate their risk through judicious group policy settings.

Bibliography

- Microsoft_NetworkWorld – Microsoft says Chrome Frame Doubles Attack Surface (<http://www.networkworld.com/community/blog/microsoft-says-google-chrome-frame-doubles-at>)
- Chrome Frame Administrator's Guide (<http://www.chromium.org/developers/how-tos/chrome-frame-getting-started/chrome-frame-administrator-s-guide#TOC-Default-renderer>)
- Chrome Frame Policy List (<http://www.chromium.org/administrators/policy-list-3>)
- Chrome Frame Source Code (http://src.chromium.org/viewvc/chrome/trunk/src/chrome_frame/)
- Internet Explorer Architecture (<http://msdn.microsoft.com/en-us/library/aa741312%28v=vs.85%29.aspx>)
- About The Browser (Internet Explorer) (<http://msdn.microsoft.com/en-us/library/aa741313%28v=vs.85%29.aspx>)
- Active Documents (<http://msdn.microsoft.com/en-us/library/bx9c54kf%28v=VS.71%29.aspx>)
- Active Document Container (<http://msdn.microsoft.com/en-us/library/644x1yy6%28VS.71%29.aspx>)

Appendix B

Google Chrome

ASLR Results

Module	ASLR
%USERPROFILE%\AppData\Local\Google\Chrome\Application\chrome.exe	Enabled
C:\Windows\SYSTEM32\ntdll.dll	Enabled
C:\Windows\system32\kernel32.dll	Enabled
C:\Windows\system32\KERNELBASE.dll	Enabled
C:\Windows\system32\USER32.dll	Enabled
C:\Windows\system32\GDI32.dll	Enabled
C:\Windows\system32\LPK.dll	Enabled
C:\Windows\system32\USP10.dll	Enabled
C:\Windows\system32\msvcrt.dll	Enabled
C:\Windows\system32\SHELL32.dll	Enabled
C:\Windows\system32\SHLWAPI.dll	Enabled
C:\Windows\system32\USERENV.dll	Enabled
C:\Windows\system32\RPCRT4.dll	Enabled
C:\Windows\system32\profapi.dll	Enabled
C:\Windows\system32\WTSAPI32.dll	Enabled
C:\Windows\system32\VERSION.dll	Enabled
C:\Windows\system32\ADVAPI32.dll	Enabled
C:\Windows\SYSTEM32\sechost.dll	Enabled
C:\Windows\system32\IMM32.DLL	Enabled
C:\Windows\system32\MSCTF.dll	Enabled
C:\Windows\system32\ole32.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\chrome.dll	Enabled
C:\Windows\system32\OLEAUT32.dll	Enabled
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2\COMCTL32.dll	Enabled
C:\Windows\system32\OLEACC.dll	Enabled
C:\Windows\system32\RICHED20.dll	Enabled
C:\Windows\system32\PSAPI.DLL	Enabled
C:\Windows\system32\WINMM.dll	Enabled
C:\Windows\system32\DNSAPI.dll	Enabled
C:\Windows\system32\WS2_32.dll	Enabled
C:\Windows\system32\NSI.dll	Enabled
C:\Windows\system32\MSIMG32.dll	Enabled
C:\Windows\system32\Secur32.dll	Enabled
C:\Windows\system32\SSPICLI.DLL	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\icudt.dll	Enabled
C:\Windows\system32\CRYPTBASE.dll	Enabled

Module	ASLR
C:\Windows\system32\IPHLPAPI.DLL	Enabled
C:\Windows\system32\WINNSI.DLL	Enabled
C:\Windows\system32\ntmarta.dll	Enabled
C:\Windows\system32\WLDAP32.dll	Enabled
C:\Windows\system32\GPAPI.dll	Enabled
C:\Windows\system32\mswsock.dll	Enabled
C:\Windows\System32\wship6.dll	Enabled
C:\Windows\system32\dhcpcsvc6.DLL	Enabled
C:\Windows\system32\dhcpcsvc.DLL	Enabled
C:\Windows\system32\mscms.dll	Enabled
C:\Windows\System32\wshtcpip.dll	Enabled
C:\Windows\system32\rasadhlp.dll	Enabled
C:\Windows\system32\WINTRUST.dll	Enabled
C:\Windows\system32\CRYPT32.dll	Enabled
C:\Windows\system32\MSASN1.dll	Enabled
C:\Windows\system32\dwmapi.dll	Enabled
C:\Windows\system32\UxTheme.dll	Enabled
C:\Windows\system32\SXS.DLL	Enabled
C:\Windows\system32\CLBCatQ.DLL	Enabled
C:\Windows\System32\fwpuclnt.dll	Enabled
C:\Windows\system32\PROPSYS.dll	Enabled
C:\Windows\system32\LINKINFO.dll	Enabled
C:\Windows\system32\apphelp.dll	Enabled
C:\Windows\System32\shdocvw.dll	Enabled
C:\Windows\system32\SETUPAPI.dll	Enabled
C:\Windows\system32\CFGMGR32.dll	Enabled
C:\Windows\system32\DEVOBJ.dll	Enabled
C:\Windows\system32\WINHTTP.dll	Enabled
C:\Windows\system32\webio.dll	Enabled
C:\Windows\system32\credssp.dll	Enabled
C:\Windows\system32\CRYPTSP.dll	Enabled
C:\Windows\system32\rsaenh.dll	Enabled
C:\Windows\system32\cryptnet.dll	Enabled
C:\Windows\system32\SensApi.dll	Enabled
C:\Windows\system32\Cabinet.dll	Enabled
C:\Windows\system32\DEVRTL.dll	Enabled
C:\Windows\system32\ncrypt.dll	Enabled
C:\Windows\system32\bcrypt.dll	Enabled
C:\Windows\system32\bcryptprimitives.dll	Enabled
C:\Windows\system32\NLAapi.dll	Enabled
%USERPROFILE%\AppData\Local\Apps\2.0\TH279LXB.K2W\H5KXY3PX.NLL\clie...exe_f84b370c827b5c7a_0001.0003_none_f6c591a8ff607af3\GoogleUpdateSetup.exe	Enabled
%USERPROFILE%\AppData\Local\Apps\2.0\TH279LXB.K2W\H5KXY3PX.NLL\goog...app_	Enabled

Module	ASLR
f84b370c827b5c7a_0001.0003_5cbb67db0893f7c4\clickonce_bootstrap.exe	
%USERPROFILE%\AppData\Local\Apps\2.0\TH279LXB.K2W\H5KXY3PX.NLL\goog...app_f84b370c827b5c7a_0001.0003_5cbb67db0893f7c4\GoogleUpdateSetup.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\avcodec-52.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\avformat-52.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\avutil-50.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\chrome_frame_helper.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\chrome_frame_helper.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\chrome_launcher.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\d3dcompiler_43.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\d3dx9_43.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\flashplayerrcplapp.cpl	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\gcswf32.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\libegl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\libglesv2.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\nacl64.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\nacl64.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\npchrome_frame.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\pdf.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\ppgooglenaclplug-inchrome.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Installer\setup.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\am.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ar.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\bg.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\bn.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ca.dll	Enabled

Module	ASLR
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\cs.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\da.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\de.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\el.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\en-GB.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\en-US.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\es-419.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\es.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\et.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\fa.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\fi.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\fil.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\fr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\gu.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\he.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\hi.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\hr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\hu.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\id.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\it.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ja.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\kn.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ko	Enabled

Module	ASLR
.dll	
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\it.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\iv.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ml.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\mr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\nb.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\nl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\pl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\pt-BR.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\pt-PT.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ro.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ru.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\sk.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\sl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\sr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\sv.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\sw.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\ta.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\te.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\th.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\tr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\uk.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\vi.dll	Enabled

Module	ASLR
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\zh-CN.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales\zh-TW.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\GoogleCrashHandler.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\GoogleUpdate.exe	Disabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\GoogleUpdateBroker.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\GoogleUpdateOnDemand.exe	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdate.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_am.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ar.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_bg.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_bn.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ca.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_cs.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_da.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_de.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_el.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_en-GB.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_en.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_es-419.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_es.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_et.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_fa.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_fi.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_fil.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_fr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_gu.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_hi.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_hr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_hu.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_id.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_is.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_it.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_iw.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ja.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_kn.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ko.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_lt.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_lv.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ml.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_mr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ms.dll	Enabled

Module	ASLR
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_nl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_no.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_pl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_pt-BR.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_pt-PT.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ro.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ru.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_sk.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_sl.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_sr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_sv.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_sw.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ta.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_te.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_th.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_tr.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_uk.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_ur.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_vi.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_zh-CN.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\goopdateres_zh-TW.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\npGoogleUpdate3.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\psmachine.dll	Enabled
%USERPROFILE%\AppData\Local\Google\Update\1.3.21.57\psuser.dll	Enabled

GS Results

The follow IDA Pro databases were derived from file installed or used by Google Chrome. The original files can be found in the following directories:

Any items labeled **INDETERMINATE** consist of libraries from which debugging symbols were not acquired.

- %USERPROFILE%\AppData\Local\Google\Update\1.3.21.57
- %USERPROFILE%\AppData\Local\Google\Chrome\Application
- %USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122
- %USERPROFILE%\AppData\Local\Google\Chrome\Application\12.0.742.122\Locales
- C:\Windows\System32

Module	Found /GS
ADVAPI32.idb	TRUE
am.idb	INDETERMINATE
apphelp.idb	TRUE
ar.idb	INDETERMINATE
avcodec-52.idb	INDETERMINATE
avformat-52.idb	INDETERMINATE
avutil-50.idb	INDETERMINATE
bcrypt.idb	TRUE
bcryptprimitives.idb	TRUE
bg.idb	INDETERMINATE
bn.idb	INDETERMINATE
ca.idb	INDETERMINATE
Cabinet.idb	TRUE
CFGMR32.idb	TRUE
chrome_dll.idb	TRUE
chrome_exe.idb	TRUE
chrome_frame_helper.idb	TRUE
chrome_launcher.idb	TRUE
CLBCatQ.idb	TRUE
clickonce_bootstrap.idb	INDETERMINATE
COMCTL32.idb	TRUE
credssp.idb	TRUE
CRYPT32.idb	TRUE
CRYPTBASE.idb	TRUE
cryptnet.idb	TRUE
CRYPTSP.idb	TRUE
cs.idb	INDETERMINATE
d3dcompiler_43.idb	TRUE
d3dx9_43.idb	TRUE
da.idb	INDETERMINATE

Module	Found /GS
de.idb	INDETERMINATE
DEVOBJ.idb	TRUE
DEVRTL.idb	TRUE
dhcpcsvc.idb	TRUE
dhcpcsvc6.idb	TRUE
DNSAPI.idb	TRUE
dwmapi.idb	TRUE
el.idb	INDETERMINATE
en-GB.idb	INDETERMINATE
en-US.idb	INDETERMINATE
es-419.idb	INDETERMINATE
es.idb	INDETERMINATE
et.idb	INDETERMINATE
fa.idb	INDETERMINATE
fi.idb	INDETERMINATE
fil.idb	INDETERMINATE
fr.idb	INDETERMINATE
fwpuclnt.idb	TRUE
gcswf32.idb	TRUE
GDI32.idb	TRUE
GoogleCrashHandler.idb	TRUE
GoogleUpdate.idb	TRUE
GoogleUpdateBroker.idb	TRUE
GoogleUpdateOnDemand.idb	TRUE
GoogleUpdateSetup.idb	INDETERMINATE
goopdate.idb	TRUE
goopdateres_am.idb	INDETERMINATE
goopdateres_ar.idb	INDETERMINATE
goopdateres_bg.idb	INDETERMINATE
goopdateres_bn.idb	INDETERMINATE
goopdateres_ca.idb	INDETERMINATE
goopdateres_cs.idb	INDETERMINATE
goopdateres_da.idb	INDETERMINATE
goopdateres_de.idb	INDETERMINATE
goopdateres_el.idb	INDETERMINATE
goopdateres_en-GB.idb	INDETERMINATE
goopdateres_en.idb	INDETERMINATE
goopdateres_es-419.idb	INDETERMINATE
goopdateres_es.idb	INDETERMINATE
goopdateres_et.idb	INDETERMINATE
goopdateres_fa.idb	INDETERMINATE
goopdateres_fi.idb	INDETERMINATE
goopdateres_fil.idb	INDETERMINATE

Module	Found /GS
goopdateres_fr.idb	INDETERMINATE
goopdateres_gu.idb	INDETERMINATE
goopdateres_hi.idb	INDETERMINATE
goopdateres_hr.idb	INDETERMINATE
goopdateres_hu.idb	INDETERMINATE
goopdateres_id.idb	INDETERMINATE
goopdateres_is.idb	INDETERMINATE
goopdateres_it.idb	INDETERMINATE
goopdateres_iw.idb	INDETERMINATE
goopdateres_ja.idb	INDETERMINATE
goopdateres_kn.idb	INDETERMINATE
goopdateres_ko.idb	INDETERMINATE
goopdateres_lt.idb	INDETERMINATE
goopdateres_lv.idb	INDETERMINATE
goopdateres_ml.idb	INDETERMINATE
goopdateres_mr.idb	INDETERMINATE
goopdateres_ms.idb	INDETERMINATE
goopdateres_nl.idb	INDETERMINATE
goopdateres_no.idb	INDETERMINATE
goopdateres_pl.idb	INDETERMINATE
goopdateres_pt-BR.idb	INDETERMINATE
goopdateres_pt-PT.idb	INDETERMINATE
goopdateres_ro.idb	INDETERMINATE
goopdateres_ru.idb	INDETERMINATE
goopdateres_sk.idb	INDETERMINATE
goopdateres_sl.idb	INDETERMINATE
goopdateres_sr.idb	INDETERMINATE
goopdateres_sv.idb	INDETERMINATE
goopdateres_sw.idb	INDETERMINATE
goopdateres_ta.idb	INDETERMINATE
goopdateres_te.idb	INDETERMINATE
goopdateres_th.idb	INDETERMINATE
goopdateres_tr.idb	INDETERMINATE
goopdateres_uk.idb	INDETERMINATE
goopdateres_ur.idb	INDETERMINATE
goopdateres_vi.idb	INDETERMINATE
goopdateres_zh-CN.idb	INDETERMINATE
goopdateres_zh-TW.idb	INDETERMINATE
GPAPI.idb	TRUE
gu.idb	INDETERMINATE
he.idb	INDETERMINATE
hi.idb	INDETERMINATE
hr.idb	INDETERMINATE

Module	Found /GS
hu.idb	INDETERMINATE
icudt.idb	INDETERMINATE
id.idb	INDETERMINATE
IMM32.idb	TRUE
IPHPAPI.idb	TRUE
it.idb	INDETERMINATE
ja.idb	INDETERMINATE
kernel32.idb	TRUE
KERNELBASE.idb	TRUE
kn.idb	INDETERMINATE
ko.idb	INDETERMINATE
libegl.idb	TRUE
libglesv2.idb	TRUE
LINKINFO.idb	TRUE
LPK.idb	TRUE
lt.idb	INDETERMINATE
lv.idb	INDETERMINATE
ml.idb	INDETERMINATE
mr.idb	INDETERMINATE
MSASN1.idb	TRUE
mscms.idb	TRUE
MSCTF.idb	TRUE
MSIMG32.idb	TRUE
msvcrt.idb	TRUE
mswsock.idb	TRUE
nb.idb	INDETERMINATE
ncrypt.idb	TRUE
nl.idb	INDETERMINATE
NLAapi.idb	TRUE
npchrome_frame.idb	TRUE
npGoogleUpdate3.idb	TRUE
NSI.idb	TRUE
ntdll.idb	TRUE
ntmarta.idb	TRUE
ole32.idb	TRUE
OLEACC.idb	TRUE
OLEAUT32.idb	TRUE
pdf.idb	TRUE
pl.idb	INDETERMINATE
ppgooglenaclplug-inchrome.idb	TRUE
profapi.idb	TRUE
PROPSYS.idb	TRUE
PSAPI.idb	TRUE

Module	Found /GS
psmachine.idb	TRUE
psuser.idb	TRUE
pt-BR.idb	INDETERMINATE
pt-PT.idb	INDETERMINATE
rasadhlp.idb	TRUE
RICHED20.idb	TRUE
ro.idb	INDETERMINATE
RPCRT4.idb	TRUE
rsaenh.idb	TRUE
ru.idb	INDETERMINATE
sechost.idb	TRUE
Secur32.idb	TRUE
SensApi.idb	TRUE
setup.idb	TRUE
SETUPAPI.idb	TRUE
shdocvw.idb	TRUE
SHELL32.idb	TRUE
SHLWAPI.idb	TRUE
sk.idb	INDETERMINATE
sl.idb	INDETERMINATE
sr.idb	INDETERMINATE
SSPICLI.idb	TRUE
sv.idb	INDETERMINATE
sw.idb	INDETERMINATE
SXS.idb	TRUE
ta.idb	INDETERMINATE
te.idb	INDETERMINATE
th.idb	INDETERMINATE
tr.idb	INDETERMINATE
uk.idb	INDETERMINATE
USER32.idb	TRUE
USERENV.idb	TRUE
USP10.idb	TRUE
UxTheme.idb	TRUE
VERSION.idb	TRUE
vi.idb	INDETERMINATE
webio.idb	TRUE
WINHTTP.idb	TRUE
WINMM.idb	TRUE
WINNSI.idb	TRUE
WINTRUST.idb	TRUE
WLDAP32.idb	TRUE
WS2_32.idb	TRUE

Module	Found /GS
wship6.idb	TRUE
wshtcpip.idb	TRUE
WTSAPI32.idb	TRUE
zh-CN.idb	INDETERMINATE
zh-TW.idb	INDETERMINATE

Internet Explorer

ASLR Results

Module	ASLR
C:\Program Files\Internet Explorer\iexplore.exe	Enabled
C:\Windows\SYSTEM32\ntdll.dll	Enabled
C:\Windows\system32\kernel32.dll	Enabled
C:\Windows\system32\KERNELBASE.dll	Enabled
C:\Windows\system32\ADVAPI32.dll	Enabled
C:\Windows\system32\msvcrt.dll	Enabled
C:\Windows\SYSTEM32\sechost.dll	Enabled
C:\Windows\system32\RPCRT4.dll	Enabled
C:\Windows\system32\USER32.dll	Enabled
C:\Windows\system32\GDI32.dll	Enabled
C:\Windows\system32\LPK.dll	Enabled
C:\Windows\system32\USP10.dll	Enabled
C:\Windows\system32\SHLWAPI.dll	Enabled
C:\Windows\system32\SHELL32.dll	Enabled
C:\Windows\system32\ole32.dll	Enabled
C:\Windows\system32\urlmon.dll	Enabled
C:\Windows\system32\OLEAUT32.dll	Enabled
C:\Windows\system32\iertutil.dll	Enabled
C:\Windows\system32\WININET.dll	Enabled
C:\Windows\system32\Normaliz.dll	Enabled
C:\Windows\system32\IMM32.DLL	Enabled
C:\Windows\system32\MSCTF.dll	Enabled
C:\Windows\system32\IEFRAME.dll	Enabled
C:\Windows\system32\PSAPI.DLL	Enabled
C:\Windows\system32\OLEACC.dll	Enabled
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2\comctl32.dll	Enabled
C:\Windows\system32\comdlg32.dll	Enabled
C:\Windows\system32\CRYPTBASE.dll	Enabled
C:\Windows\system32\Secur32.dll	Enabled
C:\Windows\system32\SSPICLI.DLL	Enabled
C:\Windows\system32\profapi.dll	Enabled
C:\Windows\system32\ntmarta.dll	Enabled
C:\Windows\system32\WLDAP32.dll	Enabled

Module	ASLR
C:\Windows\system32\WS2_32.dll	Enabled
C:\Windows\system32\NSI.dll	Enabled
C:\Windows\system32\dnsapi.DLL	Enabled
C:\Windows\system32\iphlpapi.DLL	Enabled
C:\Windows\system32\WINNSI.DLL	Enabled
C:\Windows\system32\CLBCatQ.DLL	Enabled
C:\Windows\System32\netprofm.dll	Enabled
C:\Windows\System32\nlaapi.dll	Enabled
C:\Windows\system32\CRYPTSP.dll	Enabled
C:\Windows\system32\rsaenh.dll	Enabled
C:\Windows\system32\RpcRtRemote.dll	Enabled
C:\Windows\System32\npmproxy.dll	Enabled
C:\Windows\system32\mswsock.dll	Enabled
C:\Windows\System32\wshtcpip.dll	Enabled
C:\Windows\System32\wship6.dll	Enabled
C:\Windows\system32\rasadhlp.dll	Enabled
C:\Program Files\Internet Explorer\ieproxy.dll	Enabled
C:\Windows\system32\RASAPI32.dll	Enabled
C:\Windows\system32\rasman.dll	Enabled
C:\Windows\System32\fwpuclnt.dll	Enabled
C:\Windows\system32\rtutils.dll	Enabled
C:\Windows\system32\sensapi.dll	Enabled
C:\Windows\system32\IEUI.dll	Enabled
C:\Windows\system32\MSIMG32.dll	Enabled
C:\Windows\system32\UxTheme.dll	Enabled
C:\Windows\system32\apphelp.dll	Enabled
C:\Windows\system32\xmllite.dll	Enabled
C:\Windows\system32\explorerframe.dll	Enabled
C:\Windows\system32\DUser.dll	Enabled
C:\Windows\system32\DUI70.dll	Enabled
C:\Windows\system32\PROPSYS.dll	Enabled
C:\Windows\system32\mssprxy.dll	Enabled
C:\Windows\system32\SXS.DLL	Enabled
C:\Windows\system32\MLANG.dll	Enabled
C:\Windows\system32\SETUPAPI.dll	Enabled
C:\Windows\system32\CFGMGR32.dll	Enabled
C:\Windows\system32\DEVOBJ.dll	Enabled
C:\Windows\system32\CRYPT32.dll	Enabled
C:\Windows\system32\MSASN1.dll	Enabled
C:\Program Files\Internet Explorer\ExtExport.exe	Enabled
C:\Program Files\Internet Explorer\iecleanup.exe	Enabled
C:\Program Files\Internet Explorer\iediagcmd.exe	Enabled
C:\Program Files\Internet Explorer\iedvtool.dll	Enabled

Module	ASLR
C:\Program Files\Internet Explorer\ieinstal.exe	Enabled
C:\Program Files\Internet Explorer\ielowutil.exe	Enabled
C:\Program Files\Internet Explorer\ieproxy.dll_old0	Enabled
C:\Program Files\Internet Explorer\IEShims.dll	Enabled
C:\Program Files\Internet Explorer\iessetup.dll	Enabled
C:\Program Files\Internet Explorer\jsdbgui.dll	Enabled
C:\Program Files\Internet Explorer\jsdebuggeride.dll	Enabled
C:\Program Files\Internet Explorer\JSProfilerCore.dll	Enabled
C:\Program Files\Internet Explorer\jsprofilerui.dll	Enabled
C:\Program Files\Internet Explorer\msdbg2.dll	Enabled
C:\Program Files\Internet Explorer\networkinspection.dll	Enabled
C:\Program Files\Internet Explorer\pdm.dll	Enabled
C:\Program Files\Internet Explorer\sqmapi.dll	Enabled
C:\Program Files\Internet Explorer\sqmapi.dll_old0	Enabled

GS Results

The follow IDA Pro databases were derived from files installed or used by Internet Explorer. The original files can be found in the following directories:

- C:\Windows\System32
- C:\Program Files\Internet Explorer
- C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2

Module	Found /GS
ADVAPI32.idb	TRUE
apphelp.idb	TRUE
CFGMGR32.idb	TRUE
CLBCatQ.idb	TRUE
comctl32.idb	TRUE
comdlg32.idb	TRUE
CRYPT32.idb	TRUE
CRYPTBASE.idb	TRUE
CRYPTSP.idb	TRUE
DEVOBJ.idb	TRUE
dnsapi.idb	TRUE
DUI70.idb	TRUE
DUser.idb	TRUE
explorerframe.idb	TRUE
ExtExport.idb	TRUE
fwpuclnt.idb	TRUE
GDI32.idb	TRUE
iecleanup.idb	TRUE
iediagcmd.idb	FALSE

Module	Found /GS
iedvtool.idb	TRUE
IEFRAME.idb	TRUE
ieinstal.idb	TRUE
ielowutil.idb	TRUE
ieproxy.idb	TRUE
iertutil.idb	TRUE
IEShims.idb	TRUE
iessetup.idb	TRUE
IEUI.idb	TRUE
iexplore.idb	TRUE
IMM32.idb	TRUE
iphlpapi.idb	TRUE
jsdbgui.idb	TRUE
jsdebuggeride.idb	TRUE
JSProfilerCore.idb	TRUE
jsprofilerui.idb	TRUE
kernel32.idb	TRUE
KERNELBASE.idb	TRUE
LPK.idb	TRUE
MLANG.idb	TRUE
MSASN1.idb	TRUE
MSCTF.idb	TRUE
msdbg2.idb	TRUE
MSIMG32.idb	TRUE
mssprxy.idb	TRUE
msvcrt.idb	TRUE
mswsock.idb	TRUE
netprofm.idb	TRUE
networkinspection.idb	TRUE
nlaapi.idb	TRUE
Normaliz.idb	FALSE
npmproxy.idb	TRUE
NSI.idb	TRUE
ntdll.idb	TRUE
ntmarta.idb	TRUE
ole32.idb	TRUE
OLEACC.idb	TRUE
OLEAUT32.idb	TRUE
pdm.idb	TRUE
profapi.idb	TRUE
PROPSYS.idb	TRUE
PSAPI.idb	TRUE
rasadhlp.idb	TRUE

Module	Found /GS
RASAPI32.idb	TRUE
rasman.idb	TRUE
RPCRT4.idb	TRUE
RpcRtRemote.idb	TRUE
rsaenh.idb	TRUE
rtutils.idb	TRUE
sechost.idb	TRUE
Secur32.idb	TRUE
sensapi.idb	TRUE
SETUPAPI.idb	TRUE
SHELL32.idb	TRUE
SHLWAPI.idb	TRUE
sqmapi.idb	TRUE
SSPICLI.idb	TRUE
SXS.idb	TRUE
urlmon.idb	TRUE
USER32.idb	TRUE
USP10.idb	TRUE
UxTheme.idb	TRUE
WININET.idb	TRUE
WINNSI.idb	TRUE
WLDAP32.idb	TRUE
WS2_32.idb	TRUE
wship6.idb	TRUE
wshtcpip.idb	TRUE
xmlite.idb	TRUE

Mozilla Firefox

ASLR Results

Module	ASLR
C:\Program Files\Mozilla Firefox\firefox.exe	Enabled
C:\Windows\SYSTEM32\ntdll.dll	Enabled
C:\Windows\system32\kernel32.dll	Enabled
C:\Windows\system32\KERNELBASE.dll	Enabled
C:\Program Files\Mozilla Firefox\xul.dll	Enabled
C:\Program Files\Mozilla Firefox\mozsqlite3.dll	Enabled
C:\Program Files\Mozilla Firefox\MOZCRT19.dll	Enabled
C:\Windows\system32\msvcrt.dll	Enabled
C:\Program Files\Mozilla Firefox\mozjs.dll	Enabled
C:\Program Files\Mozilla Firefox\nspr4.dll	Enabled
C:\Windows\system32\ADVAPI32.dll	Enabled
C:\Windows\SYSTEM32\sechost.dll	Enabled
C:\Windows\system32\RPCRT4.dll	Enabled
C:\Windows\system32\WSOCK32.dll	Enabled
C:\Windows\system32\WS2_32.dll	Enabled
C:\Windows\system32\NSI.dll	Enabled
C:\Windows\system32\WINMM.dll	Enabled
C:\Windows\system32\USER32.dll	Enabled
C:\Windows\system32\GDI32.dll	Enabled
C:\Windows\system32\LPK.dll	Enabled
C:\Windows\system32\USP10.dll	Enabled
C:\Program Files\Mozilla Firefox\smime3.dll	Enabled
C:\Program Files\Mozilla Firefox\nss3.dll	Enabled
C:\Program Files\Mozilla Firefox\nssutil3.dll	Enabled
C:\Program Files\Mozilla Firefox\plc4.dll	Enabled
C:\Program Files\Mozilla Firefox\plds4.dll	Enabled
C:\Program Files\Mozilla Firefox\ssl3.dll	Enabled
C:\Program Files\Mozilla Firefox\mozalloc.dll	Enabled
C:\Windows\system32\SHELL32.dll	Enabled
C:\Windows\system32\SHLWAPI.dll	Enabled
C:\Windows\system32\ole32.dll	Enabled
C:\Windows\system32\VERSION.dll	Enabled
C:\Windows\system32\WINSPOOL.DRV	Enabled
C:\Windows\system32\COMDLG32.dll	Enabled
C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2\COMCTL32.dll	Enabled
C:\Windows\system32\IMM32.dll	Enabled
C:\Windows\system32\MSCTF.dll	Enabled
C:\Windows\system32\MSIMG32.dll	Enabled
C:\Windows\system32\PSAPI.DLL	Enabled

Module	ASLR
C:\Windows\system32\OLEAUT32.dll	Enabled
C:\Program Files\Mozilla Firefox\MOZCPP19.dll	Enabled
C:\Program Files\Mozilla Firefox\xpcom.dll	Enabled
C:\Windows\system32\uxtheme.dll	Enabled
C:\Windows\system32\dwmapi.dll	Enabled
C:\Windows\system32\dbghelp.dll	Enabled
C:\Windows\system32\CRYPTBASE.dll	Enabled
C:\Windows\system32\CLBCatQ.DLL	Enabled
C:\Windows\system32\propsys.dll	Enabled
C:\Windows\system32\SETUPAPI.dll	Enabled
C:\Windows\system32\CFGMGR32.dll	Enabled
C:\Windows\system32\DEVOBJ.dll	Enabled
C:\Windows\system32\mswsock.dll	Enabled
C:\Windows\System32\wshtcpip.dll	Enabled
C:\Windows\system32\iphlpapi.dll	Enabled
C:\Windows\system32\WINNSI.DLL	Enabled
C:\Program Files\Mozilla Firefox\components\browsercomps.dll	Enabled
C:\Windows\system32\WINTRUST.dll	Enabled
C:\Windows\system32\CRYPT32.dll	Enabled
C:\Windows\system32\MSASN1.dll	Enabled
C:\Windows\system32\t2embed.dll	Enabled
C:\Windows\system32\WindowsCodecs.dll	Enabled
C:\Windows\system32\apphelp.dll	Enabled
C:\Windows\system32\EhStorShell.dll	Enabled
C:\Windows\System32\cscui.dll	Enabled
C:\Windows\System32\CSCDLL.dll	Enabled
C:\Windows\system32\CSCAPI.dll	Enabled
C:\Windows\system32\ntshrui.dll	Enabled
C:\Windows\system32\srvccli.dll	Enabled
C:\Windows\system32\slc.dll	Enabled
C:\Windows\system32\CRYPTSP.dll	Enabled
C:\Windows\system32\rsaenh.dll	Enabled
C:\Windows\system32\RpcRtRemote.dll	Enabled
C:\Windows\system32\profapi.dll	Enabled
C:\Windows\system32\dwrite.dll	Enabled
C:\Windows\system32\feclient.dll	Enabled
C:\Windows\system32\ntmarta.dll	Enabled
C:\Windows\system32\WLDAP32.dll	Enabled
C:\Windows\system32\NLAapi.dll	Enabled
C:\Windows\system32\napinsp.dll	Enabled
C:\Windows\system32\pnrpnp.dll	Enabled
C:\Windows\system32\DNSAPI.dll	Enabled
C:\Windows\System32\winrnr.dll	Enabled

Module	ASLR
C:\Windows\system32\mscms.dll	Enabled
C:\Windows\system32\USERENV.dll	Enabled
C:\Windows\System32\wship6.dll	Enabled
C:\Windows\system32\rasadhlp.dll	Enabled
C:\Windows\System32\fwpuclnt.dll	Enabled
C:\Program Files\Mozilla Firefox\softokn3.dll	Enabled
C:\Program Files\Mozilla Firefox\nssdbm3.dll	Enabled
C:\Program Files\Mozilla Firefox\freebl3.dll	Enabled
C:\Program Files\Mozilla Firefox\nssckbi.dll	Enabled
C:\Windows\system32\explorerframe.dll	Enabled
C:\Windows\system32\DUser.dll	Enabled
C:\Windows\system32\DUI70.dll	Enabled
C:\Windows\system32\shdocvw.dll	Enabled
C:\Program Files\Mozilla Firefox\AccessibleMarshal.dll	Enabled
C:\Program Files\Mozilla Firefox\crashreporter.exe	Enabled
C:\Program Files\Mozilla Firefox\D3DCompiler_43.dll	Enabled
C:\Program Files\Mozilla Firefox\d3dx9_43.dll	Enabled
C:\Program Files\Mozilla Firefox\libEGL.dll	Enabled
C:\Program Files\Mozilla Firefox\libGLESv2.dll	Enabled
C:\Program Files\Mozilla Firefox\mozcpp19.dll	Enabled
C:\Program Files\Mozilla Firefox\mozcrt19.dll	Enabled
C:\Program Files\Mozilla Firefox\plug-in-container.exe	Enabled
C:\Program Files\Mozilla Firefox\updater.exe	Enabled

GS Results

The follow IDA Pro databases were derived from file installed or used by Mozilla Firefox. The original files can be found in the following directories:

- C:\Windows\System32
- C:\Program Files\Mozilla Firefox
- C:\Program Files\Mozilla Firefox\components
- C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2

Module	Found /GS
AccessibleMarshal.idb	TRUE
ADVAPI32.idb	TRUE
apphelp.idb	TRUE
browsercomps.idb	TRUE
CFGMGR32.idb	TRUE
CLBCatQ.idb	TRUE
COMCTL32.idb	TRUE
COMDLG32.idb	TRUE
crashreporter.idb	TRUE

Module	Found /GS
CRYPT32.idb	TRUE
CRYPTBASE.idb	TRUE
CRYPTSP.idb	TRUE
CSCAPI.idb	TRUE
CSCDLL.idb	TRUE
cscui.idb	TRUE
D3DCompiler_43.idb	TRUE
d3dx9_43.idb	TRUE
dbghelp.idb	TRUE
DEVOBJ.idb	TRUE
DNSAPI.idb	TRUE
DUI70.idb	TRUE
DUser.idb	TRUE
dwmapi.idb	TRUE
dwrite.idb	TRUE
EhStorShell.idb	TRUE
explorerframe.idb	TRUE
feclient.idb	TRUE
firefox.idb	TRUE
freebl3.idb	TRUE
fwpuclnt.idb	TRUE
GDI32.idb	TRUE
IMM32.idb	TRUE
iphlpapi.idb	TRUE
kernel32.idb	TRUE
KERNELBASE.idb	TRUE
libEGL.idb	TRUE
libGLESv2.idb	TRUE
LPK.idb	TRUE
mozalloc.idb	TRUE
MOZCPP19.idb	TRUE
MOZCRT19.idb	TRUE
mozjs.idb	TRUE
mozsqlite3.idb	TRUE
MSASN1.idb	TRUE
mscms.idb	TRUE
MSCTF.idb	TRUE
MSIMG32.idb	TRUE
msvcrt.idb	TRUE
mswsock.idb	TRUE
napinsp.idb	TRUE
NLAapi.idb	TRUE
NSI.idb	TRUE

Module	Found /GS
nspr4.idb	TRUE
nss3.idb	TRUE
nssckbi.idb	TRUE
nssdbm3.idb	TRUE
nssutil3.idb	TRUE
ntdll.idb	TRUE
ntmarta.idb	TRUE
ntshrui.idb	TRUE
ole32.idb	TRUE
OLEAUT32.idb	TRUE
plc4.idb	TRUE
plds4.idb	TRUE
plug-in-container.idb	TRUE
pnrpnspl.idb	TRUE
profapi.idb	TRUE
prosys.idb	TRUE
PSAPI.idb	TRUE
rasadhlp.idb	TRUE
RPCRT4.idb	TRUE
RpcRtRemote.idb	TRUE
rsaenh.idb	TRUE
sechost.idb	TRUE
SETUPAPI.idb	TRUE
shdocvw.idb	TRUE
SHELL32.idb	TRUE
SHLWAPI.idb	TRUE
slc.idb	TRUE
smime3.idb	TRUE
softokn3.idb	TRUE
srvcli.idb	TRUE
ssl3.idb	TRUE
t2embed.idb	TRUE
updater.idb	TRUE
USER32.idb	TRUE
USERENV.idb	TRUE
USP10.idb	TRUE
uxtheme.idb	TRUE
VERSION.idb	TRUE
WindowsCodecs.idb	TRUE
WINMM.idb	TRUE
WINNSI.idb	TRUE
winrnr.idb	TRUE
WINTRUST.idb	TRUE

Module	Found /GS
WLDAP32.idb	TRUE
WS2_32.idb	TRUE
wship6.idb	TRUE
wshtcpip.idb	TRUE
WSOCK32.idb	TRUE
xpcom.idb	TRUE
xul.idb	TRUE

Tools

- **URL Blacklist Services**
 - **BrowserAutomation** - A directory containing a number of supporting utilities used in automating Internet Explorer via the Watir framework.
 - **getmalwareURLS.sh** - A utility to harvest malware URLs from public sources, concatenate, and remove duplicates, writing a time stamped linefeed-delimited output.
 - **IEURL.rb** - A wrapper for the Watir browser automation framework to automate URL requests in Internet Explorer and record results.
 - **malwareurls** – A directory containing the daily malware URLs used in testing SBL and URS.
 - **Results** – A directory containing the raw data output per-day, and other sorted data used in generating the blacklist reporting.
- **Historical Vulnerability Statistics**
 - **chrome-private-augment.zip** –Google-provided data that includes timeline data for non-publicly-accessible Chrome security bugs. This was used to augment timeline data coverage to include all publicly disclosed Chrome vulnerabilities.
 - **chrome-public-scrapers.zip** – Two HTML scrapers that process the publicly accessible Chrome Stable Release blog. One scrapes the blog itself into one file per release. The other scrapes the resulting files to produce SQL data. The input datasets are included.
 - **datadump-sql.zip** – The SQL database that was ultimately used for querying historical vulnerability statistics. This is the result of running all scrapers and conducting a manual verification pass.
 - **firefox-bugzilla-scrapers.zip** – An HTML scraper used to extract timeline information from Mozilla Bugzilla entries. The resulting data is used to update the SQL database. The input dataset is included.
 - **firefox-mfsa-scrapers.zip** – An HTML scraper used to extract vulnerability information from Mozilla Foundation Security Advisories and insert it into an SQL database. The input dataset is included.
 - **general-scripts.zip** – “csv_query.rb” was used to generate CSV files for use when creating the graphs included in this paper. “regen.sh” is a wrapper script used to create a clean copy of the SQL database.
 - **ie-scrapers.zip** – An HTML scraper used to extract vulnerability information from Microsoft Security Bulletins and insert it into an SQL database. The input dataset is included.
 - **migrations.zip** – The “schema.rb” and migrations required to generate an empty copy of SQL database.
- **Anti-Exploitation Technologies**
 - **acquire_symbols.py** – A script to acquire the public debugging symbols for each browser and its libraries.

- **AnotherActiveXTry.zip** – A small Active X Visual Studio Project which incorporates the sandbox tests
- **aslr_check.py** – A script to iterate through a list of modules, examining the *OPTIONAL_HEADER.DllCharacteristics* header (see pefile) to ascertain support for ASLR.
- **detect_gs.py** – A script to look for cross-references in IDA Pro for the *security_cookie*. A reference to this piece of data will imply that the /GS compiler option was used during compilation.
Note: We chose to look for the *security_cookie* variable because we had the ability to get debugging symbols for most of the images used by each browser. This gave us the ability to skip code heuristic checks for stack-cookie functionality.
- **file_grabber_<browser>.py** – A script was generated for each browser to parse the modules from the *!dlls -v* output along with any other executables placed on the system upon installation (acquired from Spyme Tools logs).
- **gs_check.py** – A script to iterate through a list of files and run “detect_gs.py” on each file after it is loaded into IDA Pro.
- **Ida-fy.py** – A script to iterate through a list of Windows executables, loading each into IDA Pro then subsequently running “load_symbols.py” on the IDA Pro database.
- **load_symbols.py** – A very small script that uses IDA Pro’s built-in functionality to load the pre-acquired debugging symbols from the current working directory.
- **LowIntegrityShim.bat** – A very small batch script that sets an executable’s priority to *Low Integrity*.
- **npapi.zip** – A Visual Studio project, based on the winless GeckoPluginSDK sample, which is a NPAPI based plug-in which incorporates the sandbox test suites.
- **Third Party**
 - **Windbg** – Window Debugger (Windbg) is a user-land and kernel debugger for the Windows Operating System. Specifically the “!dlls -v” command was used to get a list of the loaded executables for each browser in this study.
<http://msdn.microsoft.com/en-us/windows/hardware/gg463009>
 - **Symchk.exe** – A utility to obtain debugging symbols from a symbol server. Symchk was used to acquire symbols from Microsoft, Google and Mozilla symbol servers for their respective browsers.
<http://support.microsoft.com/kb/311503>
 - **IDA Pro** -- IDA Pro is a Windows or Linux or Mac OS X hosted multi-processor disassembler and debugger that offers so many features it is hard to describe them all. It was used for various tasks throughout this project.
<http://www.hex-rays.com/idapro/>
 - **Pefile** -- pefile is a multi-platform Python module to read and work with Portable Executable (aka PE) files.
<http://code.google.com/p/pefile/>
 - **Spyme Tools** -- SpyMe Tools is very useful in detecting Registry and Disk changes.
http://www.lcibrossolutions.com/spyme_tools.htm

- **Sysinternals Suite** – A suite of tools to help with a variety of tasks on the Windows Operating system.
<http://technet.microsoft.com/en-us/sysinternals/bb842062>